Guida alle migliori pratiche

Amazon EKS

Copyright © 2026 2024. Review license at https://github.com/aws/aws-eks-best-practices/blob/master/LICENSE

Amazon EKS: Guida alle migliori pratiche

Copyright © 2026 2024. Review license at https://github.com/aws/aws-eks-best-practices/blob/master/LICENSE

I marchi e l'immagine commerciale di Amazon non possono essere utilizzati in relazione a prodotti o servizi che non siano di Amazon, in una qualsiasi modalità che possa causare confusione tra i clienti o in una qualsiasi modalità che denigri o discrediti Amazon. Tutti gli altri marchi non di proprietà di Amazon sono di proprietà dei rispettivi proprietari, che possono o meno essere affiliati, collegati o sponsorizzati da Amazon

Table of Contents

Introduzione 1
Guide correlate 2
Contributo 2
Sicurezza 3
Come utilizzare questa guida 3
Comprensione del modello di responsabilità condivisa 3
Introduzione 1
Feedback 6
Approfondimenti 6
Strumenti e risorse 6
Modalità automatica EKS - Sicurezza 6
Architettura di sicurezza 7
Domande frequenti (FAQ) 17
Identity and Access Management 20
Controllo dell'accesso ai cluster EKS 20
Consigli per l'accesso al cluster 27
Identità e credenziali per i pod EKS 34
Identità e credenziali per i pod EKS: raccomandazioni 42
Strumenti e risorse 51
Sicurezza Pod 52
Funzionalità Linux 52
Soluzioni di sicurezza Pod 53
Raccomandazioni 62
Strumenti e risorse 51
Multilocazione 69
Soft multi-tenancy 70
Kubernetes Constructs 72
Controlli attenuanti 74
Multi-tenancy rigida 82
Direzioni future 82
Strumenti e risorse per la gestione di più cluster 83
Controlli Detective 83
Raccomandazioni 62
Strumenti e risorse 51

Sicurezza di rete 91
Controllo del traffico 92
Crittografia di rete 92
Politica di rete 92
Raccomandazioni 62
Gruppi di sicurezza 97
Quando utilizzare Network Policy vs Security Group for Pods? 99
Service Mesh Policy Enforcement o policy di rete Kubernetes 101
ThirdParty Motori di policy di rete 102
Crittografia dei dati in transito 104
Strumenti e risorse 51
Crittografia dei dati e gestione dei segreti 114
Crittografia dei dati a riposo 114
Gestione dei segreti 116
Strumenti e risorse 51
Sicurezza del runtime 119
Contesti di sicurezza e controlli Kubernetes integrati 119
Raccomandazioni 62
Strumenti e risorse 123
Sicurezza dell'infrastruttura 123
Raccomandazioni 62
Alternative 128
Strumenti e risorse 51
Conformità alle normative 131
Spostamento a sinistra 133
Strumenti e risorse 51
Risposta agli incidenti e analisi forense 135
Esempio di piano di risposta agli incidenti 135
Raccomandazioni 62
Strumenti e risorse 51
Sicurezza delle immagini 140
Raccomandazioni 62
Strumenti e risorse 51
Strategia multi-account 151
Pianificazione di una strategia di account con più carichi di lavoro per cluster multi-tenant ... 151
Cluster EKS centralizzato 152

Cluster EKS decentralizzati 158
Cluster EKS centralizzati e non centralizzati 160
Gestione degli accessi ai cluster 163
Opzioni di gestione degli accessi EKS 163
Opzione 1: AWS IAM Identity Center con API Cluster Access Management (CAM) 163
Opzione 2: AWS IAM Users/Roles mappato su gruppi Kubernetes 166
Opzione 3: fornitori OIDC 166
AWS EKS Pod Identity e IRSA per carichi di lavoro 167
Raccomandazione 168
Scalabilità automatica dei cluster 169
Modalità automatica di EKS 169
Motivi per utilizzare la modalità automatica 170
Domande frequenti 170
Karpenter 176
Raccomandazioni 62
Le migliori pratiche di Karpenter 177
Creando NodePools 181
Scheduling Pods 185
Raccomandazioni CoredNS 189
Progetti Karpenter 189
Risorse aggiuntive 90
Cluster Autoscaler 190
Panoramica di 190
Ottimizzazione per prestazioni e scalabilità 194
Ottimizzazione dei costi e della disponibilità 197
Casi d'uso avanzati 200
Parametri aggiuntivi 202
Risorse aggiuntive 90
Riferimenti 82
Affidabilità 207
Come utilizzare questa guida 3
Introduzione 1
Feedback 6
Applicazioni 210
Raccomandazioni 62
Horizontal Pod Autoscaler (HPA) 215

Vertical Pod Autoscaler (VPA) 216
Aggiornamento delle applicazioni 217
Controlli sanitari e autoguarigione 219
Raccomandazioni 219
Gestione delle interruzioni 221
Raccomandazioni 222
Osservabilità 224
Raccomandazioni 225
Piano di controllo 227
Architettura EKS 228
Raccomandazioni 229
Monitora le metriche del piano di controllo 229
Autenticazione del cluster 232
Webhook di ammissione 234
Gestione degli aggiornamenti del cluster 235
Connettività degli endpoint del cluster 235
Esecuzione di cluster di grandi dimensioni 236
Risorse aggiuntive: 237
Piano dei dati 237
Raccomandazioni 62
Rete 244
Modello di rete Kubernetes 244
Container Networking Interface (CNI) 245
Amazon Virtual Private Cloud (VPC) CNI 246
Calcolatore di sottorete 247
Considerazioni su VPC e sottoreti 248
Panoramica di 190
Raccomandazioni 62
CNI di Amazon VPC 259
Panoramica di 190
Raccomandazioni 62
Ottimizzazione dell'utilizzo degli indirizzi IP 272
Ottimizza il consumo IP a livello di nodo 273
Riduci l'esaurimento dell'IP 273
Cluster in esecuzione IPv6 279
Panoramica di 190

Raccomandazioni 62
Rete personalizzata 289
Configurazione di esempio 290
Raccomandazioni 62
Modalità prefisso per Linux 297
Raccomandazioni 62
Modalità prefisso per Windows 303
Raccomandazioni 62
Gruppi di sicurezza per pod 308
Raccomandazioni 62
Bilanciamento del carico 318
Scelta del tipo di Load Balancer 318
Fornitura di sistemi di bilanciamento del carico 319
Scelta del tipo di destinazione Load Balancer 320
Configurazione dei controlli dello stato di Load Balancer 322
Disponibilità e ciclo di vita dei Pod 323
Riferimenti 82
Appendice 328
Monitoraggio dei problemi di prestazioni della rete 329
Monitoraggio del traffico CoredNS per problemi di throttling DNS 329
Monitoraggio dei ritardi delle query DNS utilizzando le metriche di Conntrack 330
Altre importanti metriche relative alle prestazioni di rete 331
Acquisizione delle metriche per monitorare i carichi di lavoro relativi ai problemi di prestazioni di rete 331
Esecuzione di kube-proxy in modalità IPVS 339
Panoramica di 190
Scalabilità 344
Come utilizzare questa guida 344
Comprendere le dimensioni di ridimensionamento 344
Scalabilità estremamente ampia 345
Piano di controllo 345
Limita il carico di lavoro e l'interruzione dei nodi 346
Ridimensiona nodi e pod in modo sicuro 346
Usa Client-Side Cache quando esegui Kubectl 347
Disabilita la compressione kubectl 347
Shard Cluster Autoscaler 348

Priorità ed equità delle API 350
Recupero di risorse nel server API 357
Piano dei dati 364
Scalabilità automatica dei nodi 364
Utilizza molti tipi di istanze EC2 diversi 365
Preferisci nodi più grandi per ridurre il carico del server API 365
Utilizza nodi di dimensioni simili per prestazioni costanti del carico di lavoro 367
Usa le risorse di elaborazione in modo efficiente 367
Automatizza gli aggiornamenti di Amazon Machine Image (AMI) 368
Usa più volumi EBS per i contenitori 368
Evita le istanze con limiti di collegamento EBS bassi se i carichi di lavoro utilizzano volumi EBS 370
Disabilita la registrazione non necessaria su disco 370
Istanze di patch disponibili quando la velocità di aggiornamento del sistema operativo è una necessità 371
Servizi cluster 372
Scalare CoredNS 372
Scala verticalmente il server Kubernetes Metrics 374
Durata CoredNS lameduck 374
Sonda di prontezza CoredNS 374
Agenti di registrazione e monitoraggio 375
Carichi di lavoro 376
Utilizzare IPv6 per il collegamento in rete tramite pod 376
Limita il numero di servizi per namespace 377
Comprendi le quote di Elastic Load Balancer 377
Usa Route 53, Global Accelerator o CloudFront 378
Utilizza EndpointSlices al posto degli endpoint 378
Se possibile, usa segreti immutabili ed esterni 379
Limita la cronologia delle distribuzioni 379
Disabilita enableServiceLinks per impostazione predefinita 380
Limita i webhook ad ammissione dinamica per risorsa 380
Confronta i carichi di lavoro tra più cluster 381
La teoria alla base del ridimensionamento 382
Nodes vs. Churn Rate 382
Pensare in termini di domande al secondo 382
Scalabilità dei componenti distribuiti 383

Colli di bottiglia a monte e a valle 383
Scala in base alle metriche 385
Monitoraggio del piano di controllo 388
Server API 388
Dov'è il problema? 388
Pianificatore 394
Kube Controller Manager 396
ECCETERA 398
Efficienza e scalabilità dei nodi 399
Selezione del nodo 399
Node Bin-packing 401
Utilizzo vs. saturazione 408
Impostazione dei limiti della CPU 416
Kubernetes SLOs 420
Kubernetes SLOs 420
Metriche SLI di Kubernetes 423
SLOs sul tuo cluster 427
Limiti noti e Service Quotas 428
Altri AWS Service Quotas 428
Limitazione delle richieste AWS 434
Altri limiti noti 435
Aggiornamenti del cluster 436
Panoramica di 436
Prima dell'aggiornamento 436
Mantieni il tuo cluster up-to-date 437
Consulta il calendario delle release di EKS 438
Scopri come si applica il modello di responsabilità condivisa agli aggiornamenti del cluster 438
Aggiorna i cluster sul posto 439
Aggiorna il piano di controllo e il piano dati in sequenza 439
Utilizza la documentazione EKS per creare una lista di controllo per gli aggiornamenti 440
Aggiorna componenti aggiuntivi e componenti utilizzando l'API Kubernetes 440
Verifica i requisiti EKS di base prima dell'aggiornamento 442
Verifica gli indirizzi IP disponibili 442
Verifica il ruolo EKS IAM 444
Esegui la migrazione ai componenti aggiuntivi EKS 444
Identifica e correggi l'utilizzo delle API rimosse prima di aggiornare il piano di controllo 445

Cluster Insights ... 445
K ube-no-trouble ... 447
Plutone ... 448
Resources ... 448
Aggiorna i carichi di lavoro Kubernetes. Usa kubectl-convert per aggiornare i manifesti ... 450
Configura PodDisruptionBudgets e garantisci topologySpreadConstraints la disponibilità dei tuoi carichi di lavoro durante l'aggiornamento del piano dati ... 450
Utilizza Managed Node Groups o Karpenter per semplificare gli aggiornamenti del piano dati .. 452
Conferma la compatibilità della versione con i nodi esistenti e il piano di controllo ... 452
Abilita la scadenza dei nodi per i nodi gestiti da Karpenter ... 453
Usa la funzione Drift per i nodi gestiti da Karpenter ... 453
Usa eksctl per automatizzare gli aggiornamenti per i gruppi di nodi autogestiti ... 454
Backup del cluster prima dell'aggiornamento ... 454
Riavvia gli schieramenti di Fargate dopo aver aggiornato il piano di controllo ... 454
Valuta Blue/Green i cluster come alternativa agli aggiornamenti dei cluster in loco ... 455
Tieni traccia delle principali modifiche pianificate nel progetto Kubernetes: pensa al futuro ... 456
Linee guida specifiche sulla rimozione delle funzionalità ... 456
Rimozione di Dockershim nella versione 1.25 - Usa Detector for Docker Socket (DDS) ... 456
Rimozione della PodSecurityPolicy versione 1.25 - Migrazione agli standard di sicurezza Pod o a una soluzione policy-as-code ... 456
Deprecazione del driver di archiviazione In-Tree nella versione 1.23 - Migrazione ai driver CSI (Container Storage Interface) ... 457
Risorse aggiuntive ... 457
ClowdHaus Guida all'aggiornamento di EKS ... 457
GoNoGo ... 457
Ottimizzazione dei costi ... 458
Linee guida generali ... 458
Best practice EKS per l'ottimizzazione dei costi ... 458
Come utilizzare questa guida ... 3
Caratteristiche principali dei servizi AWS e Kubernetes ... 459
Feedback ... 6
Framework ... 460
Il pilastro See: misurazione e responsabilità ... 460
Il pilastro Save: ottimizzazione dei costi ... 461
Il pilastro del piano: pianificazione e previsione ... 462
Il pilastro Run ... 462

Riferimenti ... 82
Consapevolezza ... 463
Raccomandazioni ... 62
Altri strumenti ... 470
Calcolo ... 471
Dimensiona correttamente i tuoi carichi di lavoro ... 472
Ridurre i consumi ... 473
Riduci la capacità inutilizzata ... 474
Ottimizza i tipi di capacità di elaborazione ... 480
Ottimizza l'utilizzo dell'elaborazione ... 484
Rete ... 485
Comunicazione da Pod a Pod ... 485
Comunicazione da Load Balancer a Pod ... 498
Trasferimento dati dal registro dei contenitori ... 500
Trasferimento di dati verso Internet e servizi AWS ... 501
Trasferimento di dati tra VPCs ... 504
Utilizzo di una Service Mesh ... 507
Risorse aggiuntive ... 90
Storage ... 516
Panoramica ... 190
Volumi effimeri ... 516
Volumi persistenti ... 517
Altre considerazioni ... 524
Osservabilità ... 525
Introduzione ... 525
Registrazione ... 525
Piano di controllo EKS ... 525
Piano dati EKS ... 526
Metriche ... 531
Tracciamento ... 534
Risorse aggiuntive: ... 90
Windows ... 538
Gestione AMI ... 538
Gestione della propria AMI Windows ottimizzata per Amazon EKS ... 539
Configurazione di un avvio più rapido per EKS personalizzato ottimizzato AMIs ... 540
Memorizzazione nella cache dei livelli base di Windows su livelli personalizzati AMIs ... 541

Post del blog 542
gMSA per contenitori Windows 542
Cos'è un account gMSA 542
Caso d'uso di Windows container e gMSA 543
Rafforzamento di Windows Server 545
Riduzione della superficie di attacco con Windows Server Core 545
Evitare le connessioni RDP 546
Amazon Inspector 547
Amazon GuardDuty 548
Sicurezza in Amazon EC2 per Windows 548
Scansione di immagini Windows 549
Versioni e licenze di Windows 549
Versione di Windows Server 549
Licenze 550
Registrazione dei log 550
Raccomandazioni per la registrazione 550
Monitoraggio dei contenitori Windows 551
Rete Windows 556
Panoramica di Windows Container Networking 556
Gestione degli indirizzi IP 557
Opzioni CNI (Container Network Interface) 560
Politiche di rete 560
Gestione della memoria e dei sistemi 561
Sistema di prenotazione e memoria Kubelet 561
Requisiti di memoria per contenitori Windows 562
Conclusioni 562
Gestione dell'infrastruttura 562
Spingere e tirare immagini Windows 563
Documentazione di riferimento 565
Pianificazione 565
Assegnazione PODs ai nodi: best practice 566
Garantire che i carichi di lavoro specifici del sistema operativo arrivino sull'host container appropriato 566
Gestione di più build di Windows nello stesso cluster 567
Semplificazione e tolleranza nei manifesti Pod utilizzando NodeSelector RuntimeClass 568
Managed Node Group Support 569

Documentazione aggiuntiva 570
Pod Security per contenitori Windows 570
Opzioni di archiviazione 573
Che cos'è un plug-in in-tree e un plug-in di out-of-tree volume? 573
Plugin In-tree Volume per Windows 573
Out-of-tree per Windows 577
File server Amazon FSx per Windows 578
Rafforzamento delle immagini dei contenitori Windows 579
1. Configura le politiche dell'account (password o blocco) utilizzando le politiche di sicurezza locali e il registro 580
2. Policy di audit 582
3. Procedure consigliate di sicurezza IIS per contenitori Windows 585
4. Principio del privilegio minimo 590
Considerazioni finali: perché proteggere i contenitori Windows è fondamentale nell'odierno panorama delle minacce 593
Ibrido 595
Disconnessione dalla rete 595
Best practice 596
Failover del pod Kubernetes 601
Traffico di rete dell'applicazione 607
Credenziali dell'host 612
AI/ML 615
Feedback 615
Calcolo 615
Ottimizzazione delle risorse GPU e gestione dei costi 615
Resilienza dei nodi e gestione dei lavori di formazione 630
Scalabilità e prestazioni delle applicazioni 633
Allocazione dinamica delle risorse per la gestione avanzata delle GPU 635
Rete 681
Prendi in considerazione una maggiore larghezza di banda di rete o un adattatore Elastic Fabric per applicazioni con un'elevata comunicazione tra i nodi 682
Aumenta il numero di indirizzi IP disponibili per consentire tempi di avvio dei pod più rapidi . 682
Sicurezza 683
Sicurezza e conformità 684
Archiviazione 685
Gestione e archiviazione dei dati 685

Osservabilità 698
Monitoraggio e osservabilità 698
Osservabilità e metriche 699
Performance 715
Gestione degli artefatti ML, framework di servizio e ottimizzazione delle startup 716
Ottimizzazione dei tempi di recupero delle immagini dei contenitori 718
Prendi in considerazione NVMe lo storage in kubelet e containerd 721
Contributi 723
Riepilogo per i contributori esistenti 723
Configura un ambiente di modifica locale 723
Fork e clona il repository 723
Apri l'area di lavoro VS Code 724
Modificare un file 724
Invia una Pull Request 724
Usa l'editor basato sul web github.dev 725
Modifica una singola pagina 725
Visualizza e imposta l'ID per una pagina 725
Imposta l'ID della pagina 726
Creazione di una nuova pagina 726
Crea metadati di pagina 726
Aggiungi al sommario 727
Inserisci un'immagine 727
Controlla lo stile con Vale 728
Crea un'anteprima locale 728
AsciiDoc Cheat Sheet 729
Formattazione di base 729
Headers 729
Elenchi 729
Link 729
Immagini 730
Blocchi di codice 730
Tabelle 730
Ammonizioni 730
Include 731
..... dccxxxii

Guida alle best practice di Amazon EKS

> **Tip**
>
> Esplora le best practice tramite i workshop Amazon EKS.

Benvenuto nelle guide alle best practice di EKS. L'obiettivo principale di questo progetto è offrire una serie di best practice per le operazioni del secondo giorno per Amazon EKS. Abbiamo deciso di pubblicare queste linee guida GitHub in modo da poter iterare rapidamente, fornire consigli tempestivi ed efficaci per una varietà di problemi e incorporare facilmente i suggerimenti della community più ampia.

Al momento abbiamo pubblicato guide per i seguenti argomenti:

- Le migliori pratiche per la sicurezza
- Le migliori pratiche per l'affidabilità
- Migliori pratiche per la scalabilità automatica dei cluster: Karpenter
- Migliori pratiche per l'autoscaling dei cluster: cluster-autoscaler
- Migliori pratiche per la scalabilità automatica dei cluster: EKS Auto Mode
- Migliori pratiche per il networking
- Le migliori pratiche per la scalabilità
- Le migliori pratiche per gli aggiornamenti dei cluster
- Le migliori pratiche per l'ottimizzazione dei costi
- Procedure consigliate per l'esecuzione di contenitori Windows
- Procedure consigliate per le implementazioni ibride
- Le migliori pratiche per l'esecuzione dei carichi di lavoro AI/ML

Abbiamo anche reso open source una CLI (Command Line Interface) basata su Python chiamata hardeneks per verificare alcuni dei consigli di questa guida.

In futuro pubblicheremo linee guida sulle migliori pratiche per prestazioni, ottimizzazione dei costi ed eccellenza operativa.

Guide correlate

Oltre alla EKS User Guide, AWS ha pubblicato diverse altre guide che possono aiutarti nell'implementazione di EKS.

- Guide alle best practice per i contenitori EMR
- Dati su EKS
- Best practice di osservabilità di AWS
- Progetti Amazon EKS per Terraform
- Avvio rapido di Amazon EKS Blueprints

Contributo

Ti invitiamo a contribuire a queste guide. Se hai implementato una pratica che si è dimostrata efficace, condividila con noi aprendo un numero o una pull request. Allo stesso modo, se scopri un errore o un difetto nella guida che abbiamo già pubblicato, invia un PR per correggerlo. Le linee guida per l'invio PRs sono disponibili nelle nostre Linee guida per i contributi.

Le migliori pratiche per la sicurezza

> **Tip**
>
> Esplora le best practice tramite i workshop Amazon EKS.

Questa guida fornisce consigli sulla protezione delle informazioni, dei sistemi e degli asset che dipendono da EKS, offrendo al contempo valore aziendale attraverso valutazioni del rischio e strategie di mitigazione. Le linee guida qui riportate fanno parte di una serie di guide sulle best practice pubblicate da AWS per aiutare i clienti a implementare EKS in conformità con le migliori pratiche. Le guide per le prestazioni, l'eccellenza operativa, l'ottimizzazione dei costi e l'affidabilità saranno disponibili nei prossimi mesi.

Come utilizzare questa guida

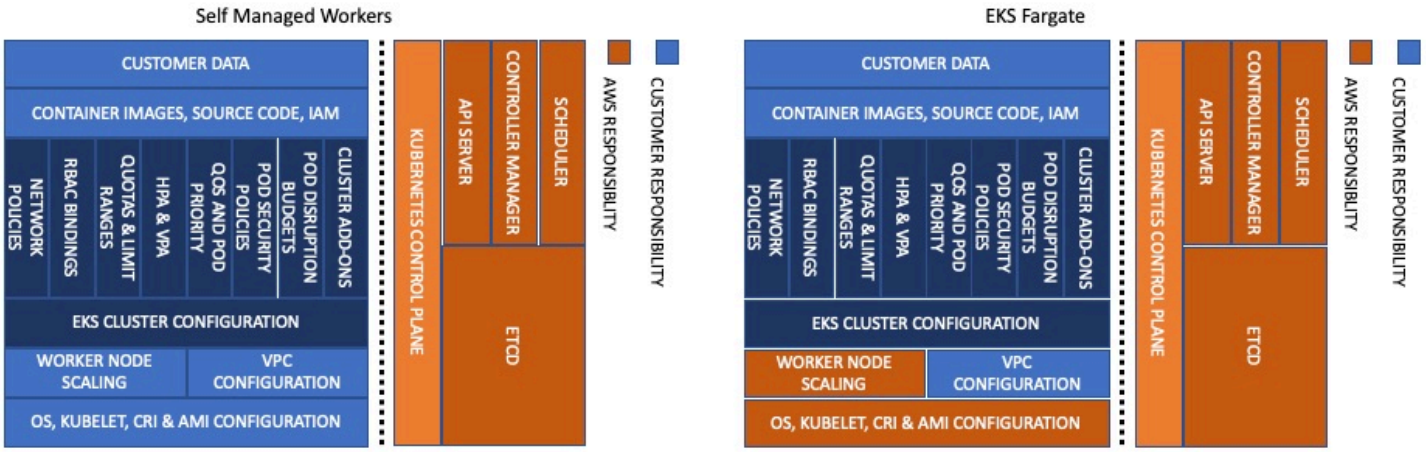
Questa guida è destinata ai professionisti della sicurezza responsabili dell'implementazione e del monitoraggio dell'efficacia dei controlli di sicurezza per i cluster EKS e i carichi di lavoro che supportano. La guida è organizzata in diverse aree tematiche per facilitarne la fruizione. Ogni argomento inizia con una breve panoramica, seguita da un elenco di consigli e best practice per proteggere i cluster EKS. Non è necessario leggere gli argomenti in un ordine particolare.

Comprensione del modello di responsabilità condivisa

La sicurezza e la conformità sono considerate responsabilità condivise quando si utilizza un servizio gestito come EKS. In generale, AWS è responsabile della sicurezza «del» cloud mentre tu, il cliente, sei responsabile della sicurezza «nel» cloud. Con EKS, AWS è responsabile della gestione del piano di controllo Kubernetes gestito da EKS. Ciò include i nodi del piano di controllo Kubernetes, il database ETCD e altre infrastrutture necessarie ad AWS per fornire un servizio sicuro e affidabile. In qualità di utente di EKS, sei in gran parte responsabile degli argomenti di questa guida, ad esempio IAM, sicurezza dei pod, sicurezza di runtime, sicurezza di rete e così via.

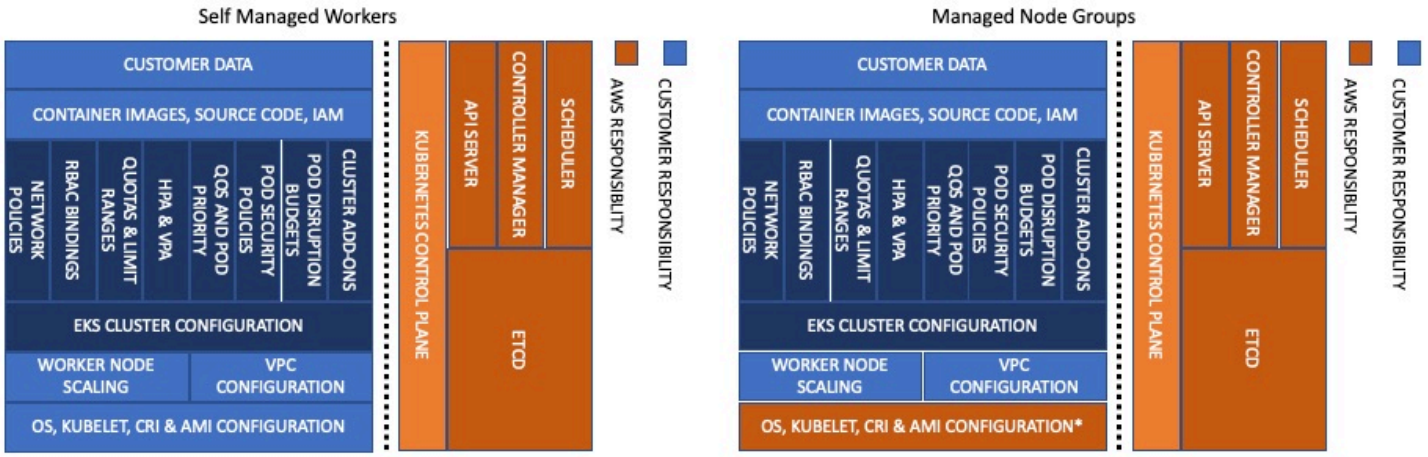
Per quanto riguarda la sicurezza dell'infrastruttura, AWS si assumerà ulteriori responsabilità man mano che passerai dai lavoratori autogestiti ai gruppi di nodi gestiti, a Fargate. Ad esempio, con Fargate, AWS diventa responsabile della protezione del sottostante instance/runtime utilizzato per eseguire i tuoi Pod.

Modello di responsabilità condivisa - Fargate



AWS si assumerà anche la responsabilità di mantenere aggiornata l'AMI ottimizzata per EKS con le versioni delle patch e le patch di sicurezza di Kubernetes. I clienti che utilizzano Managed Node Groups (MNG) sono responsabili dell'aggiornamento dei propri Nodegroup all'AMI più recente tramite API EKS, CLI, Cloudformation o AWS Console. Inoltre, a differenza di Fargate, non MNGs scalerà automaticamente l'infrastruttura/il cluster. Ciò può essere gestito dal cluster-autoscaler o da altre tecnologie come Karpenter, l'autoscalingnativo di AWS, Ocean o Escalator di Atlassian. SpotInst

Modello di responsabilità condivisa - MNG



Prima di progettare il sistema, è importante sapere qual è la linea di demarcazione tra le proprie responsabilità e il fornitore del servizio (AWS).

Per ulteriori informazioni sul modello di responsabilità condivisa, consulta/https://aws.amazon.com/compliance/shared-responsibility-model

Introduzione

Esistono diverse aree di best practice in materia di sicurezza pertinenti quando si utilizza un servizio Kubernetes gestito come EKS:

- Identity and Access Management
- Pod Security
- Sicurezza in fase di esecuzione
- Sicurezza di rete
- Multilocazione
- Account multiplo per multi-tenancy
- Controlli Detective
- Sicurezza dell'infrastruttura
- Crittografia dei dati e gestione dei segreti
- Conformità alle normative
- Risposta agli incidenti e analisi forense
- Sicurezza delle immagini

Come parte della progettazione di qualsiasi sistema, è necessario considerare le sue implicazioni in termini di sicurezza e le pratiche che possono influire sul livello di sicurezza. Ad esempio, è necessario controllare chi può eseguire azioni su un insieme di risorse. È inoltre necessaria la capacità di identificare rapidamente gli incidenti di sicurezza, proteggere i sistemi e i servizi da accessi non autorizzati e mantenere la riservatezza e l'integrità dei dati attraverso la protezione dei dati. Disporre di una serie di processi ben definiti e collaudati per rispondere agli incidenti di sicurezza migliorerà anche il vostro livello di sicurezza. Questi strumenti e tecniche sono importanti perché supportano obiettivi come la prevenzione delle perdite finanziarie o la conformità agli obblighi normativi.

AWS aiuta le organizzazioni a raggiungere i propri obiettivi di sicurezza e conformità offrendo un ricco set di servizi di sicurezza che si sono evoluti sulla base del feedback di un'ampia gamma di clienti attenti alla sicurezza. Offrendo una base altamente sicura, i clienti possono dedicare meno tempo al «lavoro indifferenziato» e più tempo al raggiungimento dei propri obiettivi aziendali.

Feedback

Questa guida viene pubblicata in GitHub modo da raccogliere feedback e suggerimenti diretti dalla comunità più ampia EKS/Kubernetes . Se hai una best practice che ritieni debba essere inclusa nella guida, segnala un problema o invia un PR nell' GitHub archivio. La nostra intenzione è quella di aggiornare periodicamente la guida man mano che vengono aggiunte nuove funzionalità al servizio o quando si evolve una nuova best practice.

Approfondimenti

Questo white paper sulla sicurezza di Kubernetes, sponsorizzato dal Security Audit Working Group, descrive gli aspetti chiave della superficie di attacco e dell'architettura di sicurezza di Kubernetes con l'obiettivo di aiutare i professionisti della sicurezza a prendere solide decisioni di progettazione e implementazione.

Il CNCF ha pubblicato anche un white paper sulla sicurezza nativa del cloud. Il paper esamina l'evoluzione del panorama tecnologico e sostiene l'adozione di pratiche di sicurezza in linea con i DevOps processi e le metodologie agili.

Strumenti e risorse

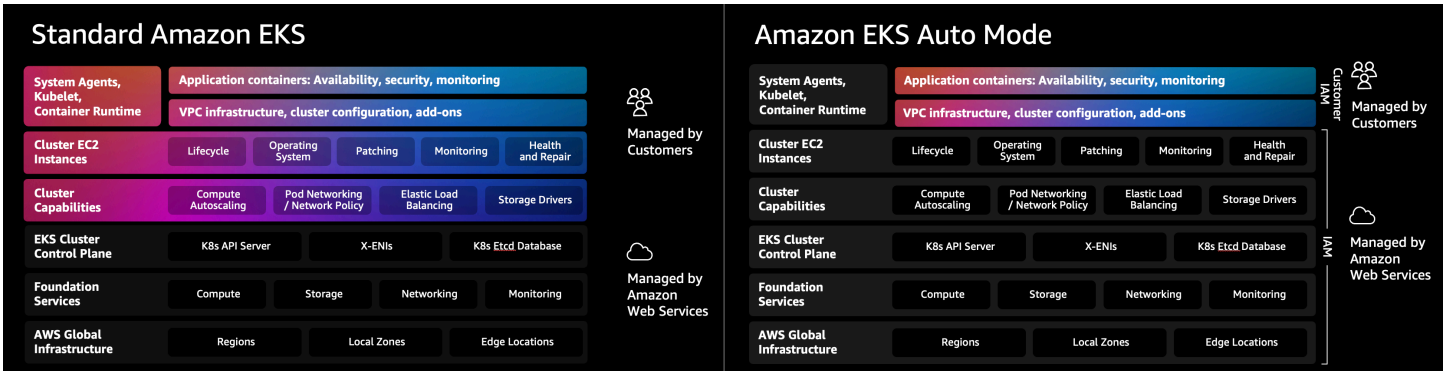
Workshop di immersione nella sicurezza di Amazon EKS

Modalità automatica EKS - Sicurezza

> **Tip**
>
> Esplora le best practice tramite i workshop Amazon EKS.

La modalità automatica di Amazon EKS introduce funzionalità di sicurezza avanzate estendendo la gestione della sicurezza di AWS oltre il piano di controllo per includere nodi di lavoro e componenti principali del cluster. Questo modello di sicurezza completo aiuta le organizzazioni a mantenere una solida posizione di sicurezza riducendo al contempo il sovraccarico operativo.

Modello di responsabilità condivisa - EKS Auto Mode



I principali miglioramenti della sicurezza in modalità automatica EKS includono:

- Sistema operativo minimale ottimizzato per i container con superficie di attacco ridotta
- Best practice di sicurezza applicate tramite EC2 Managed Instances
- Gestione automatizzata delle patch di sicurezza con rotazione obbligatoria dei nodi
- Isolamento dei nodi e limiti di sicurezza integrati
- Integrazione IAM semplificata con autorizzazioni minime richieste

EKS Auto Mode applica questi controlli di sicurezza per impostazione predefinita, aiutando le organizzazioni a soddisfare i propri requisiti di sicurezza e conformità semplificando al contempo le operazioni del cluster. Questo approccio è in linea con defense-in-depth i principi e fornisce più livelli di controlli di sicurezza a livello di infrastruttura, nodo e carico di lavoro.

> **Important**
>
> Sebbene EKS Auto Mode offra funzionalità di sicurezza avanzate, le organizzazioni devono comunque implementare controlli di sicurezza appropriati a livello di applicazione e seguire le migliori pratiche di sicurezza per i carichi di lavoro in esecuzione sul cluster.

Architettura di sicurezza

EKS Auto Mode implementa i controlli di sicurezza su più livelli dell'infrastruttura EKS, dal piano di controllo ai singoli nodi. La comprensione di questa architettura è fondamentale per gestire e proteggere efficacemente i cluster EKS.

Control Plane Security

Il piano di controllo EKS in modalità EKS Auto mantiene gli stessi elevati standard di sicurezza dei cluster EKS tradizionali, aggiungendo al contempo nuove funzionalità di sicurezza:

- Envelope Envelope: tutti i dati dell'API Kubernetes vengono crittografati automaticamente utilizzando la crittografia a busta.
- Integrazione KMS: utilizza AWS KMS con il provider Kubernetes KMS v2, con opzioni per chiavi di proprietà di AWS o chiavi gestite dal cliente (CMK).
- Gestione avanzata dei componenti: i componenti critici come l'auto-scaling, la gestione ENI e i controller EBS vengono spostati all'esterno del cluster e gestiti da AWS.
- Controlli di sicurezza e funzionalità di audit migliorati: le autorizzazioni richieste da EKS Auto Mode, oltre ai cluster EKS standard, sono completamente gestite tramite il ruolo IAM del cluster anziché i ruoli dei singoli nodi.

Integrazione IAM e gestione degli accessi

EKS Auto Mode offre un'integrazione migliorata con AWS Identity and Access Management (IAM) tramite EKS Access Entries e EKS Pod Identity.

Gestione degli accessi ai cluster

EKS Auto Mode introduce miglioramenti alla gestione degli accessi ai cluster tramite l'API Cluster Access Management (CAM):

- Modalità di autenticazione standardizzate tramite `EKS_API`
- Sicurezza migliorata grazie alla gestione degli accessi basata su API
- Controllo semplificato degli accessi tramite Access Entries e Access Policies

È possibile creare voci di accesso per gestire l'accesso al cluster:

```
aws eks create-access-entry \
    --cluster-name ${EKS_CLUSTER_NAME} \
    --principal-arn arn:aws:iam::${ACCOUNT_ID}:role/${IAM_ROLE_NAME} \
    --type STANDARD
```

> **Important**
>
> Sebbene sia ancora possibile creare un cluster EKS Auto Mode con modalità di `CONFIG_MAP_AND_API` autenticazione, questo non è l'approccio standard e si consiglia vivamente di utilizzare la modalità di `API` autenticazione predefinita per i nuovi cluster. `API`l'autenticazione basata offre una maggiore sicurezza e una gestione semplificata degli accessi rispetto all'approccio ConfigMap basato sulle versioni precedenti.

EKS Pod Identity

EKS Auto Mode viene fornito con Pod Identity Agent già distribuito, che consente un modo semplificato per concedere le autorizzazioni AWS IAM ai pod:

- Gestione semplificata delle autorizzazioni IAM senza configurazione del provider OIDC
- Sovraccarico operativo ridotto rispetto a IRSA
- Sicurezza migliorata grazie alla codifica delle sessioni e al supporto ABAC

```
aws eks create-pod-identity-association \
  --cluster-name ${EKS_CLUSTER_NAME} \
  --role-arn arn:aws:iam::${AWS_ACCOUNT_ID}:role/${IAM_ROLE_NAME} \
  --namespace ${NAMESPACE} \
  --service-account ${SERVICE_ACCOUNT_NAME}
```

> **Important**
>
> Pod Identity è l'approccio consigliato per le autorizzazioni IAM in modalità EKS Auto in quanto offre funzionalità di sicurezza avanzate e una gestione semplificata rispetto a IRSA.

Ruolo IAM del nodo

EKS Auto Mode utilizza una nuova modalità `AmazonEKSWorkerNodeMinimalPolicy` che fornisce solo le autorizzazioni necessarie per il funzionamento dei nodi EKS Auto Mode. Queste autorizzazioni:

- Fornisci un set ridotto di autorizzazioni rispetto alle politiche tradizionali dei nodi

- Rispetta il principio del privilegio minimo
- Sono gestiti e aggiornati automaticamente da AWS

Questo approccio politico minimale aiuta a migliorare il livello di sicurezza limitando le autorizzazioni disponibili per il nodo e i relativi carichi di lavoro.

Sicurezza dei nodi

EKS Auto Mode introduce diversi miglioramenti significativi della sicurezza a livello di nodo:

Sicurezza delle istanze gestite EC2

I nodi EKS Auto Mode utilizzano istanze gestite Amazon EC2 con proprietà di sicurezza avanzate:

- Restrizioni imposte da IAM che impediscono operazioni che potrebbero compromettere la capacità di AWS di gestire i nodi
- Modelli di infrastruttura immutabili in cui le modifiche alla configurazione richiedono la sostituzione dei nodi
- Sostituzione obbligatoria dei nodi entro 21 giorni per garantire aggiornamenti di sicurezza regolari
- Accesso limitato ai metadati dell'istanza utilizzando limiti IMDSv2 di hop controllati

Sicurezza del sistema operativo

Il sistema operativo è una variante personalizzata di Bottlerocket, ottimizzata per EKS Auto Mode, che include:

- Filesystem root di sola lettura
- SELinux abilitato di default con controlli di accesso obbligatori
- Isolamento automatico dei Pod mediante etichette SELinux MCS uniche
- Accesso SSH disabilitato e rimozione di servizi non necessari
- Patch di sicurezza automatizzate tramite rotazione dei nodi

Sicurezza dei componenti del nodo

I componenti del nodo sono configurati secondo le migliori pratiche di sicurezza:

- Kubelet è configurato con impostazioni predefinite sicure
- Configurazione rafforzata del runtime del contenitore
- Gestione e rotazione automatizzate dei certificati
- node-to-control-planeComunicazione limitata

Sicurezza di rete

EKS Auto Mode implementa diverse funzionalità di sicurezza di rete per garantire una comunicazione sicura all'interno del cluster e con risorse esterne:

Politica di rete VPC CNI

EKS Auto Mode sfrutta il supporto nativo della politica di rete Kubernetes del plug-in Amazon VPC CNI:

- Si integra con l'API Kubernetes Network Policy upstream
- Consente un controllo granulare sulla comunicazione pod-to-pod
- Supporta sia le regole di ingresso che quelle di uscita

Per abilitare il supporto delle policy di rete in modalità automatica EKS, è necessario configurare il componente aggiuntivo VPC CNI con un manifesto. `configMap` Ecco un esempio:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: amazon-vpc-cni
  namespace: kube-system
data:
  enable-network-policy: "true"
```

È inoltre necessario definire la configurazione del supporto delle politiche di rete nella classe Node, come illustrato di seguito:

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: example-node-class
```

```
spec:
  networkPolicy: DefaultAllow
  networkPolicyEventLogs: Enabled
```

Una volta abilitato, puoi creare politiche di rete per controllare il traffico:

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny
spec:
  podSelector: {}
  policyTypes:
  - Ingress
  - Egress
```

Gestione ENI migliorata

EKS Auto Mode offre una maggiore sicurezza per la gestione dell'interfaccia di rete elastica (ENI):

- Allegato e configurazione ENI gestiti da AWS
- Separazione del traffico di controllo dal traffico di dati
- Gestione automatizzata degli indirizzi IP con privilegi ridotti richiesti sui nodi

Sicurezza dello storage

EKS Auto Mode offre funzionalità di sicurezza avanzate per lo storage temporaneo e persistente:

Archiviazione effimera

- Tutti i dati scritti su volumi temporanei vengono crittografati automaticamente
- Utilizza l'algoritmo crittografico AES-256 standard del settore
- Crittografia e decrittografia gestite senza problemi dal servizio

Volumi EBS

- I volumi EBS root e dati sono sempre crittografati
- I volumi sono configurati per essere eliminati al termine dell'istanza

- È disponibile un'opzione per specificare chiavi KMS personalizzate per la crittografia

Integrazione EFS

- Support per la crittografia in transito con EFS
- Crittografia automatica a riposo per i file system EFS
- Integrazione con punti di accesso EFS per un migliore controllo degli accessi

> **Important**
>
> Quando utilizzate EFS con EKS Auto Mode, assicuratevi che le impostazioni di crittografia appropriate siano configurate a livello di file system EFS, poiché EKS Auto Mode non gestisce direttamente la crittografia EFS.

Monitoraggio e registrazione

EKS Auto Mode offre funzionalità avanzate di monitoraggio e registrazione per aiutarvi a mantenere la visibilità sul livello di sicurezza e sullo stato operativo del cluster.

Registrazione del piano di controllo

EKS Auto Mode mantiene le stesse funzionalità di registrazione del piano di controllo di EKS standard, tuttavia abilita tutti i log per impostazione predefinita per un monitoraggio avanzato.

- I log vengono inviati ad Amazon CloudWatch Logs
- Per impostazione predefinita, EKS Auto Mode abilita tutti i log del piano di controllo: server API, audit, authenticator, controller manager e scheduler
- EKS Auto Mode consente una visibilità dettagliata delle operazioni del cluster e degli eventi di sicurezza

> **Important**
>
> La registrazione del Control Plane comporta costi aggiuntivi per l'archiviazione dei log. CloudWatch Valuta attentamente la tua strategia di registrazione per bilanciare le esigenze di sicurezza con la gestione dei costi.

Registrazione a livello di nodo

La modalità automatica EKS migliora la registrazione a livello di nodo:

- I registri di sistema vengono raccolti automaticamente e sono accessibili tramite Logs CloudWatch
- I log dei nodi vengono conservati anche dopo la chiusura del nodo, facilitando l'analisi post-incidente
- Visibilità migliorata sugli eventi di sicurezza a livello di nodo e sui problemi operativi

GuardDuty Integrazione con Amazon

I cluster EKS Auto Mode si integrano perfettamente con Amazon GuardDuty per un rilevamento avanzato delle minacce. Le caratteristiche includono:

- Scansione automatizzata dei registri di audit del piano di controllo
- Monitoraggio del runtime che può essere abilitato per il monitoraggio dei carichi di lavoro
- Integrazione con i GuardDuty risultati esistenti e i meccanismi di allarme

Per abilitare la protezione EKS Auto Mode su Amazon GuardDuty for Kubernetes Audit Logs, puoi eseguire il seguente comando:

```
aws guardduty update-detector \
    --detector-id 12abc34d567e8fa901bc2d34e56789f0 \
    --data-sources '{"Kubernetes":{"AuditLogs":{"Enable":true}}}'
```

GuardDuty Integrazione Amazon per la sicurezza in fase di esecuzione

Amazon GuardDuty fornisce il monitoraggio essenziale della sicurezza in fase di esecuzione per i cluster EKS Auto Mode, offrendo funzionalità complete di rilevamento delle minacce e monitoraggio della sicurezza. Questa integrazione è particolarmente importante in quanto aiuta a identificare potenziali minacce alla sicurezza e attività dannose in tempo reale.

GuardDuty Caratteristiche principali di EKS Auto Mode

- Monitoraggio del runtime:
  - Monitoraggio continuo del comportamento in fase di esecuzione
  - Rilevamento di attività dannose o sospette

  - Identificazione di potenziali tentativi di fuga dal container
  - Monitoraggio dell'esecuzione di processi o delle connessioni di rete insolite
- Rilevamento di minacce specifico per Kubernetes:
  - Identificazione di tentativi di implementazione di pod sospetti
  - Rilevamento di contenitori compromessi
  - Monitoraggio del lancio di container privilegiati
  - Identificazione dell'utilizzo sospetto dell'account di servizio
- Tipi di ricerca completi:
  - Policy:Kubernetes/* - Rileva le violazioni delle migliori pratiche di sicurezza
  - Impact:Kubernetes/* - Identifica le risorse potenzialmente interessate
  - Discovery:Kubernetes/* - Rileva le attività di ricognizione
  - Execution:Kubernetes/* - Identifica modelli di esecuzione sospetti
  - Persistence:Kubernetes/* - Rileva potenziali minacce persistenti

Per abilitare la protezione EKS Auto Mode su Amazon GuardDuty for Kubernetes Audit Logs and Runtime Monitoring, puoi eseguire il seguente comando:

```
aws guardduty update-detector \
    --detector-id 12abc34d567e8fa901bc2d34e56789f0 \
    --data-sources '{
        "Kubernetes": {
            "AuditLogs": {"Enable": true},
            "RuntimeMonitoring": {"Enable": true}
        }
    }'
```

> **Important**
>
> GuardDuty Il monitoraggio del runtime è supportato automaticamente nei cluster EKS Auto Mode, offrendo una maggiore visibilità della sicurezza senza configurazioni aggiuntive a livello di nodo.

GuardDuty Integrazione dei risultati

GuardDuty i risultati possono essere integrati con altri servizi AWS per una risposta automatica:

- EventBridge Regole:

```
{
  "source": ["aws.guardduty"],
  "detail-type": ["GuardDuty Finding"],
  "detail": {
    "type": ["Runtime:Container/*", "Runtime:Kubernetes/*"],
    "severity": [4, 5, 6, 7, 8]
  }
}
```

- Integrazione con Security Hub:

```
# Enable Security Hub integration
aws securityhub enable-security-hub \
    --enable-default-standards \
    --tags '{"Environment":"Production"}' \
    --region us-west-2
```

Migliori pratiche per l' GuardDuty utilizzo della modalità automatica EKS

1. Abilita tutti i tipi di ricerca:
   - Abilita sia il monitoraggio dei log di controllo di Kubernetes che il monitoraggio del runtime
   - Configura i risultati per tutti i livelli di gravità
2. Implementa la risposta automatizzata:
   - Crea EventBridge regole per risultati di elevata gravità
   - Integrazione con AWS Security Hub per la gestione centralizzata della sicurezza
   - Imposta azioni di riparazione automatizzate, se del caso
3. Revisione e ottimizzazione regolari:
   - Esamina regolarmente GuardDuty i risultati
   - Ottimizza le soglie di rilevamento in base al tuo ambiente
   - Aggiorna le procedure di risposta in base a nuovi tipi di risultati
4. Gestione tra account:
   - Prendi in considerazione l'utilizzo di un account GuardDuty amministratore per la gestione centralizzata

- Abilita l'aggregazione dei risultati su più account

> **Warning**
>
> Sebbene GuardDuty fornisca un monitoraggio completo della sicurezza, dovrebbe far parte di una defense-in-depth strategia che includa altri controlli di sicurezza come le politiche di rete, gli standard di sicurezza dei Pod e la corretta configurazione RBAC.

Domande frequenti (FAQ)

D: In che modo EKS Auto Mode differisce dallo standard EKS in termini di sicurezza? R: La modalità automatica EKS offre una maggiore sicurezza tramite istanze gestite EC2, patch automatizzate, rotazione obbligatoria dei nodi e controlli di sicurezza integrati. Riduce il sovraccarico operativo mantenendo al contempo una solida posizione di sicurezza grazie alla gestione da parte di AWS di una maggior parte degli aspetti di sicurezza.

D: Posso ancora utilizzare gli strumenti e le politiche di sicurezza esistenti con EKS Auto Mode? R: Sì, EKS Auto Mode è compatibile con la maggior parte degli strumenti e delle politiche di sicurezza esistenti. Tuttavia, alcuni strumenti di sicurezza a livello di nodo potrebbero richiedere un adattamento a causa della natura gestita dei nodi EKS Auto Mode.

D: Come posso implementare agenti di sicurezza e strumenti di monitoraggio in EKS Auto Mode? R: In modalità automatica EKS, gli agenti di sicurezza e gli strumenti di monitoraggio devono essere distribuiti come carichi di lavoro Kubernetes (in genere DaemonSets, implementa un'istanza del Pod su ogni nodo per impostazione predefinita) anziché installati direttamente sul sistema operativo del nodo. Questo approccio è in linea con il modello di infrastruttura immutabile di EKS Auto Mode. Esempio:

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: security-agent
  namespace: security
spec:
  selector:
    matchLabels:
      app: security-agent
```

```
  template:
    metadata:
      labels:
        app: security-agent
    spec:
      containers:
      - name: security-agent
        image: security-vendor/agent:latest
        securityContext:
          privileged: false
          # Use specific capabilities instead of privileged mode
          capabilities:
            add: ["NET_ADMIN", "SYS_ADMIN"]
```

D: Le soluzioni di sicurezza di terze parti sono compatibili con EKS Auto Mode? R: Molte delle più diffuse soluzioni di sicurezza di terze parti sono state aggiornate per supportare EKS Auto Mode, tuttavia è sempre consigliabile verificare la versione specifica e i requisiti di implementazione con il proprio fornitore di sicurezza, poiché il supporto per EKS Auto Mode potrebbe richiedere versioni aggiornate o configurazioni di implementazione specifiche.

D: Quali sono le limitazioni per gli agenti di sicurezza in EKS Auto Mode? R: Le limitazioni principali includono:

- Nessun accesso diretto per modificare il sistema operativo del nodo
- Nessuna persistenza tra le rotazioni dei nodi
- Deve essere compatibile con la distribuzione basata su container
- È necessario rispettare l'immutabilità dei nodi
- Potrebbe richiedere diverse configurazioni di privilegi
- Qualsiasi modifica persistente ai nodi deve essere eseguita tramite risorse `NodePools` e `NodeClasses` risorse.

> **Note**
>
> Sebbene EKS Auto Mode possa richiedere modifiche alla strategia di implementazione degli strumenti di sicurezza, queste modifiche spesso si traducono in configurazioni più gestibili e sicure, in linea con le migliori pratiche native del cloud. EKS Auto Mode prevede di assumere completamente la maggior parte delle funzionalità che gestisce. Pertanto, qualsiasi modifica

manuale apportata a tali funzionalità, se possibile, potrebbe essere sovrascritta o eliminata da EKS Auto Mode.

D: Posso usare la modalità personalizzata AMIs con EKS Auto Mode? R: Al momento, la modalità automatica EKS non supporta la personalizzazione AMIs. Ciò è dovuto alla progettazione in quanto AWS gestisce la sicurezza, l'applicazione di patch e la manutenzione dei nodi come parte del modello di responsabilità condivisa. I nodi EKS Auto Mode utilizzano una variante specializzata di Bottlerocket ottimizzata e gestita da AWS.

D: Con che frequenza i nodi vengono ruotati automaticamente in EKS Auto Mode? R: I nodi in modalità automatica EKS hanno una durata massima di 21 giorni. Verranno sostituiti automaticamente prima di questo limite, garantendo aggiornamenti di sicurezza regolari e l'applicazione delle patch.

D: Posso accedere tramite SSH ai nodi EKS Auto Mode per la risoluzione dei problemi? R: No, l'accesso SSH diretto non è disponibile in modalità automatica EKS. È invece possibile utilizzare la NodeDiagnostic Custom Resource Definition (CRD) per raccogliere i registri di sistema e le informazioni di debug.

D: Il supporto per le policy di rete è abilitato per impostazione predefinita in EKS Auto Mode? R: Per ora, il supporto delle politiche di rete deve essere abilitato in modo esplicito tramite la configurazione del componente aggiuntivo VPC CNI. Una volta abilitato, puoi utilizzare le politiche di rete Kubernetes standard.

D: Devo usare IRSA o Pod Identity con EKS Auto Mode? R: Sebbene entrambi siano supportati, Pod Identity è l'approccio consigliato in EKS Auto Mode poiché include già il componente aggiuntivo dell'agente Pod Identity Security e offre funzionalità di sicurezza avanzate e una gestione semplificata.

D: Posso ancora usare aws-auth ConfigMap in modalità automatica EKS? R: `aws-auth` ConfigMap È una funzionalità obsoleta. Si consiglia di utilizzare l'approccio predefinito dell'autenticazione basata su API per una maggiore sicurezza e una gestione semplificata degli accessi.

D: Come posso monitorare gli eventi di sicurezza in EKS Auto Mode? R: EKS Auto Mode si integra con diverse soluzioni di monitoraggio GuardDuty, tra cui CloudWatch, e CloudTrail. GuardDuty fornisce un monitoraggio avanzato della sicurezza in fase di esecuzione, specifico per i carichi di lavoro EKS.

D: Come si raccolgono i log dai nodi EKS Auto Mode? R: Usa il NodeDiagnostic CRD, che carica automaticamente i log in un bucket S3. Puoi anche usare CloudWatch Container Insights e AWS Distro per OpenTelemetry.

> **Note**
>
> Questa sezione delle domande frequenti viene aggiornata regolarmente man mano che vengono aggiunte nuove funzionalità a EKS Auto Mode e quando riceviamo domande frequenti dai clienti.

Identity and Access Management

> **Tip**
>
> Esplora le best practice tramite i workshop Amazon EKS.

Identity and Access Management (IAM) è un servizio AWS che svolge due funzioni essenziali: autenticazione e autorizzazione. L'autenticazione implica la verifica di un'identità, mentre l'autorizzazione regola le azioni che possono essere eseguite dalle risorse AWS. All'interno di AWS, una risorsa può essere un altro servizio AWS, ad esempio EC2, o un principale AWS come un utente o un ruolo IAM. Le regole che regolano le azioni che una risorsa è autorizzata a eseguire sono espresse come politiche IAM.

Controllo dell'accesso ai cluster EKS

Il progetto Kubernetes supporta una varietà di strategie diverse per autenticare le richieste al servizio kube-apiserver, ad esempio Bearer Tokens, certificati X.509, OIDC, ecc. Attualmente EKS offre il supporto nativo per l'autenticazione dei token webhook, i token degli account di servizio e, dal 21 febbraio 2021, l'autenticazione OIDC.

La strategia di autenticazione webhook richiede un webhook che verifica i token bearer. Su EKS, questi token bearer vengono generati dalla CLI AWS o dal aws-iam-authenticatorclient quando esegui i comandi. `kubectl` Mentre esegui i comandi, il token viene passato al kube-apiserver che lo inoltra al webhook di autenticazione. Se la richiesta è ben formata, il webhook richiama un URL prefirmato incorporato nel corpo del token. Questo URL convalida la firma della richiesta e restituisce informazioni sull'utente, ad esempio l'account dell'utente, Arn e il kube-apiserver. UserId

Per generare manualmente un token di autenticazione, digitate il seguente comando in una finestra di terminale:

```
aws eks get-token --cluster-name <cluster_name> --region <region>
```

L'output dovrebbe essere simile al seguente:

```
{
    "kind": "ExecCredential",
    "apiVersion": "client.authentication.k8s.io/v1alpha1",
    "spec": {},
    "status": {
        "expirationTimestamp": "2024-12-20T17:38:48Z",
        "token": "k8s-aws-
v1.aHR0cHM6Ly9zdHMudXMtd2VzdC0yLmFtYXpvbmF3cy5jb20vP0FjdGlvbj1HZ...."
    }
}
```

Puoi anche ottenere un token a livello di codice. Di seguito è riportato un esempio scritto in Go:

```
package main

import (
  "fmt"
  "log"
  "sigs.k8s.io/aws-iam-authenticator/pkg/token"
)

func main()  {
  g, _ := token.NewGenerator(false, false)
  tk, err := g.Get("<cluster_name>")
  if err != nil {
    log.Fatal(err)
  }
  fmt.Println(tk)
}
```

L'output dovrebbe essere simile al seguente:

```
{
  "kind": "ExecCredential",
  "apiVersion": "client.authentication.k8s.io/v1alpha1",
```

```
  "spec": {},
  "status": {
    "expirationTimestamp": "2020-02-19T16:08:27Z",
    "token": "k8s-aws-
v1.aHR0cHM6Ly9zdHMuYW1hem9uYXdzLmNvbS8_QWN0aW9uPUdldENhbGxlcklkZW50aXR5JlZlcnNpb249MjAxMS0wNi0x
  }
}
```

Ogni token inizia con `k8s-aws-v1.` seguito da una stringa codificata in base64. La stringa, una volta decodificata, dovrebbe assomigliare a qualcosa di simile a questa:

```
https://sts.amazonaws.com/?Action=GetCallerIdentity&Version=2011-06-15&X-
Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=XXXXJPFRILKNSRC2W5QA
%2F20200219%2Fus-xxxx-1%2Fsts%2Faws4_request&X-Amz-Date=20200219T155427Z&X-
Amz-Expires=60&X-Amz-SignedHeaders=host%3Bx-k8s-aws-id&X-Amz-
Signature=XXXf8f3285e320ddb5e683a5c9a405301ad76546f24f28111fdad09cf648a393
```

Il token è costituito da un URL prefirmato che include una credenziale e una firma Amazon. Per ulteriori dettagli, consulta https://docs.aws.amazon.com/STS/ latest/APIReference/API _ GetCallerIdentity .html.

Il token ha un tempo di vita (TTL) di 15 minuti, dopodiché sarà necessario generare un nuovo token. Questa operazione viene gestita automaticamente quando utilizzi un client, ad esempio`kubectl`, se utilizzi la dashboard di Kubernetes, dovrai generare un nuovo token e riautenticarti ogni volta che il token scade.

Una volta che l'identità dell'utente è stata autenticata dal servizio AWS IAM, il kube-apiserver legge il file `aws-auth` ConfigMap nel `kube-system` Namespace per determinare il gruppo RBAC da associare all'utente. `aws-auth` ConfigMap Viene utilizzato per creare una mappatura statica tra i principali IAM, ovvero IAM Users and Roles e i gruppi Kubernetes RBAC. È possibile fare riferimento ai gruppi RBAC in Kubernetes o. RoleBindings ClusterRoleBindings Sono simili ai ruoli IAM in quanto definiscono un insieme di azioni (verbi) che possono essere eseguite su una raccolta di risorse (oggetti) Kubernetes.

CloudWatch query per aiutare gli utenti a identificare i client che inviano richieste all'endpoint STS globale

Esegui la CloudWatch query seguente per ottenere l'endpoint sts. Se stsendpoint è uguale a «sts.amazonaws.com», allora è un endpoint STS globale. Se stsendpoint è uguale a «sts». <region>.amazonaws.com», allora è un endpoint STS regionale.

```
fields @timestamp, @message, @logStream, @log,stsendpoint
| filter @logStream like /authenticator/
| filter @message like /stsendpoint/
| sort @timestamp desc
| limit 10000
```

Cluster Access Manager

Cluster Access Manager, ora il modo preferito per gestire l'accesso dei principali AWS IAM ai cluster Amazon EKS, è una funzionalità dell'API AWS ed è una funzionalità opzionale per i cluster EKS v1.23 e versioni successive (nuovi o esistenti). Semplifica la mappatura delle identità tra AWS IAM e KubernetesRBACs, eliminando la necessità di passare da AWS a Kubernetes APIs o di modificarla `aws-auth` ConfigMap per la gestione degli accessi, riducendo il sovraccarico operativo e aiutando a risolvere le configurazioni errate. Lo strumento consente inoltre agli amministratori del cluster di revocare o perfezionare `cluster-admin` le autorizzazioni concesse automaticamente al principale AWS IAM utilizzato per creare il cluster.

Questa API si basa su due concetti:

- Voci di accesso: un'identità di cluster direttamente collegata a un principale AWS IAM (utente o ruolo) autorizzata ad autenticarsi su un cluster Amazon EKS.
- Policy di accesso: sono politiche specifiche di Amazon EKS che forniscono l'autorizzazione a un Access Entry per eseguire azioni nel cluster Amazon EKS.

Al momento del lancio Amazon EKS supporta solo policy predefinite e gestite da AWS. Le policy di accesso non sono entità IAM e sono definite e gestite da Amazon EKS.

Cluster Access Manager consente la combinazione di RBAC upstream con policy di accesso che supportano le decisioni di autorizzazione e approvazione (ma non negazione) sulle decisioni di Kubernetes AuthZ relative alle richieste dei server API. Una decisione di rifiuto viene presa quando entrambi gli autorizzatori RBAC a monte e Amazon EKS non sono in grado di determinare l'esito della valutazione di una richiesta.

Con questa funzionalità, Amazon EKS supporta tre modalità di autenticazione:

1. `CONFIG_MAP`per continuare a utilizzare esclusivamente `aws-auth` ConfigMap.

2. `API_AND_CONFIG_MAP`per reperire i principali IAM autenticati sia da EKS Access Entry APIs che da `aws-auth` ConfigMap, dando priorità agli Access Entries. Ideale per migrare le autorizzazioni esistenti su Access Entries. `aws-auth`
3. `API`affidarsi esclusivamente a EKS Access Entry. APIs Questo è il nuovo approccio consigliato.

Per iniziare, gli amministratori dei cluster possono creare o aggiornare i cluster Amazon EKS, impostare l'autenticazione `API_AND_CONFIG_MAP` o il `API` metodo preferito e definire gli Access Entries per concedere l'accesso ai principali AWS IAM desiderati.

```
$ aws eks create-cluster \
    --name <CLUSTER_NAME> \
    --role-arn <CLUSTER_ROLE_ARN> \
    --resources-vpc-config
 subnetIds=<value>,endpointPublicAccess=true,endpointPrivateAccess=true \
    --logging '{"clusterLogging":[{"types":
["api","audit","authenticator","controllerManager","scheduler"],"enabled":true}]}' \
    --access-config
 authenticationMode=API_AND_CONFIG_MAP,bootstrapClusterCreatorAdminPermissions=false
```

Il comando precedente è un esempio di creazione di un cluster Amazon EKS già senza le autorizzazioni di amministratore del creatore del cluster.

È possibile aggiornare la configurazione dei cluster Amazon EKS per abilitare `API` AuthenticationMode utilizzando il `update-cluster-config` comando, per farlo sui cluster esistenti è `CONFIG_MAP` necessario prima eseguire l'aggiornamento a e poi a. `API_AND_CONFIG_MAP` `API` Queste operazioni non possono essere annullate, il che significa che non è possibile passare da `API` `API_AND_CONFIG_MAP` o `CONFIG_MAP` e viceversa. `API_AND_CONFIG_MAP` `CONFIG_MAP`

```
$ aws eks update-cluster-config \
    --name <CLUSTER_NAME> \
    --access-config authenticationMode=API
```

L'API supporta i comandi per aggiungere e revocare l'accesso al cluster, nonché per convalidare le politiche di accesso e le voci di accesso esistenti per il cluster specificato. Le politiche predefinite vengono create per corrispondere a RBACs Kubernetes come segue.

Politica di accesso EKS	Kubernetes RBAC
Amazon EKSCluster AdminPolicy	amministratore del cluster
EKSAdminPolitica di Amazon	admin
EKSEditPolitica di Amazon	modifica
EKSViewPolitica di Amazon	visualizzazione

```
$ aws eks list-access-policies
{
    "accessPolicies": [
        {
            "name": "AmazonEKSAdminPolicy",
            "arn": "arn:aws:eks::aws:cluster-access-policy/AmazonEKSAdminPolicy"
        },
        {
            "name": "AmazonEKSClusterAdminPolicy",
            "arn": "arn:aws:eks::aws:cluster-access-policy/AmazonEKSClusterAdminPolicy"
        },
        {
            "name": "AmazonEKSEditPolicy",
            "arn": "arn:aws:eks::aws:cluster-access-policy/AmazonEKSEditPolicy"
        },
        {
            "name": "AmazonEKSViewPolicy",
            "arn": "arn:aws:eks::aws:cluster-access-policy/AmazonEKSViewPolicy"
        }
    ]
}

$ aws eks list-access-entries --cluster-name <CLUSTER_NAME>

{
    "accessEntries": []
}
```

Non sono disponibili voci di accesso quando il cluster viene creato senza l'autorizzazione di amministratore del creatore del cluster, che è l'unica voce creata per impostazione predefinita.

Il **aws-auth** ConfigMap (obsoleto)

Un modo in cui l'integrazione di Kubernetes con l'autenticazione AWS può essere effettuata tramite `aws-auth` ConfigMap, che risiede nel Namespace. `kube-system` È responsabile della mappatura dell'autenticazione AWS IAM Identities (Users, Groups and Roles) all'autorizzazione di controllo degli accessi basato sui ruoli (RBAC) di Kubernetes. `aws-auth` ConfigMap Viene creato automaticamente nel cluster Amazon EKS durante la fase di provisioning. È stato inizialmente creato per consentire ai nodi di unirsi al cluster, ma come accennato in precedenza, è possibile utilizzarlo anche ConfigMap per aggiungere RBACs l'accesso ai principali IAM.

Per controllare quello del tuo cluster `aws-auth` ConfigMap, puoi usare il seguente comando.

```
kubectl -n kube-system get configmap aws-auth -o yaml
```

Questo è un esempio di configurazione predefinita di `aws-auth`ConfigMap.

```
apiVersion: v1
data:
  mapRoles: |
    - groups:
      - system:bootstrappers
      - system:nodes
      - system:node-proxier
      rolearn: arn:aws:iam::<AWS_ACCOUNT_ID>:role/kube-system-<SELF_GENERATED_UUID>
      username: system:node:{{SessionName}}
kind: ConfigMap
metadata:
  creationTimestamp: "2023-10-22T18:19:30Z"
  name: aws-auth
  namespace: kube-system
```

La sessione principale di questo ConfigMap, si trova `data` sotto il `mapRoles` blocco, che è fondamentalmente composto da 3 parametri.

- gruppi: Kubernetes group/groups a cui mappare il ruolo IAM. Può essere un gruppo predefinito o un gruppo personalizzato specificato in un o. `clusterrolebinding` `rolebinding` Nell'esempio precedente abbiamo dichiarato solo gruppi di sistema.
- rolearn: L'ARN del ruolo AWS IAM deve essere mappato al gruppo/gruppi Kubernetes add, utilizzando il seguente formato. `arn:<PARTITION>:iam::<AWS_ACCOUNT_ID>:role/role-name`

- username: il nome utente all'interno di Kubernetes da mappare al ruolo AWS IAM. Può essere qualsiasi nome personalizzato.

È anche possibile mappare le autorizzazioni per gli utenti di AWS IAM, definendo un nuovo blocco di configurazione per`mapUsers`, `data` in the `aws-auth`ConfigMap, che sostituisce il parametro rolearn per userarn, tuttavia come best practice è sempre consigliato all'utente. `mapRoles`

Per gestire le autorizzazioni, puoi modificare l'`aws-auth` ConfigMap aggiunta o la rimozione dell'accesso al tuo cluster Amazon EKS. Sebbene sia possibile modificarlo `aws-auth` ConfigMap manualmente, è consigliabile utilizzare strumenti come`eksctl`, poiché si tratta di una configurazione molto sensibile e una configurazione imprecisa può bloccarti all'esterno del cluster Amazon EKS. Consulta la sottosezione Usa gli strumenti per apportare modifiche all' ConfigMapaws-auth di seguito per maggiori dettagli.

Vantaggi rispetto alla gestione basata sugli accessi ConfigMap

1. Riduzione del rischio di errori di configurazione: la gestione diretta basata su API elimina gli errori comuni associati alla modifica manuale. ConfigMap Questo aiuta a prevenire eliminazioni accidentali o errori di sintassi che potrebbero impedire agli utenti di accedere al cluster.
2. Principio avanzato del privilegio minimo: elimina la necessità dell'autorizzazione di amministratore del cluster dall'identità del creatore del cluster e consente un'assegnazione delle autorizzazioni più granulare e appropriata. Puoi scegliere di aggiungere questa autorizzazione per i casi d'uso di break-glass.
3. Modello di sicurezza avanzato: fornisce la convalida integrata delle voci di accesso prima che vengano applicate. Inoltre, offre una più stretta integrazione con AWS IAM per l'autenticazione.
4. Operazioni semplificate: offre un modo più intuitivo per gestire le autorizzazioni tramite strumenti nativi di AWS.

Consigli per l'accesso al cluster

Combina IAM Identity Center con l'API CAM

- Gestione semplificata: utilizzando l'API Cluster Access Management in combinazione con IAM Identity Center, gli amministratori possono gestire l'accesso al cluster EKS insieme ad altri servizi AWS, riducendo la necessità di passare da un'interfaccia all'altra o di modificare manualmente. ConfigMaps

- Utilizza le voci di accesso per gestire le autorizzazioni Kubernetes dei principali IAM dall'esterno del cluster. Puoi aggiungere e gestire l'accesso al cluster utilizzando l'API EKS, l'interfaccia a riga di comando AWS, AWS SDKs CloudFormation, AWS e la console di gestione AWS. Ciò significa che puoi gestire gli utenti con gli stessi strumenti con cui hai creato il cluster.
- Sfrutta l'automazione, come dimostrato in questo esempio, per distribuire cluster con AWS IAM Identity Center come IdP con API CAM come punto di ingresso.
- Le autorizzazioni granulari di Kubernetes possono essere applicate mappando utenti o gruppi Kubernetes con principi IAM associati alle identità SSO tramite voci di accesso e policy di accesso.
- Per iniziare, segui Cambia la modalità di autenticazione per utilizzare le voci di accesso, quindi Migrazione delle voci aws-auth esistenti per accedere alle voci. ConfigMap

Rendi privato l'EKS Cluster Endpoint

Per impostazione predefinita, quando si effettua il provisioning di un cluster EKS, l'endpoint del cluster API è impostato su pubblico, ovvero è accessibile da Internet. Nonostante sia accessibile da Internet, l'endpoint è ancora considerato sicuro perché richiede che tutte le richieste API siano autenticate da IAM e quindi autorizzate da Kubernetes RBAC. Detto questo, se la tua politica di sicurezza aziendale impone di limitare l'accesso all'API da Internet o ti impedisce di instradare il traffico all'esterno del VPC del cluster, puoi:

- Configurate l'endpoint del cluster EKS in modo che sia privato. Per ulteriori informazioni su questo argomento, vedere Modifica dell'accesso agli endpoint del cluster.
- Lascia pubblico l'endpoint del cluster e specifica quali blocchi CIDR possono comunicare con l'endpoint del cluster. I blocchi sono in effetti un insieme di indirizzi IP pubblici inseriti nella lista bianca a cui è consentito accedere all'endpoint del cluster.
- Configura l'accesso pubblico con una serie di blocchi CIDR inseriti nella whitelist e imposta l'accesso privato agli endpoint su abilitato. Ciò consentirà l'accesso pubblico da una gamma specifica di utenti pubblici, forzando al IPs contempo tutto il traffico di rete tra i kubelet (worker) e l'API Kubernetes attraverso l'account incrociato ENIs che viene fornito nel VPC del cluster quando viene effettuato il provisioning del piano di controllo.

Non utilizzare un token di account di servizio per l'autenticazione

Un token di account di servizio è una credenziale statica di lunga durata. Se viene compromessa, smarrita o rubata, un utente malintenzionato potrebbe essere in grado di eseguire tutte le azioni associate a quel token fino all'eliminazione dell'account del servizio. A volte, potrebbe essere

necessario concedere un'eccezione per le applicazioni che devono utilizzare l'API Kubernetes dall'esterno del cluster, ad esempio un'applicazione di pipeline CI/CD. Se tali applicazioni vengono eseguite su un'infrastruttura AWS, come le istanze EC2, prendi in considerazione l'utilizzo di un profilo di istanza e la mappatura su un ruolo RBAC di Kubernetes.

Utilizza l'accesso meno privilegiato alle risorse AWS

A un utente IAM non è necessario assegnare privilegi alle risorse AWS per accedere all'API Kubernetes. Se devi concedere a un utente IAM l'accesso a un cluster EKS, crea una voce `aws-auth` ConfigMap per quell'utente che corrisponda a uno specifico gruppo Kubernetes RBAC.

Rimuovi le autorizzazioni di amministratore del cluster dal cluster creator principal

Per impostazione predefinita, i cluster Amazon EKS vengono creati con un'`cluster-admin`autorizzazione permanente associata al cluster creator principal. Con l'API Cluster Access Manager, è possibile creare cluster senza che questa autorizzazione imposti su`false`, quando si utilizza la `--access-config bootstrapClusterCreatorAdminPermissions` modalità `API_AND_CONFIG_MAP` di `API` autenticazione. La revoca di questo accesso è considerata una best practice per evitare modifiche indesiderate alla configurazione del cluster. La procedura per revocare questo accesso segue la stessa procedura per revocare qualsiasi altro accesso al cluster.

L'API offre la flessibilità necessaria per dissociare solo un principale IAM da una policy di accesso, in questo caso il. `AmazonEKSClusterAdminPolicy`

```
$ aws eks list-associated-access-policies \
    --cluster-name <CLUSTER_NAME> \
    --principal-arn <IAM_PRINCIPAL_ARN>

$ aws eks disassociate-access-policy --cluster-name <CLUSTER_NAME> \
    --principal-arn <IAM_PRINCIPAL_ARN. \
    --policy-arn arn:aws:eks::aws:cluster-access-policy/AmazonEKSClusterAdminPolicy
```

Oppure rimuovendo completamente la voce di accesso associata all'`cluster-admin`autorizzazione.

```
$ aws eks list-access-entries --cluster-name <CLUSTER_NAME>

{
    "accessEntries": []
}
```

```
$ aws eks delete-access-entry --cluster-name <CLUSTER_NAME> \
  --principal-arn <IAM_PRINCIPAL_ARN>
```

Questo accesso può essere nuovamente concesso se necessario durante uno scenario di incidente, emergenza o rottura del vetro in cui il cluster è altrimenti inaccessibile.

Se il cluster `aws-auth` ConfigMap è ancora configurato con il metodo di `CONFIG_MAP` autenticazione, a tutti gli utenti aggiuntivi dovrebbe essere concesso l'`aws-auth` ConfigMapaccesso al cluster tramite e dopo la configurazione il ruolo assegnato all'entità che ha creato il cluster, che può essere eliminato e ricreato solo in caso di incidente, emergenza o scenario di rottura del vetro, o se il cluster `aws-auth` ConfigMap è danneggiato e il cluster è altrimenti inaccessibile. Ciò può essere particolarmente utile nei cluster di produzione.

Usa IAM Roles quando più utenti necessitano di un accesso identico al cluster

Invece di creare una voce per ogni singolo utente IAM, consenti a tali utenti di assumere un ruolo IAM e mappare tale ruolo a un gruppo Kubernetes RBAC. Questo sarà più facile da gestire, soprattutto con l'aumentare del numero di utenti che richiedono l'accesso.

> **Important**
>
> Quando si accede al cluster EKS con l'entità IAM mappata da `aws-auth` ConfigMap, il nome utente descritto viene registrato nel campo utente del registro di controllo di Kubernetes. Se utilizzi un ruolo IAM, gli utenti effettivi che assumono quel ruolo non vengono registrati e non possono essere controllati.

Se usi ancora `aws-auth` ConfigMap come metodo di autenticazione, quando assegni le autorizzazioni RBAC di K8s a un ruolo IAM, dovresti includere\ {{}} nel tuo nome utente. SessionName In questo modo, il registro di controllo registrerà il nome della sessione in modo da poter tenere traccia dell'utente effettivo che assume questo ruolo insieme al registro. CloudTrail

```
- rolearn: arn:aws:iam::XXXXXXXXXXXX:role/testRole
  username: testRole:{{SessionName}}
  groups:
    - system:masters
```

> In Kubernetes 1.20 e versioni successive, questa modifica non è più necessaria, poiché è `user.extra.sessionName.0` stata aggiunta al registro di controllo di Kubernetes.

Utilizza l'accesso meno privilegiato durante la creazione e RoleBindings ClusterRoleBindings

Come il punto precedente sulla concessione dell'accesso alle risorse AWS, RoleBindings e ClusterRoleBindings dovrebbe includere solo il set di autorizzazioni necessarie per eseguire una funzione specifica. Evita di utilizzarlo `["*"]` nei tuoi ruoli e ClusterRoles a meno che non sia assolutamente necessario. Se non sei sicuro delle autorizzazioni da assegnare, prendi in considerazione l'utilizzo di uno strumento come audit2rbac per generare automaticamente ruoli e associazioni in base alle chiamate API osservate nel Kubernetes Audit Log.

Crea un cluster utilizzando un processo automatizzato

Come illustrato nei passaggi precedenti, quando si crea un cluster Amazon EKS, se non si utilizza la modalità di utilizzo `API_AND_CONFIG_MAP` o di `API` autenticazione e non si sceglie di delegare `cluster-admin` le autorizzazioni al creatore del cluster, all'utente o ruolo dell'entità IAM, ad esempio un utente federato che crea il cluster, vengono automaticamente concesse `system:masters` le autorizzazioni nella configurazione RBAC del cluster. Pur essendo una buona pratica per rimuovere questa autorizzazione, come descritto qui, se si utilizza il metodo di `CONFIG_MAP` autenticazione `aws-auth` ConfigMap, tale accesso non può essere revocato. Pertanto è una buona idea creare il cluster con una pipeline di automazione dell'infrastruttura legata a un ruolo IAM dedicato, senza che altri utenti o entità si assumano autorizzazioni e verificare regolarmente le autorizzazioni, le policy e chi ha accesso per attivare la pipeline. Inoltre, questo ruolo non deve essere utilizzato per eseguire azioni di routine sul cluster ed essere utilizzato esclusivamente per azioni a livello di cluster attivate dalla pipeline, ad esempio tramite modifiche al codice SCM.

Crea il cluster con un ruolo IAM dedicato

Quando crei un cluster Amazon EKS, all'utente o al ruolo dell'entità IAM, ad esempio un utente federato che crea il cluster, vengono automaticamente concesse `system:masters` le autorizzazioni nella configurazione RBAC del cluster. Questo accesso non può essere rimosso e non è gestito tramite. `aws-auth` ConfigMap Pertanto è una buona idea creare il cluster con un ruolo IAM dedicato e verificare regolarmente chi può assumere tale ruolo. Questo ruolo non deve essere utilizzato per eseguire azioni di routine sul cluster e, a tal fine, dovrebbe essere concesso l'accesso al cluster `aws-auth` ConfigMap ad altri utenti tramite il. Una volta `aws-auth` ConfigMap configurato, il ruolo deve essere protetto e utilizzato solo in modalità temporanea con privilegi elevati /break glass per scenari in cui il cluster è altrimenti inaccessibile. Ciò può essere particolarmente utile nei cluster in cui non è configurato l'accesso diretto degli utenti.

Controlla regolarmente l'accesso al cluster

È probabile che chi richiede l'accesso cambi nel tempo. Pianifica di controllarli periodicamente `aws-auth` ConfigMap per vedere a chi è stato concesso l'accesso e quali diritti sono stati assegnati. Puoi anche utilizzare strumenti open source come kubectl-who-canrbac-lookup per esaminare i ruoli associati a un particolare account, utente o gruppo di servizio. Esploreremo ulteriormente questo argomento quando arriveremo alla sezione sull'auditing. Altre idee sono disponibili in questo articolo di NCC Group.

Se ti affidi a **`aws-auth`** ConfigMap, usa gli strumenti per apportare modifiche.

Un aws-auth formattato in modo errato ConfigMap può causare la perdita dell'accesso al cluster. Se è necessario apportare modifiche al, utilizzare uno strumento. ConfigMap

eksctl La `eksctl` CLI include un comando per aggiungere mappature di identità a aws-auth. ConfigMap

Visualizza la guida CLI:

```
$ eksctl create iamidentitymapping --help
...
```

Controlla le identità mappate sul tuo cluster Amazon EKS.

```
$ eksctl get iamidentitymapping --cluster $CLUSTER_NAME --region $AWS_REGION
ARN                                                                    USERNAME
               GROUPS                                                   ACCOUNT
arn:aws:iam::788355785855:role/kube-system-<SELF_GENERATED_UUID>      system:node:
{{SessionName}}     system:bootstrappers,system:nodes,system:node-proxier
```

Assegna a un ruolo IAM un amministratore del cluster:

```
$ eksctl create iamidentitymapping --cluster  <CLUSTER_NAME> --region=<region> --arn
 arn:aws:iam::123456:role/testing --group system:masters --username admin
...
```

Per ulteriori informazioni, consulta i `eksctl`documenti

aws-auth di keikoproj

`aws-auth`by keikoproj include sia una libreria cli che una go.

Scarica e visualizza la guida CLI di aiuto:

```
$ go get github.com/keikoproj/aws-auth
...
$ aws-auth help
...
```

In alternativa, installalo `aws-auth` con il gestore di plugin krew per kubectl.

```
$ kubectl krew install aws-auth
...
$ kubectl aws-auth
...
```

Consulta i documenti aws-auth su GitHub per ulteriori informazioni, inclusa la libreria go.

CLI AWS IAM Authenticator

Il `aws-iam-authenticator` progetto include una CLI per l'aggiornamento di. ConfigMap

Scarica una versione su. GitHub

Aggiungi le autorizzazioni del cluster a un ruolo IAM:

```
$ ./aws-iam-authenticator add role --rolearn arn:aws:iam::185309785115:role/lil-dev-
role-cluster --username lil-dev-user --groups system:masters --kubeconfig ~/.kube/
config
...
```

Approcci alternativi all'autenticazione e alla gestione degli accessi

Sebbene IAM sia il modo preferito per autenticare gli utenti che devono accedere a un cluster EKS, è possibile utilizzare un provider di identità OIDC, ad esempio GitHub utilizzando un proxy di autenticazione e l'imitazione di Kubernetes. I post relativi a due di queste soluzioni sono stati pubblicati sul blog Open Source di AWS:

- Autenticazione su EKS tramite GitHub credenziali con Teleport
- Autenticazione OIDC coerente su più cluster EKS utilizzando kube-oidc-proxy

> **Important**
>
> EKS supporta nativamente l'autenticazione OIDC senza utilizzare un proxy. Per ulteriori informazioni, leggi il blog di lancio, Introducing OIDC identity provider authentication for Amazon EKS. Per un esempio che mostra come configurare EKS con Dex, un popolare provider OIDC open source con connettori per una varietà di metodi di autenticazione diversi, consulta Using Dex & dex-k8s-authenticator per l'autenticazione su Amazon EKS. Come descritto nei blog, gli utenti autenticati da un provider OIDC verranno visualizzati nel registro username/group di controllo di Kubernetes.

Puoi anche usare AWS SSO per federare AWS con un provider di identità esterno, ad esempio Azure AD. Se decidi di utilizzarlo, AWS CLI v2.0 include un'opzione per creare un profilo denominato che semplifica l'associazione di una sessione SSO alla sessione CLI corrente e l'assunzione di un ruolo IAM. Tieni presente che devi assumere un ruolo prima di iniziare, `kubectl` poiché il ruolo IAM viene utilizzato per determinare il gruppo RBAC Kubernetes dell'utente.

Identità e credenziali per i pod EKS

Alcune applicazioni eseguite all'interno di un cluster Kubernetes richiedono l'autorizzazione per chiamare l'API Kubernetes per funzionare correttamente. Ad esempio, il controller AWS Load Balancer deve essere in grado di elencare gli endpoint di un servizio. Il controller deve inoltre essere in grado di richiamare AWS per APIs effettuare il provisioning e configurare un ALB. In questa sezione esploreremo le migliori pratiche per l'assegnazione di diritti e privilegi ai Pods.

Account di servizio Kubernetes

Un account di servizio è un tipo speciale di oggetto che consente di assegnare un ruolo Kubernetes RBAC a un pod. Un account di servizio predefinito viene creato automaticamente per ogni Namespace all'interno di un cluster. Quando distribuisci un pod in un Namespace senza fare riferimento a un account di servizio specifico, l'account di servizio predefinito per quel Namespace verrà automaticamente assegnato al Pod e il Secret, ovvero il token dell'account di servizio (JWT) per quell'account di servizio, verrà montato sul pod come volume in. `/var/run/secrets/kubernetes.io/serviceaccount` La decodifica del token dell'account di servizio in quella directory rivelerà i seguenti metadati:

```
{
  "iss": "kubernetes/serviceaccount",
  "kubernetes.io/serviceaccount/namespace": "default",
```

```
  "kubernetes.io/serviceaccount/secret.name": "default-token-5pv4z",
  "kubernetes.io/serviceaccount/service-account.name": "default",
  "kubernetes.io/serviceaccount/service-account.uid":
 "3b36ddb5-438c-11ea-9438-063a49b60fba",
  "sub": "system:serviceaccount:default:default"
}
```

L'account di servizio predefinito dispone delle seguenti autorizzazioni per l'API Kubernetes.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  annotations:
    rbac.authorization.kubernetes.io/autoupdate: "true"
  creationTimestamp: "2020-01-30T18:13:25Z"
  labels:
    kubernetes.io/bootstrapping: rbac-defaults
  name: system:discovery
  resourceVersion: "43"
  selfLink: /apis/rbac.authorization.k8s.io/v1/clusterroles/system%3Adiscovery
  uid: 350d2ab8-438c-11ea-9438-063a49b60fba
rules:
- nonResourceURLs:
  - /api
  - /api/*
  - /apis
  - /apis/*
  - /healthz
  - /openapi
  - /openapi/*
  - /version
  - /version/
  verbs:
  - get
```

Questo ruolo autorizza gli utenti non autenticati e autenticati a leggere le informazioni sull'API ed è considerato sicuro in quanto accessibile al pubblico.

Quando un'applicazione in esecuzione all'interno di un Pod chiama Kubernetes APIs, al Pod deve essere assegnato un account di servizio che gli conceda esplicitamente l'autorizzazione a chiamarli. APIs Analogamente alle linee guida per l'accesso degli utenti, il ruolo o l'account ClusterRole associato a un account di servizio devono essere limitati alle risorse e ai metodi API di

cui l'applicazione ha bisogno per funzionare e nient'altro. Per utilizzare un account di servizio non predefinito, è sufficiente impostare il `spec.serviceAccountName` campo di un Pod sul nome dell'account di servizio che si desidera utilizzare. Per ulteriori informazioni sulla creazione di account di servizio, consultate https://kubernetes. io/docs/reference/access-authn-authz/rbac/#. service-account-permissions

> **Note**
>
> Prima di Kubernetes 1.24, Kubernetes creava automaticamente un segreto per ogni account di servizio. Questo segreto era montato sul pod all'indirizzo/var/run/secrets/ kubernetes.io/serviceaccounte veniva utilizzato dal pod per l'autenticazione sul server dell'API Kubernetes. In Kubernetes 1.24, un token di account di servizio viene generato dinamicamente all'esecuzione del pod ed è valido solo per un'ora per impostazione predefinita. Non verrà creato un segreto per l'account di servizio. Se hai un'applicazione che viene eseguita all'esterno del cluster e che deve autenticarsi nell'API Kubernetes, ad esempio Jenkins, dovrai creare un tipo segreto `kubernetes.io/service-account-token` insieme a un'annotazione che faccia riferimento all'account del servizio, ad esempio. `metadata.annotations.kubernetes.io/service-account.name: <SERVICE_ACCOUNT_NAME>` I segreti creati in questo modo non scadono.

IAM Roles for Service Accounts (IRSA)

IRSA è una funzionalità che consente di assegnare un ruolo IAM a un account di servizio Kubernetes. Funziona sfruttando una funzionalità di Kubernetes nota come Service Account Token Volume Projection. Quando i Pod sono configurati con un account di servizio che fa riferimento a un ruolo IAM, il server API Kubernetes chiamerà l'endpoint di rilevamento OIDC pubblico per il cluster all'avvio. L'endpoint firma crittograficamente il token OIDC emesso da Kubernetes e il token risultante montato come volume. Questo token firmato consente al Pod di chiamare il ruolo IAM APIs associato ad AWS. Quando viene richiamata un'API AWS, AWS SDKs chiama`sts:AssumeRoleWithWebIdentity`. Dopo aver convalidato la firma del token, IAM scambia il token emesso da Kubernetes con una credenziale di ruolo AWS temporanea.

Quando si utilizza IRSA, è importante riutilizzare le sessioni SDK AWS per evitare chiamate non necessarie ad AWS STS.

La decodifica del token (JWT) per IRSA produrrà un output simile all'esempio riportato di seguito:

```
{
```

```
  "aud": [
    "sts.amazonaws.com"
  ],
  "exp": 1582306514,
  "iat": 1582220114,
  "iss": "https://oidc.eks.us-west-2.amazonaws.com/id/
D43CF17C27A865933144EA99A26FB128",
  "kubernetes.io": {
    "namespace": "default",
    "pod": {
      "name": "alpine-57b5664646-rf966",
      "uid": "5a20f883-5407-11ea-a85c-0e62b7a4a436"
    },
    "serviceaccount": {
      "name": "s3-read-only",
      "uid": "a720ba5c-5406-11ea-9438-063a49b60fba"
    }
  },
  "nbf": 1582220114,
  "sub": "system:serviceaccount:default:s3-read-only"
}
```

Questo particolare token concede al Pod i privilegi di sola visualizzazione a S3 assumendo un ruolo IAM. Quando l'applicazione tenta di leggere da S3, il token viene scambiato con un set temporaneo di credenziali IAM simile al seguente:

```
{
    "AssumedRoleUser": {
        "AssumedRoleId": "AROA36C6WWEJULFUYMPB6:abc",
        "Arn": "arn:aws:sts::123456789012:assumed-role/eksctl-winterfell-addon-
iamserviceaccount-de-Role1-1D61LT75JH3MB/abc"
    },
    "Audience": "sts.amazonaws.com",
    "Provider": "arn:aws:iam::123456789012:oidc-provider/oidc.eks.us-
west-2.amazonaws.com/id/D43CF17C27A865933144EA99A26FB128",
    "SubjectFromWebIdentityToken": "system:serviceaccount:default:s3-read-only",
    "Credentials": {
        "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
        "SessionToken": "FwoGZXIvYXdzEGMaDMLxAZkuLpmSwYXShiL9A1S0X87VBC1mHCrRe/
pB2oesl1eXxUYnPJyC9ayOoXMvqXQsomq0xs6OqZ3vaa5Iw1HIyA4Cv1suLaOCoU3hNvOIJ6C94H1vU0siQYk7DIq9Av5RZ
Zv2A6zp5xGz9cWj2f0aD9v66vX4bexOs5t/YYhwuwAvkkJPSIGvxja0xRThnceHyFHKtj0Hbi/
PWAtlI8YJcDX69cM30JAHDdQHltm/4scFptW1hlvMaPWReCAaCrsHrATyka7ttw5YlUyvZ8EPogj6fwHlxmrXM9h1Bqdiko
LgkWsCTGlYcY8z3zkigJMbYn07ewTL5Ss7LazTJJa758I7PZan/
```

```
v3xQHd5DEc5WBneiV3iOznDFgup0VAMkIviVjVCkszaPSVEdK2NU7jtrh6Jfm7bU/3P6ZGCkyDLIa8MBn9KPXeJd/
yjTk5IifIwO/mDpGNUribg6TPxhzZ8b/XdZO1kS1gVgqjXyVCM+BRBh6C4H21w/eMzjCtDIpoxt5rGKL6Nu/
IFMipoC4fgx6LIIHwtGYMG7SWQi7OsMAkiwZRg0n68/RqWgLzBt/4pfjSRYuk=",
        "Expiration": "2020-02-20T18:49:50Z",
        "AccessKeyId": "ASIAIOSFODNN7EXAMPLE"
    }
}
```

Un webhook mutante che viene eseguito come parte del piano di controllo EKS inserisce l'AWS Role ARN e il percorso di un file di token di identità Web nel Pod come variabili di ambiente. Questi valori possono essere forniti anche manualmente.

```
AWS_ROLE_ARN=arn:aws:iam::AWS_ACCOUNT_ID:role/IAM_ROLE_NAME
AWS_WEB_IDENTITY_TOKEN_FILE=/var/run/secrets/eks.amazonaws.com/serviceaccount/token
```

Il kubelet ruoterà automaticamente il token proiettato quando supera l'80% del suo TTL totale o dopo 24 ore. AWS è SDKs responsabile del ricaricamento del token quando ruota. Per ulteriori informazioni su IRSA, consulta https://docs.aws.amazon.com/eks/ latest/userguide/iam - -technical-overview.html. roles-for-service-accounts

EKS Pod Identity

EKS Pod Identities è una funzionalità lanciata al re:Invent 2023 che ti consente di assegnare un ruolo IAM a un account di servizio Kubernetes, senza la necessità di configurare un provider di identità (IDP) Open Id Connect (OIDC) per ogni cluster del tuo account AWS. Per utilizzare EKS Pod Identity, devi implementare un agente che funzioni come pod su ogni nodo di lavoro idoneo. DaemonSet Questo agente è reso disponibile come componente aggiuntivo EKS ed è un prerequisito per utilizzare la funzione EKS Pod Identity. Le tue applicazioni devono utilizzare una versione supportata dell'SDK AWS per utilizzare questa funzionalità.

Quando le identità dei pod EKS sono configurate per un Pod, EKS monterà e aggiornerà un token di identità del pod su. `/var/run/secrets/pods.eks.amazonaws.com/serviceaccount/eks-pod-identity-token` Questo token verrà utilizzato dall'SDK AWS per comunicare con l'EKS Pod Identity Agent, che utilizza il token di identità del pod e il ruolo IAM dell'agente per creare credenziali temporanee per i tuoi pod chiamando l'API. AssumeRoleForPodIdentity Il token di identità del pod fornito ai tuoi pod è un JWT emesso dal tuo cluster EKS e firmato crittograficamente, con dichiarazioni JWT appropriate da utilizzare con EKS Pod Identities.

Per ulteriori informazioni su EKS Pod Identities, consulta questo blog.

Non è necessario apportare modifiche al codice dell'applicazione per utilizzare EKS Pod Identities. Le versioni AWS SDK supportate scopriranno automaticamente le credenziali rese disponibili con EKS Pod Identities utilizzando la catena di fornitori di credenziali. Come IRSA, le identità dei pod EKS impostano le variabili all'interno dei pod per indirizzarli a trovare le credenziali AWS.

Utilizzo dei ruoli IAM per EKS Pod Identities

- EKS Pod Identities può assumere direttamente solo un ruolo IAM che appartiene allo stesso account AWS del cluster EKS. Per accedere a un ruolo IAM in un altro account AWS, devi assumere quel ruolo configurando un profilo nella configurazione dell'SDK o nel codice dell'applicazione.
- Quando le identità EKS Pod vengono configurate per gli account di servizio, la persona o il processo che configura la Pod Identity Association deve avere l'`iam:PassRole`autorizzazione per quel ruolo.
- A ciascun account di servizio può essere associato un solo ruolo IAM tramite EKS Pod Identities, tuttavia è possibile associare lo stesso ruolo IAM a più account di servizio.
- I ruoli IAM utilizzati con EKS Pod Identities devono consentire al `pods.eks.amazonaws.com` Service Principal di assumerli e impostare i tag di sessione. Di seguito è riportato un esempio di policy di trust dei ruoli che consente a EKS Pod Identities di utilizzare un ruolo IAM:

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "pods.eks.amazonaws.com"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ],
      "Condition": {
        "StringEquals": {
          "aws:SourceOrgId": "${aws:ResourceOrgId}"
        }
      }
    }
  ]
```

```
}
```

AWS consiglia di utilizzare chiavi condizionali come quelle `aws:SourceOrgId` che aiutano a proteggersi dal problema del sostituto confuso tra diversi servizi. Nell'esempio precedente, role trust policy, `ResourceOrgId` è una variabile uguale all'ID dell'organizzazione AWS Organizations dell'organizzazione AWS a cui appartiene l'account AWS. EKS passerà un valore `aws:SourceOrgId` uguale a quello quando assumerà un ruolo con EKS Pod Identities.

ABAC ed EKS Pod Identities

Quando EKS Pod Identities assume un ruolo IAM, imposta i seguenti tag di sessione:

Tag di sessione EKS Pod Identities	Valore
spazio dei nomi kubernetes	Lo spazio dei nomi in cui viene eseguito il pod associato a EKS Pod Identities.
kubernetes-service-account	Il nome dell'account del servizio Kubernetes associato a EKS Pod Identities
eks-cluster-arn	L'ARN del cluster EKS, ad es. `arn:${Partition}:eks:${Region}:${Account}:cluster/${ClusterName}` L'ARN del cluster è unico, ma se un cluster viene eliminato e ricreato nella stessa regione con lo stesso nome, all'interno dello stesso account AWS, avrà lo stesso ARN.
eks-cluster-name	Il nome del cluster EKS. Tieni presente che i nomi dei cluster EKS possono essere gli stessi all'interno del tuo account AWS e i cluster EKS in altri account AWS.
kubernetes-pod-name	Il nome del pod in EKS.
kubernetes-pod-uid	L'UID del pod in EKS.

Questi tag di sessione consentono di utilizzare Attribute Based Access Control (ABAC) per concedere l'accesso alle risorse AWS solo a specifici account di servizio Kubernetes. Nel fare ciò, è molto

importante capire che gli account dei servizi Kubernetes sono unici solo all'interno di uno spazio dei nomi e gli spazi dei nomi Kubernetes sono unici solo all'interno di un cluster EKS. È possibile accedere a questi tag di sessione nelle politiche AWS utilizzando la chiave `aws:PrincipalTag/<tag-key>` global condition, ad esempio `aws:PrincipalTag/eks-cluster-arn`

Ad esempio, se desideri concedere l'accesso solo a un account di servizio specifico per accedere a una risorsa AWS nel tuo account con una IAM o una policy delle risorse, dovresti controllare `eks-cluster-arn` e `kubernetes-namespace` `kubernetes-service-account` etichettare e assicurarti che solo gli account di servizio del cluster previsto abbiano accesso a quella risorsa, come altri cluster potrebbero avere lo stesso `kubernetes-service-accounts` e`kubernetes-namespaces`.

Questo esempio di policy S3 Bucket concede l'accesso agli oggetti nel bucket S3 a cui è collegato, solo se `eks-cluster-arn` tutti soddisfano i valori previsti `kubernetes-service-accountkubernetes-namespace`, dove il cluster EKS è ospitato nell'account AWS. `111122223333`

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:root"
            },
            "Action": "s3:*",
            "Resource":            [
                "arn:aws:s3:::ExampleBucket/*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:PrincipalTag/kubernetes-service-account": "s3objectservice",
                    "aws:PrincipalTag/eks-cluster-arn": "arn:aws:eks:us-
west-2:111122223333:cluster/ProductionCluster",
                    "aws:PrincipalTag/kubernetes-namespace": "s3datanamespace"
                }
            }
        }
    ]
}
```

EKS Pod Identities rispetto a IRSA

Sia EKS Pod Identities che IRSA sono metodi preferiti per fornire credenziali AWS temporanee ai tuoi pod EKS. A meno che non abbiate casi d'uso specifici per IRSA, vi consigliamo di utilizzare EKS Pod Identities quando utilizzate EKS. Questa tabella consente di confrontare le due funzionalità.

#	EKS Pod Identity	IRSA
Richiede l'autorizzazione per creare un IDP OIDC nei tuoi account AWS?	No	Sì
Richiede una configurazione IDP unica per cluster	No	Sì
Imposta i tag di sessione pertinenti da utilizzare con ABAC	Sì	No
Richiede un iam: PassRole Check?	Sì	No
Utilizza AWS STS Quota dal tuo account AWS?	No	Sì
Può accedere ad altri account AWS	Indirettamente con il concatenamento dei ruoli	Direttamente con sts: AssumeRoleWithWebIdentity
Compatibile con AWS SDKs	Sì	Sì
Richiede Pod Identity Agent Daemonset sui nodi?	Sì	No

Identità e credenziali per i pod EKS: raccomandazioni

Aggiorna il daemonset aws-node per utilizzare IRSA

Al momento, il daemonset aws-node è configurato per utilizzare un ruolo assegnato alle istanze EC2 da assegnare ai pod. IPs Questo ruolo include diverse policy gestite da AWS, ad esempio

Amazoneks_CNI_Policy, che consentono efficacemente a tutti i pod in esecuzione su un nodo di indirizzare gli indirizzi IP o di EC2 ContainerRegistryReadOnly estrarre immagini da ECR. attach/detach ENIs assign/unassign Poiché ciò rappresenta un rischio per il cluster, si consiglia di aggiornare il daemset aws-node per utilizzare IRSA. Uno script per eseguire questa operazione è disponibile nell'archivio di questa guida.

Il daemonset aws-node supporta EKS Pod Identities nelle versioni v1.15.5 e successive.

Limita l'accesso al profilo dell'istanza assegnato al nodo di lavoro

Quando si utilizzano IRSA o EKS Pod Identities, la catena di credenziali del pod viene aggiornata in modo da utilizzare innanzitutto le identità IRSA o EKS Pod. Tuttavia, il pod può comunque ereditare i diritti del profilo di istanza assegnato al nodo di lavoro. Per i pod che non richiedono queste autorizzazioni, puoi bloccare l'accesso ai metadati dell'istanza per garantire che le applicazioni dispongano solo delle autorizzazioni necessarie e non dei relativi nodi.

> **Warning**
>
> Il blocco dell'accesso ai metadati dell'istanza impedirà ai pod che non utilizzano IRSA o EKS Pod Identities di ereditare il ruolo assegnato al nodo di lavoro.

È possibile bloccare l'accesso ai metadati dell'istanza richiedendo che l'istanza utilizzi IMDSv2 solo l'istanza e aggiornando il conteggio degli hop a 1, come nell'esempio seguente. Puoi anche includere queste impostazioni nel modello di avvio del gruppo di nodi. Non disabilitate i metadati dell'istanza poiché ciò impedirà il corretto funzionamento di componenti come il gestore di terminazione dei nodi e altri elementi che si basano sui metadati dell'istanza.

```
$ aws ec2 modify-instance-metadata-options --instance-id <value> --http-tokens required
 --http-put-response-hop-limit 1
...
```

Se utilizzi Terraform per creare modelli di avvio da utilizzare con Managed Node Groups, aggiungi il blocco di metadati per configurare il conteggio degli hop come mostrato in questo frammento di codice:

```
tf hl_lines="7" resource "aws_launch_template" "foo" { name = "foo" …
 metadata_options { http_endpoint = "enabled" http_tokens = "required"
http_put_response_hop_limit = 1 instance_metadata_tags = "enabled" } …
```

Puoi anche bloccare l'accesso di un pod ai metadati EC2 manipolando iptables sul nodo. Per ulteriori informazioni su questo metodo, consulta Limitazione dell'accesso al servizio di metadati dell'istanza.

Se hai un'applicazione che utilizza una versione precedente dell'SDK AWS che non supporta IRSA o EKS Pod Identities, devi aggiornare la versione SDK.

Ambita la policy di fiducia di IAM Role per IRSA Roles in base al nome dell'account di servizio, allo spazio dei nomi e al cluster

La policy di fiducia può essere estesa a un Namespace o a un account di servizio specifico all'interno di un Namespace. Quando si utilizza IRSA, è meglio rendere la policy di fiducia del ruolo il più esplicita possibile includendo il nome dell'account del servizio. Ciò impedirà efficacemente che altri Pod all'interno dello stesso Namespace assumano il ruolo. La CLI lo `eksctl` farà automaticamente quando la utilizzerai per creare ruoli di servizio accounts/IAM . Vedi https://eksctl. io/usage/iamserviceaccounts/per ulteriori informazioni.

Quando si lavora direttamente con IAM, si aggiunge una condizione alla politica di fiducia del ruolo, che utilizza condizioni per garantire che il `:sub` claim corrisponda allo spazio dei nomi e all'account di servizio previsti. Ad esempio, prima avevamo un token IRSA con un'affermazione secondaria di «system:serviceaccount:default:s3-read-only». Questo è lo spazio `s3-read-only` dei nomi e l'account del servizio è. `default` Dovresti utilizzare una condizione come la seguente per assicurarti che solo il tuo account di servizio in un determinato namespace del tuo cluster possa assumere quel ruolo:

```
 "Condition": {
     "StringEquals": {
         "oidc.eks.us-west-2.amazonaws.com/id/D43CF17C27A865933144EA99A26FB128:aud":
"sts.amazonaws.com",
         "oidc.eks.us-west-2.amazonaws.com/id/D43CF17C27A865933144EA99A26FB128:sub":
"system:serviceaccount:default:s3-read-only"
     }
 }
```

Utilizza un ruolo IAM per applicazione

Sia con IRSA che con EKS Pod Identity, è consigliabile assegnare a ciascuna applicazione il proprio ruolo IAM. Ciò offre un migliore isolamento in quanto è possibile modificare un'applicazione senza influire su un'altra e consente di applicare il principio del privilegio minimo concedendo a un'applicazione solo le autorizzazioni necessarie.

Quando si utilizza ABAC con EKS Pod Identity, è possibile utilizzare un ruolo IAM comune su più account di servizio e fare affidamento sui relativi attributi di sessione per il controllo degli accessi. Ciò è particolarmente utile quando si opera su larga scala, poiché ABAC consente di operare con un minor numero di ruoli IAM.

Quando la tua applicazione necessita dell'accesso a IMDS, usa IMDSv2 e aumenta il limite di hop sulle istanze EC2 a 2

IMDSv2richiede l'utilizzo di una richiesta PUT per ottenere un token di sessione. La richiesta PUT iniziale deve includere un TTL per il token di sessione. Le versioni più recenti degli SDK AWS gestiranno automaticamente questo e il rinnovo di detto token. È anche importante tenere presente che il limite di hop predefinito sulle istanze EC2 è intenzionalmente impostato su 1 per impedire l'inoltro IP. Di conseguenza, i Pod che richiedono un token di sessione eseguito su istanze EC2 potrebbero alla fine scadere e tornare a utilizzare il flusso di dati. IMDSv1 EKS aggiunge il supporto IMDSv2 abilitando sia la v1 che la v2 e modificando il limite di hop a 2 sui nodi forniti da eksctl o con i modelli ufficiali. CloudFormation

Disattiva il montaggio automatico dei token degli account di servizio

Se la tua applicazione non ha bisogno di chiamare l'API Kubernetes, imposta l'`automountServiceAccountToken`attributo su `false` in PodSpec for your application o applica una patch all'account di servizio predefinito in ogni namespace in modo che non venga più montato automaticamente sui pod. Esempio:

```
kubectl patch serviceaccount default -p $'automountServiceAccountToken: false'
```

Utilizza account di servizio dedicati per ogni applicazione

Ogni applicazione deve avere un proprio account di servizio dedicato. Questo vale per gli account di servizio per l'API Kubernetes, nonché per IRSA ed EKS Pod Identity.

> **Important**
>
> Se si utilizza un blue/green approccio agli aggiornamenti del cluster anziché eseguire un aggiornamento del cluster sul posto quando si utilizza IRSA, sarà necessario aggiornare la policy di fiducia di ciascuno dei ruoli IRSA IAM con l'endpoint OIDC del nuovo cluster. Un aggiornamento del blue/green cluster consiste nella creazione di un cluster che esegue una versione più recente di Kubernetes insieme al vecchio cluster e nell'utilizzo di un

sistema di bilanciamento del carico o di una service mesh per spostare senza problemi il traffico dai servizi in esecuzione sul vecchio cluster al nuovo cluster. Quando si utilizzano gli aggiornamenti blue/green del cluster con EKS Pod Identity, è necessario creare associazioni di identità dei pod tra i ruoli IAM e gli account di servizio nel nuovo cluster. E aggiorna la policy di fiducia dei ruoli IAM se hai una `sourceArn` condizione.

Esegui l'applicazione come utente non root

I contenitori vengono eseguiti come root per impostazione predefinita. Sebbene ciò consenta loro di leggere il file del token di identità Web, l'esecuzione di un contenitore come root non è considerata una buona pratica. In alternativa, valuta la possibilità di aggiungere l'`spec.securityContext.runAsUser`attributo a PodSpec. Il valore di `runAsUser` è un valore arbitrario.

Nell'esempio seguente, tutti i processi all'interno del Pod verranno eseguiti con l'ID utente specificato nel `runAsUser` campo.

```
apiVersion: v1
kind: Pod
metadata:
  name: security-context-demo
spec:
  securityContext:
    runAsUser: 1000
    runAsGroup: 3000
  containers:
  - name: sec-ctx-demo
    image: busybox
    command: [ "sh", "-c", "sleep 1h" ]
```

Quando si esegue un contenitore come utente non root, si impedisce al contenitore di leggere il token dell'account del servizio IRSA perché al token vengono assegnate le autorizzazioni 0600 [root] per impostazione predefinita. Se aggiorni SecurityContext per il tuo contenitore in modo da includere fsgroup=65534 [Nobody], consentirà al contenitore di leggere il token.

```
spec:
  securityContext:
    fsGroup: 65534
```

In Kubernetes 1.19 e versioni successive, questa modifica non è più necessaria e le applicazioni possono leggere il token dell'account del servizio IRSA senza aggiungerle al gruppo Nobody.

Concedi l'accesso meno privilegiato alle applicazioni

Action Hero è un'utilità che puoi eseguire insieme all'applicazione per identificare le chiamate API AWS e le autorizzazioni IAM corrispondenti di cui l'applicazione ha bisogno per funzionare correttamente. È simile a IAM Access Advisor in quanto consente di limitare gradualmente l'ambito dei ruoli IAM assegnati alle applicazioni. Consulta la documentazione sulla concessione dell'accesso con privilegi minimi alle risorse AWS per ulteriori informazioni.

Valuta la possibilità di impostare un limite di autorizzazioni per i ruoli IAM utilizzati con IRSA e Pod Identities. Puoi utilizzare il limite delle autorizzazioni per garantire che i ruoli utilizzati da IRSA o Pod Identities non possano superare il livello massimo di autorizzazioni. Per una guida di esempio su come iniziare con i limiti delle autorizzazioni con un esempio di politica sui limiti delle autorizzazioni, consulta questo repository su github.

Rivedi e revoca l'accesso anonimo non necessario al tuo cluster EKS

Idealmente, l'accesso anonimo dovrebbe essere disabilitato per tutte le azioni API. L'accesso anonimo viene concesso creando un RoleBinding or ClusterRoleBinding per il sistema utente integrato di Kubernetes: anonymous. Puoi utilizzare lo strumento rbac-lookup per identificare le autorizzazioni che system:anonymous user ha sul tuo cluster:

```
./rbac-lookup | grep -P 'system:(anonymous)|(unauthenticated)'
system:anonymous               cluster-wide        ClusterRole/system:discovery
system:unauthenticated         cluster-wide        ClusterRole/system:discovery
system:unauthenticated         cluster-wide        ClusterRole/system:public-info-
viewer
```

Qualsiasi ruolo o ClusterRole diverso da system: non public-info-viewer deve essere associato a system:anonymous user o system:unauthenticated group.

Potrebbero esserci alcuni motivi legittimi per abilitare l'accesso anonimo a determinati utenti. APIs Se questo è il caso del tuo cluster, assicurati che solo quelli specifici APIs siano accessibili agli utenti anonimi e l'esposizione di quelli APIs senza autenticazione non rende il cluster vulnerabile.

Prima della Kubernetes/EKS versione 1.14, system:unauthenticated group era associato a system:discovery e system:basic-user per impostazione predefinita. ClusterRoles Tieni presente che anche se hai aggiornato il cluster alla versione 1.14 o successiva, queste autorizzazioni potrebbero

essere ancora abilitate sul cluster, poiché gli aggiornamenti del cluster non revocano queste autorizzazioni. Per verificare quali ClusterRoles hanno «system:unauthenticated» tranne system: public-info-viewer puoi eseguire il seguente comando (richiede jq util):

```
kubectl get ClusterRoleBinding -o json | jq -r '.items[] | select(.subjects[]?.name
 =="system:unauthenticated") | select(.metadata.name != "system:public-info-viewer")
 | .metadata.name'
```

E «system:unauthenticated» può essere rimosso da tutti i ruoli tranne «system:" usando: public-info-viewer

```
kubectl get ClusterRoleBinding -o json | jq -r '.items[] | select(.subjects[]?.name
 =="system:unauthenticated") | select(.metadata.name != "system:public-info-viewer") |
 del(.subjects[] | select(.name =="system:unauthenticated"))' | kubectl apply -f -
```

In alternativa, puoi controllarlo e rimuoverlo manualmente con kubectl describe e kubectl edit. Per verificare se system:unauthenticated group ha i permessi system:discovery sul tuo cluster, esegui il seguente comando:

```
kubectl describe clusterrolebindings system:discovery

Name:         system:discovery
Labels:       kubernetes.io/bootstrapping=rbac-defaults
Annotations:  rbac.authorization.kubernetes.io/autoupdate: true
Role:
  Kind:  ClusterRole
  Name:  system:discovery
Subjects:
  Kind   Name                    Namespace
  ----   ----                    ---------
  Group  system:authenticated
  Group  system:unauthenticated
```

Per verificare se system:unauthenticated group dispone dell'autorizzazione system:basic-user sul tuo cluster, esegui il seguente comando:

```
kubectl describe clusterrolebindings system:basic-user

Name:         system:basic-user
Labels:       kubernetes.io/bootstrapping=rbac-defaults
Annotations:  rbac.authorization.kubernetes.io/autoupdate: true
```

```
Role:
  Kind:  ClusterRole
  Name:  system:basic-user
Subjects:
  Kind   Name                    Namespace
  ----   ----                    ---------
  Group  system:authenticated
  Group  system:unauthenticated
```

Se system:unauthenticated group è associato a system:discovery e/o system:basic-user sul tuo cluster, dovresti dissociare questi ruoli dal gruppo system:unauthenticated. ClusterRoles Modifica ClusterRoleBinding system:discovery usando il seguente comando:

```
kubectl edit clusterrolebindings system:discovery
```

Il comando precedente aprirà la definizione corrente di system:discovery ClusterRoleBinding in un editor come mostrato di seguito:

```
# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this file will
 be
# reopened with the relevant failures.
#
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  annotations:
    rbac.authorization.kubernetes.io/autoupdate: "true"
  creationTimestamp: "2021-06-17T20:50:49Z"
  labels:
    kubernetes.io/bootstrapping: rbac-defaults
  name: system:discovery
  resourceVersion: "24502985"
  selfLink: /apis/rbac.authorization.k8s.io/v1/clusterrolebindings/system%3Adiscovery
  uid: b7936268-5043-431a-a0e1-171a423abeb6
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: system:discovery
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
```

```
  name: system:authenticated
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: system:unauthenticated
```

Eliminate la voce relativa al gruppo system:unauthenticated dalla sezione «topics» nella schermata precedente dell'editor.

Ripeti gli stessi passaggi per system:basic-user. ClusterRoleBinding

Riutilizza le sessioni SDK AWS con IRSA

Quando usi IRSA, le applicazioni scritte con l'SDK AWS utilizzano il token distribuito ai tuoi pod per effettuare chiamate `sts:AssumeRoleWithWebIdentity` per generare credenziali AWS temporanee. Questo è diverso dagli altri servizi di calcolo AWS, in cui il servizio di elaborazione fornisce credenziali AWS temporanee direttamente alla risorsa di calcolo AWS, come una funzione lambda. Ciò significa che ogni volta che viene inizializzata una sessione SDK AWS, viene effettuata una chiamata ad AWS `AssumeRoleWithWebIdentity` STS for. Se la tua applicazione si ridimensiona rapidamente e inizializza molte sessioni SDK AWS, potresti riscontrare una limitazione da parte di AWS STS poiché il tuo codice effettuerà molte chiamate. `AssumeRoleWithWebIdentity`

Per evitare questo scenario, consigliamo di riutilizzare le sessioni SDK AWS all'interno dell'applicazione in modo da non effettuare chiamate non `AssumeRoleWithWebIdentity` necessarie.

Nel codice di esempio seguente, viene creata una sessione utilizzando l'SDK python boto3 e quella stessa sessione viene utilizzata per creare client e interagire con Amazon S3 e Amazon SQS. `AssumeRoleWithWebIdentity`viene chiamato solo una volta e l'SDK AWS aggiornerà automaticamente le credenziali `my_session` quando scadono.

```
import boto3

= Create your own session

my_session = boto3.session.Session()

= Now we can create low-level clients from our session

sqs = my_session.client(``sqs``) s3 = my_session.client(``s3``)
```

```
s3response = s3.list_buckets() sqsresponse = sqs.list_queues()

#print the response from the S3 and SQS APIs print("`s3 response:`")
print(s3response) print("`—`") print("`sqs response:`")
print(sqsresponse)
```

Se stai migrando un'applicazione da un altro servizio di calcolo AWS, come EC2, a EKS con IRSA, questo è un dettaglio particolarmente importante. Su altri servizi di elaborazione, l'inizializzazione di una sessione SDK AWS non richiama AWS STS a meno che tu non lo richieda.

Approcci alternativi

Sebbene le identità dei pod IRSA e EKS siano i metodi preferiti per assegnare un'identità AWS a un pod, richiedono l'inclusione di una versione recente di AWS SDKs nell'applicazione. Per un elenco completo di quelli SDKs che attualmente supportano IRSA, consulta -.html. https:// docs.aws.amazon.com/eks/ latest/userguide/iam-roles-for-service-accounts-minimum-sdk.html, for EKS Pod Identities, see https://docs.aws.amazon.com/eks/latest/userguide/pod id-minimum-sdk Se disponi di un'applicazione che non puoi aggiornare immediatamente con un SDK compatibile, sono disponibili diverse soluzioni create dalla community per assegnare ruoli IAM ai pod Kubernetes, tra cui kube2iam e kiam. Sebbene AWS non approvi, approvi o supporti l'uso di queste soluzioni, esse vengono spesso utilizzate dalla comunità in generale per ottenere risultati simili a quelli di IRSA ed EKS Pod Identities.

Se hai bisogno di utilizzare una di queste soluzioni non fornite da AWS, esercita la dovuta diligenza e assicurati di comprendere le implicazioni di sicurezza che ciò comporta.

Strumenti e risorse

- Workshop di immersione nella sicurezza di Amazon EKS - Identity and Access Management
- Modello Terraform EKS Blueprints - Cluster Amazon EKS completamente privato
- Modello di progetti Terraform EKS - Single Sign-On di IAM Identity Center per Amazon EKS Cluster
- Modello Terraform EKS Blueprints - Okta Single Sign-On per Amazon EKS Cluster
- audit2rbac
- rbac.dev Un elenco di risorse aggiuntive, inclusi blog e strumenti, per Kubernetes RBAC
- Eroe d'azione
- kube2iam
- kiam

Sicurezza Pod

> **Tip**
>
> Esplora le best practice tramite i workshop Amazon EKS.

Le specifiche del pod includono una varietà di attributi diversi che possono rafforzare o indebolire il tuo livello di sicurezza generale. In qualità di professionista di Kubernetes, la tua preoccupazione principale dovrebbe essere impedire che un processo in esecuzione in un container sfugga ai limiti di isolamento del runtime del container e ottenga l'accesso all'host sottostante.

Funzionalità Linux

Per impostazione predefinita, i processi eseguiti all'interno di un contenitore vengono eseguiti nel contesto dell'utente root [Linux]. Sebbene le azioni di root all'interno di un contenitore siano parzialmente limitate dall'insieme di funzionalità Linux che il runtime del contenitore assegna ai contenitori, questi privilegi predefiniti potrebbero consentire a un utente malintenzionato di aumentare i propri privilegi .7.html. and/or gain access to sensitive information bound to the host, including Secrets and ConfigMaps. Below is a list of the default capabilities assigned to containers. For additional information about each capability, see http://man7.org/linux/man-pages/man7/capabilities

```
CAP_AUDIT_WRITE, CAP_CHOWN, CAP_DAC_OVERRIDE, CAP_FOWNER, CAP_FSETID,
CAP_KILL, CAP_MKNOD, CAP_NET_BIND_SERVICE, CAP_NET_RAW, CAP_SETGID,
CAP_SETUID, CAP_SETFCAP, CAP_SETPCAP, CAP_SYS_CHROOT
```

Example

Per impostazione predefinita, ai pod EC2 e Fargate vengono assegnate le funzionalità sopra menzionate. Inoltre, le funzionalità Linux possono essere eliminate solo dai pod Fargate.

I pod eseguiti con privilegi ereditano tutte le funzionalità Linux associate al root sull'host. Questo dovrebbe essere evitato se possibile.

Autorizzazione dei nodi

Tutti i nodi di lavoro Kubernetes utilizzano una modalità di autorizzazione chiamata Node Authorization. Node Authorization autorizza tutte le richieste API che provengono dal kubelet e consente ai nodi di eseguire le seguenti azioni:

Operazioni di lettura:

- services
- endpoint
- nodi
- pod
- segreti, mappe di configurazione, dichiarazioni di volume persistenti e volumi persistenti relativi ai pod legati al nodo di Kubelet

Operazioni di scrittura:

- nodi e stato del nodo (abilita il plugin di `NodeRestriction` ammissione per limitare un kubelet a modificare il proprio nodo)
- pods e pod status (abilita il plugin di `NodeRestriction` ammissione per limitare un kubelet a modificare i pod legati a se stesso)
- eventi

Operazioni relative all'autenticazione:

- Accesso in lettura/scrittura all'API CertificateSigningRequest (CSR) per il bootstrap TLS
- la capacità di creare e per controlli delegati TokenReview SubjectAccessReview authentication/ authorization

EKS utilizza il controller di ammissione delle restrizioni del nodo che consente al nodo di modificare solo un insieme limitato di attributi del nodo e oggetti pod associati al nodo. Tuttavia, un utente malintenzionato che riesce ad accedere all'host sarà comunque in grado di raccogliere informazioni sensibili sull'ambiente dall'API Kubernetes che potrebbero consentirgli di spostarsi lateralmente all'interno del cluster.

Soluzioni di sicurezza Pod

Politica di sicurezza Pod (PSP)

In passato, le risorse Pod Security Policy (PSP) venivano utilizzate per specificare una serie di requisiti che i pod dovevano soddisfare prima di poter essere creati. A partire dalla versione

1.21 di Kubernetes, PSP è diventata obsoleta. La loro rimozione è prevista nella versione 1.25 di Kubernetes.

> **Important**
>
> PSPs sono obsoleti nella versione 1.21 di Kubernetes. Avrai tempo fino alla versione 1.25 o circa 2 anni per passare a un'alternativa. Questo documento spiega la motivazione di questa deprecazione.

Migrazione a una nuova soluzione di sicurezza dei pod

Poiché PSPs sono stati rimossi a partire dalla versione 1.25 di Kubernetes, gli amministratori e gli operatori del cluster devono sostituire tali controlli di sicurezza. Due soluzioni possono soddisfare questa esigenza:

- Policy-as-code soluzioni (PAC) dell'ecosistema Kubernetes
- Standard di sicurezza Kubernetes Pod (PSS)

Entrambe le soluzioni PAC e PSS possono coesistere con PSP; possono essere utilizzate in cluster prima che PSP venga rimossa. Questo facilita l'adozione durante la migrazione da PSP. Consulta questo documento per valutare la migrazione da PSP a PSS.

Kyverno, una delle soluzioni PAC descritte di seguito, ha fornito linee guida specifiche in un post sul blog per la migrazione dalla sua soluzione, che includono politiche analoghe, PSPs confronti di funzionalità e una procedura di migrazione. Ulteriori informazioni e linee guida sulla migrazione a Kyverno rispetto a Pod Security Admission (PSA) sono state pubblicate sul blog di AWS qui.

Policy-as-code (PAC)

Policy-as-code Le soluzioni (PAC) forniscono barriere per guidare gli utenti del cluster e prevenire comportamenti indesiderati, attraverso controlli prescritti e automatizzati. PAC utilizza i Kubernetes Dynamic Admission Controller per intercettare il flusso di richieste del server dell'API Kubernetes, tramite una chiamata webhook, e modificare e convalidare i payload delle richieste, in base a policy scritte e archiviate come codice. La mutazione e la convalida avvengono prima che la richiesta del server API comporti una modifica al cluster. Le soluzioni PAC utilizzano politiche per abbinare e agire sui payload delle richieste del server API, in base alla tassonomia e ai valori.

Sono disponibili diverse soluzioni PAC open source per Kubernetes. Queste soluzioni non fanno parte del progetto Kubernetes, ma provengono dall'ecosistema Kubernetes. Alcune soluzioni PAC sono elencate di seguito.

- OPA/Gatekeeper
- Open Policy Agent (OPA)
- Kyverno
- Kubewarden
- Politica JS

Per ulteriori informazioni sulle soluzioni PAC e su come aiutarvi a scegliere la soluzione appropriata per le vostre esigenze, consultate i collegamenti seguenti.

- Contromisure basate su politiche per Kubernetes — Parte 1
- Contromisure basate su politiche per Kubernetes — Parte 2

Pod Security Standards (PSS) e Pod Security Admission (PSA)

In risposta all'obsolescenza della PSP e alla continua necessità di controllare la sicurezza dei pod out-of-the-box, con una soluzione Kubernetes integrata, il Kubernetes Auth Special Interest Group ha creato i Pod Security Standards (PSS) e Pod Security Admission (PSA). L'iniziativa PSA include un progetto webhook per i controller di ammissione che implementa i controlli definiti nel PSS. Questo approccio di controllo delle ammissioni è simile a quello utilizzato nelle soluzioni PAC.

Secondo la documentazione di Kubernetes, il PSS «definisce tre diverse politiche per coprire ampiamente lo spettro della sicurezza. Queste politiche sono cumulative e vanno da quelle altamente permissive a quelle altamente restrittive. `»

Queste politiche sono definite come:

- Privilegiato: politica senza restrizioni (non sicura), che fornisce il livello di autorizzazioni più ampio possibile. Questa politica consente l'aumento dei privilegi noto. È l'assenza di una politica. Ciò è utile per applicazioni come agenti di registrazione CNIs, driver di archiviazione e altre applicazioni a livello di sistema che richiedono un accesso privilegiato.
- Linea di base: politica minimamente restrittiva che impedisce l'aumento dei privilegi noti. Consente la configurazione Pod predefinita (specificata in modo minimo). La politica di base vieta l'uso di

HostNetwork, HostPID, HostIPC, HostPath, HostPort, l'impossibilità di aggiungere funzionalità Linux e diverse altre restrizioni.

- Limitato: politica fortemente limitata, che segue le attuali best practice di rafforzamento dei Pod. Questa politica eredita dalla linea di base e aggiunge ulteriori restrizioni, come l'impossibilità di essere eseguita come root o come gruppo root. Le politiche limitate possono influire sulla capacità di funzionamento di un'applicazione. Sono principalmente destinate all'esecuzione di applicazioni critiche per la sicurezza.

Queste policy definiscono i profili per l'esecuzione dei pod, disposti in tre livelli di accesso privilegiato e limitato.

Per implementare i controlli definiti dal PSS, PSA opera in tre modalità:

- imporre: le violazioni delle norme comporteranno il rifiuto del pod.
- audit: le violazioni delle politiche attiveranno l'aggiunta di un'annotazione di controllo all'evento registrato nel registro di controllo, ma sono altrimenti consentite.
- avviso: le violazioni delle norme attiveranno un avviso rivolto all'utente, ma sono altrimenti consentite.

Queste modalità e i livelli di profilo (restrizione) sono configurati a livello di Kubernetes Namespace, utilizzando etichette, come illustrato nell'esempio seguente.

```
apiVersion: v1
kind: Namespace
metadata:
  name: policy-test
  labels:
    pod-security.kubernetes.io/enforce: restricted
```

Se utilizzate indipendentemente, queste modalità operative hanno risposte diverse che si traducono in esperienze utente diverse. La modalità enforce impedirà la creazione di pod se le rispettive PodSpecs violano il livello di restrizione configurato. Tuttavia, in questa modalità, non verrà impedito l'applicazione al cluster di oggetti Kubernetes diversi dai pod che creano pod, come Deployments, anche se il PodSpec viola il PSS applicato. In questo caso verrà applicata la distribuzione, mentre verrà impedita l'applicazione dei pod.

Si tratta di un'esperienza utente difficile, in quanto non vi è alcuna indicazione immediata che l'oggetto Deployment applicato correttamente smentisca la mancata creazione del

pod. Il PodSpecs offensivo non creerà pod. L'ispezione della risorsa Deployment con `kubectl get deploy <DEPLOYMENT_NAME> -oyaml` mostrerà il messaggio proveniente dall'`.status.conditions`elemento (i) pod (i) fallito, come illustrato di seguito.

```
...
status:
  conditions:
    - lastTransitionTime: "2022-01-20T01:02:08Z"
      lastUpdateTime: "2022-01-20T01:02:08Z"
      message: 'pods "test-688f68dc87-tw587" is forbidden: violates PodSecurity
 "restricted:latest":
       allowPrivilegeEscalation != false (container "test" must set
 securityContext.allowPrivilegeEscalation=false),
       unrestricted capabilities (container "test" must set
 securityContext.capabilities.drop=["ALL"]),
       runAsNonRoot != true (pod or container "test" must set
 securityContext.runAsNonRoot=true),
       seccompProfile (pod or container "test" must set
 securityContext.seccompProfile.type
       to "RuntimeDefault" or "Localhost")'
      reason: FailedCreate
      status: "True"
      type: ReplicaFailure
...
```

In entrambe le modalità di controllo e di avviso, le restrizioni relative ai pod non impediscono la creazione e l'avvio di pod che violano le norme. Tuttavia, in queste modalità, le annotazioni di controllo sugli eventi del registro di controllo del server API e gli avvisi ai client del server API, come kubectl, vengono attivati, rispettivamente, quando i pod, così come gli oggetti che creano i pod, contengono PodSpecs con violazioni. `kubectl`Di seguito viene visualizzato un messaggio di avviso.

```
Warning: would violate PodSecurity "restricted:latest":
 allowPrivilegeEscalation != false (container "test" must set
 securityContext.allowPrivilegeEscalation=false), unrestricted capabilities (container
 "test" must set securityContext.capabilities.drop=["ALL"]), runAsNonRoot != true (pod
 or container "test" must set securityContext.runAsNonRoot=true), seccompProfile (pod
 or container "test" must set securityContext.seccompProfile.type to "RuntimeDefault"
 or "Localhost")
deployment.apps/test created
```

Le modalità di controllo e avviso PSA sono utili per introdurre il PSS senza influire negativamente sulle operazioni del cluster.

Le modalità operative PSA non si escludono a vicenda e possono essere utilizzate in modo cumulativo. Come illustrato di seguito, le modalità multiple possono essere configurate in un unico namespace.

```
apiVersion: v1
kind: Namespace
metadata:
  name: policy-test
  labels:
    pod-security.kubernetes.io/audit: restricted
    pod-security.kubernetes.io/enforce: restricted
    pod-security.kubernetes.io/warn: restricted
```

Nell'esempio precedente, gli avvisi e le annotazioni di controllo di facile utilizzo vengono forniti quando si applicano Deployments, mentre l'applicazione delle violazioni viene fornita anche a livello di pod. In effetti, più etichette PSA possono utilizzare diversi livelli di profilo, come illustrato di seguito.

```
apiVersion: v1
kind: Namespace
metadata:
  name: policy-test
  labels:
    pod-security.kubernetes.io/enforce: baseline
    pod-security.kubernetes.io/warn: restricted
```

Nell'esempio precedente, PSA è configurato per consentire la creazione di tutti i pod che soddisfano il livello di profilo di base e quindi avvisa i pod (e gli oggetti che creano pod) che violano il livello di profilo limitato. Si tratta di un approccio utile per determinare i possibili impatti quando si passa dai profili di base a quelli con restrizioni.

Pod esistenti

Se uno spazio dei nomi con pod esistenti viene modificato per utilizzare un profilo PSS più restrittivo, le modalità audit e warn produrranno messaggi appropriati; tuttavia, la modalità enforce non eliminerà i pod. I messaggi di avviso sono riportati di seguito.

```
Warning: existing pods in namespace "policy-test" violate the new PodSecurity enforce
 level "restricted:latest"
```

```
Warning: test-688f68dc87-htm8x: allowPrivilegeEscalation != false, unrestricted
 capabilities, runAsNonRoot != true, seccompProfile
namespace/policy-test configured
```

Esenzioni

PSA utilizza le esenzioni per escludere l'applicazione di violazioni ai pod che altrimenti sarebbero state applicate. Queste esenzioni sono elencate di seguito.

- Nomi utente: le richieste degli utenti con un nome utente autenticato (o impersonato) esente vengono ignorate.
- RuntimeClassNames: i pod e le risorse del carico di lavoro che specificano un nome di classe runtime esente vengono ignorati.
- Namespace: i pod e le risorse del carico di lavoro in uno spazio dei nomi esente vengono ignorati.

Queste esenzioni vengono applicate staticamente nella configurazione del controller di ammissione PSA come parte della configurazione del server API.

Nell'implementazione Validating Webhook, le esenzioni possono essere configurate all'interno di una ConfigMaprisorsa Kubernetes che viene montata come volume nel contenitore. pod-security-webhook

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: pod-security-webhook
  namespace: pod-security-webhook
data:
  podsecurityconfiguration.yaml: |
    apiVersion: pod-security.admission.config.k8s.io/v1
    kind: PodSecurityConfiguration
    defaults:
      enforce: "restricted"
      enforce-version: "latest"
      audit: "restricted"
      audit-version: "latest"
      warn: "restricted"
      warn-version: "latest"
    exemptions:
      # Array of authenticated usernames to exempt.
      usernames: []
```

```
      # Array of runtime class names to exempt.
      runtimeClasses: []
      # Array of namespaces to exempt.
      namespaces: ["kube-system","policy-test1"]
```

Come illustrato nello ConfigMap YAML precedente, il livello PSS predefinito a livello di cluster è stato impostato su limitato per tutte le modalità PSA, audit, enforce e warn. Ciò influisce su tutti i namespace, ad eccezione di quelli esentati:. `namespaces: ["kube-system","policy-test1"]` Inoltre, nella ValidatingWebhookConfigurationrisorsa, vista di seguito, anche lo spazio dei pod-security-webhooknomi è esente dal PSS configurato.

```
...
webhooks:
  # Audit annotations will be prefixed with this name
  - name: "pod-security-webhook.kubernetes.io"
    # Fail-closed admission webhooks can present operational challenges.
    # You may want to consider using a failure policy of Ignore, but should
    # consider the security tradeoffs.
    failurePolicy: Fail
    namespaceSelector:
      # Exempt the webhook itself to avoid a circular dependency.
      matchExpressions:
        - key: kubernetes.io/metadata.name
          operator: NotIn
          values: ["pod-security-webhook"]
...
```

> **Important**
>
> Pod Security Admissions è diventato stabile in Kubernetes v1.25. Se volevi utilizzare la funzionalità Pod Security Admission prima che fosse abilitata per impostazione predefinita, dovevi installare il controller di ammissione dinamico (webhook mutante). Le istruzioni per l'installazione e la configurazione del webhook sono disponibili qui.

Scelta tra policy-as-code e Pod Security Standards

I Pod Security Standards (PSS) sono stati sviluppati per sostituire la Pod Security Policy (PSP), fornendo una soluzione integrata in Kubernetes e che non richiedeva soluzioni dell'ecosistema Kubernetes. Detto questo, le soluzioni policy-as-code (PAC) sono notevolmente più flessibili.

Il seguente elenco di pro e contro è stato progettato per aiutarti a prendere una decisione più informata sulla tua soluzione di sicurezza per i pod.

Policy-as-code (rispetto agli standard di sicurezza dei Pod)

Pro:

- Più flessibile e più granulare (fino agli attributi delle risorse, se necessario)
- Non si concentra solo sui pod, ma può essere utilizzato per diverse risorse e azioni
- Non solo applicato a livello di namespace
- Più maturo degli standard di sicurezza Pod
- Le decisioni possono basarsi su qualsiasi elemento presente nel payload della richiesta del server API, nonché sulle risorse del cluster esistenti e sui dati esterni (a seconda della soluzione)
- Supporta la modifica delle richieste del server API prima della convalida (dipende dalla soluzione)
- Può generare policy complementari e risorse Kubernetes (a seconda della soluzione). Dalle policy dei pod, Kyverno può generare automaticamente policy per controller di livello superiore, come Deployments. Kyverno può anche generare risorse Kubernetes aggiuntive «`quando viene creata una nuova risorsa o quando la fonte viene aggiornata`» utilizzando Generate Rules.)
- Può essere utilizzato per spostarsi a sinistra, nelle pipeline CICD, prima di effettuare chiamate al server API Kubernetes (dipende dalla soluzione)
- Può essere utilizzato per implementare comportamenti che non sono necessariamente correlati alla sicurezza, come le migliori pratiche, gli standard organizzativi, ecc.
- Può essere utilizzato in casi d'uso non Kubernetes (dipende dalla soluzione)
- Grazie alla flessibilità, l'esperienza utente può essere adattata alle esigenze degli utenti

Contro:

- Non integrato in Kubernetes
- Più complesso da imparare, configurare e supportare
- La redazione delle politiche potrebbe richiedere nuove skills/languages/capabilities

Pod Security Admission (rispetto a policy-as-code)

Pro:

- Integrato in Kubernetes

- Più semplice da configurare
- Nessuna nuova lingua da usare o regole da creare
- Se il livello di ammissione predefinito del cluster è configurato su privilegiato, è possibile utilizzare le etichette dei namespace per inserire i namespace nei profili di sicurezza dei pod.

Contro:

- Non così flessibile o granulare come policy-as-code
- Solo 3 livelli di restrizioni
- Concentrato principalmente sui pod

Riepilogo

Se al momento non disponete di una soluzione di sicurezza per i pod, oltre a PSP, e la configurazione di sicurezza dei pod richiesta corrisponde al modello definito nei Pod Security Standards (PSS), allora una soluzione più semplice potrebbe essere quella di adottare il PSS al posto di una soluzione. policy-as-code Tuttavia, se la posizione di sicurezza dei tuoi pod non si adatta al modello PSS o prevedi di aggiungere controlli aggiuntivi, oltre a quelli definiti da PSS, una policy-as-code soluzione sembrerebbe più adatta.

Raccomandazioni

Utilizza più modalità Pod Security Admission (PSA) per una migliore esperienza utente

Come accennato in precedenza, la modalità PSA enforce impedisce l'applicazione di pod con violazioni PSS, ma non blocca i controller di livello superiore, come Deployments. In effetti, il Deployment verrà applicato correttamente senza alcuna indicazione che i pod non siano stati applicati. Sebbene sia possibile utilizzare kubectl per ispezionare l'oggetto Deployment e scoprire il messaggio relativo ai pod non riusciti inviato dal PSA, l'esperienza utente potrebbe essere migliore. Per migliorare l'esperienza dell'utente, è necessario utilizzare più modalità PSA (audit, enforce, warn).

```
apiVersion: v1
kind: Namespace
metadata:
  name: policy-test
  labels:
    pod-security.kubernetes.io/audit: restricted
```

```
    pod-security.kubernetes.io/enforce: restricted
    pod-security.kubernetes.io/warn: restricted
```

Nell'esempio precedente, con la modalità enforce definita, quando si tenta di applicare un manifesto di distribuzione con violazioni PSS nel rispettivo PodSpec al server dell'API Kubernetes, il Deployment verrà applicato correttamente, ma i pod no. Inoltre, poiché sono abilitate anche le modalità di controllo e avviso, il client del server API riceverà un messaggio di avviso e anche l'evento del registro di controllo del server API verrà annotato con un messaggio.

Limita i contenitori che possono essere eseguiti come privilegiati

Come accennato, i contenitori che funzionano con privilegi ereditano tutte le funzionalità Linux assegnate a root sull'host. Raramente i contenitori necessitano di questo tipo di privilegi per funzionare correttamente. Esistono diversi metodi che possono essere utilizzati per limitare le autorizzazioni e le funzionalità dei contenitori.

> **Important**
>
> Fargate è un tipo di lancio che consente di eseguire container «serverless» in cui i contenitori di un pod vengono eseguiti sull'infrastruttura gestita da AWS. Con Fargate, non è possibile eseguire un contenitore privilegiato o configurare il pod per utilizzare HostNetwork o HostPort.

Non eseguite processi nei contenitori come root

Per impostazione predefinita, tutti i contenitori vengono eseguiti come root. Ciò potrebbe essere problematico se un utente malintenzionato è in grado di sfruttare una vulnerabilità dell'applicazione e ottenere l'accesso tramite shell al contenitore in esecuzione. È possibile mitigare questo rischio in diversi modi. Innanzitutto, rimuovendo la shell dall'immagine del contenitore. In secondo luogo, aggiungendo la direttiva USER al Dockerfile o eseguendo i contenitori nel pod come utente non root. Kubernetes PodSpec include una serie di campi, in`spec.securityContext`, che consentono di specificare il gruppo di utenti and/or in base al quale eseguire l'applicazione. Questi campi sono e rispettivamente. `runAsUser` `runAsGroup`

Per imporre l'uso di`spec.securityContext`, e dei relativi elementi associati, all'interno di Kubernetes PodSpec, è possibile aggiungere ai cluster gli standard di sicurezza policy-as-code Pod. Queste soluzioni consentono di scrivere e/o utilizzare policy o profili in grado di convalidare i payload delle richieste del server API Kubernetes in entrata, prima che vengano mantenuti in etcd. Inoltre,

policy-as-code le soluzioni possono modificare le richieste in entrata e, in alcuni casi, generare nuove richieste.

Non eseguire mai Docker in Docker o montare il socket nel contenitore

Sebbene ciò consenta di accedere comodamente alle build/run immagini nei contenitori Docker, in pratica stai cedendo il controllo completo del nodo al processo in esecuzione nel contenitore. Se devi creare immagini di container su Kubernetes, usa invece Kaniko, buildah o un servizio di compilazione simile. CodeBuild

> **Note**
>
> I cluster Kubernetes utilizzati per l'elaborazione CICD, come la creazione di immagini di container, devono essere isolati dai cluster che eseguono carichi di lavoro più generalizzati.

Limita l'uso di HostPath o, se necessario, limita i prefissi che possono essere utilizzati e configura il volume come di sola lettura

`hostPath`è un volume che monta una directory dall'host direttamente al contenitore. Raramente i pod necessitano di questo tipo di accesso, ma in tal caso è necessario essere consapevoli dei rischi. Per impostazione predefinita, i pod eseguiti come root avranno accesso in scrittura al file system esposto da HostPath. Ciò potrebbe consentire a un utente malintenzionato di modificare le impostazioni di kubelet, creare collegamenti simbolici a directory o file non esposti direttamente da HostPath, ad esempio /etc/shadow, installare chiavi ssh, leggere segreti montati sull'host e altre cose dannose. Per mitigare i rischi di HostPath, configura l'as, ad esempio: `spec.containers.volumeMounts` `readOnly`

```
volumeMounts:
- name: hostPath-volume
    readOnly: true
    mountPath: /host-path
```

È inoltre necessario utilizzare policy-as-code soluzioni per limitare le directory che possono essere utilizzate dai `hostPath` volumi o `hostPath` impedirne del tutto l'utilizzo. È possibile utilizzare le politiche Pod Security Standards Baseline o Restricted per impedire l'uso di. `hostPath`

Per ulteriori informazioni sui pericoli dell'escalation privilegiata, leggi il blog di Seth Art Bad Pods: Kubernetes Pod Privilege Escalation.

Imposta richieste e limiti per ogni contenitore per evitare contese di risorse e attacchi DoS

Un pod senza richieste o limiti può teoricamente consumare tutte le risorse disponibili su un host. Poiché su un nodo sono programmati pod aggiuntivi, il nodo potrebbe subire una pressione della CPU o della memoria che può causare la chiusura di Kubelet o l'eliminazione dei pod dal nodo. Sebbene non sia possibile evitare che ciò accada del tutto, l'impostazione di richieste e limiti aiuterà a ridurre al minimo il conflitto di risorse e a mitigare il rischio di applicazioni scritte male che consumano una quantità eccessiva di risorse.

`podSpec`Consente di specificare richieste e limiti per CPU e memoria. La CPU è considerata una risorsa comprimibile perché può essere sottoposta a sottoscrizioni eccessive. La memoria è incomprimibile, cioè non può essere condivisa tra più contenitori.

Quando si specificano richieste di CPU o memoria, si indica essenzialmente la quantità di memoria che i contenitori hanno la garanzia di ottenere. Kubernetes aggrega le richieste di tutti i contenitori in un pod per determinare su quale nodo programmare il pod. Se un contenitore supera la quantità di memoria richiesta, può essere soggetto a chiusura in caso di pressione della memoria sul nodo.

I limiti sono la quantità massima di risorse di CPU e memoria che un contenitore può consumare e corrispondono direttamente al `memory.limit_in_bytes` valore del cgroup creato per il contenitore. Un contenitore che supera il limite di memoria verrà eliminato da OOM. Se un contenitore supera il limite di CPU, verrà limitato.

> **Note**
>
> Quando si utilizza il contenitore`resources.limits`, si consiglia vivamente che l'utilizzo delle risorse del contenitore (noto anche come Resource Footprints) sia basato sui dati e accurato, in base ai test di carico. In assenza di un ingombro preciso e affidabile delle risorse, il contenitore può essere imbottito. `resources.limits` Ad esempio, `resources.limits.memory` potrebbe essere aggiunto un riempimento superiore del 20-30% rispetto ai valori massimi osservabili, per tenere conto di potenziali imprecisioni nei limiti delle risorse di memoria.

Kubernetes utilizza tre classi Quality of Service (QoS) per dare priorità ai carichi di lavoro in esecuzione su un nodo. Ciò include:

- garantito

- scoppiabile
- miglior sforzo

Se i limiti e le richieste non sono impostati, il pod viene configurato come best-effort (priorità più bassa). I pod Best-Effort sono i primi a morire quando la memoria è insufficiente. Se i limiti sono impostati su tutti i contenitori all'interno del pod o se le richieste e i limiti sono impostati sugli stessi valori e non sono uguali a 0, il pod viene configurato come garantito (priorità massima). I pod garantiti non verranno eliminati a meno che non superino i limiti di memoria configurati. Se i limiti e le richieste sono configurati con valori diversi e non uguali a 0, o se un contenitore all'interno del pod impone dei limiti e gli altri no o hanno limiti impostati per risorse diverse, i pod vengono configurati come burstable (priorità media). Questi pod offrono alcune garanzie in termini di risorse, ma possono essere eliminati quando superano la memoria richiesta.

> **Important**
>
> Le richieste non influiscono sul `memory_limit_in_bytes` valore del cgroup del contenitore; il limite di cgroup è impostato sulla quantità di memoria disponibile sull'host. Tuttavia, se il valore delle richieste viene impostato su un valore troppo basso, il pod potrebbe essere interrotto dal kubelet se il nodo è sottoposto a una pressione di memoria.

Classe	Priorità	Condizione	Condizione di uccisione
Garantita	highest	limite = richiesta! = 0	Supera solo i limiti di memoria
Scoppibile	medium	limite! = richiesta! = 0	Può essere ucciso se supera la memoria richiesta
Miglior sforzo	lowest	limite e richiesta non impostati	Il primo a farsi uccidere quando la memoria è insuffici ente

Per ulteriori informazioni sul QoS delle risorse, consulta la documentazione di Kubernetes.

Puoi forzare l'uso di richieste e limiti impostando una quota di risorse su un namespace o creando un intervallo di limiti. Una quota di risorse consente di specificare la quantità totale di risorse, ad esempio CPU e RAM, allocate a un namespace. Quando viene applicata a un namespace, ti obbliga a specificare richieste e limiti per tutti i contenitori distribuiti in quel namespace. Al contrario, gli intervalli di limiti offrono un controllo più granulare dell'allocazione delle risorse. Con gli intervalli di limiti è possibile impostare min/max le risorse di CPU e memoria per pod o per contenitore all'interno di un namespace. Puoi anche usarli per impostare valori di richiesta/limite predefiniti se non ne vengono forniti.

Policy-as-code è possibile utilizzare soluzioni per imporre richieste e limiti. o persino per creare quote di risorse e intervalli di limiti quando vengono creati i namespace.

Non consentite un'escalation privilegiata

L'escalation privilegiata consente a un processo di modificare il contesto di sicurezza in cui viene eseguito. Sudo ne è un buon esempio, così come i binari con bit SUID o SGID. L'escalation privilegiata è fondamentalmente un modo per gli utenti di eseguire un file con le autorizzazioni di un altro utente o gruppo. È possibile impedire a un contenitore di utilizzare l'escalation privilegiata implementando una policy policy-as-code mutante che imposta o impostando su. `allowPrivilegeEscalation` `false` `securityContext.allowPrivilegeEscalation` `podSpec` Policy-as-code i criteri possono essere utilizzati anche per impedire che le richieste del server API abbiano esito positivo se vengono rilevate impostazioni errate. Gli standard di sicurezza dei pod possono essere utilizzati anche per impedire ai pod di utilizzare l'escalation dei privilegi.

ServiceAccount Disabilita i supporti dei token

Per i pod che non devono accedere all'API Kubernetes, puoi disabilitare il montaggio automatico di un ServiceAccount token su una specifica del pod o per tutti i pod che ne utilizzano una particolare. ServiceAccount

Example

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-no-automount
spec:
```

```
  automountServiceAccountToken: false
```

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: sa-no-automount
automountServiceAccountToken: false
```

Disabilita il rilevamento dei servizi

Per i pod che non devono cercare o chiamare servizi interni al cluster, puoi ridurre la quantità di informazioni fornite a un pod. È possibile impostare la policy DNS del pod in modo che non utilizzi CoredNS e non esponga i servizi nello spazio dei nomi del pod come variabili di ambiente. Consulta i documenti di Kubernetes sulle variabili di ambiente per ulteriori informazioni sui collegamenti ai servizi. Il valore predefinito per la policy DNS di un pod è "ClusterFirst" che utilizza il DNS interno al cluster, mentre il valore non predefinito «Default» utilizza la risoluzione DNS del nodo sottostante. Per ulteriori informazioni, consulta i documenti di Kubernetes sulla politica DNS di Pod.

Example

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-no-service-info
spec:
    dnsPolicy: Default # "Default" is not the true default value
    enableServiceLinks: false
```

Configura le tue immagini con un file system root di sola lettura

La configurazione delle immagini con un file system root di sola lettura impedisce a un utente malintenzionato di sovrascrivere un file binario sul file system utilizzato dall'applicazione. Se l'applicazione deve scrivere sul file system, prendete in considerazione la possibilità di scrivere in una directory temporanea o di allegare e montare un volume. Puoi imporlo impostando i pod SecurityContext come segue:

```
...
securityContext:
  readOnlyRootFilesystem: true
...
```

Policy-as-code e gli standard di sicurezza dei Pod possono essere utilizzati per imporre questo comportamento.

Example

In base a Windows, i contenitori in Kubernetes `securityContext.readOnlyRootFilesystem` non possono essere impostati su quelli in esecuzione su Windows, poiché è necessario l'accesso in scrittura per l'esecuzione dei processi di registro e di sistema all'interno del contenitore. `true`

Strumenti e risorse

- Workshop di immersione nella sicurezza di Amazon EKS - Pod Security
- open-policy-agent/gatekeeper-library: La libreria di policy OPA Gatekeeper, una libreria di OPA/Gatekeeper policy che puoi usare come sostituto. PSPs
- Libreria delle politiche di Kyverno
- Una raccolta di politiche OPA e Kyverno comuni per EKS.
- Contromisure basate sulle politiche: parte 1
- Contromisure basate sulle politiche: parte 2
- Pod Security Policy Migrator è uno strumento che converte in politiche PSPs OPA/Gatekeeper o Kyverno KubeWarden
- NeuVector di SUSE, piattaforma open source per la sicurezza dei container zero-trust, fornisce politiche di processo e file system, nonché regole di controllo delle ammissioni.

Isolamento degli inquilini

Quando pensiamo alla multi-tenancy, spesso vogliamo isolare un utente o un'applicazione da altri utenti o applicazioni in esecuzione su un'infrastruttura condivisa.

Kubernetes è un orchestratore a tenant singolo, ovvero una singola istanza del piano di controllo è condivisa tra tutti i tenant all'interno di un cluster. Esistono, tuttavia, vari oggetti Kubernetes che puoi utilizzare per creare una parvenza di multi-tenancy. Ad esempio, i namespace e i controlli di accesso basati sui ruoli (RBAC) possono essere implementati per isolare logicamente i tenant gli uni dagli altri. Allo stesso modo, le quote e gli intervalli di limiti possono essere utilizzati per controllare la quantità di risorse del cluster che ogni tenant può consumare. Tuttavia, il cluster è l'unico costrutto che fornisce un solido limite di sicurezza. Questo perché un utente malintenzionato che riesce ad accedere a un host all'interno del cluster può recuperare tutti i segreti e i volumi montati su quell'host. ConfigMaps

Potrebbero anche impersonare il Kubelet, il che consentirebbe loro di manipolare gli attributi del nodo che si spostano lateralmente all'interno del cluster. and/or

Le sezioni seguenti spiegheranno come implementare l'isolamento dei tenant mitigando al contempo i rischi dell'utilizzo di un orchestratore a tenant singolo come Kubernetes.

Soft multi-tenancy

Con la multi-tenancy software, si utilizzano costrutti Kubernetes nativi, ad esempio namespace, ruoli e associazioni di ruoli e politiche di rete, per creare una separazione logica tra i tenant. RBAC, ad esempio, può impedire ai tenant di accedere o manipolare le risorse reciproche. Le quote e gli intervalli limite controllano la quantità di risorse del cluster che ogni tenant può consumare, mentre le politiche di rete possono aiutare a impedire che le applicazioni distribuite in namespace diversi comunichino tra loro.

Nessuno di questi controlli, tuttavia, impedisce ai pod di tenant diversi di condividere un nodo. Se è necessario un isolamento più forte, è possibile utilizzare un selettore di nodi, regole di antiaffinità, and/or contaminazioni e tolleranze per forzare la programmazione dei pod di tenant diversi su nodi separati, spesso denominati nodi tenant unici. In un ambiente con molti tenant, questa operazione potrebbe diventare piuttosto complicata e a costi proibitivi.

> **Important**
>
> La multi-tenancy soft implementata con Namespace non consente di fornire ai tenant un elenco filtrato di namespace perché i namespace sono un tipo con ambito globale. Se un tenant ha la possibilità di visualizzare un particolare namespace, può visualizzare tutti i namespace all'interno del cluster.

> **Warning**
>
> Con soft-multi-tenancy, i tenant mantengono la possibilità di interrogare CoredNS per tutti i servizi eseguiti all'interno del cluster per impostazione predefinita. Un utente malintenzionato potrebbe sfruttarlo eseguendo dig SRV `..svc.cluster.local` da qualsiasi pod del cluster. Se devi limitare l'accesso ai record DNS dei servizi eseguiti all'interno dei tuoi cluster, prendi in considerazione l'utilizzo dei plugin Firewall o Policy per CoredNS. Per ulteriori informazioni, vedere policy# -policy. https://github.com/coredns/ kubernetes-metadata-multi-tenancy

Kiosk è un progetto open source che può aiutare nell'implementazione del soft multi-tenancy. È implementato come una serie di controller che CRDs forniscono le seguenti funzionalità:

- Account e utenti di account per separare i tenant in un cluster Kubernetes condiviso
- Self-Service Namespace Provisioning per gli utenti dell'account
- Limiti dell'account per garantire la qualità del servizio e l'equità nella condivisione di un cluster
- Modelli di namespace per l'isolamento sicuro dei tenant e l'inizializzazione self-service dello spazio dei nomi

Loft è un'offerta commerciale dei gestori di Kiosk che aggiunge le seguenti funzionalità: DevSpace

- Accesso multicluster per concedere l'accesso agli spazi in diversi cluster
- La modalità Sleep riduce le implementazioni in uno spazio durante i periodi di inattività
- Single Sign-On con provider di autenticazione OIDC come GitHub

Esistono tre casi d'uso principali che possono essere risolti con la multi-tenancy software.

Impostazione aziendale

La prima è in un ambiente aziendale in cui gli «inquilini» sono parzialmente affidabili in quanto sono dipendenti, appaltatori o sono altrimenti autorizzati dall'organizzazione. In genere, ogni tenant si allinea a una divisione amministrativa, ad esempio un reparto o un team.

In questo tipo di impostazione, un amministratore del cluster è in genere responsabile della creazione di namespace e della gestione delle politiche. Possono anche implementare un modello di amministrazione delegata in cui a determinate persone viene affidata la supervisione di un namespace, che consente loro di eseguire operazioni CRUD per oggetti non correlati alle politiche come distribuzioni, servizi, pod, lavori, ecc.

L'isolamento fornito dal runtime di un container può essere accettabile all'interno di questa impostazione o potrebbe essere necessario aumentarlo con controlli aggiuntivi per la sicurezza dei pod. Potrebbe anche essere necessario limitare la comunicazione tra servizi in namespace diversi se è richiesto un isolamento più rigoroso.

Kubernetes come servizio

Al contrario, la multi-tenancy soft può essere utilizzata nelle impostazioni in cui si desidera offrire Kubernetes as a service (KaaS). Con KaaS, l'applicazione è ospitata in un cluster condiviso insieme

a una raccolta di controller che forniscono una serie di servizi PaaS. CRDs I tenant interagiscono direttamente con il server API Kubernetes e sono autorizzati a eseguire operazioni CRUD su oggetti non conformi alle policy. Esiste anche un elemento di self-service in quanto i tenant possono essere autorizzati a creare e gestire i propri namespace. In questo tipo di ambiente, si presume che i tenant eseguano codice non attendibile.

Per isolare i tenant in questo tipo di ambiente, è probabile che sia necessario implementare politiche di rete rigorose e il sandboxing dei pod. Il sandboxing consiste nell'eseguire i contenitori di un pod all'interno di una micro VM come Firecracker o in un kernel dello spazio utente. Oggi puoi creare pod in modalità sandbox con EKS Fargate.

Software as a Service (SaaS)

L'ultimo caso d'uso per la multi-tenancy software è in un ambiente (Software-as-a-ServiceSaaS). In questo ambiente, ogni tenant è associato a una particolare istanza di un'applicazione in esecuzione all'interno del cluster. Ogni istanza ha spesso i propri dati e utilizza controlli di accesso separati che di solito sono indipendenti da Kubernetes RBAC.

A differenza degli altri casi d'uso, il tenant in un'impostazione SaaS non si interfaccia direttamente con l'API Kubernetes. L'applicazione SaaS è invece responsabile dell'interfacciamento con l'API Kubernetes per creare gli oggetti necessari a supportare ciascun tenant.

Kubernetes Constructs

In ognuno di questi casi vengono utilizzati i seguenti costrutti per isolare gli inquilini gli uni dagli altri:

Spazi dei nomi

I namespace sono fondamentali per l'implementazione della multi-tenancy soft. Consentono di dividere il cluster in partizioni logiche. Le quote, le politiche di rete, gli account di servizio e altri oggetti necessari per implementare la multi-tenancy rientrano nell'ambito di un namespace.

Policy di rete

Per impostazione predefinita, tutti i pod in un cluster Kubernetes possono comunicare tra loro. Questo comportamento può essere modificato utilizzando le policy di rete.

Le politiche di rete limitano la comunicazione tra i pod utilizzando etichette o intervalli di indirizzi IP. In un ambiente multi-tenant in cui è richiesto un rigoroso isolamento della rete tra i tenant, consigliamo di iniziare con una regola predefinita che neghi la comunicazione tra i pod e un'altra regola che

consenta a tutti i pod di interrogare il server DNS per la risoluzione dei nomi. A questo punto, puoi iniziare ad aggiungere altre regole permissive che consentano la comunicazione all'interno di un namespace. Questo può essere ulteriormente perfezionato se necessario.

> **Note**
>
> Amazon VPC CNI ora supporta le politiche di rete Kubernetes per creare policy in grado di isolare i carichi di lavoro sensibili e proteggerli da accessi non autorizzati durante l'esecuzione di Kubernetes su AWS. Ciò significa che puoi utilizzare tutte le funzionalità dell'API Network Policy all'interno del tuo cluster Amazon EKS. Questo livello di controllo granulare consente di implementare il principio del privilegio minimo, che garantisce che solo i pod autorizzati possano comunicare tra loro.

> **Important**
>
> Le politiche di rete sono necessarie ma non sufficienti. L'applicazione delle politiche di rete richiede un motore di policy come Calico o Cilium.

Controllo degli accessi basato sui ruoli (RBAC)

I ruoli e le associazioni di ruolo sono gli oggetti Kubernetes utilizzati per imporre il controllo degli accessi basato sui ruoli (RBAC) in Kubernetes. I ruoli contengono elenchi di azioni che possono essere eseguite sugli oggetti del cluster. Le associazioni di ruolo specificano gli individui o i gruppi a cui si applicano i ruoli. Nelle impostazioni aziendali e KaaS, RBAC può essere utilizzato per consentire l'amministrazione di oggetti da parte di gruppi o individui selezionati.

Quote

Le quote vengono utilizzate per definire i limiti dei carichi di lavoro ospitati nel cluster. Con le quote, puoi specificare la quantità massima di CPU e memoria che un pod può consumare oppure puoi limitare il numero di risorse che possono essere allocate in un cluster o in un namespace. Gli intervalli di limiti consentono di dichiarare valori minimi, massimi e predefiniti per ogni limite.

L'impegno eccessivo delle risorse in un cluster condiviso è spesso utile perché consente di massimizzare le risorse. Tuttavia, l'accesso illimitato a un cluster può causare una carenza di risorse, con conseguente peggioramento delle prestazioni e perdita di disponibilità delle applicazioni. Se le richieste di un pod sono impostate su un valore troppo basso e l'utilizzo effettivo delle risorse

supera la capacità del nodo, il nodo inizierà a subire pressioni sulla CPU o sulla memoria. Quando ciò accade, i pod possono essere riavviati and/or e rimossi dal nodo.

Per evitare che ciò accada, dovreste pianificare di imporre quote ai namespace in un ambiente multi-tenant per obbligare i tenant a specificare richieste e limiti durante la pianificazione dei loro pod sul cluster. Inoltre, attenuerà una potenziale interruzione del servizio limitando la quantità di risorse che un pod può consumare.

È inoltre possibile utilizzare le quote per ripartire le risorse del cluster in base alla spesa del tenant. Ciò è particolarmente utile nello scenario KaaS.

Priorità e priorità del pod

La priorità e la priorità dei Pod possono essere utili quando si desidera dare maggiore importanza a un Pod rispetto agli altri Pod. Ad esempio, con la priorità dei pod è possibile configurare i pod del cliente A in modo che funzionino con una priorità più alta rispetto al cliente B. Quando la capacità disponibile è insufficiente, lo scheduler rimuoverà i pod con priorità più bassa dal cliente B per ospitare i pod con priorità più alta del cliente A. Ciò può essere particolarmente utile in un ambiente SaaS in cui i clienti disposti a pagare un premio ricevono una priorità più elevata.

> **Important**
>
> La priorità dei Pod può avere un effetto indesiderato su altri Pod con priorità inferiore. Ad esempio, anche se i pod delle vittime vengono eliminati senza problemi, ma ciò non PodDisruptionBudget è garantito, ciò potrebbe compromettere un'applicazione con priorità inferiore che si basa su un quorum di Pod, vedi Limitazioni della prelazione.

Controlli attenuanti

La tua principale preoccupazione come amministratore di un ambiente multi-tenant è impedire a un utente malintenzionato di accedere all'host sottostante. È necessario prendere in considerazione i seguenti controlli per mitigare questo rischio:

Ambienti di esecuzione in modalità sandbox per contenitori

Il sandboxing è una tecnica mediante la quale ogni contenitore viene eseguito nella propria macchina virtuale isolata. Le tecnologie che eseguono il sandboxing tramite pod includono Firecracker e Firekube di Weave.

Per ulteriori informazioni sul tentativo di rendere Firecracker un runtime supportato per EKS, consulta 1238496944684597248.html. https://threadreaderapp.com/thread/

Open Policy Agent (OPA) e Gatekeeper

Gatekeeper è un controller di ammissione di Kubernetes che applica le politiche create con OPA. Con OPA puoi creare una policy che esegua i pod degli inquilini su istanze separate o con una priorità più alta rispetto agli altri inquilini. Una raccolta di politiche OPA comuni è disponibile nell'archivio di questo progetto. GitHub

Esiste anche un plug-in OPA sperimentale per CoreDNS che consente di utilizzare filter/control OPA per i record restituiti da CoredNS.

Kyverno

Kyverno è un motore di policy nativo di Kubernetes in grado di convalidare, modificare e generare configurazioni con policy come risorse Kubernetes. Kyverno utilizza overlay in stile Kustomize per la convalida, supporta JSON Patch e la patch di unione strategica per la mutazione e può clonare risorse su namespace sulla base di trigger flessibili.

Puoi usare Kyverno per isolare i namespace, applicare la sicurezza dei pod e altre best practice e generare configurazioni predefinite come le politiche di rete. Nel repository di questo progetto sono inclusi diversi esempi. GitHub Molti altri sono inclusi nella raccolta delle politiche sul sito web di Kyverno.

Isolamento dei carichi di lavoro dei tenant su nodi specifici

È possibile utilizzare la limitazione dei carichi di lavoro dei tenant in modo che vengano eseguiti su nodi specifici per aumentare l'isolamento nel modello soft multi-tenancy. Con questo approccio, i carichi di lavoro specifici dei tenant vengono eseguiti solo su nodi forniti per i rispettivi tenant. Per ottenere questo isolamento, vengono utilizzate le proprietà native di Kubernetes (affinità dei nodi, fattori e tolleranze) per indirizzare nodi specifici per la pianificazione dei pod e impedire che i pod di altri tenant vengano pianificati sui nodi specifici del tenant.

Parte 1 - Affinità dei nodi

L'affinità dei nodi Kubernetes viene utilizzata per indirizzare i nodi per la pianificazione, in base alle etichette dei nodi. Con le regole di affinità dei nodi, i pod vengono attratti da nodi specifici che corrispondono ai termini del selettore. Nella specifica del pod seguente, l'affinità dei `requiredDuringSchedulingIgnoredDuringExecution` nodi viene applicata al rispettivo

pod. Il risultato è che il pod prenderà di mira i nodi etichettati con la seguente chiave/valore:. `node-restriction.kubernetes.io/tenant: tenants-x`

```
...
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: node-restriction.kubernetes.io/tenant
            operator: In
            values:
            - tenants-x
...
```

Con questa affinità tra i nodi, l'etichetta è necessaria durante la pianificazione, ma non durante l'esecuzione; se le etichette dei nodi sottostanti cambiano, i pod non verranno rimossi esclusivamente a causa di quella modifica dell'etichetta. Tuttavia, la pianificazione futura potrebbe risentirne.

> **Warning**
>
> Il prefisso dell'etichetta `node-restriction.kubernetes.io/` ha un significato speciale in Kubernetes. NodeRestrictionche è abilitato per i cluster EKS `kubelet` impedisce adding/removing/updating di etichettare con questo prefisso. Gli aggressori non sono in grado di utilizzare `kubelet's credentials to update the node object or modify the system setup to pass these labels into `kubelet` poiché `kubelet` non è consentito modificare queste etichette. Se questo prefisso viene utilizzato per la pianificazione di tutta la pianificazione da pod a nodo, previene gli scenari in cui un utente malintenzionato potrebbe voler attirare un diverso set di carichi di lavoro su un nodo modificando le etichette dei nodi.

Example

Invece dell'affinità dei nodi, avremmo potuto usare il selettore di nodi. Tuttavia, l'affinità dei nodi è più espressiva e consente di considerare più condizioni durante la pianificazione dei pod. Per ulteriori informazioni sulle differenze e sulle scelte di pianificazione più avanzate, consulta questo post del blog CNCF sulla pianificazione avanzata da pod a nodo di Kubernetes.

Parte 2 - Macchie e tolleranze

Attirare i pod verso i nodi è solo la prima parte di questo approccio in tre parti. Affinché questo approccio funzioni, dobbiamo impedire ai pod di programmarsi su nodi per i quali i pod non sono autorizzati. Per respingere i pod indesiderati o non autorizzati, Kubernetes utilizza i nodi taints. I taint vengono utilizzati per creare condizioni sui nodi che impediscono la pianificazione dei pod. Il colore seguente utilizza una coppia chiave-valore di. `tenant: tenants-x`

```
...
    taints:
      - key: tenant
        value: tenants-x
        effect: NoSchedule
...
```

Dato il nodo precedente`taint`, solo i pod che tollerano la contaminazione potranno essere programmati sul nodo. Per consentire la programmazione dei pod autorizzati sul nodo, le specifiche dei rispettivi pod devono includere una `toleration` a to the taint, come illustrato di seguito.

```
...
  tolerations:
  - effect: NoSchedule
    key: tenant
    operator: Equal
    value: tenants-x
...
```

Ai pod con quanto sopra non `toleration` verrà impedito di programmare sul nodo, almeno non a causa di quella specifica contaminazione. I taint vengono utilizzati anche da Kubernetes per interrompere temporaneamente la pianificazione dei pod in determinate condizioni, come la pressione delle risorse del nodo. Grazie all'affinità tra i nodi, alle contaminazioni e alle tolleranze, possiamo attirare efficacemente i pod desiderati verso nodi specifici e respingere i pod indesiderati.

> **Important**
>
> Alcuni pod Kubernetes sono necessari per funzionare su tutti i nodi. Esempi di questi pod sono quelli avviati dalla Container Network Interface (CNI) e dai daemonset kube-proxy.
> A tal fine, le specifiche di questi pod contengono tolleranze molto permissive, per tollerare diverse macchie. Si deve prestare attenzione a non modificare queste tolleranze. La modifica

di queste tolleranze potrebbe comportare un funzionamento errato del cluster. Inoltre, è possibile utilizzare strumenti di gestione delle politiche, come OPA/Gatekeeper e Kyverno per scrivere politiche di convalida che impediscano ai pod non autorizzati di utilizzare queste tolleranze permissive.

Parte 3 - Gestione basata su policy per la selezione dei nodi

Esistono diversi strumenti che possono essere utilizzati per aiutare a gestire l'affinità dei nodi e le tolleranze delle specifiche dei pod, inclusa l'applicazione delle regole nelle pipeline CICD. Tuttavia, l'applicazione dell'isolamento dovrebbe essere effettuata anche a livello di cluster Kubernetes. A tal fine, è possibile utilizzare strumenti di gestione delle politiche per modificare le richieste del server API Kubernetes in entrata, in base ai payload delle richieste, per applicare le rispettive regole di affinità dei nodi e le tolleranze sopra menzionate.

Ad esempio, i pod destinati allo spazio dei nomi tenants-x possono essere timbrati con l'affinità e la tolleranza dei nodi corrette per consentire la pianificazione sui nodi tenants-x. Utilizzando strumenti di gestione delle policy configurati utilizzando il Kubernetes Mutating Admission Webhook, è possibile utilizzare le policy per modificare le specifiche dei pod in entrata. Le mutazioni aggiungono gli elementi necessari per consentire la pianificazione desiderata. Di seguito è riportato un esempio di OPA/Gatekeeper politica che aggiunge un'affinità tra i nodi.

```
apiVersion: mutations.gatekeeper.sh/v1alpha1
kind: Assign
metadata:
  name: mutator-add-nodeaffinity-pod
  annotations:
    aws-eks-best-practices/description: >-
      Adds Node affinity - https://kubernetes.io/docs/concepts/scheduling-eviction/
assign-pod-node/#node-affinity
spec:
  applyTo:
  - groups: [""]
    kinds: ["Pod"]
    versions: ["v1"]
  match:
    namespaces: ["tenants-x"]
  location:
 "spec.affinity.nodeAffinity.requiredDuringSchedulingIgnoredDuringExecution.nodeSelectorTerms"
  parameters:
    assign:
```

```
      value:
        - matchExpressions:
          - key: "tenant"
            operator: In
            values:
            - "tenants-x"
```

La politica di cui sopra viene applicata a una richiesta del server API Kubernetes, per applicare un pod allo spazio dei nomi tenants-x. La policy aggiunge la regola di affinità dei `requiredDuringSchedulingIgnoredDuringExecution` nodi, in modo che i pod vengano attratti dai nodi con l'etichetta. `tenant: tenants-x`

Una seconda politica, illustrata di seguito, aggiunge la tolleranza alla stessa specifica del pod, utilizzando gli stessi criteri di corrispondenza dello spazio dei nomi e dei gruppi, dei tipi e delle versioni di destinazione.

```
apiVersion: mutations.gatekeeper.sh/v1alpha1
kind: Assign
metadata:
  name: mutator-add-toleration-pod
  annotations:
    aws-eks-best-practices/description: >-
      Adds toleration - https://kubernetes.io/docs/concepts/scheduling-eviction/taint-
and-toleration/
spec:
  applyTo:
  - groups: [""]
    kinds: ["Pod"]
    versions: ["v1"]
  match:
    namespaces: ["tenants-x"]
  location: "spec.tolerations"
  parameters:
    assign:
      value:
      - key: "tenant"
        operator: "Equal"
        value: "tenants-x"
        effect: "NoSchedule"
```

Le politiche di cui sopra sono specifiche dei pod; ciò è dovuto ai percorsi degli elementi mutati negli elementi delle policy. `location` È possibile scrivere politiche aggiuntive per gestire le risorse che

creano pod, come le risorse Deployment e Job. Le politiche elencate e altri esempi sono disponibili nel GitHubprogetto complementare di questa guida.

Il risultato di queste due mutazioni è che i baccelli sono attratti dal nodo desiderato e allo stesso tempo non sono respinti dalla specifica contaminazione del nodo. Per verificarlo, possiamo vedere i frammenti di output di due `kubectl` chiamate con cui etichettare i nodi e inserire i pod nel `tenant=tenants-x` namespace. `tenants-x`

```
kubectl get nodes -l tenant=tenants-x
NAME
ip-10-0-11-255...
ip-10-0-28-81...
ip-10-0-43-107...

kubectl -n tenants-x get pods -owide
NAME                                 READY   STATUS    RESTARTS   AGE   IP
 NODE
tenant-test-deploy-58b895ff87-2q7xw   1/1     Running   0          13s   10.0.42.143
 ip-10-0-43-107...
tenant-test-deploy-58b895ff87-9b6hg   1/1     Running   0          13s   10.0.18.145
 ip-10-0-28-81...
tenant-test-deploy-58b895ff87-nxvw5   1/1     Running   0          13s   10.0.30.117
 ip-10-0-28-81...
tenant-test-deploy-58b895ff87-vw796   1/1     Running   0          13s   10.0.3.113
 ip-10-0-11-255...
tenant-test-pod                       1/1     Running   0          13s   10.0.35.83
 ip-10-0-43-107...
```

Come possiamo vedere dagli output precedenti, tutti i pod sono programmati sui nodi etichettati con. `tenant=tenants-x` In poche parole, i pod funzioneranno solo sui nodi desiderati, mentre gli altri pod (senza le affinità e le tolleranze richieste) no. I carichi di lavoro degli inquilini sono efficacemente isolati.

Di seguito è riportato un esempio di specifica mutata del pod.

```
apiVersion: v1
kind: Pod
metadata:
  name: tenant-test-pod
  namespace: tenants-x
spec:
  affinity:
```

```
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: tenant
            operator: In
            values:
            - tenants-x
...
  tolerations:
  - effect: NoSchedule
    key: tenant
    operator: Equal
    value: tenants-x
...
```

> **Important**
>
> Gli strumenti di gestione delle policy integrati nel flusso di richieste del server dell'API Kubernetes, che utilizzano webhook di ammissione mutanti e convalidanti, sono progettati per rispondere alla richiesta del server API entro un periodo di tempo specificato. Di solito si tratta di 3 secondi o meno. Se la chiamata webhook non riesce a restituire una risposta entro il tempo configurato, la and/or convalida della mutazione della richiesta del server API in entrata può avvenire o meno. Questo comportamento dipende dal fatto che le configurazioni del webhook di ammissione siano impostate su Fail Open o Fail Close.

Negli esempi precedenti, abbiamo utilizzato politiche scritte per OPA/Gatekeeper. Tuttavia, esistono altri strumenti di gestione delle policy che gestiscono anche il nostro caso d'uso relativo alla selezione dei nodi. Ad esempio, questa policy di Kyverno potrebbe essere utilizzata per gestire la mutazione dell'affinità dei nodi.

> **Note**
>
> Se funzionano correttamente, la modifica delle politiche influirà sulle modifiche desiderate ai payload delle richieste del server API in entrata. Tuttavia, è necessario includere anche le politiche di convalida per verificare che si verifichino le modifiche desiderate, prima che le modifiche possano persistere. Ciò è particolarmente importante quando si utilizzano queste

politiche per tenant-to-node l'isolamento. È inoltre consigliabile includere politiche di controllo per verificare regolarmente la presenza di configurazioni indesiderate nel cluster.

Riferimenti

- k-rail Progettato per aiutarti a proteggere un ambiente multi-tenant attraverso l'applicazione di determinate politiche.
- Pratiche di sicurezza per applicazioni MultiTenant SaaS che utilizzano Amazon EKS

Multi-tenancy rigida

La multi-tenancy rigida può essere implementata fornendo cluster separati per ogni tenant. Sebbene ciò fornisca un isolamento molto forte tra gli inquilini, presenta diversi inconvenienti.

Innanzitutto, quando si hanno molti inquilini, questo approccio può diventare rapidamente costoso. Non solo dovrai pagare i costi del piano di controllo per ogni cluster, ma non potrai condividere le risorse di elaborazione tra i cluster. Ciò alla fine causerà una frammentazione in cui un sottoinsieme dei cluster viene sottoutilizzato mentre altri vengono sovrautilizzati.

In secondo luogo, probabilmente dovrai acquistare o creare strumenti speciali per gestire tutti questi cluster. Col tempo, la gestione di centinaia o migliaia di cluster potrebbe semplicemente diventare troppo complicata.

Infine, la creazione di un cluster per tenant sarà lenta rispetto alla creazione di un namespace. Tuttavia, può essere necessario un approccio basato sulla tenancy in settori altamente regolamentati o in ambienti SaaS in cui è richiesto un forte isolamento.

Direzioni future

La community Kubernetes ha riconosciuto le attuali carenze della multi-tenancy soft e le sfide legate alla multi-tenancy rigida. Il Multi-Tenancy Special Interest Group (SIG) sta cercando di ovviare a queste carenze attraverso diversi progetti di incubazione, tra cui Hierarchical Namespace Controller (HNC) e Virtual Cluster.

La proposta HNC (KEP) descrive un modo per creare relazioni padre-figlio tra namespace con l'ereditarietà degli oggetti [policy] insieme alla possibilità per gli amministratori tenant di creare sottonamespace.

La proposta Virtual Cluster descrive un meccanismo per la creazione di istanze separate dei servizi del piano di controllo, tra cui il server API, il controller manager e lo scheduler, per ogni tenant all'interno del cluster (noto anche come «Kubernetes on Kubernetes»).

La proposta Multi-Tenancy Benchmarks fornisce linee guida per la condivisione di cluster utilizzando namespace per l'isolamento e la segmentazione e uno strumento a riga di comando kubectl-mtb per convalidare la conformità alle linee guida.

Strumenti e risorse per la gestione di più cluster

- Banzai Cloud
- Comandante
- Lente
- Nirmata
- Rafay
- Allevatore
- Weave Flux

Controllo e registrazione

> **Tip**
>
> Esplora le best practice tramite i workshop Amazon EKS.

La raccolta e l'analisi dei log [di controllo] è utile per una serie di motivi diversi. I log possono aiutare nell'analisi e nell'attribuzione delle cause principali, ad esempio nell'attribuire una modifica a un particolare utente. Una volta raccolti un numero sufficiente di log, è possibile utilizzarli anche per rilevare comportamenti anomali. Su EKS, i log di controllo vengono inviati ad Amazon Cloudwatch Logs. La politica di audit per EKS è la seguente:

```
apiVersion: audit.k8s.io/v1beta1
kind: Policy
rules:
  # Log full request and response for changes to aws-auth ConfigMap in kube-system
 namespace
  - level: RequestResponse
```

```
    namespaces: ["kube-system"]
    verbs: ["update", "patch", "delete"]
    resources:
      - group: "" # core
        resources: ["configmaps"]
        resourceNames: ["aws-auth"]
    omitStages:
      - "RequestReceived"
  # Do not log watch operations performed by kube-proxy on endpoints and services
  - level: None
    users: ["system:kube-proxy"]
    verbs: ["watch"]
    resources:
      - group: "" # core
        resources: ["endpoints", "services", "services/status"]
  # Do not log get operations performed by kubelet on nodes and their statuses
  - level: None
    users: ["kubelet"] # legacy kubelet identity
    verbs: ["get"]
    resources:
      - group: "" # core
        resources: ["nodes", "nodes/status"]
  # Do not log get operations performed by the system:nodes group on nodes and their
 statuses
  - level: None
    userGroups: ["system:nodes"]
    verbs: ["get"]
    resources:
      - group: "" # core
        resources: ["nodes", "nodes/status"]
  # Do not log get and update operations performed by controller manager, scheduler,
 and endpoint-controller on endpoints in kube-system namespace
  - level: None
    users:
      - system:kube-controller-manager
      - system:kube-scheduler
      - system:serviceaccount:kube-system:endpoint-controller
    verbs: ["get", "update"]
    namespaces: ["kube-system"]
    resources:
      - group: "" # core
        resources: ["endpoints"]
  # Do not log get operations performed by apiserver on namespaces and their statuses/
finalizations
```

```
  - level: None
    users: ["system:apiserver"]
    verbs: ["get"]
    resources:
      - group: "" # core
        resources: ["namespaces", "namespaces/status", "namespaces/finalize"]
  # Do not log get and list operations performed by controller manager on
 metrics.k8s.io resources
  - level: None
    users:
      - system:kube-controller-manager
    verbs: ["get", "list"]
    resources:
      - group: "metrics.k8s.io"
  # Do not log access to health, version, and swagger non-resource URLs
  - level: None
    nonResourceURLs:
      - /healthz*
      - /version
      - /swagger*
  # Do not log events resources
  - level: None
    resources:
      - group: "" # core
        resources: ["events"]
  # Log request for updates/patches to nodes and pods statuses by kubelet and node
 problem detector
  - level: Request
    users: ["kubelet", "system:node-problem-detector", "system:serviceaccount:kube-
system:node-problem-detector"]
    verbs: ["update", "patch"]
    resources:
      - group: "" # core
        resources: ["nodes/status", "pods/status"]
    omitStages:
      - "RequestReceived"
  # Log request for updates/patches to nodes and pods statuses by system:nodes group
  - level: Request
    userGroups: ["system:nodes"]
    verbs: ["update", "patch"]
    resources:
      - group: "" # core
        resources: ["nodes/status", "pods/status"]
    omitStages:
```

```
    - "RequestReceived"
# Log delete collection requests by namespace-controller in kube-system namespace
- level: Request
  users: ["system:serviceaccount:kube-system:namespace-controller"]
  verbs: ["deletecollection"]
  omitStages:
    - "RequestReceived"
# Log metadata for secrets, configmaps, and tokenreviews to protect sensitive data
- level: Metadata
  resources:
    - group: "" # core
      resources: ["secrets", "configmaps"]
    - group: authentication.k8s.io
      resources: ["tokenreviews"]
  omitStages:
    - "RequestReceived"
# Log requests for serviceaccounts/token resources
- level: Request
  resources:
    - group: "" # core
      resources: ["serviceaccounts/token"]
# Log get, list, and watch requests for various resource groups
- level: Request
  verbs: ["get", "list", "watch"]
  resources:
    - group: "" # core
    - group: "admissionregistration.k8s.io"
    - group: "apiextensions.k8s.io"
    - group: "apiregistration.k8s.io"
    - group: "apps"
    - group: "authentication.k8s.io"
    - group: "authorization.k8s.io"
    - group: "autoscaling"
    - group: "batch"
    - group: "certificates.k8s.io"
    - group: "extensions"
    - group: "metrics.k8s.io"
    - group: "networking.k8s.io"
    - group: "policy"
    - group: "rbac.authorization.k8s.io"
    - group: "scheduling.k8s.io"
    - group: "settings.k8s.io"
    - group: "storage.k8s.io"
  omitStages:
```

```
    - "RequestReceived"
# Default logging level for known APIs to log request and response
- level: RequestResponse
  resources:
    - group: "" # core
    - group: "admissionregistration.k8s.io"
    - group: "apiextensions.k8s.io"
    - group: "apiregistration.k8s.io"
    - group: "apps"
    - group: "authentication.k8s.io"
    - group: "authorization.k8s.io"
    - group: "autoscaling"
    - group: "batch"
    - group: "certificates.k8s.io"
    - group: "extensions"
    - group: "metrics.k8s.io"
    - group: "networking.k8s.io"
    - group: "policy"
    - group: "rbac.authorization.k8s.io"
    - group: "scheduling.k8s.io"
    - group: "settings.k8s.io"
    - group: "storage.k8s.io"
  omitStages:
    - "RequestReceived"
# Default logging level for all other requests to log metadata only
- level: Metadata
  omitStages:
    - "RequestReceived"
```

Raccomandazioni

Abilita i registri di controllo

I log di controllo fanno parte dei log del piano di controllo Kubernetes gestiti da EKS e gestiti da EKS. Le istruzioni per i log enabling/disabling del piano di controllo, che includono i log per il server dell'API Kubernetes, il controller manager e lo scheduler, insieme al registro di controllo, sono disponibili qui, -plane-logs.html# -export. https://docs.aws.amazon.com/eks/ latest/userguide/control enabling-control-plane-log

> **Note**
>
> Quando abiliti la registrazione del piano di controllo, dovrai sostenere dei costi per l'archiviazione dei log. CloudWatch Ciò solleva una questione più ampia relativa ai costi correnti della sicurezza. In ultima analisi, dovrete soppesare tali costi rispetto al costo di una violazione della sicurezza, ad esempio perdite finanziarie, danni alla vostra reputazione, ecc. Potresti scoprire di poter proteggere adeguatamente il tuo ambiente implementando solo alcuni dei consigli di questa guida.

> **Warning**
>
> La dimensione massima per una voce CloudWatch Logs è di 1 MB, mentre la dimensione massima della richiesta dell'API Kubernetes è 1,5 MiB. Le voci di registro superiori a 1 MB verranno troncate o includeranno solo i metadati della richiesta.

Utilizza i metadati di controllo

I log di controllo di Kubernetes includono due annotazioni che indicano se una richiesta è stata autorizzata `authorization.k8s.io/decision` o meno e il motivo della decisione. `authorization.k8s.io/reason` Utilizza questi attributi per accertare il motivo per cui è stata consentita una particolare chiamata API.

Crea allarmi per eventi sospetti

Crea un allarme per avvisarti automaticamente in caso di aumento di 403 risposte proibite e 401 non autorizzate, quindi utilizza attributi come `hostsourceIPs`, e `k8s_user.username` per scoprire da dove provengono tali richieste.

Analizza i log con Log Insights

Usa CloudWatch Log Insights per monitorare le modifiche agli oggetti RBAC, ad esempio Ruoli, RoleBindings, ClusterRoles e. ClusterRoleBindings Di seguito sono riportate alcune query di esempio:

Elenca gli aggiornamenti a: `aws-auth` ConfigMap

```
fields @timestamp, @message
```

```
| filter @logStream like "kube-apiserver-audit"
| filter verb in ["update", "patch"]
| filter objectRef.resource = "configmaps" and objectRef.name = "aws-auth" and
 objectRef.namespace = "kube-system"
| sort @timestamp desc
```

Elenca la creazione di nuovi webhook di convalida o le modifiche apportate:

```
fields @timestamp, @message
| filter @logStream like "kube-apiserver-audit"
| filter verb in ["create", "update", "patch"] and responseStatus.code = 201
| filter objectRef.resource = "validatingwebhookconfigurations"
| sort @timestamp desc
```

Elenchi per creare, aggiornare ed eliminare le operazioni relative ai ruoli:

```
fields @timestamp, @message
| sort @timestamp desc
| limit 100
| filter objectRef.resource="roles" and verb in ["create", "update", "patch", "delete"]
```

Operazioni di creazione, aggiornamento ed eliminazione degli elenchi per RoleBindings:

```
fields @timestamp, @message
| sort @timestamp desc
| limit 100
| filter objectRef.resource="rolebindings" and verb in ["create", "update", "patch",
 "delete"]
```

Elenca le operazioni di creazione, aggiornamento ed eliminazione per ClusterRoles:

```
fields @timestamp, @message
| sort @timestamp desc
| limit 100
| filter objectRef.resource="clusterroles" and verb in ["create", "update", "patch",
 "delete"]
```

Elenca le operazioni di creazione, aggiornamento ed eliminazione per ClusterRoleBindings:

```
fields @timestamp, @message
| sort @timestamp desc
```

```
| limit 100
| filter objectRef.resource="clusterrolebindings" and verb in ["create", "update",
 "patch", "delete"]
```

Registra le operazioni di lettura non autorizzate su Secrets:

```
fields @timestamp, @message
| sort @timestamp desc
| limit 100
| filter objectRef.resource="secrets" and verb in ["get", "watch", "list"] and
 responseStatus.code="401"
| stats count() by bin(1m)
```

Elenco delle richieste anonime non riuscite:

```
fields @timestamp, @message, sourceIPs.0
| sort @timestamp desc
| limit 100
| filter user.username="system:anonymous" and responseStatus.code in ["401", "403"]
```

Controlla i tuoi CloudTrail registri

Le APIs chiamate AWS tramite pod che utilizzano IAM Roles for Service Accounts (IRSA) vengono automaticamente registrate CloudTrail insieme al nome dell'account di servizio. Se nel registro compare il nome di un account di servizio che non era esplicitamente autorizzato a chiamare un'API, potrebbe essere un'indicazione che la policy di fiducia del ruolo IAM non è stata configurata correttamente. In generale, Cloudtrail è un ottimo modo per attribuire chiamate API AWS a principi IAM specifici.

Usa Insights per scoprire attività sospette CloudTrail

CloudTrail insights analizza automaticamente gli eventi di gestione delle scritture provenienti dai CloudTrail trail e ti avvisa di attività insolite. Questo può aiutarti a identificare quando c'è un aumento del volume di chiamate in scrittura APIs nel tuo account AWS, anche da pod che utilizzano IRSA per assumere un ruolo IAM. Per ulteriori informazioni, consulta Announcing CloudTrail Insights: Identify and Response to Unusual API Activity.

Risorse aggiuntive

Con l'aumento del volume dei log, l'analisi e il filtraggio con Log Insights o un altro strumento di analisi dei log potrebbero diventare inefficaci. In alternativa, potresti prendere in considerazione

l'utilizzo di Sysdig Falco e ekscloudwatch. Falco analizza i log di controllo e segnala anomalie o abusi per un periodo di tempo prolungato. Il progetto ekscloudwatch inoltra gli eventi dei registri di controllo a Falco per l'analisi. CloudWatch Falco fornisce una serie di regole di controllo predefinite oltre alla possibilità di aggiungerne di proprie.

Un'altra opzione potrebbe essere quella di archiviare i log di controllo in S3 e utilizzare l'algoritmo SageMaker Random Cut Forest per comportamenti anomali che richiedono ulteriori indagini.

Strumenti e risorse

I seguenti progetti commerciali e open source possono essere utilizzati per valutare l'allineamento del cluster alle migliori pratiche consolidate:

- Workshop di immersione nella sicurezza di Amazon EKS - Detective Controls
- kubeaudit
- kube-scan assegna un punteggio di rischio ai carichi di lavoro in esecuzione nel cluster in conformità con il framework Kubernetes Common Configuration Scoring System
- kubesec.io
- polare
- A tribordo
- Snyk
- Kubescape Kubescape è uno strumento di sicurezza Kubernetes open source che analizza cluster, file YAML e grafici Helm. Rileva configurazioni errate in base a più framework (inclusi NSA-CISA e MITRE ATT&CK®).

Sicurezza di rete

> **Tip**
>
> Esplora le best practice tramite i workshop Amazon EKS.

La sicurezza di rete ha diverse sfaccettature. Il primo riguarda l'applicazione di regole che limitano il flusso del traffico di rete tra i servizi. Il secondo riguarda la crittografia del traffico mentre è in transito. I meccanismi per implementare queste misure di sicurezza su EKS sono vari, ma spesso includono i seguenti elementi:

Controllo del traffico

- Policy di rete
- Gruppi di sicurezza

Crittografia di rete

- Service Mesh
- Interfacce di rete per container () CNIs
- Controller di ingresso e sistemi di bilanciamento del carico
- Istanze Nitro
- ACM Private CA con cert-manager

Politica di rete

All'interno di un cluster Kubernetes, tutte le comunicazioni da Pod a Pod sono consentite per impostazione predefinita. Sebbene questa flessibilità possa contribuire a promuovere la sperimentazione, non è considerata sicura. Le policy di rete di Kubernetes offrono un meccanismo per limitare il traffico di rete tra i Pod (spesso denominato traffico Est/Ovest) e tra i Pod e i servizi esterni. Le politiche di rete Kubernetes operano ai livelli 3 e 4 del modello OSI. Le policy di rete utilizzano pod, selettori di namespace ed etichette per identificare i pod di origine e destinazione, ma possono anche includere indirizzi IP, numeri di porta, protocolli o una combinazione di questi. Le politiche di rete possono essere applicate alle connessioni in entrata o in uscita al pod, spesso chiamate regole di ingresso e uscita.

Con il supporto nativo delle policy di rete del plugin Amazon VPC CNI, puoi implementare policy di rete per proteggere il traffico di rete nei cluster Kubernetes. Questo si integra con l'API Kubernetes Network Policy upstream, garantendo compatibilità e aderenza agli standard Kubernetes. Puoi definire le politiche utilizzando diversi identificatori supportati dall'API upstream. Per impostazione predefinita, tutto il traffico in entrata e in uscita è consentito a un pod. Quando viene specificata una policy di rete con un PolicyType Ingress, solo le connessioni consentite nel pod sono quelle provenienti dal nodo del pod e quelle consentite dalle regole di ingresso. Lo stesso vale per le regole di uscita. Se vengono definite più regole, al momento della decisione viene presa in considerazione l'unione di tutte le regole. Pertanto, l'ordine di valutazione non influisce sul risultato della politica.

> **Important**
>
> Quando si effettua il provisioning per la prima volta di un cluster EKS, la funzionalità VPC CNI Network Policy non è abilitata per impostazione predefinita. Assicurati di aver distribuito la versione del componente aggiuntivo VPC CNI supportata e `ENABLE_NETWORK_POLICY` imposta il flag sul componente aggiuntivo vpc-cni `true` per abilitarlo. Consulta la guida per l'utente di Amazon EKS per istruzioni dettagliate.

Raccomandazioni

Guida introduttiva alle politiche di rete: segui il principio dei privilegi minimi

Crea una politica di rifiuto predefinita

Come per le politiche RBAC, si consiglia di seguire i principi di accesso con privilegi minimi per quanto riguarda le politiche di rete. Inizia creando una politica di negazione totale che limiti tutto il traffico in entrata e in uscita all'interno di un namespace.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny
  namespace: default
spec:
  podSelector: {}
  policyTypes:
  - Ingress
  - Egress
```

rifiuto per default

		Target		Source				
#	Policy name	Namespace	Pods	Namespace	Pods	Subnet	Ports	
1	default-deny	default	Any	Any	Any	Any	Any	

> **Note**
>
> L'immagine qui sopra è stata creata dal network policy viewer di Tufin.

Crea una regola per consentire le query DNS

Una volta impostata la regola deny all predefinita, puoi iniziare a sovrapporre regole aggiuntive, ad esempio una regola che consente ai pod di interrogare CoredNS per la risoluzione dei nomi.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-dns-access
  namespace: default
spec:
  podSelector:
    matchLabels: {}
  policyTypes:
  - Egress
  egress:
  - to:
    - namespaceSelector:
        matchLabels:
          kubernetes.io/metadata.name: kube-system
      podSelector:
        matchLabels:
          k8s-app: kube-dns
    ports:
    - protocol: UDP
      port: 53
```

allow-dns-access

		Target		Destination				
#	Policy name	Namespace	Pods	Namespace	Pods	Subnet	Ports	
1	allow-dns-access	default	Any	name: kube-system	Any		UDP: 53	

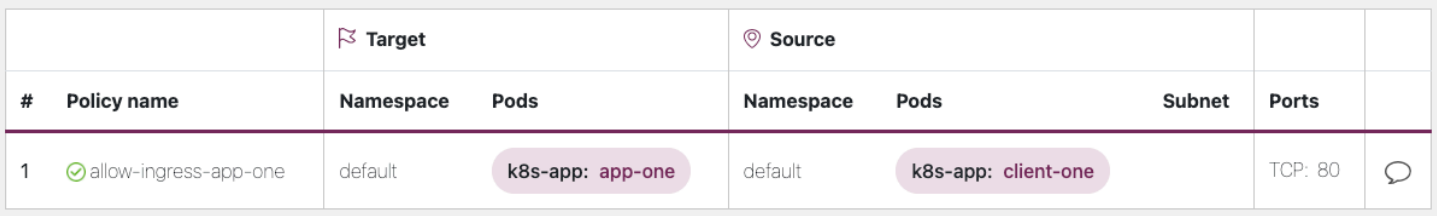
Aggiungi regole in modo incrementale per consentire in modo selettivo il flusso di traffico tra namespace e pod

Comprendi i requisiti dell'applicazione e crea regole di ingresso e uscita granulari in base alle esigenze. L'esempio seguente mostra come limitare il traffico in ingresso dalla porta 80 a quella da. `app-one` `client-one` Questo aiuta a ridurre al minimo la superficie di attacco e riduce il rischio di accesso non autorizzato.

```
apiVersion: networking.k8s.io/v1
```

```
kind: NetworkPolicy
metadata:
  name: allow-ingress-app-one
  namespace: default
spec:
  podSelector:
    matchLabels:
      k8s-app: app-one
  policyTypes:
  - Ingress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          k8s-app: client-one
    ports:
    - protocol: TCP
      port: 80
```

allow-ingress-app-one



		Target		Source				
#	Policy name	Namespace	Pods	Namespace	Pods	Subnet	Ports	
1	allow-ingress-app-one	default	k8s-app: app-one	default	k8s-app: client-one		TCP: 80	

Monitoraggio dell'applicazione delle politiche di rete

- Usa l'editor delle politiche di rete
  - L'editor delle politiche di rete aiuta con le visualizzazioni, il punteggio di sicurezza, si genera automaticamente dai log dei flussi di rete
  - Crea politiche di rete in modo interattivo
- Registri di controllo
  - Esamina regolarmente i registri di controllo del tuo cluster EKS
  - I registri di controllo forniscono numerose informazioni sulle azioni eseguite sul cluster, comprese le modifiche alle politiche di rete
  - Utilizzate queste informazioni per tenere traccia delle modifiche alle politiche di rete nel tempo e rilevare eventuali modifiche non autorizzate o impreviste
- Test automatizzati

  - Implementa i test automatizzati creando un ambiente di test che rispecchi il tuo ambiente di produzione e distribuisci periodicamente carichi di lavoro che tentano di violare le politiche di rete.
- Monitoraggio delle metriche
  - Configura i tuoi agenti di osservabilità per acquisire le metriche di Prometheus dagli agenti del nodo VPC CNI, che consentono di monitorare lo stato degli agenti e gli errori dell'sdk.
- Controlla regolarmente le politiche di rete
  - Controllate periodicamente le vostre politiche di rete per assicurarvi che soddisfino i requisiti delle applicazioni correnti. Man mano che l'applicazione si evolve, un audit offre l'opportunità di rimuovere le regole di ingresso e uscita ridondanti e di assicurarsi che le applicazioni non dispongano di autorizzazioni eccessive.
- Assicurati che esistano politiche di rete utilizzando Open Policy Agent (OPA)
  - Utilizzate la politica OPA come illustrata di seguito per garantire che la politica di rete esista sempre prima dell'onboarding dei pod delle applicazioni. Questa politica nega l'onboarding dei pod k8s con un'etichetta se non esiste una politica di rete corrispondente. `k8s-app: sample-app`

```
package kubernetes.admission
import data.kubernetes.networkpolicies

deny[msg] {
    input.request.kind.kind == "Pod"
    pod_label_value := {v["k8s-app"] | v := input.request.object.metadata.labels}
    contains_label(pod_label_value, "sample-app")
    np_label_value := {v["k8s-app"] | v :=
 networkpolicies[_].spec.podSelector.matchLabels}
    not contains_label(np_label_value, "sample-app")
    msg:= sprintf("The Pod %v could not be created because it is missing an associated
 Network Policy.", [input.request.object.metadata.name])
}
contains_label(arr, val) {
    arr[_] == val
}
```

Risoluzione dei problemi

Monitora vpc-network-policy-controller i log di Node-Agent

Abilita i log del gestore del controller del piano di controllo EKS per diagnosticare la funzionalità delle politiche di rete. È possibile trasmettere i log del control plane a un gruppo di CloudWatch log e utilizzare CloudWatchLog insights per eseguire interrogazioni avanzate. Dai log, è possibile visualizzare quali oggetti dell'endpoint pod sono stati risolti in una policy di rete, lo stato di riconciliazione delle policy ed eseguire il debug se la policy funziona come previsto.

Inoltre, Amazon VPC CNI consente di abilitare la raccolta e l'esportazione dei log di applicazione delle policy su Amazon Cloudwatch dai nodi di lavoro EKS. Una volta abilitato, puoi sfruttare CloudWatchContainer Insights per fornire informazioni sull'utilizzo relativo alle politiche di rete.

Amazon VPC CNI fornisce anche un SDK che fornisce un'interfaccia per interagire con i programmi eBPF sul nodo. L'SDK viene installato quando `aws-node` viene distribuito sui nodi. È possibile trovare il file binario SDK installato `/opt/cni/bin` nella directory del nodo. Al momento del lancio, l'SDK fornisce supporto per funzionalità fondamentali come l'ispezione di programmi e mappe eBPF.

```
sudo /opt/cni/bin/aws-eks-na-cli ebpf progs
```

Registra i metadati del traffico di rete

AWS VPC Flow Logs acquisisce i metadati sul traffico che scorre attraverso un VPC, come l'indirizzo IP e la porta di origine e destinazione, insieme ai pacchetti accettati/eliminati. Queste informazioni potrebbero essere analizzate per individuare attività sospette o insolite tra le risorse all'interno del VPC, inclusi i Pods. Tuttavia, poiché gli indirizzi IP dei pod cambiano spesso man mano che vengono sostituiti, i Flow Logs potrebbero non essere sufficienti da soli. Calico Enterprise estende i Flow Logs con etichette per pod e altri metadati, semplificando la decifrazione dei flussi di traffico tra i pod.

Gruppi di sicurezza

EKS utilizza AWS VPC Security Groups (SGs) per controllare il traffico tra il piano di controllo Kubernetes e i nodi di lavoro del cluster. I gruppi di sicurezza vengono utilizzati anche per controllare il traffico tra i nodi di lavoro e altre risorse VPC e gli indirizzi IP esterni. Quando esegui il provisioning di un cluster EKS (con Kubernetes versione 1.14-eks.3 o successiva), viene creato automaticamente un gruppo di sicurezza del cluster. Questo gruppo di sicurezza consente una comunicazione senza restrizioni tra il piano di controllo EKS e i nodi dei gruppi di nodi gestiti. Per semplicità, si consiglia di aggiungere il cluster SG a tutti i gruppi di nodi, inclusi i gruppi di nodi non gestiti.

Prima della versione 1.14 di Kubernetes e della versione EKS.3, esistevano gruppi di sicurezza separati configurati per il piano di controllo EKS e i gruppi di nodi. Le regole minime e suggerite per i gruppi di sicurezza del piano di controllo e del gruppo di nodi sono disponibili all'indirizzo -group-reqs.html. https://docs.aws.amazon.com/eks/ latest/userguide/sec Le regole minime per il gruppo di sicurezza del piano di controllo consentono la porta 443 in entrata dal nodo di lavoro SG. Questa regola è ciò che consente ai kubelet di comunicare con il server API Kubernetes. Include anche la porta 10250 per il traffico in uscita verso il nodo di lavoro SG; 10250 è la porta su cui i kubelet ascoltano. Analogamente, le regole del gruppo minimo di nodi consentono la porta 10250 in entrata dal piano di controllo SG e la porta 443 in uscita verso il piano di controllo SG. Infine esiste una regola che consente la comunicazione senza restrizioni tra i nodi all'interno di un gruppo di nodi.

Se devi controllare la comunicazione tra i servizi eseguiti all'interno del cluster e i servizi eseguiti all'esterno del cluster, ad esempio un database RDS, prendi in considerazione i gruppi di sicurezza per i pod. Con i gruppi di sicurezza per i pod, puoi assegnare un gruppo di sicurezza esistente a una raccolta di pod.

> **Warning**
>
> Se si fa riferimento a un gruppo di sicurezza che non esiste prima della creazione dei pod, i pod non verranno pianificati.

È possibile controllare quali pod sono assegnati a un gruppo di sicurezza creando un `SecurityGroupPolicy` oggetto e specificando un o un. `PodSelector` `ServiceAccountSelector` L'impostazione dei selettori su `{}` assegnerà il SGs riferimento in a tutti i pod in un namespace o `SecurityGroupPolicy` a tutti gli account di servizio in uno spazio dei nomi. Assicurati di aver preso visione di tutte le considerazioni prima di implementare i gruppi di sicurezza per i pod.

> **Important**
>
> Se si utilizza SGs for pod, è necessario creare SGs che consentano la porta 53 in uscita verso il gruppo di sicurezza del cluster. Allo stesso modo, è necessario aggiornare il gruppo di sicurezza del cluster in modo che accetti il traffico in entrata della porta 53 dal gruppo di sicurezza dei pod.

> **Important**
>
> I limiti per i gruppi di sicurezza si applicano ancora quando si utilizzano i gruppi di sicurezza per i pod, quindi usali con cautela.

> **Important**
>
> È necessario creare regole per il traffico in entrata dal gruppo di sicurezza del cluster (kubelet) per tutte le sonde configurate per il pod.

> **Important**
>
> I gruppi di sicurezza per i pod si basano su una funzionalità nota come ENI trunking, creata per aumentare la densità ENI di un'istanza EC2. Quando un pod viene assegnato a un SG, un controller VPC associa un ramo ENI del gruppo di nodi al pod. Se non ci sono abbastanza branch ENIs disponibili in un gruppo di nodi al momento della pianificazione del pod, il pod rimarrà nello stato in sospeso. Il numero di branch che ENIs un'istanza può supportare varia a seconda dell'istanzatype/family. See https://docs.aws.amazon.com/eks/latest/userguide/security: groups-for-pods .html# supported-instance-types per ulteriori dettagli.

Sebbene i gruppi di sicurezza per i pod offrano un modo nativo di AWS per controllare il traffico di rete all'interno e all'esterno del cluster senza il sovraccarico di un demone di policy, sono disponibili altre opzioni. Ad esempio, il motore di policy Cilium consente di fare riferimento a un nome DNS in una policy di rete. Calico Enterprise include un'opzione per mappare le policy di rete ai gruppi di sicurezza AWS. Se hai implementato una service mesh come Istio, puoi utilizzare un gateway di uscita per limitare l'uscita della rete a domini o indirizzi IP specifici e completamente qualificati. Per ulteriori informazioni su questa opzione, leggi la serie in tre parti sul controllo del traffico in uscita in Istio.

Quando utilizzare Network Policy vs Security Group for Pods?

Quando utilizzare la politica di rete Kubernetes

- Controllo del traffico pod-to-pod

  - Adatto per controllare il traffico di rete tra i pod all'interno di un cluster (traffico est-ovest)
- Controlla il traffico a livello di indirizzo IP o porta (OSI layer 3 o 4)

Quando usare AWS Security groups for pods (SGP)

- Sfrutta le configurazioni AWS esistenti
  - Se disponi già di un set complesso di gruppi di sicurezza EC2 che gestiscono l'accesso ai servizi AWS e stai migrando le applicazioni dalle istanze EC2 a EKS, SGPs può essere un'ottima scelta che ti consente di riutilizzare le risorse dei gruppi di sicurezza e applicarle ai tuoi pod.
- Controlla l'accesso ai servizi AWS
  - Le tue applicazioni in esecuzione all'interno di un cluster EKS vogliono comunicare con altri servizi AWS (database RDS), da utilizzare SGPs come meccanismo efficiente per controllare il traffico dai pod ai servizi AWS.
- Isolamento del traffico Pod & Node
  - Se desideri separare completamente il traffico dei pod dal resto del traffico del nodo, usa la `POD_SECURITY_GROUP_ENFORCING_MODE=strict` modalità SGP in.

Le migliori pratiche relative all'utilizzo dei gruppi di sicurezza per i pod e della politica di rete

- Sicurezza a più livelli
  - Utilizza una combinazione di policy di rete SGP e Kubernetes per un approccio di sicurezza a più livelli
  - Utilizzalo SGPs per limitare l'accesso a livello di rete ai servizi AWS che non fanno parte di un cluster, mentre le policy di rete Kubernetes possono limitare il traffico di rete tra i pod all'interno del cluster
- Principio del privilegio minimo
  - Consenti solo il traffico necessario tra pod o namespace
- Segmenta le tue applicazioni
  - Ove possibile, segmenta le applicazioni in base alla politica di rete per ridurre il raggio di esplosione se un'applicazione viene compromessa
- Mantieni le politiche semplici e chiare

- Le policy di rete Kubernetes possono essere piuttosto granulari e complesse, è meglio mantenerle il più semplici possibile per ridurre il rischio di errori di configurazione e alleggerire il sovraccarico di gestione
- Riduci la superficie di attacco
  - Riduci al minimo la superficie di attacco limitando l'esposizione delle tue applicazioni

> **Important**
>
> Security Groups for pods offre due modalità di applicazione: e. `strict` `standard` È necessario utilizzare `standard` la modalità quando si utilizzano le funzionalità Network Policy e Security Groups for pods in un cluster EKS.

Quando si tratta di sicurezza della rete, un approccio a più livelli è spesso la soluzione più efficace. L'uso combinato della policy di rete di Kubernetes e di SGP può fornire una solida defense-in-depth strategia per le applicazioni eseguite in EKS.

Service Mesh Policy Enforcement o policy di rete Kubernetes

A `service mesh` è un livello di infrastruttura dedicato che puoi aggiungere alle tue applicazioni. Consente di aggiungere in modo trasparente funzionalità come l'osservabilità, la gestione del traffico e la sicurezza, senza aggiungerle al proprio codice.

Service Mesh applica le policy al Layer 7 (applicazione) del modello OSI, mentre le policy di rete Kubernetes operano al Layer 3 (rete) e al Layer 4 (trasporto). Ci sono molte offerte in questo spazio come AWSAppMesh, Istio, Linkerd, ecc.,

Quando utilizzare Service Mesh per l'applicazione delle politiche

- Hai già investito in una service mesh
- Hai bisogno di funzionalità più avanzate come la gestione del traffico, l'osservabilità e la sicurezza
  - Controllo del traffico, bilanciamento del carico, interruzione del circuito, limitazione della velocità, timeout ecc.
  - Informazioni dettagliate sulle prestazioni dei tuoi servizi (latenza, tassi di errore, richieste al secondo, volumi di richieste ecc.)
  - Desiderate implementare e sfruttare Service Mesh per funzionalità di sicurezza come gli MTL

Scegli la policy di rete Kubernetes per casi d'uso più semplici

- Limita i pod che possono comunicare tra loro
- Le policy di rete richiedono meno risorse rispetto a una service mesh, il che le rende adatte per casi d'uso più semplici o per cluster più piccoli in cui il sovraccarico di esecuzione e gestione di una service mesh potrebbe non essere giustificato

> **Note**
>
> Le policy di rete e la Service Mesh possono anche essere utilizzate insieme. Utilizza le policy di rete per fornire un livello base di sicurezza e isolamento tra i pod, quindi utilizza una service mesh per aggiungere funzionalità aggiuntive come la gestione del traffico, l'osservabilità e la sicurezza.

ThirdParty Motori di policy di rete

Prendi in considerazione un motore di policy di rete di terze parti quando hai requisiti di policy avanzati come politiche di rete globali, supporto per regole basate sui nomi host DNS, regole di livello 7, regole ServiceAccount basate e deny/log azioni esplicite, ecc. Calico è un motore di policy open source di Tigera che funziona bene con EKS. Oltre a implementare l'intero set di funzionalità delle policy di rete di Kubernetes, Calico supporta policy di rete estese con un set più ricco di funzionalità, incluso il supporto per le regole di livello 7, ad esempio HTTP, se integrato con Istio. Le policy di Calico possono essere applicate a namespace, pod, account di servizio o a livello globale. Quando le politiche riguardano un account di servizio, associa una serie di regole a quell'account di servizio. ingress/egress Con le regole RBAC appropriate, è possibile impedire ai team di ignorare tali regole, consentendo ai professionisti della sicurezza IT di delegare in modo sicuro l'amministrazione dei namespace. Isovalent, i manutentori di Cilium, hanno anche esteso le politiche di rete per includere il supporto parziale per le regole di livello 7, ad esempio HTTP. Cilium supporta anche i nomi di host DNS, che possono essere utili per limitare il traffico tra Kubernetes Services/Pods e le risorse che vengono eseguite all'interno o all'esterno del tuo VPC. Al contrario, Calico Enterprise include una funzionalità che consente di mappare una policy di rete Kubernetes a un gruppo di sicurezza AWS, oltre a nomi host DNS.

Puoi trovare un elenco delle politiche di rete Kubernetes comuni all'indirizzo. https://github.com/ahmetb/ kubernetes-network-policy-recipes Un insieme di regole simili per Calico è disponibile all'indirizzo https://docs.projectcalico. org/security/calico-politica di rete.

Migrazione al motore di policy di rete Amazon VPC CNI

Per mantenere la coerenza ed evitare comportamenti imprevisti nelle comunicazioni tra i pod, si consiglia di implementare un solo Network Policy Engine nel cluster. Se desideri migrare da 3P a VPC CNI Network Policy Engine, ti consigliamo di convertire le tue risorse 3P esistenti in Kubernetes prima di abilitare il supporto NetworkPolicy CRDs delle policy di rete VPC CNI. NetworkPolicy Inoltre, verifica le politiche migrate in un cluster di test separato prima di applicarle nel tuo ambiente di produzione. Ciò consente di identificare e risolvere eventuali problemi o incongruenze nel comportamento di comunicazione dei pod.

Strumento di migrazione

Per aiutarti nel processo di migrazione, abbiamo sviluppato uno strumento chiamato K8s Network Policy Migrator che converte le policy di rete esistenti in politiche di Calico/Cilium rete native di CRDs Kubernetes. Dopo la conversione, puoi testare direttamente le politiche di rete convertite sui tuoi nuovi cluster che eseguono il controller di policy di rete VPC CNI. Lo strumento è progettato per aiutarti a semplificare il processo di migrazione e garantire una transizione senza intoppi.

> **Important**
>
> Lo strumento di migrazione convertirà solo le politiche 3P compatibili con l'API nativa delle politiche di rete Kubernetes. Se utilizzi le funzionalità avanzate di policy di rete offerte dai plugin 3P, Migration Tool le ignorerà e le segnalerà.

Tieni presente che lo strumento di migrazione non è attualmente supportato dal team di progettazione delle policy di rete CNI di AWS VPC, ma viene messo a disposizione dei clienti con la massima diligenza. Ti invitiamo a utilizzare questo strumento per facilitare il processo di migrazione. Nel caso in cui riscontrassi problemi o bug con lo strumento, ti chiediamo gentilmente di creare un problema. GitHub Il tuo feedback è prezioso per noi e contribuirà al miglioramento continuo dei nostri servizi.

Risorse aggiuntive

- Kubernetes e Tigera: politiche di rete, sicurezza e audit
- Calico Enterprise
- Cilium
- NetworkPolicyEditor, un editor interattivo di policy di Cilium

- Inspektor Gadget consiglia il gadget per le politiche di rete Suggerisce politiche di rete basate su un'analisi del traffico di rete

Crittografia dei dati in transito

Le applicazioni che devono essere conformi a PCI, HIPAA o altre normative potrebbero dover crittografare i dati mentre sono in transito. Al giorno d'oggi TLS è di fatto la scelta ideale per crittografare il traffico via cavo. TLS, come il suo predecessore SSL, fornisce comunicazioni sicure su una rete utilizzando protocolli crittografici. TLS utilizza la crittografia simmetrica in cui le chiavi per crittografare i dati vengono generate sulla base di un segreto condiviso negoziato all'inizio della sessione. Di seguito sono riportati alcuni modi in cui è possibile crittografare i dati in un ambiente Kubernetes.

Istanze Nitro

Il traffico scambiato tra i seguenti tipi di istanze Nitro, ad esempio C5n, G4, I3en, M5dn, M5n, P3dn, R5dn e R5n, viene crittografato automaticamente per impostazione predefinita. Quando è presente un hop intermedio, come un gateway di transito o un sistema di bilanciamento del carico, il traffico non è crittografato. Vedi Encryption in transit per ulteriori dettagli sulla crittografia in transito e per l'elenco completo dei tipi di istanze che supportano la crittografia di rete per impostazione predefinita.

Interfacce di rete per container () CNIs

WeaveNetpuò essere configurato per crittografare automaticamente tutto il traffico utilizzando la NaCl crittografia per il traffico «sleeve» e l' IPsec ESP per il traffico datapath veloce.

Service Mesh

La crittografia in transito può essere implementata anche con una mesh di servizi come App Mesh, Linkerd v2 e Istio. AppMesh supporta MTL con certificati X.509 o Envoy's Secret Discovery Service (SDS). Linkerd e Istio supportano entrambi gli MTL.

Il aws-app-mesh-examplesGitHub repository fornisce procedure dettagliate per la configurazione degli MTL utilizzando certificati X.509 e SPIRE come provider SDS con il contenitore Envoy:

- Configurazione degli MTL utilizzando i certificati X.509
- Configurazione di TLS tramite SPIRE (SDS)

App Mesh supporta anche la crittografia TLS con un certificato privato emesso da AWS Certificate Manager (ACM) o un certificato archiviato nel file system locale del nodo virtuale.

Il aws-app-mesh-examplesGitHub repository fornisce procedure dettagliate per la configurazione di TLS utilizzando certificati emessi da ACM e certificati inclusi nel pacchetto Envoy:

- Configurazione di TLS con certificati TLS forniti da file
- Configurazione di TLS con AWS Certificate Manager

Controller di ingresso e sistemi di bilanciamento del carico

I controller di ingresso consentono di indirizzare in modo intelligente l'elaborazione e richiedono un uso intensivo della CPU. HTTP/S traffic that emanates from outside the cluster to services running inside the cluster. Oftentimes, these Ingresses are fronted by a layer 4 load balancer, like the Classic Load Balancer or the Network Load Balancer (NLB). Encrypted traffic can be terminated at different places within the network, e.g. at the load balancer, at the ingress resource, or the Pod. How and where you terminate your SSL connection will ultimately be dictated by your organization's network security policy. For instance, if you have a policy that requires end-to-end encryption, you will have to decrypt the traffic at the Pod. This will place additional burden on your Pod as it will have to spend cycles establishing the initial handshake. Overall SSL/TLS Di conseguenza, se hai la flessibilità, prova a eseguire l'offload SSL presso Ingress o il load balancer.

Usa la crittografia con i sistemi di bilanciamento del carico AWS Elastic

AWS Application Load Balancer (ALB) e Network Load Balancer (NLB) supportano entrambi la crittografia del trasporto (SSL e TLS). L'`alb.ingress.kubernetes.io/certificate-arn`annotazione per l'ALB consente di specificare quali certificati aggiungere all'ALB. Se ometti l'annotazione, il controller tenterà di aggiungere certificati ai listener che lo richiedono abbinando i certificati AWS Certificate Manager (ACM) disponibili utilizzando il campo host. A partire da EKS v1.15 puoi usare l'`service.beta.kubernetes.io/aws-load-balancer-ssl-cert`annotazione con l'NLB come mostrato nell'esempio seguente.

```
apiVersion: v1
kind: Service
metadata:
  name: demo-app
  namespace: default
  labels:
    app: demo-app
```

```
  annotations:
     service.beta.kubernetes.io/aws-load-balancer-type: "nlb"
     service.beta.kubernetes.io/aws-load-balancer-ssl-cert: "<certificate ARN>"
     service.beta.kubernetes.io/aws-load-balancer-ssl-ports: "443"
     service.beta.kubernetes.io/aws-load-balancer-backend-protocol: "http"
spec:
  type: LoadBalancer
  ports:
  - port: 443
    targetPort: 80
    protocol: TCP
  selector:
    app: demo-app
//---
kind: Deployment
apiVersion: apps/v1
metadata:
  name: nginx
  namespace: default
  labels:
    app: demo-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: demo-app
  template:
    metadata:
      labels:
        app: demo-app
    spec:
      containers:
        - name: nginx
          image: nginx
          ports:
            - containerPort: 443
              protocol: TCP
            - containerPort: 80
              protocol: TCP
```

Di seguito sono riportati altri esempi di terminazione. SSL/TLS

- Proteggi l'ingresso di EKS con Contour e Let's Encrypt The Way GitOps

- Come faccio a terminare il traffico HTTPS sui carichi di lavoro Amazon EKS con ACM?

> **Important**
>
> Alcuni ingressi, come il controller AWS LB, implementano l' SSL/TLS utilizzo di Annotations anziché come parte delle specifiche Ingress.

ACM Private CA con cert-manager

Puoi abilitare TLS e mTLS per proteggere i carichi di lavoro delle applicazioni EKS all'ingresso, sul pod e tra i pod utilizzando ACM Private Certificate Authority (CA) e cert-manager, un popolare componente aggiuntivo di Kubernetes per distribuire, rinnovare e revocare i certificati. ACM Private CA è una CA gestita, sicura e altamente disponibile senza i costi iniziali e di manutenzione della gestione della propria CA. Se utilizzi l'autorità di certificazione Kubernetes predefinita, esiste l'opportunità di migliorare la sicurezza e soddisfare i requisiti di conformità con ACM Private CA. ACM Private CA protegge le chiavi private nei moduli di sicurezza hardware FIPS 140-2 di livello 3 (molto sicuri), rispetto alla CA predefinita che archivia le chiavi codificate in memoria (meno sicure). Una CA centralizzata offre inoltre un maggiore controllo e una migliore verificabilità dei certificati privati sia all'interno che all'esterno di un ambiente Kubernetes.

Modalità CA di breve durata per TLS reciproco tra carichi di lavoro

Quando si utilizza ACM Private CA for mTLS in EKS, si consiglia di utilizzare certificati di breve durata con modalità CA di breve durata. Sebbene sia possibile emettere certificati di breve durata in modalità CA generica, l'utilizzo della modalità CA di breve durata risulta più conveniente (circa il 75% in meno rispetto alla modalità generale) per i casi d'uso in cui è necessario emettere frequentemente nuovi certificati. Inoltre, dovresti provare ad allineare il periodo di validità dei certificati privati alla durata dei pod del tuo cluster EKS. Scopri di più su ACM Private CA e sui suoi vantaggi qui.

Istruzioni di configurazione ACM

Inizia creando una CA privata seguendo le procedure fornite nei documenti tecnici di ACM Private CA. Una volta che hai una CA privata, installa cert-manager utilizzando le normali istruzioni di installazione. Dopo aver installato cert-manager, installa il plugin Private CA Kubernetes cert-manager seguendo le istruzioni di configurazione riportate in. GitHub Il plug-in consente al cert-manager di richiedere certificati privati da ACM Private CA.

Ora che hai una CA privata e un cluster EKS con cert-manager e il plug-in installati, è il momento di impostare le autorizzazioni e creare l'emittente. Aggiorna le autorizzazioni IAM del ruolo del nodo EKS per consentire l'accesso a ACM Private CA. Sostituisci il valore <CA_ARN> con il valore della tua CA privata:

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "awspcaissuer",
            "Action": [
                "acm-pca:DescribeCertificateAuthority",
                "acm-pca:GetCertificate",
                "acm-pca:IssueCertificate"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:acm-pca:us-west-2:123456789012:certificate-
authority/12345678-1234-1234-1234-123456789012"
        }
    ]
}
```

È possibile utilizzare anche Service Roles for IAM Accounts o IRSA. Consulta la sezione Risorse aggiuntive di seguito per esempi completi.

Crea un emittente in Amazon EKS creando un file di definizione delle risorse personalizzato denominato cluster-issuer.yaml contenente il testo seguente, sostituendo <CA_ARN> le informazioni con la tua CA privata. <Region>

```
apiVersion: awspca.cert-manager.io/v1beta1
kind: AWSPCAClusterIssuer
metadata:
          name: demo-test-root-ca
spec:
          arn: <CA_ARN>
          region: <Region>
```

Implementa l'emittente che hai creato.

```
kubectl apply -f cluster-issuer.yaml
```

Il tuo cluster EKS è configurato per richiedere certificati da Private CA. Ora puoi utilizzare la `Certificate` risorsa di cert-manager per emettere certificati modificando i valori del `issuerRef` campo nell'emittente CA privato che hai creato sopra. Per maggiori dettagli su come specificare e richiedere le risorse relative ai certificati, consulta la guida alle risorse per i certificati di cert-manager. Vedi alcuni esempi qui.

ACM Private CA con Istio e cert-manager

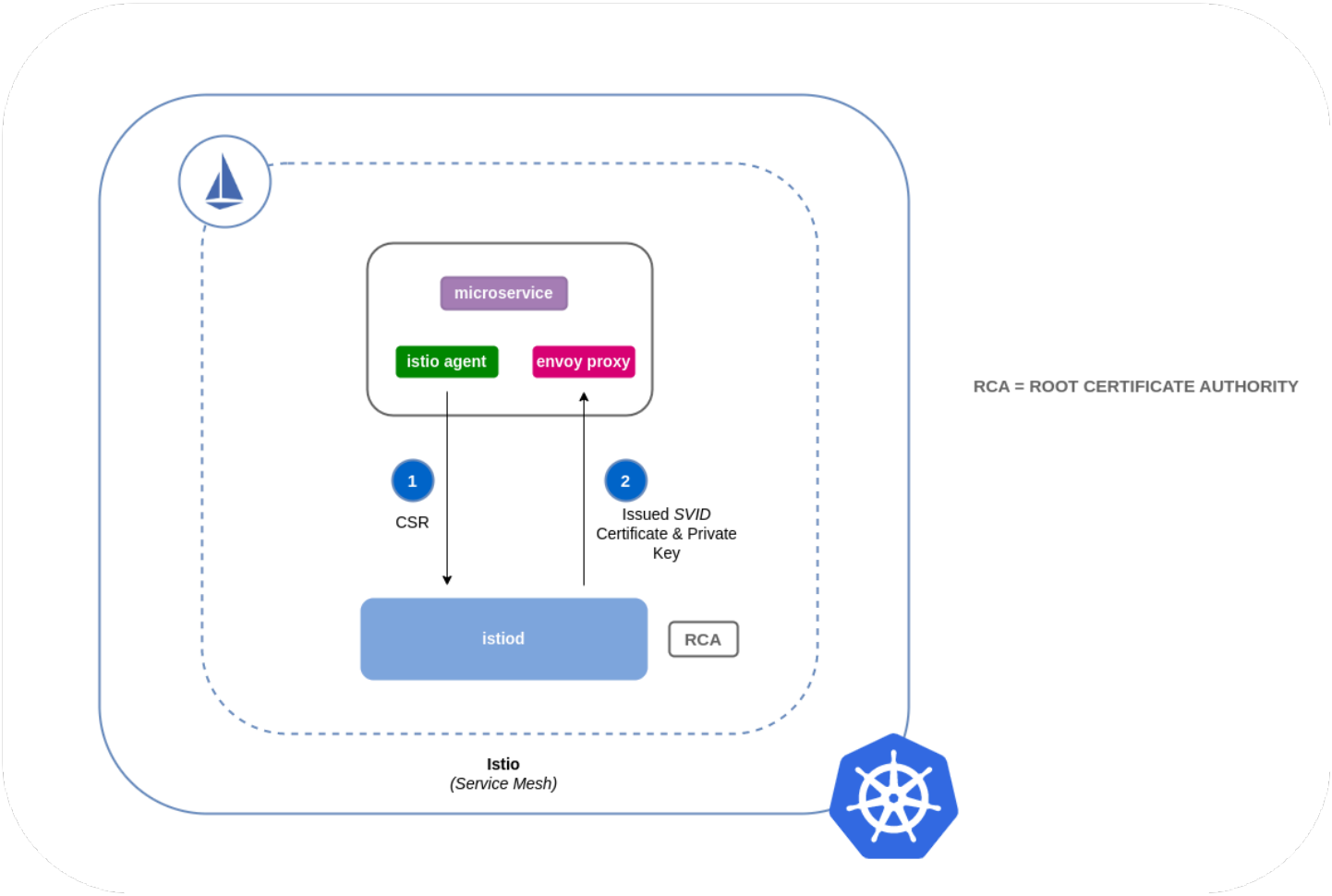
Se esegui Istio nel tuo cluster EKS, puoi disabilitare il piano di controllo Istio (in particolare`istiod`) dal funzionamento come Autorità di certificazione (CA) principale e configurare ACM Private CA come CA principale per MTL tra i carichi di lavoro. Se utilizzi questo approccio, prendi in considerazione l'utilizzo della modalità CA di breve durata in ACM Private CA. Per maggiori dettagli, consulta la sezione precedente e questo post del blog.

Come funziona la firma dei certificati in Istio (impostazione predefinita)

I carichi di lavoro in Kubernetes vengono identificati utilizzando gli account di servizio. Se non specifichi un account di servizio, Kubernetes ne assegnerà automaticamente uno al tuo carico di lavoro. Inoltre, gli account di servizio montano automaticamente un token associato. Questo token viene utilizzato dall'account di servizio per i carichi di lavoro per l'autenticazione con l'API Kubernetes. L'account di servizio può essere sufficiente come identità per Kubernetes, ma Istio dispone di un proprio sistema di gestione delle identità e di CA. Quando un carico di lavoro si avvia con il suo proxy sidecar Envoy, ha bisogno di un'identità assegnata da Istio per poter essere considerato affidabile e poter comunicare con gli altri servizi della rete mesh.

Per ottenere questa identità da Istio, `istio-agent` invia una richiesta nota come richiesta di firma del certificato (o CSR) al piano di controllo Istio. Questa CSR contiene il token dell'account del servizio in modo che l'identità del carico di lavoro possa essere verificata prima dell'elaborazione. Questo processo di verifica è gestito da`istiod`, che funge sia da Autorità di registrazione (o RA) che da CA. La RA funge da guardiano che assicura che solo la CSR verificata pervenga alla CA. Una volta verificata, la CSR verrà inoltrata alla CA che emetterà quindi un certificato contenente un'identità SPIFFE con l'account del servizio. Questo certificato è chiamato documento di identità verificabile SPIFFE (o SVID). Lo SVID viene assegnato al servizio richiedente per scopi di identificazione e per crittografare il traffico in transito tra i servizi di comunicazione.

Flusso predefinito per le richieste di firma dei certificati Istio:



Come funziona la firma dei certificati in Istio con ACM Private CA

Puoi utilizzare un componente aggiuntivo cert-manager chiamato Istio Certificate Signing Request agent (istio-csr) per integrare Istio con ACM Private CA. Questo agente consente di proteggere i carichi di lavoro Istio e i componenti del piano di controllo con gli emittenti dei gestori di certificati, in questo caso ACM Private CA. L'agente istio-csr espone lo stesso servizio fornito da istiod nella configurazione predefinita di convalida delle entrate. CSRs Tuttavia, dopo la verifica, convertirà le richieste in risorse supportate da cert manager (ad esempio integrazioni con emittenti CA esterni).

Ogni volta che viene generata una CSR da un carico di lavoro, questa verrà inoltrata a istio-csr, che richiederà i certificati da ACM Private CA. Questa comunicazione tra istio-csr e ACM Private CA è abilitata dal plugin AWS Private CA issuer. Cert manager utilizza questo plugin per richiedere certificati TLS da ACM Private CA. Il plug-in emittente comunicherà con il servizio ACM Private CA per richiedere un certificato firmato per il carico di lavoro. Una volta firmato, il certificato verrà restituito a istio-csr, che leggerà la richiesta firmata e la restituirà al carico di lavoro che ha avviato la CSR.

Richieste di firma dei certificati Flow for Istio con istio-csr

image: istio-csr-with-acm -private-ca.png [Flow per le richieste di firma dei certificati Istio con istio-csr]

Istruzioni di configurazione di Istio con CA privata

1. Inizia seguendo le stesse istruzioni di configurazione in questa sezione per completare quanto segue:
2. Creare una CA privata
3. Installa cert-manager
4. Installa il plugin dell'emittente
5. Imposta le autorizzazioni e crea un emittente. L'emittente rappresenta la CA e viene utilizzato per firmare `istiod` e integrare i certificati del carico di lavoro. Comunicherà con ACM Private CA.
6. Crea un `istio-system` namespace. È qui che verranno distribuite le `istiod certificate` e altre risorse Istio.
7. Installa Istio CSR configurato con AWS Private CA Issuer Plugin. Puoi conservare le richieste di firma dei certificati per i carichi di lavoro per verificare che vengano approvate e firmate (). `preserveCertificateRequests=true`

```
helm install -n cert-manager cert-manager-istio-csr jetstack/cert-manager-istio-csr \
--set "app.certmanager.issuer.group=awspca.cert-manager.io" \
--set "app.certmanager.issuer.kind=AWSPCAClusterIssuer" \
--set "app.certmanager.issuer.name=<the-name-of-the-issuer-you-created>" \
--set "app.certmanager.preserveCertificateRequests=true" \
--set "app.server.maxCertificateDuration=48h" \
--set "app.tls.certificateDuration=24h" \
--set "app.tls.istiodCertificateDuration=24h" \
--set "app.tls.rootCAFile=/var/run/secrets/istio-csr/ca.pem" \
--set "volumeMounts[0].name=root-ca" \
--set "volumeMounts[0].mountPath=/var/run/secrets/istio-csr" \
--set "volumes[0].name=root-ca" \
--set "volumes[0].secret.secretName=istio-root-ca"
```

8. Installa Istio con configurazioni personalizzate da sostituire `istiod` con `cert-manager istio-csr` come fornitore di certificati per la mesh. Questo processo può essere eseguito utilizzando l'operatore Istio.

```
apiVersion: install.istio.io/v1alpha1
kind: IstioOperator
```

```
metadata:
  name: istio
  namespace: istio-system
spec:
  profile: "demo"
  hub: gcr.io/istio-release
  values:
  global:
    # Change certificate provider to cert-manager istio agent for istio agent
    caAddress: cert-manager-istio-csr.cert-manager.svc:443
  components:
    pilot:
      k8s:
        env:
          # Disable istiod CA Sever functionality
        - name: ENABLE_CA_SERVER
          value: "false"
        overlays:
        - apiVersion: apps/v1
          kind: Deployment
          name: istiod
          patches:

            # Mount istiod serving and webhook certificate from Secret mount
          - path: spec.template.spec.containers.[name:discovery].args[7]
            value: "--tlsCertFile=/etc/cert-manager/tls/tls.crt"
          - path: spec.template.spec.containers.[name:discovery].args[8]
            value: "--tlsKeyFile=/etc/cert-manager/tls/tls.key"
          - path: spec.template.spec.containers.[name:discovery].args[9]
            value: "--caCertFile=/etc/cert-manager/ca/root-cert.pem"

          - path: spec.template.spec.containers.[name:discovery].volumeMounts[6]
            value:
              name: cert-manager
              mountPath: "/etc/cert-manager/tls"
              readOnly: true
          - path: spec.template.spec.containers.[name:discovery].volumeMounts[7]
            value:
              name: ca-root-cert
              mountPath: "/etc/cert-manager/ca"
              readOnly: true

          - path: spec.template.spec.volumes[6]
            value:
```

```
            name: cert-manager
            secret:
              secretName: istiod-tls
        - path: spec.template.spec.volumes[7]
          value:
            name: ca-root-cert
            configMap:
              defaultMode: 420
              name: istio-ca-root-cert
```

9. Distribuisci la risorsa personalizzata sopra riportata che hai creato.

```
istioctl operator init
kubectl apply -f istio-custom-config.yaml
```

10. Ora puoi distribuire un carico di lavoro nella rete mesh del tuo cluster EKS e applicare gli MTL.

Richieste di firma dei certificati Istio

image: istio-csr-requests .png [richieste di firma del certificato Istio]

Strumenti e risorse

- Workshop di immersione sulla sicurezza di Amazon EKS - Sicurezza di rete
- Come implementare cert-manager e il plug-in ACM Private CA per abilitare TLS in EKS.
- Configurazione della crittografia end-to-end TLS su Amazon EKS con il nuovo AWS Load Balancer Controller e ACM Private CA.
- Plugin cert-manager privato CA Kubernetes attivo. GitHub
- Guida per l'utente del plug-in cert-manager di CA Kubernetes privato.
- Come utilizzare la modalità di certificazione di breve durata di AWS Private Certificate Authority
- egress-operator Un operatore e un plug-in DNS per controllare il traffico in uscita dal cluster senza ispezione del protocollo
- NeuVector di SUSE, piattaforma open source di sicurezza per container zero-trust, fornisce policy, regole di rete, prevenzione della perdita di dati (DLP), web application firewall (WAF) e firme relative alle minacce di rete.

Crittografia dei dati e gestione dei segreti

Crittografia dei dati a riposo

Esistono tre diverse opzioni di storage AWS native che puoi utilizzare con Kubernetes: EBS, EFS e for Lustre. FSx Tutti e tre offrono la crittografia di dati a riposo utilizzando una chiave gestita dal servizio o una chiave cliente principale (CMK). Per EBS puoi utilizzare il driver di storage in-tree o il driver EBS CSI. Entrambi includono parametri per la crittografia dei volumi e la fornitura di un CMK. Per EFS, è possibile utilizzare il driver EFS CSI, tuttavia, a differenza di EBS, il driver EFS CSI non supporta il provisioning dinamico. Se desideri utilizzare EFS con EKS, dovrai fornire e configurare la crittografia a riposo per il file system prima di creare un PV. Per ulteriori informazioni sulla crittografia dei file EFS, consulta Encrypting Data at Rest. Oltre a offrire la crittografia at-rest, EFS e FSx for Lustre includono un'opzione per crittografare i dati in transito. FSx for Lustre lo fa per impostazione predefinita. Per EFS, puoi aggiungere la crittografia di trasporto aggiungendo il `tls` parametro a `mountOptions` nel tuo PV come in questo esempio:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: efs-pv
spec:
  capacity:
    storage: 5Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: efs-sc
  mountOptions:
    - tls
  csi:
    driver: efs.csi.aws.com
    volumeHandle: <file_system_id>
```

Il driver FSx CSI supporta il provisioning dinamico dei file system Lustre. Per impostazione predefinita, crittografa i dati con una chiave gestita dal servizio, sebbene sia disponibile un'opzione per fornire la propria CMK, come in questo esempio:

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
```

```
metadata:
  name: fsx-sc
provisioner: fsx.csi.aws.com
parameters:
  subnetId: subnet-056da83524edbe641
  securityGroupIds: sg-086f61ea73388fb6b
  deploymentType: PERSISTENT_1
  kmsKeyId: <kms_arn>
```

> **Important**
>
> A partire dal 28 maggio 2020, tutti i dati scritti nel volume temporaneo dei pod EKS Fargate sono crittografati per impostazione predefinita utilizzando un algoritmo crittografico AES-256 standard del settore. Non sono necessarie modifiche all'applicazione poiché la crittografia e la decrittografia vengono gestite senza problemi dal servizio.

Crittografa i dati inattivi

La crittografia dei dati inattivi è considerata una best practice. Se non sei sicuro che la crittografia sia necessaria, crittografa i tuoi dati.

Ruota periodicamente il CMKs

Configura KMS per ruotare automaticamente il tuo. CMKs In questo modo le chiavi verranno ruotate una volta all'anno, salvando quelle vecchie a tempo indeterminato, in modo che i dati possano ancora essere decifrati. Per ulteriori informazioni, consulta Rotazione delle chiavi master del cliente

Usa i punti di accesso EFS per semplificare l'accesso ai set di dati condivisi

Se disponi di set di dati condivisi con permessi di file POSIX diversi o desideri limitare l'accesso a parte del file system condiviso creando punti di montaggio diversi, prendi in considerazione l'utilizzo di punti di accesso EFS. Per ulteriori informazioni sull'utilizzo dei punti di accesso, consulta -access-points.html. https://docs.aws.amazon.com/efs/ latest/ug/efs Oggi, se si desidera utilizzare un punto di accesso (AP), è necessario fare riferimento all'AP nel `volumeHandle` parametro del PV.

> **Important**
>
> A partire dal 23 marzo 2021, il driver EFS CSI supporta il provisioning dinamico degli Access Point EFS. Gli access point sono punti di ingresso specifici dell'applicazione in un file system

EFS che semplificano la condivisione di un file system tra più pod. Ogni file system EFS può avere fino a 120 file system PVs. Per ulteriori informazioni, consulta Introduzione al provisioning dinamico CSI di Amazon EFS.

Gestione dei segreti

I segreti di Kubernetes vengono utilizzati per archiviare informazioni sensibili, come certificati utente, password o chiavi API. Vengono mantenute in etcd come stringhe codificate in base64. Su EKS, i volumi EBS per i nodi etcd sono crittografati con la crittografia EBS. Un pod può recuperare un oggetto segreto di Kubernetes facendo riferimento al segreto in. `podSpec` Questi segreti possono essere mappati su una variabile di ambiente o montati come volume. Per ulteriori informazioni sulla creazione di segreti, vedere https://kubernetes. io/docs/concepts/configuration/secret/.

Warning

I segreti in un particolare namespace possono essere referenziati da tutti i pod nello spazio dei nomi del segreto.

Warning

L'autorizzatore del nodo consente a Kubelet di leggere tutti i segreti montati sul nodo.

Usa AWS KMS per la crittografia in busta dei segreti di Kubernetes

Ciò ti consente di crittografare i tuoi segreti con una chiave di crittografia dei dati unica (DEK). Il DEK viene quindi crittografato utilizzando una chiave di crittografia (KEK) di AWS KMS che può essere ruotata automaticamente in base a una pianificazione ricorrente. Con il plug-in KMS per Kubernetes, tutti i segreti di Kubernetes vengono archiviati in etcd in testo cifrato anziché in testo semplice e possono essere decrittografati solo dal server API Kubernetes. Per ulteriori dettagli, consulta Utilizzo del supporto del provider di crittografia EKS per una difesa approfondita

Verifica l'uso di Kubernetes Secrets

Su EKS, attiva la registrazione degli audit e crea un filtro CloudWatch metrico e un allarme per avvisarti quando viene utilizzato un segreto (opzionale). Di seguito è riportato un

esempio di filtro metrico per il registro di controllo di Kubernetes,. `{($.verb="get") && ($.objectRef.resource="secret")}` Puoi anche utilizzare le seguenti query con Log Insights: CloudWatch

```
fields @timestamp, @message
| sort @timestamp desc
| limit 100
| stats count(*) by objectRef.name as secret
| filter verb="get" and objectRef.resource="secrets"
```

La query precedente mostrerà il numero di volte in cui è stato effettuato l'accesso a un segreto entro un periodo di tempo specifico.

```
fields @timestamp, @message
| sort @timestamp desc
| limit 100
| filter verb="get" and objectRef.resource="secrets"
| display objectRef.namespace, objectRef.name, user.username, responseStatus.code
```

Questa query mostrerà il segreto, insieme allo spazio dei nomi e al nome utente dell'utente che ha tentato di accedere al segreto e al codice di risposta.

Ruota periodicamente i tuoi segreti

Kubernetes non ruota automaticamente i segreti. Se devi ruotare i segreti, prendi in considerazione l'utilizzo di un archivio segreto esterno, ad esempio Vault o AWS Secrets Manager.

Usa namespace separati per isolare i segreti da diverse applicazioni

Se disponi di segreti che non possono essere condivisi tra le applicazioni in uno spazio dei nomi, crea uno spazio dei nomi separato per tali applicazioni.

Utilizzate i montaggi di volume anziché le variabili di ambiente

I valori delle variabili di ambiente possono apparire involontariamente nei log. I segreti montati come volumi vengono istanziati come volumi tmpfs (un file system basato su RAM) che vengono rimossi automaticamente dal nodo quando il pod viene eliminato.

Utilizza un provider di segreti esterno

Esistono diverse alternative valide all'utilizzo dei segreti di Kubernetes, tra cui AWS Secrets Manager e Hashicorp's Vault. Questi servizi offrono funzionalità come controlli granulari degli accessi, crittografia avanzata e rotazione automatica dei segreti che non sono disponibili con Kubernetes Secrets. Sealed Secrets di Bitnami è un altro approccio che utilizza la crittografia asimmetrica per creare «segreti sigillati». Una chiave pubblica viene utilizzata per crittografare il segreto, mentre la chiave privata utilizzata per decrittografare il segreto viene conservata all'interno del cluster, consentendoti di archiviare in modo sicuro i segreti sigillati in sistemi di controllo del codice sorgente come Git. Per ulteriori informazioni, consulta Gestione della distribuzione dei segreti in Kubernetes utilizzando Sealed Secrets.

Con l'aumento dell'uso di archivi segreti esterni, è cresciuta anche la necessità di integrarli con Kubernetes. Secret Store CSI Driver è un progetto comunitario che utilizza il modello di driver CSI per recuperare segreti da archivi segreti esterni. Attualmente, il driver supporta AWS Secrets Manager, Azure, Vault e GCP. Il provider AWS supporta sia AWS Secrets Manager che AWS Parameter Store. Può anche essere configurato per ruotare i segreti quando scadono e sincronizzare i segreti di AWS Secrets Manager con Kubernetes Secrets. La sincronizzazione dei segreti può essere utile quando è necessario fare riferimento a un segreto come variabile di ambiente anziché leggerlo da un volume.

> **Note**
>
> Quando il driver CSI dell'archivio segreto deve recuperare un segreto, assume il ruolo IRSA assegnato al pod che fa riferimento a un segreto. Il codice per questa operazione può essere trovato qui.

Per ulteriori informazioni su AWS Secrets & Configuration Provider (ASCP), fai riferimento alle seguenti risorse:

- Come utilizzare AWS Secrets Configuration Provider con il driver CSI Kubernetes Secret Store
- Integrazione dei segreti di Secrets Manager con il driver CSI Kubernetes Secrets Store

external-secrets è un altro modo per utilizzare un archivio segreto esterno con Kubernetes. Come il driver CSI, external-secrets funziona su una varietà di backend diversi, incluso AWS Secrets Manager. La differenza è che, anziché recuperare i segreti dall'archivio segreto esterno, external-secrets copia i segreti da questi backend su Kubernetes come segreti. Ciò ti consente di gestire

i segreti utilizzando il tuo archivio segreto preferito e di interagire con i segreti in modo nativo di Kubernetes.

Strumenti e risorse

- Workshop Amazon EKS Security Immersion - Crittografia dei dati e gestione dei segreti

Sicurezza del runtime

La sicurezza in fase di esecuzione offre una protezione attiva per i container mentre sono in esecuzione. L'idea è quella di rilevare e and/or impedire che si verifichino attività dannose all'interno del contenitore. Ciò può essere ottenuto con una serie di meccanismi presenti nel kernel Linux o nelle estensioni del kernel integrate con Kubernetes, come le funzionalità di Linux, l'elaborazione sicura (seccomp) o. AppArmor SELinux Esistono anche opzioni come Amazon GuardDuty e strumenti di terze parti che possono aiutare a stabilire linee di base e rilevare attività anomale con una configurazione meno manuale dei meccanismi del kernel Linux.

> **Important**
>
> Al momento Kubernetes non fornisce alcun meccanismo nativo per caricare seccomp o profili su Nodes. AppArmor SELinux Devono essere caricati manualmente o installati sui nodi quando vengono avviati. Questa operazione deve essere eseguita prima di farvi riferimento nei Pod, poiché lo scheduler non è a conoscenza di quali nodi abbiano dei profili. Scopri di seguito come strumenti come Security Profiles Operator possono aiutare ad automatizzare il provisioning dei profili sui nodi.

Contesti di sicurezza e controlli Kubernetes integrati

Molti meccanismi di sicurezza del runtime di Linux sono strettamente integrati con Kubernetes e possono essere configurati tramite contesti di sicurezza Kubernetes. Una di queste opzioni è il `privileged` flag, che è di default e, se abilitato, è `false` essenzialmente equivalente a root sull'host. È quasi sempre inappropriato abilitare la modalità privilegiata nei carichi di lavoro di produzione, ma esistono molti altri controlli che possono fornire privilegi più granulari ai contenitori, a seconda dei casi.

Funzionalità Linux

Le funzionalità di Linux consentono di concedere determinate funzionalità a un Pod o a un contenitore senza fornire tutte le capacità dell'utente root. Gli esempi includonoCAP_NET_ADMIN, che consente di configurare interfacce di rete o firewall o che consente la manipolazione dell'orologio di sistema. CAP_SYS_TIME

Seccomp

Con secure computing (seccomp) è possibile impedire a un'applicazione containerizzata di effettuare determinate chiamate di sistema al kernel del sistema operativo host sottostante. Sebbene il sistema operativo Linux abbia poche centinaia di chiamate di sistema, la maggior parte di esse non è necessaria per l'esecuzione dei container. Limitando le chiamate di sistema che un container può effettuare, è possibile ridurre efficacemente la superficie di attacco dell'applicazione.

Seccomp funziona intercettando le chiamate di sistema e permettendo il passaggio solo a quelle che sono state consentite nell'elenco. Docker ha un profilo seccomp predefinito adatto alla maggior parte dei carichi di lavoro generici e altri runtime di container come containerd forniscono impostazioni predefinite comparabili. Puoi configurare il tuo contenitore o Pod per utilizzare il profilo seccomp predefinito del runtime del contenitore aggiungendo quanto segue alla sezione delle specifiche del Pod: securityContext

```
securityContext:
  seccompProfile:
    type: RuntimeDefault
```

A partire dalla 1.22 (in alpha, stabile a partire dalla 1.27), quanto sopra RuntimeDefault può essere utilizzato per tutti i Pod su un nodo utilizzando un singolo flag kubelet,. --seccomp-default Quindi il profilo specificato in securityContext è necessario solo per altri profili.

È anche possibile creare profili personalizzati per cose che richiedono privilegi aggiuntivi. Questa operazione può essere molto noiosa da eseguire manualmente, ma esistono strumenti come Inspektor Gadget (consigliato anche nella sezione sulla sicurezza di rete per la generazione di politiche di rete) e Security Profiles Operator che supportano l'utilizzo di strumenti come eBPF o log per registrare i requisiti di privilegio di base come profili seccomp. Security Profiles Operator consente inoltre di automatizzare l'implementazione dei profili registrati nei nodi per l'utilizzo da parte di pod e contenitori.

AppArmor e SELinux

AppArmor e SELinux sono noti come sistemi di controllo degli accessi obbligatori o MAC. Sono concettualmente simili a seccomp ma con funzionalità diverse APIs , che consentono il controllo degli accessi, ad esempio, per percorsi di file system o porte di rete specifici. Il supporto per questi strumenti dipende dalla distribuzione Linux, con Debian/Ubuntu supporto AppArmor e supporto per RHEL/CentOS/Bottlerocket/Amazon Linux 2023 SELinux. Vedi anche la sezione sulla sicurezza dell'infrastruttura per ulteriori discussioni su SELinux.

Entrambi AppArmor SELinux sono integrati con Kubernetes, ma a partire da Kubernetes 1.28 i AppArmor profili devono essere specificati tramite annotazioni, mentre le SELinux etichette possono essere impostate direttamente tramite il SELinux campo Opzioni nel contesto di sicurezza.

Come per i profili seccomp, il Security Profiles Operator sopra menzionato può fornire assistenza nella distribuzione dei profili sui nodi del cluster. (In futuro, il progetto mira anche a generare profili per AppArmor e SELinux come fa per seccomp.)

Raccomandazioni

Usa Amazon GuardDuty per il monitoraggio del runtime e il rilevamento delle minacce ai tuoi ambienti EKS

Se al momento non disponi di una soluzione per il monitoraggio continuo dei runtime EKS, l'analisi dei log di controllo EKS e la scansione alla ricerca di malware e altre attività sospette, Amazon consiglia vivamente l'uso di Amazon ai clienti che desiderano un modo semplice, veloce, sicuro, scalabile ed economico GuardDuty per proteggere i propri ambienti AWS con un solo clic. Amazon GuardDuty è un servizio di monitoraggio della sicurezza che analizza ed elabora fonti di dati fondamentali, come eventi di CloudTrail gestione AWS, registri CloudTrail eventi AWS, log di flusso VPC (da istanze Amazon EC2), log di audit Kubernetes e log DNS. Include anche il monitoraggio del runtime EKS. Utilizza feed di intelligence sulle minacce costantemente aggiornati, come elenchi di indirizzi IP e domini dannosi, e l'apprendimento automatico per identificare attività impreviste, potenzialmente non autorizzate e dannose all'interno del tuo ambiente AWS. Ciò può includere problemi come l'aumento dei privilegi, l'uso di credenziali esposte o la comunicazione con indirizzi IP e domini dannosi, la presenza di malware sulle istanze Amazon EC2 e sui carichi di lavoro dei container EKS o l'individuazione di attività API sospette. GuardDuty ti informa sullo stato del tuo ambiente AWS producendo risultati di sicurezza che puoi visualizzare nella GuardDuty console o tramite Amazon EventBridge. GuardDuty fornisce inoltre supporto per esportare i risultati in un

bucket Amazon Simple Storage Service (S3) e integrarli con altri servizi come AWS Security Hub and Detective.

Guarda questo talk tecnico online di AWS «Enhanced threat detection for Amazon EKS with Amazon GuardDuty - AWS Online Tech Talks» per scoprire come abilitare queste funzionalità di sicurezza EKS aggiuntive step-by-step in pochi minuti.

Facoltativamente: utilizza una soluzione di terze parti per il monitoraggio del runtime

Creare e gestire i profili seccomp e Apparmor può essere difficile se non hai familiarità con la sicurezza di Linux. Se non avete il tempo di diventare esperti, prendete in considerazione l'utilizzo di una soluzione commerciale di terze parti. Molti di loro sono andati oltre i profili statici come Apparmor e seccomp e hanno iniziato a utilizzare l'apprendimento automatico per bloccare o avvisare di attività sospette. Alcune di queste soluzioni sono disponibili di seguito nella sezione strumenti. Altre opzioni sono disponibili su AWS Marketplace for Containers.

Prendi in considerazione le funzionalità di add/dropping Linux prima di scrivere policy seccomp

Le funzionalità implicano vari controlli nelle funzioni del kernel raggiungibili tramite syscalls.
Se il controllo fallisce, il syscall restituisce in genere un errore. Il controllo può essere eseguito direttamente all'inizio di una specifica syscall o più in profondità nel kernel in aree che potrebbero essere raggiungibili tramite più syscall diverse (come la scrittura su uno specifico file privilegiato). Seccomp, d'altra parte, è un filtro syscall che viene applicato a tutte le syscall prima che vengano eseguite. Un processo può impostare un filtro che consenta loro di revocare il diritto di eseguire determinate syscall o argomenti specifici per determinate syscall.

Prima di usare seccomp, valuta se le funzionalità di adding/removing Linux ti offrono il controllo di cui hai bisogno. Per ulteriori informazioni, vedere Impostazione delle funzionalità per- containers.

Scopri se puoi raggiungere i tuoi obiettivi utilizzando Pod Security Policies () PSPs

Le politiche di sicurezza di Pod offrono molti modi diversi per migliorare il livello di sicurezza senza introdurre complessità eccessive. Esplora le opzioni disponibili PSPs prima di avventurarti nella creazione di profili seccomp e Apparmor.

> **Warning**
>
> A partire da Kubernetes 1.25, PSPs sono stati rimossi e sostituiti con il controller Pod Security Admission. Le alternative di terze parti esistenti includono e Kyverno. OPA/Gatekeeper Una

raccolta di vincoli e modelli di vincoli di Gatekeeper per l'implementazione delle politiche che si trovano comunemente in PSPs può essere recuperata dal repository della libreria Gatekeeper in poi. GitHub Inoltre, nella libreria delle politiche di Kyverno sono disponibili molti sostituti di, inclusa la raccolta completa di Pod Security Standards. PSPs

Strumenti e risorse

- 7 cose che dovresti sapere prima di iniziare
- AppArmorCaricatore
- Configurazione di nodi con profili
- Security Profiles Operator è un miglioramento di Kubernetes che mira a semplificare l'utilizzo da parte degli utenti SELinux, in seccomp e nei cluster Kubernetes. AppArmor Fornisce funzionalità sia per la generazione di profili dall'esecuzione di carichi di lavoro sia per il caricamento di profili sui nodi Kubernetes da utilizzare nei Pod.
- Inspektor Gadget consente di ispezionare, tracciare e profilare molti aspetti del comportamento di runtime su Kubernetes, inclusa l'assistenza nella generazione di profili seccomp.
- Aqua
- Qualys
- impilatore
- Sysdig Secure
- Prisma
- NeuVector di SUSE, piattaforma open source per la sicurezza dei container zero-trust, fornisce regole di profilo di processo e regole di accesso ai file.

Protezione dell'infrastruttura (host)

Per quanto sia importante proteggere le immagini dei container, è altrettanto importante salvaguardare l'infrastruttura che le gestisce. Questa sezione esplora diversi modi per mitigare i rischi derivanti dagli attacchi lanciati direttamente contro l'host. Queste linee guida devono essere utilizzate insieme a quelle descritte nella sezione Runtime Security.

Raccomandazioni

Utilizza un sistema operativo ottimizzato per l'esecuzione di contenitori

Prendi in considerazione l'utilizzo di Flatcar Linux, Project Atomic, RancherOS e Bottlerocket, un sistema operativo speciale di AWS progettato per l'esecuzione di container Linux. Include una superficie di attacco ridotta, un'immagine del disco verificata all'avvio e l'utilizzo di limiti di autorizzazione impostati. SELinux

In alternativa, utilizza l'AMI ottimizzata EKS per i nodi di lavoro Kubernetes. L'AMI ottimizzata per EKS viene rilasciata regolarmente e contiene un set minimo di pacchetti e binari del sistema operativo necessari per eseguire i carichi di lavoro containerizzati.

Consulta la specifica di build Amazon EKS AMI RHEL per uno script di configurazione di esempio che può essere usato per creare un'AMI Amazon EKS personalizzata in esecuzione su Red Hat Enterprise Linux utilizzando Hashicorp Packer. Questo script può essere ulteriormente sfruttato per creare EKS custom conformi a STIG. AMIs

Mantieni aggiornato il tuo sistema operativo Worker Node

Indipendentemente dal fatto che utilizzi un sistema operativo host ottimizzato per contenitori come Bottlerocket o un Amazon Machine Image più grande, ma comunque minimalista, come EKS ottimizzato AMIs, è consigliabile mantenere queste immagini del sistema operativo host aggiornate con le ultime patch di sicurezza.

Per l'ottimizzazione per EKS AMIs, controlla regolarmente il canale CHANGELOG e/o le note di rilascio e automatizza l'implementazione di immagini aggiornate dei nodi di lavoro nel tuo cluster.

Trattate la vostra infrastruttura come immutabile e automatizzate la sostituzione dei nodi di lavoro

Anziché eseguire aggiornamenti sul posto, sostituisci i tuoi dipendenti quando diventa disponibile una nuova patch o un aggiornamento. Questo può essere affrontato in un paio di modi. Puoi aggiungere istanze a un gruppo di scalabilità automatica esistente utilizzando l'AMI più recente mentre cordoni e dreni i nodi in sequenza fino a quando tutti i nodi del gruppo non siano stati sostituiti con l'AMI più recente. In alternativa, puoi aggiungere istanze a un nuovo gruppo di nodi mentre cordoni e svuoti in sequenza i nodi del vecchio gruppo di nodi fino alla sostituzione di tutti i nodi. I gruppi di nodi gestiti da EKS utilizzano il primo approccio e visualizzeranno un messaggio nella console per

aggiornare i lavoratori quando sarà disponibile una nuova AMI. `eksctl`ha anche un meccanismo per creare gruppi di nodi con l'AMI più recente e per isolare e drenare con eleganza i pod dai gruppi di nodi prima che le istanze vengano terminate. Se si decide di utilizzare un metodo diverso per sostituire i nodi di lavoro, si consiglia vivamente di automatizzare il processo per ridurre al minimo la supervisione umana, poiché probabilmente sarà necessario sostituire i lavoratori regolarmente man mano che ne vengono rilasciati di nuovi updates/patches e quando il piano di controllo viene aggiornato.

Con EKS Fargate, AWS aggiornerà automaticamente l'infrastruttura sottostante non appena gli aggiornamenti saranno disponibili. Spesso questa operazione può essere eseguita senza problemi, ma a volte un aggiornamento può causare la riprogrammazione del pod. Pertanto, ti consigliamo di creare distribuzioni con più repliche quando esegui l'applicazione come pod Fargate.

Esegui periodicamente kube-bench per verificare la conformità ai benchmark CIS per Kubernetes

kube-bench è un progetto open source di Aqua che valuta il cluster rispetto ai benchmark CIS per Kubernetes. Il benchmark descrive le migliori pratiche per proteggere i cluster Kubernetes non gestiti. Il benchmark CIS Kubernetes comprende il piano di controllo e il piano dati. Poiché Amazon EKS fornisce un piano di controllo completamente gestito, non tutti i consigli del benchmark CIS Kubernetes sono applicabili. Per garantire che questo ambito rifletta il modo in cui viene implementato Amazon EKS, AWS ha creato il benchmark CIS Amazon EKS. Il benchmark EKS eredita da CIS Kubernetes Benchmark con input aggiuntivi dalla community con considerazioni specifiche sulla configurazione per i cluster EKS.

Quando esegui kube-bench su un cluster EKS, segui queste istruzioni di Aqua Security. Per ulteriori informazioni, consulta Introduzione al benchmark CIS Amazon EKS.

Riduci al minimo l'accesso ai nodi di lavoro

Invece di abilitare l'accesso SSH, usa SSM Session Manager quando devi accedere in remoto a un host. A differenza delle chiavi SSH che possono essere perse, copiate o condivise, Session Manager consente di controllare l'accesso alle istanze EC2 utilizzando IAM. Inoltre, fornisce un audit trail e un registro dei comandi eseguiti sull'istanza.

A partire dal 19 agosto 2020, i gruppi di nodi gestiti supportano AMI personalizzate e modelli di lancio EC2. Ciò consente di incorporare l'agente SSM nell'AMI o installarlo mentre il nodo di lavoro viene avviato. Se preferisci non modificare l'AMI ottimizzata o il modello di avvio di ASG, puoi installare l'agente SSM con un DaemonSet as in questo esempio.

Policy IAM minima per l'accesso SSH basato su SSM

La policy gestita da `AmazonSSMManagedInstanceCore` AWS contiene una serie di autorizzazioni che non sono necessarie per SSM Session Manager/SSM RunCommand se stai solo cercando di evitare l'accesso SSH. In particolare, sono preoccupanti `*` le autorizzazioni `ssm:GetParameter(s)` che consentirebbero al ruolo di accedere a tutti i parametri in Parameter Store (inclusa SecureStrings la chiave AWS managed KMS configurata).

La seguente policy IAM contiene il set minimo di autorizzazioni per abilitare l'accesso ai nodi tramite SSM Systems Manager.

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "EnableAccessViaSSMSessionManager",
      "Effect": "Allow",
      "Action": [
        "ssmmessages:OpenDataChannel",
        "ssmmessages:OpenControlChannel",
        "ssmmessages:CreateDataChannel",
        "ssmmessages:CreateControlChannel",
        "ssm:UpdateInstanceInformation"
      ],
      "Resource": "*"
    },
    {
      "Sid": "EnableSSMRunCommand",
      "Effect": "Allow",
      "Action": [
        "ssm:UpdateInstanceInformation",
        "ec2messages:SendReply",
        "ec2messages:GetMessages",
        "ec2messages:GetEndpoint",
        "ec2messages:FailMessage",
        "ec2messages:DeleteMessage",
        "ec2messages:AcknowledgeMessage"
      ],
      "Resource": "*"
    }
  ]
}
```

Una volta implementata questa politica e installato il plug-in Session Manager, è possibile eseguire

```
aws ssm start-session --target [INSTANCE_ID_OF_EKS_NODE]
```

per accedere al nodo.

> **Note**
>
> Potresti anche prendere in considerazione l'aggiunta di autorizzazioni per abilitare la registrazione di Session Manager.

Distribuisci i lavoratori su sottoreti private

Impiegando i lavoratori su sottoreti private, si riduce al minimo la loro esposizione a Internet, dove spesso hanno origine gli attacchi. A partire dal 22 aprile 2020, l'assegnazione di indirizzi IP pubblici ai nodi di un gruppo di nodi gestiti sarà controllata dalla sottorete in cui vengono distribuiti. In precedenza, ai nodi di un gruppo di nodi gestiti veniva assegnato automaticamente un IP pubblico. Se scegli di distribuire i nodi di lavoro su sottoreti pubbliche, implementa regole restrittive dei gruppi di sicurezza AWS per limitarne l'esposizione.

Esegui Amazon Inspector per valutare gli host in termini di esposizione, vulnerabilità e deviazioni dalle best practice

Puoi utilizzare Amazon Inspector per verificare eventuali accessi di rete non intenzionali ai tuoi nodi e le vulnerabilità nelle istanze Amazon EC2 sottostanti.

Amazon Inspector può fornire dati CVE (Common Vulnerabilities and Exposures) per le istanze Amazon EC2 solo se l'agente Amazon EC2 Systems Manager (SSM) è installato e abilitato. Questo agente è preinstallato su diverse Amazon Machine Images (AMIs), incluso Amazon Linux AMIs ottimizzato per EKS. Indipendentemente dallo stato dell'agente SSM, tutte le istanze Amazon EC2 vengono analizzate per individuare eventuali problemi di raggiungibilità della rete. Per ulteriori informazioni sulla configurazione delle scansioni per Amazon EC2, consulta Scansione delle istanze Amazon EC2.

> **Important**
>
> Inspector non può essere eseguito sull'infrastruttura utilizzata per eseguire i pod Fargate.

Alternative

Corri SELinux

> **Note**
>
> Disponibile su Red Hat Enterprise Linux (RHEL), CentOS, Bottlerocket e Amazon Linux 2023

SELinux fornisce un ulteriore livello di sicurezza per mantenere i container isolati l'uno dall'altro e dall'host. SELinux consente agli amministratori di applicare i controlli di accesso obbligatori (MAC) per ogni utente, applicazione, processo e file. Immaginatelo come un backstop che limita le operazioni che possono essere eseguite a risorse specifiche sulla base di una serie di etichette. Su EKS, SELinux può essere usato per impedire ai container di accedere alle rispettive risorse.

Le SELinux politiche relative ai container sono definite nel pacchetto container-selinux. Docker CE richiede questo pacchetto (insieme alle sue dipendenze) in modo che i processi e i file creati da Docker (o da altri runtime dei container) vengano eseguiti con un accesso limitato al sistema. I contenitori sfruttano l'`container_t`etichetta che è un alias di. `svirt_lxc_net_t` Queste politiche impediscono efficacemente ai container di accedere a determinate funzionalità dell'host.

Quando SELinux configuri per Docker, Docker etichetta automaticamente i carichi di lavoro `container_t` in base al tipo e assegna a ciascun contenitore un livello MCS univoco. Questo isolerà i contenitori l'uno dall'altro. Se hai bisogno di restrizioni più flessibili, puoi creare il tuo profilo in SElinux cui concedere a un contenitore le autorizzazioni per aree specifiche del file system. Questo è simile al PSPs fatto che è possibile creare profili diversi per contenitori/pod diversi. Ad esempio, puoi avere un profilo per i carichi di lavoro generali con una serie di controlli restrittivi e un altro per gli elementi che richiedono un accesso privilegiato.

SELinux for Containers dispone di una serie di opzioni che possono essere configurate per modificare le restrizioni predefinite. I seguenti valori SELinux booleani possono essere abilitati o disabilitati in base alle esigenze:

Booleano	Predefinita	Description
`container_connect_any`	`off`	Consenti ai container di accedere alle porte privilegiate sull'host. Ad esempio, se disponi di un contenitore

Booleano	Predefinita	Description
		che deve mappare le porte su 443 o 80 sull'host.
`container_manage_cgroup`	`off`	Consenti ai contenitori di gestire la configurazione di cgroup. Ad esempio, un contenitore che esegue systemd avrà bisogno che questo sia abilitato.
`container_use_cephfs`	`off`	Consenti ai contenitori di utilizzare un file system ceph.

Per impostazione predefinita, i contenitori possono visualizzare `/usr` e read/execute leggere la maggior parte dei contenuti. `/etc` I file `/var/lib/docker` sottostanti `/var/lib/containers` hanno l'etichetta`container_var_lib_t`. Per visualizzare un elenco completo delle etichette predefinite, consulta il file container.fc.

```
docker container run -it \
  -v /var/lib/docker/image/overlay2/repositories.json:/host/repositories.json \
  centos:7 cat /host/repositories.json
# cat: /host/repositories.json: Permission denied

docker container run -it \
  -v /etc/passwd:/host/etc/passwd \
  centos:7 cat /host/etc/passwd
# cat: /host/etc/passwd: Permission denied
```

I file etichettati con `container_file_t` sono gli unici file scrivibili dai contenitori. Se si desidera che un volume mount sia scrivibile, è necessario specificare `:z` o alla `:Z` fine.

- `:z`rietichetterà i file in modo che il contenitore possa leggere/scrivere
- `:Z`rietichetterà i file in modo che solo il contenitore possa leggere/scrivere

```
ls -Z /var/lib/misc
```

```
# -rw-r--r--. root root system_u:object_r:var_lib_t:s0   postfix.aliasesdb-stamp

docker container run -it \
  -v /var/lib/misc:/host/var/lib/misc:z \
  centos:7 echo "Relabeled!"

ls -Z /var/lib/misc
#-rw-r--r--. root root system_u:object_r:container_file_t:s0 postfix.aliasesdb-stamp
```

```
docker container run -it \
  -v /var/log:/host/var/log:Z \
  fluentbit:latest
```

In Kubernetes, la rietichettatura è leggermente diversa. Invece di far rietichettare automaticamente i file da Docker, puoi specificare un'etichetta MCS personalizzata per eseguire il pod. I volumi che supportano la rietichettatura verranno automaticamente rietichettati in modo da essere accessibili. I pod con un'etichetta MCS corrispondente potranno accedere al volume. Se hai bisogno di un isolamento rigoroso, imposta un'etichetta MCS diversa per ogni pod.

```
securityContext:
  seLinuxOptions:
    # Provide a unique MCS label per container
    # You can specify user, role, and type also
    # enforcement based on type and level (svert)
    level: s0:c144:c154
```

In questo esempio `s0:c144:c154` corrisponde a un'etichetta MCS assegnata a un file a cui il contenitore può accedere.

Su EKS è possibile creare politiche che consentano l'esecuzione di contenitori privilegiati, come FluentD e creare SELinux una politica per consentirgli di leggere da /var/log sull'host senza dover rietichettare la directory host. I pod con la stessa etichetta saranno in grado di accedere agli stessi volumi host.

Abbiamo implementato esempi AMIs per Amazon EKS SELinux configurati su CentOS 7 e RHEL 7. Questi AMIs sono stati sviluppati per dimostrare implementazioni di esempio che soddisfano i requisiti di clienti altamente regolamentati.

> **Warning**
>
> SELinux ignorerà i contenitori in cui il tipo non è limitato.

Strumenti e risorse

- SELinuxKubernetes RBAC e politiche di sicurezza delle spedizioni per applicazioni on-premise
- Rafforzamento iterativo di Kubernetes
- Audit 2 Allow
- SEAlert
- Genera SELinux politiche per contenitori con Udica descrive uno strumento che esamina i file delle specifiche dei container per le funzionalità, le porte e i punti di montaggio di Linux e genera una serie di SELinux regole che consentono al contenitore di funzionare correttamente
- Playbook AMI Hardening per rafforzare il sistema operativo per soddisfare diversi requisiti normativi
- Keiko Upgrade Manager è un progetto open source di Intuit che orchestra la rotazione dei nodi di lavoro.
- Sysdig Secure
- eksctl

Conformità

La conformità è una responsabilità condivisa tra AWS e i consumatori dei suoi servizi. In generale, AWS è responsabile della «sicurezza del cloud» mentre i suoi utenti sono responsabili della «sicurezza nel cloud». La linea che definisce le responsabilità di AWS e dei suoi utenti varia a seconda del servizio. Ad esempio, con Fargate, AWS è responsabile della gestione della sicurezza fisica dei suoi data center, dell'hardware, dell'infrastruttura virtuale (Amazon EC2) e del runtime dei container (Docker). Gli utenti di Fargate sono responsabili della protezione dell'immagine del contenitore e della relativa applicazione. Sapere chi è responsabile di ciò è un fattore importante quando si eseguono carichi di lavoro che devono rispettare gli standard di conformità.

La tabella seguente mostra i programmi di conformità a cui si conformano i diversi servizi di container.

Programma di conformità	Amazon ECS Orchestrator	Amazon EKS Orchestrator	ECS Fargate	Amazon ECR
PCI DSS livello 1	1	1	1	1
Conforme a HIPAA	1	1	1	1
SOC I	1	1	1	1
SOCI II	1	1	1	1
SOCI III	1	1	1	1
ISO 27001:2013	1	1	1	1
ISO 9001:2015	1	1	1	1
ISO 27017:2015	1	1	1	1
ISO 27018:2019	1	1	1	1
IRAP	1	1	1	1
FedRAMP Moderate (Est/Ovest)	1	1	0	1
FedRAMP High () GovCloud	1	1	0	1
HA FATTO CC SRG	1	Recensione DISA () IL5	0	1
HIPAA BAA	1	1	1	1
MTCS	1	1	0	1
C5	1	1	0	1
K-ISMS	1	1	0	1
ENS High	1	1	0	1

Programma di conformità	Amazon ECS Orchestrator	Amazon EKS Orchestrator	ECS Fargate	Amazon ECR
OSPAR	1	1	0	1
HITRUST CSF	1	1	1	1

Lo stato di conformità cambia nel tempo. Per lo stato più recente, consulta sempre https://aws.amazon.com/compliance/ services-in-scope /.

Per ulteriori informazioni sui modelli e le best practice di accreditamento sul cloud, consulta il white paper di AWS Accreditation Models for Secure Cloud Adoption

Spostamento a sinistra

Il concetto di spostamento a sinistra implica l'individuazione di violazioni ed errori delle politiche nelle prime fasi del ciclo di vita dello sviluppo del software. Dal punto di vista della sicurezza, questo può essere molto utile. Uno sviluppatore, ad esempio, può risolvere i problemi relativi alla configurazione prima che l'applicazione venga distribuita nel cluster. Individuare tempestivamente errori come questo aiuterà a prevenire l'implementazione di configurazioni che violano le policy.

La politica come codice

La politica può essere pensata come un insieme di regole per governare i comportamenti, ad esempio i comportamenti consentiti o quelli proibiti. Ad esempio, potresti avere una politica che afferma che tutti i Dockerfile devono includere una direttiva USER che fa funzionare il contenitore come utente non root. Come documento, una politica come questa può essere difficile da scoprire e applicare. Potrebbe inoltre diventare obsoleta man mano che i requisiti cambiano. Con le soluzioni Policy as Code (PaC), puoi automatizzare i controlli di sicurezza, conformità e privacy per rilevare, prevenire, ridurre e contrastare le minacce note e persistenti. Inoltre, offrono un meccanismo per codificare le politiche e gestirle come si fanno con altri artefatti di codice. Il vantaggio di questo approccio è che puoi riutilizzare le tue GitOps strategie DevOps e per gestire e applicare in modo coerente le policy tra flotte di cluster Kubernetes. Fai riferimento a Pod Security per informazioni sulle opzioni PaC e sul futuro di PSPs.

Utilizza policy-as-code gli strumenti nelle pipeline per rilevare le violazioni prima della distribuzione

- OPA è un motore di policy open source che fa parte del CNCF. Viene utilizzato per prendere decisioni politiche e può essere eseguito in vari modi, ad esempio come libreria linguistica o servizio. Le politiche OPA sono scritte in un Domain Specific Language (DSL) chiamato Rego. Sebbene venga spesso eseguito come parte di un Kubernetes Dynamic Admission Controller come progetto Gatekeeper, OPA può anche essere incorporato nella tua pipeline. CI/CD Ciò consente agli sviluppatori di ottenere un feedback sulla loro configurazione nelle prime fasi del ciclo di rilascio, che può successivamente aiutarli a risolvere i problemi prima di passare alla produzione. Una raccolta di politiche OPA comuni è disponibile nell' GitHubarchivio di questo progetto.
- Conftest è basato su OPA e offre un'esperienza incentrata sugli sviluppatori per testare la configurazione di Kubernetes.
- Kyverno è un motore di policy progettato per Kubernetes. Con Kyverno, le policy vengono gestite come risorse Kubernetes e non è necessaria alcuna nuova lingua per scrivere le policy. Ciò consente di utilizzare strumenti familiari come kubectl, git e kustomize per gestire le politiche. Le policy di Kyverno possono convalidare, modificare e generare risorse Kubernetes, oltre a garantire la sicurezza della catena di fornitura delle immagini OCI. La CLI di Kyverno può essere utilizzata per testare le politiche e convalidare le risorse come parte di una pipeline. CI/CD Tutte le politiche della community di Kyverno sono disponibili sul sito Web di Kyverno e, per esempi di utilizzo della CLI di Kyverno per scrivere test nelle pipeline, consulta l'archivio delle politiche.

Strumenti e risorse

- Workshop Amazon EKS Security Immersion - Conformità normativa
- panca cubica
- docker-bench-security
- AWS Inspector
- Kubernetes Security Review Una valutazione di sicurezza di terze parti di Kubernetes 1.13.4 (2019)
- NeuVector di SUSE, piattaforma open source di sicurezza per container zero-trust, fornisce report di conformità e controlli di conformità personalizzati

Risposta agli incidenti e analisi forensi

La capacità di reagire rapidamente a un incidente può contribuire a ridurre al minimo i danni causati da una violazione. Disporre di un sistema di allarme affidabile in grado di avvisarvi di comportamenti sospetti è il primo passo di un buon piano di risposta agli incidenti. Quando si verifica un incidente, è necessario decidere rapidamente se distruggere e sostituire il contenitore interessato o isolare e ispezionare il contenitore. Se si sceglie di isolare il contenitore nell'ambito di un'indagine forense e di un'analisi delle cause alla radice, è necessario seguire la seguente serie di attività:

Esempio di piano di risposta agli incidenti

Identifica il Pod e il nodo di lavoro incriminati

La prima cosa da fare dovrebbe essere quella di isolare il danno. Inizia identificando dove si è verificata la violazione e isola il Pod e il relativo nodo dal resto dell'infrastruttura.

Identifica i Pod e i nodi di lavoro offensivi utilizzando il nome del carico di lavoro

Se conosci il nome e lo spazio dei nomi del pod incriminato, puoi identificare il nodo di lavoro che esegue il pod nel modo seguente:

```
kubectl get pods <name> --namespace <namespace> -o=jsonpath='{.spec.nodeName}{"\n"}'
```

Se una risorsa del carico di lavoro come una distribuzione è stata compromessa, è probabile che tutti i pod che fanno parte della risorsa del carico di lavoro siano compromessi. Usa il comando seguente per elencare tutti i pod della Workload Resource e i nodi su cui sono in esecuzione:

```
selector=$(kubectl get deployments <name> \
 --namespace <namespace> -o json | jq -j \
'.spec.selector.matchLabels | to_entries | .[] | "\(.key)=\(.value)"')

kubectl get pods --namespace <namespace> --selector=$selector \
-o json | jq -r '.items[] | "\(.metadata.name) \(.spec.nodeName)"'
```

Il comando precedente è per le distribuzioni. È possibile eseguire lo stesso comando per altre risorse del carico di lavoro come replicasets, statefulsets, ecc.

Identifica i Pod e i nodi di lavoro incriminati utilizzando il nome dell'account di servizio

In alcuni casi, è possibile che un account di servizio sia compromesso. È probabile che i pod che utilizzano l'account di servizio identificato siano compromessi. È possibile identificare tutti i pod utilizzando l'account di servizio e i nodi su cui sono in esecuzione con il seguente comando:

```
kubectl get pods -o json --namespace <namespace> | \
    jq -r '.items[] |
    select(.spec.serviceAccount == "<service account name>") |
    "\(.metadata.name) \(.spec.nodeName)"'
```

Identifica i pod con immagini e nodi di lavoro vulnerabili o compromessi

In alcuni casi, potresti scoprire che un'immagine del contenitore utilizzata nei pod del tuo cluster è dannosa o compromessa. Un'immagine del contenitore è dannosa o compromessa, se è stata rilevata che contiene malware, è un'immagine notoriamente difettosa o ha un CVE che è stato sfruttato. Dovresti considerare compromessi tutti i pod che utilizzano l'immagine del contenitore. È possibile identificare i pod utilizzando l'immagine e i nodi su cui sono in esecuzione con il seguente comando:

```
IMAGE=<Name of the malicious/compromised image>

kubectl get pods -o json --all-namespaces | \
    jq -r --arg image "$IMAGE" '.items[] |
    select(.spec.containers[] | .image == $image) |
    "\(.metadata.name) \(.metadata.namespace) \(.spec.nodeName)"'
```

Isola il Pod creando una politica di rete che neghi tutto il traffico in ingresso e in uscita verso il pod

Una regola di negazione totale del traffico può aiutare a fermare un attacco già in corso interrompendo tutte le connessioni al pod. La seguente politica di rete verrà applicata a un pod con l'etichetta. `app=web`

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny
spec:
```

```
  podSelector:
    matchLabels:
      app: web
  policyTypes:
  - Ingress
  - Egress
```

> **Important**
>
> Una politica di rete può rivelarsi inefficace se un utente malintenzionato ha ottenuto l'accesso all'host sottostante. Se sospetti che ciò sia accaduto, puoi utilizzare AWS Security Groups per isolare un host compromesso dagli altri host. Quando modifichi il gruppo di sicurezza di un host, tieni presente che ciò avrà un impatto su tutti i contenitori in esecuzione su quell'host.

Se necessario, revocate le credenziali di sicurezza temporanee assegnate al pod o al nodo di lavoro

Se al nodo di lavoro è stato assegnato un ruolo IAM che consente ai Pods di accedere ad altre risorse AWS, rimuovi tali ruoli dall'istanza per prevenire ulteriori danni causati dall'attacco. Allo stesso modo, se al pod è stato assegnato un ruolo IAM, valuta se puoi rimuovere in sicurezza le policy IAM dal ruolo senza influire sugli altri carichi di lavoro.

Cordona il nodo di lavoro

Collegando il nodo di lavoro interessato, stai informando lo scheduler di evitare di programmare i pod sul nodo interessato. Ciò ti consentirà di rimuovere il nodo per lo studio forense senza interrompere altri carichi di lavoro.

> **Note**
>
> Questa guida non è applicabile a Fargate, dove ogni pod Fargate viene eseguito nel proprio ambiente in modalità sandbox. Invece di isolare i pod Fargate interessati, isolate i pod Fargate interessati applicando una politica di rete che neghi tutto il traffico in ingresso e in uscita.

Abilita la protezione dalla terminazione sul nodo di lavoro interessato

Un utente malintenzionato può tentare di cancellare i propri misfatti chiudendo un nodo interessato. L'attivazione della protezione dalla terminazione può impedire che ciò accada. La protezione scale-in dell'istanza proteggerà il nodo da un evento di scalabilità interna.

> **Warning**
>
> Non è possibile abilitare la protezione dalla terminazione su un'istanza Spot.

Etichetta il reato Pod/Node con un'etichetta che indichi che fa parte di un'indagine attiva

Questo servirà da avvertimento agli amministratori del cluster affinché non interferiscano con le persone interessate Pods/Nodes fino al completamento dell'indagine.

Acquisisci artefatti volatili sul nodo di lavoro

- Acquisisci la memoria del sistema operativo. Ciò acquisirà il demone Docker (o altro runtime del contenitore) e i relativi sottoprocessi per contenitore. Ciò può essere ottenuto utilizzando strumenti come LiMe e Volatility o tramite strumenti di livello superiore come Automated Forensics Orchestrator for Amazon EC2 che si basano su di essi.
- Esegui un dump ad albero netstat dei processi in esecuzione e delle porte aperte. Questo catturerà il demone docker e il suo sottoprocesso per contenitore.
- Esegui comandi per salvare lo stato a livello di contenitore prima che le prove vengano alterate. Puoi utilizzare le funzionalità del runtime del contenitore per acquisire informazioni sui contenitori attualmente in esecuzione. Ad esempio, con Containerd, puoi fare quanto segue:
  - `crictl ps`per i processi in esecuzione.
  - `crictl logs CONTAINER`per i log conservati a livello di demone.

    Lo stesso potrebbe essere ottenuto con containerd utilizzando la CLI nerdctl, al posto di (ad es.). `docker` `nerdctl inspect` Sono disponibili alcuni comandi aggiuntivi a seconda del runtime del contenitore. Ad esempio, Docker deve visualizzare `docker diff` le modifiche al filesystem del contenitore o `docker checkpoint` salvare tutto lo stato del contenitore, inclusa la memoria volatile (RAM). Consulta questo post sul blog di Kubernetes per discutere di funzionalità simili con i runtime containerd o CRI-O.

- Metti in pausa il contenitore per l'acquisizione forense.
- Crea un'istantanea dei volumi EBS dell'istanza.

Ridistribuisci il Pod o la risorsa del carico di lavoro compromessa

Dopo aver raccolto i dati per l'analisi forense, puoi ridistribuire il pod o la risorsa del carico di lavoro compromessi.

Per prima cosa, implementa la correzione della vulnerabilità che è stata compromessa e avvia nuovi pod sostitutivi. Quindi elimina i pod vulnerabili.

Se i pod vulnerabili sono gestiti da una risorsa di carico di lavoro Kubernetes di livello superiore (ad esempio, una distribuzione o DaemonSet), la loro eliminazione ne programmerà di nuovi. Quindi i pod vulnerabili verranno lanciati nuovamente. In tal caso, è necessario distribuire una nuova risorsa sostitutiva per il carico di lavoro dopo aver corretto la vulnerabilità. Quindi è necessario eliminare il carico di lavoro vulnerabile.

Raccomandazioni

Leggi il white paper di AWS Security Incident Response

Sebbene questa sezione fornisca una breve panoramica insieme ad alcuni consigli per gestire le sospette violazioni della sicurezza, l'argomento è trattato in modo esaustivo nel white paper AWS Security Incident Response.

Pratica giornate dedicate ai giochi di sicurezza

Dividi i tuoi addetti alla sicurezza in 2 squadre: rossa e blu. La squadra rossa si concentrerà sulla ricerca di vulnerabilità nei diversi sistemi, mentre la squadra blu si occuperà di difendersi da esse. Se non disponi di un numero sufficiente di professionisti della sicurezza per creare team separati, valuta la possibilità di assumere un'entità esterna che conosca gli exploit di Kubernetes.

Kubesploit è un framework di test di penetrazione che puoi utilizzare per condurre giornate di gioco. CyberArk A differenza di altri strumenti che scansionano il cluster alla ricerca di vulnerabilità, kubesploit simula un attacco nel mondo reale. Questo dà alla tua squadra blu l'opportunità di mettere in pratica la sua risposta a un attacco e di valutarne l'efficacia.

Esegui test di penetrazione sul tuo cluster

Attaccare periodicamente il tuo cluster può aiutarti a scoprire vulnerabilità ed errori di configurazione. Prima di iniziare, segui le linee guida relative ai test di penetrazione prima di eseguire un test sul cluster.

Strumenti e risorse

- kube-hunter, uno strumento di test di penetrazione per Kubernetes.
- Gremlin, un toolkit di ingegneria del caos che puoi usare per simulare attacchi contro le tue applicazioni e la tua infrastruttura.
- Attacco e difesa delle installazioni Kubernetes
- kubesploit
- NeuVector di SUSE, piattaforma open source di sicurezza per container zero-trust, fornisce segnalazioni di vulnerabilità e rischi, nonché notifica degli eventi di sicurezza
- Minacce persistenti avanzate
- Attacco e difesa pratici di Kubernetes
- Compromissione del cluster Kubernetes sfruttando le autorizzazioni RBAC

Sicurezza delle immagini

L'immagine di container deve essere considerata la prima linea di difesa contro un attacco. Un'immagine non sicura e mal costruita può consentire a un utente malintenzionato di sfuggire ai limiti del container e accedere all'host. Una volta sull'host, un malintenzionato può accedere a informazioni sensibili o spostarsi lateralmente all'interno del cluster o con il tuo account AWS. Le seguenti best practice aiuteranno a mitigare il rischio che ciò accada.

Raccomandazioni

Crea immagini minime

Inizia rimuovendo tutti i file binari estranei dall'immagine di container. Se stai usando un'immagine sconosciuta di Dockerhub, ispeziona l'immagine usando un'applicazione come Dive che può mostrarti il contenuto di ciascuno dei livelli del contenitore. Rimuovi tutti i file binari con i bit SETUID e SETGID in quanto possono essere usati per aumentare i privilegi e valuta la possibilità di rimuovere tutte le

shell e le utilità come nc e curl che possono essere utilizzate per scopi malvagi. Puoi trovare i file con bit SETUID e SETGID con il seguente comando:

```
find / -perm /6000 -type f -exec ls -ld {} \;
```

Per rimuovere i permessi speciali da questi file, aggiungi la seguente direttiva all'immagine del contenitore:

```
RUN find / -xdev -perm /6000 -type f -exec chmod a-s {} \; || true
```

Colloquialmente, questa operazione è nota come snaturamento dell'immagine.

Usa build in più fasi

L'utilizzo di build in più fasi è un modo per creare immagini minime. Spesso, le build in più fasi vengono utilizzate per automatizzare parti del ciclo di integrazione continua. Ad esempio, è possibile utilizzare build in più fasi per linkare il codice sorgente o eseguire analisi statiche del codice. Ciò offre agli sviluppatori l'opportunità di ottenere un feedback quasi immediato invece di attendere l'esecuzione di una pipeline. Le build in più fasi sono interessanti dal punto di vista della sicurezza perché consentono di ridurre al minimo le dimensioni dell'immagine finale inserita nel registro dei contenitori. Le immagini di container prive di strumenti di compilazione e altri file binari estranei migliorano il livello di sicurezza, riducendo la superficie di attacco dell'immagine. Per ulteriori informazioni sulle build in più fasi, consulta la documentazione delle build in più fasi di Docker.

Crea Software Bill of Materials () SBOMs per l'immagine del tuo contenitore

Una «lista dei materiali del software» (SBOM) è un inventario annidato degli artefatti software che compongono l'immagine del contenitore. La SBOM è un elemento fondamentale nella sicurezza del software e nella gestione del rischio della catena di fornitura del software. La generazione, l'archiviazione degli SBOM in un archivio centrale e la scansione delle vulnerabilità aiutano a risolvere SBOMs i seguenti problemi:

- Visibilità: scopri quali componenti costituiscono l'immagine del tuo contenitore. L'archiviazione in un repository centrale consente di SBOMs essere controllati e scansionati in qualsiasi momento, anche dopo l'implementazione, per rilevare e rispondere a nuove vulnerabilità, come le vulnerabilità zero-day.
- Verifica della provenienza: garanzia che le ipotesi esistenti sulla provenienza e sull'origine di un artefatto siano vere e che l'artefatto o i relativi metadati non siano stati manomessi durante i processi di costruzione o consegna.

- Affidabilità: garanzia che un determinato artefatto e il suo contenuto facciano ciò che si presume facciano, ossia siano adatti a uno scopo. Ciò implica valutare se il codice è sicuro da eseguire e prendere decisioni informate sui rischi associati all'esecuzione del codice. L'affidabilità è garantita dalla creazione di un rapporto di esecuzione della pipeline attestato insieme al rapporto SBOM attestato e al rapporto di scansione CVE attestato, per assicurare ai consumatori dell'immagine che l'immagine è effettivamente creata con mezzi sicuri (pipeline) con componenti sicuri.
- Verifica della fiducia delle dipendenze: controllo ricorsivo dell'albero delle dipendenze di un artefatto per verificarne l'affidabilità e la provenienza degli artefatti utilizzati. Il drift in SBOMs può aiutare a rilevare attività dannose, tra cui dipendenze non autorizzate e non attendibili e tentativi di infiltrazione.

I seguenti strumenti possono essere utilizzati per generare SBOM:

- Amazon Inspector può essere utilizzato per creare ed esportare. SBOMs
- Syft di Anchore può essere utilizzato anche per la generazione di SBOM. Per scansioni di vulnerabilità più rapide, l'SBOM generato per un'immagine del contenitore può essere utilizzato come input per la scansione. L'SBOM e il rapporto di scansione vengono quindi attestati e allegati all'immagine prima di inviarla a un repository OCI centrale come Amazon ECR per scopi di revisione e controllo.

Scopri di più su come proteggere la tua catena di fornitura del software consultando la guida CNCF Software Supply Chain Best Practices.

Scansiona regolarmente le immagini per individuare eventuali vulnerabilità

Analogamente alle macchine virtuali, le immagini dei container possono contenere file binari e librerie di applicazioni con vulnerabilità o sviluppare vulnerabilità nel tempo. Il modo migliore per proteggersi dagli attacchi è quello di analizzare regolarmente le immagini con uno scanner di immagini. Le immagini archiviate in Amazon ECR possono essere scansionate in modalità push o su richiesta (una volta nell'arco di 24 ore). Attualmente ECR supporta due tipi di scansione: Basic e Enhanced. La scansione di base sfrutta Clair, una soluzione di scansione di immagini open source gratuita. La scansione avanzata utilizza Amazon Inspector per fornire scansioni automatiche continue a costi aggiuntivi. Dopo la scansione di un'immagine, i risultati vengono registrati nel flusso di eventi per l'accesso ECR. EventBridge È inoltre possibile visualizzare i risultati di una scansione dall'interno della console ECR. Le immagini con una vulnerabilità ALTA o CRITICA devono essere eliminate o ricostruite. Un'immagine implementata che sviluppa una vulnerabilità deve essere sostituita il prima possibile.

Sapere dove sono state distribuite le immagini con vulnerabilità è essenziale per proteggere l'ambiente. Sebbene sia possibile creare autonomamente una soluzione di tracciamento delle immagini, esistono già diverse offerte commerciali che forniscono questa e altre funzionalità avanzate pronte all'uso, tra cui:

- Grype
- Palo Alto - Prisma Cloud (twistcli)
- Acqua
- Kubei
- Trivy
- Snyk

Un webhook di convalida Kubernetes potrebbe essere utilizzato anche per verificare che le immagini siano prive di vulnerabilità critiche. I webhook di convalida vengono richiamati prima dell'API Kubernetes. In genere vengono utilizzati per rifiutare le richieste che non soddisfano i criteri di convalida definiti nel webhook. Questo è un esempio di webhook serverless che richiama l'API ECR describeImageScan Findings per determinare se un pod sta estraendo un'immagine con vulnerabilità critiche. Se vengono rilevate vulnerabilità, il pod viene rifiutato e un messaggio con un elenco di CVEs viene restituito come evento.

Usa gli attestati per convalidare l'integrità degli artefatti

Un attestato è una «dichiarazione» firmata crittograficamente che afferma che qualcosa, ad esempio un «predicato», ad esempio l'esecuzione di una pipeline o lo SBOM o il rapporto di scansione delle vulnerabilità, riguarda un'altra cosa: un «soggetto», ovvero l'immagine del contenitore.

Gli attestati aiutano gli utenti a verificare che un artefatto provenga da una fonte attendibile nella catena di fornitura del software. Ad esempio, possiamo utilizzare l'immagine di un contenitore senza conoscere tutti i componenti software o le dipendenze inclusi in tale immagine. Tuttavia, se ci fidiamo di ciò che il produttore dell'immagine del contenitore dice sul software presente, possiamo utilizzare l'attestato del produttore per fare affidamento su quell'artefatto. Ciò significa che possiamo procedere a utilizzare l'artefatto in modo sicuro nel nostro flusso di lavoro anziché eseguire l'analisi da soli.

- Gli attestati possono essere creati utilizzando AWS Signer o Sigstore cosign.
- I controller di ammissione Kubernetes come Kyverno possono essere utilizzati per verificare gli attestati.

- Fai riferimento a questo workshop per saperne di più sulle best practice di gestione della catena di fornitura del software su AWS utilizzando strumenti open source con argomenti come la creazione e l'allegazione di attestati a un'immagine del contenitore.

Crea politiche IAM per i repository ECR

Al giorno d'oggi, non è raro che un'organizzazione abbia più team di sviluppo che operano in modo indipendente all'interno di un account AWS condiviso. Se questi team non hanno bisogno di condividere le risorse, potresti voler creare una serie di policy IAM che limitino l'accesso ai repository con cui ogni team può interagire. Un buon modo per implementarlo è utilizzare i namespace ECR. I namespace sono un modo per raggruppare repository simili. Ad esempio, tutti i registri per il team A possono essere preceduti dal prefisso team-a/ mentre quelli per il team B possono utilizzare il prefisso team-b/. La politica per limitare l'accesso potrebbe essere simile alla seguente:

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AllowPushPull",
      "Effect": "Allow",
      "Action": [
        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchGetImage",
        "ecr:BatchCheckLayerAvailability",
        "ecr:PutImage",
        "ecr:InitiateLayerUpload",
        "ecr:UploadLayerPart",
        "ecr:CompleteLayerUpload"
      ],
      "Resource": [
        "arn:aws:ecr:us-east-1:123456789012:repository/team-a/*"
      ]
    }
  ]
}
```

Prendi in considerazione l'utilizzo di endpoint privati ECR

L'API ECR ha un endpoint pubblico. Di conseguenza, è possibile accedere ai registri ECR da Internet purché la richiesta sia stata autenticata e autorizzata da IAM. Per coloro che devono operare in un

ambiente sandbox in cui il VPC del cluster non dispone di un Internet Gateway (IGW), è possibile configurare un endpoint privato per ECR. La creazione di un endpoint privato consente di accedere in modo privato all'API ECR tramite un indirizzo IP privato anziché indirizzare il traffico su Internet. Per ulteriori informazioni su questo argomento, consulta Endpoint VPC con interfaccia Amazon ECR.

Implementa politiche sugli endpoint per ECR

La policy predefinita per gli endpoint consente l'accesso a tutti gli archivi ECR all'interno di una regione. Ciò potrebbe consentire attacker/insider a un utente di esfiltrare i dati impacchettandoli come immagine di un contenitore e inviandoli a un registro in un altro account AWS. Per mitigare questo rischio è necessario creare una policy sugli endpoint che limiti l'accesso delle API ai repository ECR. Ad esempio, la seguente policy consente a tutti i principi AWS del tuo account di eseguire tutte le azioni sui tuoi e solo sui tuoi repository ECR:

```
{
  "Statement": [
    {
      "Sid": "LimitECRAccess",
      "Principal": "*",
      "Action": "*",
      "Effect": "Allow",
      "Resource": "arn:aws:ecr:<region>:<account_id>:repository/*"
    }
  ]
}
```

Puoi migliorarlo ulteriormente impostando una condizione che utilizzi il nuovo `PrincipalOrgID` attributo che impedirà la creazione pushing/pulling di immagini in base a un principio IAM che non fa parte della tua AWS Organization. Vedi, aws: PrincipalOrg ID per ulteriori dettagli. Consigliamo di applicare la medesima policy sia agli endpoint `com.amazonaws.<region>.ecr.dkr` che agli endpoint `com.amazonaws.<region>.ecr.api`. Poiché EKS estrae le immagini per kube-proxy, coredns e aws-node da ECR, sarà necessario aggiungere l'ID dell'account del registro, ad esempio `602401143452.dkr.ecr.us-west-2.amazonaws.com/` all'elenco delle risorse nella policy degli endpoint, o modificare la politica per consentire l'estrazione e limitare i push all'ID dell'account. La tabella seguente mostra la mappatura tra gli account AWS da cui vengono vendute le immagini EKS e la regione del cluster.

Numero account	Region
602401143452	Tutte le regioni commerciali ad eccezione di quelle elencate di seguito
—	—
800184023465	ap-east-1 - Asia Pacifico (Hong Kong)
558608220178	me-south-1 - Medio Oriente (Bahrein)
918309763551	cn-north-1 - Cina (Pechino)
961992271922	cn-northwest-1 - Cina (Ningxia)

Per ulteriori informazioni sull'utilizzo delle policy degli endpoint, consulta Utilizzo delle policy degli endpoint VPC per controllare l'accesso ad Amazon ECR.

Implementa le politiche del ciclo di vita per ECR

La NIST Application Container Security Guide mette in guardia dal rischio di «immagini obsolete nei registri», osservando che nel tempo le vecchie immagini con pacchetti out-of-date software vulnerabili dovrebbero essere rimosse per evitare la distribuzione e l'esposizione accidentali. Ogni repository ECR può avere una politica del ciclo di vita che stabilisce regole per la scadenza delle immagini. La documentazione ufficiale di AWS descrive come configurare le regole di test, valutarle e quindi applicarle. Nei documenti ufficiali sono disponibili diversi esempi di politiche relative al ciclo di vita che mostrano diversi modi di filtrare le immagini in un repository:

- Filtraggio per età o numero di immagini
- Filtraggio per immagini con o senza tag
- Filtraggio per tag di immagine, in più regole o in un'unica regola

??? + avviso Se l'immagine per un'applicazione a esecuzione prolungata viene eliminata da ECR, può causare errori di estrazione dell'immagine quando l'applicazione viene ridistribuita o ridimensionata orizzontalmente. Quando utilizzate le policy relative al ciclo di vita delle immagini, assicuratevi di CI/CD adottare delle buone pratiche per mantenere aggiornate le distribuzioni e le

immagini a cui fanno riferimento e create sempre regole di scadenza [dell'immagine] che tengano conto della frequenza con cui eseguite rilasci/distribuzioni.

Creazione di un set di immagini curate

Piuttosto che consentire agli sviluppatori di creare le proprie immagini, prendete in considerazione la possibilità di creare una serie di immagini controllate per i diversi stack di applicazioni dell'organizzazione. In tal modo, gli sviluppatori possono fare a meno di imparare a comporre Dockerfile e concentrarsi piuttosto sulla scrittura di codice. Man mano che le modifiche vengono unite in Master, una CI/CD pipeline può compilare automaticamente l'asset, archiviarlo in un repository di artefatti e copiare l'elemento nell'immagine appropriata prima di inviarlo a un registro Docker come ECR. Per lo meno dovresti creare un set di immagini di base da cui gli sviluppatori possano creare i propri Dockerfile. Idealmente, dovresti evitare di estrarre immagini da Dockerhub perché 1/ non sempre sai cosa c'è nell'immagine e 2/ circa un quinto delle prime 1000 immagini presenta delle vulnerabilità. Un elenco di queste immagini e delle relative vulnerabilità è disponibile qui.

Aggiungi la direttiva USER ai tuoi Dockerfiles per eseguirli come utente non root

Come accennato nella sezione sulla sicurezza dei pod, dovresti evitare di eseguire container come root. Sebbene sia possibile configurarlo come parte di PodSpec, è una buona abitudine utilizzare la `USER` direttiva sui Dockerfile. La `USER` direttiva imposta l'UID da utilizzare durante l'esecuzione `RUN` o l'`CMD`istruzione che appare dopo la direttiva USER. `ENTRYPOINT`

Lint i tuoi Dockerfile

Linting può essere utilizzato per verificare che i Dockerfile aderiscano a una serie di linee guida predefinite, ad esempio l'inclusione della `USER` direttiva o il requisito che tutte le immagini siano etichettate. dockerfile_lint è un progetto open source RedHat che verifica le migliori pratiche comuni e include un motore di regole che puoi utilizzare per creare le tue regole per il linting dei Dockerfile. Può essere incorporato in una pipeline CI, in quanto le build con Dockerfile che violano una regola falliranno automaticamente.

Crea immagini partendo da zero

Ridurre la superficie di attacco delle immagini dei container dovrebbe essere l'obiettivo principale nella creazione di immagini. Il modo ideale per farlo è creare immagini minime prive di file binari che possano essere utilizzate per sfruttare le vulnerabilità. Fortunatamente, Docker ha un meccanismo da cui creare immagini. `scratch` Con linguaggi come Go, puoi creare un binario collegato statico e farvi riferimento nel tuo Dockerfile come in questo esempio:

```
############################
# STEP 1 build executable binary
############################
FROM golang:alpine AS builder# Install git.
# Git is required for fetching the dependencies.
RUN apk update && apk add --no-cache gitWORKDIR $GOPATH/src/mypackage/myapp/COPY . . #
 Fetch dependencies.
# Using go get.
RUN go get -d -v# Build the binary.
RUN go build -o /go/bin/hello

############################
# STEP 2 build a small image
############################
FROM scratch# Copy our static executable.
COPY --from=builder /go/bin/hello /go/bin/hello# Run the hello binary.
ENTRYPOINT ["/go/bin/hello"]
```

Questo crea un'immagine del contenitore composta dalla tua applicazione e nient'altro, rendendola estremamente sicura.

Usa tag immutabili con ECR

I tag immutabili ti obbligano ad aggiornare il tag dell'immagine ad ogni invio al repository di immagini. Ciò può impedire a un utente malintenzionato di sovrascrivere un'immagine con una versione dannosa senza modificare i tag dell'immagine. Inoltre, consente di identificare un'immagine in modo semplice e univoco.

Firma le tue immagini SBOMs, le esecuzioni della pipeline e i report sulle vulnerabilità

Quando Docker è stato introdotto per la prima volta, non esisteva un modello crittografico per la verifica delle immagini dei container. Con la v2, Docker ha aggiunto dei digest al manifesto dell'immagine. Ciò ha consentito di eseguire l'hashing della configurazione di un'immagine e di utilizzare l'hash per generare un ID per l'immagine. Quando la firma delle immagini è abilitata, il motore Docker verifica la firma del manifesto, assicurando che il contenuto sia stato prodotto da una fonte affidabile e che non si siano verificate manomissioni. Dopo aver scaricato ogni layer, il motore verifica il digest del layer, assicurando che il contenuto corrisponda al contenuto specificato nel manifesto. La firma delle immagini consente in modo efficace di creare una catena di approvvigionamento sicura, attraverso la verifica delle firme digitali associate all'immagine.

Possiamo usare AWS Signer o Sigstore Cosign per firmare immagini di container, creare attestazioni, report di scansione delle vulnerabilità e report di SBOMs esecuzione della pipeline. Queste attestazioni assicurano l'affidabilità e l'integrità dell'immagine, che è effettivamente creata dalla pipeline affidabile senza alcuna interferenza o manomissione e che contiene solo i componenti software documentati (nello SBOM), verificati e considerati attendibili dall'editore dell'immagine. Queste attestazioni possono essere allegate all'immagine del contenitore e inviate al repository.

Nella prossima sezione vedremo come utilizzare gli artefatti attestati per gli audit e la verifica dei controllori di ammissione.

Verifica dell'integrità delle immagini tramite il controller di ammissione Kubernetes

Possiamo verificare le firme delle immagini e gli artefatti attestati in modo automatizzato prima di distribuire l'immagine sul cluster Kubernetes di destinazione utilizzando un controller di ammissione dinamico e ammettere le distribuzioni solo quando i metadati di sicurezza degli artefatti sono conformi alle politiche del controller di ammissione.

Ad esempio, possiamo scrivere una politica che verifichi crittograficamente la firma di un'immagine, un SBOM attestato, un rapporto di esecuzione della pipeline attestato o un rapporto di scansione CVE attestato. Possiamo scrivere condizioni nella policy per controllare i dati nel report, ad esempio una scansione CVE non dovrebbe presentare alcuna criticità. CVEs La distribuzione è consentita solo per le immagini che soddisfano queste condizioni e tutte le altre implementazioni verranno rifiutate dal responsabile delle ammissioni.

Alcuni esempi di controller di ammissione includono:

- Kyverno
- Guardiano dell'OPA
- Portieri è
- Ratifica
- Kritis
- Tutorial Grafeas
- Buono

Aggiorna i pacchetti nelle immagini del contenitore

Dovresti includere RUN `apt-get update && apt-get upgrade` nei tuoi Dockerfile per aggiornare i pacchetti nelle tue immagini. Sebbene l'aggiornamento richieda l'esecuzione come root,

ciò avviene durante la fase di creazione dell'immagine. L'applicazione non deve essere eseguita come root. È possibile installare gli aggiornamenti e quindi passare a un altro utente con la direttiva USER. Se l'immagine di base viene eseguita come utente non root, passa a root e viceversa; non affidarti esclusivamente ai manutentori dell'immagine di base per installare gli ultimi aggiornamenti di sicurezza.

Esegui `apt-get clean` per eliminare i file di installazione da. `/var/cache/apt/archives/` È possibile eseguire l'operazione anche `rm -rf /var/lib/apt/lists/*` dopo aver installato i pacchetti. Ciò rimuove i file di indice o gli elenchi di pacchetti disponibili per l'installazione. Tieni presente che questi comandi possono essere diversi per ogni gestore di pacchetti. Esempio:

```
RUN apt-get update && apt-get install -y \
    curl \
    git \
    libsqlite3-dev \
    && apt-get clean && rm -rf /var/lib/apt/lists/*
```

Strumenti e risorse

- Workshop di immersione nella sicurezza di Amazon EKS - Sicurezza delle immagini
- docker-slim Crea immagini minime sicure
- dockle Verifica che il tuo Dockerfile sia in linea con le migliori pratiche per la creazione di immagini sicure
- dockerfile-lint Linter basato su regole per Dockerfiles
- hadolint Un linter intelligente per file docker
- Gatekeeper e OPA Un controller di ammissione basato su policy
- Kyverno Un motore di policy nativo di Kubernetes
- in-toto Consente all'utente di verificare se una fase della catena di fornitura era destinata ad essere eseguita e se la fase è stata eseguita dall'attore giusto
- Notary Un progetto per firmare le immagini dei container
- Notaio v2
- Grafeas Un'API aperta per i metadati degli artefatti per controllare e governare la catena di fornitura del software
- NeuVector di SUSE, piattaforma open source e zero-trust per la sicurezza dei container, fornisce la scansione di container, immagini e registri per individuare vulnerabilità, segreti e conformità.

Strategia multi-account

AWS consiglia di utilizzare una strategia con più account e organizzazioni AWS per isolare e gestire applicazioni e dati aziendali. L'utilizzo di una strategia con più account offre molti vantaggi:

- Quote di servizi API AWS aumentate. Le quote vengono applicate agli account AWS e l'utilizzo di più account per i carichi di lavoro aumenta la quota complessiva disponibile per i carichi di lavoro.
- Policy di Identity and Access Management (IAM) più semplici. Concedere ai carichi di lavoro e agli operatori che li supportano l'accesso solo ai propri account AWS significa meno tempo per la creazione di policy IAM granulari per raggiungere il principio del privilegio minimo.
- Isolamento migliorato delle risorse AWS. In base alla progettazione, tutte le risorse fornite all'interno di un account sono logicamente isolate dalle risorse fornite in altri account. Questo limite di isolamento consente di limitare i rischi di problemi relativi all'applicazione, di errori di configurazione o di azioni dannose. Se si verifica un problema all'interno di un account, è possibile ridurre o eliminare l'impatto sui carichi di lavoro contenuti in altri account.
- Altri vantaggi, come descritto nel white paper AWS Multi Account Strategy

Le seguenti sezioni spiegheranno come implementare una strategia multiaccount per i carichi di lavoro EKS utilizzando un approccio di cluster EKS centralizzato o decentralizzato.

Pianificazione di una strategia di account con più carichi di lavoro per cluster multi-tenant

In una strategia AWS con più account, le risorse che appartengono a un determinato carico di lavoro come bucket S3, cluster ElastiCache e tabelle DynamoDB vengono tutte create in un account AWS che contiene tutte le risorse per quel carico di lavoro. Questi vengono definiti account di carico di lavoro e il cluster EKS viene distribuito in un account denominato account cluster. Gli account del cluster verranno esaminati nella prossima sezione. La distribuzione delle risorse in un account di carico di lavoro dedicato è simile alla distribuzione delle risorse Kubernetes in un namespace dedicato.

Gli account del carico di lavoro possono quindi essere ulteriormente suddivisi per ciclo di vita di sviluppo del software o per altri requisiti, se necessario. Ad esempio, un determinato carico di lavoro può avere un account di produzione, un account di sviluppo o account per ospitare istanze di quel carico di lavoro in una regione specifica. Ulteriori informazioni sono disponibili in questo white paper di AWS.

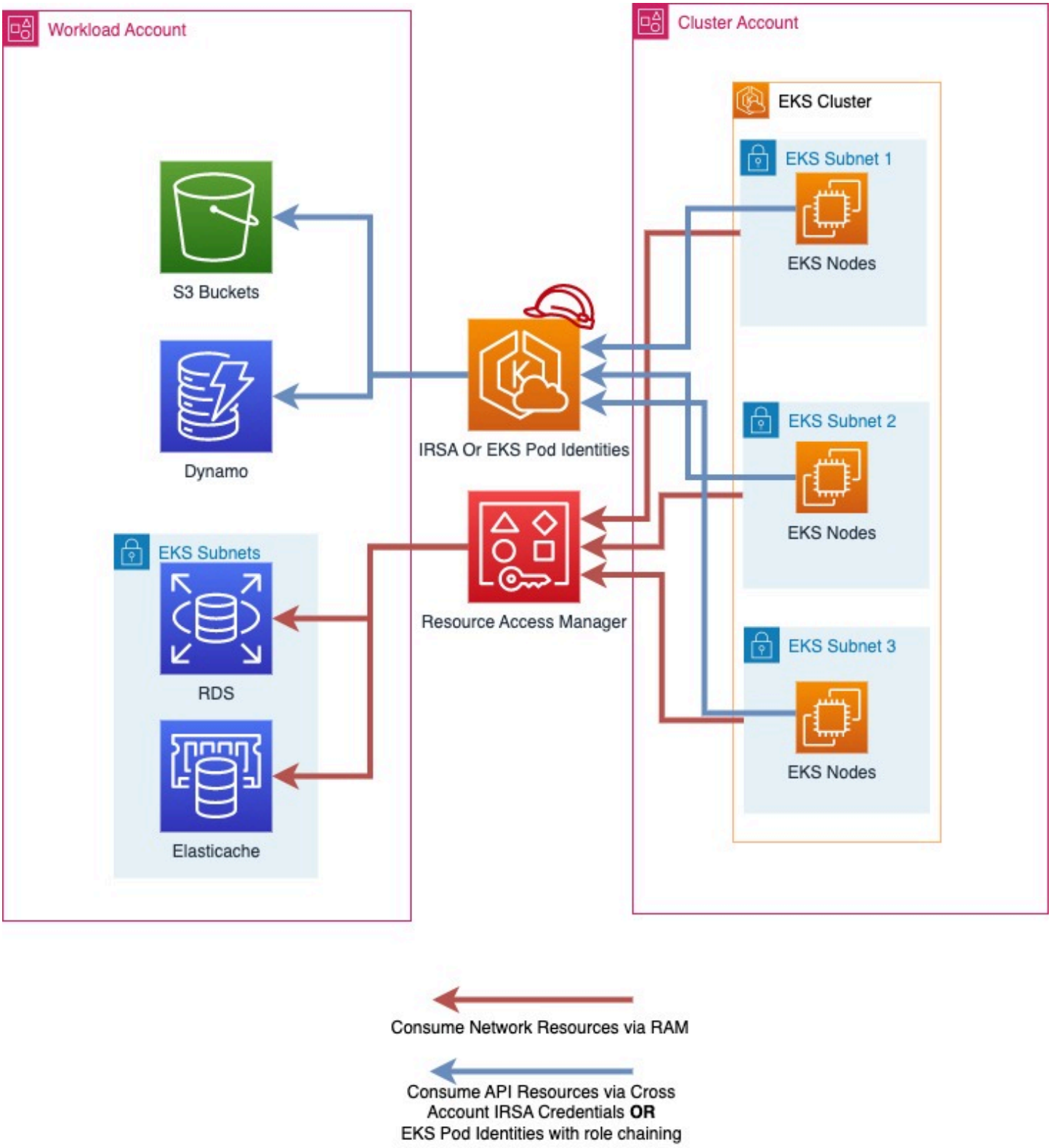
È possibile adottare i seguenti approcci quando si implementa la strategia EKS per più account:

Cluster EKS centralizzato

In questo approccio, il tuo cluster EKS verrà distribuito in un unico account AWS chiamato. `Cluster Account` Utilizzando i ruoli IAM for Service Accounts (IRSA) o EKS Pod Identities per fornire credenziali AWS temporanee e AWS Resource Access Manager (RAM) per semplificare l'accesso alla rete, puoi adottare una strategia multi-account per il tuo cluster EKS multi-tenant. L'account del cluster conterrà il VPC, le sottoreti, il cluster EKS, le risorse di calcolo EC2/Fargate (nodi di lavoro) e tutte le configurazioni di rete aggiuntive necessarie per eseguire il cluster EKS.

In una strategia di account con più carichi di lavoro per cluster multi-tenant, gli account AWS in genere si allineano ai namespace Kubernetes come meccanismo per isolare i gruppi di risorse. Le migliori pratiche per l'isolamento dei tenant all'interno di un cluster EKS devono comunque essere seguite quando si implementa una strategia multi-account per cluster EKS multi-tenant.

È possibile averne più di uno `Cluster Accounts` nella propria organizzazione AWS ed è consigliabile averne più di uno in linea con le `Cluster Accounts` esigenze del ciclo di vita dello sviluppo del software. Per carichi di lavoro che operano su larga scala, potresti averne bisogno di più `Cluster Accounts` per garantire che ci siano abbastanza quote di servizio Kubernetes e AWS disponibili per tutti i tuoi carichi di lavoro.



|Nel diagramma precedente, la RAM AWS viene utilizzata per condividere sottoreti da un account cluster a un account di carico di lavoro. Quindi i carichi di lavoro in esecuzione nei pod EKS utilizzano

IRSA o EKS Pod Identities e il concatenamento dei ruoli per assumere un ruolo nel proprio account di carico di lavoro e accedere alle proprie risorse AWS.

Implementazione di una strategia di account con carichi di lavoro multipli per cluster multi-tenant

Condivisione di sottoreti con AWS Resource Access Manager

AWS Resource Access Manager (RAM) consente di condividere risorse tra account AWS.

Se la RAM è abilitata per la tua AWS Organization, puoi condividere le sottoreti VPC dall'account Cluster ai tuoi account di carico di lavoro. Ciò consentirà alle risorse AWS di proprietà dei tuoi account di carico di lavoro, come i database Amazon ElastiCache Clusters o Amazon Relational Database Service (RDS), di essere distribuite nello stesso VPC del cluster EKS e di essere consumabili dai carichi di lavoro in esecuzione sul tuo cluster EKS.

Per condividere una risorsa tramite RAM, apri la RAM nella console AWS dell'account del cluster e seleziona «Resource Shares» e «Create Resource Share». Assegna un nome a Resource Share e seleziona le sottoreti che desideri condividere. Seleziona nuovamente Avanti e inserisci l'account a 12 cifre IDs per gli account di carico di lavoro con cui desideri condividere le sottoreti, seleziona nuovamente Avanti e fai clic su Crea condivisione di risorse per terminare. Dopo questo passaggio, l'account del carico di lavoro può distribuire risorse in tali sottoreti.

Le condivisioni RAM possono anche essere create a livello di codice o utilizzando l'infrastruttura come codice.

Scelta tra EKS Pod Identities e IRSA

Al re:Invent 2023, AWS ha lanciato EKS Pod Identities come un modo più semplice per fornire credenziali AWS temporanee ai tuoi pod su EKS. Sia IRSA che EKS Pod Identities sono metodi validi per fornire credenziali AWS temporanee ai tuoi pod EKS e continueranno a essere supportate. Dovresti considerare quale metodo di distribuzione soddisfa meglio le tue esigenze.

Quando si lavora con un cluster EKS e più account AWS, IRSA può assumere direttamente ruoli in account AWS diversi dall'account in cui è ospitato direttamente il cluster EKS, mentre le identità EKS Pod richiedono la configurazione del concatenamento dei ruoli. Consulta la documentazione EKS per un confronto approfondito.

Accesso alle risorse API AWS con IAM Roles For Service Accounts

IAM Roles for Service Accounts (IRSA) ti consente di fornire credenziali AWS temporanee ai tuoi carichi di lavoro in esecuzione su EKS. IRSA può essere utilizzato per ottenere credenziali temporanee per i ruoli IAM negli account del carico di lavoro dall'account del cluster. Ciò consente ai carichi di lavoro in esecuzione sui cluster EKS nell'account cluster di consumare senza problemi le risorse dell'API AWS, come i bucket S3 ospitati nell'account del carico di lavoro, e di utilizzare l'autenticazione IAM per risorse come Amazon RDS Databases o Amazon EFS. FileSystems

È possibile accedere alle risorse API AWS e ad altre risorse che utilizzano l'autenticazione IAM in un account di carico di lavoro solo tramite credenziali per i ruoli IAM nello stesso account di carico di lavoro, tranne nei casi in cui l'accesso tra account è possibile ed è stato abilitato esplicitamente.

Abilitazione di IRSA per l'accesso a più account

Per consentire a IRSA per i carichi di lavoro nel tuo account cluster di accedere alle risorse nei tuoi account di carico di lavoro, devi prima creare un provider di identità IAM OIDC nel tuo account di carico di lavoro. Questa operazione può essere eseguita con la stessa procedura per la configurazione di IRSA, ad eccezione del fatto che l'Identity Provider verrà creato nell'account del carico di lavoro.

Quindi, quando configuri IRSA per i tuoi carichi di lavoro su EKS, puoi seguire gli stessi passaggi della documentazione, ma utilizzare l'ID account a 12 cifre dell'account di carico di lavoro, come indicato nella sezione «Esempio: creare un provider di identità dal cluster di un altro account».

Una volta configurato, l'applicazione in esecuzione in EKS sarà in grado di utilizzare direttamente il proprio account di servizio per assumere un ruolo nell'account del carico di lavoro e utilizzare le risorse al suo interno.

Accesso alle risorse API AWS con le identità dei pod EKS

EKS Pod Identities è un nuovo modo di fornire credenziali AWS ai carichi di lavoro in esecuzione su EKS. Le identità dei pod EKS semplificano la configurazione delle risorse AWS in quanto non è più necessario gestire le configurazioni OIDC per fornire le credenziali AWS ai pod su EKS.

Abilitazione di EKS Pod Identities per l'accesso da più account

A differenza di IRSA, EKS Pod Identities può essere utilizzato solo per concedere l'accesso diretto a un ruolo nello stesso account del cluster EKS. Per accedere a un ruolo in un altro account AWS, i pod che utilizzano EKS Pod Identities devono eseguire il Role Chaining.

Il concatenamento dei ruoli può essere configurato in un profilo di applicazione con il relativo file di configurazione aws utilizzando il Process Credentials Provider disponibile in vari AWS. SDKs `credential_process`può essere utilizzato come fonte di credenziali durante la configurazione di un profilo, ad esempio:

```
# Content of the AWS Config file
[profile account_b_role]
source_profile = account_a_role
role_arn = arn:aws:iam::444455556666:role/account-b-role

[profile account_a_role]
credential_process = /eks-credential-processrole.sh
```

L'origine dello script chiamato da credential_process:

```
#!/bin/bash
# Content of the eks-credential-processrole.sh
# This will retreive the credential from the pod identities agent,
# and return it to the AWS SDK when referenced in a profile
curl -H "Authorization: $(cat $AWS_CONTAINER_AUTHORIZATION_TOKEN_FILE)"
 $AWS_CONTAINER_CREDENTIALS_FULL_URI | jq -c '{AccessKeyId: .AccessKeyId,
 SecretAccessKey: .SecretAccessKey, SessionToken: .Token, Expiration: .Expiration,
 Version: 1}'
```

Puoi creare un file di configurazione aws come mostrato sopra con entrambi i ruoli Account A e B e specificare le variabili AWS_CONFIG_FILE e AWS_PROFILE env nelle specifiche del tuo pod. Il webhook EKS Pod identity non ha la precedenza se le variabili env esistono già nelle specifiche del pod.

```
# Snippet of the PodSpec
containers:
  - name: container-name
    image: container-image:version
    env:
    - name: AWS_CONFIG_FILE
      value: path-to-customer-provided-aws-config-file
    - name: AWS_PROFILE
      value: account_b_role
```

Quando si configurano le politiche di trust dei ruoli per il concatenamento dei ruoli con le identità dei pod EKS, è possibile fare riferimento agli attributi specifici EKS come tag di sessione e utilizzare il

controllo degli accessi basato sugli attributi (ABAC) per limitare l'accesso ai ruoli IAM solo a sessioni di identità EKS Pod specifiche, come l'account di servizio Kubernetes a cui appartiene un pod.

Tieni presente che alcuni di questi attributi potrebbero non essere universalmente univoci, ad esempio due cluster EKS possono avere namespace identici e un cluster può avere account di servizio con nomi identici in tutti i namespace. Pertanto, quando si concede l'accesso tramite EKS Pod Identities e ABAC, è consigliabile considerare sempre l'arn e lo spazio dei nomi del cluster quando si concede l'accesso a un account di servizio.

ABAC ed EKS Pod Identities per l'accesso su più account

Quando utilizzi EKS Pod Identities per assumere ruoli (concatenamento di ruoli) in altri account nell'ambito di una strategia multiaccount, hai la possibilità di assegnare un ruolo IAM unico per ogni account di servizio che deve accedere a un altro account, oppure utilizzare un ruolo IAM comune su più account di servizio e utilizzare ABAC per controllare a quali account può accedere.

Per utilizzare ABAC per controllare quali account di servizio possono assumere un ruolo in un altro account con il concatenamento dei ruoli, si crea una dichiarazione sulla politica di fiducia dei ruoli che consente l'assunzione di un ruolo da parte di una sessione di ruolo solo quando sono presenti i valori previsti. La seguente politica di fiducia dei ruoli consentirà a un ruolo dell'account del cluster EKS (ID account 111122223333) di assumere un ruolo solo se i `kubernetes-namespace` tag `eks-cluster-arn` e i tag hanno tutti il `kubernetes-service-account` valore previsto.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:root"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:PrincipalTag/kubernetes-service-account":
 "PayrollApplication",
                    "aws:PrincipalTag/eks-cluster-arn": "arn:aws:eks:us-
east-1:111122223333:cluster/ProductionCluster",
                    "aws:PrincipalTag/kubernetes-namespace": "PayrollNamespace"
                }
            }
```

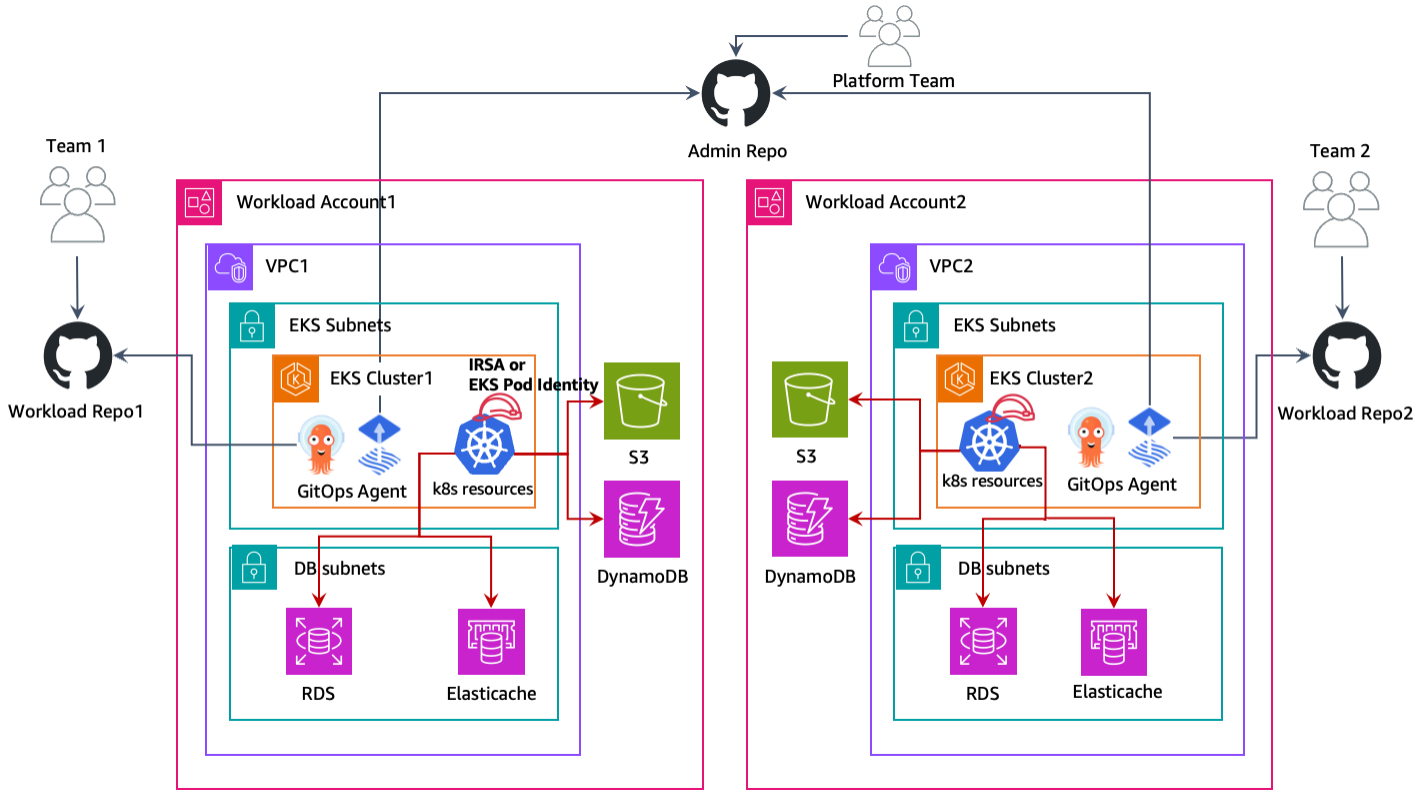
```
        }
    ]
}
```

Quando si utilizza questa strategia, è consigliabile garantire che il ruolo IAM comune disponga solo di `sts:AssumeRole` autorizzazioni e nessun altro accesso AWS.

Quando si utilizza ABAC, è importante controllare chi ha la possibilità di etichettare i ruoli e gli utenti IAM solo a coloro che ne hanno la stretta necessità. Qualcuno con la capacità di taggare un ruolo o un utente IAM sarebbe in grado di impostare tag roles/users identici a quelli impostati da EKS Pod Identities e potrebbe essere in grado di aumentare i propri privilegi. Puoi limitare chi ha l'accesso per impostare i tag e i tag sul ruolo `kubernetes-` e `eks-` sugli utenti IAM utilizzando la policy IAM o la Service Control Policy (SCP).

Cluster EKS decentralizzati

In questo approccio, i cluster EKS vengono distribuiti nei rispettivi account AWS per carichi di lavoro e convivono con altre risorse AWS come bucket Amazon S3 VPCs, tabelle Amazon DynamoDB, ecc. Ogni account di carico di lavoro è indipendente, autosufficiente e gestito dai rispettivi cluster di Unit/Application teams. This model allows the creation of reusable blueprints for various cluster capabilities — AI/ML business, elaborazione in batch, scopi generali, ecc. e vende i cluster in base ai requisiti del team applicativo. Sia i team delle applicazioni che quelli della piattaforma operano dai rispettivi GitOpsrepository per gestire le implementazioni nei cluster di carichi di lavoro.



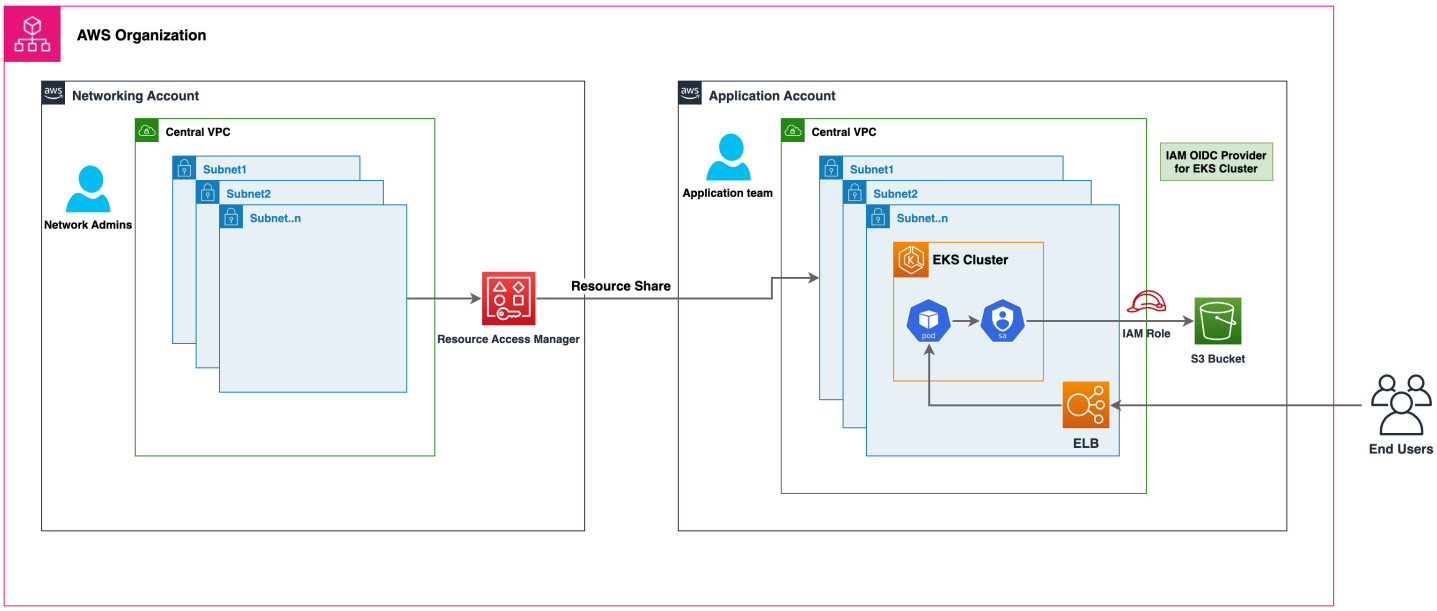
Nel diagramma precedente, i cluster Amazon EKS e altre risorse AWS vengono distribuiti nei rispettivi account di carico di lavoro. Quindi i carichi di lavoro in esecuzione nei pod EKS utilizzano IRSA o EKS Pod Identities per accedere alle proprie risorse AWS.

GitOps è un modo per gestire la distribuzione di applicazioni e infrastrutture in modo che l'intero sistema sia descritto in modo dichiarativo in un repository Git. È un modello operativo che offre la possibilità di gestire lo stato di più cluster Kubernetes utilizzando le migliori pratiche di controllo delle versioni, artefatti immutabili e automazione. In questo modello multicluster, ogni cluster di carico di lavoro viene avviato con più repository Git, che consentono a ciascun team (applicazione, piattaforma, sicurezza, ecc.) di implementare le rispettive modifiche sul cluster.

Utilizzerai i ruoli IAM for Service Accounts (IRSA) o EKS Pod Identities in ogni account per consentire ai tuoi carichi di lavoro EKS di ottenere credenziali aws temporanee per accedere in modo sicuro ad altre risorse AWS. I ruoli IAM vengono creati nei rispettivi account AWS per carichi di lavoro e li mappano agli account di servizio k8s per fornire un accesso IAM temporaneo. Pertanto, in questo approccio non è richiesto l'accesso a più account. Segui la documentazione sui ruoli IAM per gli account di servizio su come configurare ogni carico di lavoro per IRSA e la documentazione di EKS Pod Identities su come configurare le identità dei pod EKS in ogni account.

Rete centralizzata

Puoi anche utilizzare la RAM AWS per condividere le sottoreti VPC con gli account di carico di lavoro e avviare cluster Amazon EKS e altre risorse AWS al loro interno. Ciò consente la gestione/amministrazione centralizzata della rete, una connettività di rete semplificata e cluster EKS decentralizzati. Consulta questo blog di AWS per una guida dettagliata e considerazioni su questo approccio.



Nel diagramma precedente, la RAM AWS viene utilizzata per condividere sottoreti da un account di rete centrale a un account di carico di lavoro. Quindi il cluster EKS e altre risorse AWS vengono lanciati in quelle sottoreti nei rispettivi account di carico di lavoro. I pod EKS utilizzano IRSA o EKS Pod Identities per accedere alle proprie risorse AWS.

Cluster EKS centralizzati e non centralizzati

La decisione di utilizzare un sistema centralizzato o decentralizzato dipenderà dai requisiti dell'utente. Questa tabella illustra le principali differenze tra ciascuna strategia.

#	Cluster EKS centralizzato	Cluster EKS decentralizzati
Gestione dei cluster:	La gestione di un singolo cluster EKS è più semplice dell'amministrazione di più cluster	Un'automazione efficiente della gestione dei cluster è necessaria per ridurre il sovraccarico operativo legato

#	Cluster EKS centralizzato	Cluster EKS decentralizzati
		alla gestione di più cluster EKS
Efficienza dei costi:	Consente il riutilizzo del cluster EKS e delle risorse di rete, promuovendo l'efficienza dei costi	Richiede configurazioni di rete e cluster per carico di lavoro, il che richiede risorse aggiuntive
Resilienza:	La compromissione di un cluster può influire su più carichi di lavoro sul cluster centralizzato	Se un cluster viene danneggia to, il danno è limitato ai soli carichi di lavoro in esecuzion e su quel cluster. Tutti gli altri carichi di lavoro non sono interessati
Isolamento e sicurezza:	Isolation/Soft Multi-ten ancy si ottiene utilizzando costrutti nativi di k8s come. `Namespaces` I carichi di lavoro possono condividere le risorse sottostanti come CPU, memoria, ecc. Le risorse AWS sono isolate nei propri account di carico di lavoro che per impostazione predefinita non sono accessibili da altri account AWS.	Maggiore isolamento delle risorse di elaborazione poiché i carichi di lavoro vengono eseguiti in singoli cluster e nodi che non condividono alcuna risorsa. Le risorse AWS sono isolate nei propri account di carico di lavoro che per impostazione predefinita non sono accessibili da altri account AWS.

#	Cluster EKS centralizzato	Cluster EKS decentralizzati
Prestazioni e scalabità:	Man mano che i carichi di lavoro raggiungono dimension i molto elevate, è possibile che nell'account del cluster si verifichino quote di servizio Kubernetes e AWS. Puoi implementare account cluster aggiuntivi per scalare ulteriormente	Man mano che VPCs sono presenti più cluster, ogni carico di lavoro ha una maggiore quota di servizi k8s e AWS disponibili
Rete:	Viene utilizzato un singolo VPC per cluster, che consente una connettività più semplice per le applicazioni su quel cluster	Il routing deve essere stabilito tra il cluster EKS decentral izzato VPCs
Gestione degli accessi Kubernetes:	È necessario mantenere molti ruoli e utenti diversi nel cluster per fornire l'accesso a tutti i team addetti ai carichi di lavoro e garantire che le risorse Kubernetes siano adeguatamente separate	Gestione semplificata degli accessi in quanto ogni cluster è dedicato a un carico di lavoro/team
Gestione degli accessi AWS:	Le risorse AWS vengono distribuite nel proprio account a cui è possibile accedere per impostazione predefinita solo con ruoli IAM nell'acco unt del carico di lavoro. I ruoli IAM negli account di carico di lavoro si presuppongono su più account con IRSA o EKS Pod Identities.	Le risorse AWS vengono distribuite nel proprio account a cui è possibile accedere per impostazione predefinita solo con ruoli IAM nell'acco unt del carico di lavoro. I ruoli IAM negli account di carico di lavoro vengono assegnati direttamente ai pod con IRSA o EKS Pod Identities

Gestione degli accessi ai cluster

Una gestione efficace degli accessi è fondamentale per mantenere la sicurezza e l'integrità dei cluster Amazon EKS. Questa guida esplora varie opzioni per la gestione degli accessi EKS, con particolare attenzione all'utilizzo di AWS IAM Identity Center (precedentemente AWS SSO). Confronteremo diversi approcci, discuteremo i loro compromessi ed evidenzieremo le limitazioni e le considerazioni note.

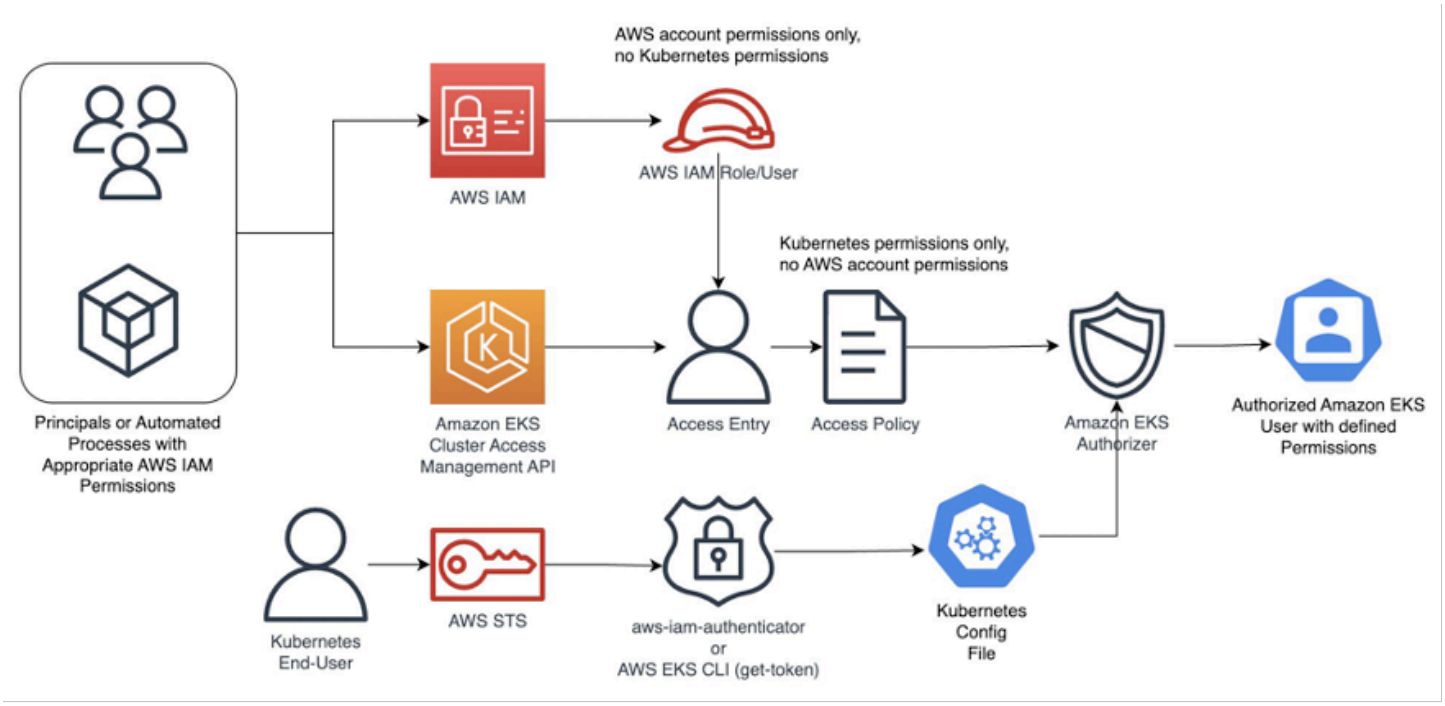
Opzioni di gestione degli accessi EKS

> **Note**
>
> ConfigMapla gestione degli accessi basata su tecnologia (aws-auth ConfigMap) è obsoleta e sostituita dall'API Cluster Access Management (CAM). Per i nuovi cluster EKS, implementa l'API CAM per gestire l'accesso ai cluster. Per i cluster esistenti che utilizzano aws-auth ConfigMap, esegui la migrazione all'utilizzo dell'API CAM.

Opzione 1: AWS IAM Identity Center con API Cluster Access Management (CAM)

- Gestione centralizzata di utenti e autorizzazioni
- Integrazione con provider di identità esistenti (ad esempio Microsoft AD, Okta PingId e altri)
- L'API CAM utilizza Access Entries per collegare i principali (utenti o ruoli) di AWS IAM al cluster EKS. Queste voci funzionano con le identità gestite di IAM Identity Center e consentono agli amministratori di controllare l'accesso al cluster per utenti e gruppi definiti in Identity Center.

Flusso di autenticazione del cluster EKS:



1. I principali (utenti umani) o i processi automatizzati si autenticano tramite AWS IAM presentando le autorizzazioni appropriate all'account AWS. In questa fase, vengono mappati sul principale AWS IAM appropriato (ruolo o utente).
2. Successivamente, una voce di accesso EKS mappa questo principale IAM su un principale Kubernetes RBAC (utente o gruppo) definendo una politica di accesso appropriata, che contiene solo le autorizzazioni Kubernetes.
3. Quando un utente finale Kubernetes tenta di accedere a un cluster, la sua richiesta di autenticazione viene elaborata dalla nostra CLI di aws-iam-authenticator AWS EKS e convalidata rispetto al contesto del cluster nel file kubeconfig.
4. Infine, l'autorizzatore EKS verifica le autorizzazioni associate all'accesso dell'utente autenticato e concede o nega l'accesso di conseguenza.
    - L'API utilizza policy di accesso specifiche di Amazon EKS per definire il livello di autorizzazione per ogni Access Entry. Queste policy possono essere mappate su ruoli e autorizzazioni configurati in IAM Identity Center, garantendo un controllo coerente degli accessi tra i servizi AWS e i cluster EKS.

Vantaggi rispetto alla gestione degli accessi ConfigMap basata:

1. Riduzione del rischio di errori di configurazione: la gestione diretta basata su API elimina gli errori comuni associati alla modifica manuale. ConfigMap Questo aiuta a prevenire eliminazioni accidentali o errori di sintassi che potrebbero impedire agli utenti di accedere al cluster.
2. Principio avanzato del privilegio minimo: elimina la necessità dell'autorizzazione di amministratore del cluster dall'identità del creatore del cluster e consente un'assegnazione delle autorizzazioni più granulare e appropriata. Puoi scegliere di aggiungere questa autorizzazione per i casi d'uso di break-glass.
3. Modello di sicurezza avanzato: fornisce la convalida integrata delle voci di accesso prima che vengano applicate. Inoltre, offre una più stretta integrazione con AWS IAM per l'autenticazione.
4. Operazioni semplificate: offre un modo più intuitivo per gestire le autorizzazioni tramite strumenti nativi di AWS.

Migliori pratiche:

1. Usa AWS Organizations per gestire più account e applicare policy di controllo dei servizi (SCPs).
2. Implementa il principio del privilegio minimo creando set di autorizzazioni specifici per diversi ruoli EKS (ad esempio amministratore, sviluppatore, sola lettura).
3. Utilizza il controllo degli accessi basato sugli attributi (ABAC) per assegnare dinamicamente le autorizzazioni ai pod in base agli attributi utente.
4. Controlla e rivedi regolarmente le autorizzazioni di accesso.

Considerazioni/limitazioni:

1. I ruoli ARNs generati da Identity Center hanno suffissi casuali, il che li rende difficili da usare nelle configurazioni statiche.
2. Supporto limitato per autorizzazioni granulari a livello di risorse Kubernetes. È necessaria una configurazione aggiuntiva per i ruoli RBAC personalizzati di Kubernetes. Oltre all'RBAC nativo di Kubernetes, prendi in considerazione l'utilizzo di Kyverno per la gestione avanzata delle autorizzazioni nei cluster EKS.

Opzione 2: AWS IAM Users/Roles mappato su gruppi Kubernetes

Pro:

1. Controllo granulare sulle autorizzazioni IAM.
2. Ruolo prevedibile e statico ARNs

Contro:

1. Maggiore sovraccarico di gestione per gli account utente
2. Mancanza di gestione centralizzata delle identità
3. Potenziale di proliferazione delle entità IAM

Migliori pratiche:

1. Utilizza i ruoli IAM anziché gli utenti IAM per migliorare la sicurezza e la gestibilità
2. Implementa una convenzione di denominazione per i ruoli per garantire coerenza e facilità di gestione
3. Utilizza le condizioni delle policy IAM per limitare l'accesso in base a tag o altri attributi.
4. Ruota regolarmente le chiavi di accesso e rivedi le autorizzazioni.

Considerazioni/limitazioni:

1. Problemi di scalabilità nella gestione di un numero elevato di utenti o ruoli
2. Nessuna funzionalità Single Sign-On integrata

Opzione 3: fornitori OIDC

Pro:

1. Integrazione con i sistemi di gestione delle identità esistenti
2. Sovraccarico di gestione ridotto per gli account utente

Contro:

1. Complessità di configurazione aggiuntiva
2. Potenziale aumento della latenza durante l'autenticazione
3. Dipendenza da un provider di identità esterno

Migliori pratiche:

1. Configura con attenzione il provider OIDC per garantire una convalida sicura dei token.
2. Utilizza token di breve durata e implementa meccanismi di aggiornamento dei token.
3. Controlla e aggiorna regolarmente le configurazioni OIDC.

Consulta questa guida per un'implementazione di riferimento sull'integrazione di provider Single Sign-On esterni con Amazon EKS

Considerazioni/limitazioni:

1. Integrazione nativa limitata con i servizi AWS rispetto a IAM.
2. L'URL dell'emittente del provider OIDC deve essere accessibile al pubblico affinché EKS possa scoprire le chiavi di firma.

AWS EKS Pod Identity e IRSA per carichi di lavoro

Amazon EKS offre due modi per concedere le autorizzazioni AWS IAM ai carichi di lavoro eseguiti nei cluster Amazon EKS: ruoli IAM per account di servizio (IRSA) e EKS Pod Identities.

Sebbene sia IRSA che EKS Pod Identities offrano i vantaggi dell'accesso con privilegi minimi, dell'isolamento delle credenziali e della verificabilità, EKS Pod Identity è il metodo consigliato per concedere le autorizzazioni ai carichi di lavoro.

Per una guida dettagliata sull'identità e le credenziali per i pod EKS, consulta la sezione Identità e credenziali delle migliori pratiche di sicurezza.

Raccomandazione

Combina IAM Identity Center con l'API CAM

- Gestione semplificata: utilizzando l'API Cluster Access Management in combinazione con IAM Identity Center, gli amministratori possono gestire l'accesso al cluster EKS insieme ad altri servizi AWS, riducendo la necessità di passare da un'interfaccia all'altra o di modificare manualmente. ConfigMaps
- Utilizza le voci di accesso per gestire le autorizzazioni Kubernetes dei principali IAM dall'esterno del cluster. Puoi aggiungere e gestire l'accesso al cluster utilizzando l'API EKS, l'interfaccia a riga di comando AWS, AWS SDKs CloudFormation, AWS e la console di gestione AWS. Ciò significa che puoi gestire gli utenti con gli stessi strumenti con cui hai creato il cluster.
- Le autorizzazioni granulari di Kubernetes possono essere applicate mappando utenti o gruppi Kubernetes con principi IAM associati alle identità SSO tramite voci di accesso e policy di accesso.
- Per iniziare, segui Cambia la modalità di autenticazione per utilizzare le voci di accesso, quindi Migrazione delle voci aws-auth esistenti per accedere alle voci. ConfigMap

Procedure consigliate per la scalabilità automatica dei cluster

Questa guida fornisce consigli su Cluster Autoscaling, incluse indicazioni per Auto Mode, Karpenter e Kubernetes Cluster Autoscaler.

Argomenti

- Modalità automatica di EKS
- Karpenter
- Cluster Autoscaler

Modalità automatica di EKS

> **Tip**
>
> Esplora le best practice tramite i workshop Amazon EKS.

Amazon EKS Auto Mode rappresenta un'evoluzione significativa nella gestione dell'infrastruttura Kubernetes, combinando un'infrastruttura cluster sicura e scalabile con funzionalità Kubernetes integrate gestite da AWS. Il servizio fornisce operazioni completamente gestite sui nodi di lavoro, eliminando la necessità per i clienti di configurare gruppi o gruppi di nodi gestiti. AutoScaling

La differenza architetturale fondamentale è che EKS Auto Mode utilizza un sistema basato su Karpenter che effettua automaticamente il provisioning delle istanze EC2 in risposta alle richieste dei pod. Queste istanze vengono eseguite su Bottlerocket AMIs con componenti aggiuntivi preinstallati come i driver CSI EBS, che rendono l'infrastruttura realmente gestita da AWS. A differenza dei metodi di scalabilità tradizionali:

- Il tradizionale Cluster Autoscaler (CAS) richiede la gestione manuale dei gruppi di nodi e può creare solo nodi con un singolo tipo di istanza per gruppo di nodi
- Karpenter autogestito offre maggiore flessibilità lavorando con l'API EC2 Fleet e può fornire diversi tipi di istanze, ma richiede la gestione dei clienti
- EKS Auto Mode gestisce automaticamente tutte le operazioni di scalabilità tramite sistemi gestiti e NodePools NodeClasses

Il nuovo sistema introduce diversi miglioramenti operativi:

- Scalabilità automatica basata su pod senza configurazione manuale del gruppo di nodi
- Controller di bilanciamento del carico gestiti integrati che vengono creati automaticamente in base alle risorse Ingress ALB/NLB
- Funzionalità di sicurezza integrate con identità Pod preconfigurata
- Durata massima del nodo di 21 giorni con sostituzione automatica

Dal punto di vista dei costi, EKS Auto Mode mantiene i prezzi standard di EC2 aggiungendo una commissione di gestione solo per i nodi gestiti in modalità automatica. È importante sottolineare che i clienti possono ancora combinare nodi gestiti in modalità automatica con nodi autogestiti nello stesso cluster.

Sebbene AWS gestisca la maggior parte degli aspetti operativi, i clienti mantengono la responsabilità della gestione delle versioni dei cluster e possono eseguire aggiornamenti controllati che attivano aggiornamenti continui dei nodi di lavoro.

Motivi per utilizzare la modalità automatica

La modalità automatica è pensata per gli utenti che desiderano i vantaggi di Kubernetes ed EKS ma devono ridurre al minimo gli oneri operativi legati a Kubernetes, come gli aggiornamenti, e gli elementi installation/maintenance critici della piattaforma come l'auto-scaling, il bilanciamento del carico e lo storage. Auto Mode fa un ulteriore passo avanti con EKS nella riduzione al minimo del carico di lavoro indifferenziato associato alla manutenzione di Kubernetes

Domande frequenti

Qual è la differenza tra EKS Auto Mode e Open Source Karpenter?

EKS Auto Mode è un'ampia suite di funzionalità che semplificano l'esecuzione di Kubernetes di livello di produzione. Una di queste funzionalità sono i vantaggi dell'auto-scaling di Karpenter, completamente gestito. Dal punto di vista operativo, l'unica differenza è che in EKS Auto Mode non è necessario gestire autonomamente l'implementazione, la scalabilità e l'aggiornamento dei pod Karpenter. Tutte le altre operazioni sono gestite NodeClasses e NodePools funzionano allo stesso modo dell'open source Karpenter.

Posso eseguire gruppi di nodi gestiti insieme a nodi gestiti in modalità automatica?

Sì, è possibile eseguire nodi statici tramite gruppi di nodi gestiti insieme ai nodi con scalabilità automatica forniti con la modalità Auto

Posso migrare un cluster dalla modalità EKS standard a EKS Auto Mode?

Sì, le istruzioni per abilitare la modalità automatica EKS su un cluster esistente sono disponibili nella documentazione ufficiale di AWS

Cose da notare: 1. Dopo aver abilitato la modalità automatica, ti consigliamo di disinstallare tutti i componenti che hai installato e che ora sono gestiti da Auto Mode, come Karpenter o AWS Load Balancer Controller 2. Devi assicurarti che i componenti aggiuntivi installati lo siano. up-to-date Consulta la documentazione.

Come posso configurare NodePools in modalità automatica EKS?

Un nuovo cluster verrà preconfigurato con due NodePools

per uso generico

```yaml
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  generation: 1
  labels:
    app.kubernetes.io/managed-by: eks
  name: general-purpose
spec:
  disruption:
    budgets:
      - nodes: 10%
    consolidateAfter: 30s
    consolidationPolicy: WhenEmptyOrUnderutilized
  template:
    spec:
      expireAfter: 336h
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: default
      requirements:
        - key: karpenter.sh/capacity-type
          operator: In
          values:
            - on-demand
        - key: eks.amazonaws.com/instance-category
          operator: In
          values:
            - c
            - m
            - r
        - key: eks.amazonaws.com/instance-generation
          operator: Gt
          values:
            - '4'
        - key: kubernetes.io/arch
          operator: In
          values:
            - amd64
        - key: kubernetes.io/os
          operator: In
          values:
            - linux
      terminationGracePeriod: 24h0m0s
```

Questo NodePool indica a Karpenter di avviare nodi con le seguenti caratteristiche:

1. Tipo di capacità «On Demand»
2. Tipi di istanza di C, M o R
3. Generazione di 4 istanze
4. Architettura AMD
5. Sistema operativo Linux

Definisce inoltre cos'è la logica di ridimensionamento dichiarando che solo il 10% di tutti i nodi può trovarsi in uno stato interrotto in un dato momento e che il consolidamento dovrebbe avvenire solo quando i nodi sono vuoti o sottoutilizzati.

sistema

```yaml
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  generation: 1
  labels:
    app.kubernetes.io/managed-by: eks
  name: system
spec:
  disruption:
    budgets:
      - nodes: 10%
    consolidateAfter: 30s
    consolidationPolicy: WhenEmptyOrUnderutilized
  template:
    spec:
      expireAfter: 336h
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: default
      requirements:
        - key: karpenter.sh/capacity-type
          operator: In
          values:
            - on-demand
        - key: eks.amazonaws.com/instance-category
          operator: In
          values:
            - c
            - m
            - r
        - key: eks.amazonaws.com/instance-generation
          operator: Gt
          values:
            - '4'
        - key: kubernetes.io/arch
          operator: In
          values:
            - amd64
            - arm64
        - key: kubernetes.io/os
          operator: In
          values:
            - linux
      taints:
        - effect: NoSchedule
```

NodePool È simile a «uso generico», ad eccezione delle seguenti differenze:

1. Consente l'utilizzo di nodi con architettura ARM e architettura AMD
2. Contamina questi nodi con un NoSchedule a a meno che non ci sia una tolleranza per ««CriticalAddonsOnly. È destinato all'uso interno dei componenti aggiuntivi EKS

personalizzato

È possibile crearne di personalizzati in NodePools base alle proprie esigenze. Per saperne di più, NodePools consulta la documentazione di Karpenter.

Posso personalizzare l'AMI utilizzata da Auto Mode quando vengono lanciati nuovi nodi?

No, al momento gli unici supporti AMIs sono per Bottlerocket fornito da Amazon

Come posso installare strumenti o agenti personalizzati sui miei host Kubernetes?

Poiché la personalizzazione AMI non è supportata, se hai bisogno di un software a livello di host per cose come la scansione di sicurezza, dovresti distribuire il carico di lavoro come Kubernetes. DaemonSet

Quali componenti sono in esecuzione nel piano dati del mio cluster quando eseguo il provisioning di un nuovo cluster EKS Auto Mode?

Se crei un cluster con eksctl o la console AWS, gli unici pod in esecuzione in un cluster EKS Auto Mode sono i pod Kubernetes Metrics Server. Gli altri componenti di EKS Auto Mode come Karpenter, AWS Load Balancer Controller e EBS CSI Driver sono tutti in esecuzione e gestiti fuori dal cluster.

Quali componenti gestiti sono in esecuzione per supportare il mio nuovo cluster EKS Auto Mode?

EKS Auto Mode automatizza completamente l'implementazione della maggior parte delle parti di un piano dati necessarie per Kubernetes di livello di produzione. Questo include:

- Karpenter, per l'auto-scaling del calcolo del cluster
- AWS Load Balancer Controller per consentire di esporre facilmente i servizi Kubernetes tramite l'integrazione automatizzata di Elastic Load Balancer

- WEBS CSI
- PATTUMIERA IN PVC
- EKS Pod Identity Agent

Come posso risolvere i componenti della modalità automatica che prima funzionavano come pod nel mio cluster?

Con EKS Auto Mode, molti componenti come AWS Load Balancer Controller e Karpenter vengono gestiti per te all'esterno del cluster, quindi non avrai la stessa visibilità sui log a cui sei abituato quando gestisci in modo autonomo. Se ti trovi in una situazione in cui devi risolvere la funzionalità di una funzionalità della modalità automatica, crea un AWS Support Ticket.

Karpenter

> **Tip**
>
> Esplora le best practice tramite i workshop Amazon EKS.

Karpenter è un progetto open source progettato per migliorare la gestione del ciclo di vita dei nodi all'interno dei cluster Kubernetes. Automatizza il provisioning e il deprovisioning dei nodi in base alle esigenze di pianificazione specifiche dei pod, consentendo una scalabilità efficiente e l'ottimizzazione dei costi. Le sue funzioni principali sono:

- Monitora i pod che lo scheduler Kubernetes non è in grado di pianificare a causa di limiti di risorse.
- Valuta i requisiti di pianificazione (richieste di risorse, selettori di nodi, affinità, tolleranze, ecc.) dei pod non programmabili.
- Fornisci nuovi nodi che soddisfino i requisiti di tali pod.
- Rimuovi i nodi quando non sono più necessari.

Con Karpenter, puoi definire NodePools vincoli sul provisioning dei nodi come contaminazioni, etichette, requisiti (tipi di istanze, zone, ecc.) e limiti sulle risorse totali assegnate. Quando si distribuiscono carichi di lavoro, è possibile specificare vari vincoli di pianificazione nelle specifiche del pod, come affinità delle risorse, tolleranze e vincoli di diffusione della topologia. requests/limits, node selectors, node/pod Karpenter fornirà quindi nodi della giusta dimensione in base a queste specifiche.

Motivi per utilizzare Karpenter

Prima del lancio di Karpenter, gli utenti di Kubernetes si affidavano principalmente ai gruppi Amazon EC2 Auto Scaling e al Kubernetes Cluster Autoscaler (CAS) per regolare dinamicamente la capacità di elaborazione dei propri cluster. Con Karpenter, non è necessario creare dozzine di gruppi di nodi per ottenere la flessibilità e la diversità offerte da Karpenter. A differenza di CAS, Karpenter non è così strettamente associato alle versioni di Kubernetes e non richiede il passaggio da AWS a Kubernetes. APIs

Karpenter consolida le responsabilità di orchestrazione delle istanze all'interno di un unico sistema, che è più semplice, stabile e compatibile con i cluster. Karpenter è stato progettato per superare alcune delle sfide presentate da Cluster Autoscaler fornendo modi semplificati per:

- Fornisci nodi in base ai requisiti del carico di lavoro.
- Crea configurazioni di nodi diverse per tipo di istanza, utilizzando opzioni flessibiliNodePool . Invece di gestire molti gruppi di nodi personalizzati specifici, Karpenter potrebbe consentirti di gestire diverse capacità di carico di lavoro con un'unica soluzione flessibile. NodePool
- Ottieni una migliore pianificazione dei pod su larga scala avviando rapidamente nodi e pianificando i pod.

Per informazioni e documentazione sull'uso di Karpenter, visitate il sito karpenter.sh.

Raccomandazioni

Le migliori pratiche sono suddivise in sezioni relative a Karpenter stesso e alla pianificazione dei pod NodePools.

Le migliori pratiche di Karpenter

Le seguenti best practice riguardano argomenti relativi a Karpenter stesso.

Blocca AMIs nei cluster di produzione

Ti consigliamo vivamente di aggiungere le note Amazon Machine Images (AMIs) utilizzate da Karpenter per i cluster di produzione. L'utilizzo `amiSelector` con un alias impostato su o un altro metodo che comporti l'implementazione non testata AMIs man mano che vengono rilasciati, comporta il rischio di errori del carico di lavoro e tempi di inattività nei cluster di produzione. `@latest` Di conseguenza, consigliamo vivamente di aggiungere versioni funzionanti testate di AMIs per i cluster

di produzione mentre si testano le versioni più recenti in cluster non di produzione. Ad esempio, è possibile impostare un alias nel modo seguente: NodeClass

```
amiSelectorTerms
  - alias: al2023@v20240807
```

Per informazioni sulla gestione e il blocco AMIs in Karpenter, consulta Managing AMIs nella documentazione di Karpenter.

Usa Karpenter per carichi di lavoro con esigenze di capacità mutevoli

Karpenter avvicina la gestione della scalabilità a APIs quella nativa di Kubernetes rispetto ad Autoscaling Groups () e Managed Node Groups (MNG). ASGs ASG e MNG sono astrazioni native di AWS in cui la scalabilità viene attivata in base a metriche a livello di AWS, come il carico della CPU EC2. Cluster Autoscaler collega le astrazioni Kubernetes alle astrazioni AWS, ma per questo motivo perde una certa flessibilità, ad esempio la pianificazione per una zona di disponibilità specifica.

Karpenter rimuove un livello di astrazione AWS per portare parte della flessibilità direttamente in Kubernetes. Karpenter è ideale per cluster con carichi di lavoro che incontrano periodi di domanda elevata e impennata o hanno requisiti di elaborazione diversi. MNGs e ASGs sono ideali per i cluster che eseguono carichi di lavoro che tendono ad essere più statici e coerenti. È possibile utilizzare una combinazione di nodi gestiti dinamicamente e staticamente, a seconda delle esigenze.

Prendi in considerazione altri progetti di scalabilità automatica quando...

Hai bisogno di funzionalità che sono ancora in fase di sviluppo in Karpenter. Poiché Karpenter è un progetto relativamente nuovo, per il momento prendi in considerazione altri progetti di scalabilità automatica se hai bisogno di funzionalità che non fanno ancora parte di Karpenter.

Esegui il controller Karpenter su EKS Fargate o su un nodo di lavoro che appartiene a un gruppo di nodi

Karpenter viene installato utilizzando un grafico Helm. Il grafico Helm installa il controller Karpenter e un pod webhook come implementazione che deve essere eseguita prima che il controller possa essere utilizzato per scalare il cluster. Consigliamo un minimo di un piccolo gruppo di nodi con almeno un nodo di lavoro. In alternativa, è possibile eseguire questi pod su EKS Fargate creando un profilo Fargate per il namespace. `karpenter` In questo modo tutti i pod distribuiti in questo spazio dei nomi verranno eseguiti su EKS Fargate. Non eseguite Karpenter su un nodo gestito da Karpenter.

Karpenter non supporta modelli di avvio personalizzati

Non è disponibile il supporto per i modelli di avvio personalizzati con la v1. APIs È possibile utilizzare dati utente personalizzati specificando and/or direttamente custom AMIs in. EC2 NodeClass Ulteriori informazioni su come eseguire questa operazione sono disponibili all'indirizzo NodeClasses.

Escludi i tipi di istanze che non si adattano al tuo carico di lavoro

Valuta la possibilità di escludere tipi di istanze specifici con la `node.kubernetes.io/instance-type` chiave se non sono richiesti dai carichi di lavoro in esecuzione nel cluster.

L'esempio seguente mostra come evitare il provisioning di istanze Graviton di grandi dimensioni.

```
- key: node.kubernetes.io/instance-type
  operator: NotIn
  values:
  - m6g.16xlarge
  - m6gd.16xlarge
  - r6g.16xlarge
  - r6gd.16xlarge
  - c6g.16xlarge
```

Abilita la gestione delle interruzioni quando usi Spot

Karpenter supporta la gestione nativa delle interruzioni ed è in grado di gestire eventi di interruzione involontaria come interruzioni delle istanze Spot, eventi di manutenzione programmata, eventi di terminazione/arresto delle istanze che potrebbero interrompere i carichi di lavoro. Quando Karpenter rileva tali eventi per i nodi, contamina, prosciuga e chiude automaticamente i nodi interessati in anticipo per avviare una corretta pulizia dei carichi di lavoro prima dell'interruzione. In caso di interruzioni Spot con un preavviso di 2 minuti, Karpenter avvia rapidamente un nuovo nodo in modo che i pod possano essere spostati prima che l'istanza venga recuperata. Per abilitare la gestione delle interruzioni, configurate l'argomento `--interruption-queue` CLI con il nome della coda SQS fornita a tale scopo. Non è consigliabile utilizzare la gestione delle interruzioni di Karpenter insieme a Node Termination Handler, come spiegato qui.

I pod che richiedono un checkpoint o altre forme di drenaggio graduale, che richiedono 2 minuti prima dello spegnimento, dovrebbero consentire a Karpenter di gestire le interruzioni all'interno dei propri cluster.

Cluster privato Amazon EKS senza accesso a Internet in uscita

Quando si effettua il provisioning di un cluster EKS in un VPC senza percorso verso Internet, è necessario assicurarsi di aver configurato l'ambiente in base ai requisiti del cluster privato indicati nella documentazione EKS. Inoltre, devi assicurarti di aver creato un endpoint regionale VPC STS nel tuo VPC. In caso contrario, verranno visualizzati errori simili a quelli visualizzati di seguito.

```
{"level":"FATAL","time":"2024-02-29T14:28:34.392Z","logger":"controller","message":"Checking
 EC2 API connectivity, WebIdentityErr: failed to retrieve credentials
\ncaused by: RequestError: send request failed\ncaused by: Post
 \"https://sts.<region>.amazonaws.com/\": dial tcp 54.239.32.126:443: i/o
 timeout","commit":"596ea97"}
```

Queste modifiche sono necessarie in un cluster privato perché il controller Karpenter utilizza IAM Roles for Service Accounts (IRSA). I pod configurati con IRSA acquisiscono le credenziali chiamando l'API AWS Security Token Service (AWS STS). Se non è disponibile un accesso a Internet in uscita, devi creare e utilizzare un endpoint VPC AWS STS nel tuo VPC.

I cluster privati richiedono anche la creazione di un endpoint VPC per SSM. Quando Karpenter tenta di effettuare il provisioning di un nuovo nodo, interroga le configurazioni del modello Launch e un parametro SSM. Se non hai un endpoint VPC SSM nel tuo VPC, causerà il seguente errore:

```
{"level":"ERROR","time":"2024-02-29T14:28:12.889Z","logger":"controller","message":"Unable
 to hydrate the AWS launch template cache, RequestCanceled: request context canceled
\ncaused by: context canceled","commit":"596ea97","tag-key":"karpenter.k8s.aws/
cluster","tag-value":"eks-workshop"}
...
{"level":"ERROR","time":"2024-02-29T15:08:58.869Z","logger":"controller.nodeclass","message":"d
 amis from ssm, getting ssm parameter \"/aws/service/eks/optimized-ami/1.27/amazon-
linux-2/recommended/image_id\", RequestError: send request failed\ncaused by:
 Post \"https://ssm.<region>.amazonaws.com/\": dial tcp 67.220.228.252:443: i/o
 timeout","commit":"596ea97","ec2nodeclass":"default","query":"/aws/service/eks/
optimized-ami/1.27/amazon-linux-2/recommended/image_id"}
```

Non esiste un endpoint VPC per l'API Price List Query. Di conseguenza, i dati sui prezzi diventeranno obsoleti nel tempo. Karpenter aggira questo problema includendo i dati sui prezzi su richiesta nel suo file binario, ma aggiorna tali dati solo quando Karpenter viene aggiornato. Le richieste non riuscite di dati sui prezzi genereranno i seguenti messaggi di errore:

```
{"level":"ERROR","time":"2024-02-29T15:08:58.522Z","logger":"controller.pricing","message":"ret
 on-demand pricing data, RequestError: send request failed\ncaused by: Post
```

```
 \"https://api.pricing.<region>.amazonaws.com/\": dial tcp 18.196.224.8:443:
 i/o timeout; RequestError: send request failed\ncaused by: Post \"https://
api.pricing.<region>.amazonaws.com/\": dial tcp 18.185.143.117:443: i/o
 timeout","commit":"596ea97"}
```

Fai riferimento a questa documentazione per utilizzare Karpenter in un cluster EKS completamente privato e per sapere quali endpoint VPC creare.

Creando NodePools

Le seguenti best practice riguardano argomenti relativi alla creazione NodePools.

Crea più di uno NodePools quando...

Quando team diversi condividono un cluster e devono eseguire i propri carichi di lavoro su nodi di lavoro diversi o hanno requisiti di sistema operativo o tipo di istanza diversi, creane diversi NodePools. Ad esempio, un team potrebbe voler usare Bottlerocket, mentre un altro potrebbe voler usare Amazon Linux. Allo stesso modo, un team potrebbe avere accesso a costosi hardware GPU che non sarebbero necessari a un altro team. L'utilizzo di più risorse NodePools assicura che le risorse più appropriate siano disponibili per ogni team.

Creazioni NodePools che si escludano a vicenda o ponderate

Si consiglia di creare creazioni NodePools che si escludano a vicenda o ponderate per fornire un comportamento di pianificazione coerente. Se non lo sono e ne NodePools vengono abbinate più di una, Karpenter sceglierà a caso quali usare, generando risultati inaspettati. Di seguito sono riportati alcuni esempi utili per la creazione di più elementi: NodePools

Creare una NodePool con GPU e consentire l'esecuzione solo di carichi di lavoro speciali su questi (costosi) nodi:

```
# NodePool for GPU Instances with Taints
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: gpu
spec:
  disruption:
    consolidateAfter: 1m
    consolidationPolicy: WhenEmptyOrUnderutilized
  template:
```

```
    metadata: {}
    spec:
      nodeClassRef:
        group: karpenter.k8s.aws
        kind: EC2NodeClass
        name: default
      expireAfter: Never
      requirements:
      - key: node.kubernetes.io/instance-type
        operator: In
        values:
        - p3.8xlarge
        - p3.16xlarge
      - key: kubernetes.io/os
        operator: In
        values:
        - linux
      - key: kubernetes.io/arch
        operator: In
        values:
        - amd64
      - key: karpenter.sh/capacity-type
        operator: In
        values:
        - on-demand
      taints:
      - effect: NoSchedule
        key: nvidia.com/gpu
        value: "true"
```

Implementazione con tolleranza alla contaminazione:

```
# Deployment of GPU Workload will have tolerations defined
apiVersion: apps/v1
kind: Deployment
metadata:
  name: inflate-gpu
spec:
    spec:
      tolerations:
      - key: "nvidia.com/gpu"
        operator: "Exists"
        effect: "NoSchedule"
```

Per una distribuzione generale per un altro team, le NodePool specifiche potrebbero includere NodeAffinity. Un Deployment potrebbe quindi essere utilizzato per abbinare. nodeSelectorTerms `billing-team`

```
# NodePool for regular EC2 instances
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: generalcompute
spec:
  template:
    metadata:
      labels:
        billing-team: my-team
    spec:
      nodeClassRef:
        group: karpenter.k8s.aws
        kind: EC2NodeClass
        name: default
      expireAfter: Never
      requirements:
      - key: node.kubernetes.io/instance-type
        operator: In
        values:
        - m5.large
        - m5.xlarge
        - m5.2xlarge
        - c5.large
        - c5.xlarge
        - c5a.large
        - c5a.xlarge
        - r5.large
        - r5.xlarge
      - key: kubernetes.io/os
        operator: In
        values:
        - linux
      - key: kubernetes.io/arch
        operator: In
        values:
        - amd64
      - key: karpenter.sh/capacity-type
        operator: In
```

```
        values:
        - on-demand
```

Distribuzione tramite NodeAffinity:

```
# Deployment will have spec.affinity.nodeAffinity defined
kind: Deployment
metadata:
  name: workload-my-team
spec:
  replicas: 200
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
              - matchExpressions:
                - key: "billing-team"
                  operator: "In"
                  values: ["my-team"]
```

Utilizza i timer (TTL) per eliminare automaticamente i nodi dal cluster

È possibile utilizzare i timer sui nodi assegnati per impostare quando eliminare i nodi privi di pod per il carico di lavoro o che hanno raggiunto un orario di scadenza. La scadenza dei nodi può essere utilizzata come mezzo di aggiornamento, in modo che i nodi vengano ritirati e sostituiti con versioni aggiornate. Vedi Scadenza nella documentazione di Karpenter per informazioni sull'utilizzo per configurare la scadenza dei nodi. `spec.template.spec`

Evita di limitare eccessivamente i tipi di istanze che Karpenter può fornire, specialmente quando utilizzi Spot

Quando utilizza Spot, Karpenter utilizza la strategia di allocazione Price Capacity Optimized per effettuare il provisioning delle istanze EC2. Questa strategia prevede che EC2 effettui il provisioning delle istanze provenienti dai pool più profondi per il numero di istanze che stai avviando e che presentano il minor rischio di interruzione. EC2 Fleet richiede quindi le istanze Spot dal prezzo più basso di questi pool. Più tipi di istanze consenti a Karpenter di utilizzare, migliore è la capacità di EC2 di ottimizzare il runtime dell'istanza spot. Per impostazione predefinita, Karpenter utilizzerà tutti i tipi di istanza offerti da EC2 nella regione e nelle zone di disponibilità in cui è distribuito il cluster. Karpenter sceglie in modo intelligente dal set di tutti i tipi di istanze in base ai pod in sospeso per assicurarsi che

i pod siano programmati su istanze di dimensioni e equipaggiate in modo appropriato. Ad esempio, se il tuo pod non richiede una GPU, Karpenter non lo programmerà su un tipo di istanza EC2 che supporti una GPU. Se non sei sicuro dei tipi di istanze da utilizzare, puoi eseguire Amazon ec2-instance-selector per generare un elenco di tipi di istanze che soddisfano i tuoi requisiti di calcolo. Ad esempio, la CLI utilizza memoria, vCPU, architettura e regione come parametri di input e fornisce un elenco di istanze EC2 che soddisfano tali vincoli.

```
$ ec2-instance-selector --memory 4 --vcpus 2 --cpu-architecture x86_64 -r ap-
southeast-1
c5.large
c5a.large
c5ad.large
c5d.large
c6i.large
t2.medium
t3.medium
t3a.medium
```

Non dovresti imporre troppi vincoli a Karpenter quando usi le istanze Spot perché ciò può influire sulla disponibilità delle tue applicazioni. Supponiamo, ad esempio, che tutte le istanze di un particolare tipo vengano recuperate e che non siano disponibili alternative adeguate per sostituirle. I pod rimarranno in sospeso fino a quando non verrà ripristinata la capacità spot per i tipi di istanze configurati. È possibile ridurre il rischio di errori di capacità insufficiente distribuendo le istanze su diverse zone di disponibilità, poiché i pool spot sono diversi tra loro. AZs Detto questo, la best practice generale consiste nel consentire a Karpenter di utilizzare una serie diversificata di tipi di istanze quando utilizza Spot.

Scheduling Pods

Le seguenti best practice riguardano la distribuzione dei pod in un cluster utilizzando Karpenter per il provisioning dei nodi.

Segui le migliori pratiche EKS per un'elevata disponibilità

Se devi eseguire applicazioni ad alta disponibilità, segui le raccomandazioni generali sulle migliori pratiche EKS. Consultate la documentazione Topology Spread in Karpenter per i dettagli su come distribuire i pod tra nodi e zone. Usa Disruption Budgets per impostare il numero minimo di pod disponibili che devono essere mantenuti, in caso di tentativi di rimuovere o eliminare i pod.

Usa vincoli a più livelli per limitare le funzionalità di elaborazione disponibili dal tuo provider di servizi cloud

Il modello di vincoli a più livelli di Karpenter ti consente di creare un insieme complesso di vincoli di implementazione NodePool e pod per ottenere le migliori corrispondenze possibili per la pianificazione dei pod. Di seguito sono riportati alcuni esempi di vincoli che una specifica del pod può richiedere:

- È necessario eseguire in zone di disponibilità in cui sono disponibili solo applicazioni particolari. Supponiamo, ad esempio, di avere un pod che deve comunicare con un'altra applicazione in esecuzione su un'istanza EC2 che risiede in una particolare zona di disponibilità. Se il tuo obiettivo è ridurre il traffico cross-AZ nel tuo VPC, potresti voler collocare i pod nell'AZ in cui si trova l'istanza EC2. Questo tipo di targeting viene spesso realizzato utilizzando selettori di nodi. Per ulteriori informazioni sui selettori di nodi, consulta la documentazione di Kubernetes.
- Richiede determinati tipi di processori o altro hardware. Consulta la sezione Acceleratori dei documenti di Karpenter per un esempio di specifica del pod che richiede l'esecuzione del pod su una GPU.

Crea allarmi di fatturazione per monitorare la spesa di elaborazione

Quando configuri il cluster per la scalabilità automatica, devi creare allarmi di fatturazione per avvisarti quando la spesa supera una soglia e aggiungere limiti di risorse alla configurazione di Karpenter. L'impostazione dei limiti delle risorse con Karpenter è simile all'impostazione della capacità massima di un gruppo di autoscaling AWS in quanto rappresenta la quantità massima di risorse di calcolo che possono essere istanziate da un Karpenter. NodePool

> **Note**
>
> Non è possibile impostare un limite globale per l'intero cluster. I limiti si applicano a situazioni specifiche NodePools.

Lo snippet riportato di seguito indica a Karpenter di fornire solo un massimo di 1000 core CPU e 1000 Gi di memoria. Karpenter smetterà di aggiungere capacità solo quando il limite viene raggiunto o superato. Quando viene superato un limite, il controller Karpenter `memory resource usage of 1001 exceeds limit of 1000` scriverà un messaggio dall'aspetto simile nei log del controller. Se state indirizzando i log dei container verso i CloudWatch log, potete creare un filtro metrico per

cercare modelli o termini specifici nei log e quindi creare un CloudWatchallarme per avvisarvi quando la soglia delle metriche configurate viene superata.

Per ulteriori informazioni sull'uso dei limiti con Karpenter, consulta Impostazione dei limiti delle risorse nella documentazione di Karpenter.

```
spec:
  limits:
    cpu: 1000
    memory: 1000Gi
```

Se non utilizzi limiti o vincoli i tipi di istanze che Karpenter può fornire, Karpenter continuerà ad aggiungere capacità di calcolo al cluster in base alle esigenze. La configurazione di Karpenter in questo modo consente al cluster di scalare liberamente, ma può anche avere implicazioni significative in termini di costi. È per questo motivo che consigliamo di configurare gli allarmi di fatturazione. Gli allarmi di fatturazione ti consentono di essere avvisato e avvisato in modo proattivo quando gli addebiti stimati calcolati sul/i tuo/i account superano una soglia definita. Per ulteriori informazioni, consulta Configurazione di un allarme di CloudWatch fatturazione Amazon per monitorare in modo proattivo gli addebiti stimati.

Potresti anche voler abilitare Cost Anomaly Detection, una funzionalità di AWS Cost Management che utilizza l'apprendimento automatico per monitorare continuamente costi e utilizzo per rilevare spese insolite. Ulteriori informazioni sono disponibili nella guida introduttiva di AWS Cost Anomaly Detection. Se sei arrivato al punto di creare un budget in AWS Budgets, puoi anche configurare un'azione per avvisarti quando viene superata una soglia specifica. Con Budget Actions puoi inviare un'e-mail, pubblicare un messaggio su un argomento SNS o inviare un messaggio a un chatbot come Slack. Per ulteriori informazioni, consulta Configurazione delle azioni di AWS Budgets.

Usa l'do-not-disrupt annotazione karpenter.sh/ per impedire a Karpenter di eseguire il deprovisioning di un nodo

Se state eseguendo un'applicazione critica su un nodo fornito da Karpenter, ad esempio un processo batch a esecuzione prolungata o un'applicazione stateful, e il TTL del nodo è scaduto, l'applicazione verrà interrotta quando l'istanza viene terminata. Aggiungendo un'`karpenter.sh/do-not-disrupt`annotazione al pod, si ordina a Karpenter di conservare il nodo fino alla chiusura del Pod o alla rimozione dell'annotazione. `karpenter.sh/do-not-disrupt` Per ulteriori informazioni, consultate la documentazione di Distruption.

Se gli unici pod non daemonset rimasti su un nodo sono quelli associati ai job, Karpenter è in grado di indirizzare e terminare quei nodi a condizione che lo stato del processo sia riuscito o non riuscito.

Configura requests=limits per tutte le risorse diverse dalla CPU quando usi il consolidamento

Il consolidamento e la pianificazione in generale funzionano confrontando le richieste di risorse dei pod con la quantità di risorse allocabili su un nodo. I limiti delle risorse non vengono considerati. Ad esempio, i pod con un limite di memoria superiore alla richiesta di memoria possono superare la richiesta. Se diversi pod sullo stesso nodo si interrompono contemporaneamente, alcuni pod possono essere chiusi a causa di una condizione di esaurimento della memoria (OOM). Il consolidamento può aumentare la probabilità che ciò si verifichi, in quanto consiste nel comprimere i pod nei nodi solo tenendo conto delle loro richieste.

Viene utilizzato LimitRanges per configurare le impostazioni predefinite per le richieste e i limiti delle risorse

Poiché Kubernetes non imposta richieste o limiti predefiniti, il consumo di risorse da parte di un contenitore dall'host, dalla CPU e dalla memoria sottostanti non è vincolato. Lo scheduler Kubernetes esamina le richieste totali di un pod (il più alto tra le richieste totali dai contenitori del pod o il totale delle risorse dai contenitori Init del pod) per determinare su quale nodo di lavoro pianificare il pod. Analogamente, Karpenter prende in considerazione le richieste di un pod per determinare il tipo di istanza da fornire. È possibile utilizzare un intervallo limite per applicare un valore predefinito ragionevole per un namespace, nel caso in cui le richieste di risorse non siano specificate da alcuni pod.

Vedi Configurazione delle richieste e dei limiti di memoria predefiniti per un namespace

Applica richieste di risorse accurate a tutti i carichi di lavoro

Karpenter è in grado di avviare i nodi più adatti ai carichi di lavoro quando le informazioni sui requisiti dei carichi di lavoro sono accurate. Ciò è particolarmente importante se si utilizza la funzionalità di consolidamento di Karpenter.

Vedi Configurazione e dimensione delle risorse Requests/Limits per tutti i carichi di lavoro

Raccomandazioni CoredNS

Aggiorna la configurazione di CoredNS per mantenere l'affidabilità

Quando si implementano i pod CoredNS su nodi gestiti da Karpenter, data la natura dinamica di Karpenter in nuovi nodi terminating/creating rapidamente per allinearsi alla domanda, è consigliabile attenersi alle seguenti best practice:

Durata CoredNS lameduck

Sonda di prontezza CoredNS

Ciò garantirà che le query DNS non vengano indirizzate a un Pod CoredNS che non è ancora pronto o è stato terminato.

Progetti Karpenter

Poiché Karpenter adotta un approccio incentrato sulle applicazioni per fornire capacità di calcolo al piano dati di Kubernetes, ci sono scenari di carico di lavoro comuni che potresti chiederti come configurarli correttamente. Karpenter Blueprints è un repository che include un elenco di scenari di carico di lavoro comuni che seguono le migliori pratiche descritte qui. Avrai a disposizione tutte le risorse necessarie anche per creare un cluster EKS con Karpenter configurato e testare ciascuno dei blueprint inclusi nel repository. Puoi combinare diversi blueprint per creare finalmente quello che ti serve per i tuoi carichi di lavoro.

Risorse aggiuntive

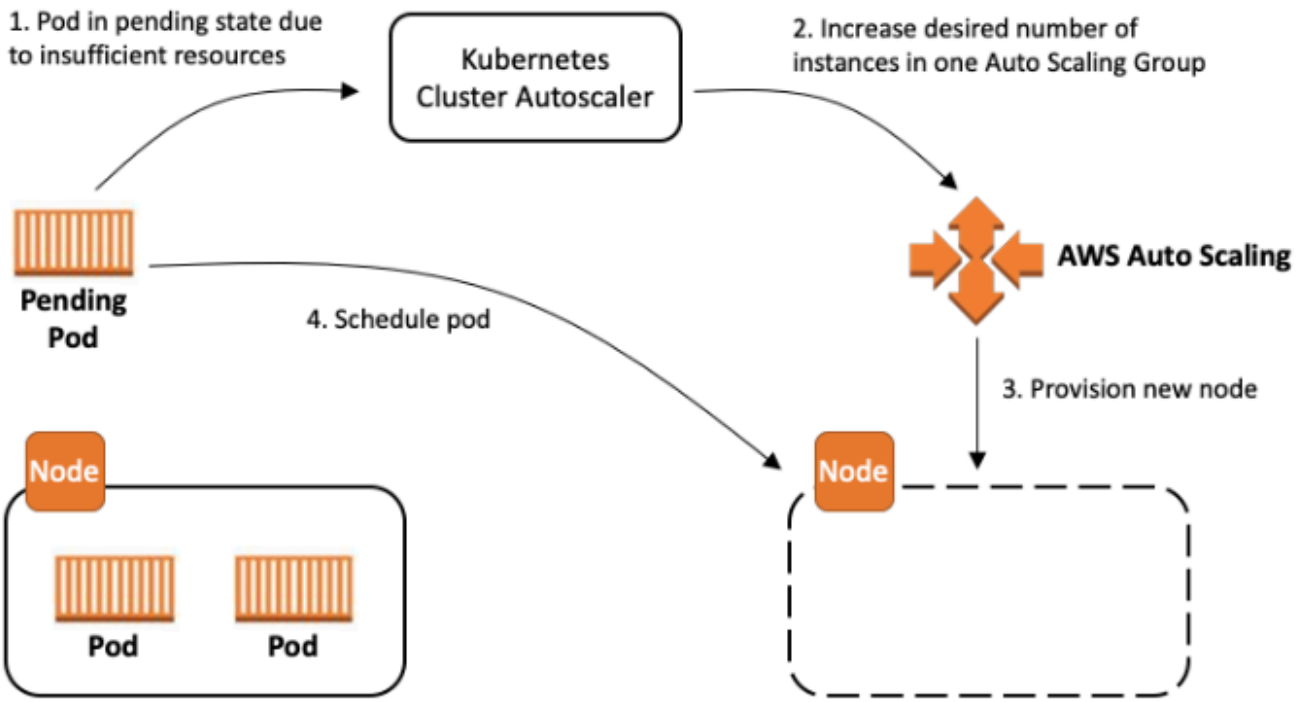
- Workshop Karpenter Immersion Day
- Workshop sull'ottimizzazione dei costi di Karpenter
- Workshop EKS - Karpenter
- Karpenter contro Cluster Autoscaler
- Sessione Karpenter al re:Invent 2023
- Tutorial: Esegui i cluster Kubernetes a un prezzo inferiore con Amazon EC2 Spot e Karpenter

Cluster Autoscaler

> **Tip**
>
> Esplora le best practice tramite i workshop Amazon EKS.

Panoramica di

Kubernetes Cluster Autoscaler è una popolare soluzione Cluster Autoscaling gestita da SIG Autoscaling. È responsabile di garantire che il cluster disponga di nodi sufficienti per pianificare i pod senza sprecare risorse. Rileva i pod che non riescono a programmarsi e i nodi sottoutilizzati. Quindi simula l'aggiunta o la rimozione di nodi prima di applicare la modifica al cluster. L'implementazione di AWS Cloud Provider all'interno di Cluster Autoscaler controlla il `.DesiredReplicas` campo dei tuoi gruppi di Auto Scaling EC2.



Questa guida fornirà un modello mentale per configurare Cluster Autoscaler e scegliere il miglior set di compromessi per soddisfare i requisiti della tua organizzazione. Sebbene non esista un'unica configurazione ottimale, esiste una serie di opzioni di configurazione che consentono di scendere a compromessi tra prestazioni, scalabilità, costi e disponibilità. Inoltre, questa guida fornirà suggerimenti e best practice per ottimizzare la configurazione per AWS.

Glossario

La seguente terminologia verrà utilizzata frequentemente in questo documento. Questi termini possono avere un significato ampio, ma sono limitati alle definizioni riportate di seguito ai fini del presente documento.

La scalabilità si riferisce alle prestazioni di Cluster Autoscaler all'aumentare del numero di pod e nodi del cluster Kubernetes. Quando vengono raggiunti i limiti di scalabilità, le prestazioni e la funzionalità di Cluster Autoscaler diminuiscono. Poiché Cluster Autoscaler supera i limiti di scalabilità, non può più aggiungere o rimuovere nodi nel cluster.

Le prestazioni si riferiscono alla rapidità con cui Cluster Autoscaler è in grado di prendere ed eseguire decisioni di scalabilità. Un Cluster Autoscaler dalle prestazioni perfette prenderebbe istantaneamente una decisione e attiverebbe un'azione di scalabilità in risposta a stimoli, ad esempio se un pod diventasse non programmabile.

La disponibilità significa che i pod possono essere programmati rapidamente e senza interruzioni. Ciò include quando è necessario pianificare i pod appena creati e quando un nodo ridimensionato interrompe tutti i pod rimanenti programmati.

Il costo è determinato dalla decisione alla base degli eventi di scalabilità orizzontale e scalabile in base agli eventi. Le risorse vengono sprecate se un nodo esistente viene sottoutilizzato o viene aggiunto un nuovo nodo troppo grande per i pod in entrata. A seconda del caso d'uso, la chiusura anticipata dei pod può comportare dei costi dovuti a una decisione aggressiva di ridimensionamento.

I gruppi di nodi sono un concetto astratto di Kubernetes per un gruppo di nodi all'interno di un cluster. Non è una vera risorsa Kubernetes, ma esiste come astrazione in Cluster Autoscaler, Cluster API e altri componenti. I nodi all'interno di un gruppo di nodi condividono proprietà come etichette e taint, ma possono essere costituiti da più zone di disponibilità o tipi di istanze.

I gruppi di Auto Scaling EC2 possono essere utilizzati come implementazione dei gruppi di nodi su EC2. I gruppi di Auto Scaling EC2 sono configurati per avviare istanze che si uniscono automaticamente ai rispettivi cluster Kubernetes e applicare etichette e tag alla risorsa Node corrispondente nell'API Kubernetes.

I gruppi di nodi gestiti EC2 sono un'altra implementazione dei gruppi di nodi su EC2. Eliminano la complessità della configurazione manuale di EC2 Autoscaling Scaling Groups e forniscono funzionalità di gestione aggiuntive come l'aggiornamento della versione del nodo e la chiusura graduale dei nodi.

Funzionamento del Cluster Autoscaler

Il Cluster Autoscaler viene in genere installato come implementazione nel cluster. Utilizza l'elezione dei leader per garantire un'elevata disponibilità, ma il lavoro viene svolto da una singola replica alla volta. Non è scalabile orizzontalmente. Per le configurazioni di base, l'impostazione predefinita dovrebbe funzionare immediatamente utilizzando le istruzioni di installazione fornite, ma ci sono alcune cose da tenere a mente.

Verifica che:

- La versione di Cluster Autoscaler corrisponde alla versione del Cluster. La compatibilità tra versioni non è testata o supportata.
- Auto Discovery è abilitato, a meno che non esistano casi d'uso avanzati specifici che impediscono l'uso di questa modalità.

Utilizza l'accesso meno privilegiato al ruolo IAM

Quando si utilizza l'Auto Discovery, si consiglia vivamente di utilizzare l'accesso con privilegi minimi limitando le azioni `autoscaling:SetDesiredCapacity` e `autoscaling:TerminateInstanceInAutoScalingGroup` ai gruppi di Auto Scaling che hanno come ambito il cluster corrente.

Ciò impedirà a un Cluster Autoscaler in esecuzione in un cluster di modificare i gruppi di nodi in un cluster diverso anche se l'`--node-group-auto-discovery`argomento non è stato limitato ai gruppi di nodi del cluster utilizzando i tag (ad esempio). `k8s.io/cluster-autoscaler/<cluster-name>`

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "autoscaling:SetDesiredCapacity",
                "autoscaling:TerminateInstanceInAutoScalingGroup"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/k8s.io/cluster-autoscaler/enabled": "true",
```

```
                    "aws:ResourceTag/k8s.io/cluster-autoscaler/my-cluster": "owned"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "autoscaling:DescribeAutoScalingGroups",
                "autoscaling:DescribeAutoScalingInstances",
                "autoscaling:DescribeLaunchConfigurations",
                "autoscaling:DescribeScalingActivities",
                "autoscaling:DescribeTags",
                "ec2:DescribeImages",
                "ec2:DescribeInstanceTypes",
                "ec2:DescribeLaunchTemplateVersions",
                "ec2:GetInstanceTypesFromInstanceRequirements",
                "eks:DescribeNodegroup"
            ],
            "Resource": "*"
        }
    ]
}
```

Configurazione dei gruppi di nodi

Una scalabilità automatica efficace inizia con la corretta configurazione di un set di gruppi di nodi per il cluster. La selezione del giusto set di gruppi di nodi è fondamentale per massimizzare la disponibilità e ridurre i costi dei carichi di lavoro. AWS implementa i gruppi di nodi utilizzando i gruppi di Auto Scaling EC2, che sono flessibili per un gran numero di casi d'uso. Tuttavia, Cluster Autoscaler formula alcune ipotesi sui gruppi di nodi. Mantenere le configurazioni del gruppo EC2 Auto Scaling coerenti con questi presupposti ridurrà al minimo i comportamenti indesiderati.

Verifica che:

- Ogni nodo in un gruppo di nodi ha proprietà di pianificazione identiche, come Labels, Taints e Resources.
  - Infatti MixedInstancePolicies, i tipi di istanza devono avere la stessa forma per CPU, memoria e GPU
  - Il primo tipo di istanza specificato nella politica verrà utilizzato per simulare la pianificazione.
  - Se la politica prevede tipi di istanze aggiuntivi con più risorse, le risorse potrebbero essere sprecate dopo la scalabilità orizzontale.

  - Se la tua policy prevede tipi di istanze aggiuntivi con meno risorse, i pod potrebbero non riuscire a pianificare le istanze.
- I gruppi di nodi con molti nodi sono preferiti rispetto a molti gruppi di nodi con meno nodi. Questo avrà il maggiore impatto sulla scalabilità.
- Ove possibile, preferisci le funzionalità EC2 quando entrambi i sistemi forniscono supporto (ad esempio regioni,) MixedInstancePolicy

> **Note**
>
> Consigliamo di utilizzare EKS Managed Node Groups. I gruppi di nodi gestiti sono dotati di potenti funzionalità di gestione, tra cui funzionalità per Cluster Autoscaler come l'individuazione automatica dei gruppi di EC2 Auto Scaling e la chiusura graduale dei nodi.

Ottimizzazione per prestazioni e scalabilità

La comprensione della complessità di runtime dell'algoritmo di scalabilità automatica vi aiuterà a ottimizzare Cluster Autoscaler per continuare a funzionare senza problemi in cluster di grandi dimensioni con più di 1.000 nodi.

Le manopole principali per ottimizzare la scalabilità di Cluster Autoscaler sono le risorse fornite al processo, l'intervallo di scansione dell'algoritmo e il numero di gruppi di nodi nel cluster. Esistono altri fattori coinvolti nella vera complessità di runtime di questo algoritmo, come la pianificazione, la complessità dei plugin e il numero di pod. Questi sono considerati parametri non configurabili in quanto sono naturali per il carico di lavoro del cluster e non possono essere regolati facilmente.

Cluster Autoscaler carica in memoria lo stato dell'intero cluster, inclusi pod, nodi e gruppi di nodi. In ogni intervallo di scansione, l'algoritmo identifica i pod non programmabili e simula la pianificazione per ogni gruppo di nodi. La regolazione di questi fattori comporta diversi compromessi che devono essere attentamente considerati in base al caso d'uso.

Scalabilità automatica verticale del Cluster Autoscaler

Il modo più semplice per scalare Cluster Autoscaler su cluster più grandi consiste nell'aumentare le richieste di risorse per la sua implementazione. È necessario aumentare sia la memoria che la CPU per i cluster di grandi dimensioni, sebbene ciò vari in modo significativo a seconda delle dimensioni del cluster. L'algoritmo di scalabilità automatica archivia tutti i pod e i nodi in memoria, il che può

comportare un ingombro di memoria superiore a un gigabyte in alcuni casi. L'aumento delle risorse viene in genere eseguito manualmente. Se ritieni che l'ottimizzazione costante delle risorse crei un onere operativo, prendi in considerazione l'utilizzo di Addon Resizer o Vertical Pod Autoscaler.

Ridurre il numero di gruppi di nodi

Ridurre al minimo il numero di gruppi di nodi è un modo per garantire che Cluster Autoscaler continui a funzionare bene su cluster di grandi dimensioni. Questo può essere difficile per alcune organizzazioni che strutturano i propri gruppi di nodi per team o per applicazione. Sebbene questo sia completamente supportato dall'API Kubernetes, è considerato un anti-pattern di Cluster Autoscaler con ripercussioni sulla scalabilità. Esistono molte ragioni per utilizzare più gruppi di nodi (ad esempio Spot o GPUs), ma in molti casi esistono design alternativi che ottengono lo stesso effetto utilizzando un numero limitato di gruppi.

Verifica che:

- L'isolamento dei pod viene eseguito utilizzando i namespace anziché i gruppi di nodi.
  - Ciò potrebbe non essere possibile nei cluster multi-tenant a bassa attendibilità.
  - Pod ResourceRequests e ResourceLimits sono impostati correttamente per evitare conflitti di risorse.
  - I tipi di istanze più grandi si tradurranno in un imballaggio in contenitori più ottimale e in una riduzione del sovraccarico del sistema per i pod.
- NodeTaints oppure NodeSelectors vengono utilizzati per programmare i pod come eccezione, non come regola.
- Le risorse regionali sono definite come un singolo gruppo di Auto Scaling EC2 con più zone di disponibilità.

Riduzione dell'intervallo di scansione

Un intervallo di scansione basso (ad esempio 10 secondi) assicurerà che Cluster Autoscaler risponda il più rapidamente possibile quando i pod diventano non programmabili. Tuttavia, ogni scansione genera molte chiamate API all'API Kubernetes e al gruppo Auto Scaling EC2 o al gruppo di nodi gestiti EKS. APIs Queste chiamate API possono comportare una limitazione della velocità o addirittura l'indisponibilità del servizio per Kubernetes Control Plane.

L'intervallo di scansione predefinito è di 10 secondi, ma su AWS, l'avvio di un nodo richiede molto più tempo per lanciare una nuova istanza. Ciò significa che è possibile aumentare l'intervallo senza

aumentare significativamente il tempo complessivo di scalabilità. Ad esempio, se occorrono 2 minuti per avviare un nodo, la modifica dell'intervallo a 1 minuto comporterà un compromesso tra chiamate API ridotte di 6 volte e incrementi di scalabilità più lenti del 38%.

Condivisione tra gruppi di nodi

Il Cluster Autoscaler può essere configurato per funzionare su un set specifico di gruppi di nodi. Utilizzando questa funzionalità, è possibile distribuire più istanze di Cluster Autoscaler, ognuna configurata per funzionare su un diverso set di gruppi di nodi. Questa strategia consente di utilizzare un numero arbitrario di gruppi di nodi, scambiando i costi con la scalabilità. Consigliamo di utilizzarla solo come ultima risorsa per migliorare le prestazioni.

Il Cluster Autoscaler non è stato originariamente progettato per questa configurazione, quindi presenta alcuni effetti collaterali. Poiché gli shard non comunicano, è possibile che più autoscaler tentino di pianificare un pod non programmabile. Ciò può comportare una scalabilità non necessaria di più gruppi di nodi. Questi nodi aggiuntivi verranno ridimensionati dopo il`scale-down-delay`.

```
metadata:
  name: cluster-autoscaler
  namespace: cluster-autoscaler-1

...

--nodes=1:10:k8s-worker-asg-1
--nodes=1:10:k8s-worker-asg-2

---

metadata:
  name: cluster-autoscaler
  namespace: cluster-autoscaler-2

...

--nodes=1:10:k8s-worker-asg-3
--nodes=1:10:k8s-worker-asg-4
```

Verifica che:

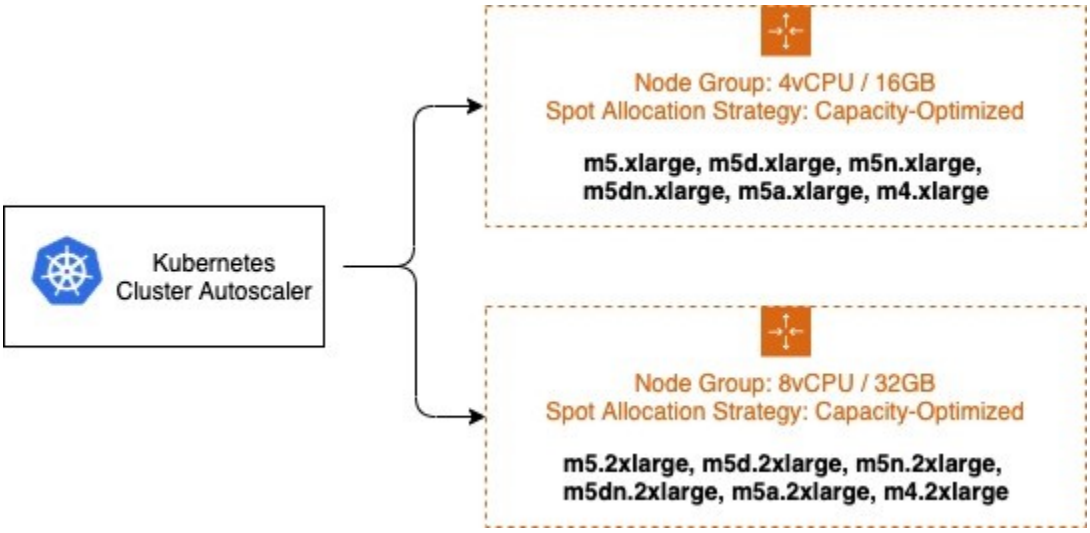
- Ogni shard è configurato per puntare a un set unico di gruppi di Auto Scaling EC2
- Ogni shard viene distribuito in un namespace separato per evitare conflitti elettorali tra i leader

Ottimizzazione dei costi e della disponibilità

Spot Instances

Puoi utilizzare le istanze Spot nei tuoi gruppi di nodi e risparmiare fino al 90% sul prezzo su richiesta. Con un compromesso, le istanze Spot possono essere interrotte in qualsiasi momento quando EC2 ha bisogno di recuperare la capacità. Si verificheranno errori di capacità insufficiente quando il gruppo EC2 Auto Scaling non può aumentare a causa della mancanza di capacità disponibile. Massimizzare la diversità selezionando più famiglie di istanze può aumentare le possibilità di ottenere la scalabilità desiderata attingendo a molti pool di capacità Spot e ridurre l'impatto delle interruzioni delle istanze Spot sulla disponibilità del cluster. Le policy di istanza miste con istanze spot sono un ottimo modo per aumentare la diversità senza aumentare il numero di gruppi di nodi. Tieni presente che se hai bisogno di risorse garantite, utilizza le istanze on demand anziché le istanze Spot.

È fondamentale che tutti i tipi di istanza abbiano una capacità di risorse simile durante la configurazione delle politiche per istanze miste. Il simulatore di pianificazione di Autoscaler utilizza il primo di. InstanceType MixedInstancePolicy Se i tipi di istanza successivi sono più grandi, le risorse potrebbero essere sprecate dopo un ampliamento. Se sono più piccoli, i pod potrebbero non riuscire a programmarsi sulle nuove istanze a causa della capacità insufficiente. Ad esempio, le istanze M4, M5, M5a e M5n hanno tutte quantità di CPU e memoria simili e sono ottime candidate per un. MixedInstancePolicy Lo strumento EC2 Instance Selector può aiutarti a identificare tipi di istanze simili.



Si consiglia di isolare la capacità On-Demand e Spot in gruppi di Auto Scaling EC2 separati. Questa soluzione è preferibile rispetto all'utilizzo di una strategia di capacità di base, poiché le proprietà di pianificazione sono fondamentalmente diverse. Poiché le istanze Spot vengono interrotte in

qualsiasi momento (quando EC2 ha bisogno di recuperare la capacità), gli utenti spesso alterano i nodi presvuotabili, richiedendo un'esplicita tolleranza dei pod rispetto al comportamento di prelazione. Queste contaminazioni determinano proprietà di pianificazione diverse per i nodi, quindi devono essere separati in più gruppi di Auto Scaling EC2.

Cluster Autoscaler si basa sul concetto di Expander, che fornisce diverse strategie per la selezione del gruppo di nodi da scalare. La strategia `--expander=least-waste` è una buona impostazione predefinita per uso generale e, se intendi utilizzare più gruppi di nodi per la diversificazione delle istanze Spot (come descritto nell'immagine precedente), potrebbe aiutare a ottimizzare ulteriormente i costi dei gruppi di nodi scalando il gruppo che sarebbe meglio utilizzato dopo l'attività di scalabilità.

Assegnazione di priorità a un gruppo di nodi /ASG

È inoltre possibile configurare la scalabilità automatica basata sulla priorità utilizzando l'espansore Priority. `--expander=priority`consente al cluster di dare priorità a un gruppo di nodi /ASG e, se non è in grado di scalare per qualsiasi motivo, sceglierà il gruppo di nodi successivo nell'elenco con priorità. Ciò è utile in situazioni in cui, ad esempio, si desidera utilizzare i tipi di istanza P3 perché la loro GPU offre prestazioni ottimali per il carico di lavoro, ma come seconda opzione è possibile utilizzare anche i tipi di istanze P2.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-autoscaler-priority-expander
  namespace: kube-system
data:
  priorities: |-
    10:
      - .*p2-node-group.*
    50:
      - .*p3-node-group.*
```

Cluster Autoscaler cercherà di scalare il gruppo Auto Scaling EC2 corrispondente al nome p3-node-group. Se questa operazione non riesce all'interno*`--max-node-provision-time`*, tenterà di scalare un gruppo di Auto Scaling EC2 corrispondente al nome p2-node-group. Questo valore predefinito è di 15 minuti e può essere ridotto per una selezione del gruppo di nodi più reattiva, tuttavia, se il valore è troppo basso, può causare scale-out non necessari.

Overprovisioning

Cluster Autoscaler riduce al minimo i costi garantendo che i nodi vengano aggiunti al cluster solo quando necessario e rimossi quando non vengono utilizzati. Ciò influisce in modo significativo sulla latenza di implementazione perché molti pod saranno costretti ad attendere la scalabilità di un nodo prima di poter essere pianificati. I nodi possono richiedere più minuti per diventare disponibili, il che può aumentare la latenza di pianificazione del pod di un ordine di grandezza.

Questo può essere mitigato usando overprovisioning, che tratta i costi per la latenza di programmazione. L'overprovisioning viene implementato utilizzando pod temporanei con priorità negativa, che occupano spazio nel cluster. Quando i pod appena creati non sono programmabili e hanno una priorità più alta, i pod temporanei avranno la precedenza per liberare spazio. I pod temporanei diventano quindi non programmabili, attivando Cluster Autoscaler per ridimensionare i nuovi nodi sovradimensionati.

L'overprovisioning presenta altri vantaggi meno evidenti. Senza overprovisioning, uno degli effetti collaterali di un cluster altamente utilizzato è che i pod prenderanno decisioni di pianificazione meno ottimali utilizzando la regola di Pod o Node Affinity. `preferredDuringSchedulingIgnoredDuringExecution` Un caso d'uso comune consiste nel separare i pod per un'applicazione ad alta disponibilità tra le zone di disponibilità utilizzando. AntiAffinity L'overprovisioning può aumentare in modo significativo la possibilità che sia disponibile un nodo della zona corretta.

La quantità di capacità sovradimensionata è una decisione aziendale attenta per l'organizzazione. Fondamentalmente, si tratta di un compromesso tra prestazioni e costi. Un modo per prendere questa decisione è determinare la frequenza media di scalabilità e dividerla per il tempo necessario per scalare un nuovo nodo. Ad esempio, se in media è necessario un nuovo nodo ogni 30 secondi ed EC2 impiega 30 secondi per fornire un nuovo nodo, un singolo nodo di overprovisioning assicurerà che sia sempre disponibile un nodo aggiuntivo, riducendo la latenza di pianificazione di 30 secondi al costo di una singola istanza EC2 aggiuntiva. Per migliorare le decisioni di pianificazione zonale, sovrapponi un numero di nodi pari al numero di zone di disponibilità nel tuo gruppo di Auto Scaling EC2 per garantire che lo scheduler possa selezionare la zona migliore per i pod in entrata.

Previeni lo sfratto da Scale Down

Alcuni carichi di lavoro sono costosi da espellere. L'analisi dei big data, le attività di machine learning e i test runner alla fine verranno completati, ma devono essere riavviati se interrotti. Cluster Autoscaler tenterà di ridimensionare qualsiasi nodo al di sotto di scale-down-utilization-threshold,

interrompendo tutti i pod rimanenti sul nodo. Ciò può essere evitato assicurando che i pod costosi da rimuovere siano protetti da un'etichetta riconosciuta dal Cluster Autoscaler.

Verifica che:

- I pod costosi da sfrattare hanno l'annotazione `cluster-autoscaler.kubernetes.io/safe-to-evict=false`

Casi d'uso avanzati

Volumi EBS

Lo storage persistente è fondamentale per la creazione di applicazioni con stato, come database o cache distribuite. I volumi EBS consentono questo caso d'uso su Kubernetes, ma sono limitati a una zona specifica. Queste applicazioni possono essere altamente disponibili se suddivise su più AZ utilizzando un volume EBS separato per ogni AZ. Cluster Autoscaler può quindi bilanciare la scalabilità dei gruppi di autoscaling EC2.

Verifica che:

- Il bilanciamento del gruppo di nodi è abilitato impostando `balance-similar-node-groups=true`.
- I gruppi di nodi sono configurati con impostazioni identiche, fatta eccezione per zone di disponibilità e volumi EBS diversi.

Pianificazione congiunta

I processi di formazione distribuiti di Machine Learning traggono vantaggio in modo significativo dalla latenza ridotta delle configurazioni dei nodi della stessa zona. Questi carichi di lavoro distribuiscono più pod in una zona specifica. Ciò può essere ottenuto impostando Pod Affinity per tutti i pod co-programmati o utilizzando Node Affinity. `topologyKey: failure-domain.beta.kubernetes.io/zone` Cluster Autoscaler ridimensionerà quindi una zona specifica per soddisfare le richieste. Potresti voler allocare più gruppi di Auto Scaling EC2, uno per zona di disponibilità, per abilitare il failover per l'intero carico di lavoro co-programmato.

Verifica che:

- Il bilanciamento dei gruppi di nodi è abilitato impostando `balance-similar-node-groups=false`

- Node Affinity e/o Pod Preemption vengono utilizzati quando i cluster includono gruppi di nodi regionali e zonali.
  - Usa Node Affinity per forzare o incoraggiare i pod regionali a evitare i gruppi di nodi zonali e viceversa.
  - Se i pod zonali si collegano a gruppi di nodi regionali, ciò comporterà uno squilibrio della capacità dei pod regionali.
  - Se i tuoi carichi di lavoro zonali sono in grado di tollerare interruzioni e spostamenti, configura Pod Preemption per consentire ai pod su scala regionale di forzare la priorità e la riprogrammazione in una zona meno contesa.

Acceleratori

Alcuni cluster sfruttano acceleratori hardware specializzati come la GPU. In caso di scalabilità orizzontale, il plug-in del dispositivo acceleratore può impiegare diversi minuti per pubblicizzare la risorsa nel cluster. Cluster Autoscaler ha simulato che questo nodo avrà l'acceleratore, ma finché l'acceleratore non sarà pronto e non aggiornerà le risorse disponibili del nodo, i pod in sospeso non possono essere programmati sul nodo. Ciò può comportare la ripetizione inutile della scalabilità orizzontale.

Inoltre, i nodi con acceleratori e un elevato utilizzo della CPU o della memoria non verranno presi in considerazione per la riduzione, anche se l'acceleratore non è utilizzato. Questo comportamento può essere costoso a causa del costo relativo degli acceleratori. Al contrario, Cluster Autoscaler può applicare regole speciali per considerare i nodi da ridimensionare se hanno acceleratori non occupati.

Per garantire il comportamento corretto in questi casi, puoi configurare il kubelet sui nodi dell'acceleratore per etichettare il nodo prima che si unisca al cluster. Cluster Autoscaler utilizzerà questo selettore di etichette per attivare il comportamento ottimizzato dell'acceleratore.

Verifica che:

- Il Kubelet per i nodi GPU è configurato con `--node-labels k8s.amazonaws.com/accelerator=$ACCELERATOR_TYPE`
- I nodi con acceleratori aderiscono alla regola delle proprietà di pianificazione identica sopra indicata.

Scalabilità da 0

Cluster Autoscaler è in grado di scalare i gruppi di nodi da e verso lo zero, il che può portare a significativi risparmi sui costi. Rileva le risorse di CPU, memoria e GPU di un gruppo di Auto Scaling ispezionando le risorse specificate InstanceType nel suo o. LaunchConfiguration LaunchTemplate Alcuni pod richiedono risorse aggiuntive come `WindowsENI` or o specifiche `PrivateIPv4Address` NodeSelectors o Taints che non possono essere scoperte tramite. LaunchConfiguration Cluster Autoscaler può tenere conto di questi fattori scoprendoli dai tag del gruppo Auto Scaling di EC2. Esempio:

```
Key: k8s.io/cluster-autoscaler/node-template/resources/$RESOURCE_NAME
Value: 5
Key: k8s.io/cluster-autoscaler/node-template/label/$LABEL_KEY
Value: $LABEL_VALUE
Key: k8s.io/cluster-autoscaler/node-template/taint/$TAINT_KEY
Value: NoSchedule
```

> **Note**
>
> Tieni presente che quando si esegue la scalabilità a zero, la capacità viene restituita a EC2 e potrebbe non essere più disponibile in futuro.

Parametri aggiuntivi

Ci sono molte opzioni di configurazione che possono essere utilizzate per ottimizzare il comportamento e le prestazioni di Cluster Autoscaler. Un elenco completo dei parametri è disponibile su GitHub.

Parametro	Description	Predefinita
intervallo di scansione	Con quale frequenza il cluster viene rivalutato per aumentarn e o ridurne la scalabilità	10 secondi
max-empty-bulk-delete	Numero massimo di nodi vuoti che possono essere eliminati contemporaneamente.	10

Parametro	Description	Predefinita
scale-down-delay-after-aggiungi	Quanto tempo dopo l'aumento della scala verso il basso riprende la valutazione	10 minuti
scale-down-delay-after-eliminare	Quanto tempo dopo l'eliminazione del nodo riprende la valutazione della scala ridotta, il valore predefinito è scan-interval	intervallo di scansione
scale-down-delay-after-fallimento	Quanto tempo dopo l'errore di ridimensionamento riprende la valutazione	3 minuti
scale-down-unneeded-time	Per quanto tempo non deve essere necessario un nodo prima che sia idoneo per la scalabilità verso il basso	10 minuti
scale-down-unready-time	Per quanto tempo un nodo non pronto non deve essere necessario prima che sia idoneo per la scalabilità verso il basso	20 minuti
scale-down-utilization-threshold	Livello di utilizzo del nodo, definito come la somma delle risorse richieste divisa per la capacità, al di sotto del quale è possibile prendere in considerazione la possibilità di ridimensionare un nodo	0,5

Parametro	Description	Predefinita
scale-down-non-empty-conteggio dei candidati	Numero massimo di nodi non vuoti considerati in un'iterazione come candidati per essere ridimensionati con drain. Un valore inferiore significa una migliore reattività della CA, ma è possibile una latenza di scalabilità inferiore. Un valore più elevato può influire sulle prestazioni della CA con cluster di grandi dimensioni (centinaia di nodi). Imposta su un valore non positivo per disattivare questa euristica: CA non limiterà il numero di nodi che considera. »	30

Parametro	Description	Predefinita
scale-down-candidates-pool-rapporto	Un rapporto tra nodi considerati come candidati aggiuntivi non vuoti da ridimensionare quando alcuni candidati dell'iterazione precedente non sono più validi. Un valore inferiore significa una migliore reattività della CA, ma è possibile una latenza di scalabilità più lenta. Un valore più elevato può influire sulle prestazioni della CA con cluster di grandi dimensioni (centinaia di nodi). Imposta su 1.0 per disattivare questa euristica: CA accetterà tutti i nodi come candidati aggiuntivi.	0.1
scale-down-candidates-pool-conteggio dei minuti	Numero minimo di nodi considerati come candidati aggiuntivi non vuoti da ridimensionare quando alcuni candidati dell'iterazione precedente non sono più validi. Nel calcolare la dimensione del pool per i candidati aggiuntivi, prendiamo `max(#nodes * scale-down-candidates-pool-ratio, scale-down-candidates-pool-min-count)`	50

Risorse aggiuntive

Questa pagina contiene un elenco di presentazioni e demo di Cluster Autoscaler. Se desideri aggiungere una presentazione o una demo qui, invia una pull request.

Presentazione/demo	Presentatori
Scalabilità automatica e ottimizzazione dei costi su Kubernetes: da 0 a 100	Guy Templeton, Skyscanner e Jiaxin Shan, Amazon
Approfondimento su SIG-Autoscaling	Maciek Pytel e Marcin Wielgus

Riferimenti

- https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/FAQ.md
- https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/cloudprovider/aws/README.md
- https://github.com/aws/selettore di istanze amazon-ec2-
- https://github.com/aws/aws-node-termination-handler

Migliori pratiche per l'affidabilità

Questa sezione fornisce indicazioni su come rendere i carichi di lavoro in esecuzione su EKS resilienti e altamente disponibili

Come utilizzare questa guida

Questa guida è destinata a sviluppatori e architetti che desiderano sviluppare e gestire servizi ad alta disponibilità e con tolleranza ai guasti in EKS. La guida è organizzata in diverse aree tematiche per facilitarne la fruizione. Ogni argomento inizia con una breve panoramica, seguita da un elenco di consigli e best practice per l'affidabilità dei cluster EKS.

Introduzione

Le migliori pratiche di affidabilità per EKS sono state raggruppate nei seguenti argomenti:

- Applicazioni
- Piano di controllo
- Piano dei dati

Cosa rende affidabile un sistema? Se un sistema può funzionare in modo coerente e soddisfare le esigenze nonostante i cambiamenti dell'ambiente nel corso di un periodo di tempo, può essere definito affidabile. A tal fine, il sistema deve rilevare i guasti, correggersi automaticamente e avere la capacità di scalare in base alla domanda.

I clienti possono utilizzare Kubernetes come base per gestire applicazioni e servizi mission-critical in modo affidabile. Ma oltre a incorporare principi di progettazione delle applicazioni basate su container, l'esecuzione affidabile dei carichi di lavoro richiede anche un'infrastruttura affidabile. In Kubernetes, l'infrastruttura comprende il piano di controllo e il piano dati.

EKS fornisce un piano di controllo Kubernetes di livello di produzione progettato per essere altamente disponibile e tollerante ai guasti.

In EKS, AWS è responsabile dell'affidabilità del piano di controllo Kubernetes. EKS esegue il piano di controllo Kubernetes su tre zone di disponibilità in una regione AWS. Gestisce automaticamente la disponibilità e la scalabilità dei server API Kubernetes e del cluster etcd.

La responsabilità dell'affidabilità del piano dati è condivisa tra te, il cliente e AWS. EKS offre quattro opzioni di nodi di lavoro per l'implementazione del piano dati Kubernetes.

EKS Auto Mode, che è l'opzione più gestita, gestisce il provisioning, il ridimensionamento e gli aggiornamenti del piano dati oltre a fornire funzionalità gestite di elaborazione, rete e archiviazione. La modalità AMIs automatica viene rilasciata frequentemente e i cluster vengono aggiornati automaticamente all'AMI più recente per implementare correzioni CVE e patch di sicurezza. È possibile controllare quando ciò si verifica configurando i controlli delle interruzioni nella modalità Auto. NodePools

Fargate gestisce il provisioning e la scalabilità del piano dati eseguendo un Pod per nodo. La terza opzione, i gruppi di nodi gestiti, gestisce il provisioning e gli aggiornamenti del piano dati. Infine, i nodi autogestiti sono l'opzione meno gestita per il piano dati. Maggiore è il piano dati gestito da AWS, minore è la responsabilità.

I gruppi di nodi gestiti automatizzano il provisioning e la gestione del ciclo di vita dei nodi EC2. Puoi utilizzare l'API EKS (utilizzando la console EKS, l'API AWS, l'AWS CLICloudFormation, Terraform `oeksctl`) per creare, scalare e aggiornare nodi gestiti. I nodi gestiti eseguono istanze Amazon Linux 2 EC2 ottimizzate per EKS nel tuo account e puoi installare pacchetti software personalizzati abilitando l'accesso SSH. Quando effettui il provisioning dei nodi gestiti, questi vengono eseguiti come parte di un gruppo di Auto Scaling gestito da EKS che può estendersi su più zone di disponibilità; il controllo avviene tramite le sottoreti fornite durante la creazione di nodi gestiti. EKS inoltre contrassegna automaticamente i nodi gestiti in modo che possano essere utilizzati con Cluster Autoscaler.

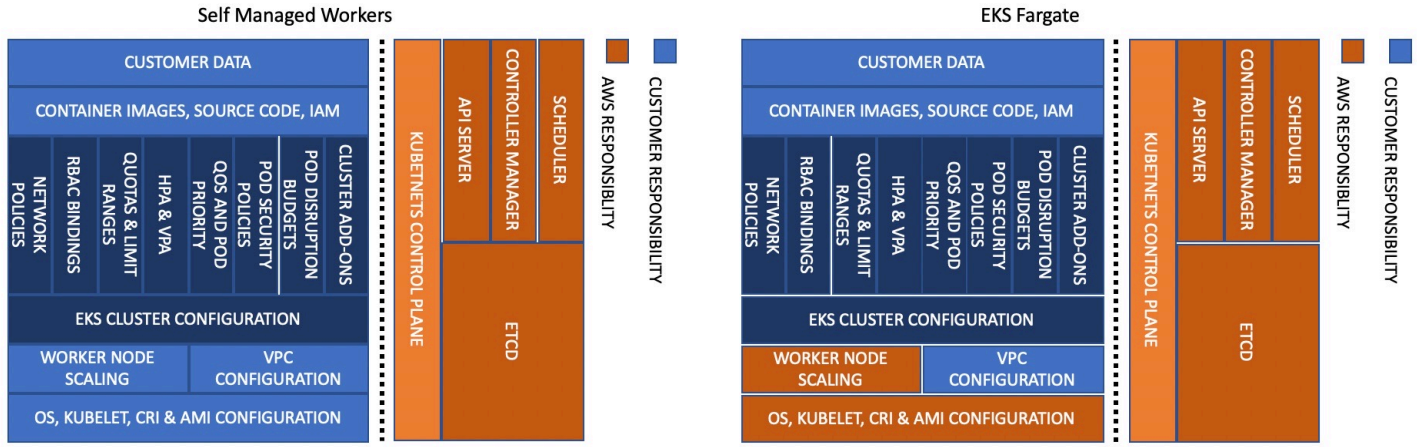
Amazon EKS segue il modello di responsabilità condivisa CVEs e le patch di sicurezza sui gruppi di nodi gestiti. Poiché i nodi gestiti eseguono sistemi ottimizzati per Amazon EKS, AMIs Amazon EKS è responsabile della creazione di versioni patchate di questi AMIs durante la correzione dei bug. L'utente è invece responsabile della distribuzione di queste versioni AMI con patch ai gruppi di nodi gestiti.

EKS gestisce anche l'aggiornamento dei nodi, sebbene sia necessario avviare il processo di aggiornamento. Il processo di aggiornamento del nodo gestito è spiegato nella documentazione EKS.

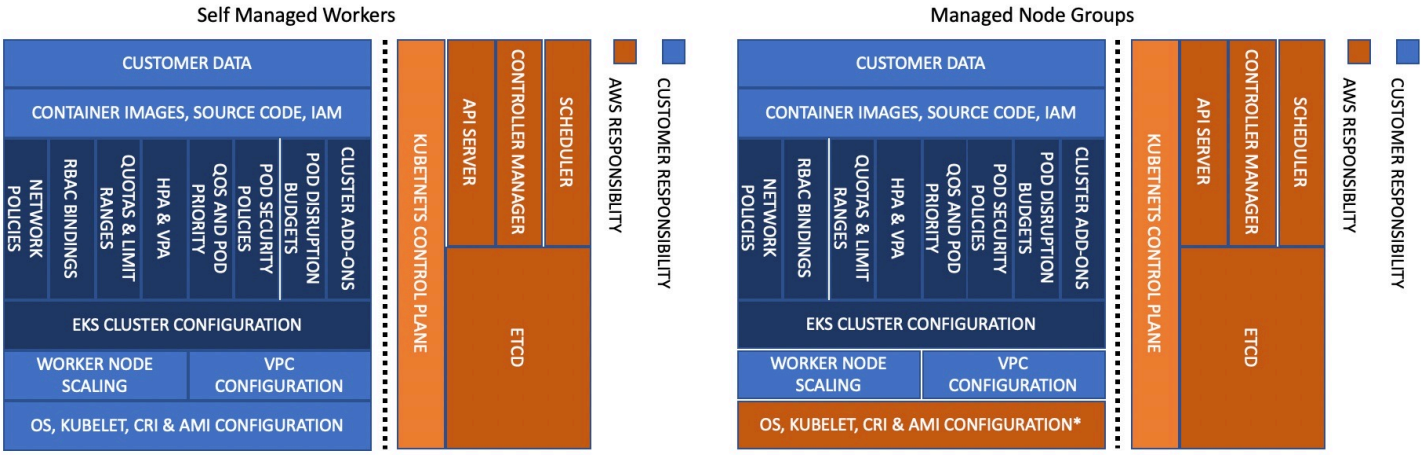
Se esegui nodi autogestiti, puoi utilizzare l'AMI Linux ottimizzata per Amazon EKS per creare nodi di lavoro. L'utente è responsabile dell'applicazione di patch e dell'aggiornamento dell'AMI e dei nodi. È consigliabile utilizzare l'`eksctl`infrastruttura come strumenti di codice per il provisioning di nodi autogestiti CloudFormation, poiché in questo modo sarà più semplice aggiornare i nodi autogestiti. Prendi in considerazione la migrazione a nuovi nodi quando aggiorni i nodi di lavoro, poiché il

processo di migrazione altera il vecchio gruppo di nodi `NoSchedule` e prosciuga i nodi dopo che un nuovo stack è pronto ad accettare il carico di lavoro del pod esistente. Tuttavia, puoi anche eseguire un aggiornamento sul posto dei nodi [autogestiti.]

Modello di responsabilità condivisa - Fargate



Modello di responsabilità condivisa - MNG



Questa guida include una serie di consigli che puoi utilizzare per migliorare l'affidabilità del tuo piano dati EKS, dei componenti principali di Kubernetes e delle tue applicazioni.

Feedback

Questa guida è stata pubblicata GitHub per raccogliere feedback e suggerimenti diretti dalla community più ampia. EKS/Kubernetes Se hai una best practice che ritieni debba essere inclusa nella guida, segnala un problema o invia un PR nell' GitHub archivio. Intendiamo aggiornare

periodicamente la guida man mano che vengono aggiunte nuove funzionalità al servizio o quando si evolve una nuova best practice.

Esecuzione di applicazioni ad alta disponibilità

I tuoi clienti si aspettano che la tua applicazione sia sempre disponibile, anche quando apporti modifiche e soprattutto durante i picchi di traffico. Un'architettura scalabile e resiliente mantiene le applicazioni e i servizi operativi senza interruzioni, il che rende felici gli utenti. Un'infrastruttura scalabile cresce e si restringe in base alle esigenze dell'azienda. L'eliminazione dei singoli punti di errore è un passaggio fondamentale per migliorare la disponibilità di un'applicazione e renderla resiliente.

Con Kubernetes, puoi utilizzare le tue applicazioni ed eseguirle in modo altamente disponibile e resiliente. La sua gestione dichiarativa assicura che, una volta configurata l'applicazione, Kubernetes cercherà continuamente di far corrispondere lo stato corrente con lo stato desiderato.

Raccomandazioni

Configura Pod Disruption Budgets

I Pod Disruption Budgets vengono utilizzati per limitare la quantità di interruzioni simultanee che un'applicazione subirà. Dovrebbero essere configurati per i carichi di lavoro se è importante avere sempre una parte di quel carico di lavoro disponibile. EKS Auto Mode, Karpenter e Cluster Autoscaler conoscono e rispettano i budget Pod Disruption configurati in fase di ridimensionamento. EKS Auto Mode, Karpenter e Managed Node Groups rispettano anche i Pod Disruption Budgets durante l'aggiornamento dei nodi

Evita di utilizzare Singleton Pods

Se l'intera applicazione viene eseguita in un singolo Pod, l'applicazione non sarà disponibile se quel Pod viene terminato. Invece di distribuire le applicazioni utilizzando singoli pod, create Deployments. Se un Pod creato da un Deployment si guasta o viene terminato, il Deployment controller avvierà un nuovo pod per garantire che il numero specificato di Pod di replica sia sempre in esecuzione.

Esegui più repliche

L'esecuzione di più Repliche Pods di un'app utilizzando un Deployment ne aiuta l'esecuzione in modo ad alta disponibilità. Se una replica fallisce, le repliche rimanenti continueranno a funzionare, anche se a capacità ridotta, fino a quando Kubernetes non creerà un altro Pod per compensare la perdita.

Inoltre, puoi utilizzare Horizontal Pod Autoscaler per ridimensionare automaticamente le repliche in base alla richiesta del carico di lavoro.

Pianifica le repliche tra i nodi

L'esecuzione di più repliche non sarà molto utile se tutte le repliche sono in esecuzione sullo stesso nodo e il nodo non è più disponibile. Prendi in considerazione l'utilizzo dei vincoli di antiaffinità dei pod o di diffusione della topologia dei pod per distribuire le repliche di una distribuzione su più nodi di lavoro.

È possibile migliorare ulteriormente l'affidabilità di un'applicazione tipica eseguendola su più applicazioni. AZs

Utilizzo delle regole anti-affinità Pod

Il manifesto seguente indica allo scheduler di Kubernetes di preferire posizionare i pod su nodi separati e. AZs Non richiede nodi distinti o AZ perché, in tal caso, Kubernetes non sarà in grado di pianificare alcun pod una volta che c'è un pod in esecuzione in ciascuna AZ. Se l'applicazione richiede solo tre repliche, puoi utilizzare `requiredDuringSchedulingIgnoredDuringExecution` for `topologyKey: topology.kubernetes.io/zone` e Kubernetes scheduler non pianificherà due pod nella stessa AZ.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: spread-host-az
  labels:
    app: web-server
spec:
  replicas: 4
  selector:
    matchLabels:
      app: web-server
  template:
    metadata:
      labels:
        app: web-server
    spec:
      affinity:
        podAntiAffinity:
```

```
        preferredDuringSchedulingIgnoredDuringExecution:
        - podAffinityTerm:
            labelSelector:
              matchExpressions:
              - key: app
                operator: In
                values:
                - web-server
            topologyKey: topology.kubernetes.io/zone
          weight: 100
        - podAffinityTerm:
            labelSelector:
              matchExpressions:
              - key: app
                operator: In
                values:
                - web-server
            topologyKey: kubernetes.io/hostname
          weight: 99
      containers:
      - name: web-app
        image: nginx:1.16-alpine
```

Utilizzo dei vincoli di diffusione della topologia Pod

Analogamente alle regole di antiaffinità dei pod, i vincoli di diffusione della topologia dei pod consentono di rendere disponibile l'applicazione su diversi domini di errore (o topologia) come host o. AZs Questo approccio funziona molto bene quando si cerca di garantire la tolleranza agli errori e la disponibilità utilizzando più repliche in ciascuno dei diversi domini di topologia. Le regole di antiaffinità dei pod, d'altra parte, possono facilmente produrre un risultato in cui si ha una singola replica in un dominio di topologia, perché i pod con un'antiaffinità l'uno verso l'altro hanno un effetto repellente. In questi casi, una singola replica su un nodo dedicato non è ideale per la tolleranza agli errori né rappresenta un buon uso delle risorse. Con i vincoli di diffusione della topologia, si ha un maggiore controllo sulla diffusione o sulla distribuzione che lo scheduler dovrebbe cercare di applicare tra i domini della topologia. Ecco alcune proprietà importanti da utilizzare in questo approccio:

1. `maxSkew`Viene utilizzato per controllare o determinare il punto massimo in cui le cose possono essere irregolari tra i domini della topologia. Ad esempio, se un'applicazione ha 10 repliche ed è distribuita su 3 AZs, non è possibile ottenere una distribuzione uniforme, ma è possibile influenzare l'irregolarità della distribuzione. In questo caso, `maxSkew` può essere un valore compreso tra 1 e 10. Un valore pari a 1 significa che è possibile che si ottenga uno spread pari `4,3,3` `3,4,3` o

`3,3,4` superiore a 3 AZs. Al contrario, un valore di 10 significa che potresti potenzialmente finire con uno spread pari `10,0,0` `0,10,0` o compreso `0,0,10` tra 3 AZs.

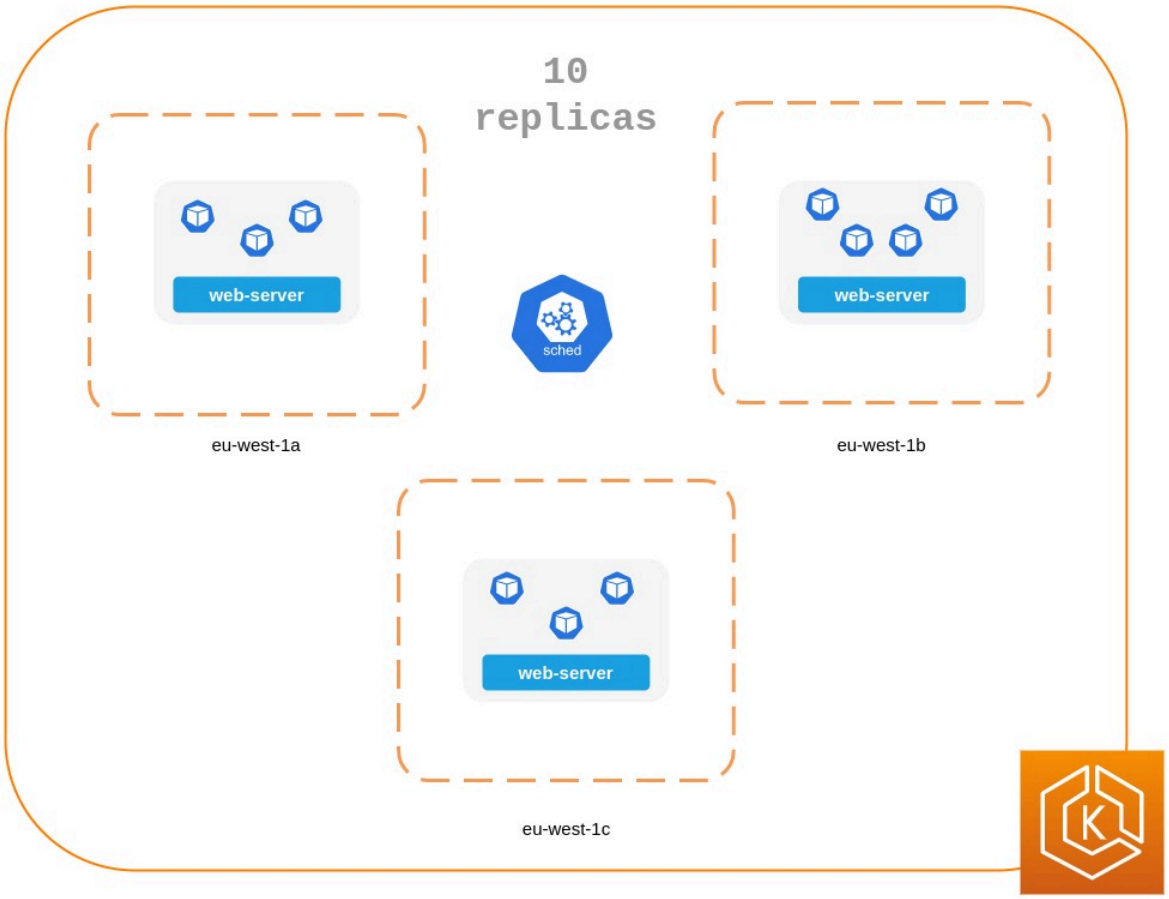
2. `topologyKey`è una chiave per una delle etichette dei nodi e definisce il tipo di dominio topologico da utilizzare per la distribuzione dei pod. Ad esempio, uno spread zonale avrebbe la seguente coppia chiave-valore:

```
topologyKey: "topology.kubernetes.io/zone"
```

3. La `whenUnsatisfiable` proprietà viene utilizzata per determinare come si desidera che lo scheduler risponda se i vincoli desiderati non possono essere soddisfatti.
4. `labelSelector`Viene utilizzato per trovare i pod corrispondenti in modo che lo scheduler possa esserne a conoscenza quando decide dove posizionare i pod in base ai vincoli specificati.

Oltre a quelli precedenti, ci sono altri campi che puoi approfondire nella documentazione di Kubernetes.

La topologia dei pod distribuisce i vincoli su 3 AZs




```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: spread-host-az
  labels:
    app: web-server
spec:
  replicas: 10
  selector:
    matchLabels:
      app: web-server
  template:
    metadata:
      labels:
        app: web-server
    spec:
      topologySpreadConstraints:
```

```
    - maxSkew: 1
      topologyKey: "topology.kubernetes.io/zone"
      whenUnsatisfiable: ScheduleAnyway
      labelSelector:
        matchLabels:
          app: express-test
  containers:
  - name: web-app
    image: nginx:1.16-alpine
```

Esegui Kubernetes Metrics Server

Installa il server di metrica Kubernetes per aiutarti a scalare le tue applicazioni. I componenti aggiuntivi di Kubernetes autoscaler come HPA e VPA devono tenere traccia delle metriche delle applicazioni per scalarle. Il metrics-server raccoglie i parametri delle risorse che possono essere utilizzati per prendere decisioni di scalabilità. Le metriche vengono raccolte da kubelets e fornite nel formato Metrics API.

Il server di metrica non conserva alcun dato e non è una soluzione di monitoraggio. Il suo scopo è esporre le metriche di utilizzo della CPU e della memoria ad altri sistemi. Se desideri monitorare lo stato dell'applicazione nel tempo, hai bisogno di uno strumento di monitoraggio come Prometheus o Amazon. CloudWatch

Segui la documentazione EKS per installare metrics-server nel tuo cluster EKS.

Horizontal Pod Autoscaler (HPA)

HPA può scalare automaticamente la vostra applicazione in risposta alla domanda e aiutarvi a evitare di avere ripercussioni sui clienti durante i picchi di traffico. È implementato come un anello di controllo in Kubernetes che interroga periodicamente le metriche che forniscono le metriche delle risorse. APIs

HPA può recuperare le metriche da quanto segue: 1. APIs `metrics.k8s.io`nota anche come API Resource Metrics: fornisce l'utilizzo di CPU e memoria per i pod 2. `custom.metrics.k8s.io` — Fornisce metriche provenienti da altri raccoglitori di metriche come Prometheus; queste metriche sono interne al tuo cluster Kubernetes. 3. `external.metrics.k8s.io` — Fornisce metriche esterne al cluster Kubernetes (ad esempio, SQS Queue Depth, latenza ELB).

È necessario utilizzare una di queste tre APIs per fornire la metrica necessaria per scalare l'applicazione.

Scalabilità delle applicazioni in base a metriche personalizzate o esterne

È possibile utilizzare metriche personalizzate o esterne per ridimensionare l'applicazione in base a metriche diverse dall'utilizzo della CPU o della memoria. I server Custom Metrics API forniscono l'`custom-metrics.k8s.io`API che HPA può utilizzare per la scalabilità automatica delle applicazioni.

Puoi utilizzare Prometheus Adapter for Kubernetes APIs Metrics per raccogliere metriche da Prometheus e utilizzarle con l'HPA. In questo caso, l'adattatore Prometheus esporrà le metriche Prometheus nel formato Metrics API.

Una volta distribuito l'adattatore Prometheus, puoi interrogare metriche personalizzate utilizzando kubectl. `kubectl get –raw /apis/custom.metrics.k8s.io/v1beta1/`

Le metriche esterne, come suggerisce il nome, forniscono a Horizontal Pod Autoscaler la capacità di scalare le implementazioni utilizzando metriche esterne al cluster Kubernetes. Ad esempio, nei carichi di lavoro di elaborazione in batch, è comune ridimensionare il numero di repliche in base al numero di lavori in esecuzione in una coda SQS.

Per scalare automaticamente i carichi di lavoro Kubernetes puoi utilizzare KEDA (Kubernetes Event-driven Autoscaling), un progetto open source in grado di favorire la scalabilità dei container in base a una serie di eventi personalizzati. Questo blog di AWS descrive come utilizzare Amazon Managed Service for Prometheus per l'auto-scaling dei carichi di lavoro Kubernetes.

Vertical Pod Autoscaler (VPA)

VPA regola automaticamente la riserva di CPU e memoria per i tuoi Pod per aiutarti a «dimensionare correttamente» le tue applicazioni. Per le applicazioni che devono essere scalate verticalmente, operazione che avviene aumentando l'allocazione delle risorse, è possibile utilizzare VPA per ridimensionare automaticamente le repliche dei Pod o fornire consigli di ridimensionamento.

L'applicazione potrebbe diventare temporaneamente non disponibile se VPA deve scalarla perché l'attuale implementazione di VPA non esegue modifiche sul posto ai Pod; ricrea invece il Pod che deve essere ridimensionato.

La documentazione EKS include una procedura dettagliata per la configurazione di VPA.

Il progetto Fairwinds Goldilocks fornisce una dashboard per visualizzare i consigli VPA per le richieste e i limiti di CPU e memoria. La sua modalità di aggiornamento VPA consente di scalare automaticamente i Pod in base ai consigli VPA.

Aggiornamento delle applicazioni

Le applicazioni moderne richiedono un'innovazione rapida con un elevato grado di stabilità e disponibilità. Kubernetes ti offre gli strumenti per aggiornare continuamente le tue applicazioni senza interrompere le attività dei tuoi clienti.

Diamo un'occhiata ad alcune delle migliori pratiche che consentono di implementare rapidamente le modifiche senza sacrificare la disponibilità.

Disponi di un meccanismo per eseguire i rollback

Avere un pulsante Annulla può evitare i disastri. È consigliabile testare le implementazioni in un ambiente inferiore separato (ambiente di test o sviluppo) prima di aggiornare il cluster di produzione. L'utilizzo di una pipeline CI/CD può aiutarti ad automatizzare e testare le implementazioni. Con una pipeline di distribuzione continua, è possibile tornare rapidamente alla versione precedente se l'aggiornamento risulta difettoso.

È possibile utilizzare Deployments per aggiornare un'applicazione in esecuzione. Questa operazione viene in genere eseguita aggiornando l'immagine del contenitore. Puoi usare `kubectl` per aggiornare una distribuzione in questo modo:

```
kubectl --record deployment.apps/nginx-deployment set image nginx-deployment
 nginx=nginx:1.16.1
```

L'`--record`argomento registra le modifiche al Deployment e ti aiuta se devi eseguire un rollback. `kubectl rollout history deployment`mostra le modifiche registrate alle distribuzioni nel cluster. È possibile ripristinare una modifica utilizzando. `kubectl rollout undo deployment <DEPLOYMENT_NAME>`

Per impostazione predefinita, quando aggiorni una distribuzione che richiede la creazione di pod, Deployment eseguirà un aggiornamento progressivo. In altre parole, Kubernetes aggiornerà solo una parte dei pod in esecuzione in un Deployment e non tutti i Pod contemporaneamente. Puoi controllare il modo in cui Kubernetes esegue gli aggiornamenti continui tramite la proprietà. `RollingUpdateStrategy`

Quando si esegue un aggiornamento continuo di una distribuzione, è possibile utilizzare la `Max Unavailable`proprietà per specificare il numero massimo di Pod che non possono essere disponibili durante l'aggiornamento. La `Max Surge` proprietà Deployment consente di impostare il numero massimo di Pod che possono essere creati sul numero desiderato di Pod.

Valuta la possibilità di `max unavailable` apportare modifiche per garantire che l'implementazione non arrechi problemi ai clienti. Ad esempio, Kubernetes imposta il 25% per impostazione `max unavailable` predefinita, il che significa che se hai 100 Pod, potresti avere solo 75 Pod attivi durante un'implementazione. Se la tua applicazione richiede un minimo di 80 Pod, questa implementazione può causare interruzioni. Puoi invece impostarlo `max unavailable` al 20% per assicurarti che ci siano almeno 80 Pod funzionanti durante l'implementazione.

Usa le distribuzioni blue/green

Le modifiche sono intrinsecamente rischiose, ma le modifiche che non possono essere annullate possono avere effetti potenzialmente catastrofici. Le procedure di modifica che consentono di tornare efficacemente indietro nel tempo attraverso un rollback rendono più sicuri i miglioramenti e la sperimentazione. Blue/green le implementazioni offrono un metodo per annullare rapidamente le modifiche in caso di problemi. In questa strategia di distribuzione, si crea un ambiente per la nuova versione. Questo ambiente è identico alla versione corrente dell'applicazione in fase di aggiornamento. Una volta effettuato il provisioning del nuovo ambiente, il traffico viene indirizzato al nuovo ambiente. Se la nuova versione produce i risultati desiderati senza generare errori, il vecchio ambiente viene chiuso. In caso contrario, il traffico viene ripristinato alla versione precedente.

Puoi eseguire blue/green distribuzioni in Kubernetes creando una nuova distribuzione identica a quella della versione esistente. Dopo aver verificato che i Pod nella nuova distribuzione funzionino senza errori, puoi iniziare a inviare traffico alla nuova distribuzione modificando le `selector` specifiche del servizio che indirizzano il traffico verso i Pod dell'applicazione.

Molti strumenti di integrazione continua come Flux, Jenkins e Spinnaker consentono di automatizzare le implementazioni. blue/green Il blog di AWS Containers include una procedura dettagliata sull'uso di AWS Load Balancer Controller: utilizzo di AWS Load Balancer Controller blue/green per la distribuzione, implementazione canary e test A/B

Usa le distribuzioni Canary

Le implementazioni Canary sono una variante delle implementazioni che possono eliminare in modo significativo i blue/green rischi derivanti dalle modifiche. In questa strategia di distribuzione, oltre a quella precedente, crei una nuova distribuzione con meno pod e dirottando una piccola percentuale di traffico verso la nuova distribuzione. Se le metriche indicano che la nuova versione offre prestazioni uguali o migliori rispetto alla versione esistente, si aumenta progressivamente il traffico verso la nuova distribuzione e lo si aumenta fino a deviare tutto il traffico verso la nuova distribuzione. In caso di problemi, puoi indirizzare tutto il traffico verso la vecchia distribuzione e interrompere l'invio del traffico verso la nuova distribuzione.

Sebbene Kubernetes non offra un modo nativo per eseguire implementazioni Canary, puoi utilizzare strumenti come Flagger con Istio.

Controlli sanitari e autoguarigione

Nessun software è privo di bug, ma Kubernetes può aiutarti a ridurre al minimo l'impatto dei guasti del software. In passato, se un'applicazione si bloccava, qualcuno doveva porre rimedio alla situazione riavviando l'applicazione manualmente. Kubernetes ti dà la possibilità di rilevare i guasti del software nei tuoi Pod e di sostituirli automaticamente con nuove repliche. Con Kubernetes puoi monitorare lo stato delle tue applicazioni e sostituire automaticamente le istanze non integre.

Kubernetes supporta tre tipi di controlli sanitari:

1. Sonda Liveness
2. Sonda di avvio (supportata nella versione 1.16+ di Kubernetes)
3. Sonda di prontezza

Kubelet, l'agente Kubernetes, è responsabile dell'esecuzione di tutti i controlli sopra menzionati. Kubelet può controllare lo stato di un Pod in tre modi: kubelet può eseguire un comando shell all'interno del contenitore di un Pod, inviare una richiesta HTTP GET al relativo contenitore o aprire un socket TCP su una porta specificata.

Se scegli una sonda *exec* basata, che esegue uno script di shell all'interno di un contenitore, assicurati che il comando shell esca prima della scadenza del valore. `timeoutSeconds` Altrimenti, il nodo avrà `<defunct>` dei processi che porteranno al fallimento del nodo.

Raccomandazioni

Usa Liveness Probe per rimuovere i pod malsani

La sonda Liveness è in grado di rilevare condizioni di stallo in cui il processo continua a funzionare, ma l'applicazione non risponde. Ad esempio, se utilizzi un servizio web che ascolta sulla porta 80, puoi configurare una sonda Liveness per inviare una richiesta HTTP GET sulla porta 80 di Pod. Kubelet invierà periodicamente una richiesta GET al Pod e si aspetta una risposta; se il Pod risponde tra 200 e 399, il kubelet ritiene che il Pod sia integro; in caso contrario, il Pod verrà contrassegnato come non integro. Se un Pod non supera continuamente i controlli sanitari, il kubelet lo interromperà.

È possibile utilizzarlo per ritardare la prima `initialDelaySeconds` sonda.

Quando usi Liveness Probe, assicurati che la tua applicazione non si verifichi una situazione in cui tutti i Pod falliscono contemporaneamente il Liveness Probe perché Kubernetes proverà a sostituire tutti i tuoi Pod, il che renderà l'applicazione offline. Inoltre, Kubernetes continuerà a creare nuovi Pod che non funzioneranno nemmeno con le sonde Liveness, il che metterà a dura prova il piano di controllo. Evita di configurare Liveness Probe in modo che dipenda da un fattore esterno al tuo Pod, ad esempio un database esterno. In altre parole, un external-to-your-Pod database che non risponde non dovrebbe far fallire i Liveness Probe ai tuoi Pod.

Nel post LIVENESS PROBES ARE DANGEROUS di Sandor Szücs vengono descritti i problemi che possono essere causati da sonde mal configurate.

Utilizzate Startup Probe per le applicazioni che richiedono più tempo per avviarsi

Quando l'app necessita di più tempo per l'avvio, puoi utilizzare Startup Probe per ritardare l'avvio di Liveness and Readiness Probe. Ad esempio, un'app Java che deve idratare la cache di un database può richiedere fino a due minuti prima di essere completamente funzionante. Qualsiasi sonda Liveness o Readiness finché non diventa completamente funzionante potrebbe fallire. La configurazione di una Startup Probe consentirà all'app Java di funzionare correttamente prima dell'esecuzione di Liveness o Readiness Probe.

Finché lo Startup Probe non ha esito positivo, tutte le altre Probe sono disabilitate. Puoi definire il tempo massimo di attesa che Kubernetes deve attendere per l'avvio dell'applicazione. Se, dopo il tempo massimo configurato, il Pod continua a non funzionare con Startup Probes, verrà interrotto e verrà creato un nuovo Pod.

Lo Startup Probe è simile al Liveness Probe: se falliscono, il Pod viene ricreato. Come spiega Ricardo A. nel suo post Fantastic Probes And How To Configure Them, le Startup Probes devono essere utilizzate quando il tempo di avvio di un'applicazione è imprevedibile. Se sapete che l'applicazione impiega dieci secondi per avviarsi, dovreste usare invece Liveness/Readiness Probe con. `initialDelaySeconds`

Utilizzate Readiness Probe per rilevare l'indisponibilità parziale

Mentre la sonda Liveness rileva i guasti in un'app che vengono risolti chiudendo il Pod (quindi riavviando l'app), Readiness Probe rileva le condizioni in cui l'app potrebbe essere temporaneamente non disponibile. In queste situazioni, l'app potrebbe temporaneamente non rispondere; tuttavia, si prevede che torni a funzionare correttamente una volta completata l'operazione.

Ad esempio, durante I/O operazioni intensive su disco, le applicazioni potrebbero essere temporaneamente non disponibili per gestire le richieste. In questo caso, la chiusura del Pod

dell'applicazione non è un rimedio; allo stesso tempo, le richieste aggiuntive inviate al Pod possono avere esito negativo.

Puoi utilizzare Readiness Probe per rilevare un'indisponibilità temporanea dell'app e interrompere l'invio di richieste al relativo Pod finché non torna a funzionare. A differenza di Liveness Probe, in cui un guasto comporterebbe la ricreazione del Pod, un Readiness Probe fallito significherebbe che Pod non riceverà alcun traffico dal servizio Kubernetes. Quando Readiness Probe ha esito positivo, Pod riprenderà a ricevere traffico dal Servizio.

Proprio come con Liveness Probe, evita di configurare le Readiness Probe che dipendono da una risorsa esterna al Pod (come un database). Ecco uno scenario in cui un Readiness mal configurato può rendere l'applicazione non funzionante: se il Readiness Probe di un Pod si guasta quando il database dell'app non è raggiungibile, anche le altre repliche Pod falliranno contemporaneamente poiché condividono gli stessi criteri di controllo dello stato. Impostando la sonda in questo modo, ogni volta che il database non è disponibile, i Readiness Probe del Pod si guasteranno e Kubernetes smetterà di inviare traffico a tutti i Pod.

Un effetto collaterale dell'utilizzo di Readiness Probes è che possono aumentare il tempo necessario per aggiornare le distribuzioni. Le nuove repliche non riceveranno traffico a meno che Readiness Probes non abbia esito positivo; fino ad allora, le vecchie repliche continueranno a ricevere traffico.

Gestione delle interruzioni

I Pod hanno una durata limitata: anche se hai Pod a lunga durata, è prudente assicurarsi che i Pod si spengano correttamente quando arriva il momento. A seconda della strategia di aggiornamento, gli aggiornamenti dei cluster Kubernetes potrebbero richiedere la creazione di nuovi nodi di lavoro, il che richiede che tutti i Pod vengano ricreati su nodi più recenti. Una corretta gestione delle terminazioni e i Pod Disruption Budgets possono aiutarti a evitare interruzioni del servizio, poiché i Pod vengono rimossi dai nodi più vecchi e ricreati sui nodi più recenti.

Il modo preferito per aggiornare i nodi di lavoro è creare nuovi nodi di lavoro e terminare quelli vecchi. Prima di terminare i nodi di lavoro, dovresti `drain` farlo. Quando un nodo di lavoro viene svuotato, tutti i relativi pod vengono rimossi in modo sicuro. Sicurezza è una parola chiave in questo contesto; quando i pod di un lavoratore vengono sfrattati, a quest'ultimo non viene semplicemente inviato un segnale. `SIGKILL` Viene invece inviato un `SIGTERM` segnale al processo principale (PID 1) di ogni container dei Pod che viene sfrattato. Dopo l'invio `SIGTERM` del segnale, Kubernetes concederà al processo del tempo (periodo di grazia) prima dell'invio del segnale. `SIGKILL` Questo periodo di

grazia è di 30 secondi per impostazione predefinita; puoi sovrascrivere il valore predefinito usando `grace-period` flag in kubectl o dichiararlo nel tuo Podspec. `terminationGracePeriodSeconds`

```
kubectl delete pod <pod name> –grace-period=<seconds>
```

È comune avere contenitori in cui il processo principale non ha PID 1. Considera questo contenitore di esempio basato su Python:

```
$ kubectl exec python-app -it ps
 PID USER TIME COMMAND
 1   root 0:00 {script.sh} /bin/sh ./script.sh
 5   root 0:00 python app.py
```

In questo esempio, lo script di shell riceve`SIGTERM`, il processo principale, che in questo esempio è un'applicazione Python, non riceve un `SIGTERM` segnale. Quando il Pod viene terminato, l'applicazione Python verrà interrotta bruscamente. Questo può essere risolto modificando il `ENTRYPOINT`contenitore per avviare l'applicazione Python. In alternativa, puoi usare uno strumento come dumb-init per assicurarti che la tua applicazione sia in grado di gestire i segnali.

Puoi anche usare Container hooks per eseguire uno script o una richiesta HTTP all'avvio o all'arresto del contenitore. L'azione `PreStop` hook viene eseguita prima che il contenitore riceva un `SIGTERM` segnale e deve essere completata prima che questo segnale venga inviato. Il `terminationGracePeriodSeconds` valore si applica a partire dal momento in cui l'azione `PreStop` hook inizia l'esecuzione, non da quando il `SIGTERM` segnale viene inviato.

Raccomandazioni

Proteggi i carichi di lavoro critici con Pod Disruption Budgets

Pod Disruption Budget o PDB possono interrompere temporaneamente il processo di sfratto se il numero di repliche di un'applicazione scende al di sotto della soglia dichiarata. Il processo di eliminazione continuerà una volta che il numero di repliche disponibili avrà superato la soglia. È possibile utilizzare PDB per dichiarare il numero e il `minAvailable` numero di repliche. `maxUnavailable` Ad esempio, se desideri che siano disponibili almeno tre copie della tua app, puoi creare un PDB.

```
apiVersion: policy/v1beta1
kind: PodDisruptionBudget
metadata:
```

```
  name: my-svc-pdb
spec:
  minAvailable: 3
  selector:
    matchLabels:
      app: my-svc
```

La politica PDB di cui sopra indica a Kubernetes di interrompere il processo di sfratto fino a quando non saranno disponibili tre o più repliche. Il `PodDisruptionBudgets` drenaggio dei nodi rispetti. Durante l'aggiornamento di un gruppo di nodi gestito da EKS, i nodi vengono svuotati con un timeout di quindici minuti. Dopo quindici minuti, se l'aggiornamento non è forzato (l'opzione si chiama Rolling update nella console EKS), l'aggiornamento non riesce. Se l'aggiornamento è forzato, i pod vengono eliminati.

Per i nodi autogestiti, puoi anche utilizzare strumenti come AWS Node Termination Handler, che garantisce che il piano di controllo di Kubernetes risponda in modo appropriato agli eventi che possono causare l'indisponibilità dell'istanza EC2, come gli eventi di manutenzione EC2 e le interruzioni Spot EC2. Utilizza l'API Kubernetes per collegare il nodo in modo da garantire che non siano programmati nuovi Pod, quindi lo scarica, interrompendo tutti i Pod in esecuzione.

Puoi utilizzare Pod anti-affinity per pianificare i Deploy's Pod su nodi diversi ed evitare ritardi dovuti al PDB durante gli aggiornamenti dei nodi.

Pratica l'ingegneria del caos

L'ingegneria del caos è la disciplina che consiste nella sperimentazione su un sistema distribuito per aumentare la fiducia nella capacità del sistema di resistere a condizioni di produzione turbolente.

Nel suo blog, Dominik Tornow spiega che Kubernetes è un sistema dichiarativo in cui "l'utente fornisce al sistema una rappresentazione dello stato desiderato del sistema. Il sistema considera quindi lo stato corrente e lo stato desiderato per determinare la sequenza di comandi per la transizione dallo stato corrente allo stato desiderato. » Ciò significa che Kubernetes memorizza sempre lo stato desiderato e, se il sistema si discosta, Kubernetes interverrà per ripristinare lo stato. Ad esempio, se un nodo di lavoro diventa non disponibile, Kubernetes riprogrammerà i Pod su un altro nodo di lavoro. Allo stesso modo, se un si `replica` blocca, Deployment Contoller ne creerà uno nuovo. `replica` In questo modo, i controller Kubernetes correggono automaticamente gli errori.

Strumenti di ingegneria del caos come Gremlin ti aiutano a testare la resilienza del tuo cluster Kubernetes e a identificare singoli punti di errore. Gli strumenti che introducono il caos artificiale nel cluster (e oltre) possono scoprire punti deboli sistemici, offrire l'opportunità di identificare colli di

bottiglia e configurazioni errate e correggere i problemi in un ambiente controllato. La filosofia Chaos Engineering sostiene la necessità di rompere le cose di proposito e l'infrastruttura di stress test per ridurre al minimo i tempi di inattività imprevisti.

Usa una Service Mesh

È possibile utilizzare una service mesh per migliorare la resilienza dell'applicazione. Le service mesh consentono service-to-service la comunicazione e aumentano l'osservabilità della rete di microservizi. La maggior parte dei prodotti service mesh funziona facendo funzionare un piccolo proxy di rete insieme a ciascun servizio che intercetta e ispeziona il traffico di rete dell'applicazione. È possibile inserire l'applicazione in una mesh senza modificare l'applicazione. Utilizzando le funzionalità integrate del service proxy, è possibile fare in modo che generi statistiche di rete, crei registri di accesso e aggiunga intestazioni HTTP alle richieste in uscita per il tracciamento distribuito.

Una service mesh può aiutarti a rendere i tuoi microservizi più resilienti con funzionalità come nuovi tentativi di richiesta automatici, timeout, interruzione del circuito e limitazione della velocità.

Se gestisci più cluster, puoi utilizzare una service mesh per abilitare la comunicazione tra cluster. service-to-service

Service Mesh

- Istio
- Linker D
- Console

Osservabilità

Osservabilità è un termine generico che include monitoraggio, registrazione e tracciamento. Le applicazioni basate su microservizi sono distribuite per natura. A differenza delle applicazioni monolitiche in cui è sufficiente monitorare un singolo sistema, in un'architettura applicativa distribuita è necessario monitorare le prestazioni di ogni componente. È possibile utilizzare sistemi di monitoraggio, registrazione e tracciamento distribuito a livello di cluster per identificare i problemi del cluster prima che possano causare interruzioni ai clienti.

Gli strumenti integrati di Kubernetes per la risoluzione dei problemi e il monitoraggio sono limitati. Il metrics-server raccoglie le metriche delle risorse e le archivia in memoria, ma non le memorizza.

Puoi visualizzare i log di un Pod usando kubectl, ma Kubernetes non conserva automaticamente i log. E l'implementazione della traccia distribuita viene eseguita a livello di codice dell'applicazione o utilizzando mesh di servizi.

L'estensibilità di Kubernetes brilla qui. Kubernetes ti consente di utilizzare la tua soluzione centralizzata preferita di monitoraggio, registrazione e tracciamento.

Raccomandazioni

Monitora le tue applicazioni

Il numero di metriche da monitorare nelle applicazioni moderne è in continua crescita. È utile disporre di un modo automatizzato per tracciare le applicazioni in modo da potersi concentrare sulla risoluzione delle sfide dei clienti. Strumenti di monitoraggio a livello di cluster come Prometheus o CloudWatchContainer Insights possono monitorare il cluster e il carico di lavoro e fornirti segnali quando, o preferibilmente, prima che le cose vadano male.

Gli strumenti di monitoraggio consentono di creare avvisi a cui il team operativo può abbonarsi. Prendi in considerazione le regole per attivare gli allarmi in caso di eventi che, se aggravati, possono portare a un'interruzione o influire sulle prestazioni delle applicazioni.

Se non ti è chiaro quali metriche monitorare, puoi prendere ispirazione da questi metodi:

- Metodo RED. È l'acronimo di richieste, errori e durata.
- Metodo USE. Sta per utilizzo, saturazione ed errori.

Il post di Sysdig sulle migliori pratiche per gli avvisi su Kubernetes include un elenco completo di componenti che possono influire sulla disponibilità delle applicazioni.

Usa la libreria client Prometheus per esporre le metriche delle applicazioni

Oltre a monitorare lo stato dell'applicazione e aggregare le metriche standard, puoi anche utilizzare la libreria client Prometheus per esporre metriche personalizzate specifiche dell'applicazione per migliorare l'osservabilità dell'applicazione.

Utilizza strumenti di registrazione centralizzati per raccogliere e conservare i log

La registrazione in EKS rientra in due categorie: registri del piano di controllo e registri delle applicazioni. La registrazione del piano di controllo EKS fornisce i registri di controllo e diagnostica

direttamente dal piano di controllo ai CloudWatch registri dell'account. I log delle applicazioni sono registri prodotti dai Pod in esecuzione all'interno del cluster. I log delle applicazioni includono i log prodotti dai Pod che eseguono le applicazioni di business logic e i componenti del sistema Kubernetes come CoredNS, Cluster Autoscaler, Prometheus, ecc.

EKS fornisce cinque tipi di log del piano di controllo:

1. Registri dei componenti del server API Kubernetes
2. Audit
3. Autenticatore
4. Responsabile del controller
5. Pianificatore

I registri del controller manager e dello scheduler possono aiutare a diagnosticare problemi del piano di controllo come strozzature ed errori. Per impostazione predefinita, i log del piano di controllo EKS non vengono inviati a Logs. CloudWatch Puoi abilitare la registrazione del piano di controllo e selezionare i tipi di log del piano di controllo EKS che desideri acquisire per ogni cluster del tuo account

La raccolta dei log delle applicazioni richiede l'installazione di uno strumento di aggregazione dei log come Fluent Bit, Fluentd o Container Insights nel cluster. CloudWatch

Gli strumenti di aggregazione dei log di Kubernetes funzionano come ed estraggono i log dei container dai nodi. DaemonSets I log delle applicazioni vengono quindi inviati a una destinazione centralizzata per l'archiviazione. Ad esempio, CloudWatch Container Insights può utilizzare Fluent Bit o Fluentd per raccogliere i log e inviarli a Logs per l'archiviazione. CloudWatch Fluent Bit e Fluentd supportano molti dei più diffusi sistemi di analisi dei log come Elasticsearch e InfluxDB, offrendoti la possibilità di cambiare il backend di archiviazione per i log modificando Fluent bit o la configurazione dei log di Fluentd.

Utilizzate un sistema di tracciamento distribuito per identificare i colli di bottiglia

Una tipica applicazione moderna ha componenti distribuiti sulla rete e la sua affidabilità dipende dal corretto funzionamento di ciascuno dei componenti che compongono l'applicazione. È possibile utilizzare una soluzione di tracciamento distribuito per comprendere come fluiscono le richieste e come comunicano i sistemi. Traces può mostrarvi i punti critici nella rete delle applicazioni e prevenire problemi che possono causare errori a cascata.

Sono disponibili due opzioni per implementare il tracciamento nelle applicazioni: è possibile implementare il tracciamento distribuito a livello di codice utilizzando librerie condivise o utilizzare una service mesh.

L'implementazione del tracciamento a livello di codice può essere svantaggiosa. Con questo metodo, è necessario apportare modifiche al codice. Ciò è ulteriormente complicato se si utilizzano applicazioni poliglotte. Sei anche responsabile della manutenzione di un'altra libreria, per tutti i tuoi servizi.

Le service mesh come LinkerD e Istio possono essere utilizzate per implementare il tracciamento distribuito nell'applicazione con modifiche minime al codice dell'applicazione. È possibile utilizzare Service Mesh per standardizzare la generazione, la registrazione e il tracciamento delle metriche.

Strumenti di tracciamento come AWS X-Ray e Jaeger supportano sia le librerie condivise che le implementazioni di service mesh.

Prendi in considerazione l'utilizzo di uno strumento di tracciamento come AWS X-Ray o Jaeger che supporti entrambe le implementazioni (libreria condivisa e service mesh) in modo da non dover cambiare strumento se successivamente adotterai la service mesh.

Piano di controllo EKS

> **Tip**
>
> Esplora le best practice tramite i workshop Amazon EKS.

Amazon Elastic Kubernetes Service (EKS) è un servizio Kubernetes gestito che semplifica l'esecuzione di Kubernetes su AWS senza dover installare, utilizzare e mantenere il proprio piano di controllo Kubernetes o nodi di lavoro. Funziona a upstream di Kubernetes ed è certificato conforme a Kubernetes. Questa conformità garantisce che EKS supporti le API Kubernetes, proprio come la versione della community open source che puoi installare su EC2 o in locale. Le applicazioni esistenti in esecuzione su Kubernetes upstream sono compatibili con Amazon EKS.

EKS gestisce automaticamente la disponibilità e la scalabilità dei nodi del piano di controllo Kubernetes e sostituisce automaticamente i nodi del piano di controllo non integri.

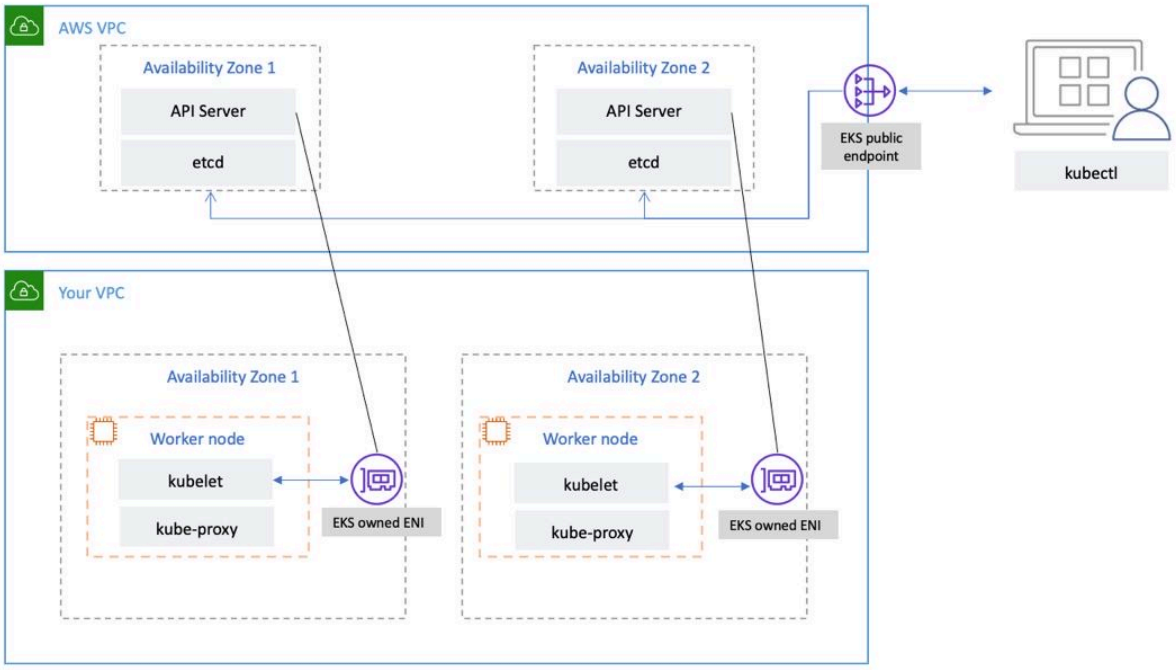
Architettura EKS

L'architettura EKS è progettata per eliminare ogni singolo punto di errore che possa compromettere la disponibilità e la durabilità del piano di controllo Kubernetes.

Il piano di controllo Kubernetes gestito da EKS funziona all'interno di un VPC gestito da EKS. Il piano di controllo EKS comprende i nodi del server dell'API Kubernetes, il cluster ecc. Nodi server API Kubernetes che eseguono componenti come il server API, lo scheduler e vengono eseguiti `kube-controller-manager` in un gruppo di auto-scaling. EKS esegue un minimo di due nodi server API in zone di disponibilità distinte (AZs) all'interno della regione AWS. Allo stesso modo, per motivi di durabilità, i nodi del server etcd funzionano anche in un gruppo di auto-scaling che si estende su tre. AZs EKS esegue un gateway NAT in ogni AZ, mentre i server API e i server etcd vengono eseguiti in una sottorete privata. Questa architettura garantisce che un evento in una singola AZ non influisca sulla disponibilità del cluster EKS.

Quando crei un nuovo cluster, Amazon EKS crea un endpoint ad alta disponibilità per il server API Kubernetes gestito che usi per comunicare con il cluster (utilizzando strumenti come). `kubectl` L'endpoint gestito utilizza NLB per bilanciare il carico dei server API Kubernetes. EKS fornisce anche due ENI diversi AZs per facilitare la comunicazione con i nodi di lavoro.

Connettività di rete EKS Data Plane



Puoi configurare se il server API del tuo cluster Kubernetes è raggiungibile dalla rete Internet pubblica (utilizzando l'endpoint pubblico) o tramite il tuo VPC (utilizzando EKS-Managed) o entrambi. ENIs

Indipendentemente dal fatto che utenti e nodi di lavoro si connettano al server API utilizzando l'endpoint pubblico o l'ENI gestito da EKS, esistono percorsi di connessione ridondanti.

Raccomandazioni

Esamina i seguenti consigli.

Monitora le metriche del piano di controllo

Il monitoraggio delle metriche dell'API Kubernetes può fornirti informazioni dettagliate sulle prestazioni del piano di controllo e identificare i problemi. Un piano di controllo non integro può compromettere la disponibilità dei carichi di lavoro in esecuzione all'interno del cluster. Ad esempio, controller scritti in modo errato possono sovraccaricare i server API, influendo sulla disponibilità dell'applicazione.

Kubernetes espone le metriche del piano di controllo sull'endpoint. `/metrics`

Puoi visualizzare le metriche esposte utilizzando: `kubectl`

```
kubectl get --raw /metrics
```

Queste metriche sono rappresentate in un formato di testo Prometheus.

Puoi usare Prometheus per raccogliere e archiviare queste metriche. A maggio 2020, è CloudWatch stato aggiunto il supporto per il monitoraggio delle metriche di Prometheus in Container Insights. CloudWatch Quindi puoi usare Amazon anche CloudWatch per monitorare il piano di controllo EKS. Puoi utilizzare Tutorial for Adding a New Prometheus Scrape Target: Prometheus KPI Server Metrics per raccogliere metriche e creare dashboard per monitorare il piano di controllo del cluster. CloudWatch

Puoi trovare le metriche del server dell'API Kubernetes qui. Ad esempio, `apiserver_request_duration_seconds` può indicare quanto tempo impiegano le richieste API per l'esecuzione.

Prendi in considerazione il monitoraggio di queste metriche del piano di controllo:

Server API

Metrica	Description
`apiserver_request_total`	Contatore di richieste apiserver suddiviso per ogni verbo, valore di dry run, gruppo, versione, risorsa, ambito, componente e codice di risposta HTTP.
`apiserver_request_duration_seconds*`	Istogramma della latenza di risposta in secondi per ogni verbo, valore dry run, gruppo, versione, risorsa, sottorisorsa, ambito e componente.
`apiserver_admission_controller_admission_duration_seconds*`	Istogramma di latenza del controller di ammissione in secondi, identificato per nome e suddiviso per ogni operazione e risorsa e tipo di API (convalida o ammissione).
`apiserver_admission_webhook_rejection_count`	Numero di accessi respinti dal webhook. Identificato per nome, operazione, rejection_code, tipo (validazione o ammissione), error_type (calling_webhook_error, apiserver_internal_error, no_error)
`rest_client_request_duration_seconds*`	Richiedi l'istogramma di latenza in secondi. Suddiviso per verbo e URL.
`rest_client_requests_total`	Numero di richieste HTTP, partizionate per codice di stato, metodo e host.

- Le metriche dell'istogramma includono i suffissi _bucket, _sum e _count.

ecc.

Metrica	Descri on
`etcd_request_duration_seconds*`	Istogra a di latenz della richies Etcd in secon per ogni operaz e e tipo di oggett
`apiserver_storage_db_total_size_in_bytes` o `apiserver_storage_size_bytes` (a partire da EKS v1.28)	Dimen e del databa Etcd.

- Le metriche dell'istogramma includono i suffissi _bucket, _sum e _count.

Prendi in considerazione l'utilizzo della Kubernetes Monitoring Overview Dashboard per visualizzare e monitorare le richieste del server API Kubernetes e le metriche di latenza e latenza ecc.

> **Important**
>
> Quando viene superato il limite di dimensione del database, etcd emette un allarme di assenza di spazio e smette di accettare ulteriori richieste di scrittura. In altre parole, il cluster diventa di sola lettura e tutte le richieste di modifica degli oggetti, come la creazione di nuovi pod, la scalabilità delle distribuzioni, ecc., verranno rifiutate dal server API del cluster.

Autenticazione del cluster

EKS attualmente supporta due tipi di autenticazione: i token dell'account del portatore/servizio e l'autenticazione IAM che utilizza l'autenticazione tramite token webhook. Quando gli utenti chiamano l'API Kubernetes, un webhook trasmette un token di autenticazione incluso nella richiesta a IAM. Il token, un URL con firma di base 64, viene generato dall'interfaccia a riga di comando AWS (AWS CLI).

L'utente o il ruolo IAM che crea il cluster EKS ottiene automaticamente l'accesso completo al cluster. Puoi gestire l'accesso al cluster EKS modificando la configmap di aws-auth.

Se configuri male la `aws-auth` configmap e perdi l'accesso al cluster, puoi comunque utilizzare l'utente o il ruolo del creatore del cluster per accedere al tuo cluster EKS.

Nell'improbabile eventualità che non sia possibile utilizzare il servizio IAM nella regione AWS, è possibile utilizzare anche il token bearer dell'account del servizio Kubernetes per gestire il cluster.

Crea un `super-admin` account autorizzato a eseguire tutte le azioni nel cluster:

```
kubectl -n kube-system create serviceaccount super-admin
```

Crea un'associazione di ruoli che assegna il ruolo super-admin cluster-admin:

```
kubectl create clusterrolebinding super-admin-rb --clusterrole=cluster-admin --
serviceaccount=kube-system:super-admin
```

Ottieni il segreto dell'account di servizio:

```
SECRET_NAME=`kubectl -n kube-system get serviceaccount/super-admin -o
 jsonpath='{.secrets[0].name}'`
```

Ottieni il token associato al segreto:

```
TOKEN=`kubectl -n kube-system get secret $SECRET_NAME -o jsonpath='{.data.token}'|
 base64 --decode`
```

Aggiungi account e token di servizio a `kubeconfig`:

```
kubectl config set-credentials super-admin --token=$TOKEN
```

Imposta il contesto corrente `kubeconfig` per utilizzare l'account super-admin:

```
kubectl config set-context --current --user=super-admin
```

La finale dovrebbe `kubeconfig` assomigliare a questa:

```
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data:<REDACTED>
    server: https://<CLUSTER>.gr7.us-west-2.eks.amazonaws.com
  name: arn:aws:eks:us-west-2:<account number>:cluster/<cluster name>
contexts:
- context:
    cluster: arn:aws:eks:us-west-2:<account number>:cluster/<cluster name>
    user: super-admin
  name: arn:aws:eks:us-west-2:<account number>:cluster/<cluster name>
current-context: arn:aws:eks:us-west-2:<account number>:cluster/<cluster name>
kind: Config
preferences: {}
users:
#- name: arn:aws:eks:us-west-2:<account number>:cluster/<cluster name>
#  user:
#    exec:
#      apiVersion: client.authentication.k8s.io/v1beta1
#      args:
#      - --region
#      - us-west-2
#      - eks
#      - get-token
#      - --cluster-name
#      - <<cluster name>>
#      command: aws
#      env: null
```

```
- name: super-admin
  user:
    token: <<super-admin sa's secret>>
```

Webhook di ammissione

Kubernetes offre due tipi di webhook di ammissione: webhook di ammissione convalidanti e webhook di ammissione mutanti. Questi consentono a un utente di estendere l'API Kubernetes e convalidare o modificare gli oggetti prima che vengano accettati dall'API. Una configurazione inadeguata di questi webhook può destabilizzare il piano di controllo EKS bloccando le operazioni critiche del cluster.

Per evitare di influire sulle operazioni critiche del cluster, evita di impostare webhook «catch-all» come i seguenti:

```
- name: "pod-policy.example.com"
  rules:
  - apiGroups:   ["*"]
    apiVersions: ["*"]
    operations:  ["*"]
    resources:   ["*"]
    scope: "*"
```

In alternativa, assicurati che il webhook abbia una policy di fail-open con un timeout inferiore a 30 secondi per assicurarti che, se il webhook non è disponibile, non comprometta i carichi di lavoro critici del cluster.

Blocca i pod con unsafe **`sysctls`**

`Sysctl`è un'utilità Linux che consente agli utenti di modificare i parametri del kernel durante l'esecuzione. Questi parametri del kernel controllano vari aspetti del comportamento del sistema operativo, come la rete, il file system, la memoria virtuale e la gestione dei processi.

Kubernetes consente di `sysctl` assegnare profili per i Pod. Kubernetes si classifica come sicuro e non sicuro. `systcls` `sysctls`I namespace sono sicuri nel contenitore o nel Pod e la loro impostazione non influisce sugli altri Pod sul nodo o sul nodo stesso. Al contrario, i sysctls non sicuri sono disabilitati per impostazione predefinita poiché possono potenzialmente interrompere altri Pod o rendere instabile il nodo.

Poiché gli unsafe `sysctls` sono disabilitati per impostazione predefinita, il kubelet non creerà un Pod con un profilo non sicuro. `sysctl` Se crei un Pod di questo tipo, lo scheduler assegnerà

ripetutamente tali Pod ai nodi, mentre il nodo non riesce ad avviarlo. Questo ciclo infinito alla fine mette a dura prova il piano di controllo del cluster, rendendolo instabile.

Prendi in considerazione l'utilizzo di OPA Gatekeeper o Kyverno per rifiutare i Pods con unsafe. `sysctls`

Gestione degli aggiornamenti del cluster

Da aprile 2021, il ciclo di rilascio di Kubernetes è stato modificato da quattro versioni all'anno (una volta al trimestre) a tre versioni all'anno. Una nuova versione secondaria (come 1. 21 o 1. 22) viene rilasciato all'incirca ogni quindici settimane. A partire da Kubernetes 1.19, ogni versione secondaria è supportata per circa dodici mesi dopo il primo rilascio. Con l'avvento di Kubernetes v1.28, il divario di compatibilità tra il piano di controllo e i nodi di lavoro si è esteso dalle versioni minori n-2 a n-3. Per saperne di più, consulta Best Practices for Cluster Upgrades.

Connettività degli endpoint del cluster

Quando lavori con Amazon EKS (Elastic Kubernetes Service), potresti riscontrare timeout o errori di connessione durante eventi come il ridimensionamento o l'applicazione di patch del piano di controllo di Kubernetes. Questi eventi possono causare la sostituzione delle istanze kube-apiserver, con la potenziale conseguenza della restituzione di indirizzi IP diversi durante la risoluzione del nome di dominio completo. Questo documento descrive le migliori pratiche per gli utenti dell'API Kubernetes per mantenere una connettività affidabile.

> **Note**
>
> L'implementazione di queste best practice può richiedere aggiornamenti alle configurazioni o agli script dei client per gestire efficacemente le nuove strategie di risoluzione e riprovare il DNS.

Il problema principale deriva dalla memorizzazione nella cache DNS lato client e dalla potenziale presenza di indirizzi IP obsoleti degli endpoint EKS: NLB pubblico per endpoint pubblici o X-ENI per endpoint privati. Quando le istanze kube-apiserver vengono sostituite, il Fully Qualified Domain Name (FQDN) può trasformarsi in nuovi indirizzi IP. Tuttavia, a causa delle impostazioni DNS Time to Live (TTL), che sono impostate su 60 secondi nella zona Route 53 gestita da AWS, i client possono continuare a utilizzare indirizzi IP obsoleti per un breve periodo di tempo.

Per mitigare questi problemi, gli utenti delle API Kubernetes (come kubectl, CI/CD pipeline e applicazioni personalizzate) devono implementare le seguenti best practice:

- Implementare la risoluzione DNS
- Implementa nuovi tentativi con Backoff e Jitter. Ad esempio, consulta questo articolo intitolato Failures Happen
- Implementa i timeout dei client. Imposta i timeout appropriati per evitare che le richieste di lunga durata blocchino l'applicazione. Tieni presente che alcune librerie client Kubernetes, in particolare quelle generate dai generatori OpenAPI, potrebbero non consentire di impostare facilmente timeout personalizzati.
  - Esempio 1 con kubectl:

```
kubectl get pods --request-timeout 10s # default: no timeout
```

  - Esempio 2 con Python: il client Kubernetes fornisce un parametro _request_timeout

Implementando queste best practice, puoi migliorare in modo significativo l'affidabilità e la resilienza delle tue applicazioni quando interagisci con l'API Kubernetes. Ricorda di testare accuratamente queste implementazioni, specialmente in condizioni di errore simulate, per assicurarti che si comportino come previsto durante gli eventi di scalabilità o applicazione delle patch effettivi.

Esecuzione di cluster di grandi dimensioni

EKS monitora attivamente il carico sulle istanze del piano di controllo e le ridimensiona automaticamente per garantire prestazioni elevate. Tuttavia, è necessario tenere conto dei potenziali problemi e limiti di prestazioni all'interno di Kubernetes e delle quote nei servizi AWS quando si eseguono cluster di grandi dimensioni.

- Secondo i test eseguiti dal team, i cluster con più di 1000 servizi potrebbero presentare una latenza di rete utilizzando `iptables` la modalità `kube-proxy` in. ProjectCalico La soluzione è passare alla modalità di esecuzione `kube-proxy`. `ipvs`
- Potresti anche riscontrare una limitazione delle richieste dell'API EC2 se il CNI deve richiedere gli indirizzi IP per i Pod o se devi creare nuove istanze EC2 frequentemente. Puoi ridurre le chiamate all'API EC2 configurando il CNI per memorizzare nella cache gli indirizzi IP. Puoi utilizzare tipi di istanze EC2 più grandi per ridurre gli eventi di scalabilità EC2.

Risorse aggiuntive:

- Demistificazione delle reti di cluster per i nodi di lavoro Amazon EKS
- Controllo dell'accesso agli endpoint del cluster Amazon EKS
- AWS re:Invent 2019: Amazon EKS sotto il cofano (-R1) CON421

Piano dati EKS

Per utilizzare applicazioni ad alta disponibilità e resilienza, è necessario un piano dati ad alta disponibilità e resilienza. Un piano dati elastico garantisce che Kubernetes possa scalare e correggere automaticamente le tue applicazioni. Un piano dati resiliente è costituito da due o più nodi di lavoro, può crescere e ridursi con il carico di lavoro e ripristinarsi automaticamente in caso di guasti.

Hai diverse scelte per i nodi di lavoro con EKS: nodi gestiti in modalità automatica EKS, istanze EC2 e Fargate.

EKS Auto Mode offre il percorso più semplice verso un piano dati resiliente. Auto Mode estende la gestione AWS dei cluster Kubernetes oltre il cluster stesso, per consentire ad AWS di configurare e gestire anche l'infrastruttura che consente il funzionamento regolare dei carichi di lavoro. La modalità automatica ridimensiona automaticamente il piano dati verso l'alto o verso il basso man mano che Kubernetes ridimensiona i pod e lavora per garantire continuamente che i nodi del cluster siano dimensionati in modo appropriato ed economico per i carichi di lavoro attualmente in esecuzione.

Se scegli le istanze EC2, puoi gestire tu stesso i nodi di lavoro o utilizzare i gruppi di nodi gestiti da EKS. Puoi avere un cluster con una combinazione di modalità automatica, nodi di lavoro gestiti e autogestiti e Fargate.

Fargate esegue ogni Pod in un ambiente di elaborazione isolato. Ogni Pod in esecuzione su Fargate ha il proprio nodo di lavoro. Fargate ridimensiona automaticamente il piano dati come Kubernetes ridimensiona i pod. È possibile scalare sia il piano dati che il carico di lavoro utilizzando l'autoscaler a pod orizzontale.

Il modo preferito per scalare i nodi di lavoro EC2 (se non si utilizza la modalità automatica EKS, che viene eseguita automaticamente da AWS) è utilizzare i gruppi Karpenter, Kubernetes Cluster Autoscaler o EC2 Auto Scaling.

Raccomandazioni

Distribuisci i nodi di lavoro e i carichi di lavoro su più AZs

Puoi proteggere i tuoi carichi di lavoro dai guasti in una singola AZ eseguendo nodi di lavoro e Pods in più aree. AZs È possibile controllare l'AZ in cui vengono creati i nodi di lavoro utilizzando le sottoreti in cui vengono creati i nodi.

Il metodo consigliato per distribuire i pod AZs consiste nell'utilizzare Topology Spread Constraints for Pods. Le funzionalità di scalabilità automatica come EKS Auto Mode e Karpenter sono consapevoli dei vincoli di diffusione della topologia e avvieranno automaticamente i nodi nella posizione corretta per consentire il rispetto dei vincoli imposti. AZs

La distribuzione riportata di seguito distribuisce i pod, se possibile, lasciando che funzionino comunque in caso contrario AZs :

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-server
spec:
  replicas: 3
  selector:
    matchLabels:
      app: web-server
  template:
    metadata:
      labels:
        app: web-server
    spec:
      topologySpreadConstraints:
        - maxSkew: 1
          whenUnsatisfiable: ScheduleAnyway
          topologyKey: topology.kubernetes.io/zone
          labelSelector:
            matchLabels:
              app: web-server
      containers:
      - name: web-app
        image: nginx
        resources:
          requests:
```

```
            cpu: 1
```

> **Note**
>
> `kube-scheduler`è a conoscenza dei domini di topologia solo tramite nodi che esistono con tali etichette. Se la distribuzione di cui sopra viene distribuita in un cluster con nodi in una sola zona, tutti i pod verranno programmati su quei nodi poiché `kube-scheduler` non sono a conoscenza delle altre zone. Affinché questa distribuzione della topologia funzioni come previsto con lo scheduler, i nodi devono già esistere in tutte le zone. La `minDomains` proprietà dei vincoli di diffusione della topologia viene utilizzata per comunicare allo scheduler il numero di domini idonei, anche se è in esecuzione un nodo per evitare questo problema.

> **Warning**
>
> L'impostazione `whenUnsatisfiable` su `DoNotSchedule` farà sì che i pod non siano programmabili se il vincolo di diffusione della topologia non può essere soddisfatto. Dovrebbe essere impostato solo se è preferibile che i pod non vengano eseguiti invece di violare il vincolo di diffusione della topologia.

Nelle versioni precedenti di Kubernetes, puoi utilizzare le regole di antiaffinità dei pod per pianificare i pod su più pod. AZs Il manifesto riportato di seguito indica allo scheduler di Kubernetes di preferire la pianificazione dei pod in modo distinto. AZs

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-server
  labels:
    app: web-server
spec:
  replicas: 4
  selector:
    matchLabels:
      app: web-server
  template:
    metadata:
      labels:
```

```
        app: web-server
    spec:
      affinity:
        podAntiAffinity:
          preferredDuringSchedulingIgnoredDuringExecution:
          - podAffinityTerm:
              labelSelector:
                matchExpressions:
                - key: app
                  operator: In
                  values:
                  - web-server
              topologyKey: failure-domain.beta.kubernetes.io/zone
            weight: 100
      containers:
      - name: web-app
        image: nginx
```

> **Warning**
>
> Non è necessario che i pod siano pianificati in AZs modo distinto, altrimenti il numero di pod in una distribuzione non supererà mai il numero di. AZs

Garantisci la possibilità di avviare nodi in ogni AZ quando usi i volumi EBS

Se utilizzi Amazon EBS per fornire volumi persistenti, devi assicurarti che i pod e il volume EBS associato si trovino nella stessa zona di disponibilità. Un pod non può accedere ai volumi persistenti supportati da EBS che si trovano in una AZ diversa. Lo scheduler Kubernetes sa in quale AZ si trova un nodo di lavoro dalle etichette che si trovano sul nodo e pianificherà sempre un Pod che richiede un volume EBS nella stessa AZ del volume. Tuttavia, se non ci sono nodi di lavoro disponibili nell'AZ in cui si trova il volume, il Pod non può essere pianificato.

Se utilizzi EKS Auto Mode o Karpenter, dovrai assicurarti di NodeClass selezionare le sottoreti in ogni AZ. Se si utilizzano i gruppi di nodi gestiti, è necessario assicurarsi di disporre di un gruppo di nodi in ogni AZ.

Una funzionalità di storage EBS è integrata in EKS Auto Mode, ma se si utilizza Karpenter o Managed Node Groups, sarà necessario installare anche EBS CSI.

Usa EKS Auto Mode per gestire i nodi di lavoro

EKS Auto Mode semplifica la gestione EKS fornendo cluster pronti per la produzione con un sovraccarico operativo minimo. La modalità automatica è responsabile dell'aumento o della riduzione del numero di nodi a seconda dei pod in esecuzione nel cluster. I nodi vengono aggiornati automaticamente con patch e correzioni software, e gli aggiornamenti vengono eseguiti in base alle impostazioni di NodePoolinterruzione configurate e ai Pod Disruption Budgets.

Esegui il Node Monitoring Agent

Il Node Monitoring Agent monitora e reagisce ai problemi di integrità dei nodi pubblicando gli eventi Kubernetes e aggiornando le condizioni di stato sui nodi. Il Node Monitoring Agent è incluso nei nodi EKS Auto Mode e può essere installato come componente aggiuntivo EKS per i nodi che non sono gestiti da Auto Mode.

EKS Auto Mode, Managed Node Groups e Karpenter hanno tutti la capacità di rilevare le condizioni fatali dei nodi segnalate dall'agente di monitoraggio dei nodi e di ripararli automaticamente quando si verificano tali condizioni.

Implementazione del QoS

Per le applicazioni critiche, prendi in considerazione la definizione di `requests` = `limits` per il contenitore nel Pod. Ciò garantirà che il contenitore non venga ucciso se un altro Pod richiede risorse.

È consigliabile implementare limiti di CPU e memoria per tutti i contenitori in quanto impedisce che un contenitore consumi inavvertitamente risorse di sistema influendo sulla disponibilità di altri processi co-localizzati.

Configura e dimensiona le risorse per tutti i carichi di lavoro Requests/Limits

Alcune indicazioni generali possono essere applicate al dimensionamento delle richieste di risorse e ai limiti per i carichi di lavoro:

- Non specificare limiti di risorse sulla CPU. In assenza di limiti, la richiesta influisce sulla quantità di tempo relativo alla CPU utilizzata dai container. Ciò consente ai carichi di lavoro di utilizzare l'intera CPU senza limiti artificiali o carenze.
- Per le risorse diverse dalla CPU, configuring `requests` = `limits` fornisce il comportamento più prevedibile. Se! `requests` `=limits`, inoltre, il QOS del container è stato ridotto da Guaranteed a Burstable, il che rende più probabile lo sfratto in caso di pressione dei nodi.

- Per le risorse diverse dalla CPU, non specificare un limite molto più grande della richiesta. Quanto più grandi `limits` sono le dimensioni configurate rispetto `arequests`, tanto più è probabile che i nodi vengano sovraccaricati, con conseguenti alte probabilità di interruzione del carico di lavoro.
- Le richieste di dimensioni corrette sono particolarmente importanti quando si utilizza una soluzione di auto-scaling dei nodi come Karpenter o Cluster. AutoScaler Questi strumenti esaminano le richieste di carico di lavoro per determinare il numero e la dimensione dei nodi da fornire. Se le tue richieste sono troppo piccole e con limiti più elevati, potresti scoprire che i carichi di lavoro vengono eliminati o l'OOM bloccato se sono stati raggruppati in un nodo ristretto.

Determinare le richieste di risorse può essere difficile, ma strumenti come Vertical Pod Autoscaler possono aiutarvi a «dimensionare correttamente» le richieste osservando l'utilizzo delle risorse del contenitore in fase di esecuzione. Altri strumenti che possono essere utili per determinare le dimensioni delle richieste includono:

- Riccioli d'oro
- Parca
- Profiler
- rg

Configura le quote di risorse per i namespace

I namespace sono concepiti per l'uso negli ambienti con molti utenti distribuiti su più team o progetti. Forniscono un ambito per i nomi e sono un modo per dividere le risorse del cluster tra più team, progetti e carichi di lavoro. È possibile limitare il consumo aggregato di risorse in un namespace. L'`ResourceQuota`oggetto può limitare la quantità di oggetti che possono essere creati in un namespace per tipo, nonché la quantità totale di risorse di elaborazione che possono essere consumate dalle risorse di quel progetto. È possibile limitare la somma totale di risorse di archiviazione and/or (CPU e memoria) che possono essere richieste in un determinato spazio dei nomi.

Se la quota di risorse è abilitata per uno spazio dei nomi per risorse di calcolo come CPU e memoria, gli utenti devono specificare le richieste o i limiti per ogni contenitore in tale spazio dei nomi.

Prendi in considerazione la possibilità di configurare le quote per ogni namespace. Prendi in considerazione l'utilizzo `LimitRanges` per applicare automaticamente limiti preconfigurati ai contenitori all'interno di un namespace.

Limita l'utilizzo delle risorse del contenitore all'interno di un namespace

Le quote di risorse aiutano a limitare la quantità di risorse che un namespace può utilizzare. L'`LimitRange`oggetto può aiutarti a implementare le risorse minime e massime richieste da un contenitore. Utilizzando `LimitRange` you puoi impostare una richiesta e dei limiti predefiniti per i contenitori, il che è utile se l'impostazione dei limiti delle risorse di elaborazione non è una pratica standard nell'organizzazione. Come suggerisce il nome, `LimitRange` può imporre l'utilizzo minimo e massimo delle risorse di calcolo per Pod o Container in un namespace. Inoltre, applica la richiesta di archiviazione minima e massima per in un namespace. PersistentVolumeClaim

Valuta la possibilità di `ResourceQuota` utilizzarlo insieme `LimitRange` a per imporre limiti a livello di contenitore e namespace. L'impostazione di questi limiti garantirà che un contenitore o un namespace non influiscano sulle risorse utilizzate dagli altri tenant del cluster.

Usa NodeLocal DNSCache

È possibile migliorare le prestazioni del Cluster DNS NodeLocalDNSCacheeseguendo. Questa funzionalità esegue un agente di caching DNS sui nodi del cluster come. DaemonSet Tutti i pod utilizzano l'agente di caching DNS in esecuzione sul nodo per la risoluzione dei nomi anziché utilizzare Service. `kube-dns` Questa funzionalità è inclusa automaticamente in EKS Auto Mode.

Configurare CoredNS con scalabilità automatica

Un altro metodo per migliorare le prestazioni del Cluster DNS consiste nell'abilitare l'auto-scaling integrato dei CoredNS Pods.

Questa funzionalità monitora continuamente lo stato del cluster, incluso il numero di nodi e core della CPU. Sulla base di tali informazioni, il controller adatterà dinamicamente il numero di repliche dell'implementazione CoreDNS in un cluster EKS.

Le migliori pratiche per il networking

> **Tip**
>
> Esplora le best practice tramite i workshop Amazon EKS.

È fondamentale comprendere il networking Kubernetes per gestire il cluster e le applicazioni in modo efficiente. Il Pod networking, chiamato anche cluster networking, è il fulcro del networking Kubernetes. Kubernetes supporta i plugin Container Network Interface (CNI) per le reti di cluster.

Amazon EKS supporta ufficialmente il plug-in CNI Amazon Virtual Private Cloud (VPC) per implementare il networking Kubernetes Pod. Il VPC CNI fornisce l'integrazione nativa con AWS VPC e funziona in modalità underlay. In modalità underlay, i pod e gli host si trovano sullo stesso livello di rete e condividono lo spazio dei nomi di rete. L'indirizzo IP del Pod è coerente dal punto di vista del cluster e del VPC.

Questa guida presenta l'Amazon VPC Container Network Interface (VPC CNI) nel contesto della rete di cluster Kubernetes. Il VPC CNI è il plug-in di rete predefinito supportato da EKS e quindi è al centro della guida. Il VPC CNI è altamente configurabile per supportare diversi casi d'uso. Questa guida include inoltre sezioni dedicate su diversi casi d'uso, modalità operative e sottocomponenti di VPC CNI, seguite dai consigli.

Amazon EKS esegue Kubernetes upstream ed è certificato conforme a Kubernetes. Sebbene sia possibile utilizzare plugin CNI alternativi, questa guida non fornisce consigli per la gestione dei plugin alternativi. CNIs Consulta la documentazione EKS Alternate CNI per un elenco di partner e risorse per una gestione efficace delle alternative. CNIs

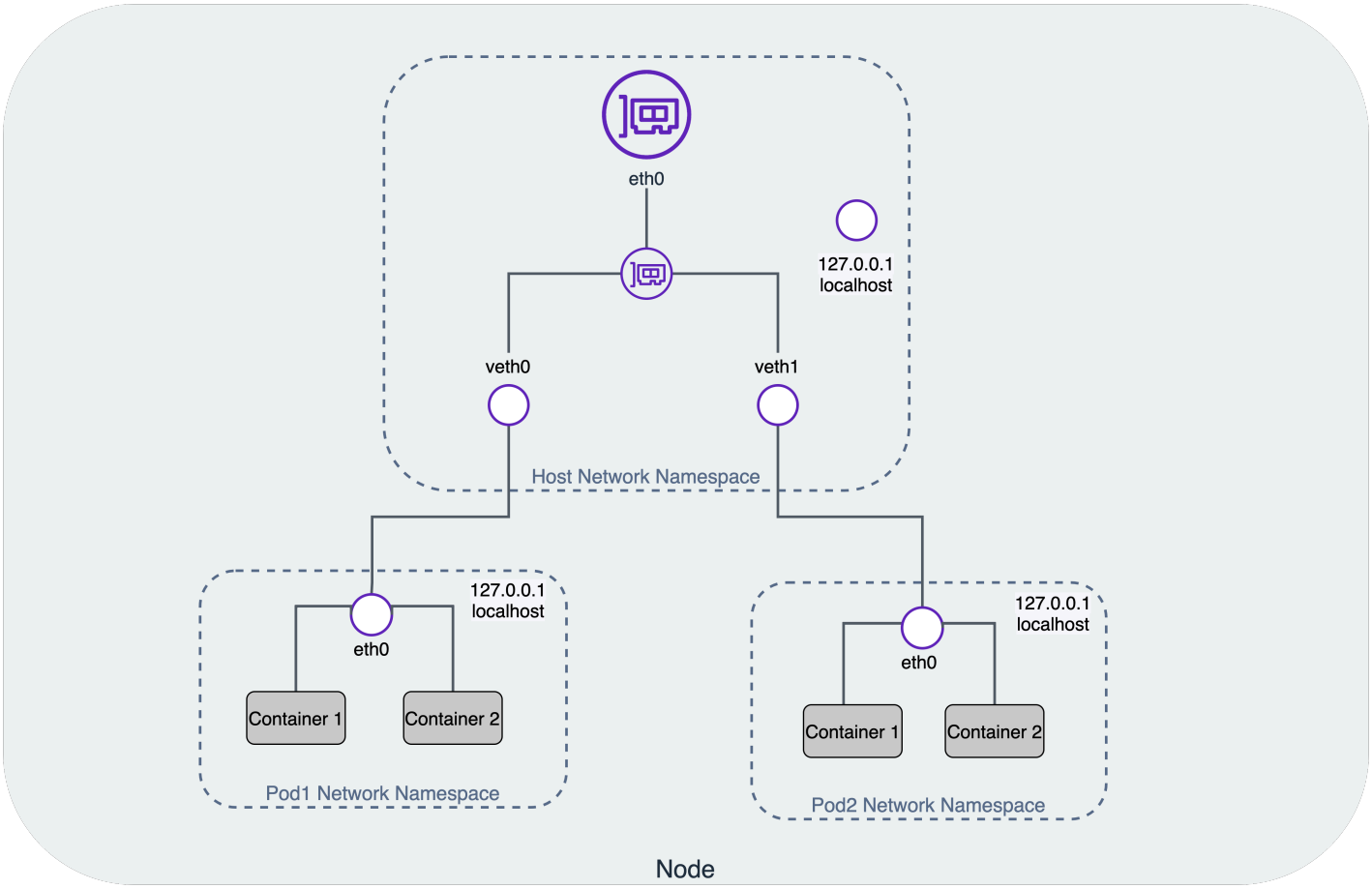
Modello di rete Kubernetes

Kubernetes stabilisce i seguenti requisiti per le reti di cluster:

- I pod programmati sullo stesso nodo devono essere in grado di comunicare con altri Pod senza utilizzare NAT (Network Address Translation).
- Tutti i demoni di sistema (processi in background, ad esempio kubelet) in esecuzione su un particolare nodo possono comunicare con i Pod in esecuzione sullo stesso nodo.

- I pod che utilizzano la rete host devono essere in grado di contattare tutti gli altri Pod su tutti gli altri nodi senza utilizzare NAT.

Consulta il modello di rete Kubernetes per i dettagli su ciò che Kubernetes si aspetta dalle implementazioni di rete compatibili. La figura seguente illustra la relazione tra gli spazi dei nomi della rete Pod e lo spazio dei nomi della rete host.



Container Networking Interface (CNI)

Kubernetes supporta le specifiche e i plugin CNI per implementare il modello di rete Kubernetes. Un CNI è costituito da una specifica (versione attuale 1.0.0) e da librerie per la scrittura di plugin per configurare le interfacce di rete nei contenitori, oltre a una serie di plugin supportati. Il CNI si occupa solo della connettività di rete dei contenitori e della rimozione delle risorse allocate quando il contenitore viene eliminato.

Il plugin CNI viene abilitato passando a kubelet l'opzione della riga di comando. `--network-plugin=cni` Kubelet legge un file da `--cni-conf-dir` (default etc/cni/net /.d) e utilizza la

configurazione CNI da quel file per configurare la rete di ciascun Pod. Il file di configurazione CNI deve corrispondere alla specifica CNI (minimo v0.4.0) e tutti i plugin CNI richiesti a cui fa riferimento la configurazione devono essere presenti nella directory (default/). `--cni-bin-dir` opt/cni/bin Se nella directory sono presenti più file di configurazione CNI, kubelet utilizza il file di configurazione che viene visualizzato per primo per nome in ordine lessicografico.

Amazon Virtual Private Cloud (VPC) CNI

Il VPC CNI fornito da AWS è il componente aggiuntivo di rete predefinito per i cluster EKS. Il componente aggiuntivo VPC CNI viene installato per impostazione predefinita durante il provisioning dei cluster EKS. VPC CNI viene eseguito su nodi di lavoro Kubernetes. Il componente aggiuntivo VPC CNI è costituito dal binario CNI e dal plug-in IP Address Management (ipamd). Il CNI assegna un indirizzo IP dalla rete VPC a un Pod. ipamd gestisce AWS Elastic Networking Interfaces (ENIs) su ogni nodo Kubernetes e mantiene il pool caldo di. IPs Il VPC CNI offre opzioni di configurazione per la preallocazione ENIs e gli indirizzi IP per tempi di avvio rapidi dei Pod. Consulta Amazon VPC CNI per le best practice consigliate per la gestione dei plugin.

Amazon EKS consiglia di specificare le sottoreti in almeno due zone di disponibilità quando si crea un cluster. Amazon VPC CNI alloca gli indirizzi IP ai pod dalle sottoreti dei nodi. Consigliamo vivamente di verificare la presenza di indirizzi IP disponibili nelle sottoreti. Prendi in considerazione i consigli su VPC e Subnet prima di implementare i cluster EKS.

Amazon VPC CNI alloca un pool di indirizzi IP caldi ENIs e secondari dalla sottorete collegata all'ENI primario del nodo. Questa modalità di VPC CNI è chiamata modalità IP secondaria. Il numero di indirizzi IP e quindi il numero di Pod (densità dei Pod) è definito dal numero di indirizzi IP ENIs e dall'indirizzo IP per ENI (limiti) definito dal tipo di istanza. La modalità secondaria è quella predefinita e funziona bene per piccoli cluster con tipi di istanze più piccoli. Prendi in considerazione l'utilizzo della modalità prefisso se riscontri problemi di densità dei pod. Puoi anche aumentare gli indirizzi IP disponibili sul nodo per i Pod assegnando prefissi a. ENIs

Amazon VPC CNI si integra nativamente con AWS VPC e consente agli utenti di applicare le best practice di rete e sicurezza AWS VPC esistenti per la creazione di cluster Kubernetes. Ciò include la possibilità di utilizzare log di flusso VPC, politiche di routing VPC e gruppi di sicurezza per l'isolamento del traffico di rete. Per impostazione predefinita, il CNI di Amazon VPC applica ai Pods il gruppo di sicurezza associato all'ENI primario sul nodo. Valuta la possibilità di abilitare i gruppi di sicurezza per i Pod quando desideri assegnare regole di rete diverse a un Pod.

Per impostazione predefinita, VPC CNI assegna gli indirizzi IP ai Pod dalla sottorete assegnata all'ENI primario di un nodo. È comune riscontrare una carenza di IPv4 indirizzi quando si gestiscono cluster di grandi dimensioni con migliaia di carichi di lavoro. AWS VPC consente di estendere la disponibilità IPs assegnando un secondario CIDRs per ovviare all'esaurimento dei blocchi CIDR. IPv4 AWS VPC CNI consente di utilizzare un intervallo CIDR di sottorete diverso per i pod. Questa funzionalità di VPC CNI si chiama rete personalizzata. Potresti prendere in considerazione l'utilizzo di reti personalizzate da utilizzare `100.64.0.0/10` e `198.19.0.0/16` CIDRs (CG-NAT) con EKS. In questo modo puoi creare efficacemente un ambiente in cui i Pod non consumano più alcun RFC1918 indirizzo IP dal tuo VPC.

La rete personalizzata è un'opzione per risolvere il problema dell'esaurimento degli IPv4 indirizzi, ma richiede un sovraccarico operativo. Per risolvere questo problema, consigliamo di utilizzare IPv6 cluster anziché reti personalizzate. In particolare, ti consigliamo di effettuare la migrazione ai IPv6 cluster se hai completamente esaurito tutto lo spazio di IPv4 indirizzi disponibile per il tuo VPC. Valuta i piani di supporto IPv6 della tua organizzazione e valuta se investire in qualcosa IPv6 possa avere più valore a lungo termine.

Il supporto di EKS IPv6 si concentra sulla risoluzione del problema dell'esaurimento dell'IP causato da uno spazio di IPv4 indirizzi limitato. In risposta ai problemi di IPv4 esaurimento dei clienti, EKS ha dato la priorità ai IPv6 soli pod rispetto ai pod dual-stack. Cioè, i Pod possono essere in grado di accedere alle IPv4 risorse, ma a loro non viene assegnato un IPv4 indirizzo dall'intervallo CIDR VPC. Il VPC CNI assegna IPv6 gli indirizzi ai pod dal blocco CIDR VPC gestito da AWS. IPv6

Calcolatore di sottorete

Questo progetto include un documento Excel di Subnet Calculator. Questo documento di calcolo simula il consumo di indirizzi IP di un carico di lavoro specifico in diverse opzioni di configurazione ENI, come e. `WARM_IP_TARGET` `WARM_ENI_TARGET` Il documento include due fogli, il primo per la modalità Warm ENI e il secondo per la modalità Warm IP. Consulta la guida VPC CNI per ulteriori informazioni su queste modalità.

Ingressi:

- Dimensione CIDR della sottorete
- Target ENI caldo o target IP caldo
- Elenco delle istanze
  - tipo, numero e numero di pod di carico di lavoro pianificati per istanza

Uscite:

- Numero totale di pod ospitati
- Numero di sottoreti utilizzate IPs
- Numero di sottorete IPs rimanenti
- Dettagli a livello di istanza
  - Numero di Warm IPs/ENIs per istanza
  - Numero di attivi IPs/ENIs per istanza

Considerazioni su VPC e sottorete

> **Tip**
>
> Esplora le best practice tramite i workshop Amazon EKS.

La gestione di un cluster EKS richiede la conoscenza della rete AWS VPC, oltre alla rete Kubernetes.

Ti consigliamo di comprendere i meccanismi di comunicazione del piano di controllo EKS prima di iniziare a progettare il tuo VPC o implementare i cluster in quelli esistenti. VPCs

Fai riferimento alle considerazioni su Cluster VPC e alle considerazioni sul gruppo di sicurezza Amazon EKS quando progetti un VPC e delle sottoreti da utilizzare con EKS.

Panoramica di

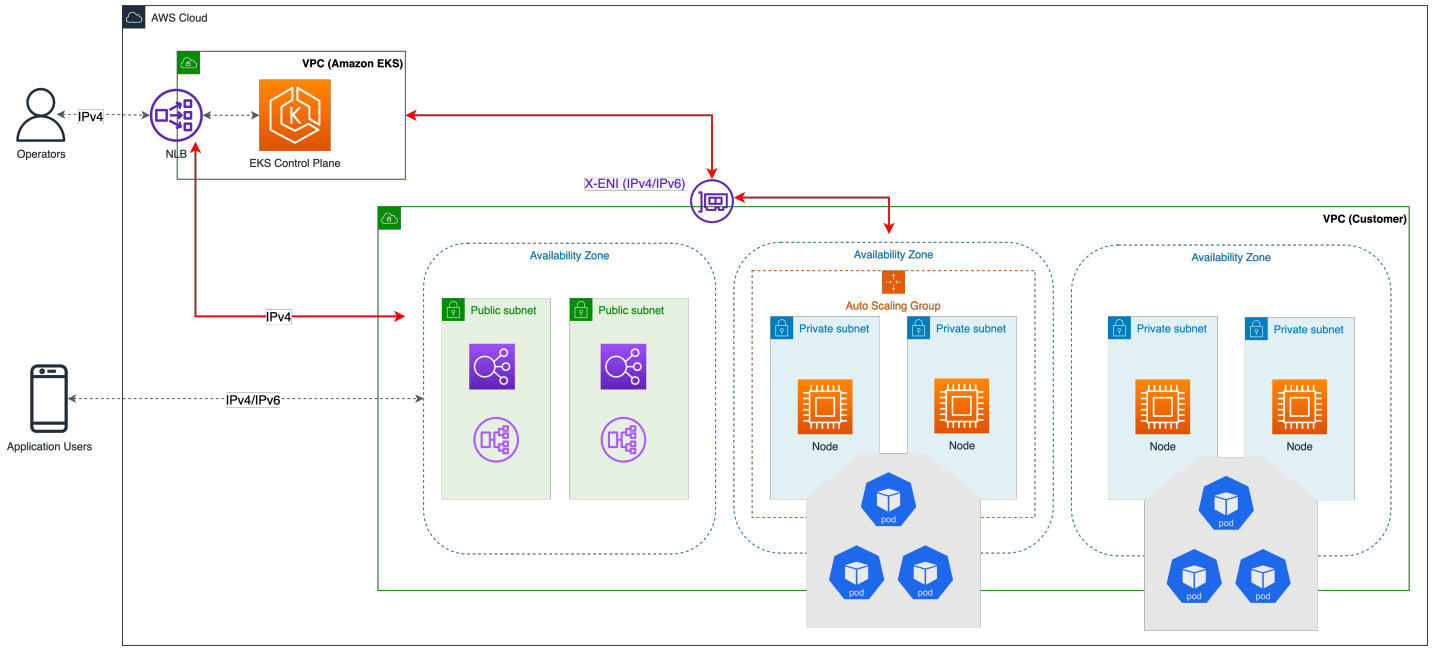
Architettura del cluster EKS

Un cluster EKS è composto da due VPCs:

- Un VPC gestito da AWS che ospita il piano di controllo Kubernetes. Questo VPC non viene visualizzato nell'account del cliente.
- Un VPC gestito dal cliente che ospita i nodi Kubernetes. È qui che vengono eseguiti i container e altre infrastrutture AWS gestite dai clienti, come i sistemi di bilanciamento del carico utilizzati dal cluster. Questo VPC viene visualizzato nell'account del cliente. È necessario creare un VPC gestito dal cliente prima di creare un cluster. L'eksctl crea un VPC se non ne fornisci uno.

I nodi del VPC del cliente necessitano della possibilità di connettersi all'endpoint del server API gestito nel VPC AWS. Ciò consente ai nodi di registrarsi nel piano di controllo di Kubernetes e ricevere richieste per eseguire i Pod delle applicazioni.

I nodi si connettono al piano di controllo EKS tramite (a) un endpoint pubblico EKS o (b) un'interfaccia di rete elastica Cross-Account (X-ENI) gestita da EKS. Quando viene creato un cluster, è necessario specificare almeno due sottoreti VPC. EKS inserisce un X-ENI in ogni sottorete specificata durante la creazione del cluster (denominata anche sottoreti del cluster). Il server API Kubernetes utilizza questi Cross-Account ENIs per comunicare con i nodi distribuiti nelle sottoreti VPC del cluster gestite dal cliente.



All'avvio del nodo, viene eseguito lo script di bootstrap EKS e vengono installati i file di configurazione del nodo Kubernetes. Come parte del processo di avvio su ogni istanza, vengono avviati gli agenti di runtime del contenitore, gli agenti kubelet e i nodi Kubernetes.

Per registrare un nodo, Kubelet contatta l'endpoint del cluster Kubernetes. Stabilisce una connessione con l'endpoint pubblico esterno al VPC o con l'endpoint privato all'interno del VPC. Kubelet riceve istruzioni API e fornisce regolarmente aggiornamenti di stato e battiti cardiaci all'endpoint.

Comunicazione EKS Control Plane

EKS offre due modi per controllare l'accesso all'endpoint del cluster. Il controllo degli accessi agli endpoint ti consente di scegliere se l'endpoint può essere raggiunto dalla rete Internet pubblica o

solo tramite il tuo VPC. Puoi attivare l'endpoint pubblico (che è l'impostazione predefinita), l'endpoint privato o entrambi contemporaneamente.

La configurazione dell'endpoint dell'API del cluster determina il percorso utilizzato dai nodi per comunicare con il piano di controllo. Tieni presente che queste impostazioni degli endpoint possono essere modificate in qualsiasi momento tramite la console EKS o l'API.

Endpoint pubblico

Questo è il comportamento di default per nuovi cluster Amazon EKS. Quando è abilitato solo l'endpoint pubblico per il cluster, le richieste API Kubernetes che provengono dal VPC del cluster (ad esempio dal nodo di lavoro al piano di controllo della comunicazione) escono dal VPC, ma non dalla rete di Amazon. Affinché i nodi possano connettersi al piano di controllo, devono disporre di un indirizzo IP pubblico e un percorso verso un gateway Internet o un percorso verso un gateway NAT dove possono utilizzare l'indirizzo IP pubblico del gateway NAT.

Endpoint pubblico e privato

Quando gli endpoint pubblici e privati sono abilitati, le richieste API Kubernetes dall'interno del VPC comunicano al piano di controllo tramite la X all'interno del VPC. ENIs Il server API del cluster è accessibile da Internet.

Endpoint privato

Non c'è accesso pubblico al server API da Internet quando è abilitato solo un endpoint privato. Tutto il traffico verso il server API del cluster deve provenire dal VPC del cluster o da una rete connessa. I nodi comunicano con il server API tramite X- ENIs all'interno del tuo VPC. Tieni presente che gli strumenti di gestione del cluster devono avere accesso all'endpoint privato. Scopri di più su come connettersi a un endpoint privato del cluster Amazon EKS dall'esterno di Amazon VPC.

Tieni presente che l'endpoint del server API del cluster viene risolto dai server DNS pubblici in un indirizzo IP privato dal VPC. In passato, l'endpoint poteva essere risolto solo dall'interno del VPC.

Configurazioni VPC

Supporto IPv4 e indirizzamento di Amazon VPC. IPv6 Amazon EKS supporta IPv4 per impostazione predefinita. A un VPC deve essere associato un blocco IPv4 CIDR. Facoltativamente, puoi associare più blocchi CIDR (IPv4 Classless Inter-Domain Routing) e più blocchi CIDR al tuo VPC. IPv6 Quando si crea un VPC, è necessario specificare un blocco IPv4 CIDR per il VPC dagli intervalli di IPv4 indirizzi privati come specificato in RFC 1918. La dimensione del blocco consentita è compresa tra un `/16` prefisso (65.536 indirizzi IP) e un prefisso (16 indirizzi IP). `/28`

Quando si crea un nuovo VPC, è possibile collegare un singolo blocco IPv6 CIDR e fino a cinque se si modifica un VPC esistente. La lunghezza del prefisso della dimensione del blocco IPv6 CIDR può essere compresa tra /44 e /60 e per le IPv6 sottoreti può essere compresa tra /44/ e /64. Puoi richiedere un blocco IPv6 CIDR dal pool di IPv6 indirizzi gestito da Amazon. Per ulteriori informazioni, consulta la sezione VPC CIDR block della VPC User Guide.

I cluster Amazon EKS supportano entrambi IPv4 e IPv6. Per impostazione predefinita, i cluster EKS utilizzano l' IPv4 IP. La specificazione IPv6 al momento della creazione del cluster abiliterà l'uso IPv6 dei cluster. IPv6 i cluster richiedono dual-stack e sottoreti VPCs .

Amazon EKS consiglia di utilizzare almeno due sottoreti che si trovano in zone di disponibilità diverse durante la creazione del cluster. Le sottoreti che trasmetti durante la creazione del cluster sono note come sottoreti del cluster. Quando crei un cluster, Amazon EKS crea fino a 4 account incrociati (x-account o x-ENIs) ENIs nelle sottoreti specificate. Gli x- ENIs vengono sempre distribuiti e utilizzati per il traffico di amministrazione del cluster, ad esempio per la consegna dei log, l'esecuzione e il proxy. Consulta la guida per l'utente EKS per i dettagli completi sui requisiti relativi a VPC e sottorete.

I nodi di lavoro Kubernetes possono essere eseguiti nelle sottoreti del cluster, ma non è consigliato. Durante gli upgrade dei cluster, Amazon EKS fornisce componenti aggiuntivi ENIs nelle sottoreti del cluster. Quando il cluster si ridimensiona orizzontalmente, i nodi di lavoro e i pod possono consumare quanto disponibile IPs nella sottorete del cluster. Quindi, per assicurarti che ci sia abbastanza spazio disponibile, IPs potresti prendere in considerazione l'utilizzo di sottoreti di cluster dedicate con /28 netmask.

I nodi di lavoro Kubernetes possono essere eseguiti in una sottorete pubblica o privata. Il fatto che una sottorete sia pubblica o privata si riferisce al fatto che il traffico all'interno della sottorete venga instradato attraverso un gateway Internet. Le sottoreti pubbliche dispongono di una tabella di routing per accedere a Internet tramite il gateway Internet, a differenza delle sottoreti private.

Il traffico che ha origine da qualche altra parte e raggiunge i nodi si chiama ingresso. Il traffico che proviene dai nodi e esce dalla rete viene chiamato in uscita. I nodi con indirizzi IP pubblici o elastici (EIPs) all'interno di una sottorete configurata con un gateway Internet consentono l'ingresso dall'esterno del VPC. Le sottoreti private di solito hanno un routing verso un gateway NAT, che non consente al traffico in ingresso ai nodi delle sottoreti dall'esterno del VPC, pur consentendo al traffico proveniente dai nodi di uscire dal VPC (uscita).

Nel mondo, ogni indirizzo è instradabile su Internet. IPv6 Gli IPv6 indirizzi associati ai nodi e ai pod sono pubblici. Le sottoreti private sono supportate dall'implementazione di un gateway Internet di sola uscita (EIGW) in un VPC, che consente il traffico in uscita e blocca tutto il traffico in entrata. Le

migliori pratiche per l'implementazione delle IPv6 sottoreti sono disponibili nella guida per l'utente di VPC.

Puoi configurare VPC e sottoreti in tre modi diversi:

Utilizzando solo sottoreti pubbliche

Nelle stesse sottoreti pubbliche, vengono creati sia i nodi che le risorse in ingresso (come i sistemi di bilanciamento del carico). Etichetta la sottorete pubblica con `kubernetes.io/role/elb`per creare sistemi di bilanciamento del carico che si affacciano su Internet. In questa configurazione, l'endpoint del cluster può essere configurato come pubblico, privato o entrambi (pubblico e privato).

Utilizzo di sottoreti private e pubbliche

I nodi vengono creati su sottoreti private, mentre le risorse Ingress vengono istanziate in sottoreti pubbliche. È possibile abilitare l'accesso pubblico, privato o entrambi (pubblico e privato) all'endpoint del cluster. A seconda della configurazione dell'endpoint del cluster, il traffico del nodo entrerà tramite il gateway NAT o l'ENI.

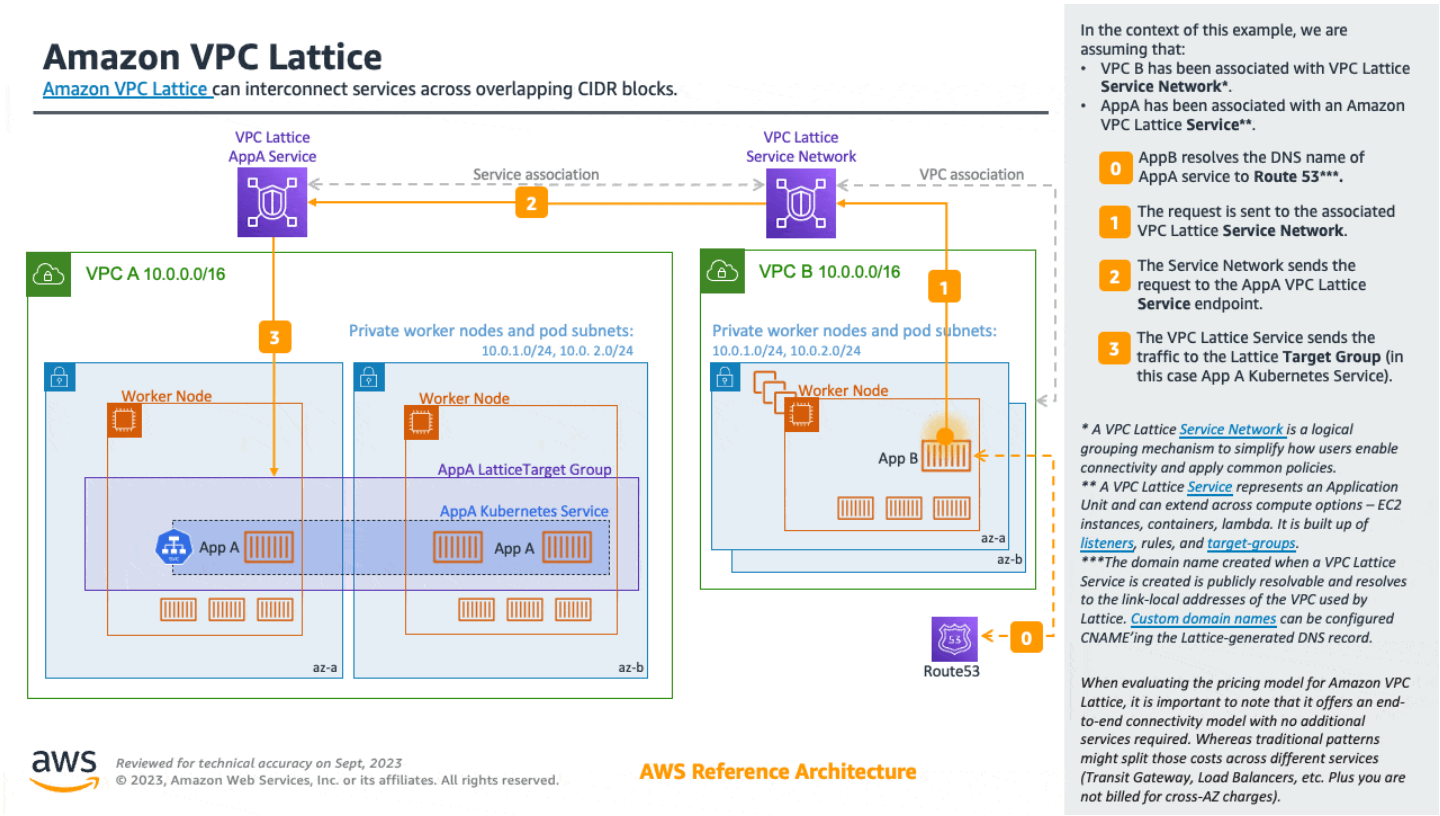
Utilizzo solo di sottoreti private

Sia i nodi che l'ingresso vengono creati in sottoreti private. Utilizzo del tag `kubernetes.io/role/internal-elb`subnet per costruire bilanciatori di carico interni. L'accesso all'endpoint del cluster richiederà una connessione VPN. È necessario attivare AWS PrivateLink for EC2 e tutti i repository Amazon ECR e S3. Deve essere abilitato solo l'endpoint privato del cluster. Suggeriamo di esaminare i requisiti del cluster privato EKS prima di effettuare il provisioning dei cluster privati.

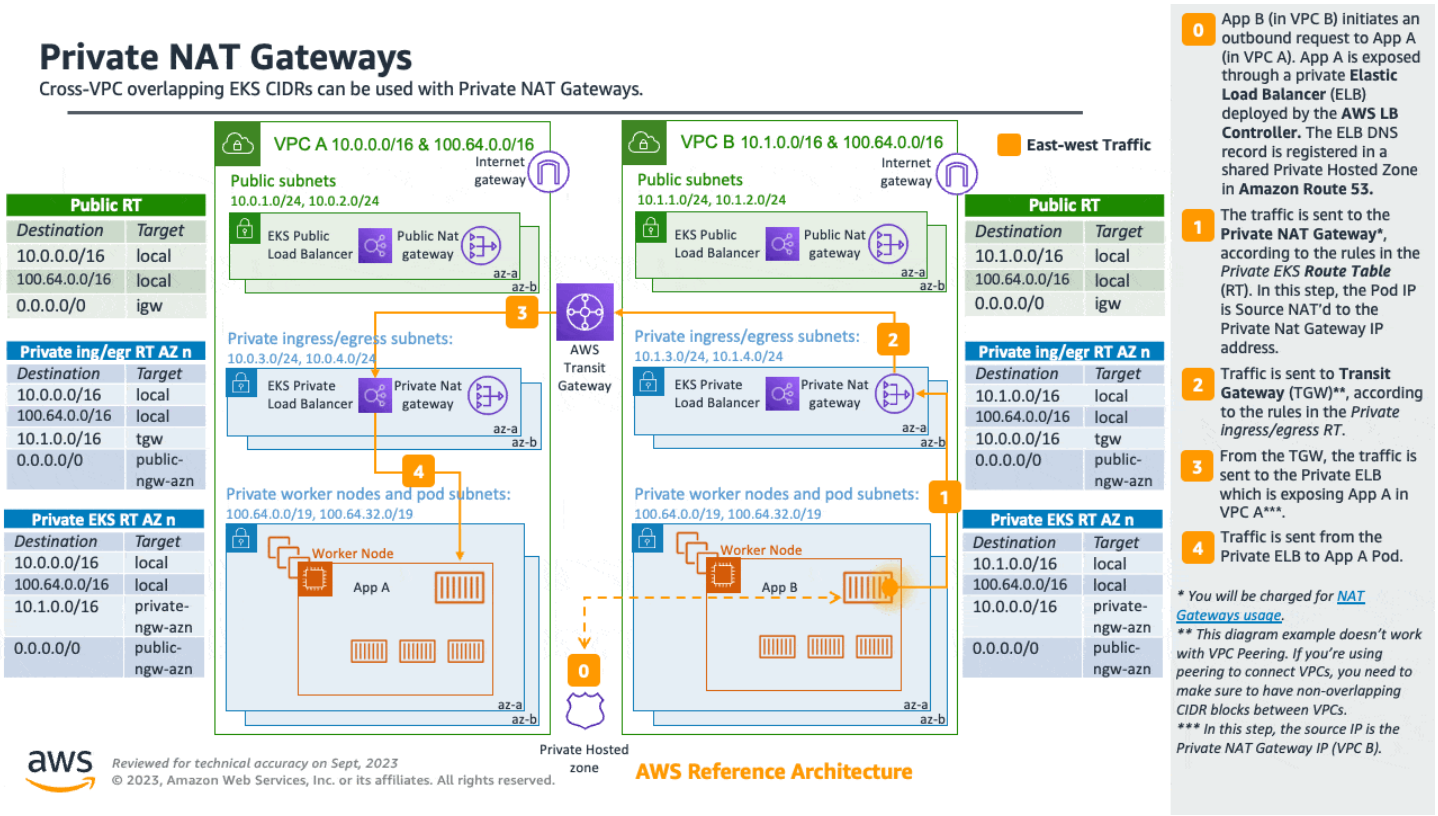
Comunicazione attraverso VPCs

Esistono molti scenari in cui è necessario distribuire cluster EKS multipli VPCs e separati su questi. VPCs

Puoi utilizzare Amazon VPC Lattice per connettere in modo coerente e sicuro i servizi tra più VPCs account (senza richiedere una connettività aggiuntiva fornita da servizi come il peering VPC, AWS o AWS Transit Gateway). PrivateLink Scopri di più qui.



Amazon VPC Lattice opera nello spazio degli indirizzi locali del collegamento in IPv4 e IPv6, fornendo connettività tra servizi che possono avere indirizzi sovrapposti. IPv4 Per l'efficienza operativa, consigliamo vivamente di distribuire cluster e nodi EKS su intervalli IP che non si sovrappongano. Nel caso in cui l'infrastruttura VPCs includa intervalli IP sovrapposti, è necessario progettare la rete di conseguenza. Suggeriamo Private NAT Gateway o VPC CNI in modalità di rete personalizzata insieme al gateway di transito per integrare i carichi di lavoro su EKS per risolvere le sfide CIDR sovrapposte preservando gli indirizzi IP instradabili. RFC1918

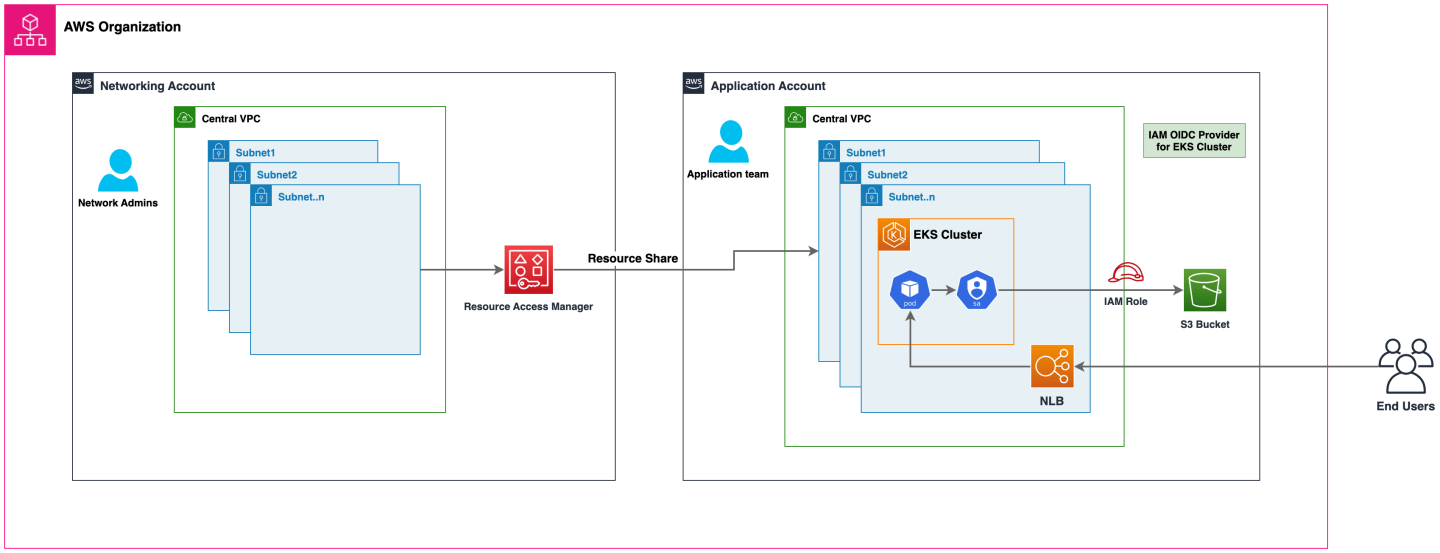


Prendi in considerazione l'utilizzo di AWS PrivateLink, noto anche come servizio endpoint, se sei il fornitore di servizi e desideri condividere il servizio e l'ingresso Kubernetes (ALB o NLB) con il VPC del cliente in account separati.

Condivisione del VPC su più account

Molte aziende hanno adottato Amazon condiviso VPCs come mezzo per semplificare l'amministrazione di rete, ridurre i costi e migliorare la sicurezza su più account AWS in un'organizzazione AWS. Utilizzano AWS Resource Access Manager (RAM) per condividere in modo sicuro le risorse AWS supportate con singoli account AWS, unità organizzative (OUs) o l'intera AWS Organization.

Puoi distribuire cluster Amazon EKS, gruppi di nodi gestiti e altre risorse AWS di supporto (come gruppi di sicurezza LoadBalancers, endpoint, ecc.) in sottoreti VPC condivise da un altro account AWS utilizzando la RAM AWS. La figura seguente mostra un esempio di architettura di alto livello. Ciò consente ai team di rete centrali di controllare i costrutti di rete come VPCs le sottoreti e così via, mentre consente ai team di applicazioni o piattaforme di distribuire i cluster Amazon EKS nei rispettivi account AWS. Una panoramica completa di questo scenario è disponibile in questo repository github.



Considerazioni sull'utilizzo di sottoreti condivise

- I cluster e i nodi di lavoro Amazon EKS possono essere creati all'interno di sottoreti condivise che fanno tutte parte dello stesso VPC. Amazon EKS non supporta la creazione di cluster su più VPCs cluster.
- Amazon EKS utilizza AWS VPC Security Groups (SGs) per controllare il traffico tra il piano di controllo Kubernetes e i nodi di lavoro del cluster. I gruppi di sicurezza vengono utilizzati anche per controllare il traffico tra i nodi di lavoro e altre risorse VPC e gli indirizzi IP esterni. È necessario creare questi gruppi di sicurezza nell' application/participant account. Assicurati che i gruppi di sicurezza che intendi utilizzare per i tuoi pod si trovino anche nell'account del partecipante. Puoi configurare le regole in entrata e in uscita all'interno dei tuoi gruppi di sicurezza per consentire il traffico necessario da e verso i gruppi di sicurezza situati nell'account Central VPC.
- Crea ruoli IAM e policy associate all'interno dell'account partecipante in cui risiede il tuo cluster Amazon EKS. Questi ruoli e policy IAM sono essenziali per concedere le autorizzazioni necessarie ai cluster Kubernetes gestiti da Amazon EKS, nonché ai nodi e ai pod in esecuzione su Fargate. Le autorizzazioni consentono ad Amazon EKS di effettuare chiamate verso altri servizi AWS per tuo conto.
- Puoi seguire i seguenti approcci per consentire l'accesso da più account a risorse AWS come bucket Amazon S3, tabelle Dynamodb, ecc. dai pod k8s:
  - Approccio basato sulle politiche basate sulle risorse: se il servizio AWS supporta le policy relative alle risorse, puoi aggiungere una policy basata sulle risorse appropriata per consentire l'accesso tra account ai ruoli IAM assegnati ai pod kubernetes. In questo scenario, il provider OIDC, i ruoli IAM e le politiche di autorizzazione sono presenti nell'account dell'applicazione. Per trovare i

servizi AWS che supportano le policy basate sulle risorse, consulta i servizi AWS compatibili con IAM e cerca i servizi con Sì nella colonna Resource Based.

- Approccio del provider OIDC: le risorse IAM come le policy OIDC Provider, IAM Roles, Permission e Trust verranno create in un altro account AWS partecipante in cui esistono le risorse. Questi ruoli verranno assegnati ai pod Kubernetes nell'account dell'applicazione, in modo che possano accedere alle risorse di più account. Consulta il blog Cross account IAM roles for Kubernetes service accounts per una panoramica completa di questo approccio.

- Puoi distribuire le risorse Amazon Elastic Loadbalancer (ELB) (ALB o NLB) per instradare il traffico verso i pod k8s nelle applicazioni o negli account di rete centrale. Consulta la procedura dettagliata Expose Amazon EKS Pods Through Cross-Account Load Balancer per istruzioni dettagliate sulla distribuzione delle risorse ELB nell'account di rete centrale. Questa opzione offre una maggiore flessibilità, in quanto garantisce all'account di rete centrale il pieno controllo sulla configurazione di sicurezza delle risorse Load Balancer.
- Quando si utilizza `custom networking feature` Amazon VPC CNI, è necessario utilizzare le mappature ID delle zone di disponibilità (AZ) elencate nell'account di rete centrale per crearle ciascuna. `ENIConfig` Ciò è dovuto alla mappatura casuale dei nomi fisici AZs con quelli AZ in ogni account AWS.

Gruppi di sicurezza

Un gruppo di sicurezza controlla il traffico consentito per raggiungere e lasciare le risorse a cui è associato. Amazon EKS utilizza gruppi di sicurezza per gestire la comunicazione tra il piano di controllo e i nodi. Quando crei un cluster, Amazon EKS crea un gruppo di sicurezza denominato `eks-cluster-sg-my-cluster-uniqueID`. EKS associa questi gruppi di sicurezza ai nodi ENIs e gestiti. Le regole predefinite consentono il flusso del traffico tra il cluster e i nodi e il traffico in uscita verso qualsiasi destinazione.

Quando si crea un cluster, è possibile specificare i propri gruppi di sicurezza. Consulta i consigli per i gruppi di sicurezza quando specifichi i propri gruppi di sicurezza.

Raccomandazioni

Prendi in considerazione l'implementazione Multi-AZ

Le regioni AWS forniscono più zone di disponibilità (AZ) fisicamente separate e isolate, collegate con reti a bassa latenza, alto throughput e altamente ridondante. Con le zone di disponibilità, puoi progettare e gestire applicazioni che eseguono automaticamente il failover tra le zone di disponibilità

senza interruzioni. Amazon EKS consiglia vivamente di distribuire i cluster EKS in più zone di disponibilità. Prendi in considerazione la possibilità di specificare le sottoreti in almeno due zone di disponibilità quando crei il cluster.

Kubelet in esecuzione sui nodi aggiunge automaticamente etichette all'oggetto nodo come. `topology.kubernetes.io/region=us-west-2` Consigliamo di utilizzare le etichette dei nodi insieme ai vincoli di diffusione della topologia Pod per controllare la distribuzione dei Pod tra le zone. Questi suggerimenti consentono allo scheduler Kubernetes di posizionare i Pod per una migliore disponibilità prevista, riducendo il rischio che un errore correlato influisca sull'intero carico di lavoro. Consulta Assegnazione di nodi ai pod per vedere esempi di vincoli relativi al selettore di nodi e allo spread AZ.

È possibile definire le sottoreti o le zone di disponibilità quando si creano nodi. I nodi vengono inseriti nelle sottoreti del cluster se non è configurata alcuna sottorete. Il supporto EKS per i gruppi di nodi gestiti distribuisce automaticamente i nodi su più zone di disponibilità in base alla capacità disponibile. Karpenter rispetterà il posizionamento dello spread AZ ridimensionando i nodi in modo da specificare AZs se i carichi di lavoro definiscono i limiti di diffusione della topologia.

Gli AWS Elastic Load Balancer sono gestiti dal controller AWS Load Balancer per un cluster Kubernetes. Fornisce un Application Load Balancer (ALB) per le risorse in ingresso di Kubernetes e un Network Load Balancer (NLB) per i servizi Kubernetes di tipo Loadbalancer. Il controller Elastic Load Balancer utilizza i tag per individuare le sottoreti. Il controller ELB richiede un minimo di due zone di disponibilità (AZs) per fornire correttamente le risorse in ingresso. Valuta la possibilità di impostare almeno due sottoreti AZs per sfruttare la sicurezza e l'affidabilità della ridondanza geografica.

Distribuisci i nodi in sottoreti private

Un VPC che include sottoreti private e pubbliche è il metodo ideale per implementare carichi di lavoro Kubernetes su EKS. Prendi in considerazione l'impostazione di un minimo di due sottoreti pubbliche e due sottoreti private in due zone di disponibilità distinte. La tabella di routing correlata di una sottorete pubblica contiene un percorso verso un gateway Internet. I pod sono in grado di interagire con Internet tramite un gateway NAT. Le sottoreti private sono supportate da gateway Internet in ambiente (EIGW) di sola uscita. IPv6

L'istanziazione di nodi in sottoreti private offre il massimo controllo sul traffico verso i nodi ed è efficace per la maggior parte delle applicazioni Kubernetes. Le risorse in ingresso (come i sistemi di bilanciamento del carico) vengono istanziate nelle sottoreti pubbliche e indirizzano il traffico verso i Pod che operano su sottoreti private.

Prendi in considerazione la modalità solo privata se richiedi sicurezza e isolamento della rete rigorosi. In questa configurazione, tre sottoreti private vengono distribuite in zone di disponibilità distinte all'interno del VPC della regione AWS. Le risorse distribuite nelle sottoreti non possono accedere a Internet, né Internet può accedere alle risorse nelle sottoreti. Affinché la tua applicazione Kubernetes possa accedere ad altri servizi AWS, devi configurare gli endpoint gateway delle PrivateLink interfacce and/or . Puoi configurare sistemi di bilanciamento del carico interni per reindirizzare il traffico verso i Pods utilizzando AWS Load Balancer Controller. Le sottoreti private devono essere contrassegnate con tag (`kubernetes.io/role/internal-elb: 1`) affinché il controller fornisca i bilanciatori del carico. Affinché i nodi possano registrarsi nel cluster, l'endpoint del cluster deve essere impostato sulla modalità privata. Consulta la guida ai cluster privati per conoscere i requisiti e le considerazioni completi.

Prendi in considerazione la modalità pubblica e privata per Cluster Endpoint

Amazon EKS offre modalità endpoint cluster public-and-private solo pubbliche e private. La modalità predefinita è solo pubblica, tuttavia consigliamo di configurare gli endpoint del cluster in modalità pubblica e privata. Questa opzione consente alle chiamate API Kubernetes all'interno del VPC del cluster (come la node-to-control-plane comunicazione) di utilizzare l'endpoint VPC privato e il traffico di rimanere all'interno del VPC del cluster. Il server API del cluster, invece, può essere raggiunto da Internet. Tuttavia, consigliamo vivamente di limitare i blocchi CIDR che possono utilizzare l'endpoint pubblico. Scopri come configurare l'accesso agli endpoint pubblici e privati, inclusa la limitazione dei blocchi CIDR.

Ti suggeriamo un endpoint solo privato quando hai bisogno di sicurezza e isolamento della rete. Ti consigliamo di utilizzare una delle opzioni elencate nella guida per l'utente di EKS per connetterti a un server API in modo privato.

Configura attentamente i gruppi di sicurezza

Amazon EKS supporta l'utilizzo di gruppi di sicurezza personalizzati. Qualsiasi gruppo di sicurezza personalizzato deve consentire la comunicazione tra i nodi e il piano di controllo Kubernetes. Controlla i requisiti delle porte e configura le regole manualmente quando la tua organizzazione non consente una comunicazione aperta.

EKS applica i gruppi di sicurezza personalizzati forniti durante la creazione del cluster alle interfacce gestite (X-ENIs). Tuttavia, non li associa immediatamente ai nodi. Durante la creazione di gruppi di nodi, si consiglia vivamente di associare manualmente i gruppi di sicurezza personalizzati. Valuta la possibilità di abilitare securityGroupSelectori Termini per consentire ai modelli di nodi Karpenter di individuare i gruppi di sicurezza personalizzati durante la scalabilità automatica dei nodi.

Consigliamo vivamente di creare un gruppo di sicurezza per consentire tutto il traffico di comunicazione tra nodi. Durante il processo di bootstrap, i nodi richiedono la connettività Internet in uscita per accedere all'endpoint del cluster. Valuta i requisiti di accesso verso l'esterno, come la connessione locale e l'accesso al registro dei contenitori, e imposta le regole in modo appropriato. Prima di apportare modifiche alla produzione, ti consigliamo vivamente di controllare attentamente le connessioni nel tuo ambiente di sviluppo.

Implementa i gateway NAT in ogni zona di disponibilità

Se distribuisci nodi in sottoreti private (IPv4 and IPv6), prendi in considerazione la creazione di un gateway NAT in ciascuna zona di disponibilità (AZ) per garantire un'architettura indipendente dalla zona e ridurre le spese tra le AZ. Ogni gateway NAT in una zona AZ è implementato con ridondanza.

CNI di Amazon VPC

> **Tip**
>
> Esplora le best practice tramite i workshop Amazon EKS.

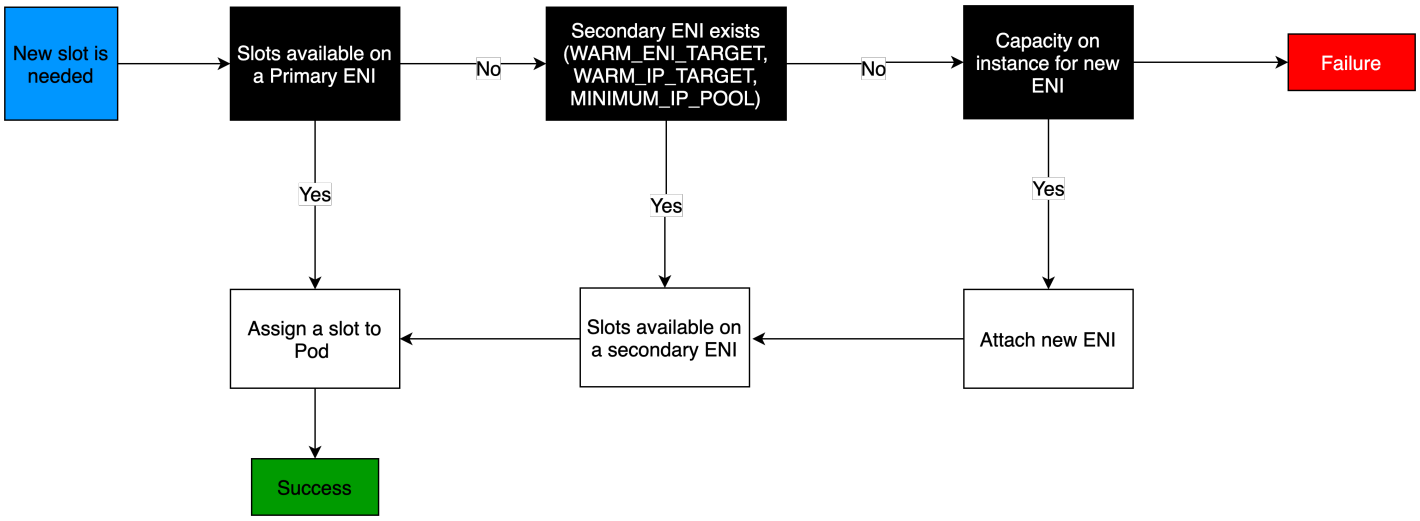
Amazon EKS implementa la rete di cluster tramite il plug-in Amazon VPC Container Network Interface, noto anche come VPC CNI. Il plug-in CNI consente ai Kubernetes Pods di avere lo stesso indirizzo IP della rete VPC. Più specificamente, tutti i contenitori all'interno del Pod condividono uno spazio dei nomi di rete e possono comunicare tra loro utilizzando le porte locali.

Amazon VPC CNI ha due componenti:

- CNI Binary, che configurerà la rete Pod per consentire la comunicazione. Pod-to-Pod Il binario CNI viene eseguito su un file system root del nodo e viene richiamato dal kubelet quando viene aggiunto un nuovo Pod o viene rimosso un Pod esistente dal nodo.
- ipamd, un demone IPAM (IP Address Management) locale al nodo che funziona da molto tempo ed è responsabile di:
  - gestione su un nodo e ENIs
  - mantenimento di un pool caldo di indirizzi IP o prefissi disponibili

Quando viene creata un'istanza, EC2 crea e collega un ENI primario associato a una sottorete principale. La sottorete principale può essere pubblica o privata. I Pod eseguiti in modalità HostNetwork utilizzano l'indirizzo IP principale assegnato al nodo ENI primario e condividono lo stesso spazio dei nomi di rete dell'host.

Il plugin CNI gestisce le interfacce di rete elastiche (ENI) sul nodo. Quando viene effettuato il provisioning di un nodo, il plug-in CNI alloca automaticamente un pool di slot (IPs o prefissi) dalla sottorete del nodo all'ENI primario. Questo pool è noto come pool caldo e la sua dimensione è determinata dal tipo di istanza del nodo. A seconda delle impostazioni CNI, uno slot può essere un indirizzo IP o un prefisso. Quando è stato assegnato uno slot su un ENI, il CNI può aggiungere ai nodi un pool di slot aggiuntivi ENIs con un pool di slot caldi. Questi aggiuntivi ENIs sono chiamati secondari ENIs. Ogni ENI può supportare solo un certo numero di slot, in base al tipo di istanza. Il CNI assegna un numero maggiore ENIs di slot alle istanze in base al numero di slot necessari, che di solito corrisponde al numero di Pod. Questo processo continua fino a quando il nodo non è più in grado di supportare ENI aggiuntivi. Il CNI preassegna anche «warm» ENIs e slot per un avvio più rapido dei Pod. Nota che ogni tipo di istanza ha un numero massimo di istanze ENIs che possono essere allegate. Questo è un vincolo sulla densità dei Pod (numero di Pod per nodo), oltre alle risorse di calcolo.



Il numero massimo di interfacce di rete e il numero massimo di slot utilizzabili variano a seconda del tipo di istanza EC2. Poiché ogni Pod utilizza un indirizzo IP su uno slot, il numero di Pod che puoi eseguire su una particolare istanza EC2 dipende da quanti ENIs possono essere collegati ad essa e da quanti slot supporta ogni ENI. Ti consigliamo di impostare il numero massimo di Pods per guida utente EKS per evitare l'esaurimento della CPU e delle risorse di memoria dell'istanza. I pod che utilizzano `hostNetwork` sono esclusi da questo calcolo. Potresti prendere in considerazione l'utilizzo

di uno script chiamato max-pod-calculator.sh per calcolare il numero massimo di Pod consigliati da EKS per un determinato tipo di istanza.

Panoramica di

La modalità IP secondaria è la modalità predefinita per VPC CNI. Questa guida fornisce una panoramica generica del comportamento VPC CNI quando è abilitata la modalità IP secondaria. La funzionalità di ipamd (allocazione degli indirizzi IP) può variare a seconda delle impostazioni di configurazione per VPC CNI, ad esempio, e. the section called “Modalità prefisso per Linux” the section called “Gruppi di sicurezza per pod” the section called “Rete personalizzata”

Amazon VPC CNI viene distribuito come un daemonset Kubernetes denominato aws-node sui nodi di lavoro. Quando viene effettuato il provisioning di un nodo di lavoro, a esso è associato un ENI predefinito, denominato ENI primario. Il CNI alloca un pool caldo ENIs e secondario di indirizzi IP dalla sottorete collegata all'ENI primario del nodo. Per impostazione predefinita, ipamd tenta di allocare un ENI aggiuntivo al nodo. L'IPAMD assegna un ENI aggiuntivo quando viene pianificato un singolo Pod e gli viene assegnato un indirizzo IP secondario dall'ENI primario. Questa ENI «calda» consente una rete Pod più veloce. Man mano che il pool di indirizzi IP secondari si esaurisce, il CNI aggiunge un altro ENI a cui assegnarne altri.

Il numero di ENIs indirizzi IP in un pool sono configurati tramite variabili di ambiente denominate WARM_ENI_TARGET, WARM_IP_TARGET, MINIMUM_IP_TARGET. Il Daemonset verificherà periodicamente che sia presente un numero sufficiente di allegati. `aws-node` ENIs Un numero sufficiente di ENIs sono allegati quando tutte le `WARM_ENI_TARGET` `MINIMUM_IP_TARGET` condizioni o `WARM_IP_TARGET` e sono soddisfatte. Se gli ENI collegati sono insufficienti, il CNI effettuerà una chiamata API a EC2 per allegarne altri fino al raggiungimento del `MAX_ENI` limite.

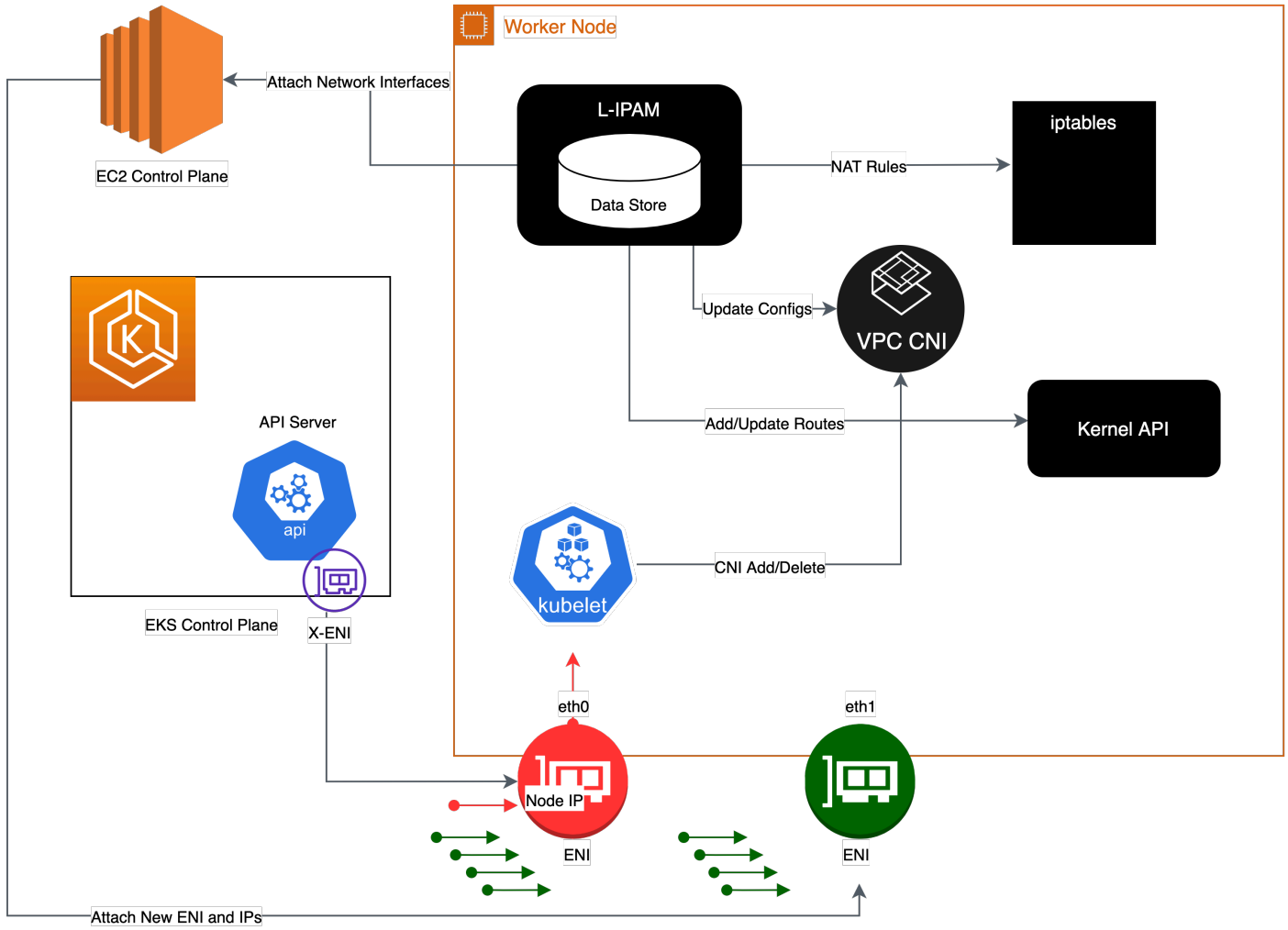
- `WARM_ENI_TARGET`- Numero intero, i valori maggiori di 0 indicano il requisito abilitato
  - Il numero di Warm ENIs da mantenere. Un ENI è «caldo» quando è collegato come ENI secondario a un nodo, ma non è utilizzato da nessun Pod. Più specificamente, nessun indirizzo IP dell'ENI è stato associato a un Pod.
  - Esempio: si consideri un'istanza con 2 ENIs, ciascuna ENI che supporta 5 indirizzi IP. WARM_ENI_TARGET è impostato su 1. Se all'istanza sono associati esattamente 5 indirizzi IP, il CNI ne mantiene 2 ENIs collegati all'istanza. Il primo ENI è in uso e vengono utilizzati tutti i 5 possibili indirizzi IP di questo ENI. Il secondo ENI è «caldo» con tutti e 5 gli indirizzi IP in pool. Se viene avviato un altro Pod sull'istanza, sarà necessario un sesto indirizzo IP. Il CNI assegnerà a questo sesto Pod un indirizzo IP dal secondo ENI e dal 5 IPs dal pool. Il secondo ENI è ora in

uso e non è più in uno stato «caldo». Il CNI assegnerà un terzo ENI al mantenimento di almeno 1 ENI caldo.

> **Note**
>
> I file warm ENIs continuano a consumare gli indirizzi IP del CIDR del tuo VPC. Gli indirizzi IP sono «inutilizzati» o «attivi» finché non vengono associati a un carico di lavoro, ad esempio un Pod.

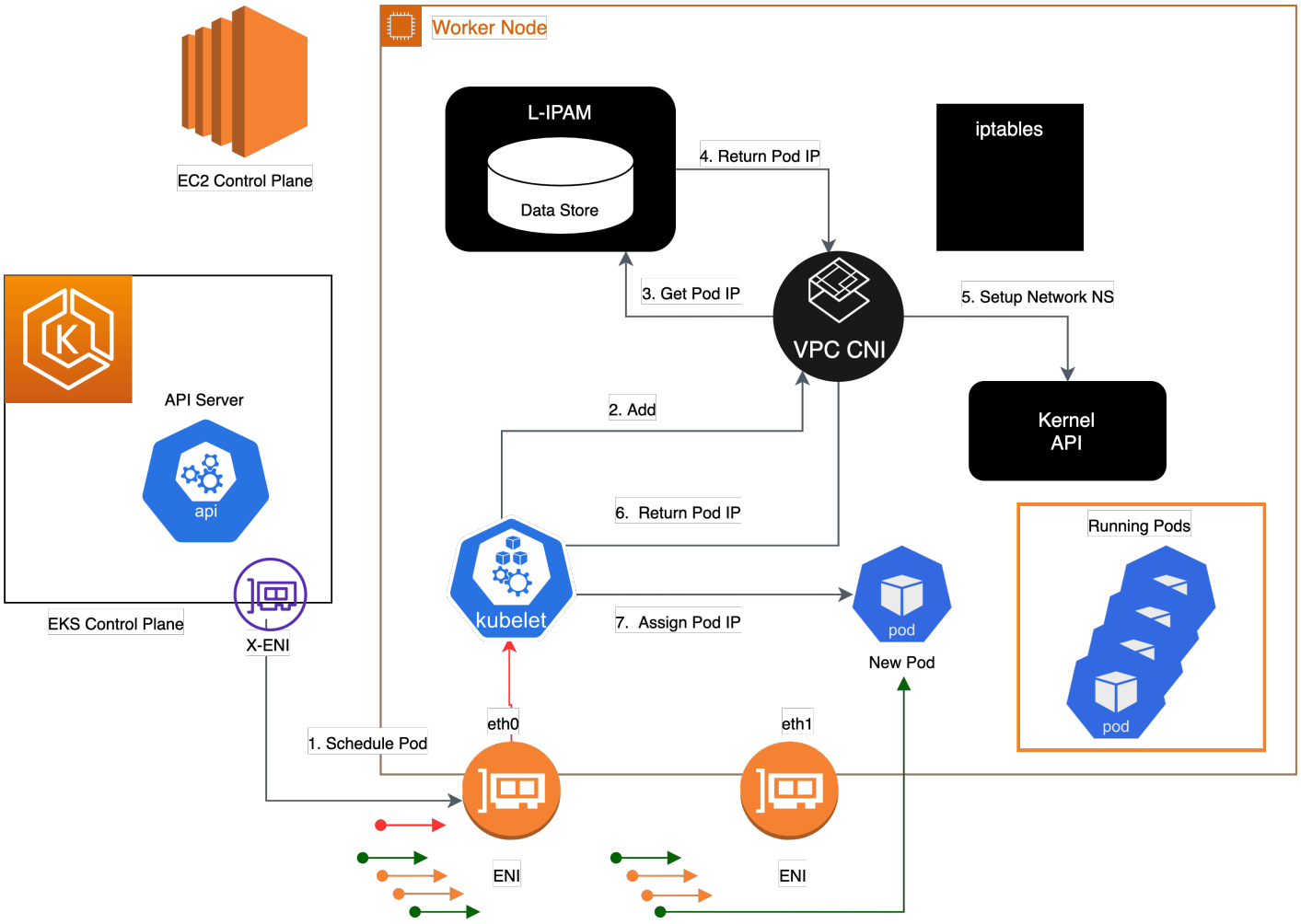
- `WARM_IP_TARGET`, Numero intero, I valori maggiori di 0 indicano il requisito abilitato
  - Il numero di indirizzi IP Warm da mantenere. Un Warm IP è disponibile su un ENI collegato attivamente, ma non è stato assegnato a un Pod. In altre parole, il numero di Warm IPs disponibili è il numero di Warm IPs che può essere assegnato a un Pod senza richiedere un ENI aggiuntivo.
- Esempio: si consideri un'istanza con 1 ENI, ogni ENI che supporta 20 indirizzi IP. WARM_IP_TARGET è impostato su 5. WARM_ENI_TARGET è impostato su 0. Verrà collegato solo 1 ENI fino a quando non sarà necessario un 16° indirizzo IP. Quindi, il CNI collegherà un secondo ENI, consumando 20 possibili indirizzi dalla sottorete CIDR.
- `MINIMUM_IP_TARGET`, Numero intero, I valori maggiori di 0 indicano il requisito abilitato
  - Il numero minimo di indirizzi IP da assegnare in qualsiasi momento. Viene comunemente utilizzato per anticipare l'assegnazione di più ENIs istanze all'avvio dell'istanza.
  - Esempio: si consideri un'istanza appena lanciata. Ha 1 ENI e ogni ENI supporta 10 indirizzi IP. MINIMUM_IP_TARGET è impostato su 100. L'ENI ne allega immediatamente altri 9 ENIs per un totale di 100 indirizzi. Ciò avviene indipendentemente dai valori WARM_IP_TARGET o WARM_ENI_TARGET.

Questo progetto include un documento Excel di Subnet Calculator. Questo documento di calcolo simula il consumo di indirizzi IP di un carico di lavoro specifico in diverse opzioni di configurazione ENI, come e. `WARM_IP_TARGET` `WARM_ENI_TARGET`

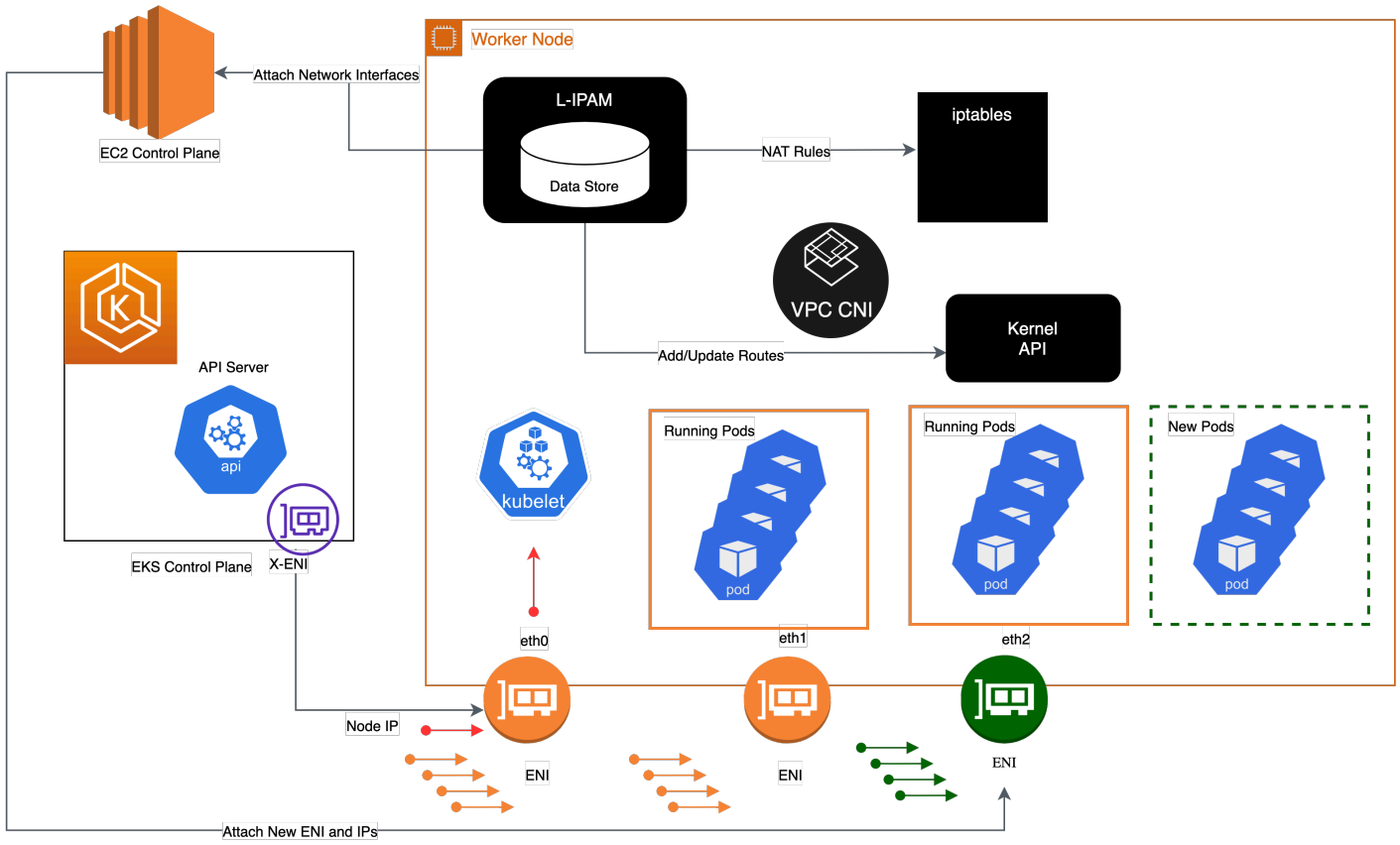


Quando Kubelet riceve una richiesta di aggiunta Pod, il binario CNI richiede a ipamd un indirizzo IP disponibile, che ipamd fornisce quindi al Pod. Il binario CNI collega l'host e la rete Pod.

Per impostazione predefinita, i pod distribuiti su un nodo vengono assegnati agli stessi gruppi di sicurezza dell'ENI principale. In alternativa, i Pod possono essere configurati con diversi gruppi di sicurezza.



Man mano che il pool di indirizzi IP si riduce numericamente, il plug-in allega automaticamente un'altra interfaccia di rete elastica all'istanza e alloca un altro set di indirizzi IP secondari a tale interfaccia. Questo processo continua finché il nodo non è più in grado di supportare altre interfacce di rete elastiche.



Quando un Pod viene eliminato, VPC CNI inserisce l'indirizzo IP del Pod in una cache di raffreddamento di 30 secondi. Le cache IPs in a cooldown non vengono assegnate ai nuovi Pod. Al termine del periodo di raffreddamento, VPC CNI riporta il Pod IP nella piscina calda. Il periodo di riflessione impedisce che gli indirizzi IP dei Pod vengano riciclati prematuramente e consente a kube-proxy su tutti i nodi del cluster di completare l'aggiornamento delle regole iptables. Quando il numero IPs o ENIs supera il numero di impostazioni della piscina calda, il plugin ipamd ritorna IPs ENIs al VPC.

Come descritto sopra in modalità IP secondario, ogni Pod riceve un indirizzo IP privato secondario da uno degli ENI collegati a un'istanza. Poiché ogni Pod utilizza un indirizzo IP, il numero di Pod che è possibile eseguire su una particolare istanza EC2 dipende da quanti ne ENIs possono essere collegati e da quanti indirizzi IP supporta. Il VPC CNI controlla il file dei limiti per scoprire quanti ENIs indirizzi IP sono consentiti per ogni tipo di istanza.

È possibile utilizzare la seguente formula per determinare il numero massimo di Pod che è possibile distribuire su un nodo.

```
(Number of network interfaces for the instance type * (the number of IP addresses per
 network interface - 1)) + 2
```

Il +2 indica i Pod che richiedono una rete host, come kube-proxy e VPC CNI. Amazon EKS richiede che kube-proxy e VPC CNI siano operativi su ogni nodo e questi requisiti vengono presi in considerazione nel valore max-pods. Se desideri eseguire pod di rete host aggiuntivi, valuta la possibilità di aggiornare il valore max-pods. È possibile specificare `--kubelet-extra-args "–max-pods=110"` come dati utente nel modello di avvio.

Ad esempio, su un cluster con 3 nodi c5.large (3 ENIs e massimo 10 IPs per ENI), quando il cluster si avvia e ha 2 pod CoredNS, il CNI consumerà 50 indirizzi IP e ne manterrà 43 in un pool caldo. IPs La piscina calda consente un avvio più rapido dei Pod quando l'applicazione viene distribuita.

Nodo 1 (con pod CoredNS): ENIs 2, 20 assegnati IPs

Nodo 2 (con pod CoredNS): ENIs 2, 20 assegnati IPs

Nodo 3 (nessun Pod): 1 ENI. 10 IPs assegnati.

Per il nodo 1 e il nodo 2 (configurazione identica):

- 2 ENIs × 10 IPs per ENI = 20 in IPs totale
- Sottrai 2 principali IPs (1 per ENI) = 18 IPs
- Sottrai 1 IP per il pod CoredNS = 17 disponibili IPs
- Quindi ognuno di questi nodi ne ha 17 IPs in una piscina calda

Per il nodo 3:

- 1 ENI × 10 IPs = 10 in IPs totale
- Sottrai 1 IP primario = 9 IPs disponibile nella piscina calda

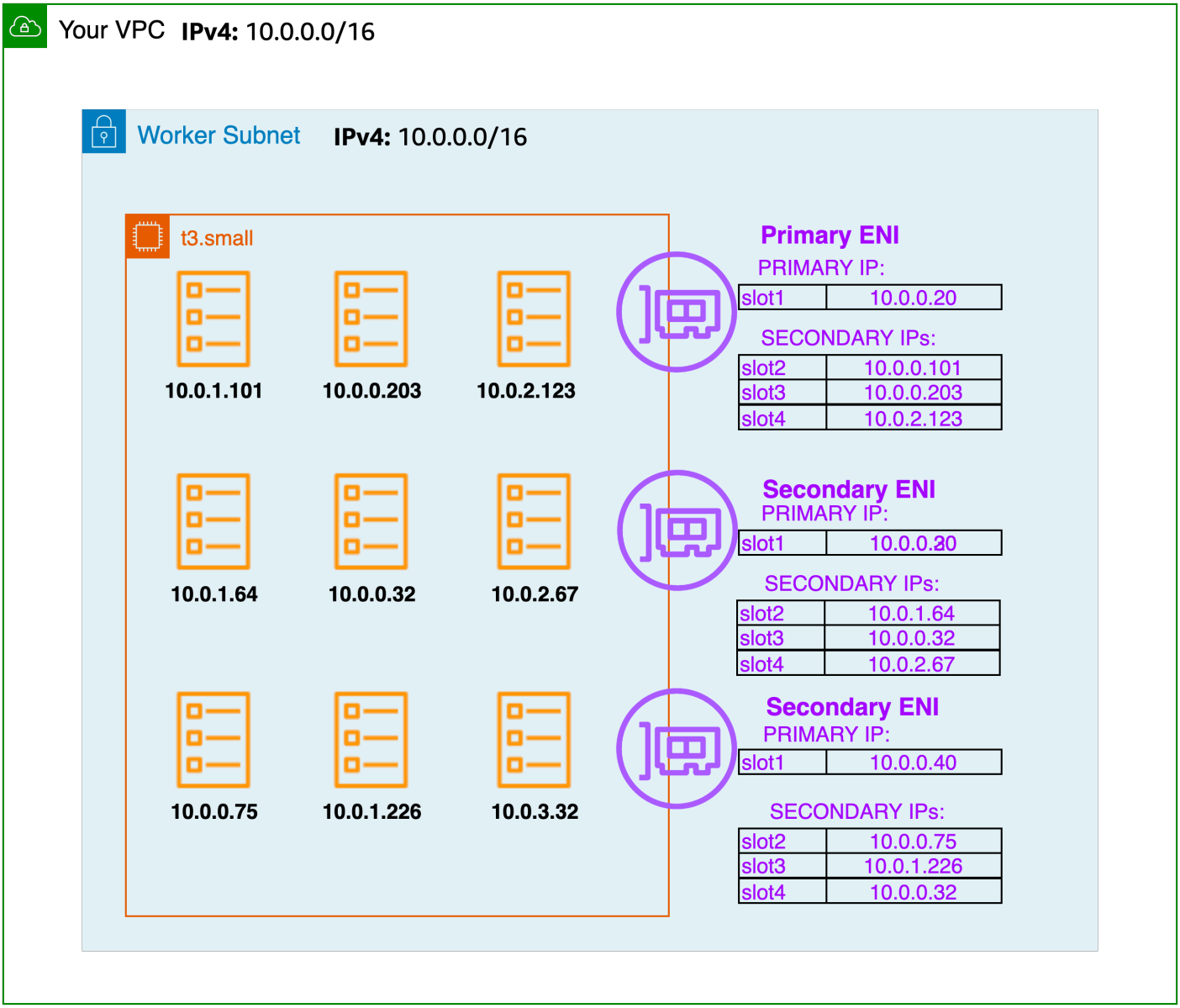
Calcolo totale del pool caldo: - 17 (Nodo 1) + 17 (Nodo 2) + 9 (Nodo 3) = 43 IPs

Tieni presente che i pod dell'infrastruttura, spesso eseguiti come set di daemon, contribuiscono ciascuno al numero massimo di pod. Questi possono includere:

- CoreDNS
- Amazon Elastic LoadBalancer

- Pod operativi per metrics-server

Ti suggeriamo di pianificare la tua infrastruttura combinando le capacità di questi Pod. Per un elenco del numero massimo di Pod supportati da ogni tipo di istanza, consulta eni-max-pods .txt su. GitHub



Raccomandazioni

Implementa il cluster EKS con la modalità automatica

Quando utilizzi EKS Auto Mode per creare un cluster, AWS gestisce la configurazione VPC Container Network Interface (CNI) per il cluster. Con la modalità automatica di Amazon EKS, non devi installare

o aggiornare componenti aggiuntivi di rete. Tuttavia, assicurati che i tuoi carichi di lavoro siano compatibili con la configurazione VPC CNI gestita.

Implementa il componente aggiuntivo gestito VPC CNI

Quando effettui il provisioning di un cluster, Amazon EKS installa automaticamente VPC CNI. Amazon EKS supporta tuttavia componenti aggiuntivi gestiti che consentono al cluster di interagire con le risorse AWS sottostanti come elaborazione, storage e rete. Ti consigliamo vivamente di implementare cluster con componenti aggiuntivi gestiti, tra cui VPC CNI.

Il componente aggiuntivo gestito di Amazon EKS offre l'installazione e la gestione di VPC CNI per i cluster Amazon EKS. I componenti aggiuntivi di Amazon EKS includono le patch di sicurezza e le correzioni di bug più recenti e sono convalidati da AWS per funzionare con Amazon EKS. Il componente aggiuntivo VPC CNI consente di garantire continuamente la sicurezza e la stabilità dei cluster Amazon EKS e di ridurre lo sforzo richiesto per installare, configurare e aggiornare i componenti aggiuntivi. Inoltre, è possibile aggiungere, aggiornare o eliminare un componente aggiuntivo gestito tramite l'API Amazon EKS, la console di gestione AWS, l'interfaccia a riga di comando AWS e eksctl.

Puoi trovare i campi gestiti di VPC CNI usando `--show-managed-fields` flag con il comando. `kubectl get`

```
kubectl get daemonset aws-node --show-managed-fields -n kube-system -o yaml
```

I componenti aggiuntivi gestiti impediscono la modifica della configurazione sovrascrivendo automaticamente le configurazioni ogni 15 minuti. Ciò significa che qualsiasi modifica ai componenti aggiuntivi gestiti, apportata tramite l'API Kubernetes dopo la creazione del componente aggiuntivo, verrà sovrascritta dal processo automatizzato di prevenzione delle derive e impostata sui valori predefiniti anche durante il processo di aggiornamento dei componenti aggiuntivi.

I campi gestiti da EKS sono elencati in ManagedFields con manager come EKS. I campi gestiti da EKS includono service account, image, image url, liveness probe, readiness probe, labels, volumes e volume mount.

> **Note**
>
> I campi utilizzati più di frequente come WARM_ENI_TARGET, WARM_IP_TARGET e MINIMUM_IP_TARGET non sono gestiti e non verranno riconciliati. Le modifiche a questi campi verranno mantenute dopo l'aggiornamento del componente aggiuntivo.

Ti consigliamo di testare il comportamento dei componenti aggiuntivi nei cluster non di produzione per una configurazione specifica prima di aggiornare i cluster di produzione. Inoltre, segui i passaggi indicati nella guida per l'utente EKS per le configurazioni aggiuntive.

Esegui la migrazione a Managed Add-On

Gestirai la compatibilità delle versioni e aggiornerai le patch di sicurezza del VPC CNI autogestito. Per aggiornare un componente aggiuntivo autogestito, è necessario utilizzare Kubernetes e le istruzioni riportate nella guida APIs per l'utente EKS. Consigliamo di passare a un componente aggiuntivo gestito per i cluster EKS esistenti e consigliamo vivamente di creare un backup delle impostazioni CNI correnti prima della migrazione. Per configurare componenti aggiuntivi gestiti, puoi utilizzare l'API Amazon EKS, la Console di gestione AWS o l'interfaccia a riga di comando AWS.

```
kubectl apply view-last-applied daemonset aws-node -n kube-system  aws-k8s-cni-old.yaml
```

Amazon EKS sostituirà le impostazioni di configurazione CNI se il campo è elencato come gestito con impostazioni predefinite. Mettiamo in guardia contro la modifica dei campi gestiti. Il componente aggiuntivo non riconcilia campi di configurazione come le variabili di ambiente caldo e le modalità CNI. I pod e le applicazioni continueranno a funzionare durante la migrazione a un CNI gestito.

Backup delle impostazioni CNI prima dell'aggiornamento

VPC CNI viene eseguito sul piano dati del cliente (nodi), pertanto Amazon EKS non aggiorna automaticamente il componente aggiuntivo (gestito e autogestito) quando vengono rilasciate nuove versioni o dopo l'aggiornamento del cluster a una nuova versione secondaria di Kubernetes. Per aggiornare il componente aggiuntivo per un cluster esistente, devi attivare un aggiornamento tramite l'API update-addon o facendo clic sul link Aggiorna ora nella console EKS per i componenti aggiuntivi. Se hai distribuito un componente aggiuntivo autogestito, segui i passaggi indicati nella sezione Aggiornamento del componente aggiuntivo VPC CNI autogestito.

Ti consigliamo vivamente di aggiornare una versione secondaria alla volta. Ad esempio, se la versione secondaria corrente è `1.9` e si desidera aggiornarla a `1.11`, è necessario prima eseguire l'aggiornamento alla versione più recente della patch di `1.10`, quindi, eseguire l'aggiornamento alla patch più recente di `1.11`.

Esegui un'ispezione di aws-node Daemonset prima di aggiornare Amazon VPC CNI. Effettua un backup delle impostazioni esistenti. Se utilizzi un componente aggiuntivo gestito, conferma di non aver aggiornato alcuna impostazione che Amazon EKS potrebbe sostituire. Consigliamo un hook

post-aggiornamento nel flusso di lavoro di automazione o una fase di applicazione manuale dopo un aggiornamento del componente aggiuntivo.

```
kubectl apply view-last-applied daemonset aws-node -n kube-system  aws-k8s-cni-old.yaml
```

Per un componente aggiuntivo autogestito, confronta il backup con `releases` on GitHub per vedere le versioni disponibili e acquisire familiarità con le modifiche della versione a cui desideri eseguire l'aggiornamento. Ti consigliamo di utilizzare Helm per gestire componenti aggiuntivi autogestiti e sfruttare i file di valori per applicare le impostazioni. Qualsiasi operazione di aggiornamento che comporti l'eliminazione di Daemonset comporterà tempi di inattività delle applicazioni e deve essere evitata.

Comprendi il contesto di sicurezza

Ti consigliamo vivamente di comprendere i contesti di sicurezza configurati per gestire VPC CNI in modo efficiente. Amazon VPC CNI ha due componenti: CNI binary e ipamd (aws-node) Daemonset. Il CNI funziona come file binario su un nodo e ha accesso al file system root del nodo, inoltre dispone di un accesso privilegiato in quanto gestisce iptables a livello di nodo. Il binario CNI viene richiamato dal kubelet quando i Pods vengono aggiunti o rimossi.

aws-node Daemonset è un processo di lunga durata responsabile della gestione degli indirizzi IP a livello di nodo. L'aws-node funziona in `hostNetwork` modalità e consente l'accesso al dispositivo di loopback e all'attività di rete di altri pod sullo stesso nodo. L'init-container aws-node funziona in modalità privilegiata e monta il socket CRI che consente a Daemonset di monitorare l'utilizzo dell'IP da parte dei Pod in esecuzione sul nodo. Amazon EKS sta lavorando per rimuovere il requisito privilegiato del contenitore init aws-node. Inoltre, aws-node deve aggiornare le voci NAT e caricare i moduli iptables e quindi funziona con i privilegi NET_ADMIN.

Amazon EKS consiglia di implementare le politiche di sicurezza definite dal manifesto aws-node per la gestione IP dei pod e delle impostazioni di rete. Valuta la possibilità di effettuare l'aggiornamento alla versione più recente di VPC CNI. Inoltre, prendi in considerazione la possibilità di aprire un GitHub problema se hai un requisito di sicurezza specifico.

Utilizza un ruolo IAM separato per CNI

L'AWS VPC CNI richiede le autorizzazioni AWS Identity and Access Management (IAM).
La policy CNI deve essere configurata prima di poter utilizzare il ruolo IAM. Puoi usare `AmazonEKS_CNI_Policy`, che è una policy gestita da AWS per IPv4 i cluster. La policy gestita da

AmazonEks CNI prevede solo le autorizzazioni per i cluster. IPv4 È necessario creare una policy IAM separata per i IPv6 cluster con le autorizzazioni elencate qui.

Per impostazione predefinita, VPC CNI eredita il ruolo IAM del nodo Amazon EKS (gruppi di nodi gestiti e autogestiti).

Si consiglia vivamente di configurare un ruolo IAM separato con le politiche pertinenti per Amazon VPC CNI. In caso contrario, i pod di Amazon VPC CNI ottengono l'autorizzazione assegnata al ruolo IAM del nodo e hanno accesso al profilo dell'istanza assegnato al nodo.

Il plugin VPC CNI crea e configura un account di servizio chiamato aws-node. Per impostazione predefinita, l'account del servizio si collega al ruolo IAM del nodo Amazon EKS con la policy CNI di Amazon EKS allegata. Per utilizzare il ruolo IAM separato, ti consigliamo di creare un nuovo account di servizio con la policy CNI di Amazon EKS allegata. Per utilizzare un nuovo account di servizio, devi ridistribuire i pod CNI. Valuta la possibilità di specificare un componente aggiuntivo gestito da CNI `--service-account-role-arn` per VPC quando crei nuovi cluster. Assicurati di rimuovere la policy CNI di Amazon EKS per entrambi IPv4 e IPv6 dal ruolo del nodo Amazon EKS.

Si consiglia di bloccare i metadati delle istanze di accesso per ridurre al minimo il raggio d'azione della violazione della sicurezza.

Gestisci i guasti di Probe Liveness/Readiness

Consigliamo di aumentare i valori di timeout della sonda Liveness e Readiness (predefinitit`imeoutSeconds: 10`) per i cluster EKS 1.20 e versioni successive per evitare che i guasti della sonda causino il blocco del Pod dell'applicazione in uno stato ContainerCreating. Questo problema è stato riscontrato nei cluster con uso intensivo di dati e di elaborazione in batch. L'elevato utilizzo della CPU causa problemi di integrità della sonda aws-node, con conseguenti richieste di CPU Pod non soddisfatte. Oltre a modificare il timeout della sonda, assicurati che le richieste di risorse della CPU (predefinite) per aws-node siano configurate correttamente. `CPU: 25m` Non suggeriamo di aggiornare le impostazioni a meno che il nodo non presenti problemi.

Ti consigliamo vivamente di eseguire sudo `bash /opt/cni/bin/aws-cni-support.sh` su un nodo mentre utilizzi il supporto di Amazon EKS. Lo script aiuterà a valutare i log di Kubelet e l'utilizzo della memoria sul nodo. Prendi in considerazione l'installazione di SSM Agent sui nodi di lavoro Amazon EKS per eseguire lo script.

Configurazione della policy di IPTables inoltro su istanze AMI non ottimizzate per EKS

Se utilizzi AMI personalizzate, assicurati di impostare la policy di inoltro di iptables su ACCEPT in kubelet.service. Molti sistemi impostano la politica di inoltro di iptables su DROP. Puoi creare AMI personalizzate utilizzando HashiCorp Packer e una specifica di build con risorse e script di configurazione dal repository AMI Amazon EKS su AWS. GitHub Puoi aggiornare kubelet.service e seguire le istruzioni specificate qui per creare un'AMI personalizzata.

Aggiorna regolarmente la versione CNI

Il VPC CNI è retrocompatibile. L'ultima versione funziona con tutte le versioni Kubernetes supportate da Amazon EKS. Inoltre, il VPC CNI è offerto come componente aggiuntivo EKS (vedi «Deploy VPC CNI Managed Add-On» sopra). Sebbene i componenti aggiuntivi EKS orchestrino gli aggiornamenti dei componenti aggiuntivi, non aggiorneranno automaticamente i componenti aggiuntivi come il CNI perché vengono eseguiti sul piano dati. Sei responsabile dell'aggiornamento del componente aggiuntivo VPC CNI dopo gli aggiornamenti dei nodi di lavoro gestiti e autogestiti.

Ottimizzazione dell'utilizzo degli indirizzi IP

> **Tip**
>
> Esplora le best practice tramite i workshop Amazon EKS.

Gli ambienti containerizzati stanno crescendo in scala a un ritmo rapido, grazie alla modernizzazione delle applicazioni. Ciò significa che vengono implementati sempre più nodi e pod di lavoro.

Il plug-in Amazon VPC CNI assegna a ciascun pod un indirizzo IP dai CIDR del VPC. Questo approccio offre la piena visibilità degli indirizzi Pod con strumenti come VPC Flow Logs e altre soluzioni di monitoraggio. A seconda del tipo di carico di lavoro, ciò può causare il consumo di un numero considerevole di indirizzi IP da parte dei pod.

Quando si progetta l'architettura di rete AWS, è importante ottimizzare il consumo IP di Amazon EKS a livello di VPC e di nodo. Questo ti aiuterà a mitigare i problemi di esaurimento dell'IP e ad aumentare la densità dei pod per nodo.

In questa sezione, discuteremo delle tecniche che possono aiutarvi a raggiungere questi obiettivi.

Ottimizza il consumo IP a livello di nodo

La delega dei prefissi è una funzionalità di Amazon Virtual Private Cloud (Amazon VPC) che consente di assegnare prefissi IPv4 o IPv6 alle istanze Amazon Elastic Compute Cloud (Amazon EC2). Aumenta gli indirizzi IP per interfaccia di rete (ENI), aumentando la densità dei pod per nodo e migliorando l'efficienza di elaborazione. La delega dei prefissi è supportata anche con Custom Networking.

Per informazioni dettagliate, consulta le sezioni Delega dei prefissi con nodi Linux e Delegazione dei prefissi con nodi Windows.

Riduci l'esaurimento dell'IP

Per evitare che i cluster consumino tutti gli indirizzi IP disponibili, consigliamo vivamente di dimensionare le sottoreti VPCs e le sottoreti tenendo conto della crescita.

L'adozione IPv6è un ottimo modo per evitare questi problemi fin dall'inizio. Tuttavia, per le organizzazioni le cui esigenze di scalabilità superano la pianificazione iniziale e non possono adottare IPv6, migliorare la progettazione del VPC è la risposta consigliata all'esaurimento degli indirizzi IP. La tecnica più comunemente utilizzata dai clienti di Amazon EKS consiste nell'aggiungere un dispositivo secondario non routabile CIDRs al VPC e configurare il VPC CNI per utilizzare questo spazio IP aggiuntivo durante l'allocazione degli indirizzi IP ai Pod. Questa operazione viene comunemente denominata rete personalizzata.

Tratteremo quali variabili del CNI di Amazon VPC puoi utilizzare per ottimizzare il pool caldo IPs assegnato ai tuoi nodi. Chiuderemo questa sezione con altri modelli architettonici che non sono intrinseci ad Amazon EKS ma che possono aiutare a mitigare l'esaurimento dell'IP.

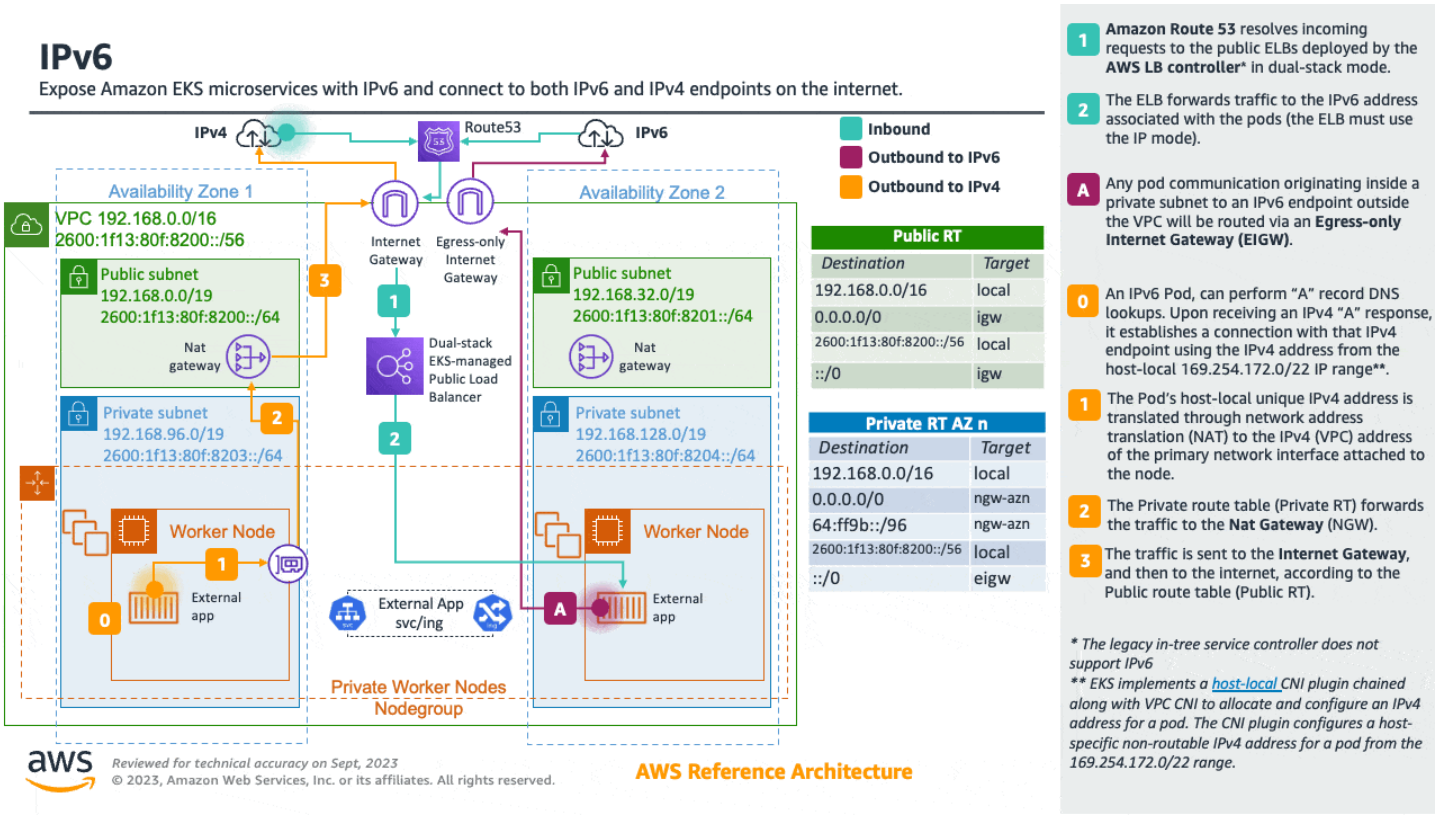
Uso (consigliato) IPv6

L'adozione IPv6 è il modo più semplice per aggirare le RFC1918 limitazioni; consigliamo vivamente di prendere in considerazione l'adozione IPv6 come prima opzione nella scelta di un'architettura di rete. IPv6 fornisce uno spazio totale di indirizzi IP notevolmente più ampio e gli amministratori dei cluster possono concentrarsi sulla migrazione e sulla scalabilità delle applicazioni senza dedicare sforzi per aggirare i limiti. IPv4

I cluster Amazon EKS supportano entrambi IPv4 e IPv6. Per impostazione predefinita, i cluster EKS utilizzano lo spazio degli IPv4 indirizzi. La specificazione di uno spazio di indirizzi IPv6 basato al

momento della creazione del cluster consentirà l'utilizzo di. IPv6 In un cluster IPv6 EKS, i pod e i servizi ricevono IPv6 indirizzi pur mantenendo la capacità degli IPv4 endpoint legacy di connettersi ai servizi in esecuzione sui IPv6 cluster e viceversa. Tutte le pod-to-pod comunicazioni all'interno di un cluster avvengono sempre al termine. IPv6 All'interno di un VPC (/56), la dimensione del blocco IPv6 CIDR per le IPv6 sottoreti è fissata a /64. Ciò fornisce 2^64 (circa 18 quintilioni) di indirizzi che consentono di scalare le implementazioni su EKS. IPv6

Per informazioni dettagliate, consulta la sezione Running IPv6 EKS Clusters e per un'esperienza pratica, consulta la sezione Understanding on IPv6 Amazon EKS del workshop Get hands-on with. IPv6



Ottimizza il consumo di IP nei IPv4 cluster

Questa sezione è dedicata ai clienti che eseguono applicazioni legacy a and/or cui non sono pronti per la migrazione. IPv6 Sebbene incoraggiamo tutte le organizzazioni a migrare verso il più presto IPv6 possibile, riconosciamo che alcune potrebbero aver ancora bisogno di cercare approcci alternativi per scalare i propri carichi di lavoro in container. IPv4 Per questo motivo, ti illustreremo anche i modelli architettonici per ottimizzare IPv4 (RFC1918) il consumo di spazio degli indirizzi con i cluster Amazon EKS.

Piano per la crescita

Come prima linea di difesa contro l'esaurimento degli indirizzi IP, consigliamo vivamente di dimensionare le sottoreti IPv4 VPCs e le sottoreti tenendo conto della crescita, per evitare che i cluster consumino tutti gli indirizzi IP disponibili. Non sarà possibile creare nuovi Pod o nodi se le sottoreti non dispongono di un numero sufficiente di indirizzi IP disponibili.

Prima di creare VPC e sottoreti, si consiglia di lavorare a ritroso rispetto alla scala del carico di lavoro richiesta. Ad esempio, quando i cluster vengono creati utilizzando eksctl (un semplice strumento CLI per la creazione e la gestione di cluster su EKS) /19 sottoreti vengono create per impostazione predefinita. Una netmask di /19 è adatta alla maggior parte dei tipi di carichi di lavoro e consente di allocare più di 8000 indirizzi.

> **Important**
>
> Quando si utilizzano le dimensioni VPCs e le sottoreti, potrebbero esserci diversi elementi (diversi dai pod e dai nodi) che possono consumare gli indirizzi IP, ad esempio Load Balancer, database RDS e altri servizi in-vpc.

Inoltre, Amazon EKS può creare fino a 4 interfacce di rete elastiche (X-ENI) necessarie per consentire la comunicazione verso il piano di controllo (maggiori informazioni qui). Durante gli upgrade dei cluster, Amazon EKS crea nuovi X- ENIs ed elimina quelli vecchi quando l'upgrade ha esito positivo. Per questo motivo consigliamo una maschera di rete di almeno /28 (16 indirizzi IP) per le sottoreti associate a un cluster EKS.

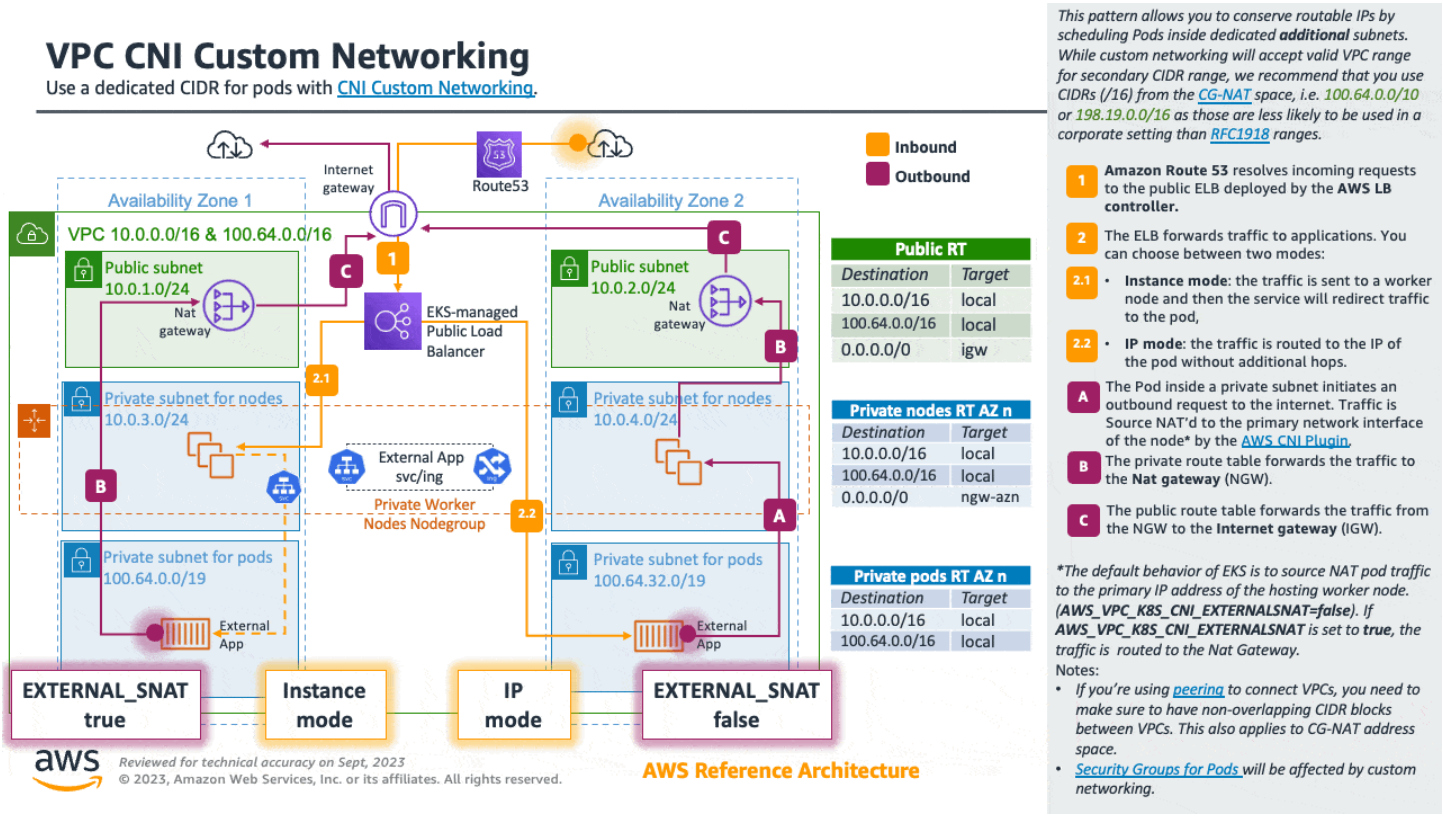
È possibile utilizzare il foglio di calcolo di esempio EKS Subnet Calculator per pianificare la rete. Il foglio di calcolo calcola l'utilizzo dell'IP in base ai carichi di lavoro e alla configurazione ENI VPC. L'utilizzo dell'IP viene confrontato con quello di una IPv4 sottorete per determinare se la configurazione e le dimensioni della sottorete sono sufficienti per il carico di lavoro. Tieni presente che, se le sottoreti del tuo VPC esauriscono gli indirizzi IP disponibili, ti suggeriamo di creare una nuova sottorete utilizzando i blocchi CIDR originali del VPC. Tieni presente che ora Amazon EKS consente la modifica delle sottoreti del cluster e dei gruppi di sicurezza.

Rete personalizzata

Se state per esaurire lo spazio RFC1918 IP, potete utilizzare il pattern Custom Networking per preservare la routability IPs programmando i Pod all'interno di sottoreti aggiuntive dedicate. Sebbene la rete personalizzata accetti un intervallo VPC valido per l'intervallo CIDR secondario, si consiglia

di utilizzare CIDRs (/16) dallo spazio CG-NAT, `100.64.0.0/10` ovvero `198.19.0.0/16` poiché è meno probabile che vengano utilizzati in un ambiente aziendale rispetto agli intervalli. RFC1918

Per informazioni dettagliate, consulta la sezione dedicata alle reti personalizzate.



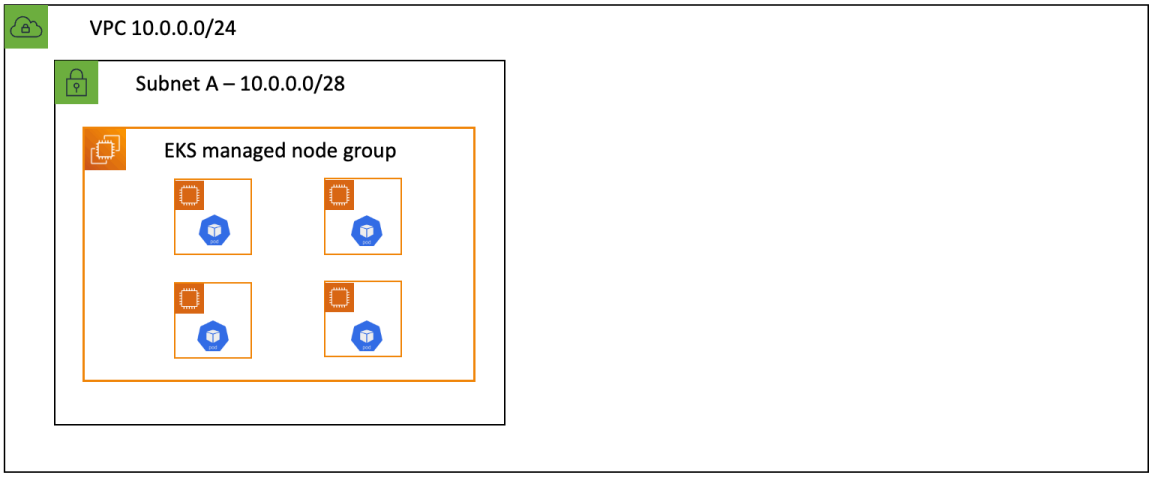
Rilevamento avanzato delle sottoreti

Enhanced Subnet Discovery offre un'alternativa di configurazione di rete semplificata per l'esaurimento dell'IP, contrassegnando nuove sottoreti in modo che siano rilevabili dal CNI di Amazon VPC. Con Enhanced Subnet Discovery, i carichi di lavoro attuali possono continuare a funzionare sulle stesse sottoreti e Amazon Elastic Kubernetes Service (Amazon EKS) può ora pianificare pod aggiuntivi sulle nuove «sottoreti utilizzabili».

Se le sottoreti correnti del tuo cluster stanno esaurendo gli indirizzi IP, puoi semplicemente aggiungere altre sottoreti al cluster Amazon EKS come segue:

1. Associa un nuovo blocco CIDR al tuo VPC.
2. Crea una nuova sottorete nel nuovo blocco CIDR e taggala con «kubernetes». io/role/cni» = «1".
3. Abilita la configurazione ENABLE_SUBNET_DISCOVERY del componente aggiuntivo Amazon VPC CNI su «true» (impostazione predefinita dalla versione 1.18.0).

Una volta abilitato Enhanced Subnet Discovery sui cluster VPC e Amazon EKS, nuove interfacce di rete elastiche ENIs () verranno collegate ai nodi Amazon EKS come descritto nel diagramma seguente:



Per ulteriori informazioni, consulta Amazon VPC CNI introduce Enhanced Subnet Discovery sul blog dei contenitori AWS.

Ottimizza la piscina calda IPs

Con la configurazione predefinita, il VPC CNI mantiene un intero ENI (e associato IPs) nella piscina calda. Ciò può richiedere un gran numero di istanze IPs, specialmente su tipi di istanze più grandi.

Se la sottorete del cluster ha un numero limitato di indirizzi IP disponibili, esamina queste variabili di ambiente di configurazione VPC CNI:

- `WARM_IP_TARGET`
- `MINIMUM_IP_TARGET`
- `WARM_ENI_TARGET`

Puoi configurare il valore di in modo che corrisponda `MINIMUM_IP_TARGET` esattamente al numero di Pod che prevedi di eseguire sui tuoi nodi. In questo modo, al momento della creazione dei Pod, il CNI potrà assegnare gli indirizzi IP dal pool caldo senza chiamare l'API EC2.

Tieni presente che l'impostazione di un valore `WARM_IP_TARGET` troppo basso comporterà ulteriori chiamate all'API EC2 e ciò potrebbe comportare una limitazione delle richieste. Per cluster di grandi dimensioni, usa «along «`MINIMUM_IP_TARGET`per evitare la limitazione delle richieste.

Per configurare queste opzioni, puoi scaricare il `aws-k8s-cni.yaml` manifesto e impostare le variabili di ambiente. Al momento in cui scriviamo, la versione più recente si trova qui. Verifica che la versione del valore di configurazione corrisponda alla versione VPC CNI installata.

> **Warning**
>
> Queste impostazioni verranno ripristinate ai valori predefiniti quando aggiorni il CNI. Effettua un backup del CNI prima di aggiornarlo. Rivedi le impostazioni di configurazione per determinare se è necessario riapplicarle dopo che l'aggiornamento è andato a buon fine.

È possibile regolare i parametri CNI all'istante senza tempi di inattività per le applicazioni esistenti, ma è consigliabile scegliere valori che supportino le esigenze di scalabilità. Ad esempio, se lavori con carichi di lavoro in batch, ti consigliamo di aggiornare l'impostazione predefinita `WARM_ENI_TARGET` per soddisfare le esigenze di scala Pod. L'impostazione `WARM_ENI_TARGET` su un valore elevato mantiene sempre il pool IP caldo necessario per eseguire carichi di lavoro in batch di grandi dimensioni ed evitare quindi ritardi nell'elaborazione dei dati.

> **Warning**
>
> Il miglioramento del design del VPC è la risposta consigliata all'esaurimento degli indirizzi IP. Prendi in considerazione soluzioni come IPv6 Secondary. CIDRs La regolazione di questi valori per ridurre al minimo il numero di Warm IPs dovrebbe essere una soluzione temporanea dopo l'esclusione di altre opzioni. Una configurazione errata di questi valori può interferire con il funzionamento del cluster. Prima di apportare modifiche a un sistema di produzione, assicuratevi di leggere le considerazioni contenute in questa pagina.

Monitora l'inventario degli indirizzi IP

Oltre alle soluzioni sopra descritte, è importante anche avere visibilità sull'utilizzo dell'IP. È possibile monitorare l'inventario degli indirizzi IP delle sottoreti utilizzando CNI Metrics Helper. Alcune delle metriche disponibili sono:

- numero massimo che ENIs il cluster può supportare
- numero di quelli ENIs già assegnati
- numero di indirizzi IP attualmente assegnati ai Pod
- numero totale e massimo di indirizzi IP disponibili

Puoi anche impostare CloudWatch allarmi per ricevere una notifica se una sottorete sta esaurendo gli indirizzi IP.

> **Warning**
>
> Assicurati che la `DISABLE_METRICS` variabile per VPC CNI sia impostata su false.

Ulteriori considerazioni

Esistono altri modelli architettonici non intrinseci ad Amazon EKS che possono contribuire all'esaurimento dell'IP. Ad esempio, puoi ottimizzare la comunicazione VPCs o condividere un VPC su più account per limitare l'allocazione degli IPv4 indirizzi.

Scopri di più su questi modelli qui:

- Progettazione di reti Amazon VPC iperscalabili,
- Crea una connettività multi-VPC sicura per più account con Amazon VPC Lattice.

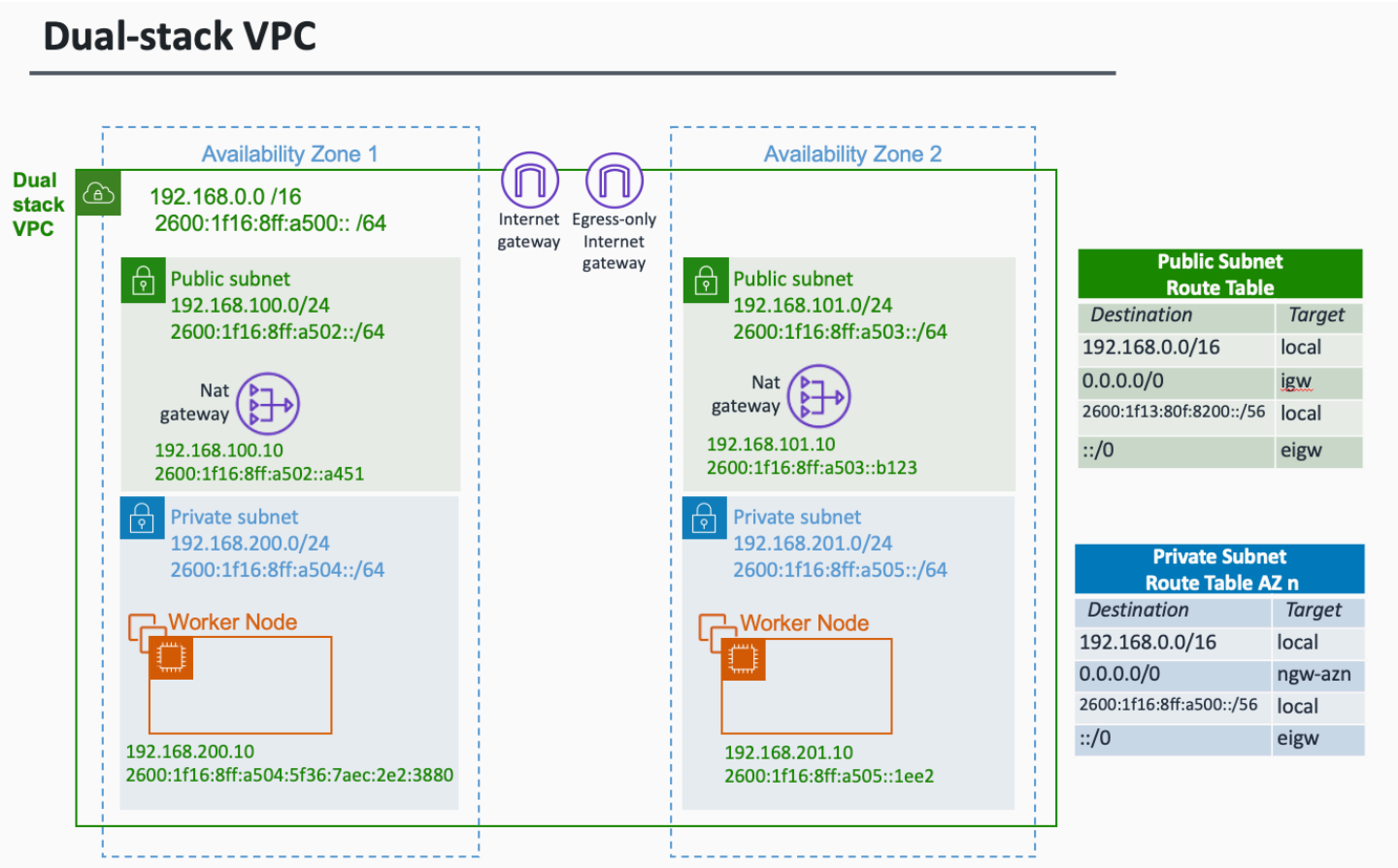
IPv6 Esecuzione di cluster EKS

EKS in IPv6 mode risolve il problema dell' IPv4 esaurimento che si manifesta spesso nei cluster EKS su larga scala. Il supporto di EKS IPv6 si concentra sulla risoluzione del problema dell' IPv4 esaurimento, che deriva dalle dimensioni limitate dello spazio degli indirizzi. IPv4 Questa è una preoccupazione importante sollevata da alcuni dei nostri clienti ed è distinta dalla funzionalità IPv4 Kubernetes/dual-stack. IPv6 EKS/ IPv6 fornirà anche la flessibilità necessaria per interconnettere

i confini di rete, riducendo così al minimo le possibilità di sovrapposizione CIDR e risolvendo IPv6 CIDRs così un problema duplice (In-Cluster, Cross-Cluster). Quando si distribuiscono i cluster EKS in modalità (--ip-family ipv6), l'azione non è reversibile. IPv6 In parole semplici, il IPv6 supporto EKS è abilitato per l'intera durata del cluster.

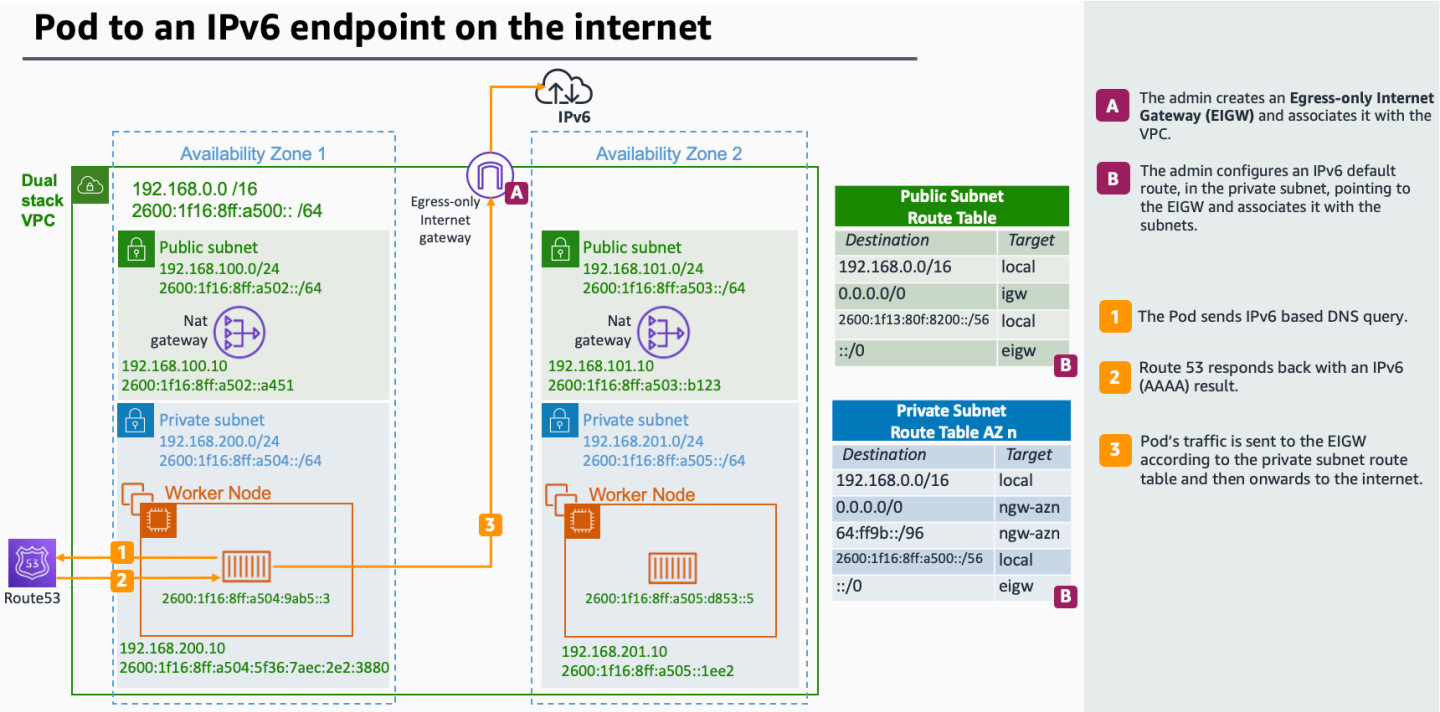
In un cluster IPv6 EKS, Pods and Services riceveranno IPv6 gli indirizzi mantenendo la compatibilità con gli IPv4 endpoint legacy. Ciò include la possibilità per gli IPv4 endpoint esterni di accedere ai servizi all'interno del cluster e i Pod di accedere agli endpoint esterni. IPv4

Il IPv6 supporto di Amazon EKS sfrutta le funzionalità IPv6 VPC native. A ogni VPC viene assegnato un prefisso di IPv4 indirizzo (la dimensione del blocco CIDR può essere compresa tra /16 e /28) e un prefisso di indirizzo /56 univoco (fisso) dall'interno del GUA (Global Unicast IPv6 Address) di Amazon; puoi assegnare un prefisso di indirizzo /64 a ciascuna sottorete del tuo VPC. IPv4 le funzionalità, come le tabelle di percorso, gli elenchi di controllo degli accessi alla rete, il peering e la risoluzione DNS, funzionano allo stesso modo in un IPv6 VPC abilitato. Il VPC viene quindi denominato VPC dual-stack, seguendo le sottoreti dual-stack, il diagramma seguente illustra il modello di base VPC che supporta i cluster basati su EKS/: IPV4 IPv6 IPv6



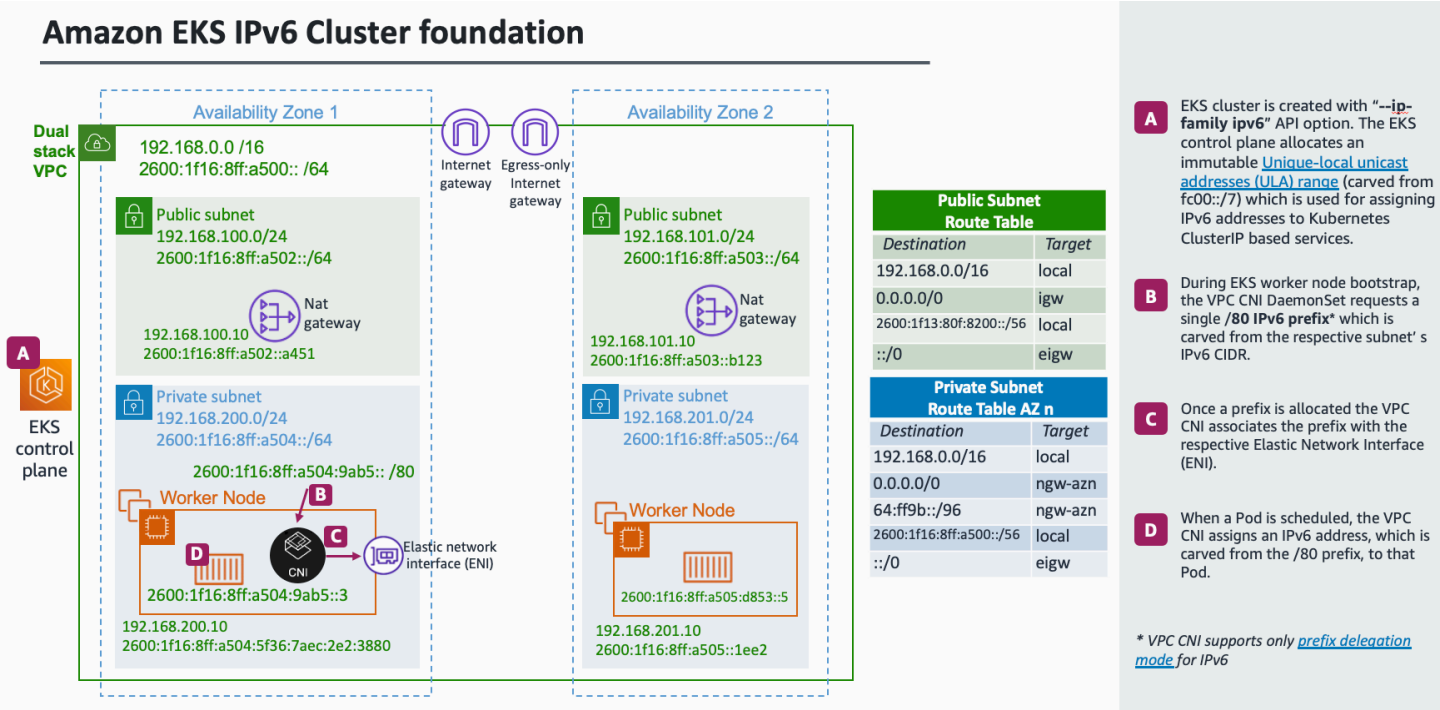
Nel IPv6 mondo, ogni indirizzo è instradabile su Internet. Per impostazione predefinita, VPC alloca IPv6 CIDR dall'intervallo GUA pubblico. Tuttavia, dall'agosto 2024 puoi anche utilizzare l' IPv6 indirizzamento privato per VPCs e le sottoreti con Amazon VPC IP Address Manager (IPAM). Per ulteriori informazioni, consulta questo post sul blog di AWS Networking e la documentazione VPC.

Il diagramma seguente mostra un flusso di uscita di IPv6 Internet Pod all'interno di un cluster EKS/: IPv6



Le migliori pratiche per l'implementazione delle IPv6 sottoreti sono disponibili nella guida per l'utente di VPC.

In un cluster IPv6 EKS, i nodi e i Pod ricevono indirizzi pubblici. IPv6 EKS assegna IPv6 gli indirizzi ai servizi in base agli indirizzi Unicast Local IPv6 Unicast Unicast Unicast (ULA) unici. Il CIDR del servizio ULA per un IPv6 cluster viene assegnato automaticamente durante la fase di creazione del cluster e non può essere specificato, a differenza di. IPv4 Il diagramma seguente illustra un modello di base del piano dati del piano di controllo del cluster IPv6 basato su EKS/:



Panoramica di

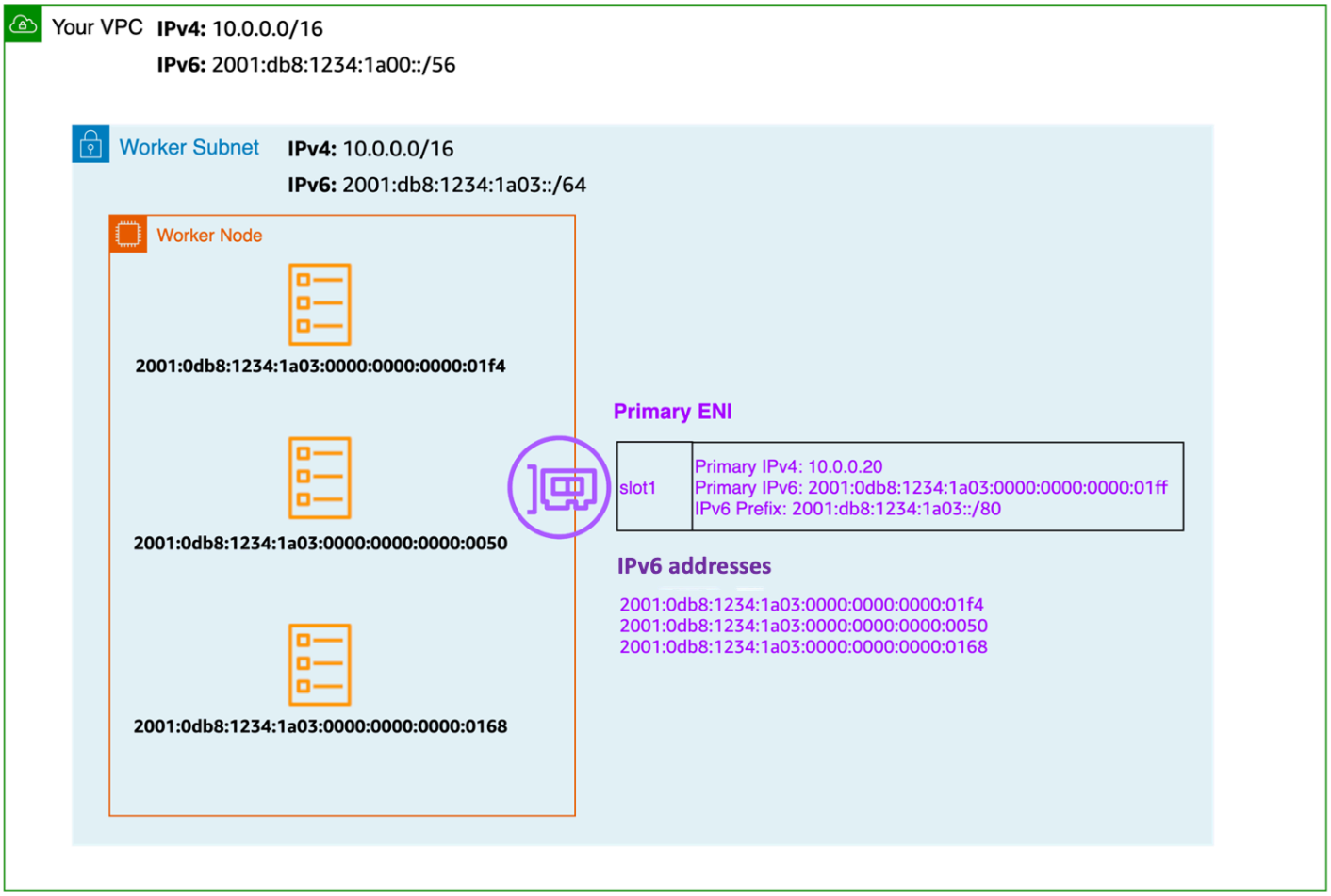
EKS/ IPv6 è supportato solo in modalità prefisso (modalità di assegnazione IP ENI del plug-in VPC-CNI). Scopri di più sulla modalità Prefix.

L'assegnazione del prefisso funziona solo su istanze EC2 basate su Nitro, quindi EKS/IPv6 è supportato solo quando il piano dati del cluster utilizza istanze EC2 basate su Nitro.

In parole semplici, un IPv6 prefisso di /80 (per worker-node) produrrà ~10^14 indirizzi, il fattore limitante non sarà più costituito solo dalla densità dei pod (per quanto riguarda le risorse). IPv6 IPs

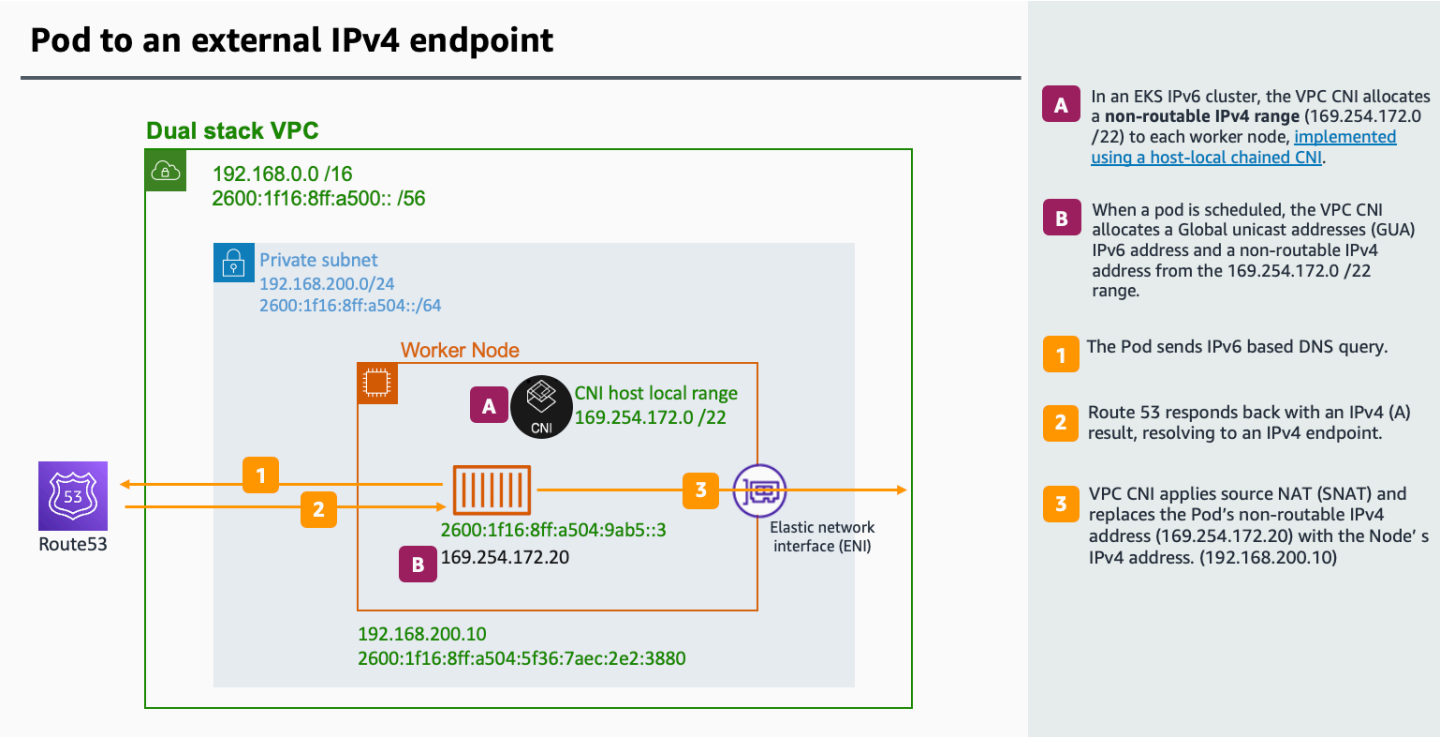
IPv6 l'assegnazione del prefisso avviene solo al momento del bootstrap del nodo di lavoro EKS. È noto che questo comportamento mitiga gli scenari in cui i IPv4 cluster EKS/ EKS/ ad alto tasso di abbandono dei Pod subiscono spesso ritardi nella pianificazione dei Pod a causa delle chiamate API limitate generate dal plug-in VPC CNI (ipamd) volto ad allocare indirizzi privati in modo tempestivo. IPv4 È noto anche che le manopole avanzate del plug-in VPC-CNI regolano inutilmente WARM_IP/ ENI, MINIMUM_IP.

Il diagramma seguente ingrandisce un'interfaccia di rete elastica (ENI) a nodi di lavoro: IPv6



A ogni nodo di lavoro EKS vengono assegnati IPv6 indirizzi IPv4 e voci DNS corrispondenti. Per un determinato nodo di lavoro, viene utilizzato un solo IPv4 indirizzo dalla sottorete dual-stack. Il supporto EKS per IPv6 consente di comunicare con gli IPv4 endpoint (AWS, on-premise, Internet) attraverso un modello di sola uscita molto apprezzato. IPv4 EKS implementa un plug-in CNI locale dell'host, secondario al plug-in VPC CNI, che alloca e configura un indirizzo per un Pod. IPv4 Il plugin CNI configura un indirizzo non routabile specifico dell'host per un Pod dell'intervallo 169.254.172.0/22. IPv4 L' IPv4 indirizzo assegnato al Pod è unico per il worker-node e non viene pubblicizzato oltre il worker-node. 169.254.172.0/22 fornisce fino a 1024 indirizzi unici che possono supportare tipi di istanze di grandi dimensioni. IPv4

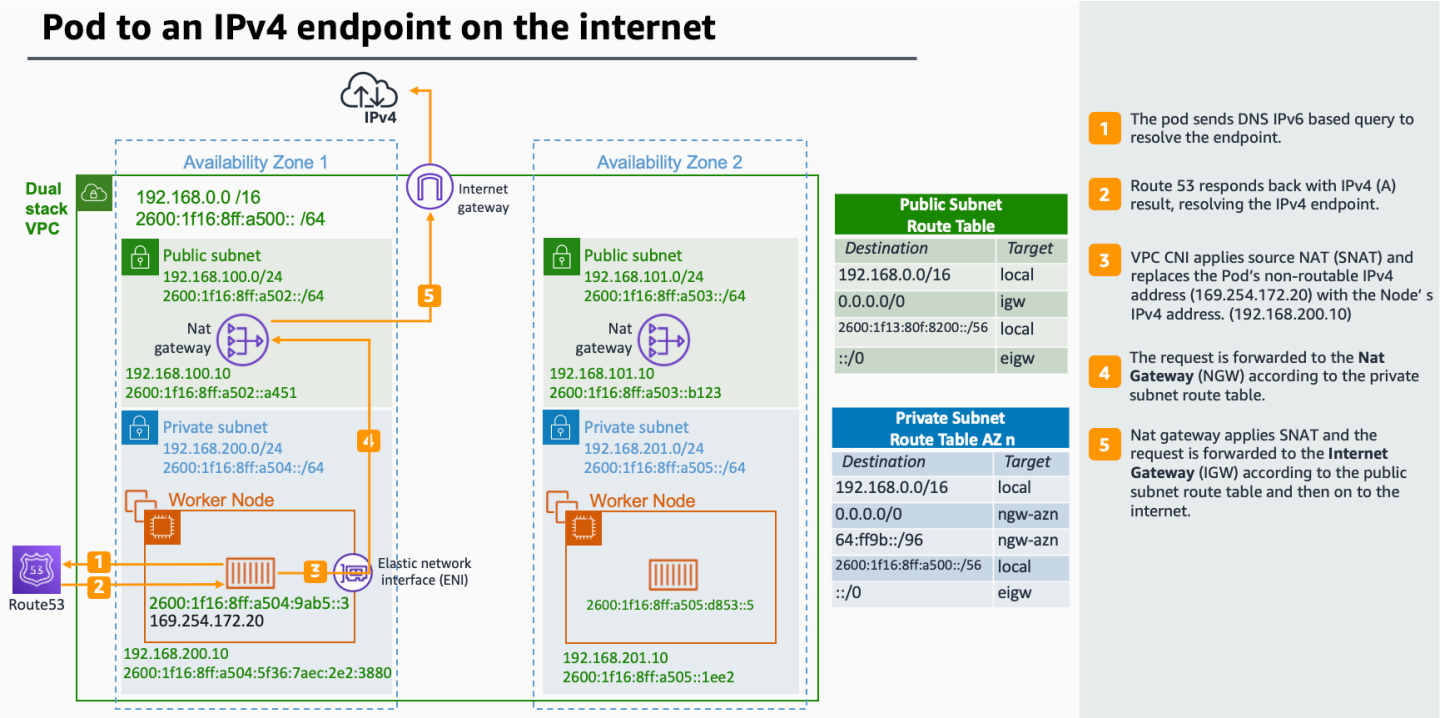
Il diagramma seguente mostra il flusso di un Pod che si connette a un endpoint esterno al confine del cluster (non Internet): IPv6 IPv4



Nel diagramma precedente Pods eseguirà una ricerca DNS per l'endpoint e, dopo aver ricevuto una risposta IPv4 «A», l'indirizzo IPv4 univoco del Pod solo per il nodo viene tradotto tramite la traduzione degli indirizzi di rete di origine (SNAT) all'indirizzo IPv4 privato (VPC) dell'interfaccia di rete principale collegata al nodo di lavoro EC2.

> **Note**
>
> Il modello precedente richiede DNS64 che sia disabilitato nelle sottoreti in cui sono in esecuzione EKS/ IPv6 Pod. Quando DNS64 è abilitato, il resolver DNS restituisce un indirizzo sintetizzato IPv6 per gli endpoint -only insieme a un indirizzo. IPv4 IPv4 Di conseguenza, il traffico passa attraverso la NAT64 funzionalità del gateway NAT (se inclusa nell'architettura) anziché rimanere all'interno del VPC come mostrato nello schema precedente. Ciò può comportare un utilizzo imprevisto del NAT Gateway e i relativi costi.

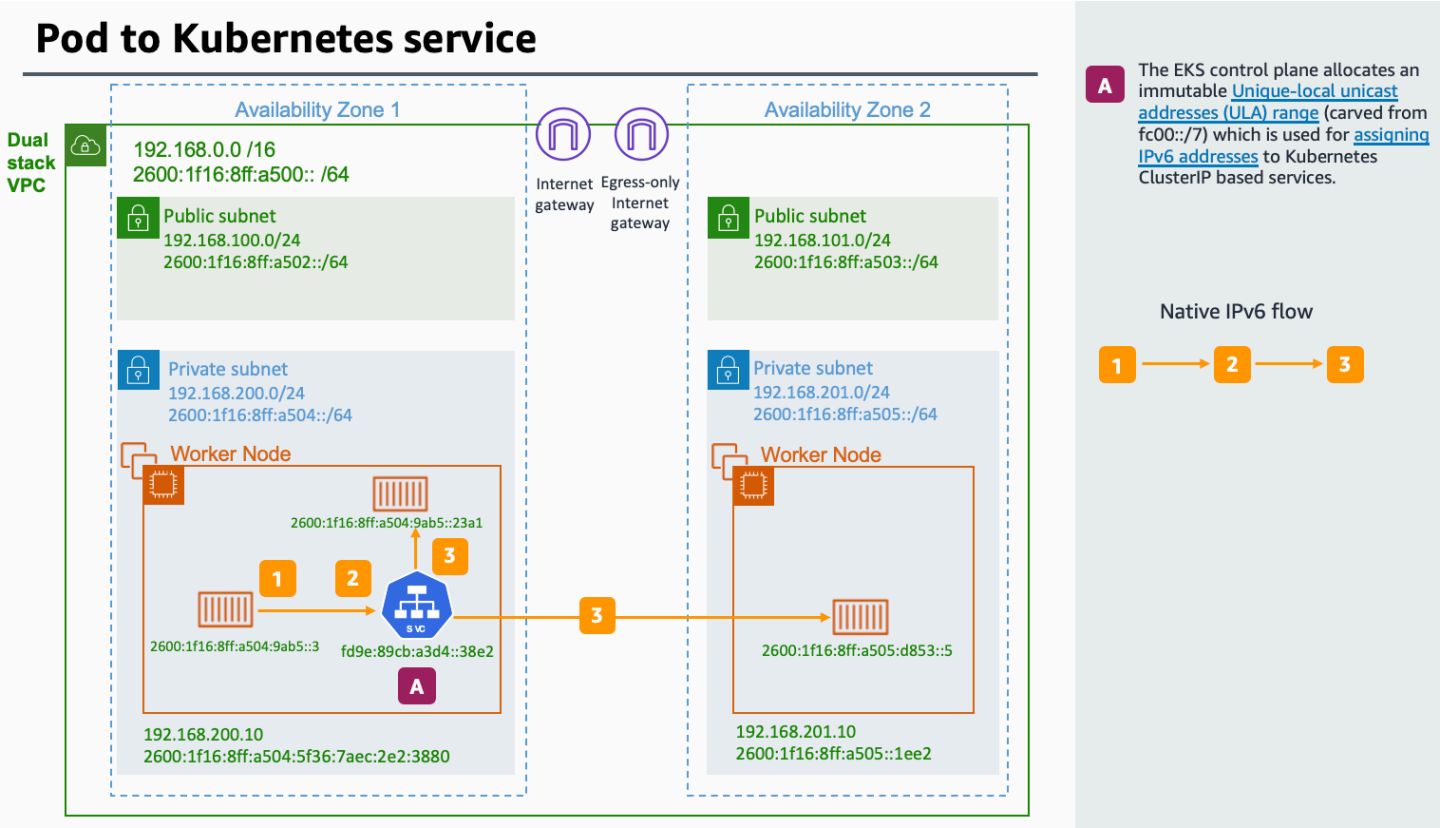
EKS/ IPv6 Pods dovrà inoltre connettersi agli IPv4 endpoint su Internet utilizzando IPv4 indirizzi pubblici, per far sì che esista un flusso simile. Il diagramma seguente illustra il flusso di un IPv6 Pod che si connette a un IPv4 endpoint esterno ai confini del cluster (instradabile su Internet):



Nel diagramma precedente Pods eseguirà una ricerca DNS per l'endpoint e, dopo aver ricevuto una risposta IPv4 «A», l'indirizzo IPv4 univoco del Pod solo per il nodo viene tradotto tramite la traduzione degli indirizzi di rete di origine (SNAT) all'indirizzo IPv4 privato (VPC) dell'interfaccia di rete principale collegata al nodo di lavoro EC2. L'indirizzo IPv4 del pod (origine IPv4: IP primario EC2) viene quindi indirizzato al gateway NAT IPv4 dove l'IP primario EC2 viene tradotto (SNAT) in un indirizzo IP IPv4 pubblico instradabile su Internet valido (IP pubblico assegnato al gateway NAT).

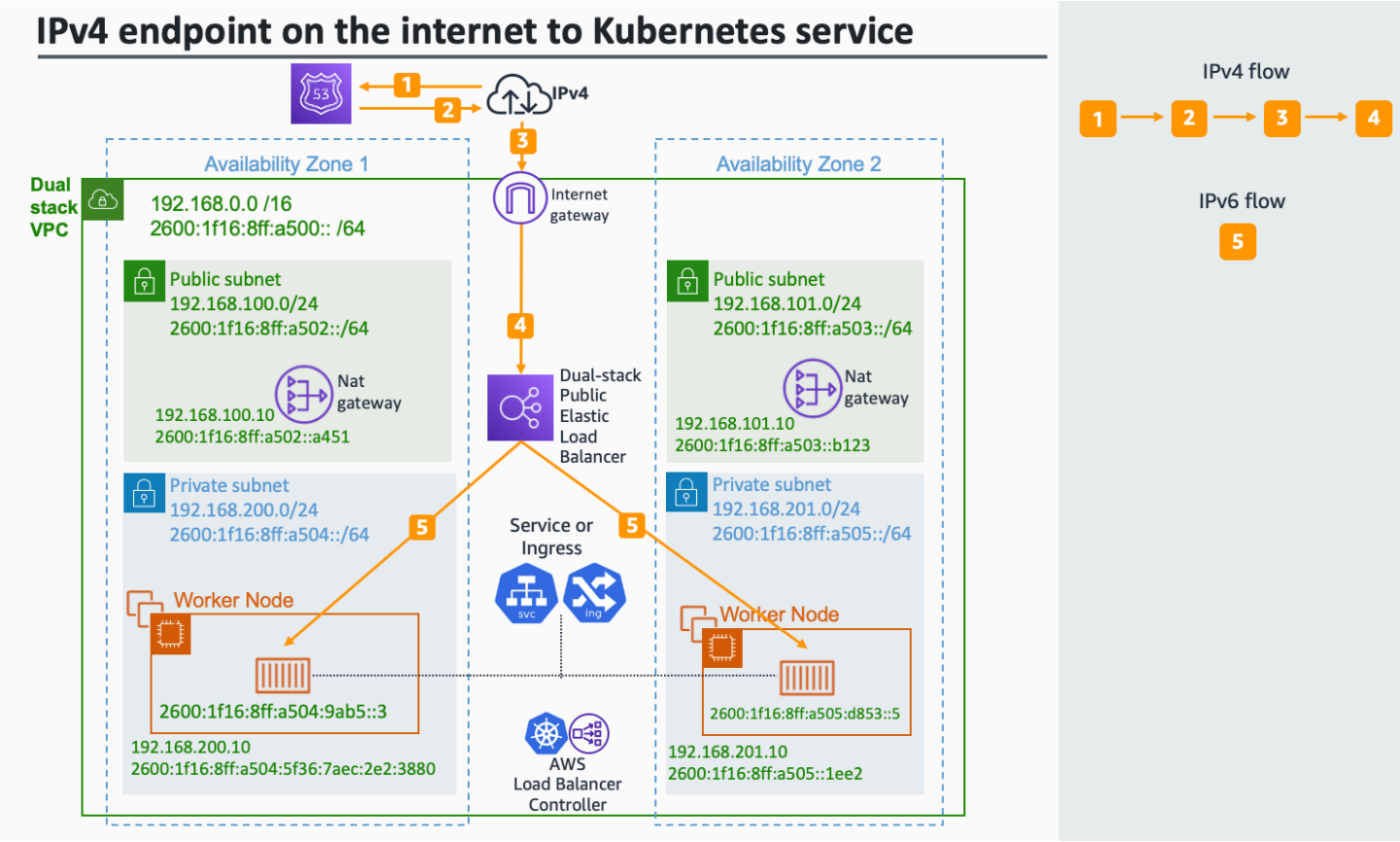
Qualsiasi comunicazione tra i nodi utilizza sempre un indirizzo. Pod-to-Pod IPv6 VPC CNI configura iptables per gestire IPv6 mentre blocca qualsiasi connessione. IPv4

I servizi Kubernetes riceveranno solo indirizzi IPv6 (ClusterIP) da Unique Local Unicast Addresses (ULA). IPv6 Il CIDR del servizio ULA per un IPv6 cluster viene assegnato automaticamente durante la fase di creazione del cluster EKS e non può essere modificato. Il diagramma seguente illustra il flusso da Pod a Kubernetes Service:



I servizi sono esposti a Internet utilizzando un sistema di bilanciamento del carico AWS. Il sistema di bilanciamento del carico riceve indirizzi pubblici IPv4 e IPv6 indirizzi, ovvero un sistema di bilanciamento del carico dual-stack. Per IPv4 i client che accedono ai servizi IPv6 Kubernetes del cluster, il load balancer esegue la traduzione. IPv4 IPv6

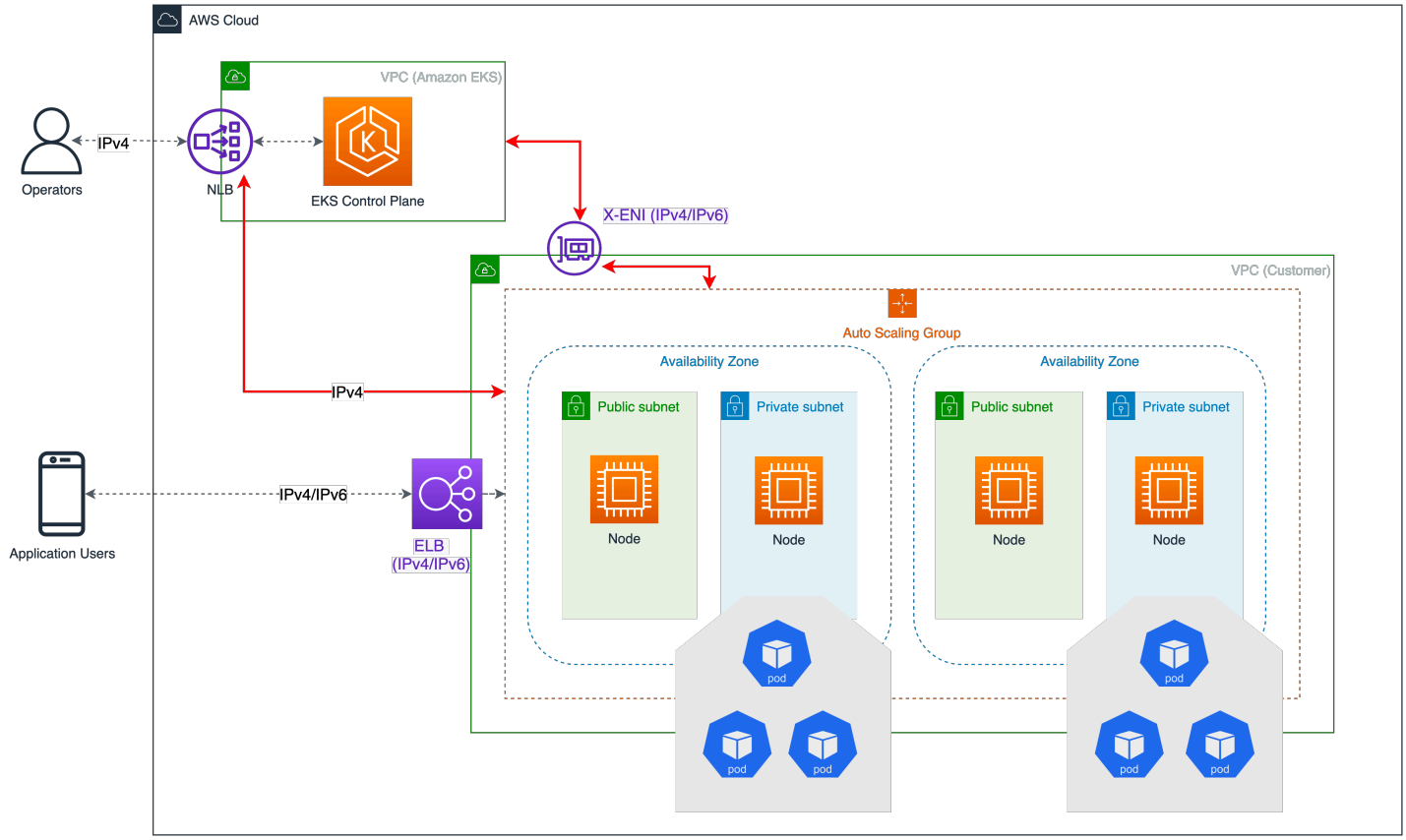
Amazon EKS consiglia di eseguire nodi di lavoro e pod in sottoreti private. Puoi creare bilanciatori di carico pubblici nelle sottoreti pubbliche che bilanciano il carico del traffico verso i Pod in esecuzione su nodi che si trovano in sottoreti private. Il diagramma seguente mostra un utente di Internet che accede a un servizio basato su EKS/ Ingress: IPv4 IPv6



> **Note**
>
> Lo schema sopra riportato richiede la distribuzione della versione più recente del controller di bilanciamento del carico AWS.

EKS Control Plane (comunicazione Data Plane)

EKS fornirà Cross-Account ENIs (X-ENIs) in modalità dual stack (IPv4/IPv6). I componenti del nodo Kubernetes come kubelet e kube-proxy sono configurati per supportare il dual stack. Kubelet e kube-proxy vengono eseguiti in modalità HostNetwork e si collegano a entrambi gli indirizzi collegati all'interfaccia di rete principale di un nodo. IPv4 IPv6 L'api-server Kubernetes comunica con i Pod e i componenti del nodo tramite la base X-. ENIs IPv6 I pod comunicano con gli api-server tramite X-ENIs e la comunicazione tra Pod e api-server utilizza sempre la modalità. IPv6



Raccomandazioni

Pianificazione basata sulle risorse di calcolo

Un singolo IPv6 prefisso è sufficiente per eseguire più Pod su un singolo nodo. Ciò rimuove efficacemente anche le limitazioni ENI e IP sul numero massimo di Pod su un nodo. Sebbene IPv6 rimuova la dipendenza diretta dai Max-Pods, quando si utilizzano allegati di prefisso con tipi di istanze più piccoli come m5.large, è probabile che si esauriscano le risorse di CPU e memoria dell'istanza molto prima di esaurirne gli indirizzi IP. È necessario impostare manualmente il valore Pod massimo consigliato da EKS se si utilizzano gruppi di nodi autogestiti o un gruppo di nodi gestiti con un ID AMI personalizzato.

È possibile utilizzare la seguente formula per determinare il numero massimo di Pod che è possibile distribuire su un nodo per un IPv6 cluster EKS.

```
((Number of network interfaces for instance type (number of prefixes per network
 interface-1)* 16) + 2
```

```
((3 ENIs)_((10 secondary IPs per ENI-1)_ 16)) + 2 = 460 (real)
```

I gruppi di nodi gestiti calcolano automaticamente il numero massimo di Pod. Evita di modificare il valore consigliato da EKS per il numero massimo di Pod per evitare errori di pianificazione dei Pod dovuti alla limitazione delle risorse.

Valuta lo scopo della rete personalizzata esistente

Se la rete personalizzata è attualmente abilitata, Amazon EKS consiglia di rivalutarne la necessità con. IPv6 Se hai scelto di utilizzare reti personalizzate per risolvere il problema dell' IPv4 esaurimento, non è più necessario con. IPv6 Se utilizzate una rete personalizzata per soddisfare un requisito di sicurezza, ad esempio una rete separata per nodi e pod, vi consigliamo di inviare una richiesta di roadmap EKS.

Pod Fargate in EKS/ Cluster IPv6

Supporti EKS IPv6 per i Pod che funzionano su Fargate. I pod in esecuzione su Fargate consumeranno IPv4 indirizzi IPv6 privati routabili VPC ricavati dagli intervalli CIDR del VPC (). IPv4 IPv6 In parole semplici, l'ampia densità del cluster EKS/Fargate Pods sarà limitata agli indirizzi disponibili e agli indirizzi. IPv4 IPv6 Si consiglia di dimensionare il dual-stack per una crescita subnets/VPC CIDRs futura. Non sarà possibile pianificare nuovi Fargate Pod se la sottorete sottostante non contiene un indirizzo disponibile, indipendentemente dagli IPv4 indirizzi disponibili. IPv6

Implementa l'AWS Load Balancer Controller (LBC)

Il controller di servizio Kubernetes integrato a monte non supporta. IPv6 Consigliamo di utilizzare la versione più recente del componente aggiuntivo AWS Load Balancer Controller. LBC implementerà un NLB dual-stack o un ALB dual-stack solo dopo aver utilizzato la definizione Kubernetes corrispondente annotata con: e service/ingress `"alb.ingress.kubernetes.io/ip-address-type: dualstack"` `"alb.ingress.kubernetes.io/target-type: ip"`

Rete personalizzata

> **Tip**
>
> Esplora le best practice tramite i workshop Amazon EKS.

Per impostazione predefinita, Amazon VPC CNI assegnerà ai Pod un indirizzo IP selezionato dalla sottorete principale. La sottorete principale è la sottorete CIDR a cui è collegato l'ENI primario, in genere la sottorete del nodo/host.

Se la sottorete CIDR è troppo piccola, il CNI potrebbe non essere in grado di acquisire un numero sufficiente di indirizzi IP secondari da assegnare ai Pod. Questa è una sfida comune per i cluster EKS. IPv4

La rete personalizzata è una soluzione a questo problema.

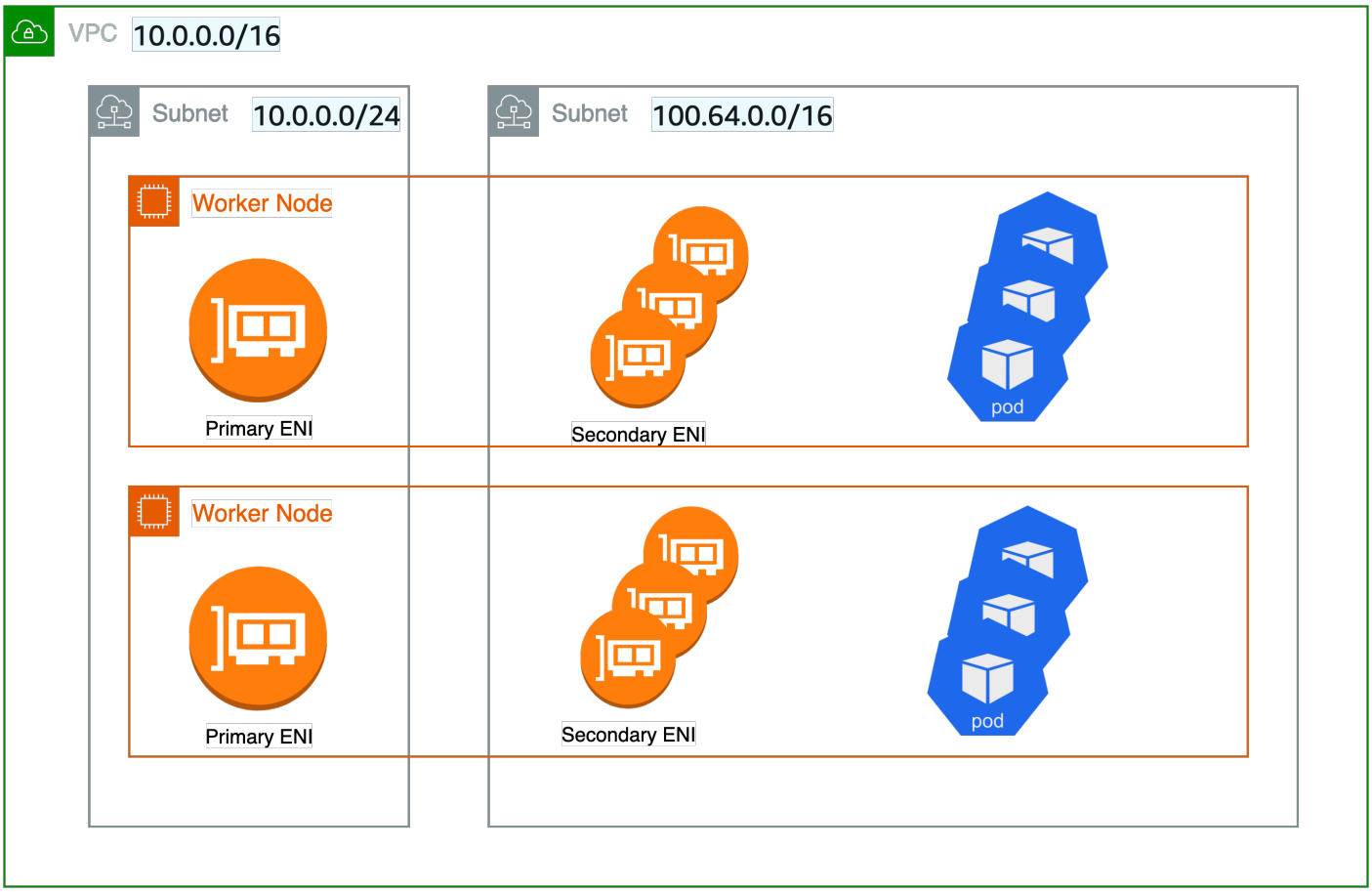
La rete personalizzata risolve il problema dell'esaurimento dell'IP assegnando il nodo e il pod IPs dagli spazi di indirizzi VPC secondari (CIDR). Il supporto di rete personalizzato supporta risorse personalizzate. ENIConfig ENIConfig Include un intervallo CIDR di sottorete alternativo (ricavato da un CIDR VPC secondario), insieme ai gruppi di sicurezza a cui apparterranno i Pod. Quando la rete personalizzata è abilitata, il VPC CNI crea una rete secondaria nella sottorete definita ENIs in. ENIConfig Il CNI assegna ai Pods un indirizzo IP da un intervallo CIDR definito in un CRD. ENIConfig

Poiché l'ENI principale non viene utilizzato da reti personalizzate, il numero massimo di Pod che è possibile eseguire su un nodo è inferiore. I Pod della rete host continuano a utilizzare l'indirizzo IP assegnato all'ENI primario. Inoltre, l'ENI primario viene utilizzato per gestire la traduzione della rete di origine e indirizzare il traffico Pods all'esterno del nodo.

Configurazione di esempio

Sebbene la rete personalizzata accetti un intervallo VPC valido per l'intervallo CIDR secondario, si consiglia di utilizzare CIDRs (/16) dallo spazio CG-NAT, ad esempio 100.64.0.0/10 o 198.19.0.0/16 poiché è meno probabile che vengano utilizzati in un ambiente aziendale rispetto ad altri intervalli. RFC1918 Per ulteriori informazioni sulle associazioni di blocchi CIDR consentite e limitate che puoi utilizzare con il tuo VPC, IPv4 consulta Restrizioni alle associazioni di blocchi CIDR nella sezione sul dimensionamento di VPC e sottorete della documentazione VPC.

Come illustrato nel diagramma seguente, l'interfaccia di rete elastica (ENI) principale del nodo di lavoro utilizza ancora l'intervallo CIDR VPC primario (in questo caso 10.0.0.0/16) ma il secondario utilizza l'intervallo CIDR VPC secondario (in questo caso ENIs 100.64.0.0/16). Ora, per fare in modo che i Pod utilizzino l'intervallo CIDR 100.64.0.0/16, è necessario configurare il plug-in CNI per utilizzare una rete personalizzata. Puoi seguire i passaggi come documentato qui.



Se desideri che il CNI utilizzi una rete personalizzata, imposta la `AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG` variabile di ambiente su. `true`

```
kubectl set env daemonset aws-node -n kube-system
 AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG=true
```

Quando`AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG=true`, il CNI assegnerà l'indirizzo IP del Pod da una sottorete definita in. `ENIConfig` La risorsa `ENIConfig` personalizzata viene utilizzata per definire la sottorete in cui verranno programmati i Pod.

```
apiVersion : crd.k8s.amazonaws.com/v1alpha1
kind : ENIConfig
metadata:
  name: us-west-2a
spec:
  securityGroups:
    - sg-0dff111a1d11c1c11
```

```
  subnet: subnet-011b111c1f11fdf11
```

Dopo aver creato le risorse `ENIconfig` personalizzate, dovrai creare nuovi nodi di lavoro e drenare i nodi esistenti. I nodi di lavoro e i Pod esistenti rimarranno inalterati.

Raccomandazioni

Usa reti personalizzate quando

Ti consigliamo di prendere in considerazione la possibilità di utilizzare una rete personalizzata se hai a che fare con IPv4 l'esaurimento e non puoi IPv6 ancora utilizzarla. Il supporto di Amazon EKS per RFC6598lo spazio ti consente di scalare i pod oltre le RFC1918sfide legate all'esaurimento. Valuta la possibilità di utilizzare la delega di prefissi con reti personalizzate per aumentare la densità dei Pod su un nodo.

Potresti prendere in considerazione una rete personalizzata se hai l'esigenza di sicurezza di eseguire i Pod su una rete diversa con requisiti di gruppo di sicurezza diversi. Quando è abilitata la rete personalizzata, i pod utilizzano sottoreti o gruppi di sicurezza diversi, come definito nell'interfaccia ENIConfig di rete principale del nodo.

La rete personalizzata è infatti l'opzione ideale per implementare più cluster e applicazioni EKS per connettere i servizi di datacenter locali. Puoi aumentare il numero di indirizzi privati (RFC1918) accessibili a EKS nel tuo VPC per servizi come Amazon Elastic Load Balancing e NAT-GW, utilizzando al contempo lo spazio CG-NAT non routabile per i tuoi pod su più cluster. La rete personalizzata con il gateway di transito e un VPC di Shared Services (inclusi i gateway NAT in diverse zone di disponibilità per un'elevata disponibilità) consente di fornire flussi di traffico scalabili e prevedibili. Questo post del blog descrive uno schema architettonico che rappresenta uno dei modi più consigliati per connettere EKS Pods a una rete di data center utilizzando una rete personalizzata.

Evita le reti personalizzate quando

Pronto per l'implementazione IPv6

Le reti personalizzate possono mitigare i problemi di esaurimento dell'IP, ma richiedono un sovraccarico operativo aggiuntivo. Se attualmente stai implementando un IPv6 VPC dual-stack IPv4 (/) o se il tuo piano include il IPv6 supporto, ti consigliamo invece di implementare i cluster. IPv6 Puoi configurare i cluster IPv6 EKS e migrare le tue app. In un cluster IPv6 EKS, sia Kubernetes che Pods ricevono un IPv6 indirizzo e possono comunicare in entrata e in uscita verso entrambi gli endpoint. IPv4 IPv6 Consulta le best practice per l'esecuzione di cluster EKS. IPv6

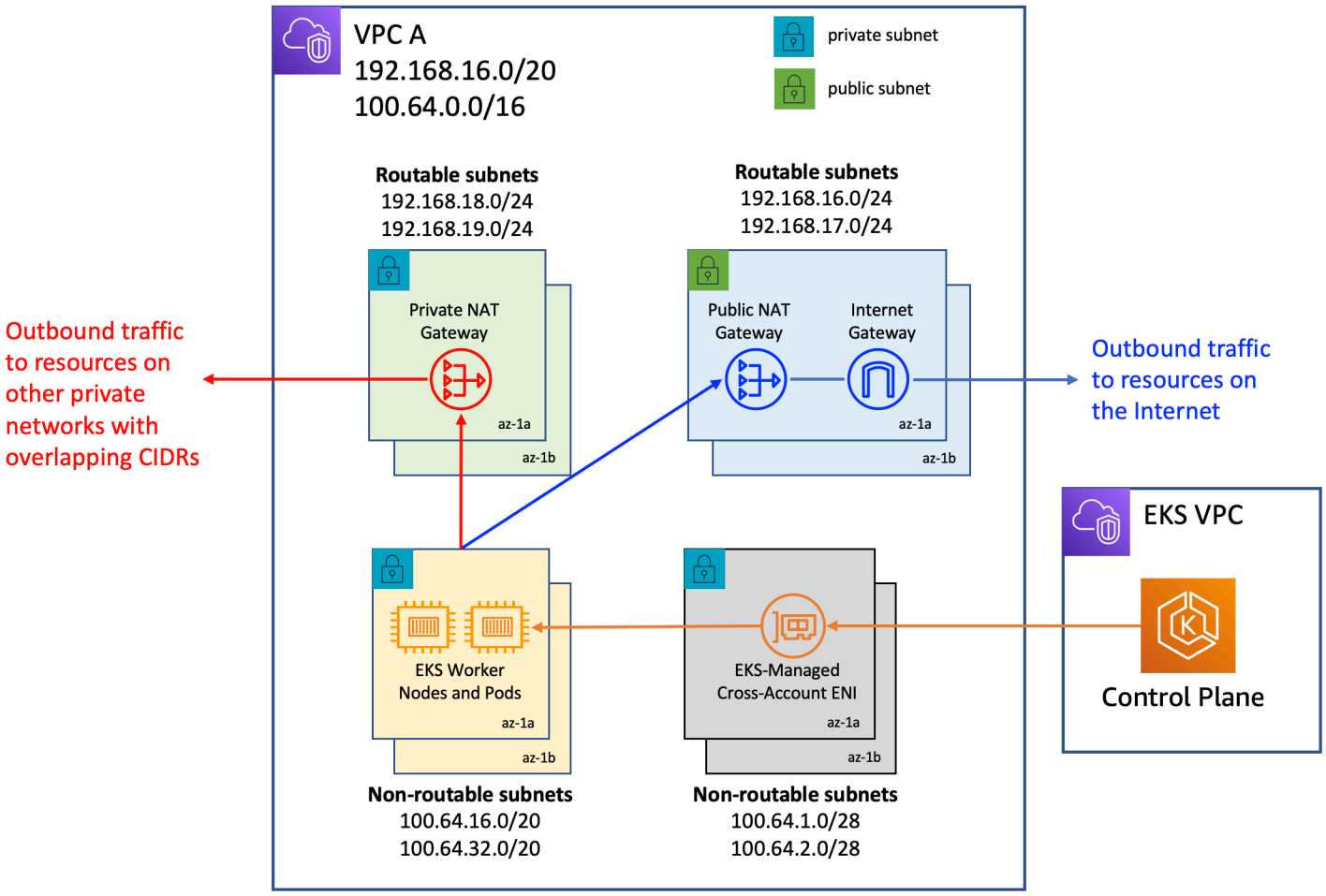
Spazio CG-NAT esaurito

Inoltre, se attualmente utilizzi CIDRs lo spazio CG-NAT o non riesci a collegare un CIDR secondario al tuo VPC del cluster, potresti dover esplorare altre opzioni, come l'utilizzo di un CNI alternativo. Ti consigliamo vivamente di ottenere supporto commerciale o di possedere le conoscenze interne per eseguire il debug e inviare patch al progetto di plug-in CNI open source. Consulta la guida per l'utente dei plugin CNI alternativi per maggiori dettagli.

Usa Private NAT Gateway

Amazon VPC ora offre funzionalità di gateway NAT privati. Il gateway NAT privato di Amazon consente alle istanze in sottoreti private di connettersi ad altre VPCs reti locali con sovrapposizioni. CIDRs Prendi in considerazione la possibilità di utilizzare il metodo descritto in questo post del blog per utilizzare un gateway NAT privato per risolvere i problemi di comunicazione dei carichi di lavoro EKS causati dalla sovrapposizione, una lamentela significativa espressa dai nostri clienti. CIDRs La rete personalizzata non è in grado di risolvere da sola le difficoltà del CIDR che si sovrappongono e ciò si aggiunge alle sfide di configurazione.

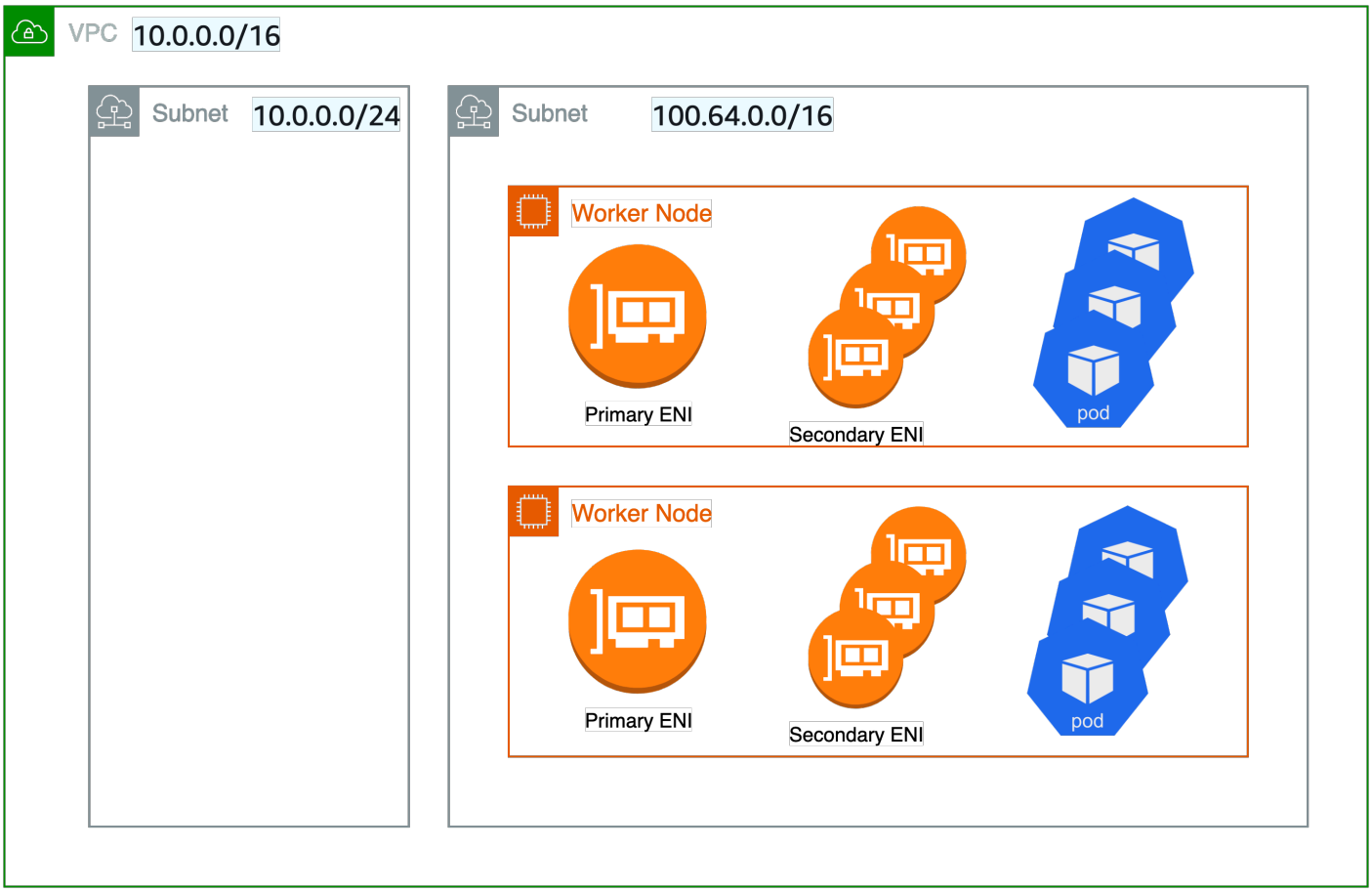
L'architettura di rete utilizzata nell'implementazione di questo post di blog segue le raccomandazioni contenute nella documentazione di Amazon VPC in Abilita la comunicazione tra reti sovrapposte. Come dimostrato in questo post del blog, puoi ampliare l'utilizzo del gateway NAT privato insieme RFC6598 agli indirizzi per gestire i problemi di esaurimento degli IP privati dei clienti. I cluster EKS, i nodi di lavoro, vengono distribuiti nell'intervallo CIDR secondario VPC 100.64.0.0/16 non routabile, mentre il gateway NAT privato, il gateway NAT, viene distribuito negli intervalli CIDR routabili. RFC1918 Il blog spiega come viene utilizzato un gateway di transito per la connessione al fine di facilitare la comunicazione tra intervalli CIDR sovrapposti non instradabili. VPCs VPCs Per i casi d'uso in cui le risorse EKS nell'intervallo di indirizzi non instradabili di un VPC devono comunicare con altre VPCs che non hanno intervalli di indirizzi sovrapposti, i clienti hanno la possibilità di utilizzare il peering VPC per interconnettere tali risorse. VPCs Questo metodo potrebbe fornire potenziali risparmi sui costi poiché tutto il transito di dati all'interno di una zona di disponibilità tramite una connessione peering VPC è ora gratuito.



Rete unica per nodi e pod

Se è necessario isolare i nodi e i pod su una rete specifica per motivi di sicurezza, si consiglia di distribuire nodi e pod in una sottorete da un blocco CIDR secondario più grande (ad esempio 100.64.0.0/8). Dopo l'installazione del nuovo CIDR nel tuo VPC, puoi distribuire un altro gruppo di nodi utilizzando il CIDR secondario e drenare i nodi originali per ridistribuire automaticamente i pod sui nuovi nodi di lavoro. Per ulteriori informazioni su come implementarlo, consulta questo post del blog.

La rete personalizzata non viene utilizzata nella configurazione rappresentata nel diagramma seguente. Piuttosto, i nodi di lavoro Kubernetes vengono distribuiti su sottoreti dell'intervallo CIDR VPC secondario del VPC, ad esempio 100.64.0.0/10. È possibile mantenere attivo il cluster EKS (il piano di controllo rimarrà sull'originale). subnet/s), but the nodes and Pods will be moved to a secondary subnet/s Questa è un'altra tecnica, anche se non convenzionale, per mitigare il pericolo di esaurimento dell'IP in un VPC. Proponiamo di svuotare i vecchi nodi prima di ridistribuire i pod sui nuovi nodi di lavoro.



Automatizza la configurazione con le etichette delle zone di disponibilità

Puoi consentire a Kubernetes di applicare automaticamente la zona di disponibilità (AZ) corrispondente ENIConfig per il nodo di lavoro.

Kubernetes aggiunge automaticamente il tag ai nodi di lavoro. `topology.kubernetes.io/zone` Amazon EKS consiglia di utilizzare la zona di disponibilità come nome di configurazione ENI quando hai solo una sottorete secondaria (CIDR alternativo) per AZ. Puoi quindi impostare l'etichetta utilizzata per scoprire il nome di configurazione ENI. `topology.kubernetes.io/zone` Nota che il tag `failure-domain.beta.kubernetes.io/zone` è obsoleto e sostituito con il tag. `topology.kubernetes.io/zone`

1. Imposta il `name` campo sulla zona di disponibilità del tuo VPC.
2. Abilita la configurazione automatica tramite il seguente comando
3. Imposta l'etichetta di configurazione tramite il seguente comando

```
kubectl set env daemonset aws-node -n kube-system
 "AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG=true"
kubectl set env daemonset aws-node -n kube-system
 "ENI_CONFIG_LABEL_DEF=topology.kubernetes.io/zone"
```

Se si dispone di più sottoreti secondarie per zona di disponibilità, è necessario crearne una specifica. `ENI_CONFIG_LABEL_DEF` Potresti prendere in considerazione la configurazione `k8s.amazonaws.com/eniConfig`e l'etichettatura dei nodi con nomi EniConfig personalizzati, `ENI_CONFIG_LABEL_DEF` come e. `k8s.amazonaws.com/eniConfig=us-west-2a-subnet-1k8s.amazonaws.com/eniConfig=us-west-2a-subnet-2`

Sostituisci i pod durante la configurazione della rete secondaria

L'attivazione di reti personalizzate non modifica i nodi esistenti. La rete personalizzata è un'azione dirompente. Anziché effettuare una sostituzione progressiva di tutti i nodi di lavoro del cluster dopo aver abilitato la rete personalizzata, suggeriamo di aggiornare il CloudFormation modello AWS nella EKS Getting Started Guide con una risorsa personalizzata che richiami una funzione Lambda per aggiornare il `aws-node` Daemonset con la variabile di ambiente per abilitare il networking personalizzato prima del provisioning dei nodi di lavoro.

Se nel cluster erano presenti nodi con Pod in esecuzione prima di passare alla funzionalità di rete CNI personalizzata, è necessario isolare e drenare i nodi per spegnere correttamente i Pod e quindi terminare i nodi. Solo i nuovi nodi che corrispondono all' ENIConfig etichetta o alle annotazioni utilizzano reti personalizzate, pertanto ai Pod pianificati su questi nuovi nodi può essere assegnato un IP dal CIDR secondario.

Calcola il numero massimo di pod per nodo

Poiché l'ENI principale del nodo non viene più utilizzato per assegnare gli indirizzi IP dei Pod, si verifica una diminuzione del numero di Pod che è possibile eseguire su un determinato tipo di istanza EC2. Per ovviare a questa limitazione, puoi utilizzare l'assegnazione dei prefissi con reti personalizzate. Con l'assegnazione del prefisso, ogni IP secondario viene sostituito con un prefisso /28 sul secondario. ENIs

Considerate il numero massimo di Pod per un'istanza m5.large con rete personalizzata.

Il numero massimo di Pod che è possibile eseguire senza l'assegnazione di prefissi è 29

- `3 ENIs - 1) * (10 secondary IPs per ENI - 1 + 2 = 20`

L'abilitazione degli allegati con prefisso aumenta il numero di Pod a 290.

- `(3 ENIs - 1) * ((10 secondary IPs per ENI - 1) * 16 + 2 = 290`

Tuttavia, suggeriamo di impostare il numero massimo di pod su 110 anziché 290 perché l'istanza ha un numero piuttosto ridotto di virtual. CPUs Nelle istanze più grandi, EKS consiglia un valore massimo di 250 pod. Quando si utilizzano allegati di prefisso con tipi di istanze più piccoli (ad esempio m5.large), è possibile che si esauriscano le risorse di CPU e memoria dell'istanza ben prima degli indirizzi IP.

> **Note**
>
> Quando il prefisso CNI assegna un prefisso /28 a un ENI, deve essere un blocco contiguo di indirizzi IP. Se la sottorete da cui viene generato il prefisso è altamente frammentata, l'allegato del prefisso potrebbe non riuscire. È possibile evitare che ciò accada creando un nuovo VPC dedicato per il cluster o riservando alla sottorete un set di CIDR esclusivamente per gli allegati del prefisso. Visita Subnet CIDR reservations per ulteriori informazioni su questo argomento.

Identifica l'uso esistente dello spazio CG-NAT

La rete personalizzata consente di mitigare il problema dell'esaurimento dell'IP, tuttavia non può risolvere tutte le sfide. Se utilizzi già lo spazio CG-NAT per il tuo cluster o semplicemente non hai la possibilità di associare un CIDR secondario al tuo VPC del cluster, ti suggeriamo di esplorare altre opzioni, come utilizzare un CNI alternativo o passare ai cluster. IPv6

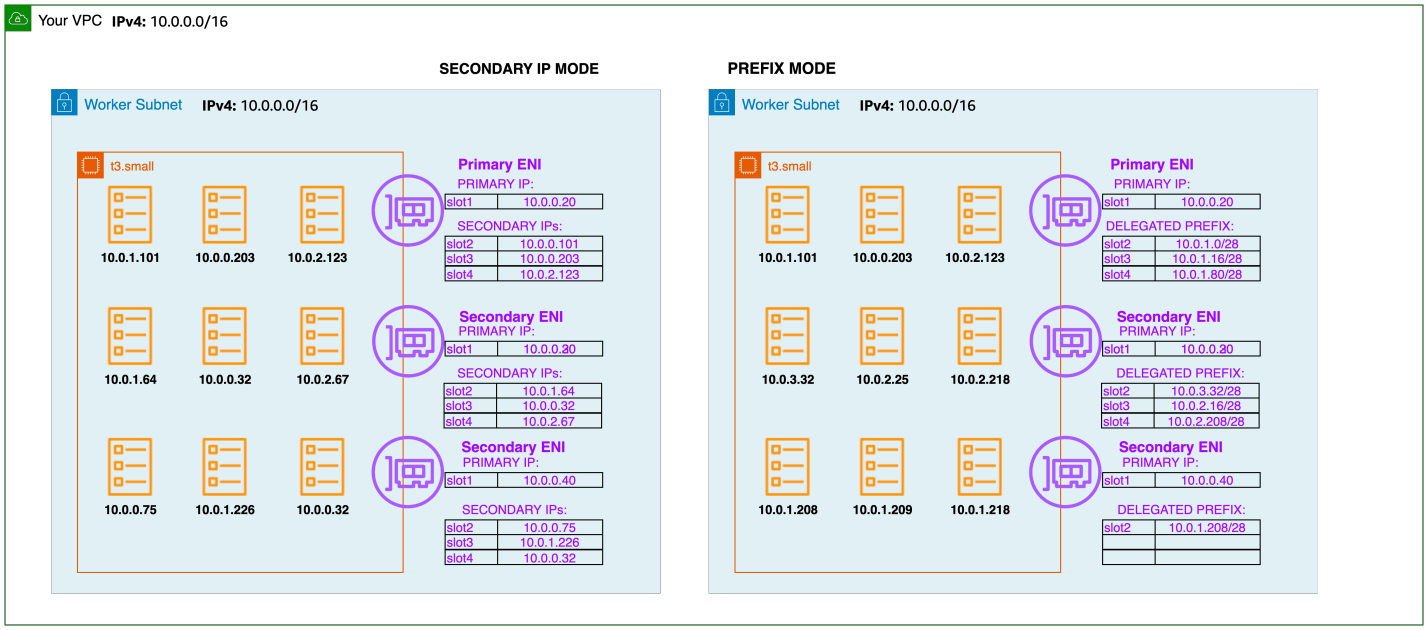
Modalità prefisso per Linux

> **Tip**
>
> Esplora le best practice tramite i workshop Amazon EKS.

Amazon VPC CNI assegna prefissi di rete alle interfacce di rete Amazon EC2 per aumentare il numero di indirizzi IP disponibili per i nodi e aumentare la densità di pod per nodo. Puoi configurare la versione 1.9.0 o successiva del componente aggiuntivo Amazon VPC CNI per assegnare IPv4 e IPv6 CIDRs anziché assegnare singoli indirizzi IP secondari alle interfacce di rete.

La modalità prefisso è abilitata per impostazione predefinita sui cluster ed è l'unica opzione supportata. IPv6 Il VPC CNI assegna un IPv6 prefisso /80 a uno slot su un ENI. Per ulteriori informazioni, consulta la IPv6 sezione di questa guida.

Con la modalità di assegnazione dei prefissi, il numero massimo di interfacce di rete elastiche per tipo di istanza rimane lo stesso, ma ora puoi configurare Amazon VPC CNI per assegnare prefissi di indirizzi /28 (16 indirizzi IP), anziché assegnare singoli IPv4 indirizzi agli slot sulle interfacce di rete. IPv4 Quando `ENABLE_PREFIX_DELEGATION` è impostato su true VPC, CNI alloca un indirizzo IP a un Pod dal prefisso assegnato a un ENI. Segui le istruzioni riportate nella guida per l'utente EKS per abilitare la modalità Prefix IP.

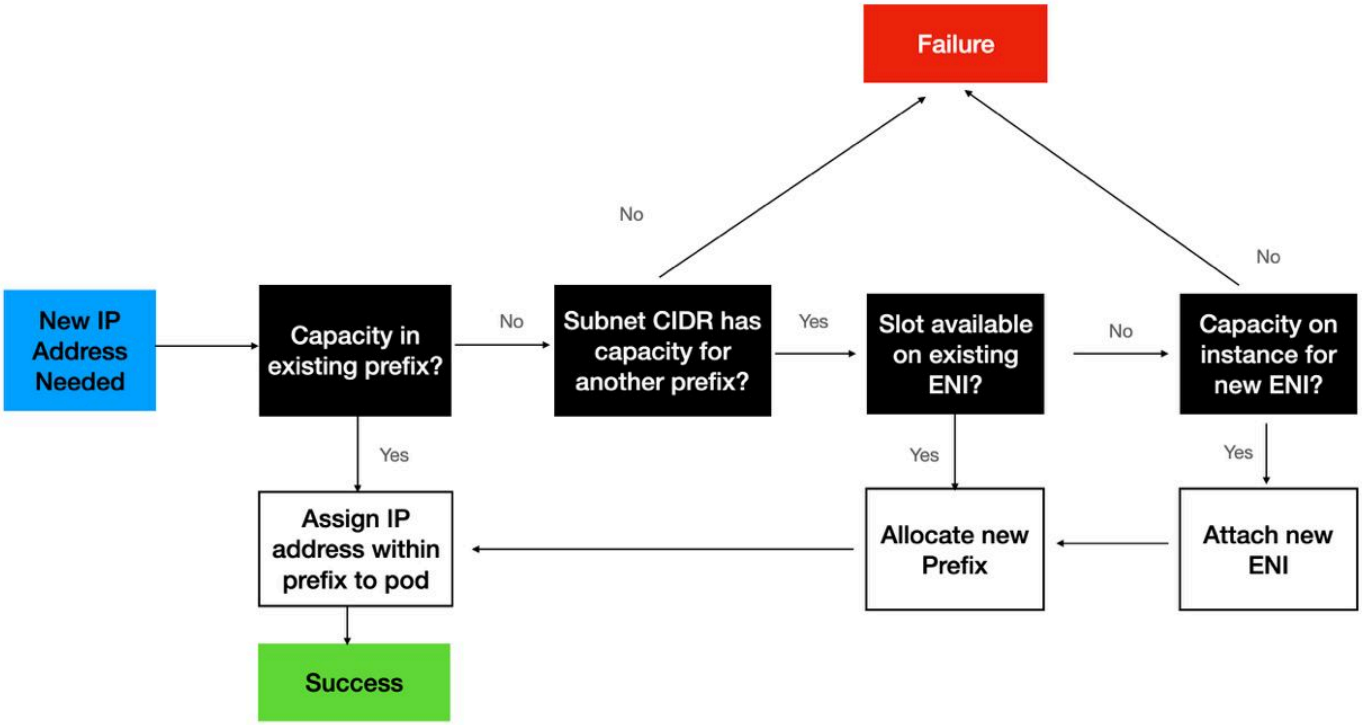


Il numero massimo di indirizzi IP che è possibile assegnare a un'interfaccia di rete dipende dal tipo di istanza. Ogni prefisso assegnato a un'interfaccia di rete conta come un unico indirizzo IP. Ad esempio, un'istanza `c5.large` ha un limite di `10` indirizzi IPv4 per interfaccia di rete. Ogni interfaccia di rete per questa istanza ha un indirizzo IPv4 primario. Se un'interfaccia di rete non ha indirizzi IPv4 secondari, è possibile assegnare fino a 9 prefissi all'interfaccia di rete. Per ogni indirizzo IPv4 aggiuntivo assegnato a un'interfaccia di rete, a questa è possibile assegnare un prefisso in meno. Consulta la documentazione di AWS EC2 sugli indirizzi IP per interfaccia di rete per tipo di istanza e sull'assegnazione di prefissi alle interfacce di rete.

Durante l'inizializzazione del nodo di lavoro, il VPC CNI assegna uno o più prefissi all'ENI primario. Il CNI preassegna un prefisso per un avvio più rapido dei pod mantenendo una piscina calda. Il numero di prefissi da conservare nella piscina calda può essere controllato impostando le variabili di ambiente.

- `WARM_PREFIX_TARGET`, il numero di prefissi da assegnare in eccesso rispetto al fabbisogno attuale.
- `WARM_IP_TARGET`, il numero di indirizzi IP da assegnare in eccesso rispetto al fabbisogno corrente.
- `MINIMUM_IP_TARGET`, il numero minimo di indirizzi IP da rendere disponibili in qualsiasi momento.
- `WARM_IP_TARGET`e `MINIMUM_IP_TARGET` se impostato avrà la precedenza`WARM_PREFIX_TARGET`.

Man mano che verranno programmati altri Pod, verranno richiesti prefissi aggiuntivi per l'ENI esistente. Innanzitutto, il VPC CNI tenta di assegnare un nuovo prefisso a un ENI esistente. Se l'ENI è al massimo, il VPC CNI tenta di allocare un nuovo ENI al nodo. Il nuovo ENIs verrà allegato fino al raggiungimento del limite ENI massimo (definito dal tipo di istanza). Quando viene collegato un nuovo ENI, ipamd assegnerà uno o più prefissi necessari per mantenere l'impostazione`WARM_PREFIX_TARGET`, `WARM_IP_TARGET` and. `MINIMUM_IP_TARGET`



Raccomandazioni

Usa la modalità Prefisso quando

Usa la modalità prefisso se riscontri problemi di densità dei Pod sui nodi di lavoro. Per evitare errori VPC CNI, consigliamo di esaminare le sottoreti alla ricerca di blocchi di indirizzi contigui per il prefisso /28 prima di migrare alla modalità prefisso. Consulta la sezione "Use Subnet Reservations to Avoid Subnet Fragmentation ()" per i dettagli sulla prenotazione delle sottoreti. IPv4

Per motivi di compatibilità con le versioni precedenti, il limite massimo di pods è impostato per supportare la modalità IP secondaria. Per aumentare la densità dei pod, specifica il `max-pods` valore in Kubelet e `--use-max-pods=false` come dati utente per i nodi. Potresti prendere in considerazione l'utilizzo dello max-pod-calculatorscript.sh per calcolare il numero massimo di pod consigliato da EKS per un determinato tipo di istanza. Fate riferimento alla guida per l'utente EKS per esempi di dati utente.

```
./max-pods-calculator.sh --instance-type m5.large --cni-version ``1.9``.0 --cni-prefix-delegation-enabled
```

La modalità di assegnazione dei prefissi è particolarmente importante per gli utenti delle reti personalizzate CNI in cui l'ENI principale non viene utilizzato per i pod. Con l'assegnazione dei prefissi, puoi comunque allegarne altri IPs su quasi tutti i tipi di istanza Nitro, anche senza l'ENI principale utilizzato per i pod.

Evita la modalità Prefisso quando

Se la sottorete è molto frammentata e non dispone di indirizzi IP disponibili sufficienti per creare prefissi /28, evitate di utilizzare la modalità prefisso. L'allegato del prefisso potrebbe non riuscire se la sottorete da cui viene prodotto il prefisso è frammentata (una sottorete molto utilizzata con indirizzi IP secondari sparsi). Questo problema può essere evitato creando una nuova sottorete e riservando un prefisso.

In modalità prefisso, il gruppo di sicurezza assegnato ai nodi di lavoro è condiviso dai Pod. Prendi in considerazione l'utilizzo dei gruppi di sicurezza per i pod se hai un requisito di sicurezza per raggiungere la conformità eseguendo applicazioni con requisiti di sicurezza di rete diversi su risorse di elaborazione condivise.

Usa tipi di istanze simili nello stesso gruppo di nodi

Il tuo gruppo di nodi può contenere istanze di molti tipi. Se un'istanza ha un numero massimo di pod basso, tale valore viene applicato a tutti i nodi del gruppo di nodi. Prendi in considerazione l'utilizzo di tipi di istanze simili in un gruppo di nodi per massimizzare l'utilizzo dei nodi. Ti consigliamo di configurare node.kubernetes.io/instance-type nella parte dei requisiti dell'API provisioner se utilizzi Karpenter per il ridimensionamento automatico dei nodi.

> **Warning**
>
> Il numero massimo di pod per tutti i nodi in un particolare gruppo di nodi è definito dal numero massimo di pod più basso di ogni singolo tipo di istanza nel gruppo di nodi.

Configura **`WARM_PREFIX_TARGET`** per conservare gli indirizzi IPv4

Il valore predefinito del manifesto di installazione per `WARM_PREFIX_TARGET` è 1. Nella maggior parte dei casi, il valore consigliato di 1 for `WARM_PREFIX_TARGET` fornirà una buona combinazione di tempi di avvio rapidi dei pod, riducendo al minimo gli indirizzi IP inutilizzati assegnati all'istanza.

Se è necessario conservare ulteriormente IPv4 gli indirizzi per nodo, l'utilizzo `WARM_IP_TARGET` e `MINIMUM_IP_TARGET` le impostazioni, che hanno la precedenza `WARM_PREFIX_TARGET` quando configurate. Impostando un valore inferiore `WARM_IP_TARGET` a 16, è possibile impedire al CNI di mantenere associato un intero prefisso in eccesso.

Preferisci assegnare nuovi prefissi piuttosto che allegare un nuovo ENI

L'assegnazione di un prefisso aggiuntivo a un ENI esistente è un'operazione API EC2 più rapida rispetto alla creazione e al collegamento di un nuovo ENI all'istanza. L'uso dei prefissi migliora le prestazioni pur essendo parsimonioso nell'allocazione degli indirizzi. IPv4 L'aggiunta di un prefisso in genere viene completata in meno di un secondo, mentre l'aggiunta di un nuovo ENI può richiedere fino a 10 secondi. Nella maggior parte dei casi d'uso, il CNI avrà bisogno di un solo ENI per nodo di lavoro quando viene eseguito in modalità prefisso. Se puoi permetterti (nel peggiore dei casi) fino a 15 unità inutilizzate IPs per nodo, ti consigliamo vivamente di utilizzare la nuova modalità di rete con assegnazione dei prefissi e di sfruttare i vantaggi in termini di prestazioni ed efficienza che ne derivano.

Usa Subnet Reservations per evitare la frammentazione della sottorete () IPv4

Quando EC2 assegna un IPv4 prefisso /28 a un ENI, deve trattarsi di un blocco contiguo di indirizzi IP provenienti dalla sottorete. Se la sottorete da cui viene generato il prefisso è frammentata (una sottorete molto utilizzata con indirizzi IP secondari sparsi), l'allegato del prefisso potrebbe fallire e nei log VPC CNI verrà visualizzato il seguente messaggio di errore:

```
failed to allocate a private IP/Prefix address: InsufficientCidrBlocks: There are not
 enough free cidr blocks in the specified subnet to satisfy the request.
```

Per evitare la frammentazione e disporre di spazio contiguo sufficiente per la creazione di prefissi, puoi utilizzare le prenotazioni CIDR della sottorete VPC per riservare lo spazio IP all'interno di una sottorete per l'uso esclusivo dei prefissi. Dopo aver creato una prenotazione, il plug-in VPC CNI chiamerà EC2 APIs per assegnare i prefissi che vengono allocati automaticamente dallo spazio riservato.

Si consiglia di creare una nuova sottorete, riservare spazio per i prefissi e abilitare l'assegnazione dei prefissi con VPC CNI per i nodi di lavoro in esecuzione in quella sottorete. Se la nuova sottorete è dedicata solo ai Pod in esecuzione nel cluster EKS con l'assegnazione del prefisso VPC CNI abilitata, puoi saltare la fase di prenotazione del prefisso.

Evita il downgrade di VPC CNI

La modalità Prefix funziona con VPC CNI versione 1.9.0 e successive. Il downgrade del componente aggiuntivo Amazon VPC CNI a una versione precedente alla 1.9.0 deve essere evitato una volta abilitata la modalità prefisso e assegnato i prefissi. ENIs È necessario eliminare e ricreare i nodi se si decide di effettuare il downgrade del VPC CNI.

Sostituisci tutti i nodi durante la transizione a Prefix Delegation

Si consiglia vivamente di creare nuovi gruppi di nodi per aumentare il numero di indirizzi IP disponibili anziché sostituire progressivamente i nodi di lavoro esistenti. Cordona e prosciuga tutti i nodi esistenti per rimuovere in sicurezza tutti i Pod esistenti. Per evitare interruzioni del servizio, suggeriamo di implementare Pod Disruption Budgets nei cluster di produzione per carichi di lavoro critici. Ai pod sui nuovi nodi verrà assegnato un IP da un prefisso assegnato a un ENI. Dopo aver confermato che i Pod sono in esecuzione, puoi eliminare i vecchi nodi e gruppi di nodi. Se utilizzi gruppi di nodi gestiti, segui i passaggi indicati qui per eliminare in sicurezza un gruppo di nodi.

Modalità prefisso per Windows

In Amazon EKS, a ogni Pod eseguito su un host Windows viene assegnato un indirizzo IP secondario dal controller di risorse VPC per impostazione predefinita. Questo indirizzo IP è un indirizzo routabile VPC allocato dalla sottorete dell'host. Su Linux, ogni ENI collegato all'istanza dispone di più slot che possono essere popolati da un indirizzo IP secondario o da un /28 CIDR (un prefisso). Gli host Windows, tuttavia, supportano solo un singolo ENI e i relativi slot disponibili. L'utilizzo solo di indirizzi IP secondari può limitare artificialmente il numero di pod eseguibili su un host Windows, anche quando è disponibile una grande quantità di indirizzi IP da assegnare.

Per aumentare la densità dei pod sugli host Windows, specialmente quando si utilizzano tipi di istanze più piccoli, è possibile abilitare i nodi Prefix Delegation for Windows. Quando la delega dei prefissi è abilitata, i IPv4 prefissi /28 vengono assegnati agli slot ENI anziché agli indirizzi IP secondari. La delega dei prefissi può essere abilitata aggiungendo la `enable-windows-prefix-delegation: "true"` voce alla mappa di configurazione. `amazon-vpc-cni` Questa è la stessa mappa di configurazione in cui è necessario impostare la `enable-windows-ipam: "true"` voce per abilitare il supporto di Windows.

Segui le istruzioni riportate nella guida per l'utente EKS per abilitare la modalità Prefix Delegation per i nodi Windows.

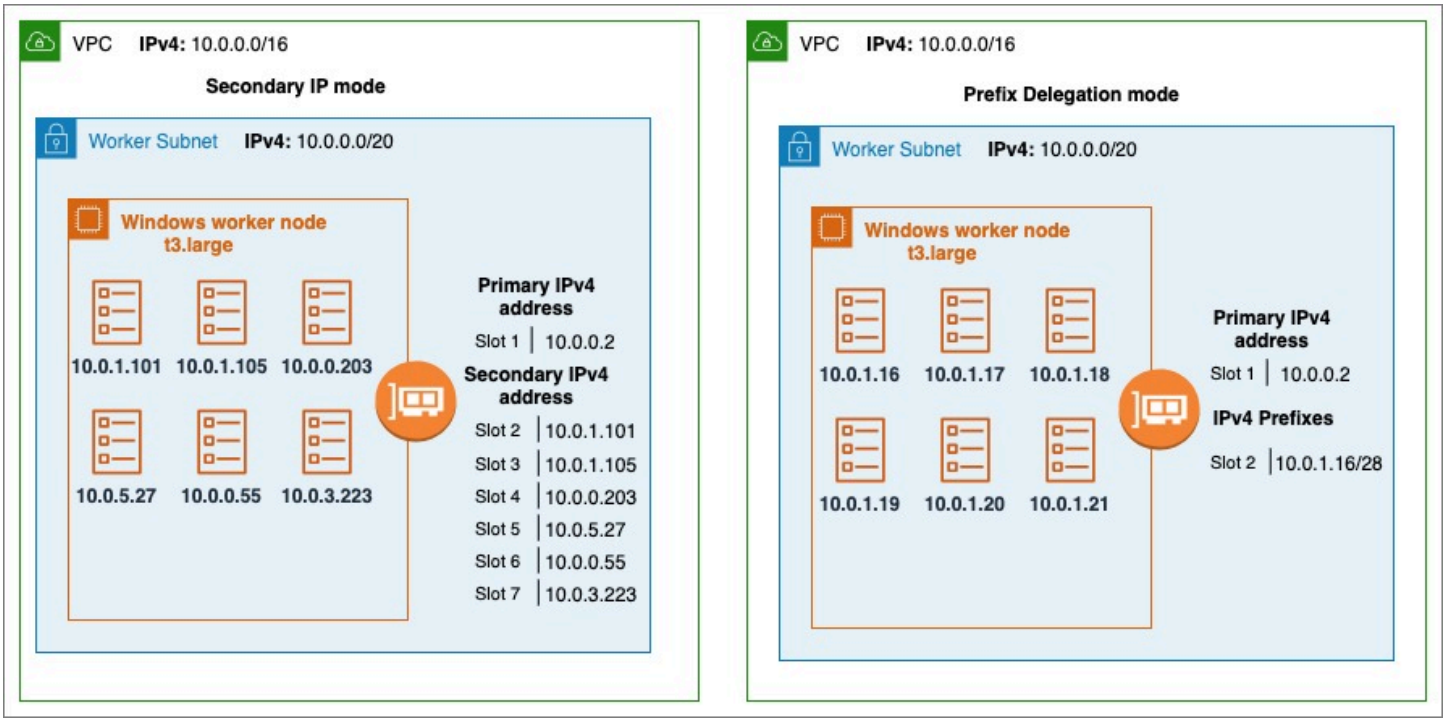


Figura: Confronto tra la modalità IP secondaria e la modalità Prefix Delegation

Il numero massimo di indirizzi IP che è possibile assegnare a un'interfaccia di rete dipende dal tipo di istanza e dalla sua dimensione. Ogni prefisso assegnato a un'interfaccia di rete utilizza uno slot disponibile. Ad esempio, un'`c5.large`istanza ha un limite di `10` slot per interfaccia di rete. Il primo slot su un'interfaccia di rete viene sempre utilizzato dall'indirizzo IP primario dell'interfaccia, lasciando all'utente 9 slot per i prefissi e gli indirizzi IP and/or secondari. Se a questi slot vengono assegnati prefissi, il nodo può supportare (9* 16) 144 indirizzi IP, mentre se vengono assegnati indirizzi IP secondari può supportare solo 9 indirizzi IP. Per ulteriori informazioni, consulta la documentazione sugli indirizzi IP per interfaccia di rete per tipo di istanza e sull'assegnazione di prefissi alle interfacce di rete.

Durante l'inizializzazione del nodo di lavoro, il VPC Resource Controller assegna uno o più prefissi all'ENI primario per un avvio più rapido del pod mantenendo un pool caldo di indirizzi IP. Il numero di prefissi da conservare nel pool caldo può essere controllato impostando i seguenti parametri di configurazione nella mappa di configurazione. `amazon-vpc-cni`

- `warm-prefix-target`, il numero di prefissi da assegnare in eccesso rispetto al fabbisogno attuale.
- `warm-ip-target`, il numero di indirizzi IP da assegnare in eccesso rispetto al fabbisogno corrente.
- `minimum-ip-target`, il numero minimo di indirizzi IP da rendere disponibili in qualsiasi momento.
- `warm-ip-target` and/or `minimum-ip-target`if set avrà la precedenza`warm-prefix-target`.

Man mano che sul nodo sono previsti più Pod, verranno richiesti prefissi aggiuntivi per l'ENI esistente. Quando un Pod è pianificato sul nodo, VPC Resource Controller tenta innanzitutto di assegnare un IPv4 indirizzo dai prefissi esistenti sul nodo. Se ciò non è possibile, verrà richiesto un nuovo IPv4 prefisso purché la sottorete abbia la capacità richiesta.

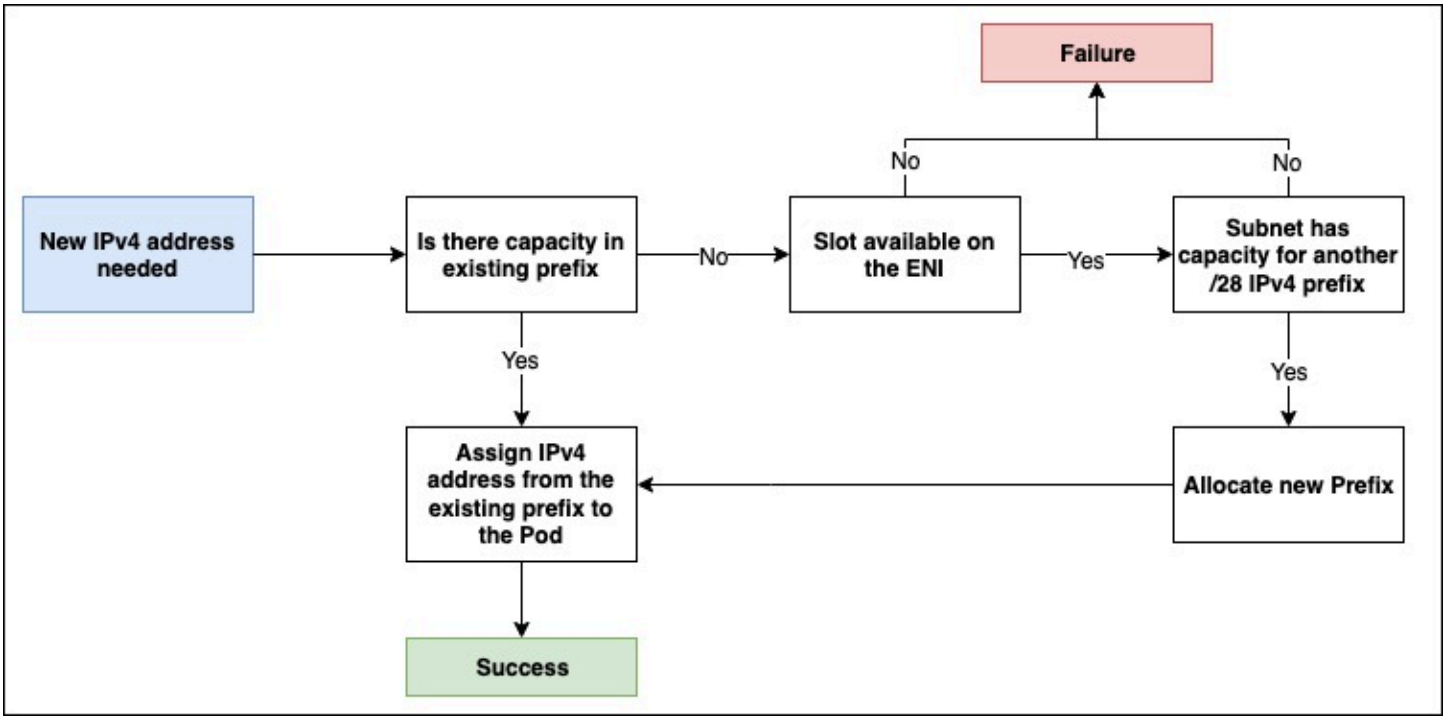


Figura: Flusso di lavoro durante l'assegnazione dell' IPv4 indirizzo al Pod

Raccomandazioni

Usa Prefix Delegation quando

Usa la delega del prefisso se riscontri problemi di densità dei Pod sui nodi di lavoro. Per evitare errori, consigliamo di esaminare le sottoreti alla ricerca di blocchi di indirizzi contigui per il prefisso /28 prima di migrare alla modalità prefisso. Consulta la sezione "Use Subnet Reservations to Avoid Subnet Fragmentation ()" per i dettagli relativi alla prenotazione delle sottoreti. IPv4

Per impostazione predefinita, i nodi `max-pods` su Windows sono impostati su. `110` Per la maggior parte dei tipi di istanze, questo dovrebbe essere sufficiente. Se desideri aumentare o diminuire questo limite, aggiungi quanto segue al comando bootstrap nei tuoi dati utente:

```
-KubeletExtraArgs '--max-pods=example-value'
```

Per maggiori dettagli sui parametri di configurazione bootstrap per i nodi Windows, consulta la documentazione qui.

Evita la delega del prefisso quando

Se la sottorete è molto frammentata e non dispone di indirizzi IP disponibili sufficienti per creare prefissi /28, evitate di utilizzare la modalità prefisso. L'allegato del prefisso potrebbe non riuscire se la sottorete da cui viene prodotto il prefisso è frammentata (una sottorete molto utilizzata con indirizzi IP secondari sparsi). Questo problema può essere evitato creando una nuova sottorete e riservando un prefisso.

Configura i parametri per la delega del prefisso per conservare gli indirizzi IPv4

`warm-prefix-targetwarm-ip-target`, e `minimum-ip-target` può essere utilizzato per ottimizzare il comportamento della prescalabilità e della scalabilità dinamica con prefissi. Per impostazione predefinita, vengono utilizzati i seguenti valori:

```
warm-ip-target: "1"
minimum-ip-target: "3"
```

Ottimizzando questi parametri di configurazione, è possibile ottenere un equilibrio ottimale tra la conservazione degli indirizzi IP e la riduzione della latenza del Pod dovuta all'assegnazione dell'indirizzo IP. Per ulteriori informazioni su questi parametri di configurazione, consulta la documentazione qui.

Utilizzate Subnet Reservations per evitare la frammentazione della sottorete () IPv4

Quando EC2 assegna un IPv4 prefisso /28 a un ENI, deve trattarsi di un blocco contiguo di indirizzi IP provenienti dalla sottorete. Se la sottorete da cui viene generato il prefisso è frammentata (una sottorete molto utilizzata con indirizzi IP secondari sparsi), l'allegato del prefisso potrebbe fallire e si verificherà il seguente evento del nodo:

```
InsufficientCidrBlocks: The specified subnet does not have enough free cidr blocks to
 satisfy the request
```

Per evitare la frammentazione e disporre di spazio contiguo sufficiente per la creazione di prefissi, utilizza le prenotazioni CIDR della sottorete VPC per riservare lo spazio IP all'interno di una sottorete per l'uso esclusivo dei prefissi. Una volta creata una prenotazione, gli indirizzi IP dei blocchi riservati non verranno assegnati ad altre risorse. In questo modo, VPC Resource Controller sarà in grado di ottenere i prefissi disponibili durante la chiamata di assegnazione al nodo ENI.

Si consiglia di creare una nuova sottorete, riservare spazio per i prefissi e abilitare l'assegnazione dei prefissi per i nodi di lavoro in esecuzione in quella sottorete. Se la nuova sottorete è dedicata

solo ai Pod in esecuzione nel cluster EKS con la delega dei prefissi abilitata, puoi saltare la fase di prenotazione del prefisso.

Sostituisci tutti i nodi durante la migrazione dalla modalità IP secondaria alla modalità Prefix Delegation o viceversa

Si consiglia vivamente di creare nuovi gruppi di nodi per aumentare il numero di indirizzi IP disponibili anziché sostituire in sequenza i nodi di lavoro esistenti.

Quando si utilizzano gruppi di nodi autogestiti, i passaggi per la transizione sarebbero i seguenti:

- Aumentate la capacità del cluster in modo che i nuovi nodi siano in grado di ospitare i vostri carichi di lavoro
- Abilita/disabilita la funzionalità di delega dei prefissi per Windows
- Cordona e svuota tutti i nodi esistenti per rimuovere in sicurezza tutti i tuoi Pod esistenti. Per evitare interruzioni del servizio, suggeriamo di implementare Pod Disruption Budgets nei cluster di produzione per carichi di lavoro critici.
- Dopo aver confermato che i Pod sono in esecuzione, puoi eliminare i vecchi nodi e gruppi di nodi. Ai pod sui nuovi nodi verrà assegnato un IPv4 indirizzo da un prefisso assegnato al nodo ENI.

Quando si utilizzano gruppi di nodi gestiti, i passaggi per la transizione sarebbero:

- Abilita/disabilita la funzionalità di delega dei prefissi per Windows
- Aggiorna il gruppo di nodi utilizzando i passaggi indicati qui. Questo esegue passaggi simili a quelli precedenti ma sono gestiti da EKS.

> **Warning**
>
> Esegui tutti i Pod su un nodo nella stessa modalità

Per Windows, ti consigliamo di evitare di eseguire i Pod sia in modalità IP secondaria che in modalità di delega del prefisso contemporaneamente. Una situazione del genere può verificarsi quando si passa dalla modalità IP secondaria alla modalità di delega con prefisso o viceversa con carichi di lavoro Windows in esecuzione.

Anche se ciò non influirà sui Pod in esecuzione, potrebbero esserci delle incongruenze per quanto riguarda la capacità dell'indirizzo IP del nodo. Ad esempio, consideriamo un nodo t3.xlarge con 14 slot per indirizzi secondari. IPv4 Se utilizzi 10 Pod, 10 slot sull'ENI verranno utilizzati da indirizzi IP secondari. Dopo aver abilitato la delega del prefisso, la capacità annunciata al server kube-api sarebbe (14 slot * 16 indirizzi IP per prefisso) 244, ma la capacità effettiva in quel momento sarebbe (4 slot rimanenti * 16 indirizzi per prefisso) 64. Questa incoerenza tra la quantità di capacità pubblicizzata e la quantità effettiva di capacità (slot rimanenti) può causare problemi se si utilizzano più Pod di quanti siano gli indirizzi IP disponibili per l'assegnazione.

Detto questo, puoi utilizzare la strategia di migrazione descritta sopra per trasferire in sicurezza i tuoi Pod dall'indirizzo IP secondario agli indirizzi ottenuti dai prefissi. Passando da una modalità all'altra, i Pod continueranno a funzionare normalmente e:

- Quando si passa dalla modalità IP secondaria alla modalità di delega del prefisso, gli indirizzi IP secondari assegnati ai pod in esecuzione non verranno rilasciati. I prefissi verranno assegnati agli slot liberi. Una volta terminato un pod, verranno rilasciati l'IP secondario e lo slot che stava utilizzando.
- Quando si passa dalla modalità di delega del prefisso alla modalità IP secondaria, viene rilasciato un prefisso quando tutti i prefissi IPs all'interno del relativo intervallo non sono più allocati ai pod. Se un IP del prefisso viene assegnato a un pod, tale prefisso verrà mantenuto fino alla chiusura dei pod.

Problemi di debug con Prefix Delegation

Puoi utilizzare la nostra guida al debug qui per approfondire il problema che stai riscontrando con la delega dei prefissi su Windows.

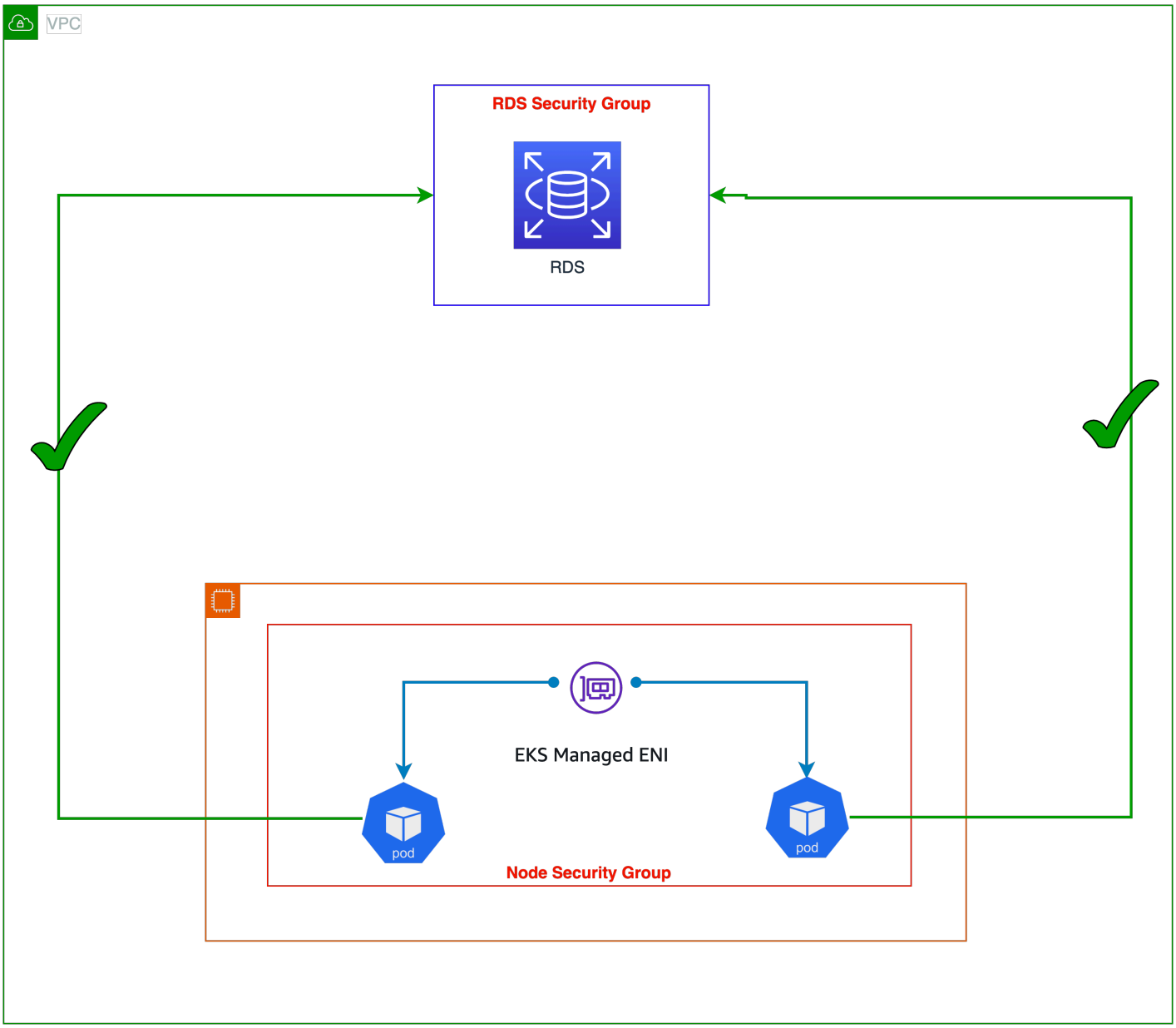
Gruppi di sicurezza per pod

> **Tip**
>
> Esplora le best practice tramite i workshop Amazon EKS.

Un gruppo di sicurezza AWS funge da firewall virtuale per le istanze EC2 per controllare il traffico in entrata e in uscita. Per impostazione predefinita, il CNI di Amazon VPC utilizzerà i gruppi di sicurezza associati all'ENI primario sul nodo. Più specificamente, ogni ENI associato all'istanza avrà gli stessi

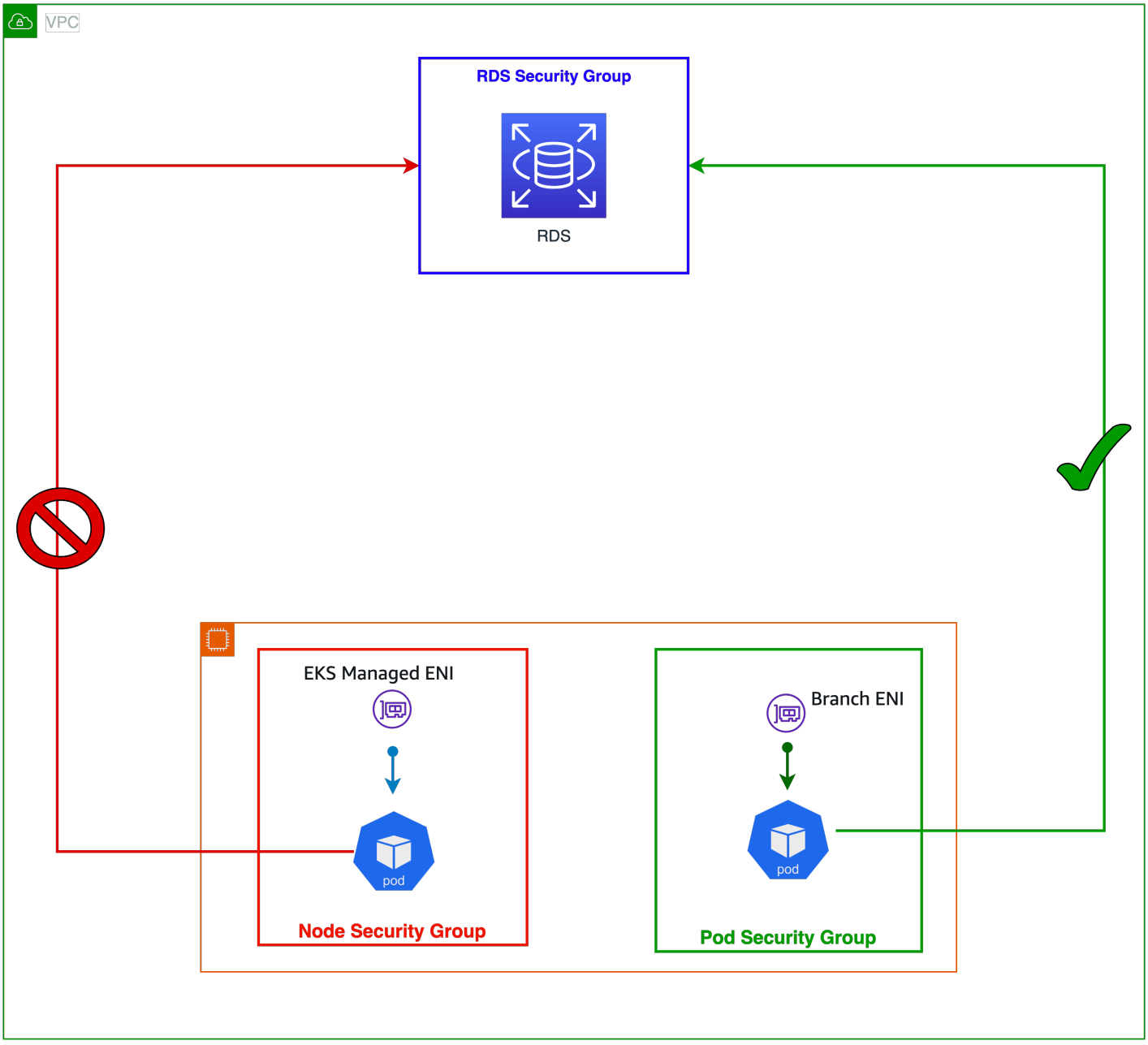
gruppi di sicurezza EC2. Pertanto, ogni Pod su un nodo condivide gli stessi gruppi di sicurezza del nodo su cui viene eseguito.

Come mostrato nell'immagine seguente, tutti i Pod delle applicazioni che operano sui nodi di lavoro avranno accesso al servizio di database RDS (considerando il gruppo di sicurezza RDS inbound allows node). I gruppi di sicurezza sono troppo generici perché si applicano a tutti i Pod in esecuzione su un nodo. I gruppi di sicurezza per Pods forniscono la segmentazione della rete per i carichi di lavoro, che è una parte essenziale di una buona strategia di difesa approfondita.



Con i gruppi di sicurezza per Pods, è possibile migliorare l'efficienza di elaborazione eseguendo applicazioni con requisiti di sicurezza di rete diversi su risorse di elaborazione condivise. Diversi tipi

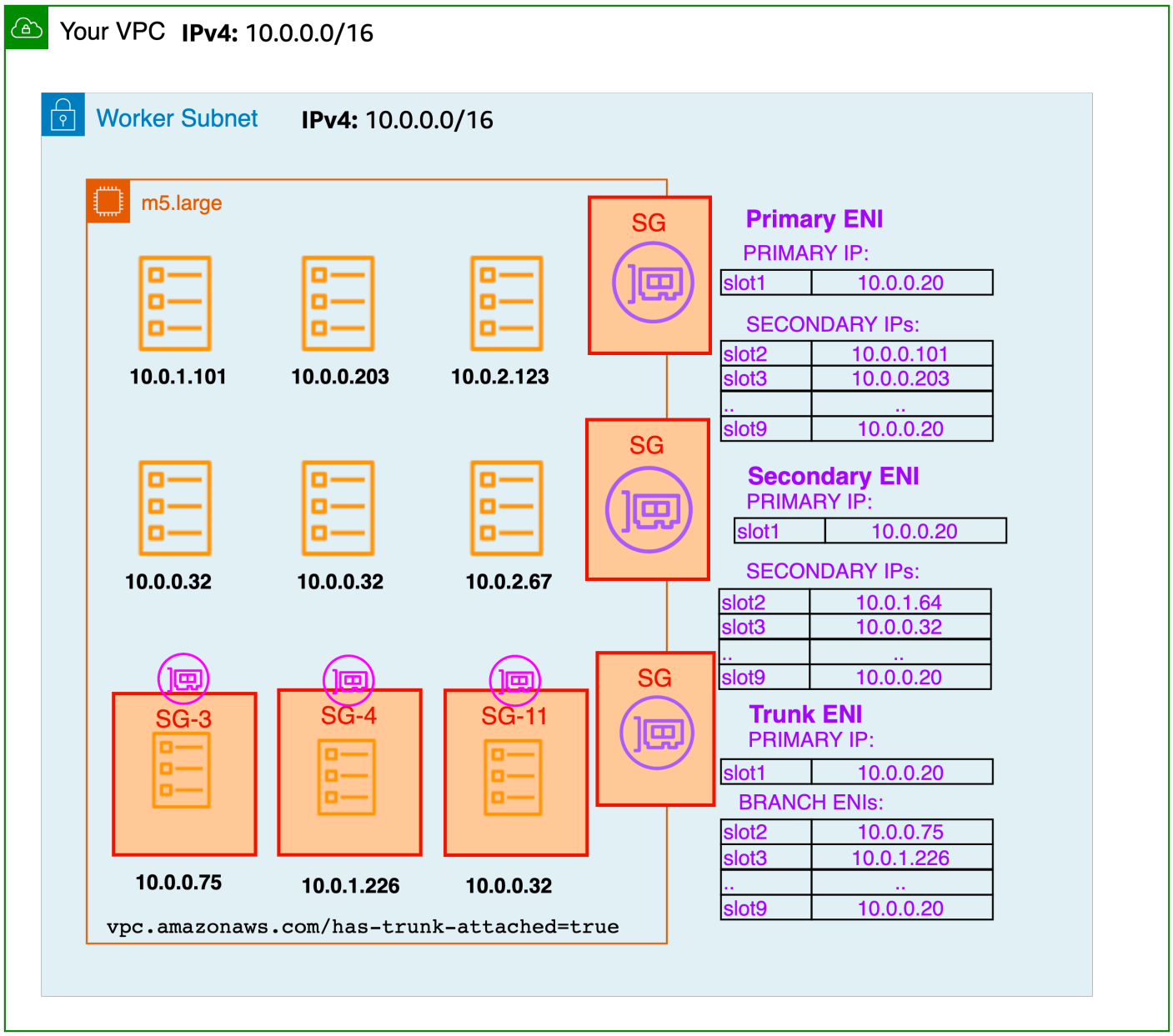
di regole di sicurezza, come Pod-to-Pod i servizi Pod-to-External AWS, possono essere definiti in un unico posto con i gruppi di sicurezza EC2 e applicati ai carichi di lavoro con Kubernetes native. APIs L'immagine seguente mostra i gruppi di sicurezza applicati a livello di Pod e come semplificano la distribuzione delle applicazioni e l'architettura dei nodi. Il Pod può ora accedere al database Amazon RDS.



È possibile abilitare i gruppi di sicurezza per i pod `ENABLE_POD_ENI=true` impostando VPC CNI. Una volta abilitato, il VPC Resource Controller in esecuzione sul piano di controllo (gestito da EKS) crea e collega un'interfaccia trunk chiamata «`aws-k8 «`s-trunk-enial nodo. L'interfaccia trunk funge

da interfaccia di rete standard collegata all'istanza. Per gestire le interfacce trunk, devi aggiungere la policy `AmazonEKSVPCResourceController` gestita al ruolo del cluster associato al tuo cluster Amazon EKS.

Il controller crea anche interfacce branch denominate «s-branch-eniaws-k8" e le associa all'interfaccia trunk. Ai pod viene assegnato un gruppo di sicurezza utilizzando la risorsa SecurityGroupPolicypersonalizzata e sono associati a un'interfaccia branch. Poiché i gruppi di sicurezza sono specificati con interfacce di rete, ora siamo in grado di pianificare i Pod che richiedono gruppi di sicurezza specifici su queste interfacce di rete aggiuntive. Consulta la sezione della Guida per l'utente di EKS sui gruppi di sicurezza per i pod, inclusi i prerequisiti di implementazione.



La capacità dell'interfaccia Branch si aggiunge ai limiti esistenti del tipo di istanza per gli indirizzi IP secondari. I pod che utilizzano gruppi di sicurezza non vengono presi in considerazione nella formula max-pods e quando si utilizza il gruppo di sicurezza per i pod è necessario considerare l'aumento del valore max-pods o accettare di utilizzare meno pod di quanti il nodo possa effettivamente supportare.

Un m5.large può avere fino a 9 interfacce di rete di filiali e fino a 27 indirizzi IP secondari assegnati alle sue interfacce di rete standard. Come mostrato nell'esempio seguente, il numero massimo di pod predefiniti per un m5.large è 29 ed EKS conta i Pod che utilizzano i gruppi di sicurezza fino al numero massimo di Pod. Consulta la guida per l'utente EKS per istruzioni su come modificare i max-pods per i nodi.

Quando i gruppi di sicurezza per Pods vengono utilizzati in combinazione con reti personalizzate, viene utilizzato il gruppo di sicurezza definito nei gruppi di sicurezza per Pods anziché il gruppo di sicurezza specificato in. ENIConfig Di conseguenza, quando la rete personalizzata è abilitata, valuta attentamente l'ordine dei gruppi di sicurezza mentre utilizzi i gruppi di sicurezza per Pod.

Raccomandazioni

Disattiva TCP Early Demux per Liveness Probe

Se utilizzi le sonde Liveness o Readiness, devi anche disabilitare TCP Early Demux, in modo che il kubelet possa connettersi ai Pod sulle interfacce di rete delle filiali tramite TCP. Questo è richiesto solo in modalità rigorosa. Per fare ciò esegui il seguente comando:

```
kubectl edit daemonset aws-node -n kube-system
```

Nella `initContainer` sezione, modificate il valore `DISABLE_TCP_EARLY_DEMUX` per `true`.

Usa Security Group For Pods per sfruttare gli investimenti di configurazione AWS esistenti.

I gruppi di sicurezza semplificano la limitazione dell'accesso alla rete alle risorse VPC, come i database RDS o le istanze EC2. Un chiaro vantaggio dei gruppi di sicurezza per Pod è l'opportunità di riutilizzare le risorse dei gruppi di sicurezza AWS esistenti. Se utilizzi gruppi di sicurezza come firewall di rete per limitare l'accesso ai tuoi servizi AWS, ti suggeriamo di applicare i gruppi di sicurezza ai Pod utilizzando ENI delle filiali. Prendi in considerazione l'utilizzo di gruppi di sicurezza per Pods se trasferisci app da istanze EC2 a EKS e limiti l'accesso ad altri servizi AWS con gruppi di sicurezza.

Configura la modalità di applicazione del gruppo di sicurezza di Pod

La versione 1.11 del plug-in Amazon VPC CNI ha aggiunto una nuova impostazione denominata `POD_SECURITY_GROUP_ENFORCING_MODE` («modalità di applicazione»). La modalità di applicazione controlla sia quali gruppi di sicurezza si applicano al pod sia se il NAT di origine è abilitato. È possibile specificare la modalità di applicazione come rigorosa o standard. L'impostazione predefinita è Strict, che riflette il comportamento precedente del CNI `ENABLE_POD_ENI` VPC impostato su. `true`

In modalità rigorosa, vengono applicati solo i gruppi di sicurezza ENI della filiale. Anche il NAT di origine è disabilitato.

In modalità Standard, vengono applicati i gruppi di sicurezza associati sia all'ENI principale che alla filiale ENI (associata al pod). Il traffico di rete deve essere conforme a entrambi i gruppi di sicurezza.

> **Warning**
>
> Qualsiasi modifica alla modalità influirà solo sui Pod appena lanciati. I Pod esistenti utilizzeranno la modalità configurata al momento della creazione del Pod. I clienti dovranno riciclare i Pod esistenti con gruppi di sicurezza se vogliono modificare il comportamento del traffico.

Modalità di applicazione: utilizza la modalità rigorosa per isolare il traffico di pod e nodi:

Per impostazione predefinita, i gruppi di sicurezza per Pods sono impostati sulla «modalità rigorosa». Usa questa impostazione se devi separare completamente il traffico Pod dal resto del traffico del nodo. In modalità rigorosa, il NAT di origine è disattivato in modo da poter utilizzare i gruppi di sicurezza ENI in uscita della filiale.

> **Warning**
>
> Quando la modalità rigorosa è abilitata, tutto il traffico in uscita da un pod lascerà il nodo ed entrerà nella rete VPC. Il traffico tra i pod sullo stesso nodo passerà attraverso il VPC. Ciò aumenta il traffico VPC e limita le funzionalità basate sui nodi. non NodeLocal DNSCache è supportato con la modalità rigorosa.

Modalità di applicazione: utilizza la modalità Standard nelle seguenti situazioni

IP di origine del client visibile ai contenitori nel Pod

Se devi mantenere l'IP di origine del client visibile ai contenitori nel Pod, valuta la possibilità di `POD_SECURITY_GROUP_ENFORCING_MODE` impostare `sustandard`. I servizi Kubernetes support externalTrafficPolicy =local per supportare la conservazione dell'IP di origine del client (cluster di tipo predefinito). Ora puoi eseguire servizi Kubernetes di tipo NodePort e LoadBalancer utilizzando destinazioni di istanza con un valore externalTrafficPolicy impostato su Local in modalità standard. `Local`preserva l'IP di origine del client ed evita un secondo hop e type Services. LoadBalancer NodePort

Implementazione NodeLocal DNSCache

Quando utilizzi gruppi di sicurezza per i pod, configura la modalità standard per supportare i pod che utilizzano. NodeLocal DNSCache NodeLocal DNSCache migliora le prestazioni del DNS del cluster eseguendo un agente di caching DNS sui nodi del cluster come. DaemonSet Ciò aiuterà i pod con i requisiti DNS QPS più elevati a interrogare Kube-DNS/CoreDNS locali con una cache locale, il che migliorerà la latenza.

NodeLocal DNSCache non è supportato in modalità rigorosa poiché tutto il traffico di rete, anche verso il nodo, entra nel VPC.

Supporto della politica di rete Kubernetes

Consigliamo di utilizzare la modalità di applicazione standard quando si utilizzano criteri di rete con Pod a cui sono associati gruppi di sicurezza.

Consigliamo vivamente di utilizzare gruppi di sicurezza per Pods per limitare l'accesso a livello di rete ai servizi AWS che non fanno parte di un cluster. Prendi in considerazione le politiche di rete per limitare il traffico di rete tra i Pod all'interno di un cluster, spesso noto come traffico. East/West

Identifica le incompatibilità con i gruppi di sicurezza per Pod

Le istanze basate su Windows e non nitro non supportano i gruppi di sicurezza per i pod.
Per utilizzare i gruppi di sicurezza con i pod, le istanze devono essere contrassegnate con. isTrunkingEnabled Utilizza le policy di rete per gestire l'accesso tra i Pod anziché i gruppi di sicurezza se i tuoi Pod non dipendono da alcun servizio AWS all'interno o all'esterno del tuo VPC.

Usa i gruppi di sicurezza per Pod per controllare in modo efficiente il traffico verso i servizi AWS

Se un'applicazione in esecuzione all'interno del cluster EKS deve comunicare con un'altra risorsa all'interno del VPC, ad esempio un database RDS, prendi in considerazione l'utilizzo SGs di for pods. Sebbene esistano motori di policy che consentono di specificare un nome CIDR o DNS, sono una scelta meno ottimale quando si comunica con servizi AWS che hanno endpoint che risiedono all'interno di un VPC.

Al contrario, le policy di rete di Kubernetes forniscono un meccanismo per controllare il traffico in ingresso e in uscita sia all'interno che all'esterno del cluster. Le policy di rete Kubernetes devono essere prese in considerazione se l'applicazione ha dipendenze limitate da altri servizi AWS. Puoi configurare policy di rete che specificano regole di uscita basate su intervalli CIDR per limitare l'accesso ai servizi AWS rispetto alla semantica nativa di AWS. SGs Puoi utilizzare le policy di rete di Kubernetes per controllare il traffico di rete tra i Pod (spesso denominato traffico) e tra i Pod e i servizi esterni East/West . Le politiche di rete Kubernetes sono implementate ai livelli OSI 3 e 4.

Amazon EKS consente di utilizzare motori di policy di rete come Calico e Cilium. Per impostazione predefinita, i motori delle politiche di rete non sono installati. Consulta le rispettive guide all'installazione per istruzioni su come eseguire la configurazione. Per ulteriori informazioni su come utilizzare la politica di rete, consulta le migliori pratiche di sicurezza EKS. La funzionalità dei nomi host DNS è disponibile nelle versioni aziendali dei motori di policy di rete, che potrebbe essere utile per controllare il traffico tra Kubernetes Services/Pods e le risorse eseguite al di fuori di AWS. Inoltre, puoi prendere in considerazione il supporto dei nomi host DNS per i servizi AWS che per impostazione predefinita non supportano i gruppi di sicurezza.

Etichetta un singolo gruppo di sicurezza per utilizzare AWS Loadbalancer Controller

Quando molti gruppi di sicurezza sono assegnati a un Pod, Amazon EKS consiglia di contrassegnare un singolo gruppo di sicurezza come `kubernetes.io/cluster/$name`condiviso o di proprietà. Il tag consente a AWS Loadbalancer Controller di aggiornare le regole dei gruppi di sicurezza per indirizzare il traffico verso i Pod. Se a un Pod viene assegnato un solo gruppo di sicurezza, l'assegnazione di un tag è facoltativa. Le autorizzazioni impostate in un gruppo di sicurezza si sommano, pertanto è sufficiente etichettare un singolo gruppo di sicurezza per consentire al controller del loadbalancer di individuare e riconciliare le regole. Inoltre aiuta a rispettare le quote predefinite definite dai gruppi di sicurezza.

Configura NAT per il traffico in uscita

Il NAT di origine è disabilitato per il traffico in uscita dai pod a cui sono assegnati gruppi di sicurezza. Per i Pod che utilizzano gruppi di sicurezza che richiedono l'accesso a Internet, avviate i nodi di lavoro su sottoreti private configurate con un gateway o un'istanza NAT e abilitate SNAT esterni nel CNI.

```
kubectl set env daemonset -n kube-system aws-node AWS_VPC_K8S_CNI_EXTERNALSNAT=true
```

Distribuisci i pod con gruppi di sicurezza in sottoreti private

I pod a cui sono assegnati gruppi di sicurezza devono essere eseguiti su nodi distribuiti su sottoreti private. Tieni presente che i pod con gruppi di sicurezza assegnati distribuiti su sottoreti pubbliche non saranno in grado di accedere a Internet.

Verifica i terminationGracePeriodsecondi nel file delle specifiche del pod

Assicurati che `terminationGracePeriodSeconds` sia diverso da zero nel file delle specifiche del Pod (impostazione predefinita: 30 secondi). Ciò è essenziale per consentire ad Amazon VPC CNI di eliminare la rete Pod dal nodo di lavoro. Se impostato su zero, il plug-in CNI non rimuove la rete Pod dall'host e l'ENI della filiale non viene ripulito in modo efficace.

Utilizzo dei gruppi di sicurezza per i pod con Fargate

I gruppi di sicurezza per i Pod eseguiti su Fargate funzionano in modo molto simile ai Pod eseguiti sui nodi di lavoro EC2. Ad esempio, è necessario creare il gruppo di sicurezza prima di SecurityGroupPolicy farvi riferimento nel file associato al Pod Fargate. Per impostazione predefinita, il gruppo di sicurezza del cluster viene assegnato a tutti i Fargate Pod quando non si assegna esplicitamente un Fargate Pod a un Pod Fargate. SecurityGroupPolicy Per semplicità, potresti voler aggiungere il gruppo di sicurezza del cluster a un Fagate Pod, SecurityGroupPolicy altrimenti dovrai aggiungere le regole minime di sicurezza del gruppo al tuo gruppo di sicurezza. È possibile trovare il gruppo di sicurezza del cluster utilizzando l'API describe-cluster.

```
 aws eks describe-cluster --name CLUSTER_NAME --query
 'cluster.resourcesVpcConfig.clusterSecurityGroupId'
```

```
cat >my-fargate-sg-policy.yaml <<EOF
apiVersion: vpcresources.k8s.aws/v1beta1
```

```
kind: SecurityGroupPolicy
metadata:
  name: my-fargate-sg-policy
  namespace: my-fargate-namespace
spec:
  podSelector:
    matchLabels:
      role: my-fargate-role
  securityGroups:
    groupIds:
      - cluster_security_group_id
      - my_fargate_pod_security_group_id
EOF
```

Le regole del gruppo di sicurezza minimo sono elencate qui. Queste regole consentono ai Fargate Pod di comunicare con servizi interni al cluster come kube-apiserver, kubelet e CoredNS. È inoltre necessario aggiungere regole per consentire le connessioni in entrata e in uscita da e verso il Fargate Pod. Ciò consentirà al tuo Pod di comunicare con altri Pod o risorse nel tuo VPC. Inoltre, è necessario includere regole che consentano a Fargate di estrarre le immagini dei container da Amazon ECR o da altri registri di container come. DockerHub Per ulteriori informazioni, consulta gli intervalli di indirizzi IP di AWS nell'AWS General Reference.

È possibile utilizzare i comandi seguenti per trovare i gruppi di sicurezza applicati a un Pod Fargate.

```
kubectl get pod FARGATE_POD -o jsonpath='{.metadata.annotations.fargate\.amazonaws
\.com/pod-sg}{"\n"}'
```

Annota il comando ENIid riportato sopra.

```
aws ec2 describe-network-interfaces --network-interface-ids ENI_ID --query
 'NetworkInterfaces[*].Groups[*]'
```

I pod Fargate esistenti devono essere eliminati e ricreati per poter applicare nuovi gruppi di sicurezza. Ad esempio, il comando seguente avvia la distribuzione dell'app di esempio. Per aggiornare pod specifici, puoi modificare lo spazio dei nomi e il nome della distribuzione nel comando seguente.

```
kubectl rollout restart -n example-ns deployment example-pod
```

Bilanciamento del carico

> **Tip**
>
> Esplora le best practice tramite i workshop Amazon EKS.

I Load Balancer ricevono il traffico in entrata e lo distribuiscono tra le destinazioni dell'applicazione prevista ospitata in un cluster EKS. Ciò migliora la resilienza dell'applicazione. Se distribuito in un cluster EKS, il controller AWS Load Balancer creerà e gestirà AWS Elastic Load Balancer per quel cluster. Quando LoadBalancer viene creato un servizio Kubernetes di tipo Kubernetes, il controller AWS Load Balancer crea un Network Load Balancer (NLB) che bilancia il carico del traffico ricevuto al livello 4 del modello OSI. Quando viene creato un oggetto Kubernetes Ingress, il controller AWS Load Balancer crea un Application Load Balancer (ALB) che bilancia il carico al livello 7 del modello OSI.

Scelta del tipo di Load Balancer

Il portafoglio AWS Elastic Load Balancing (ELB) supporta i seguenti sistemi di bilanciamento del carico: Application Load Balancer (ALB), Network Load Balancer (NLB), Gateway Load Balancers (GWLB) e Classic Load Balancers (CLB). Questa sezione sulle best practice si concentrerà su ALB e NLB, i due più importanti per i cluster EKS.

La considerazione principale nella scelta del tipo di bilanciamento del carico sono i requisiti del carico di lavoro.

Per informazioni più dettagliate e come riferimento per tutti i sistemi di bilanciamento del carico AWS, consulta la pagina Confronti di prodotti

Scegli Application Load Balancer (ALB) se il tuo carico di lavoro è HTTP/HTTPS

Se un carico di lavoro richiede il bilanciamento del carico al livello 7 del modello OSI, è possibile utilizzare il controller AWS Load Balancer per fornire un ALB; tratteremo il provisioning nella sezione seguente. L'ALB è controllato e configurato dalla risorsa Ingress menzionata in precedenza e indirizza il traffico HTTP o HTTPS verso diversi Pod all'interno del cluster. L'ALB offre ai clienti la flessibilità necessaria per modificare l'algoritmo di routing del traffico delle applicazioni; l'algoritmo di routing predefinito è round robin, ma in alternativa è disponibile anche l'algoritmo di routing delle richieste meno in sospeso.

Scegli Network Load Balancer (NLB) se il tuo carico di lavoro è TCP o se il tuo carico di lavoro richiede la conservazione dell'IP di origine dei client

Un Network Load Balancer funziona al quarto livello (Transport) del modello Open Systems Interconnection (OSI). È adatto per carichi di lavoro basati su TCP e UDP. Inoltre, per impostazione predefinita, Network Load Balancer conserva l'IP di origine dell'indirizzo dei client quando presenta il traffico al pod.

Scegli Network Load Balancer (NLB) se il tuo carico di lavoro non può utilizzare il DNS

Un altro motivo fondamentale per utilizzare l'NLB è se i tuoi client non possono utilizzare il DNS. In questo caso, l'NLB potrebbe essere più adatto al tuo carico di lavoro in quanto i Network Load IPs Balancer sono statici. Sebbene si consiglia ai client di utilizzare il DNS per la risoluzione dei nomi di dominio in indirizzi IP durante la connessione a Load Balancer, se l'applicazione di un client non supporta la risoluzione DNS e accetta solo codici rigidi IPs , un NLB è la soluzione migliore in quanto IPs sono statici e rimangono invariati per tutta la vita del NLB.

Fornitura di sistemi di bilanciamento del carico

Dopo aver determinato il Load Balancer più adatto ai tuoi carichi di lavoro, i clienti hanno a disposizione diverse opzioni per il provisioning di un load balancer.

Effettua il provisioning di Load Balancer distribuendo il controller AWS Load Balancer

Esistono due metodi principali per il provisioning dei sistemi di bilanciamento del carico all'interno di un cluster EKS.

- Utilizzo del controller di servizio nel provider di cloud AWS (legacy)
- Utilizzo del controller AWS Load Balancer (consigliato)

Per impostazione predefinita, il Kubernetes Service Controller, noto anche come controller in-tree, riconcilia il tipo di risorsa del servizio Kubernetes. LoadBalancer Questo controller è integrato nel componente AWS Cloud Provider che funziona come Kubernetes Cloud Controller Manager.

La configurazione dell'Elastic Load Balancer fornito è controllata da annotazioni che devono essere aggiunte al manifesto del servizio Kubernetes. Le annotazioni utilizzate dal Service Controller e dal controller AWS Load Balancer sono diverse.

Il Service Controller è obsoleto e attualmente riceve solo correzioni di bug critici. Quando crei un servizio Kubernetes di tipo LoadBalancer, il Service Controller crea un CLB AWS per impostazione predefinita, ma può anche creare AWS NLB se usi l'annotazione corretta. Vale la pena notare che Service Controller non supporta le risorse Kubernetes Ingress e inoltre non supporta. IPv6

Ti consigliamo di utilizzare il controller AWS Load Balancer nei cluster EKS per riconciliare le risorse Kubernetes Service e Ingress. È necessario utilizzare le annotazioni corrette nel servizio Kubernetes o nel manifesto Ingress in modo che AWS Load Balancer Controller sia responsabile del processo di riconciliazione. (anziché Service Controller)

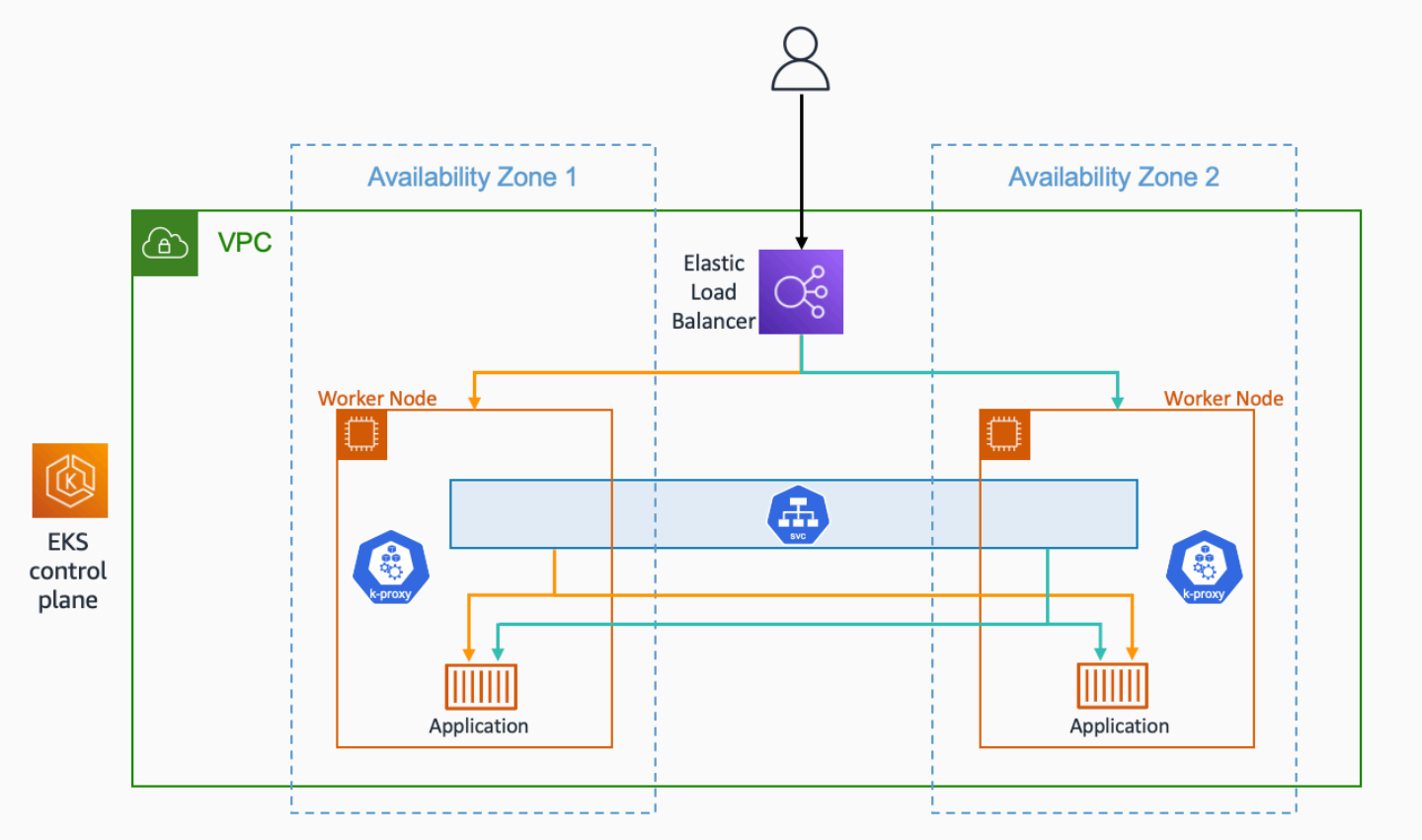
Se utilizzi la modalità automatica EKS, il controller AWS Load Balancer viene fornito automaticamente; non è necessaria alcuna installazione.

Scelta del tipo di destinazione Load Balancer

Registra i pod come destinazioni utilizzando IP Target-Type

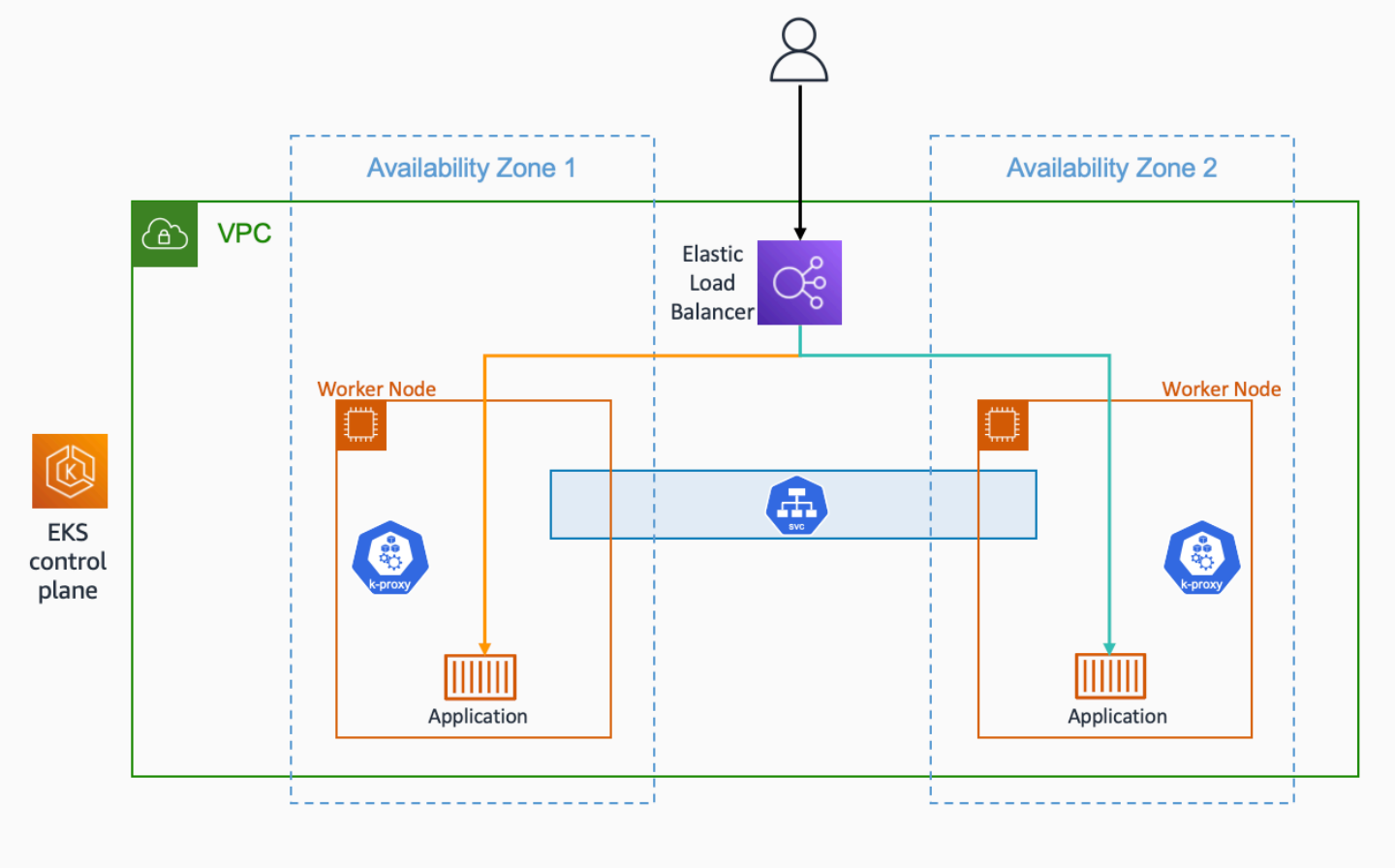
Un AWS Elastic Load Balancer: Network & Application invia il traffico ricevuto a destinazioni registrate in un gruppo target. Per un cluster EKS ci sono 2 tipi di target che è possibile registrare nel gruppo target: Instance e IP, il tipo di destinazione utilizzato ha implicazioni su ciò che viene registrato e su come il traffico viene instradato dal Load Balancer al pod. Per impostazione predefinita, il controller AWS Load Balancer registrerà gli obiettivi utilizzando il tipo «Instance» e questo target sarà l'IP del Worker Node eNodePort, di conseguenza, ciò include:

- Il traffico proveniente dal Load Balancer verrà inoltrato al Worker Node in data NodePort, questo viene elaborato dalle regole iptables (configurate da kube-proxy in esecuzione sul nodo) e inoltrato al Servizio sul suo ClusterIP (ancora sul nodo), infine il Servizio seleziona casualmente un pod registrato su di esso e inoltra il traffico ad esso. Questo flusso comporta più hop e può verificarsi una latenza aggiuntiva, soprattutto perché il Servizio a volte seleziona un pod in esecuzione su un altro nodo di lavoro che potrebbe trovarsi anche in un'altra AZ.
- Poiché il Load Balancer registra il Worker Node come destinazione, significa che il controllo di integrità inviato al target non verrà ricevuto direttamente dal pod ma dal Worker Node sul suo terminale NodePort e il traffico di controllo dello stato seguirà lo stesso percorso descritto sopra.
- Il monitoraggio e la risoluzione dei problemi sono più complessi poiché il traffico inoltrato dal Load Balancer non viene inviato direttamente ai pod e sarebbe necessario correlare attentamente il pacchetto ricevuto sul Worker Node al Service ClusterIP e infine al pod per avere end-to-end piena visibilità sul percorso del pacchetto per una corretta risoluzione dei problemi.



Al contrario, se si configura il tipo di destinazione come «IP» come consigliato, l'implicazione sarà la seguente:

- Il traffico proveniente dal Load Balancer verrà inoltrato direttamente al pod, questo semplifica il percorso di rete in quanto aggira i precedenti hop aggiuntivi dei Worker Nodes e dell'IP del Service Cluster, riduce la latenza che altrimenti si sarebbe verificata se il Servizio inoltrasse il traffico a un pod in un'altra AZ e infine rimuove l'elaborazione dell'overhead delle regole iptables sui Worker Nodes.
- Il controllo dello stato di salute del Load Balancer viene ricevuto e risposto direttamente dal pod, ciò significa che lo stato target «sano» o «non integro» è una rappresentazione diretta dello stato di salute del pod.
- Il monitoraggio e la risoluzione dei problemi sono più semplici e qualsiasi strumento utilizzato per acquisire l'indirizzo IP del pacchetto rivelerà direttamente il traffico bidirezionale tra il Load Balancer e il pod nei campi di origine e destinazione.



Per creare un AWS Elastic Load Balancing che utilizza obiettivi IP aggiungi:

- `alb.ingress.kubernetes.io/target-type: ip`annotazione sul manifesto di Ingress durante la configurazione di Kubernetes Ingress (Application Load Balancer)
- `service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: ip`annotazione nel manifesto del servizio durante la configurazione del servizio Kubernetes di tipo (Network Load LoadBalancer Balancer).

Configurazione dei controlli dello stato di Load Balancer

Sebbene Kubernetes fornisca i propri meccanismi di controllo dello stato (descritti nella sezione successiva), consigliamo di implementare i controlli di integrità ELB come protezione complementare che funziona al di fuori del piano di controllo di Kubernetes. Questo livello indipendente continua a monitorare l'applicazione anche durante:

- Interruzioni del piano di controllo Kubernetes
- Ritardi di esecuzione delle sonde

- Partizioni di rete tra kubelet e pod

Per i carichi di lavoro critici che richiedono la massima disponibilità e ripristino accelerato negli scenari sopra menzionati, i controlli di integrità ELB forniscono una rete di sicurezza essenziale che funziona insieme, e non in sostituzione, ai meccanismi nativi di Kubernetes.

Per configurare e ottimizzare i controlli di integrità sul tuo ELB, devi utilizzare annotazioni nel tuo Kubernetes Service o Ingress manifest che verrebbero riconciliate da Service Controller o AWS Load Balancer Controller.

Disponibilità e ciclo di vita dei Pod

Durante l'aggiornamento di un'applicazione, è necessario assicurarsi che l'applicazione sia sempre disponibile per l'elaborazione delle richieste, in modo che gli utenti non subiscano interruzioni. Una sfida comune in questo scenario è la sincronizzazione dello stato di disponibilità dei carichi di lavoro tra il livello Kubernetes e l'infrastruttura, ad esempio Load Balancer esterni. Le sezioni successive evidenziano le migliori pratiche per affrontare tali scenari.

> **Note**
>
> Le spiegazioni seguenti si basano sul fatto che è EndpointSlicesil sostituto consigliato per gli endpoint in Kubernetes. Le differenze tra i due sono trascurabili nel contesto degli scenari descritti di seguito. Per impostazione predefinita, AWS Load Balancer Controller utilizza gli endpoint, puoi abilitarlo EndpointSlices abilitandolo sul enable-endpoint-sliceflagcontroller.

Usa i controlli sanitari

Per impostazione predefinita, Kubernetes esegue il controllo dello stato del processo in cui il processo kubelet sul nodo verifica se il processo principale del contenitore è in esecuzione o meno. In caso contrario, per impostazione predefinita riavvia quel contenitore. Tuttavia, puoi anche configurare le sonde Kubernetes per identificare quando un processo del contenitore è in esecuzione ma in uno stato di stallo, o se un'applicazione è stata avviata correttamente o meno. Le sonde possono essere basate su meccanismi exec, grpc, HttpGet e TcpSocket. In base al tipo e al risultato della sonda, è possibile riavviare il contenitore.

Consulta la sezione Creazione del Pod nella sezione Appendice di seguito per rivisitare la sequenza degli eventi nel processo di creazione del Pod.

Usa sonde di prontezza

Per impostazione predefinita, quando tutti i contenitori all'interno di un Pod sono in esecuzione, la condizione Pod è considerata «Pronto». Tuttavia, l'applicazione potrebbe non essere ancora in grado di elaborare le richieste dei clienti. Ad esempio, l'applicazione potrebbe aver bisogno di recuperare alcuni dati o configurazioni da una risorsa esterna per poter elaborare le richieste. In tale stato non si vorrebbe né chiudere l'applicazione né inoltrarle alcuna richiesta. La sonda Readiness consente di assicurarsi che il Pod non sia considerato «Pronto», il che significa che non verrà aggiunto all' EndpointSliceoggetto, fino a quando non verrà ottenuto il risultato della sonda. `success` D'altra parte, se la sonda si guasta ulteriormente lungo la linea, il Pod viene rimosso dall'oggetto. EndpointSlice È possibile configurare una sonda di prontezza nel manifesto del Pod per ogni contenitore. `kubelet`process su ogni nodo esegue la sonda di prontezza sui contenitori su quel nodo.

Utilizza i gate Pod Readiness

Un aspetto della sonda di prontezza è il fatto che non vi è alcun meccanismo di feedback/influenza esterno, il processo Kubelet sul nodo esegue la sonda e definisce lo stato della sonda. Ciò non ha alcun impatto sulle richieste tra i microservizi stessi nel livello Kubernetes (traffico est-ovest) poiché il Controller mantiene l'elenco degli endpoint (Pods) sempre aggiornato. EndpointSlice Perché e quando avresti bisogno di un meccanismo esterno allora?

Quando esponi le tue applicazioni utilizzando il tipo di servizio Kubernetes Load Balancer o Kubernetes Ingress (per il traffico nord-sud), l'elenco dei Pod IPs per il rispettivo servizio Kubernetes deve essere propagato al load balancer dell'infrastruttura esterna in modo che il load balancer abbia anche un elenco di obiettivi aggiornato. AWS Load Balancer Controller colma questa lacuna. Quando usi AWS Load Balancer Controller e fai levatarget group: IP, proprio kube-proxy come AWS Load Balancer Controller riceve anche un aggiornamento (viawatch) e poi comunica con l'API ELB per configurare e iniziare a registrare l'IP del Pod come destinazione sull'ELB.

Quando esegui un aggiornamento continuo di una distribuzione, vengono creati nuovi Pod e non appena la condizione di un nuovo Pod è «Pronta», il Pod viene interrotto. old/existing Durante questo processo, l' EndpointSliceoggetto Kubernetes viene aggiornato più velocemente del tempo impiegato dall'ELB per registrare i nuovi Pod come obiettivi, vedi Registrazione dei target. Per un breve periodo si potrebbe verificare una discrepanza di stato tra il livello Kubernetes e il livello di infrastruttura in cui le richieste dei client potrebbero essere eliminate. Durante questo periodo, all'interno del livello Kubernetes, i nuovi Pod sarebbero pronti per elaborare le richieste, ma dal punto di vista di ELB non lo sono.

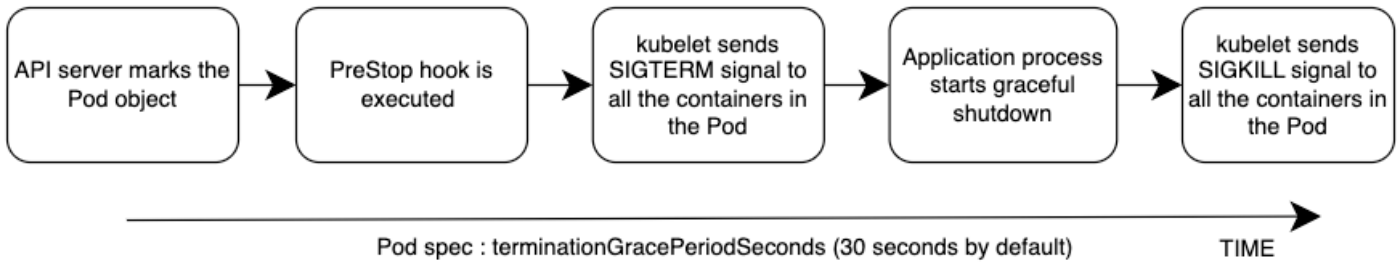
Pod Readiness Gates consente di definire requisiti aggiuntivi che devono essere soddisfatti prima che la condizione del Pod venga considerata «Pronta». Nel caso di AWS ELB, il controller AWS Load Balancer monitora lo stato del target (il Pod) sull'AWS ELB e, una volta completata la registrazione del target e il suo stato diventa «Healthy», il controller aggiorna la condizione del Pod a «Ready». Con questo approccio influisci sulla condizione del Pod in base allo stato della rete esterna, che è lo stato di destinazione su AWS ELB. Pod Readiness Gates è fondamentale negli scenari di aggiornamento progressivo in quanto consente di evitare che l'aggiornamento progressivo di una distribuzione interrompa i vecchi pod fino a quando lo stato target dei Pods appena creati non diventa «Healthy» su AWS ELB.

Chiudi le applicazioni con garbo

L'applicazione dovrebbe rispondere a un segnale SIGTERM avviando lo spegnimento graduale in modo che i client non subiscano tempi di inattività. Ciò significa che l'applicazione deve eseguire procedure di pulizia come il salvataggio dei dati, la chiusura dei descrittori di file, la chiusura delle connessioni al database, il completamento corretto delle richieste in corso e l'uscita tempestiva per soddisfare la richiesta di chiusura del Pod. È necessario impostare il periodo di grazia su un periodo sufficientemente lungo da consentire il completamento della pulizia. Per sapere come rispondere al segnale SIGTERM, puoi fare riferimento alle risorse del rispettivo linguaggio di programmazione che usi per la tua applicazione.

Se l'applicazione non è in grado di spegnersi correttamente alla ricezione di un segnale SIGTERM o se ignora/non riceve il segnale, puoi invece sfruttare PreStophook per avviare uno spegnimento graduale dell'applicazione. L'hook Prestop viene eseguito immediatamente prima dell'invio del segnale SIGTERM e può eseguire operazioni arbitrarie senza dover implementare tali operazioni nel codice dell'applicazione stesso.

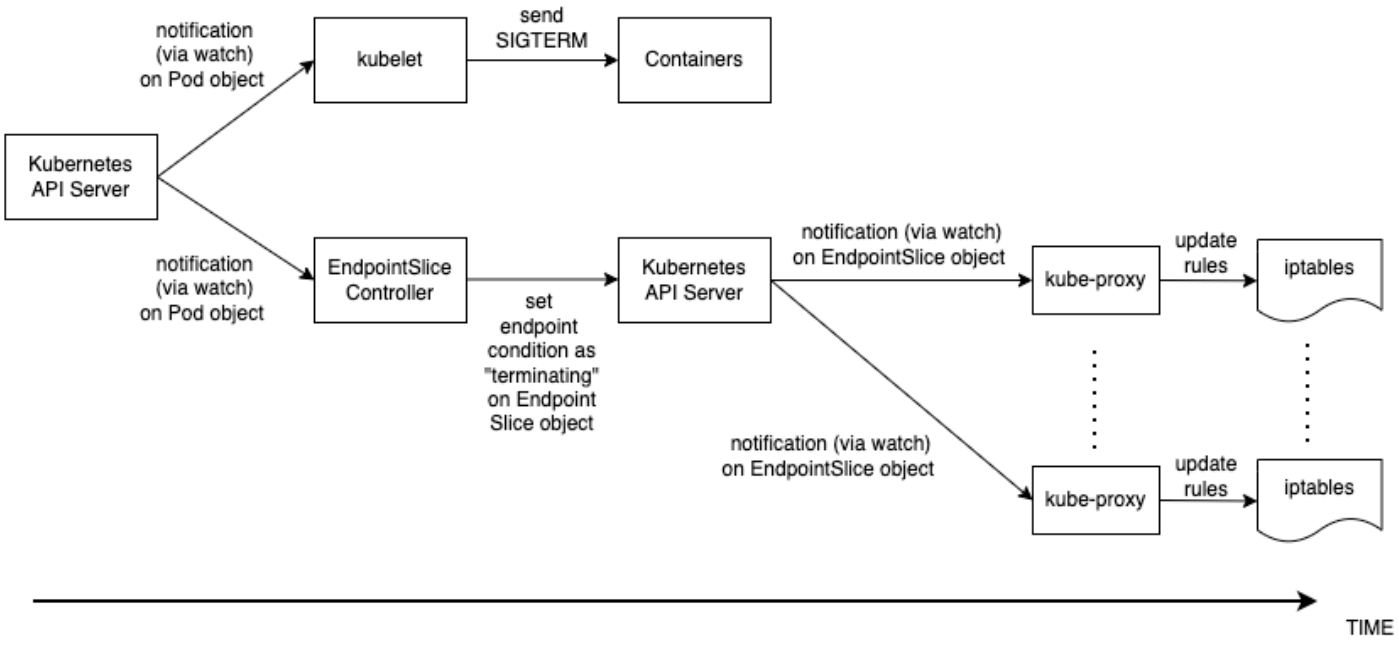
La sequenza complessiva degli eventi è mostrata nel diagramma seguente. Nota: indipendentemente dal risultato della corretta procedura di spegnimento dell'applicazione o dal risultato dell' PreStop hook, i contenitori dell'applicazione vengono infine chiusi alla fine del periodo di prova tramite SIGKILL.



Consulta la sezione Appendice riportata di seguito nella sezione Appendice per rivedere la sequenza degli eventi nel processo di eliminazione dei Pod.

Gestisci con garbo le richieste dei clienti

La sequenza di eventi nella cancellazione del Pod è diversa dalla creazione del Pod. Quando viene creato un Pod, `kubelet` aggiorna l'IP del Pod nell'API Kubernetes e solo allora l' EndpointSlice oggetto viene aggiornato. D'altra parte, quando un Pod viene chiuso, l'API Kubernetes notifica contemporaneamente sia il kubelet che il controller. EndpointSlice Ispeziona attentamente il seguente diagramma che mostra la sequenza degli eventi.



Il modo in cui lo stato si propaga dal server API fino alle regole iptables sui nodi spiegate sopra crea un'interessante condizione di gara. Perché c'è un'alta probabilità che il contenitore riceva il segnale SIGKILL molto prima che il kube-proxy su ciascun nodo aggiorni le regole iptables locali. In tal caso, due scenari degni di nota sono:

- Se la tua applicazione elimina immediatamente e senza mezzi termini le richieste e le coincidenze in volo al ricevimento di SIGTERM, significa che i clienti vedrebbero errori 50 volte superiori ovunque.
- Anche se l'applicazione garantisce che tutte le richieste e le connessioni in volo vengano elaborate completamente al ricevimento di SIGTERM, durante il periodo di grazia, le nuove richieste dei clienti verrebbero comunque inviate al contenitore dell'applicazione perché le regole di iptables potrebbero non essere ancora aggiornate. Fino a quando la procedura di pulizia non chiuderà

il socket del server sul contenitore, tali nuove richieste comporteranno nuove connessioni. Al termine del periodo di grazia, le connessioni, stabilite dopo il SIGTERM, in quel momento vengono interrotte incondizionatamente dall'invio di SIGKILL.

Impostare il periodo di grazia in Pod spec per un periodo sufficientemente lungo può risolvere questo problema, ma a seconda del ritardo di propagazione e del numero di richieste effettive dei client, è difficile prevedere il tempo necessario all'applicazione per chiudere correttamente le connessioni. Quindi l'approccio non perfetto ma più fattibile qui è quello di utilizzare un PreStop hook per ritardare il segnale SIGTERM fino all'aggiornamento delle regole di iptables per assicurarsi che non vengano inviate nuove richieste client all'applicazione, ma che continuino solo le connessioni esistenti. PreStop hook può essere un semplice gestore Exec come. `sleep 10`

Il comportamento e la raccomandazione sopra menzionati sarebbero ugualmente applicabili quando esponi le tue applicazioni utilizzando il tipo di servizio Kubernetes Load Balancer o Kubernetes Ingress (per il traffico nord-sud) utilizzando AWS Load Balancer Controller e leva. `target group: IP` Perché proprio come `kube-proxy` AWS Load Balancer Controller riceve anche un aggiornamento (tramite watch) sull' EndpointSlice oggetto e quindi comunica con l'API ELB per iniziare a cancellare l'IP del Pod dall'ELB. Tuttavia, a seconda del carico sull'API Kubernetes o sull'API ELB, anche questo può richiedere del tempo e il SIGTERM potrebbe essere già stato inviato all'applicazione molto tempo fa. Una volta che l'ELB inizia a cancellare la registrazione della destinazione, interrompe l'invio di richieste a quella destinazione in modo che l'applicazione non riceva nuove richieste e l'ELB avvia anche un ritardo di annullamento della registrazione, che per impostazione predefinita è di 300 secondi. Durante il processo di cancellazione, l'obiettivo è fondamentalmente il punto in `draining` cui l'ELB attende che le connessioni in volo verso tale destinazione si esauriscano. requests/existing Una volta scaduto il termine di cancellazione, l'obiettivo non viene utilizzato e tutte le richieste in volo dirette a quell'obiettivo vengono annullate con la forza.

Usa Pod disruption budget

Configura un Pod Disruption Budget (PDB) per le tue applicazioni. PDBlimits il numero di Pod di un'applicazione replicata che non funzionano contemporaneamente a causa di interruzioni volontarie. Garantisce che un numero o una percentuale minima di pod rimangano disponibili in una o più distribuzioni. StatefulSet Ad esempio, un'applicazione basata sul quorum deve garantire che il numero di repliche in esecuzione non sia mai inferiore al numero necessario per il quorum. Oppure un front-end web potrebbe garantire che il numero di repliche che servono il carico non scenda mai al di sotto di una certa percentuale del totale. PDB proteggerà l'applicazione da azioni come il drenaggio

dei nodi o il lancio di nuove versioni di Deployments. Tieni presente che i PDB non proteggeranno l'applicazione da interruzioni involontarie come un guasto del sistema operativo del nodo o la perdita di connettività di rete. Per ulteriori informazioni, consulta la documentazione Specificazione di un budget di interruzione delle attività per l'applicazione in Kubernetes.

Riferimenti

- KubeCon Sessione Europa 2019 - Pronti? Un approfondimento su Pod Readiness Gates for Service Health
- Libro - Kubernetes in azione
- Blog AWS - Come scalare rapidamente la tua applicazione con ALB su EKS (senza perdere traffico)

Appendice

Creazione di pod

È fondamentale capire qual è la sequenza degli eventi in uno scenario in cui viene distribuito un Pod e poi si passa healthy/ready alla ricezione e all'elaborazione delle richieste dei client. Parliamo della sequenza degli eventi.

1. Un Pod viene creato sul piano di controllo di Kubernetes (ad esempio tramite un comando kubectl, un aggiornamento di distribuzione o un'azione di ridimensionamento).
2. `kube-scheduler`assegna il Pod a un nodo del cluster.
3. Il processo kubelet in esecuzione sul nodo assegnato riceve l'aggiornamento (via`watch`) e comunica con il runtime del contenitore per avviare i contenitori definiti nelle specifiche Pod.
4. Quando i contenitori iniziano a funzionare, il kubelet aggiorna la condizione Pod come `Ready` nell'oggetto Pod nell'API Kubernetes.
5. Il EndpointSliceController riceve l'aggiornamento della condizione Pod (tramite`watch`) e aggiunge il Pod IP/Port come nuovo endpoint all'EndpointSliceoggetto (elenco di Pod IPs) del rispettivo servizio Kubernetes.
6. Il processo kube-proxy su ciascun nodo riceve l'aggiornamento (via`watch`) sull'EndpointSlice oggetto e quindi aggiorna le regole iptables su ciascun nodo, con il nuovo Pod IP/Port.

Eliminazione del pod

Proprio come la creazione dei Pod, è fondamentale capire qual è la sequenza degli eventi durante l'eliminazione dei Pod. Parliamo della sequenza degli eventi.

1. Una richiesta di eliminazione del Pod viene inviata al server dell'API Kubernetes (ad esempio tramite un `kubectl` comando, un aggiornamento di distribuzione o un'azione di ridimensionamento).
2. Il server API Kubernetes avvia un periodo di prova, che per impostazione predefinita è di 30 secondi, impostando il campo deletionTimestamp nell'oggetto Pod. (Il periodo di tolleranza può essere configurato nelle specifiche del Pod tramite) `terminationGracePeriodSeconds`
3. Il `kubelet` processo in esecuzione sul nodo riceve l'aggiornamento (tramite watch) sull'oggetto Pod e invia un segnale SIGTERM all'identificatore di processo 1 (PID 1) all'interno di ciascun contenitore in quel Pod. Quindi guarda il. `terminationGracePeriodSeconds`
4. Il EndpointSliceController riceve anche l'aggiornamento (tramite`watch`) dallo Step 2 e imposta la condizione dell'endpoint su «terminazione» nell'EndpointSliceoggetto (elenco di Pod IPs) del rispettivo servizio Kubernetes.
5. Il processo kube-proxy su ciascun nodo riceve l'aggiornamento (via`watch`) sull'EndpointSlice oggetto, quindi le regole iptables su ciascun nodo vengono aggiornate dal kube-proxy per interrompere l'inoltro delle richieste dei client al Pod.
6. `terminationGracePeriodSeconds`Alla scadenza, `kubelet` invia il segnale SIGKILL al processo principale di ogni contenitore nel Pod e lo termina forzatamente.
7. TheEndpointSliceIl controller rimuove l'endpoint dall'oggetto. EndpointSlice
8. Il server API elimina l'oggetto Pod.

Monitoraggio dei carichi di lavoro EKS per problemi di prestazioni di rete

Monitoraggio del traffico CoredNS per problemi di throttling DNS

L'esecuzione di carichi di lavoro DNS intensivi può talvolta causare errori CoredNS intermittenti dovuti alla limitazione del DNS e ciò può influire sulle applicazioni in cui è possibile riscontrare errori occasionali. UnknownHostException

Deployment for CoreDNS ha una politica anti-affinità che indica allo scheduler Kubernetes di eseguire istanze di CoredNS su nodi di lavoro separati nel cluster, ovvero dovrebbe evitare di collocare le repliche sullo stesso nodo di lavoro. Ciò riduce efficacemente il numero di query DNS per interfaccia di rete perché il traffico proveniente da ciascuna replica viene instradato attraverso un ENI diverso. Se noti che le query DNS vengono limitate a causa del limite di 1024 pacchetti al secondo, puoi 1) provare ad aumentare il numero di repliche CoredNS o 2) implementare. NodeLocal DNSCache Per ulteriori informazioni, consulta Monitor CoredNS Metrics.

Challenge

- La perdita di pacchetti avviene in pochi secondi e può essere difficile per noi monitorare correttamente questi schemi per determinare se la limitazione del DNS stia effettivamente avvenendo.
- Le query DNS vengono limitate a livello di elastic network interface. Pertanto, le query limitate non vengono visualizzate nella registrazione delle query.
- I log di flusso non acquisiscono tutto il traffico IP. (ad esempio Traffico generato da istanze quando contattano il server Amazon DNS. Se utilizzi il tuo server DNS, tutto il traffico verso quel server DNS viene registrato

Soluzione

Un modo semplice per identificare i problemi di limitazione del DNS nei nodi di lavoro è l'acquisizione dei parametri. `linklocal_allowance_exceeded` Il linklocal_allowance_exceeded è il numero di pacchetti persi perché il PPS del traffico verso i servizi proxy locali ha superato il massimo consentito dall'interfaccia di rete. Ciò influisce sul traffico verso il servizio DNS, il servizio di metadati dell'istanza e il servizio Amazon Time Sync. Invece di tracciare questo evento in tempo reale, possiamo trasmettere questa metrica anche ad Amazon Managed Service for Prometheus e visualizzarla in Amazon Managed Grafana

Monitoraggio dei ritardi delle query DNS utilizzando le metriche di Conntrack

Un'altra metrica che può aiutare a monitorare il ritardo di throttling /query di CoredNS è and. `conntrack_allowance_available` `conntrack_allowance_exceeded` Gli errori di connettività causati dal superamento delle quote di Connections Tracked possono avere un impatto maggiore rispetto a quelli derivanti dal superamento di altre quote. Quando si utilizza il protocollo TCP per il trasferimento dei dati, i pacchetti che vengono messi in coda o interrotti a causa del superamento

delle quote di rete delle istanze EC2, ad esempio larghezza di banda, PPS, ecc., vengono in genere gestiti in modo corretto grazie alle funzionalità di controllo della congestione di TCP. I flussi interessati verranno rallentati e i pacchetti persi verranno ritrasmessi. Tuttavia, quando un'istanza supera il limite consentito per le connessioni tracciate, non è possibile stabilire nuove connessioni finché alcune di quelle esistenti non vengono chiuse per fare spazio a nuove connessioni.

`conntrack_allowance_availablee` `conntrack_allowance_exceeded` aiuta i clienti a monitorare la quota di connessioni tracciate, che varia a seconda dell'istanza. Questi parametri delle prestazioni di rete offrono ai clienti la visibilità del numero di pacchetti in coda o eliminati quando vengono superate le tolleranze di rete di un'istanza, come Network Bandwidth Packets-Per-Second (PPS), Connections Tracked e Link-Local Service Access (Amazon DNS, Instance Meta Data Service, Amazon Time Sync)

`conntrack_allowance_available`è il numero di connessioni tracciate che possono essere stabilite dall'istanza prima di raggiungere il limite di connessioni tracciate per quel tipo di istanza (supportato solo per le istanze basate su nitro). `conntrack_allowance_exceeded`è il numero di pacchetti persi perché il tracciamento delle connessioni ha superato il limite massimo per l'istanza e non è stato possibile stabilire nuove connessioni.

Altre importanti metriche relative alle prestazioni di rete

Altre importanti metriche relative alle prestazioni di rete includono:

`bw_in_allowance_exceeded`(il valore ideale della metrica deve essere zero) è il numero di pacchetti in coda and/or eliminati perché la larghezza di banda aggregata in entrata ha superato il massimo per l'istanza

`bw_out_allowance_exceeded`(il valore ideale della metrica deve essere zero) è il numero di pacchetti in coda eliminati perché la larghezza di banda aggregata and/or in uscita ha superato il massimo per l'istanza

`pps_allowance_exceeded`(il valore ideale della metrica deve essere zero) è il numero di pacchetti in coda and/or eliminati perché il PPS bidirezionale ha superato il massimo per l'istanza

Acquisizione delle metriche per monitorare i carichi di lavoro relativi ai problemi di prestazioni di rete

Il driver Elastic Network Adapter (ENA) pubblica i parametri delle prestazioni di rete discussi in precedenza a partire dalle istanze in cui sono abilitati. Tutte le metriche delle prestazioni di rete

possono essere pubblicate utilizzando l' CloudWatch agente. CloudWatch Per ulteriori informazioni, consulta il blog.

Ora acquisiamo le metriche sopra discusse, le archiviamo in Amazon Managed Service for Prometheus e le visualizziamo utilizzando Amazon Managed Grafana

Prerequisiti

- ethtool - Assicurati che sui nodi di lavoro sia installato ethtool
- Uno spazio di lavoro AMP configurato nel tuo account AWS. Per istruzioni, consulta Creare uno spazio di lavoro nella Guida per l'utente AMP.
- Area di lavoro Grafana gestita da Amazon

Implementazione di Prometheus Node Exporter

Per raccogliere le metriche ethtool per i nodi di lavoro, dobbiamo implementare un collettore che raccolga le metriche e le esponga nel formato Prometheus. I seguenti comandi aggiungeranno il repository `prometheus-community` Helm e installeranno il grafico con il raccoglitore abilitato. `prometheus-node-exporter` `ethtool`

```
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts

helm upgrade -i prometheus-node-exporter prometheus-community/prometheus-node-exporter
 \
  --set extraArgs[0]="--collector.ethtool" \
  --set extraArgs[1]="--collector.ethtool.device-include=(eth|em|eno|ens|enp)[0-9s]+" \
  --set extraArgs[2]="--collector.ethtool.metrics-include=.*" \
  --set podAnnotations."prometheus\.io/scrape='true'" \
  --set podAnnotations."prometheus\.io/port='9100'"
```

Implementa il collettore ADOT per acquisire i parametri ethtool e archiviarli nell'area di lavoro Amazon Managed Service for Prometheus

Ogni cluster in cui installi AWS Distro for OpenTelemetry (ADOT) deve avere questo ruolo per concedere al tuo account di servizio AWS le autorizzazioni per archiviare i parametri in Amazon Managed Service for Prometheus. Per creare e associare il ruolo IAM al tuo account di servizio Amazon EKS tramite IRSA, segui la procedura seguente:

```
eksctl create iamserviceaccount --name adot-collector --namespace default
 --cluster <CLUSTER_NAME> --attach-policy-arn arn:aws:iam::aws:policy/
AmazonPrometheusRemoteWriteAccess --attach-policy-arn arn:aws:iam::aws:policy/
AWSXrayWriteOnlyAccess --attach-policy-arn arn:aws:iam::aws:policy/
CloudWatchAgentServerPolicy --region <REGION> --approve  --override-existing-
serviceaccounts
```

Implementiamo il collettore ADOT per acquisire le metriche dall'esportatore prometheus ethtool e archiviarle in Amazon Managed Service for Prometheus

La procedura seguente utilizza un file YAML di esempio con deployment come valore di modalità. Si tratta della modalità predefinita che implementa il raccoglitore ADOT in modo simile a un'applicazione autonoma. Questa configurazione riceve le metriche OTLP dall'applicazione di esempio e le metriche di Amazon Managed Service for Prometheus ricavate dai pod del cluster.

```
curl -o collector-config-amp.yaml https://raw.githubusercontent.com/aws-observability/
aws-otel-community/master/sample-configs/operator/collector-config-amp.yaml
```

In collector-config-amp .yaml, sostituisci quanto segue con i tuoi valori:

- modalità: distribuzione
- Account di servizio: un collezionista di punti
- punto finale: <YOUR_REMOTE_WRITE_ENDPOINT>
- regione: <YOUR_ >AWS_REGION
- nome: adot-collector

```
kubectl apply -f collector-config-amp.yaml
```

Una volta distribuito l'adot collector, i parametri verranno archiviati correttamente in Amazon Prometheus

Configura il gestore degli avvisi in Amazon Managed Service for Prometheus per inviare notifiche

Puoi utilizzare Alert Manager in Amazon Managed Service for Prometheus per configurare le regole di avviso per gli avvisi critici, quindi puoi inviare notifiche a un argomento di Amazon SNS. Configuriamo le regole di registrazione e le regole di avviso per verificare le metriche discusse finora.

Utilizzeremo il controller ACK per Amazon Managed Service for Prometheus per fornire le regole di avviso e registrazione.

Implementiamo il controller ACL per il servizio Amazon Managed Service for Prometheus:

```
export SERVICE=prometheusservice
export RELEASE_VERSION=`curl -sL https://api.github.com/repos/aws-controllers-k8s/
$SERVICE-controller/releases/latest | grep '"tag_name":' | cut -d'"' -f4`
export ACK_SYSTEM_NAMESPACE=ack-system
export AWS_REGION=us-east-1
aws ecr-public get-login-password --region us-east-1 | helm registry login --username
 AWS --password-stdin public.ecr.aws
helm install --create-namespace -n $ACK_SYSTEM_NAMESPACE ack-$SERVICE-controller \
oci://public.ecr.aws/aws-controllers-k8s/$SERVICE-chart --version=$RELEASE_VERSION --
set=aws.region=$AWS_REGION
```

Esegui il comando e dopo qualche istante dovresti vedere il seguente messaggio:

```
You are now able to create Amazon Managed Service for Prometheus (AMP) resources!

The controller is running in "cluster" mode.

The controller is configured to manage AWS resources in region: "us-east-1"

The ACK controller has been successfully installed and ACK can now be used to provision
 an Amazon Managed Service for Prometheus workspace.
```

Creiamo ora un file yaml per il provisioning della definizione del gestore degli avvisi e dei gruppi di regole. Salva il file seguente come `rulegroup.yaml`

```
apiVersion: prometheusservice.services.k8s.aws/v1alpha1
kind: RuleGroupsNamespace
metadata:
   name: default-rule
spec:
   workspaceID: <Your WORKSPACE-ID>
   name: default-rule
   configuration: |
     groups:
     - name: ppsallowance
       rules:
       - record: metric:pps_allowance_exceeded
```

```
          expr: rate(node_net_ethtool{device="eth0",type="pps_allowance_exceeded"}[30s])
        - alert: PPSAllowanceExceeded
          expr: rate(node_net_ethtool{device="eth0",type="pps_allowance_exceeded"}
 [30s]) > 0
          labels:
            severity: critical

          annotations:
            summary: Connections dropped due to total allowance exceeding for the
 (instance {{ $labels.instance }})
            description: "PPSAllowanceExceeded is greater than 0"
      - name: bw_in
        rules:
        - record: metric:bw_in_allowance_exceeded
          expr: rate(node_net_ethtool{device="eth0",type="bw_in_allowance_exceeded"}
[30s])
        - alert: BWINAllowanceExceeded
          expr: rate(node_net_ethtool{device="eth0",type="bw_in_allowance_exceeded"}
 [30s]) > 0
          labels:
            severity: critical

          annotations:
            summary: Connections dropped due to total allowance exceeding for the
 (instance {{ $labels.instance }})
            description: "BWInAllowanceExceeded is greater than 0"
      - name: bw_out
        rules:
        - record: metric:bw_out_allowance_exceeded
          expr: rate(node_net_ethtool{device="eth0",type="bw_out_allowance_exceeded"}
[30s])
        - alert: BWOutAllowanceExceeded
          expr: rate(node_net_ethtool{device="eth0",type="bw_out_allowance_exceeded"}
 [30s]) > 0
          labels:
            severity: critical

          annotations:
            summary: Connections dropped due to total allowance exceeding for the
 (instance {{ $labels.instance }})
            description: "BWoutAllowanceExceeded is greater than 0"
      - name: conntrack
        rules:
        - record: metric:conntrack_allowance_exceeded
```

```
        expr: rate(node_net_ethtool{device="eth0",type="conntrack_allowance_exceeded"}
[30s])
      - alert: ConntrackAllowanceExceeded
        expr: rate(node_net_ethtool{device="eth0",type="conntrack_allowance_exceeded"}
 [30s]) > 0
        labels:
          severity: critical

        annotations:
          summary: Connections dropped due to total allowance exceeding for the
 (instance {{ $labels.instance }})
          description: "ConnTrackAllowanceExceeded is greater than 0"
    - name: linklocal
      rules:
      - record: metric:linklocal_allowance_exceeded
        expr: rate(node_net_ethtool{device="eth0",type="linklocal_allowance_exceeded"}
[30s])
      - alert: LinkLocalAllowanceExceeded
        expr: rate(node_net_ethtool{device="eth0",type="linklocal_allowance_exceeded"}
 [30s]) > 0
        labels:
          severity: critical

        annotations:
          summary: Packets dropped due to PPS rate allowance exceeded for local
 services  (instance {{ $labels.instance }})
          description: "LinkLocalAllowanceExceeded is greater than 0"
```

Sostituisci Your WORKSPACE-ID con l'ID Workspace dell'area di lavoro che stai utilizzando.

Configuriamo ora la definizione del gestore degli avvisi. Salva il file seguente come `alertmanager.yaml`

```
apiVersion: prometheusservice.services.k8s.aws/v1alpha1
kind: AlertManagerDefinition
metadata:
  name: alert-manager
spec:
  workspaceID: <Your WORKSPACE-ID >
  configuration: |
    alertmanager_config: |
      route:
         receiver: default_receiver
```

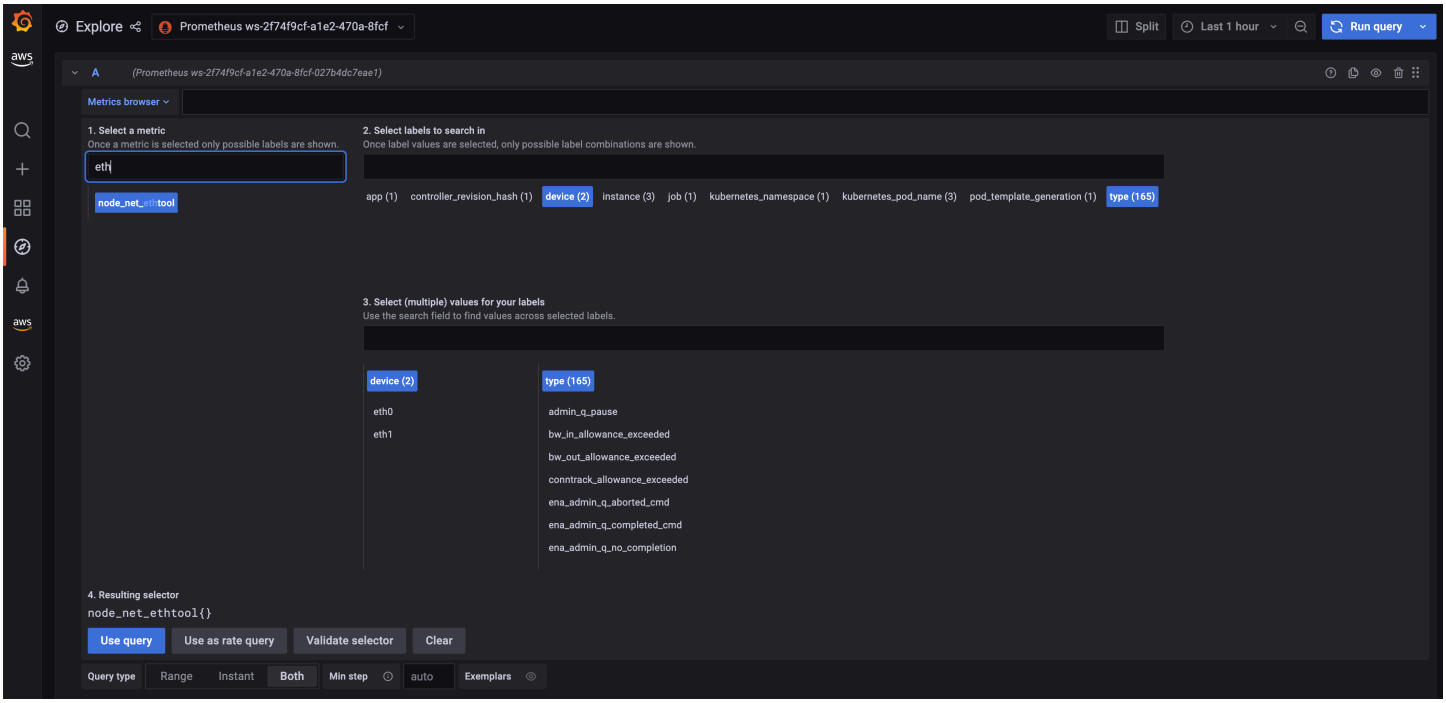
```
    receivers:
    - name: default_receiver
       sns_configs:
       - topic_arn: TOPIC-ARN
         sigv4:
           region: REGION
         message: |
           alert_type: {{ .CommonLabels.alertname }}
           event_type: {{ .CommonLabels.event_type }}
```

Sostituisci You WORKSPACE-ID con l'ID Workspace del nuovo workspace, TOPIC-ARN con l'ARN di un argomento di Amazon Simple Notification Service a cui desideri inviare gli avvisi e REGION con la regione corrente del carico di lavoro. Assicurati che il tuo spazio di lavoro disponga delle autorizzazioni per inviare messaggi ad Amazon SNS.

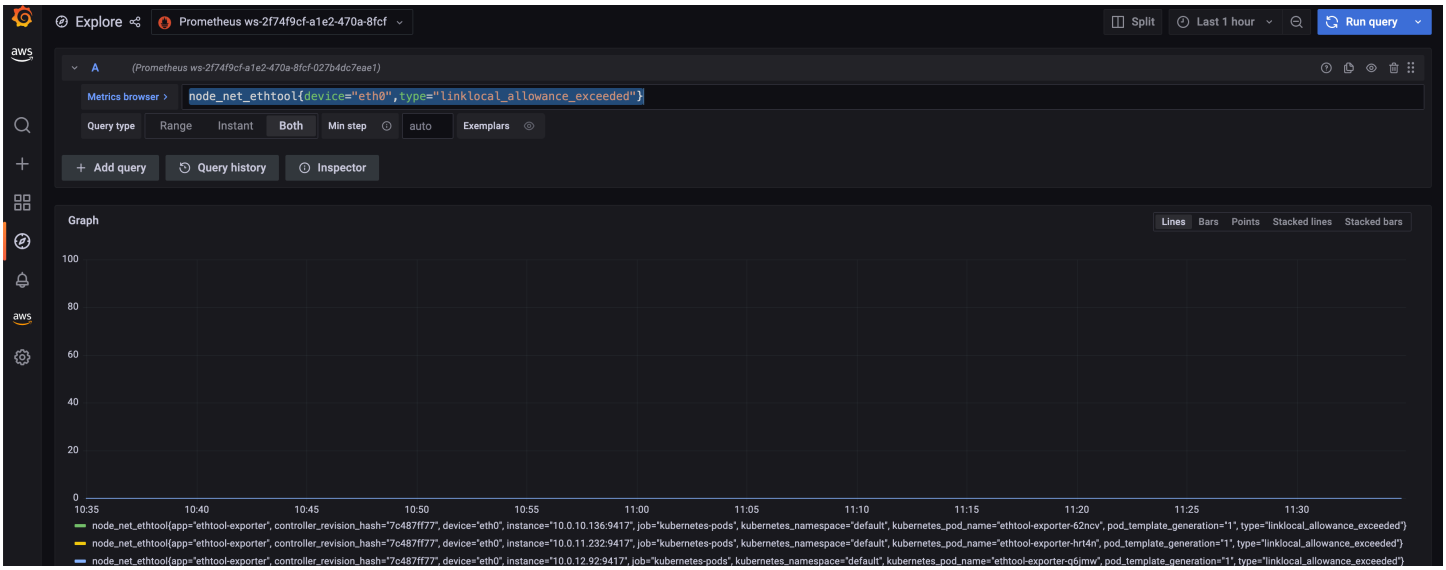
Visualizza le metriche ethtool in Amazon Managed Grafana

Visualizziamo le metriche all'interno di Amazon Managed Grafana e creiamo una dashboard. Configura Amazon Managed Service for Prometheus come origine dati all'interno della console Amazon Managed Grafana. Per istruzioni, consulta Aggiungere Amazon Prometheus come origine dati

Esaminiamo ora le metriche in Amazon Managed Grafana: fai clic sul pulsante esplora e cerca ethtool:



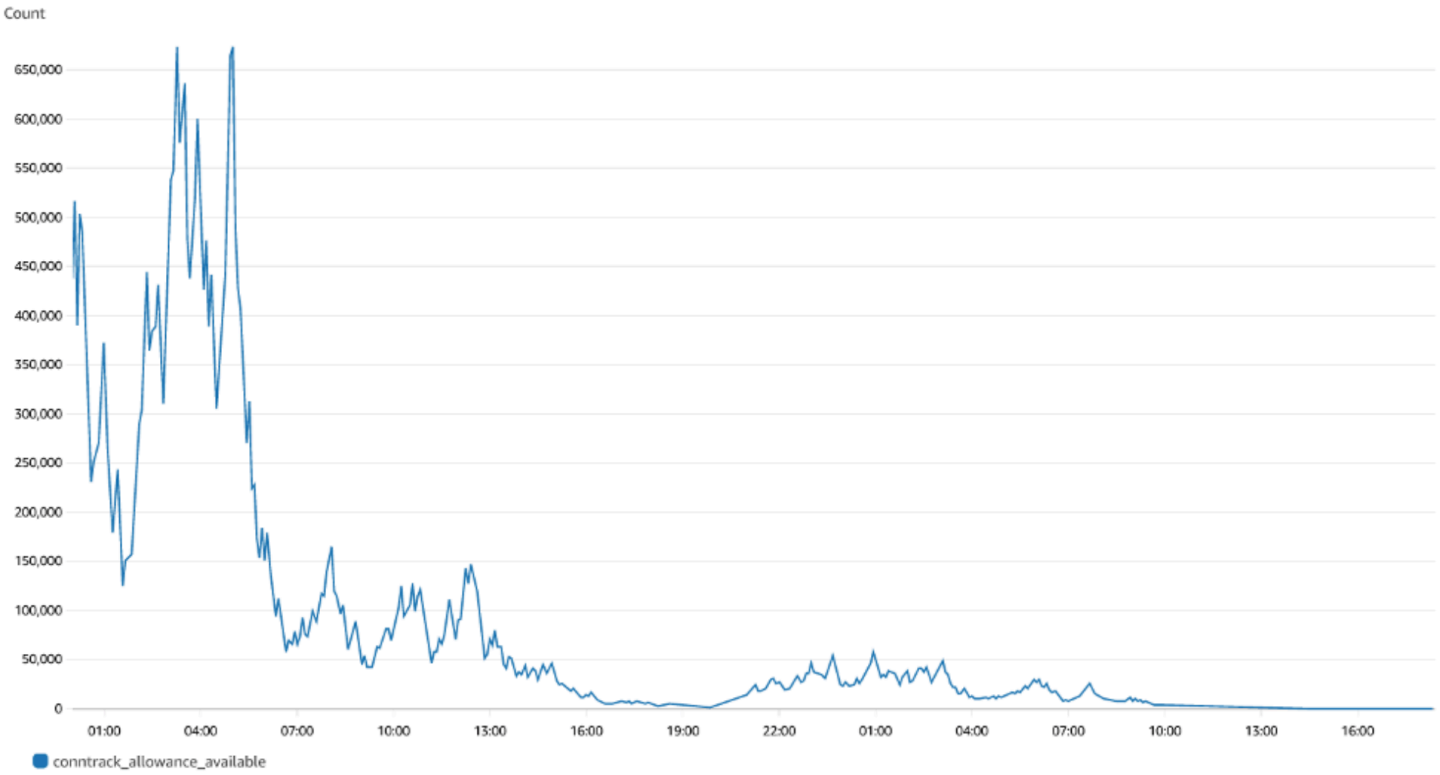
Creiamo una dashboard per la metrica linklocal_allowance_exceeded utilizzando la query. `rate(node_net_ethtool{device="eth0",type="linklocal_allowance_exceeded"}[30s])` Il risultato sarà la dashboard seguente.



Possiamo vedere chiaramente che non sono stati persi pacchetti poiché il valore è zero.

Creiamo una dashboard per la metrica conntrack_allowance_exceeded utilizzando la query. `rate(node_net_ethtool{device="eth0",type="conntrack_allowance_exceeded"}[30s])` Risulterà nella dashboard seguente.

La metrica `conntrack_allowance_exceeded` può essere visualizzata in, a condizione che si esegua un agente cloudwatch come descritto CloudWatch qui. La dashboard risultante CloudWatch sarà simile alla seguente:



Possiamo vedere chiaramente che non sono stati persi pacchetti poiché il valore è zero. Se utilizzi istanze basate su Nitro, puoi creare una dashboard simile `conntrack_allowance_available` e monitorare in modo proattivo le connessioni nella tua istanza EC2. Puoi estendere ulteriormente questa funzionalità configurando avvisi in Amazon Managed Grafana per inviare notifiche a Slack, SNS, Pagerduty ecc.

Esecuzione di kube-proxy in modalità IPVS

EKS in modalità IP Virtual Server (IPVS) risolve il problema di latenza di rete spesso riscontrato durante l'esecuzione di cluster di grandi dimensioni con oltre 1.000 servizi con kube-proxy in esecuzione nella modalità iptables precedente. Questo problema di prestazioni è il risultato dell'elaborazione sequenziale delle regole di filtraggio dei pacchetti iptables per ogni pacchetto. Questo problema di latenza è stato risolto in nftables, il successore di iptables. Tuttavia, al momento della stesura di questo documento, kube-proxy è ancora in fase di sviluppo per utilizzare nftables. Per ovviare a questo problema, puoi configurare il cluster per l'esecuzione in modalità IPVS. `kube-proxy`

Panoramica di

IPVS, che è GA dalla versione 1.11 di Kubernetes, utilizza tabelle hash anziché la ricerca lineare per elaborare i pacchetti, garantendo l'efficienza dei cluster con migliaia di nodi e servizi. IPVS è stato progettato per il bilanciamento del carico, il che lo rende una soluzione adatta per i problemi di prestazioni di rete di Kubernetes.

IPVS offre diverse opzioni per la distribuzione del traffico verso i pod di backend. Informazioni dettagliate per ciascuna opzione sono disponibili nella documentazione ufficiale di Kubernetes, ma di seguito è riportato un semplice elenco. Round Robin e Least Connections sono tra le scelte più popolari per le opzioni di bilanciamento del carico IPVS in Kubernetes.

```
- rr (Round Robin)
- wrr (Weighted Round Robin)
- lc (Least Connections)
- wlc (Weighted Least Connections)
- lblc (Locality Based Least Connections)
- lblcr (Locality Based Least Connections with Replication)
- sh (Source Hashing)
- dh (Destination Hashing)
- sed (Shortest Expected Delay)
- nq (Never Queue)
```

Implementazione

Sono necessari solo pochi passaggi per abilitare IPVS nel cluster EKS. La prima cosa da fare è assicurarsi che sulle immagini dei nodi di lavoro EKS sia installato il `ipvsadm` pacchetto di amministrazione Linux Virtual Server. Per installare questo pacchetto su un'immagine basata su Fedora, come Amazon Linux 2023, puoi eseguire il seguente comando sull'istanza del nodo di lavoro.

```
sudo dnf install -y ipvsadm
```

Su un'immagine basata su Debian, come Ubuntu, il comando di installazione sarebbe simile al seguente.

```
sudo apt-get install ipvsadm
```

Successivamente, è necessario caricare i moduli del kernel per le opzioni di configurazione IPVS elencate sopra. Si consiglia di scrivere questi moduli in un file all'interno della `/etc/modules-load.d/` directory in modo che sopravvivano al riavvio.

```
sudo sh -c 'cat << EOF > /etc/modules-load.d/ipvs.conf
ip_vs
ip_vs_rr
ip_vs_wrr
ip_vs_lc
ip_vs_wlc
ip_vs_lblc
ip_vs_lblcr
ip_vs_sh
ip_vs_dh
ip_vs_sed
ip_vs_nq
nf_conntrack
EOF'
```

È possibile eseguire il comando seguente per caricare questi moduli su un computer già in esecuzione.

```
sudo modprobe ip_vs
sudo modprobe ip_vs_rr
sudo modprobe ip_vs_wrr
sudo modprobe ip_vs_lc
sudo modprobe ip_vs_wlc
sudo modprobe ip_vs_lblc
sudo modprobe ip_vs_lblcr
sudo modprobe ip_vs_sh
sudo modprobe ip_vs_dh
sudo modprobe ip_vs_sed
sudo modprobe ip_vs_nq
sudo modprobe nf_conntrack
```

> **Note**
>
> Si consiglia vivamente di eseguire questi passaggi del nodo di lavoro come parte del processo di avvio del nodo di lavoro tramite script di dati utente o in qualsiasi script di compilazione eseguito per creare un AMI del nodo di lavoro personalizzato.

Successivamente, configurerai i tuoi cluster `kube-proxy` DaemonSet per l'esecuzione in modalità IPVS. Questa operazione viene eseguita impostando `kube-proxy` `mode` su `ipvs` e su una

delle opzioni di bilanciamento del carico sopra elencate, ad esempio: `rr` per Round Robin. `ipvs scheduler`

> **Warning**
>
> Si tratta di una modifica dirompente e deve essere eseguita al di fuori delle ore lavorative. Consigliamo di apportare queste modifiche durante la creazione iniziale del cluster EKS per ridurre al minimo gli impatti.

Puoi emettere un comando AWS CLI per abilitare IPVS aggiornando il `kube-proxy` componente aggiuntivo EKS.

```
aws eks update-addon --cluster-name $CLUSTER_NAME --addon-name kube-proxy \
  --configuration-values '{"ipvs": {"scheduler": "rr"}, "mode": "ipvs"}' \
  --resolve-conflicts OVERWRITE
```

Oppure puoi farlo modificando il file nel `kube-proxy-config` ConfigMap tuo cluster.

```
kubectl -n kube-system edit cm kube-proxy-config
```

Trova l'`scheduler`impostazione sotto `ipvs` e imposta il valore su una delle opzioni di bilanciamento del carico ipvs elencate sopra, ad esempio: `rr` per Round Robin. Trova l'`mode`impostazione, che è predefinita su`iptables`, e modifica il valore in. `ipvs` Il risultato di entrambe le opzioni dovrebbe essere simile alla configurazione riportata di seguito.

```
  iptables:
    masqueradeAll: false
    masqueradeBit: 14
    minSyncPeriod: 0s
    syncPeriod: 30s
  ipvs:
    excludeCIDRs: null
    minSyncPeriod: 0s
    scheduler: "rr"
    syncPeriod: 30s
  kind: KubeProxyConfiguration
  metricsBindAddress: 0.0.0.0:10249
  mode: "ipvs"
  nodePortAddresses: null
```

```
  oomScoreAdj: -998
  portRange: ""
  udpIdleTimeout: 250ms
```

Se i nodi di lavoro sono stati aggiunti al cluster prima di apportare queste modifiche, sarà necessario riavviare il kube-proxy DaemonSet.

```
kubectl -n kube-system rollout restart ds kube-proxy
```

Convalida

Puoi verificare che il cluster e i nodi di lavoro siano in esecuzione in modalità IPVS emettendo il seguente comando su uno dei tuoi nodi di lavoro.

```
sudo ipvsadm -L
```

Come minimo, dovresti vedere un risultato simile a quello riportato di seguito, che mostra le voci relative al servizio Kubernetes API Server all'indirizzo e al servizio CoredNS `10.100.0.1` all'indirizzo. `10.100.0.10`

```
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn InActConn
TCP  ip-10-100-0-1.us-east-1. rr
  -> ip-192-168-113-81.us-eas Masq        1      0          0
  -> ip-192-168-162-166.us-ea Masq        1      1          0
TCP  ip-10-100-0-10.us-east-1 rr
  -> ip-192-168-104-215.us-ea Masq        1      0          0
  -> ip-192-168-123-227.us-ea Masq        1      0          0
UDP  ip-10-100-0-10.us-east-1 rr
  -> ip-192-168-104-215.us-ea Masq        1      0          0
  -> ip-192-168-123-227.us-ea Masq        1      0          0
```

> **Note**
>
> Questo output di esempio proviene da un cluster EKS con un intervallo di indirizzi IP di servizio di. `10.100.0.0/16`

Le migliori pratiche di scalabilità EKS

> **Tip**
>
> Esplora le best practice tramite i workshop Amazon EKS.

Questa guida fornisce consigli per scalare i cluster EKS. L'obiettivo della scalabilità di un cluster EKS è quello di massimizzare la quantità di lavoro che un singolo cluster può eseguire. L'utilizzo di un singolo cluster EKS di grandi dimensioni può ridurre il carico operativo rispetto all'utilizzo di più cluster, ma presenta dei compromessi per cose come le implementazioni in più regioni, l'isolamento dei tenant e gli aggiornamenti dei cluster. In questo documento ci concentreremo su come ottenere la massima scalabilità con un singolo cluster.

Come utilizzare questa guida

Questa guida è destinata agli sviluppatori e agli amministratori responsabili della creazione e della gestione di cluster EKS in AWS. Si concentra su alcune pratiche generiche di scalabilità Kubernetes, ma non contiene specifiche per i cluster Kubernetes autogestiti o i cluster che funzionano al di fuori di una regione AWS con EKS Anywhere.

Ogni argomento ha una breve panoramica, seguita da consigli e best practice per gestire i cluster EKS su larga scala. Non è necessario leggere gli argomenti in un ordine particolare e i consigli non devono essere applicati senza averne testato e verificato il corretto funzionamento nei cluster.

Comprendere le dimensioni di ridimensionamento

La scalabilità è diversa dalle prestazioni e dall'affidabilità e tutte e tre devono essere prese in considerazione quando si pianificano le esigenze di cluster e carichi di lavoro. Man mano che i cluster sono scalabili, devono essere monitorati, ma questa guida non tratterà le migliori pratiche di monitoraggio. EKS è scalabile fino a raggiungere grandi dimensioni, ma è necessario pianificare come scalare un cluster oltre 300 nodi o 5.000 Pod. Non si tratta di numeri assoluti, ma derivano dalla collaborazione di questa guida con più utenti, ingegneri e professionisti del supporto.

La scalabilità in Kubernetes è multidimensionale e non esistono impostazioni o consigli specifici che funzionino in ogni situazione. Le aree principali in cui possiamo fornire indicazioni per la scalabilità includono:

Kubernetes Control Plane in un cluster EKS include tutti i servizi che AWS esegue e ridimensiona automaticamente per te (ad esempio il server API Kubernetes). La scalabilità del Control Plane è responsabilità di AWS, ma l'utilizzo responsabile del Control Plane è una tua responsabilità.

La scalabilità del piano dati di Kubernetes si occupa delle risorse AWS necessarie per il cluster e i carichi di lavoro, ma non rientrano nel piano di controllo EKS. Le risorse, tra cui istanze EC2, kubelet e storage, devono tutte essere scalate man mano che il cluster cresce.

I servizi cluster sono controller e applicazioni Kubernetes eseguiti all'interno del cluster e forniscono funzionalità per il cluster e i carichi di lavoro. Questi possono essere componenti aggiuntivi EKS e anche altri servizi o grafici Helm installati per la conformità e le integrazioni. Questi servizi dipendono spesso dai carichi di lavoro e, man mano che i carichi di lavoro aumentano, i servizi del cluster dovranno adattarsi ad essi.

I carichi di lavoro sono il motivo per cui si dispone di un cluster e devono essere scalati orizzontalmente con il cluster. Esistono integrazioni e impostazioni dei carichi di lavoro in Kubernetes che possono aiutare la scalabilità del cluster. Le astrazioni di Kubernetes, ad esempio namespace e servizi, comportano anche considerazioni architettoniche.

Scalabilità estremamente ampia

Se stai scalando un singolo cluster oltre 1.000 nodi o 50.000 pod, saremo lieti di parlare con te. Ti consigliamo di contattare il tuo team di supporto o il tuo account manager tecnico per contattare specialisti che possono aiutarti a pianificare e scalare oltre le informazioni fornite in questa guida. Amazon EKS può supportare fino a 100.000 nodi in un singolo cluster se sei selezionato per l'onboarding.

Piano di controllo Kubernetes

> **Tip**
>
> Esplora le best practice tramite i workshop Amazon EKS.

Il piano di controllo Kubernetes è composto da Kubernetes API Server, Kubernetes Controller Manager, Scheduler e altri componenti necessari per il funzionamento di Kubernetes. I limiti di scalabilità di questi componenti variano a seconda di ciò che viene eseguito nel cluster, ma le aree

con il maggiore impatto sulla scalabilità includono la versione di Kubernetes, l'utilizzo e la scalabilità dei singoli nodi.

Limita il carico di lavoro e l'interruzione dei nodi

> **Important**
>
> Per evitare di raggiungere i limiti delle API sul piano di controllo, è necessario limitare i picchi di scalabilità che aumentano le dimensioni del cluster con percentuali a due cifre alla volta (ad esempio da 1000 nodi a 1100 nodi o da 4000 a 4500 pod contemporaneamente).

Il piano di controllo EKS si ridimensionerà automaticamente man mano che il cluster cresce, ma ci sono dei limiti alla velocità di scalabilità. Quando crei per la prima volta un cluster EKS, Control Plane non sarà immediatamente in grado di scalare fino a centinaia di nodi o migliaia di pod. Per saperne di più su come EKS ha apportato miglioramenti alla scalabilità, consulta questo post sul blog.

La scalabilità di applicazioni di grandi dimensioni richiede che l'infrastruttura si adatti per diventare completamente pronte (ad esempio, sistemi di bilanciamento del carico di riscaldamento). Per controllare la velocità di scalabilità, assicurati di eseguire il ridimensionamento in base alle metriche corrette per la tua applicazione. Il ridimensionamento della CPU e della memoria potrebbe non prevedere con precisione i vincoli dell'applicazione e l'utilizzo di metriche personalizzate (ad esempio richieste al secondo) in Kubernetes Horizontal Pod Autoscaler (HPA) potrebbe essere un'opzione di scalabilità migliore.

Per utilizzare una metrica personalizzata, consulta gli esempi nella documentazione di Kubernetes. Se hai esigenze di scalabilità più avanzate o devi scalare in base a fonti esterne (ad esempio la coda AWS SQS), usa KEDA per la scalatura del carico di lavoro basata sugli eventi.

Ridimensiona nodi e pod in modo sicuro

Sostituisci le istanze a esecuzione prolungata

La sostituzione regolare dei nodi mantiene il cluster integro evitando variazioni di configurazione e problemi che si verificano solo dopo tempi di attività prolungati (ad esempio perdite di memoria lente). La sostituzione automatizzata ti fornirà procedure e procedure ottimali per gli aggiornamenti dei nodi e le patch di sicurezza. Se ogni nodo del cluster viene sostituito regolarmente, è necessaria una minore fatica per mantenere processi separati per la manutenzione continua.

Usa le impostazioni time to live (TTL) di Karpenter per sostituire le istanze dopo che sono state in esecuzione per un determinato periodo di tempo. I gruppi di nodi autogestiti possono utilizzare l'`max-instance-lifetime`impostazione per ciclare automaticamente i nodi. I gruppi di nodi gestiti non dispongono attualmente di questa funzionalità, ma puoi tenere traccia della richiesta qui GitHub.

Rimuovi i nodi sottoutilizzati

Puoi rimuovere i nodi quando non hanno carichi di lavoro in esecuzione utilizzando la soglia di scalabilità inferiore in Kubernetes Cluster Autoscaler con `--scale-down-utilization-threshold`o in Karpenter puoi utilizzare l'impostazione del provisioner. `ttlSecondsAfterEmpty`

Utilizza i budget per le interruzioni dei pod e lo spegnimento sicuro dei nodi

La rimozione di pod e nodi da un cluster Kubernetes richiede ai controller di aggiornare più risorse (ad esempio). EndpointSlices Questa operazione frequente o troppo rapida può causare limitazioni del server API e interruzioni delle applicazioni man mano che le modifiche si propagano ai controller. I Pod Disruption Budgets sono una best practice per rallentare il tasso di abbandono e proteggere la disponibilità del carico di lavoro man mano che i nodi vengono rimossi o riprogrammati in un cluster.

Usa Client-Side Cache quando esegui Kubectl

L'uso inefficiente del comando kubectl può aggiungere ulteriore carico al Kubernetes API Server. Dovresti evitare di eseguire script o automazioni che utilizzano kubectl ripetutamente (ad esempio in un ciclo for) o di eseguire comandi senza una cache locale.

`kubectl`dispone di una cache lato client che memorizza nella cache le informazioni di rilevamento dal cluster per ridurre la quantità di chiamate API richieste. La cache è abilitata per impostazione predefinita e viene aggiornata ogni 10 minuti.

Se esegui kubectl da un contenitore o senza una cache lato client, potresti riscontrare problemi di limitazione delle API. Si consiglia di conservare la cache del cluster montandola per evitare di effettuare chiamate API non necessarie. `--cache-dir`

Disabilita la compressione kubectl

La disabilitazione della compressione kubectl nel file kubeconfig può ridurre l'utilizzo dell'API e della CPU del client. Per impostazione predefinita, il server comprimerà i dati inviati al client per ottimizzare la larghezza di banda della rete. Ciò aumenta il carico della CPU sul client e sul server per ogni richiesta e la disabilitazione della compressione può ridurre il sovraccarico e la latenza se si dispone

di una larghezza di banda adeguata. Per disabilitare la compressione puoi usare il `--disable-compression=true` flag o il set `disable-compression: true` nel tuo file kubeconfig.

```
apiVersion: v1
clusters:
- cluster:
    server: serverURL
    disable-compression: true
  name: cluster
```

Shard Cluster Autoscaler

Kubernetes Cluster Autoscaler è stato testato per scalare fino a 1000 nodi. Su un cluster di grandi dimensioni con più di 1000 nodi, si consiglia di eseguire più istanze di Cluster Autoscaler in modalità shard. Ogni istanza di Cluster Autoscaler è configurata per scalare un set di gruppi di nodi. L'esempio seguente mostra 2 configurazioni di scalabilità automatica del cluster configurate per ogni gruppo di nodi in scala 4.

ClusterAutoscaler-1

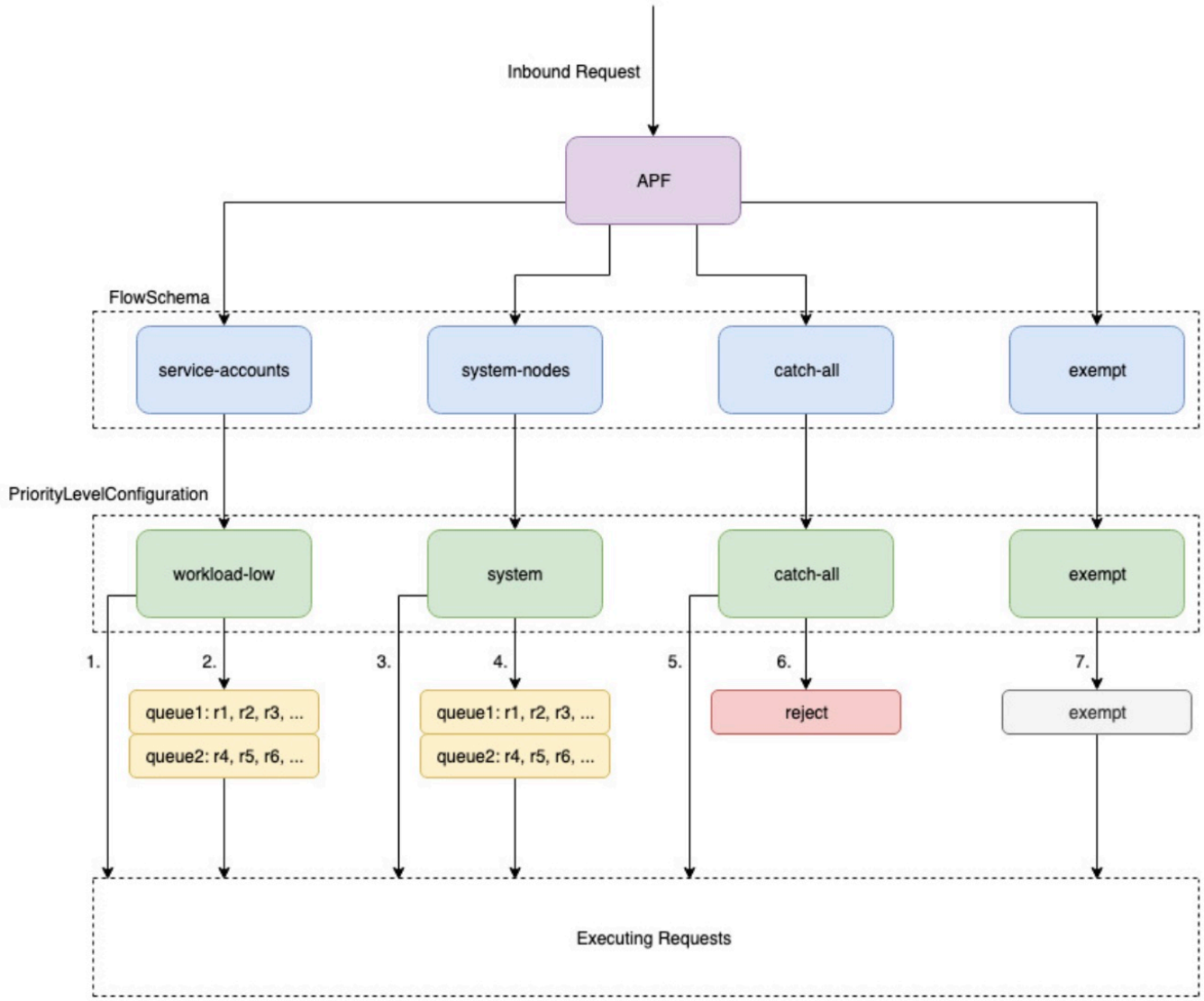
```
autoscalingGroups:
- name: eks-core-node-grp-20220823190924690000000011-80c1660e-030d-476d-cb0d-
d04d585a8fcb
  maxSize: 50
  minSize: 2
- name: eks-data_m1-20220824130553925600000011-5ec167fa-ca93-8ca4-53a5-003e1ed8d306
  maxSize: 450
  minSize: 2
- name: eks-data_m2-20220824130733258600000015-aac167fb-8bf7-429d-d032-e195af4e25f5
  maxSize: 450
  minSize: 2
- name: eks-data_m3-20220824130553914900000003-18c167fa-ca7f-23c9-0fea-f9edefbda002
  maxSize: 450
  minSize: 2
```

ClusterAutoscaler-2

```
autoscalingGroups:
- name: eks-data_m4-20220824130553925500000000f-5ec167fa-ca86-6b83-ae9d-1e07ade3e7c4
  maxSize: 450
  minSize: 2
- name: eks-data_m5-20220824130744542100000017-02c167fb-a1f7-3d9e-a583-43b4975c050c
```

```
  maxSize: 450
  minSize: 2
- name: eks-data_m6-20220824130553924300000000d-9cc167fa-ca94-132a-04ad-e43166cef41f
  maxSize: 450
  minSize: 2
- name: eks-data_m7-20220824130553921000000009-96c167fa-ca91-d767-0427-91c879ddf5af
  maxSize: 450
  minSize: 2
```

Priorità ed equità delle API



1. If workload-low has not reached its concurrency limit, the request will be dispatched immediately.
2. If workload-low has reached its concurrency limit, the request will be queued prior to dispatch.
3. If system has not reached its concurrency limit, the request will be dispatched immediately.
4. If system has reached its concurrency limit, the request will be queued prior to dispatch.
5. If catch-all has not reached its concurrency limit, the request will be dispatched immediately.
6. If catch-all has reached its concurrency limit, the request will be dropped.
7. Requests mapping to exempt will always be dispatched.

Panoramica di

Per proteggersi dal sovraccarico durante i periodi di aumento delle richieste, l'API Server limita il numero di richieste in arrivo che può avere in sospeso in un determinato momento. Una volta

superato questo limite, l'API Server inizierà a rifiutare le richieste e restituirà ai client un codice di risposta HTTP 429 per «Troppe richieste». È preferibile che il server lasci cadere le richieste e che i client riprovino più tardi piuttosto che non avere limiti sul lato server sul numero di richieste e sovraccaricare il piano di controllo, il che potrebbe comportare un peggioramento delle prestazioni o l'indisponibilità.

Il meccanismo utilizzato da Kubernetes per configurare il modo in cui queste richieste in entrata vengono suddivise tra diversi tipi di richiesta si chiama API Priority and Fairness. L'API Server configura il numero totale di richieste in arrivo che può accettare sommando i valori specificati dai flag and. `--max-requests-inflight` `--max-mutating-requests-inflight` EKS utilizza i valori predefiniti di 400 e 200 richieste per questi flag, consentendo l'invio di un totale di 600 richieste in un determinato momento. Tuttavia, poiché ridimensiona il piano di controllo a dimensioni maggiori in risposta all'aumento dell'utilizzo e all'abbandono del carico di lavoro, aumenta di conseguenza la quota di richieste in volo fino al 2000 (soggetta a modifiche). APF specifica in che modo queste quote di richieste in volo sono ulteriormente suddivise tra diversi tipi di richieste. Tieni presente che i piani di controllo EKS sono altamente disponibili con almeno 2 server API registrati in ciascun cluster. Ciò significa che il numero totale di richieste in entrata che il cluster può gestire è il doppio (o superiore se ulteriormente ridimensionato orizzontalmente) rispetto alla quota di volo impostata per kube-apiserver. Ciò equivale a diverse migliaia di cluster EKS più grandi. requests/second

Due tipi di oggetti Kubernetes, chiamati PriorityLevelConfigurations e FlowSchemas, configurano il modo in cui il numero totale di richieste viene suddiviso tra diversi tipi di richieste. Questi oggetti vengono gestiti automaticamente dal server API ed EKS utilizza la configurazione predefinita di questi oggetti per la versione secondaria di Kubernetes specificata. PriorityLevelConfigurations rappresentano una frazione del numero totale di richieste consentite. Ad esempio, al workload-high PriorityLevelConfiguration viene assegnato 98 su un totale di 600 richieste. La somma delle richieste assegnate a tutte PriorityLevelConfigurations sarà pari a 600 (o leggermente superiore a 600 perché il server API arrotonderà per eccesso se a un determinato livello viene concessa una frazione di una richiesta). Per verificare la presenza del PriorityLevelConfigurations cluster e il numero di richieste assegnate a ciascuno, è possibile eseguire il comando seguente. Queste sono le impostazioni predefinite di EKS 1.32:

```
$ kubectl get --raw /metrics | grep apiserver_flowcontrol_nominal_limit_seats
apiserver_flowcontrol_nominal_limit_seats{priority_level="catch-all"} 13
apiserver_flowcontrol_nominal_limit_seats{priority_level="exempt"} 0
apiserver_flowcontrol_nominal_limit_seats{priority_level="global-default"} 49
apiserver_flowcontrol_nominal_limit_seats{priority_level="leader-election"} 25
apiserver_flowcontrol_nominal_limit_seats{priority_level="node-high"} 98
apiserver_flowcontrol_nominal_limit_seats{priority_level="system"} 74
```

```
apiserver_flowcontrol_nominal_limit_seats{priority_level="workload-high"} 98
apiserver_flowcontrol_nominal_limit_seats{priority_level="workload-low"} 245
```

Il secondo tipo di oggetto è. FlowSchemas Le richieste del server API con un determinato set di proprietà sono classificate nello stesso gruppo FlowSchema. Queste proprietà includono l'utente autenticato o gli attributi della richiesta, come il gruppo API, lo spazio dei nomi o la risorsa. A specifica FlowSchema anche a quale PriorityLevelConfiguration tipo di richiesta deve essere mappato. I due oggetti insieme dicono: «Voglio che questo tipo di richiesta venga conteggiata per questa quota di richieste in volo». Quando una richiesta raggiunge il server API, quest'ultimo controllerà ciascuna di esse FlowSchemas finché non ne troverà una che corrisponda a tutte le proprietà richieste. Se più richieste FlowSchemas corrispondono a una richiesta, il server API sceglierà quella FlowSchema con la precedenza di corrispondenza più piccola specificata come proprietà nell'oggetto.

La mappatura di FlowSchemas to PriorityLevelConfigurations può essere visualizzata utilizzando questo comando:

```
$ kubectl get flowschemas
NAME                           PRIORITYLEVEL     MATCHINGPRECEDENCE
 DISTINGUISHERMETHOD   AGE     MISSINGPL
exempt                         exempt            1                  <none>
     7h19m   False
eks-exempt                     exempt            2                  <none>
     7h19m   False
probes                         exempt            2                  <none>
     7h19m   False
system-leader-election         leader-election   100                ByUser
     7h19m   False
endpoint-controller            workload-high     150                ByUser
     7h19m   False
workload-leader-election       leader-election   200                ByUser
     7h19m   False
system-node-high               node-high         400                ByUser
     7h19m   False
system-nodes                   system            500                ByUser
     7h19m   False
kube-controller-manager        workload-high     800                ByNamespace
     7h19m   False
kube-scheduler                 workload-high     800                ByNamespace
     7h19m   False
kube-system-service-accounts   workload-high     900                ByNamespace
     7h19m   False
```

```
eks-workload-high              workload-high     1000                ByUser
     7h14m   False
service-accounts               workload-low      9000                ByUser
     7h19m   False
global-default                 global-default    9900                ByUser
     7h19m   False
catch-all                      catch-all         10000               ByUser
     7h19m   False
```

PriorityLevelConfigurations può avere un tipo di Queue, Reject o Exempt. Per i tipi Queue e Reject, viene applicato un limite al numero massimo di richieste in entrata per quel livello di priorità, tuttavia il comportamento varia quando viene raggiunto tale limite. Ad esempio, il workload-high PriorityLevelConfiguration utilizza il tipo Queue e dispone di 98 richieste che possono essere utilizzate dai controller relativi a controller-manager, endpoint-controller, scheduler, eks e dai pod in esecuzione nello spazio dei nomi del sistema kube. Poiché viene utilizzato il tipo Queue, l'API Server tenterà di conservare le richieste in memoria e spera che il numero di richieste in entrata scenda al di sotto di 98 prima che queste richieste scadano. Se una determinata richiesta scade in coda o se troppe richieste sono già in coda, l'API Server non ha altra scelta che abbandonare la richiesta e restituire al client un 429. Tieni presente che l'accodamento può impedire a una richiesta di ricevere un 429, ma comporta il compromesso di una maggiore latenza sulla richiesta. end-to-end

Consideriamo ora il catch-all FlowSchema che viene mappato al catch-all con il tipo Reject PriorityLevelConfiguration . Se i client raggiungono il limite di 13 richieste in entrata, l'API Server non metterà in coda le richieste e le eliminerà immediatamente con un codice di risposta 429. Infine, le richieste che corrispondono a un PriorityLevelConfiguration tipo Exempt non riceveranno mai un 429 e verranno sempre inviate immediatamente. Viene utilizzato per richieste ad alta priorità come richieste healthz o richieste provenienti dal gruppo system:masters.

Monitoraggio di APF e richieste abbandonate

Per confermare se alcune richieste vengono eliminate a causa di APF, è `apiserver_flowcontrol_rejected_requests_total` possibile monitorare le metriche del server API per verificare l'impatto e. FlowSchemas PriorityLevelConfigurations Ad esempio, questa metrica mostra che 100 richieste provenienti dagli account di servizio FlowSchema sono state eliminate a causa del timeout delle richieste nelle code con un basso carico di lavoro:

```
% kubectl get --raw /metrics | grep apiserver_flowcontrol_rejected_requests_total
apiserver_flowcontrol_rejected_requests_total{flow_schema="service-
accounts",priority_level="workload-low",reason="time-out"} 100
```

Per verificare quanto manca a un determinato numero di 429 secondi la ricezione o se PriorityLevelConfiguration si verifica un aumento della latenza dovuto all'accodamento, puoi confrontare la differenza tra il limite di concorrenza e la concorrenza in uso. In questo esempio, abbiamo un buffer di 100 richieste.

```
% kubectl get --raw /metrics | grep
 'apiserver_flowcontrol_nominal_limit_seats.*workload-low'
apiserver_flowcontrol_nominal_limit_seats{priority_level="workload-low"} 245

% kubectl get --raw /metrics | grep
 'apiserver_flowcontrol_current_executing_seats.*workload-low'
apiserver_flowcontrol_current_executing_seats{flow_schema="service-
accounts",priority_level="workload-low"} 145
```

Per verificare se una determinata richiesta PriorityLevelConfiguration è in coda ma non necessariamente abbandonata, è possibile fare riferimento alla metrica per`apiserver_flowcontrol_current_inqueue_requests`:

```
% kubectl get --raw /metrics | grep
 'apiserver_flowcontrol_current_inqueue_requests.*workload-low'
apiserver_flowcontrol_current_inqueue_requests{flow_schema="service-
accounts",priority_level="workload-low"} 10
```

Altre metriche utili di Prometheus includono:

- apiserver_flowcontrol_dispatched_requests_total
- apiserver_flowcontrol_request_execution_seconds
- apiserver_flowcontrol_request_wait_duration_seconds

Consulta la documentazione originale per un elenco completo delle metriche APF.

Prevenzione delle richieste abbandonate

Evita i 429 cambiando il tuo carico di lavoro

Quando APF interrompe le richieste a causa del PriorityLevelConfiguration superamento del numero massimo consentito di richieste in volo, i clienti della zona interessata FlowSchemas possono ridurre il numero di richieste in esecuzione in un determinato momento. Ciò può essere ottenuto riducendo

il numero totale di richieste effettuate nel periodo in cui si verificano 429 richieste. Tieni presente che le richieste di lunga durata, come le costose chiamate a elenchi, sono particolarmente problematiche perché vengono conteggiate come richieste in volo per l'intera durata dell'esecuzione. Ridurre il numero di queste costose richieste o ottimizzare la latenza di queste chiamate alla lista (ad esempio, riducendo il numero di oggetti recuperati per richiesta o passando all'utilizzo di una richiesta di controllo) può aiutare a ridurre la concorrenza totale richiesta da un determinato carico di lavoro.

Evita i 429 cambiando le impostazioni APF

> **Warning**
>
> Modifica le impostazioni APF predefinite solo se sai cosa stai facendo. Le impostazioni APF non configurate correttamente possono causare l'interruzione delle richieste del server API e gravi interruzioni del carico di lavoro.

Un altro approccio per prevenire la perdita delle richieste consiste nel modificare le impostazioni predefinite FlowSchemas o PriorityLevelConfigurations installarle sui cluster EKS. EKS installa le impostazioni predefinite upstream per FlowSchemas e PriorityLevelConfigurations per la versione secondaria di Kubernetes specificata. Il server API riconcilierà automaticamente questi oggetti ai valori predefiniti se modificati, a meno che la seguente annotazione sugli oggetti non sia impostata su false:

```
metadata:
  annotations:
    apf.kubernetes.io/autoupdate-spec: "false"
```

A un livello elevato, le impostazioni APF possono essere modificate in uno dei seguenti modi:

- Assegna una maggiore capacità in volo alle richieste che ti interessano.
- Isola le richieste non essenziali o costose che possono ridurre la capacità per altri tipi di richieste.

Ciò può essere ottenuto modificando l'impostazione predefinita FlowSchemas PriorityLevelConfigurations e/o creando nuovi oggetti di questo tipo. Gli operatori possono aumentare i valori assuredConcurrencyShares per gli PriorityLevelConfigurations oggetti pertinenti per aumentare la frazione di richieste in volo a cui sono allocati. Inoltre, il numero di richieste che possono essere messe in coda in un determinato momento può essere aumentato se l'applicazione è in grado di gestire la latenza aggiuntiva causata dalle richieste messe in coda prima di essere inviate.

In alternativa, è possibile creare nuovi PriorityLevelConfigurations oggetti FlowSchema e oggetti specifici per il carico di lavoro del cliente. Tieni presente che allocando una quantità maggiore assuredConcurrencyShares a oggetti esistenti PriorityLevelConfigurations o nuovi PriorityLevelConfigurations si ridurrà il numero di richieste che possono essere gestite da altri bucket, poiché il limite complessivo rimarrà di 600 in volo per server API.

Quando si apportano modifiche ai valori predefiniti di APF, queste metriche devono essere monitorate su un cluster non di produzione per garantire che la modifica delle impostazioni non provochi 429 indesiderati:

1. La metrica per `apiserver_flowcontrol_rejected_requests_total` deve essere monitorata per tutti per garantire che nessun bucket cominci FlowSchemas a far cadere le richieste.
2. I valori relativi `apiserver_flowcontrol_nominal_limit_seats` e `apiserver_flowcontrol_current_executing_seats` devono essere confrontati per garantire che la concorrenza in uso non rischi di violare il limite per quel livello di priorità.

Un caso d'uso comune per definire un nuovo FlowSchema e PriorityLevelConfiguration riguarda l'isolamento. Supponiamo di voler isolare le chiamate a eventi di lunga durata a una lista dai pod in base alla rispettiva quota di richieste. Ciò eviterà che le richieste importanti provenienti dai pod che utilizzano gli account di servizio esistenti ricevano 429 richieste e non abbiano più FlowSchema capacità di richiesta. Ricordiamo che il numero totale di richieste in entrata è limitato, tuttavia, questo esempio mostra che le impostazioni APF possono essere modificate per suddividere meglio la capacità delle richieste per un determinato carico di lavoro:

FlowSchema Oggetto di esempio per isolare le richieste di eventi della lista:

```
apiVersion: flowcontrol.apiserver.k8s.io/v1
kind: FlowSchema
metadata:
  name: list-events-default-service-accounts
spec:
  distinguisherMethod:
    type: ByUser
  matchingPrecedence: 8000
  priorityLevelConfiguration:
    name: catch-all
  rules:
  - resourceRules:
```

```
    - apiGroups:
      - '*'
      namespaces:
      - default
      resources:
      - events
      verbs:
      - list
    subjects:
    - kind: ServiceAccount
      serviceAccount:
        name: default
        namespace: default
```

- Questo FlowSchema cattura tutte le chiamate agli eventi di elenco effettuate dagli account di servizio nello spazio dei nomi predefinito.
- La precedenza di corrispondenza 8000 è inferiore al valore di 9000 utilizzato dai service-accounts esistenti, FlowSchema quindi queste chiamate agli eventi dell'elenco corrisponderanno a -accounts anziché service-accounts. list-events-default-service
- Stiamo usando il catch-all PriorityLevelConfiguration per isolare queste richieste. Questo bucket consente di utilizzare solo 13 richieste in arrivo da queste chiamate di eventi di lunga durata. I pod inizieranno a ricevere 429 richieste non appena cercheranno di inviare più di 13 di queste richieste contemporaneamente.

Recupero di risorse nel server API

La raccolta di informazioni dal server API è un comportamento previsto per i cluster di qualsiasi dimensione. Man mano che si aumenta il numero di risorse nel cluster, la frequenza delle richieste e il volume dei dati possono rapidamente diventare un ostacolo per il piano di controllo e portare alla latenza e alla lentezza delle API. A seconda della gravità della latenza, se non si presta attenzione, si verificano tempi di inattività imprevisti.

Essere consapevoli di ciò che si richiede e con quale frequenza sono i primi passi per evitare questo tipo di problemi. Di seguito sono riportate alcune linee guida per limitare il volume delle query in base alle migliori pratiche di scalabilità. I suggerimenti contenuti in questa sezione vengono forniti a partire dalle opzioni note per garantire la scalabilità ottimale.

Usa Shared Informer

Quando si creano controller e sistemi di automazione che si integrano con l'API Kubernetes, spesso è necessario ottenere informazioni dalle risorse Kubernetes. Se effettui regolarmente sondaggi su queste risorse, ciò può causare un carico significativo sul server API.

L'utilizzo di un informatore della libreria client-go ti offrirà il vantaggio di controllare le modifiche alle risorse in base agli eventi anziché effettuare sondaggi per verificare le modifiche. Gli informatori riducono ulteriormente il carico utilizzando la cache condivisa per gli eventi e le modifiche, in modo che più controller che controllano le stesse risorse non aggiungano ulteriore carico.

I controllori dovrebbero evitare di eseguire il polling di risorse a livello di cluster senza etichette e selettori di campo, specialmente in cluster di grandi dimensioni. Ogni sondaggio non filtrato richiede l'invio di molti dati non necessari da etcd tramite il server API per essere filtrati dal client. Filtrando in base a etichette e namespace è possibile ridurre la quantità di lavoro che il server API deve eseguire per soddisfare la richiesta e i dati inviati al client.

Ottimizza l'utilizzo dell'API Kubernetes

Quando si chiama l'API Kubernetes con controller personalizzati o automazione, è importante limitare le chiamate solo alle risorse necessarie. Senza limiti, puoi causare un carico non necessario sul server API e così via.

Si consiglia di utilizzare l'argomento watch ogni volta che è possibile. Senza argomenti, il comportamento predefinito consiste nell'elencare gli oggetti. Per utilizzare watch anziché list puoi aggiungerlo `?watch=true` alla fine della tua richiesta API. Ad esempio, per inserire tutti i pod nello spazio dei nomi predefinito con un orologio, usa:

```
/api/v1/namespaces/default/pods?watch=true
```

Se state elencando oggetti, dovreste limitare l'ambito di ciò che state elencando e la quantità di dati restituiti. È possibile limitare i dati restituiti aggiungendo `limit=500` argomenti alle richieste. L'`fieldSelector`argomento e il `/namespace/` percorso possono essere utili per assicurarsi che gli elenchi abbiano un ambito ristretto quanto necessario. Ad esempio, per elencare solo i pod in esecuzione nello spazio dei nomi predefinito, utilizza il percorso e gli argomenti dell'API seguenti.

```
/api/v1/namespaces/default/pods?fieldSelector=status.phase=Running&limit=500
```

Oppure elenca tutti i pod che funzionano con:

```
/api/v1/pods?fieldSelector=status.phase=Running&limit=500
```

Un'altra opzione per limitare le chiamate di controllo o gli oggetti elencati è quella di utilizzare `resourceVersions`, come puoi leggere nella documentazione di Kubernetes. Senza `resourceVersion` argomenti, riceverai la versione più recente disponibile che richiede una lettura del quorum etcd, che è la lettura più costosa e più lenta per il database. La ResourceVersion dipende dalle risorse che state cercando di interrogare e può essere trovata sul campo. `metadata.resourseVersion` Ciò è consigliato anche nel caso in cui si utilizzino le chiamate watch e non solo le chiamate in elenco

È `resourceVersion=0` disponibile uno speciale che restituirà i risultati dalla cache del server API. Questo può ridurre il carico di etcd ma non supporta l'impaginazione.

```
/api/v1/namespaces/default/pods?resourceVersion=0
```

Si consiglia di utilizzare watch con un valore ResourceVersion impostato come valore noto più recente ricevuto dall'elenco o dall'orologio precedente. Questo viene gestito automaticamente in client-go. Ma si consiglia di ricontrollarlo se si utilizza un client k8s in altre lingue.

```
/api/v1/namespaces/default/pods?watch=true&resourceVersion=362812295
```

Se chiami l'API senza argomenti, sarà la più dispendiosa in termini di risorse per il server API e così via. Questa chiamata otterrà tutti i pod in tutti i namespace senza impaginazione o limitazione dell'ambito e richiederà la lettura del quorum da etcd.

```
/api/v1/pods
```

Evita il fragore delle mandrie DaemonSet

A DaemonSet assicura che tutti (o alcuni) nodi eseguano una copia di un pod. Quando i nodi si uniscono al cluster, il daemonset-controller crea dei pod per quei nodi. Quando i nodi lasciano il cluster, quei pod vengono raccolti nella spazzatura. L'eliminazione di a DaemonSet pulirà i pod che ha creato.

Alcuni usi tipici di a DaemonSet sono:

- Esecuzione di un demone di archiviazione del cluster su ogni nodo
- Esecuzione di un demone di raccolta dei log su ogni nodo

- Esecuzione di un demone di monitoraggio dei nodi su ogni nodo

Nei cluster con migliaia di nodi, la creazione di un nuovo nodo DaemonSet, l'aggiornamento o l'aumento del numero di nodi può comportare un carico elevato sul piano di controllo. DaemonSet Se DaemonSet i pod emettono costose richieste ai server API all'avvio del pod, possono causare un elevato utilizzo di risorse sul piano di controllo a causa di un gran numero di richieste simultanee.

Durante il normale funzionamento, è possibile utilizzare `RollingUpdate` a per garantire l'implementazione graduale di nuovi pod. DaemonSet Con una strategia di `RollingUpdate` aggiornamento, dopo aver aggiornato un DaemonSet modello, il controller elimina i vecchi DaemonSet pod e ne crea automaticamente di nuovi in modo DaemonSet controllato. Al massimo un pod DaemonSet sarà in esecuzione su ciascun nodo durante l'intero processo di aggiornamento. È possibile eseguire un'implementazione graduale `maxUnavailable` impostando su 1, su 0 e `maxSurge` `minReadySeconds` su 60. Se non specifichi una strategia di aggiornamento, per impostazione predefinita Kubernetes creerà un `RollingUpdate` con `maxUnavailable` 1, `maxSurge` 0 e 0. `minReadySeconds`

```
minReadySeconds: 60
strategy:
  type: RollingUpdate
  rollingUpdate:
    maxSurge: 0
    maxUnavailable: 1
```

A `RollingUpdate` assicura l'implementazione graduale di nuovi DaemonSet pod se DaemonSet è già stato creato e ha il numero previsto di pod su tutti i nodi. `Ready` In determinate condizioni, che non sono coperte dalle strategie, possono insorgere problemi legati al Thundering Herd. `RollingUpdate`

Evita che le greggi si agitino al momento della creazione DaemonSet

Per impostazione predefinita, indipendentemente dalla `RollingUpdate` configurazione, il daemonset-controller presente in kube-controller-manager creerà contemporaneamente dei pod per tutti i nodi corrispondenti quando ne crei uno nuovo. DaemonSet Per forzare l'implementazione graduale dei pod dopo aver creato un, puoi usare un o. DaemonSet `NodeSelector` `NodeAffinity` In questo DaemonSet modo verrà creato un oggetto corrispondente a zero nodi, quindi potrai aggiornare gradualmente i nodi per renderli idonei a far funzionare un pod DaemonSet a una velocità controllata. Puoi seguire questo approccio:

- Aggiungi un'etichetta a tutti i nodi per`run-daemonset=false`.

```
kubectl label nodes --all run-daemonset=false
```

- Crea il tuo DaemonSet con un'`NodeAffinity`impostazione che corrisponda a qualsiasi nodo senza `run-daemonset=false` etichetta. Inizialmente, ciò comporterà l' DaemonSet assenza di pod corrispondenti.

```
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
      - matchExpressions:
        - key: run-daemonset
          operator: NotIn
          values:
          - "false"
```

- Rimuovi l'`run-daemonset=false`etichetta dai tuoi nodi a una velocità controllata. Puoi usare questo script bash come esempio:

```
#!/bin/bash

nodes=$(kubectl get --raw "/api/v1/nodes" | jq -r '.items | .[].metadata.name')

for node in ${nodes[@]}; do
   echo "Removing run-daemonset label from node $node"
   kubectl label nodes $node run-daemonset-
   sleep 5
done
```

- Facoltativamente, rimuovi l'`NodeAffinity`impostazione dal tuo DaemonSet oggetto. Nota che questo attiverà anche un contenitore `RollingUpdate` e sostituirà gradualmente tutti i DaemonSet pod esistenti a causa della modifica del DaemonSet modello.

Evita che le masse si agitino grazie alla scalabilità orizzontale dei nodi

Analogamente alla DaemonSet creazione, la creazione di nuovi nodi a una velocità elevata può comportare l'avvio simultaneo di un gran numero di pod. DaemonSet È necessario creare nuovi nodi a una velocità controllata in modo che il controller crei DaemonSet i pod alla stessa velocità. Se ciò non è possibile, puoi rendere inizialmente i nuovi nodi non idonei per quelli esistenti DaemonSet utilizzando. `NodeAffinity` Successivamente, potete aggiungere gradualmente un'etichetta ai nuovi nodi in modo che il daemonset-controller crei i pod a una velocità controllata. Puoi seguire questo approccio:

- Aggiungi un'etichetta a tutti i nodi esistenti per `run-daemonset=true`

```
kubectl label nodes --all run-daemonset=true
```

- Aggiorna il tuo DaemonSet con un'`NodeAffinity`impostazione che corrisponda a qualsiasi nodo con un'`run-daemonset=true`etichetta. Nota che questo attiverà anche un pod `RollingUpdate` e sostituirà gradualmente tutti i DaemonSet pod esistenti perché il DaemonSet modello è cambiato. È necessario attendere il `RollingUpdate` completamento dell'operazione prima di passare alla fase successiva.

```
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
      - matchExpressions:
        - key: run-daemonset
          operator: In
          values:
          - "true"
```

- Crea nuovi nodi nel tuo cluster. Nota che questi nodi non avranno l'`run-daemonset=true`etichetta, quindi non DaemonSet corrisponderanno a quei nodi.
- Aggiungi l'`run-daemonset=true`etichetta ai nuovi nodi (che attualmente non hanno l'`run-daemonset`etichetta) a una velocità controllata. Puoi usare questo script bash come esempio:

```
#!/bin/bash
```

```
nodes=$(kubectl get --raw "/api/v1/nodes?labelSelector=%21run-daemonset" | jq -r
 '.items | .[].metadata.name')

for node in ${nodes[@]}; do
   echo "Adding run-daemonset=true label to node $node"
   kubectl label nodes $node run-daemonset=true
   sleep 5
done
```

- Facoltativamente, rimuovi l'`NodeAffinity`impostazione dal tuo DaemonSet oggetto e rimuovi l'`run-daemonset`etichetta da tutti i nodi.

Evita che le mandrie si agitino durante gli aggiornamenti DaemonSet

Una `RollingUpdate` politica rispetterà solo l'`maxUnavailable`impostazione dei DaemonSet pod che lo sono. `Ready` Se a DaemonSet ha solo `NotReady` pod o una grande percentuale di `NotReady` pod e ne aggiorni il modello, il daemonset-controller creerà nuovi pod contemporaneamente per tutti i pod. `NotReady` Ciò può causare problemi di Thundering Herd in presenza di un numero significativo di pod, ad esempio se i `NotReady` pod si bloccano continuamente o non riescono a estrarre immagini.

Per forzare l'implementazione graduale dei pod quando si aggiorna un contenitore DaemonSet e se ne sono presenti, è possibile modificare temporaneamente la strategia di aggiornamento in modalità `NotReady` from to. DaemonSet `RollingUpdate` `OnDelete` Con`OnDelete`, dopo aver aggiornato un DaemonSet modello, il controller crea nuovi pod dopo aver eliminato manualmente quelli vecchi, in modo da poter controllare l'implementazione di nuovi pod. Puoi seguire questo approccio:

- Controlla se ci sono dei `NotReady` pod nel tuo DaemonSet.
- In caso contrario, puoi aggiornare il DaemonSet modello in tutta sicurezza e la `RollingUpdate` strategia garantirà un'implementazione graduale.
- In caso affermativo, dovresti prima aggiornare il tuo DaemonSet per utilizzare la `OnDelete` strategia.

```
updateStrategy:
  type: OnDelete
```

- Successivamente, aggiorna il DaemonSet modello con le modifiche necessarie.

- Dopo questo aggiornamento, puoi eliminare i vecchi DaemonSet pod inviando richieste di eliminazione dei pod a una frequenza controllata. Puoi usare questo script bash come esempio in cui il DaemonSet nome è fluentd-elasticsearch nello spazio dei nomi kube-system:

```
#!/bin/bash

daemonset_pods=$(kubectl get --raw "/api/v1/namespaces/kube-system/pods?
labelSelector=name%3Dfluentd-elasticsearch" | jq -r '.items | .[].metadata.name')

for pod in ${daemonset_pods[@]}; do
   echo "Deleting pod $pod"
   kubectl delete pod $pod -n kube-system
   sleep 5
done
```

- Infine, puoi tornare alla strategia precedente. DaemonSet `RollingUpdate`

Piano dati Kubernetes

La selezione dei tipi di istanze EC2 è probabilmente una delle decisioni più difficili che i clienti devono affrontare, perché si trovano in cluster con più carichi di lavoro. Non esiste una soluzione completa. one-size-fits Ecco alcuni suggerimenti per aiutarti a evitare le insidie più comuni legate alla scalabilità del calcolo.

Scalabilità automatica dei nodi

Ti consigliamo di utilizzare la scalabilità automatica dei nodi che riduce la fatica e si integra profondamente con Kubernetes. I gruppi di nodi gestiti e Karpenter sono consigliati per cluster su larga scala.

I gruppi di nodi gestiti ti offriranno la flessibilità dei gruppi Amazon EC2 Auto Scaling con vantaggi aggiuntivi per gli upgrade e la configurazione gestiti. Può essere scalato con Kubernetes Cluster Autoscaler ed è un'opzione comune per i cluster che hanno diverse esigenze di elaborazione.

Karpenter è un autoscaler di nodi open source e nativo del carico di lavoro creato da AWS. Ridimensiona i nodi di un cluster in base ai requisiti del carico di lavoro per le risorse (ad esempio GPU) e ai limiti e alle tolleranze (ad esempio la diffusione delle zone) senza gestire gruppi di nodi. I nodi vengono creati direttamente da EC2, il che evita le quote predefinite dei gruppi di nodi (450 nodi

per gruppo) e offre una maggiore flessibilità nella selezione delle istanze con un minore sovraccarico operativo. Consigliamo ai clienti di utilizzare Karpenter quando possibile.

Utilizza molti tipi di istanze EC2 diversi

Ogni regione AWS ha un numero limitato di istanze disponibili per tipo di istanza. Se crei un cluster che utilizza un solo tipo di istanza e riduci il numero di nodi oltre la capacità della regione, riceverai un messaggio di errore indicante che non sono disponibili istanze. Per evitare questo problema, non è necessario limitare arbitrariamente il tipo di istanze che possono essere utilizzate nel cluster.

Per impostazione predefinita, Karpenter utilizzerà un'ampia gamma di tipi di istanze compatibili e sceglierà un'istanza al momento del provisioning in base ai requisiti, alla disponibilità e ai costi del carico di lavoro in sospeso. È possibile ampliare l'elenco dei tipi di istanza utilizzati nella chiave di. `karpenter.k8s.aws/instance-category` NodePools

Kubernetes Cluster Autoscaler richiede che i gruppi di nodi abbiano dimensioni simili in modo che possano essere scalati in modo coerente. È necessario creare più gruppi in base alle dimensioni della CPU e della memoria e scalarli in modo indipendente. Usa il selettore di istanze ec2-instance-selector per identificare istanze di dimensioni simili per i tuoi gruppi di nodi.

```
ec2-instance-selector --service eks --vcpus-min 8 --memory-min 16
a1.2xlarge
a1.4xlarge
a1.metal
c4.4xlarge
c4.8xlarge
c5.12xlarge
c5.18xlarge
c5.24xlarge
c5.2xlarge
c5.4xlarge
c5.9xlarge
c5.metal
```

Preferisci nodi più grandi per ridurre il carico del server API

Al momento di decidere quali tipi di istanze utilizzare, un numero inferiore di nodi di grandi dimensioni comporterà un carico inferiore sul piano di controllo di Kubernetes perché ci saranno meno kubelet e saranno in esecuzione. DaemonSets Tuttavia, i nodi di grandi dimensioni potrebbero non essere

utilizzati completamente come i nodi più piccoli. Le dimensioni dei nodi devono essere valutate in base alla disponibilità del carico di lavoro e ai requisiti di scalabilità.

Un cluster con tre istanze u-24tb1.metal (24 TB di memoria e 448 core) ha 3 kubelet e per impostazione predefinita sarebbe limitato a 110 pod per nodo. Se i tuoi pod utilizzano 4 core ciascuno, ciò potrebbe essere previsto (4 core x 110 = 440 core/nodo). Con un cluster a 3 nodi, la capacità di gestire un incidente di istanza sarebbe ridotta perché un'interruzione di un'istanza potrebbe influire su 1/3 del cluster. È necessario specificare i requisiti dei nodi e la distribuzione dei pod nei carichi di lavoro in modo che lo scheduler Kubernetes possa posizionare correttamente i carichi di lavoro.

I carichi di lavoro dovrebbero definire le risorse di cui hanno bisogno e la disponibilità richiesta tramite contaminazioni, tolleranze e. PodTopologySpread Dovrebbero preferire i nodi più grandi che possano essere utilizzati appieno e soddisfare gli obiettivi di disponibilità per ridurre il carico sul piano di controllo, ridurre le operazioni e ridurre i costi.

Kubernetes Scheduler cercherà automaticamente di distribuire i carichi di lavoro tra zone di disponibilità e host, se le risorse sono disponibili. Se non è disponibile alcuna capacità, Kubernetes Cluster Autoscaler tenterà di aggiungere nodi in ciascuna zona di disponibilità in modo uniforme. Karpenter tenterà di aggiungere nodi nel modo più rapido ed economico possibile, a meno che il carico di lavoro non specifichi altri requisiti.

Per forzare la diffusione dei carichi di lavoro con lo scheduler e la creazione di nuovi nodi tra le zone di disponibilità, è necessario utilizzare: topologySpreadConstraints

```
spec:
  topologySpreadConstraints:
    - maxSkew: 3
      topologyKey: "topology.kubernetes.io/zone"
      whenUnsatisfiable: ScheduleAnyway
      labelSelector:
        matchLabels:
          dev: my-deployment
    - maxSkew: 2
      topologyKey: "kubernetes.io/hostname"
      whenUnsatisfiable: ScheduleAnyway
      labelSelector:
        matchLabels:
          dev: my-deployment
```

Utilizza nodi di dimensioni simili per prestazioni costanti del carico di lavoro

I carichi di lavoro devono definire le dimensioni dei nodi su cui devono essere eseguiti per consentire prestazioni costanti e scalabilità prevedibile. Un carico di lavoro che richiede 500 m di CPU avrà prestazioni diverse su un'istanza con 4 core rispetto a un'istanza con 16 core. Evita i tipi di istanze che utilizzano CPUs burstable come le istanze della serie T.

Per garantire prestazioni costanti ai carichi di lavoro, un carico di lavoro può utilizzare le etichette Karpenter supportate per definire istanze di dimensioni specifiche.

```
kind: deployment
...
spec:
  template:
    spec:
    containers:
    nodeSelector:
      karpenter.k8s.aws/instance-size: 8xlarge
```

I carichi di lavoro pianificati in un cluster con Kubernetes Cluster Autoscaler devono abbinare un selettore di nodi ai gruppi di nodi in base alla corrispondenza delle etichette.

```
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: eks.amazonaws.com/nodegroup
            operator: In
            values:
            - 8-core-node-group    # match your node group name
```

Usa le risorse di elaborazione in modo efficiente

Le risorse di calcolo includono istanze EC2 e zone di disponibilità. L'uso efficace delle risorse di elaborazione aumenterà la scalabilità, la disponibilità, le prestazioni e ridurrà i costi totali. L'utilizzo efficiente delle risorse è estremamente difficile da prevedere in un ambiente di scalabilità automatica con più applicazioni. Karpenter è stato creato per fornire istanze su richiesta in base alle esigenze del carico di lavoro per massimizzare l'utilizzo e la flessibilità.

Karpenter consente ai carichi di lavoro di dichiarare il tipo di risorse di elaborazione di cui hanno bisogno senza prima creare gruppi di nodi o configurare tag di etichette per nodi specifici. Per ulteriori informazioni, consulta le best practice di Karpenter. Valuta la possibilità di abilitare il consolidamento nel tuo provider Karpenter per sostituire i nodi sottoutilizzati.

Automatizza gli aggiornamenti di Amazon Machine Image (AMI)

Mantenendo aggiornati i componenti del Worker Node ti assicurerai di disporre delle patch di sicurezza più recenti e delle funzionalità compatibili con l'API Kubernetes. L'aggiornamento del kubelet è il componente più importante per la funzionalità di Kubernetes, ma l'automazione del sistema operativo, del kernel e delle patch delle applicazioni installate localmente ridurrà la manutenzione man mano che si aumenta la scalabilità.

Si consiglia di utilizzare l'ultima AMI Bottlerocket ottimizzata per Amazon EKS o Amazon EKS ottimizzata per Amazon EKS per l'immagine del nodo. Karpenter utilizzerà automaticamente l'ultima AMI disponibile per effettuare il provisioning di nuovi nodi nel cluster. I gruppi di nodi gestiti aggiorneranno l'AMI durante un aggiornamento del gruppo di nodi ma non aggiorneranno l'ID AMI al momento del provisioning del nodo.

Per i gruppi di nodi gestiti è necessario aggiornare il modello di lancio di Auto Scaling Group (ASG) con nuove AMI IDs quando sono disponibili per il rilascio delle patch. Le versioni minori dell'AMI (ad esempio da 1.23.5 a 1.24.3) saranno disponibili nella console EKS e nell'API come aggiornamenti per il gruppo di nodi. Le versioni di rilascio delle patch (ad esempio da 1.23.5 a 1.23.6) non verranno presentate come aggiornamenti per i gruppi di nodi. Se desideri mantenere aggiornato il tuo gruppo di nodi con le versioni delle patch AMI, devi creare una nuova versione del modello di avvio e lasciare che il gruppo di nodi sostituisca le istanze con la nuova versione AMI.

Puoi trovare l'AMI più recente disponibile da questa pagina o utilizzare la CLI di AWS.

```
aws ssm get-parameter \
  --name /aws/service/eks/optimized-ami/1.24/amazon-linux-2/recommended/image_id \
  --query "Parameter.Value" \
  --output text
```

Usa più volumi EBS per i contenitori

I volumi EBS hanno una quota input/output (I/O) in base al tipo di volume (ad esempio gp3) e alla dimensione del disco. Se le tue applicazioni condividono un singolo volume root EBS con l'host, ciò può esaurire la quota di disco per l'intero host e far sì che le altre applicazioni attendano la capacità

disponibile. Le applicazioni scrivono su disco se scrivono file nella partizione overlay, montano un volume locale dall'host e anche quando si connettono allo standard out (STDOUT) a seconda dell'agente di registrazione utilizzato.

Per evitare l' I/O esaurimento del disco, è necessario montare un secondo volume nella cartella dello stato del contenitore (ad esempio /run/containerd), utilizzare volumi EBS separati per l'archiviazione del carico di lavoro e disabilitare la registrazione locale non necessaria.

Per montare un secondo volume sulle tue istanze EC2 usando eksctl puoi usare un gruppo di nodi con questa configurazione:

```
managedNodeGroups:
  - name: al2-workers
    amiFamily: AmazonLinux2
    desiredCapacity: 2
    volumeSize: 80
    additionalVolumes:
      - volumeName: '/dev/sdz'
        volumeSize: 100
    preBootstrapCommands:
    - |
      "systemctl stop containerd"
      "mkfs -t ext4 /dev/nvme1n1"
      "rm -rf /var/lib/containerd/*"
      "mount /dev/nvme1n1 /var/lib/containerd/"
      "systemctl start containerd"
```

Se stai usando terraform per effettuare il provisioning dei tuoi gruppi di nodi, consulta gli esempi in EKS Blueprints for terraform. Se si utilizza Karpenter per il provisioning dei nodi, è possibile utilizzare i dati utente del nodo per `blockDeviceMappings`aggiungere volumi aggiuntivi.

Per montare un volume EBS direttamente sul tuo pod devi usare il driver CSI AWS EBS e consumare un volume con una classe di storage.

```
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ebs-sc
provisioner: ebs.csi.aws.com
volumeBindingMode: WaitForFirstConsumer
---
```

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: ebs-claim
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: ebs-sc
  resources:
    requests:
      storage: 4Gi
---
apiVersion: v1
kind: Pod
metadata:
  name: app
spec:
  containers:
  - name: app
    image: public.ecr.aws/docker/library/nginx
    volumeMounts:
    - name: persistent-storage
      mountPath: /data
  volumes:
  - name: persistent-storage
    persistentVolumeClaim:
      claimName: ebs-claim
```

Evita le istanze con limiti di collegamento EBS bassi se i carichi di lavoro utilizzano volumi EBS

EBS è uno dei modi più semplici per i carichi di lavoro di avere uno storage persistente, ma presenta anche limitazioni di scalabilità. Ogni tipo di istanza ha un numero massimo di volumi EBS che possono essere collegati. I carichi di lavoro devono dichiarare su quali tipi di istanza devono essere eseguiti e limitare il numero di repliche su una singola istanza con caratteristiche Kubernetes.

Disabilita la registrazione non necessaria su disco

Evitate la registrazione locale non necessaria evitando di eseguire le applicazioni con la registrazione di debug in produzione e disabilitando la registrazione che legge e scrive frequentemente su disco. Journald è il servizio di registrazione locale che mantiene un buffer di log in memoria e lo scarica

periodicamente sul disco. Journald è preferito a syslog che registra immediatamente ogni riga su disco. La disabilitazione di syslog riduce anche la quantità totale di spazio di archiviazione necessaria ed evita la necessità di complicate regole di rotazione dei log. Per disabilitare syslog puoi aggiungere il seguente frammento alla tua configurazione cloud-init:

```
runcmd:
  - [ systemctl, disable, --now, syslog.service ]
```

Istanze di patch disponibili quando la velocità di aggiornamento del sistema operativo è una necessità

> **Important**
>
> L'applicazione di patch alle istanze esistenti deve essere eseguita solo quando necessario. Amazon consiglia di trattare l'infrastruttura come immutabile e di testare accuratamente gli aggiornamenti promossi negli ambienti inferiori allo stesso modo delle applicazioni. Questa sezione si applica quando ciò non è possibile.

Bastano pochi secondi per installare un pacchetto su un host Linux esistente senza interrompere i carichi di lavoro containerizzati. Il pacchetto può essere installato e convalidato senza isolare, svuotare o sostituire l'istanza.

Per sostituire un'istanza devi prima crearne, convalidarla e distribuirne una nuova. AMIs È necessario creare un'istanza sostitutiva e la vecchia istanza deve essere isolata e drenata. Quindi i carichi di lavoro devono essere creati sulla nuova istanza, verificati e ripetuti per tutte le istanze che devono essere corrette. Occorrono ore, giorni o settimane per sostituire le istanze in modo sicuro senza interrompere i carichi di lavoro.

Amazon consiglia di utilizzare un'infrastruttura immutabile costruita, testata e promossa a partire da un sistema dichiarativo automatizzato, ma se hai l'esigenza di applicare rapidamente le patch ai sistemi, dovrai applicare patch ai sistemi esistenti e sostituirli non appena ne saranno AMIs disponibili di nuovi. A causa dell'ampio differenziale di tempo tra l'applicazione delle patch e la sostituzione dei sistemi, consigliamo di utilizzare AWS Systems Manager Patch Manager per automatizzare l'applicazione delle patch ai nodi quando necessario.

I nodi di patching ti consentiranno di implementare rapidamente gli aggiornamenti di sicurezza e sostituire le istanze a intervalli regolari dopo l'aggiornamento dell'AMI. Se utilizzi un sistema operativo

con un file system root di sola lettura come Flatcar Container Linux o Bottlerocket OS, ti consigliamo di utilizzare gli operatori di aggiornamento che funzionano con tali sistemi operativi. L'operatore di aggiornamento Flatcar Linux e l'operatore di aggiornamento Bottlerocket riavvieranno le istanze per mantenere i nodi aggiornati automaticamente.

Servizi cluster

I servizi cluster vengono eseguiti all'interno di un cluster EKS, ma non sono carichi di lavoro degli utenti. Se disponi di un server Linux, spesso devi eseguire servizi come NTP, syslog e un container runtime per supportare i tuoi carichi di lavoro. I servizi cluster sono simili e supportano servizi che consentono di automatizzare e gestire il cluster. In Kubernetes questi vengono generalmente eseguiti nello spazio dei nomi del sistema Kube e alcuni vengono eseguiti come. DaemonSets

Si prevede che i servizi cluster abbiano un tempo di attività elevato e sono spesso fondamentali durante le interruzioni e per la risoluzione dei problemi. Se un servizio cluster di base non è disponibile, si rischia di perdere l'accesso ai dati che possono aiutare a ripristinare o prevenire un'interruzione (ad esempio un elevato utilizzo del disco). Dovrebbero essere eseguiti su istanze di calcolo dedicate come un gruppo di nodi separato o AWS Fargate. Ciò garantirà che i servizi del cluster non vengano influenzati sulle istanze condivise da carichi di lavoro che potrebbero aumentare o utilizzare più risorse.

Scalare CoredNS

La scalabilità di CoredNS ha due meccanismi principali. Riduzione del numero di chiamate al servizio CoredNS e aumento del numero di repliche.

Riduci le interrogazioni esterne abbassando gli ndots

L'impostazione ndots specifica quanti punti (noti anche come «punti») in un nome di dominio sono considerati sufficienti per evitare l'interrogazione del DNS. Se l'applicazione ha un'impostazione ndots pari a 5 (impostazione predefinita) e si richiedono risorse da un dominio esterno come api.example.com (2 punti), CoredNS verrà interrogato per ogni dominio di ricerca definito in /etc/resolv.conf per un dominio più specifico. Per impostazione predefinita, verranno cercati i seguenti domini prima di effettuare una richiesta esterna.

```
api.example.<namespace>.svc.cluster.local
api.example.svc.cluster.local
api.example.cluster.local
api.example.<region>.compute.internal
```

`region`I valori `namespace` and verranno sostituiti con lo spazio dei nomi dei carichi di lavoro e l'area di calcolo. Potresti avere domini di ricerca aggiuntivi in base alle impostazioni del cluster.

È possibile ridurre il numero di richieste a CoredNS abbassando l'opzione ndots del carico di lavoro o qualificando completamente le richieste di dominio includendo un trailing. (ad es.). `api.example.com.` Se il carico di lavoro si connette a servizi esterni tramite DNS, ti consigliamo di impostare ndots su 2 in modo che i carichi di lavoro non rendano superflue le query DNS del cluster. Puoi impostare un server DNS e un dominio di ricerca diversi se il carico di lavoro non richiede l'accesso ai servizi all'interno del cluster.

```
spec:
  dnsPolicy: "None"
  dnsConfig:
    options:
      - name: ndots
        value: "2"
      - name: edns0
```

Se riduci ndots a un valore troppo basso o i domini a cui ti stai connettendo non includono una specificità sufficiente (incluso il termine finale), è possibile che le ricerche DNS abbiano esito negativo. Assicurati di testare l'impatto di questa impostazione sui tuoi carichi di lavoro.

Scalabilità orizzontale di CoredNS

Le istanze CoredNS possono essere scalate aggiungendo repliche aggiuntive alla distribuzione. Si consiglia di utilizzare il NodeLocal DNS o l'autoscaler proporzionale del cluster per scalare CoredNS.

NodeLocal Il DNS richiederà l'esecuzione di un'istanza per nodo, come a sé, il DaemonSet che richiede più risorse di elaborazione nel cluster, ma eviterà le richieste DNS non riuscite e ridurrà il tempo di risposta per le query DNS nel cluster. L'autoscaler proporzionale del cluster scalerà CoredNS in base al numero di nodi o core nel cluster. Questa non è una correlazione diretta alle query di richiesta, ma può essere utile a seconda dei carichi di lavoro e delle dimensioni del cluster. La scala proporzionale predefinita prevede l'aggiunta di una replica aggiuntiva per ogni 256 core o 16 nodi del cluster, a seconda dell'evento che si verifica per primo.

Se utilizzi il componente aggiuntivo CoredNS EKS, valuta la possibilità di abilitare l'opzione di scalabilità automatica. L'autoscaler CoreDNS regola dinamicamente il numero di repliche CoreDNS monitorando il numero di nodi e i core della CPU, utilizzando una formula che utilizza il massimo di (nodi÷16) o (core CPU ÷256), aumentando immediatamente quando necessario e diminuendo gradualmente per mantenere la stabilità.

Scala verticalmente il server Kubernetes Metrics

Il Kubernetes Metrics Server supporta il ridimensionamento orizzontale e verticale. Scalando orizzontalmente il Metrics Server sarà altamente disponibile, ma non verrà scalato orizzontalmente per gestire più metriche del cluster. Dovrai scalare verticalmente il Metrics Server in base ai relativi consigli man mano che i nodi e le metriche raccolte vengono aggiunti al cluster.

Il Metrics Server conserva i dati raccolti, aggregati e serviti in memoria. Man mano che un cluster cresce, aumenta la quantità di dati archiviati dal Metrics Server. Nei cluster di grandi dimensioni, il Metrics Server richiederà più risorse di elaborazione rispetto alla memoria e alla prenotazione della CPU specificate nell'installazione predefinita. È possibile utilizzare Vertical Pod Autoscaler (VPA) o Addon Resizer per ridimensionare il Metrics Server. L'Addon Resizer si ridimensiona verticalmente in proporzione ai nodi di lavoro e alle scale VPA in base all'utilizzo della CPU e della memoria.

Durata CoredNS lameduck

I pod utilizzano il servizio per la risoluzione dei nomi. `kube-dns` Kubernetes utilizza il NAT di destinazione (DNAT) per reindirizzare il traffico `kube-dns` dai nodi ai pod di backend CoredNS. Man mano che scalate la distribuzione di CoreDNS`kube-proxy`, aggiorna le regole e le catene di iptables sui nodi per reindirizzare il traffico DNS verso i pod CoreDNS. La propagazione di nuovi endpoint durante la scalabilità verticale e l'eliminazione delle regole quando si ridimensiona CoredNS possono richiedere da 1 a 10 secondi a seconda delle dimensioni del cluster.

Questo ritardo di propagazione può causare errori di ricerca DNS quando un pod CoreDNS viene terminato ma le regole iptables del nodo non sono state aggiornate. In questo scenario, il nodo può continuare a inviare query DNS a un CoredNS Pod terminato.

Puoi ridurre gli errori di ricerca DNS impostando una durata lameduck nei tuoi pod CoredNS. In modalità lameduck, CoredNS continuerà a rispondere alle richieste in volo. L'impostazione di una durata lameduck ritarderà il processo di spegnimento di CoredNS, dando ai nodi il tempo necessario per aggiornare le regole e le catene di iptables.

Si consiglia di impostare la durata di CoredNS lameduck su 30 secondi.

Sonda di prontezza CoredNS

Si consiglia di utilizzare `/ready` al posto della sonda di prontezza di `/health` CoreDNS.

In linea con la precedente raccomandazione di impostare la durata del lameduck su 30 secondi, garantendo un tempo sufficiente per aggiornare le regole iptables del nodo prima della chiusura del

pod, l'utilizzo `/ready` invece della sonda di preparazione CoreDNS assicura che il pod CoredNS sia completamente pronto all'avvio `/health` per rispondere tempestivamente alle richieste DNS.

```
readinessProbe:
  httpGet:
    path: /ready
    port: 8181
    scheme: HTTP
```

Per ulteriori informazioni sul plugin CoredNS Ready, fare riferimento a https://coredns. io/plugins/ready/

Agenti di registrazione e monitoraggio

Gli agenti di registrazione e monitoraggio possono aggiungere un carico significativo al piano di controllo del cluster perché interrogano il server API per arricchire i log e le metriche con i metadati del carico di lavoro. L'agente su un nodo ha accesso solo alle risorse del nodo locale per visualizzare elementi come il contenitore e il nome del processo. Interrogando il server API, può aggiungere ulteriori dettagli come il nome e le etichette della distribuzione Kubernetes. Ciò può essere estremamente utile per la risoluzione dei problemi, ma compromettere la scalabilità.

Poiché esistono così tante opzioni diverse per la registrazione e il monitoraggio, non possiamo mostrare esempi per tutti i provider. Con fluentbit consigliamo di abilitare Use_Kubelet per recuperare i metadati dal kubelet locale anziché dal server API Kubernetes e di impostare un numero che riduca le chiamate ripetute quando i dati possono essere memorizzati nella cache (`Kube_Meta_Cache_TTL`ad esempio 60).

Il monitoraggio e la registrazione della scalabilità hanno due opzioni generali:

- Disattiva le integrazioni
- Campionamento e filtraggio

La disabilitazione delle integrazioni spesso non è un'opzione perché si perdono i metadati di registro. Ciò elimina il problema della scalabilità delle API, ma introdurrà altri problemi in quanto non si dispone dei metadati richiesti quando necessario.

Il campionamento e il filtraggio riducono il numero di metriche e registri raccolti. Ciò ridurrà la quantità di richieste all'API Kubernetes e ridurrà la quantità di spazio di archiviazione necessaria per le metriche e i log raccolti. La riduzione dei costi di archiviazione ridurrà il costo dell'intero sistema.

La capacità di configurare il campionamento dipende dal software dell'agente e può essere implementata in diversi punti di ingestione. È importante aggiungere il campionamento il più vicino possibile all'agente, perché è probabilmente lì che avvengono le chiamate al server API. Contatta il tuo provider per saperne di più sul supporto per il campionamento.

Se si utilizza CloudWatch and CloudWatch Logs, è possibile aggiungere il filtraggio degli agenti utilizzando i modelli descritti nella documentazione.

Per evitare la perdita di log e metriche, è consigliabile inviare i dati a un sistema in grado di memorizzare i dati nel buffer in caso di interruzione sull'endpoint ricevente. Con fluentbit puoi usare Amazon Kinesis Data Firehose per conservare temporaneamente i dati, il che può ridurre la possibilità di sovraccaricare la posizione di archiviazione finale dei dati.

Carichi di lavoro

I carichi di lavoro hanno un impatto sulla scalabilità del cluster. I carichi di lavoro che utilizzano APIs molto Kubernetes limiteranno la quantità totale di carichi di lavoro che puoi avere in un singolo cluster, ma ci sono alcune impostazioni predefinite che puoi modificare per ridurre il carico.

I carichi di lavoro in un cluster Kubernetes hanno accesso a funzionalità che si integrano con l'API Kubernetes (ad esempio Secrets and ServiceAccounts), ma queste funzionalità non sono sempre obbligatorie e devono essere disabilitate se non vengono utilizzate. La limitazione dell'accesso ai carichi di lavoro e della dipendenza dal piano di controllo di Kubernetes aumenterà il numero di carichi di lavoro eseguibili nel cluster e migliorerà la sicurezza dei cluster rimuovendo l'accesso non necessario ai carichi di lavoro e implementando pratiche con privilegi minimi. Leggi le best practice di sicurezza per ulteriori informazioni.

Utilizzare IPv6 per il collegamento in rete tramite pod

Non è possibile effettuare la transizione di un VPC da uno all'altro, IPv4 IPv6 abilitarlo IPv6 prima di effettuare il provisioning di un cluster è importante. Se IPv6 abiliti un VPC, ciò non significa che devi utilizzarlo e se i tuoi pod e servizi lo utilizzano, IPv6 puoi comunque indirizzare il traffico da e verso gli indirizzi. IPv4 Per ulteriori informazioni, consulta le best practice di rete EKS.

L'utilizzo IPv6 nel cluster evita alcuni dei limiti più comuni di scalabilità dei cluster e dei carichi di lavoro. IPv6 evita l'esaurimento degli indirizzi IP laddove non sia possibile creare pod e nodi perché non è disponibile alcun indirizzo IP. Inoltre, presenta miglioramenti delle prestazioni per nodo, poiché i pod ricevono gli indirizzi IP più velocemente riducendo il numero di allegati ENI per nodo. Puoi

ottenere prestazioni del nodo simili utilizzando la modalità IPv4 prefisso nel VPC CNI, ma devi comunque assicurarti di avere abbastanza indirizzi IP disponibili nel VPC.

Limita il numero di servizi per namespace

Il numero massimo di servizi in un namespace è 5.000 e il numero massimo di servizi in un cluster è 10.000. Per organizzare carichi di lavoro e servizi, aumentare le prestazioni ed evitare l'impatto a cascata sulle risorse con ambito namespace, consigliamo di limitare il numero di servizi per namespace a 500.

Il numero di regole delle tabelle IP create per nodo con kube-proxy aumenta con il numero totale di servizi nel cluster. La generazione di migliaia di regole di tabelle IP e il routing dei pacchetti tramite tali regole hanno un impatto sulle prestazioni dei nodi e aggiungono latenza di rete.

Crea namespace Kubernetes che comprendano un singolo ambiente applicativo purché il numero di servizi per namespace sia inferiore a 500. Ciò consentirà di mantenere l'individuazione dei servizi sufficientemente ridotta da evitare i limiti di individuazione dei servizi e può anche aiutarti a evitare le collisioni di denominazione dei servizi. Gli ambienti applicativi (ad esempio dev, test, prod) devono utilizzare cluster EKS separati anziché namespace.

Comprendi le quote di Elastic Load Balancer

Quando crei i tuoi servizi, considera il tipo di bilanciamento del carico che utilizzerai (ad esempio Network Load Balancer (NLB) o Application Load Balancer (ALB)). Ogni tipo di load balancer offre funzionalità diverse e ha quote diverse. Alcune delle quote predefinite possono essere modificate, ma ci sono alcuni valori massimi di quota che non possono essere modificati. Per visualizzare le quote e l'utilizzo del tuo account, consulta la dashboard di Service Quotas nella console AWS.

Ad esempio, gli obiettivi ALB predefiniti sono 1000. Se disponi di un servizio con più di 1000 endpoint, dovrai aumentare la quota o suddividere il servizio tra più endpoint ALBs o utilizzare Kubernetes Ingress. Le destinazioni NLB predefinite sono 3000, ma sono limitate a 500 destinazioni per AZ. Se il tuo cluster utilizza più di 500 pod per un servizio NLB, dovrai utilizzarne più di uno AZs o richiedere un aumento del limite di quota.

Un'alternativa all'utilizzo di un sistema di bilanciamento del carico abbinato a un servizio consiste nell'utilizzare un controller di ingresso. Il controller AWS Load Balancer può creare ALBs risorse in ingresso, ma potresti prendere in considerazione l'utilizzo di un controller dedicato nel tuo cluster. Un controller di ingresso interno al cluster consente di esporre più servizi Kubernetes da un unico sistema di bilanciamento del carico eseguendo un proxy inverso all'interno del cluster. I controller

dispongono di funzionalità diverse, come il supporto per l'API Gateway, che può comportare vantaggi a seconda del numero e delle dimensioni dei carichi di lavoro.

Usa Route 53, Global Accelerator o CloudFront

Per rendere disponibile un servizio che utilizza più sistemi di bilanciamento del carico come singolo endpoint, devi utilizzare Amazon CloudFront, AWS Global Accelerator o Amazon Route 53 per esporre tutti i sistemi di bilanciamento del carico come un unico endpoint rivolto al cliente. Ciascuna opzione presenta vantaggi diversi e può essere utilizzata separatamente o insieme a seconda delle esigenze.

Route 53 può esporre più sistemi di bilanciamento del carico con un nome comune e inviare traffico a ciascuno di essi in base al peso assegnato. Per ulteriori informazioni sui pesi DNS, consulta la documentazione e scopri come implementarli con il controller DNS esterno Kubernetes nella documentazione di AWS Load Balancer Controller.

Global Accelerator può indirizzare i carichi di lavoro verso la regione più vicina in base all'indirizzo IP richiesto. Ciò può essere utile per i carichi di lavoro distribuiti in più regioni, ma non migliora il routing verso un singolo cluster in una singola regione. L'uso di Route 53 in combinazione con Global Accelerator offre vantaggi aggiuntivi come il controllo dello stato e il failover automatico se non è disponibile una AZ. Puoi vedere un esempio di utilizzo di Global Accelerator con Route 53 in questo post del blog.

CloudFront può essere utilizzato con Route 53 e Global Accelerator o da solo per indirizzare il traffico verso più destinazioni. CloudFront memorizza nella cache le risorse fornite dalle fonti di origine, il che può ridurre i requisiti di larghezza di banda a seconda del tipo di servizio che si sta servendo.

Utilizza EndpointSlices al posto degli endpoint

Quando scopri i pod che corrispondono a un'etichetta di servizio, dovresti usarli EndpointSlicesal posto degli endpoint. Gli endpoint erano un modo semplice per esporre servizi su piccola scala, ma i servizi di grandi dimensioni con scalabilità automatica o aggiornamenti generano molto traffico sul piano di controllo di Kubernetes. EndpointSlices dispongono di un raggruppamento automatico che abilita cose come i suggerimenti che tengono conto della topologia.

Non tutti i controller lo utilizzano di default. EndpointSlices È necessario verificare le impostazioni del controller e abilitarlo se necessario. Per il controller AWS Load Balancer è necessario abilitare il flag `--enable-endpoint-slices` opzionale da utilizzare. EndpointSlices

Se possibile, usa segreti immutabili ed esterni

Il kubelet conserva una cache delle chiavi e dei valori correnti per i Secrets utilizzati nei volumi per i pod su quel nodo. Il kubelet controlla i Secrets per rilevare le modifiche. Man mano che il cluster cresce, il numero crescente di orologi può influire negativamente sulle prestazioni del server API.

Esistono due strategie per ridurre il numero di orologi su Secrets:

- Per le applicazioni che non richiedono l'accesso alle risorse Kubernetes, puoi disabilitare i segreti degli account di servizio con montaggio automatico impostando Token: false automountServiceAccount
- Se i segreti dell'applicazione sono statici e non verranno modificati in futuro, contrassegna il segreto come immutabile. Il kubelet non mantiene un controllo API per i segreti immutabili.

Per disabilitare il montaggio automatico di un account di servizio sui pod, puoi utilizzare la seguente impostazione nel tuo carico di lavoro. Puoi sovrascrivere queste impostazioni se carichi di lavoro specifici richiedono un account di servizio.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: app
automountServiceAccountToken: true
```

Monitora il numero di segreti nel cluster prima che superi il limite di 10.000. È possibile visualizzare il numero totale di segreti in un cluster con il seguente comando. È necessario monitorare questo limite tramite gli strumenti di monitoraggio del cluster.

```
kubectl get secrets -A | wc -l
```

È necessario impostare il monitoraggio per avvisare un amministratore del cluster prima che venga raggiunto questo limite. Prendi in considerazione l'utilizzo di opzioni di gestione dei segreti esterne come AWS Key Management Service (AWS KMS) o Hashicorp Vault con il driver CSI Secrets Store.

Limita la cronologia delle distribuzioni

I pod possono essere lenti durante la creazione, l'aggiornamento o l'eliminazione perché i vecchi oggetti vengono ancora tracciati nel cluster. Puoi ridurre il numero `revisionHistoryLimit` di

implementazioni per ripulire quelle più vecchie, il ReplicaSets che ridurrà al numero totale di oggetti tracciati da Kubernetes Controller Manager. Il limite cronologico predefinito per le distribuzioni è 10.

Se il cluster crea molti oggetti di lavoro tramite CronJobs o altri meccanismi, è necessario utilizzare l'`ttlSecondsAfterFinished`impostazione per pulire automaticamente i vecchi pod nel cluster. Ciò rimuoverà i lavori eseguiti con successo dalla cronologia dei processi dopo un determinato periodo di tempo.

Disabilita enableServiceLinks per impostazione predefinita

Quando un Pod viene eseguito su un nodo, il kubelet aggiunge un set di variabili di ambiente per ogni servizio attivo. I processi Linux hanno una dimensione massima per il loro ambiente, che può essere raggiunta se hai troppi servizi nel tuo spazio dei nomi. Il numero di servizi per namespace non deve superare 5.000. Dopodiché, il numero di variabili dell'ambiente di servizio supera i limiti della shell, causando il blocco dei Pods all'avvio.

Esistono altri motivi per cui i pod non dovrebbero utilizzare le variabili di ambiente di servizio per l'individuazione dei servizi. Alcuni esempi sono i conflitti tra i nomi delle variabili di ambiente, la perdita di nomi di servizio e la dimensione totale dell'ambiente. È necessario utilizzare CoredNS per scoprire gli endpoint del servizio.

Limita i webhook ad ammissione dinamica per risorsa

I webhook di ammissione dinamici includono webhook di ammissione e webhook mutanti. Sono endpoint API che non fanno parte del piano di controllo di Kubernetes e vengono chiamati in sequenza quando una risorsa viene inviata all'API Kubernetes. Ogni webhook ha un timeout predefinito di 10 secondi e può aumentare la quantità di tempo necessaria per una richiesta API se hai più webhook o se hai un timeout di uno di essi.

Assicurati che i tuoi webhook siano altamente disponibili, specialmente durante un incidente AZ, e che la FailurePolicy sia impostata correttamente per rifiutare la risorsa o ignorare l'errore. Non chiamate i webhook quando non sono necessari, permettendo ai comandi --dry-run kubectl di bypassare il webhook.

```
apiVersion: admission.k8s.io/v1
kind: AdmissionReview
request:
  dryRun: False
```

I webhook mutanti possono modificare le risorse in successione frequente. Se disponi di 5 webhook mutanti e distribuisci 50 risorse, etcd memorizzerà tutte le versioni di ciascuna risorsa fino alla compattazione, ogni 5 minuti, per rimuovere le vecchie versioni delle risorse modificate. In questo scenario, quando etcd rimuove le risorse sostituite, verranno rimosse 200 versioni di risorse da etcd e, a seconda delle dimensioni delle risorse, potrebbe occupare molto spazio sull'host etcd fino a quando la deframmentazione non viene eseguita ogni 15 minuti.

Questa deframmentazione può causare pause in etcd che potrebbero avere altri effetti sull'API e sui controller di Kubernetes. È consigliabile evitare la modifica frequente di risorse di grandi dimensioni o la modifica di centinaia di risorse in rapida successione.

Confronta i carichi di lavoro tra più cluster

Se hai due cluster che dovrebbero avere prestazioni simili ma non lo fanno, prova a confrontare le metriche per identificarne il motivo.

Ad esempio, il confronto della latenza dei cluster è un problema comune. Questo è in genere causato dalla differenza nel volume delle richieste API. È possibile eseguire la seguente CloudWatch LogInsight query per comprendere la differenza.

```
filter @logStream like "kube-apiserver-audit"
| stats count(*) as cnt by objectRef.apiGroup, objectRef.apiVersion,
 objectRef.resource, userAgent, verb, responseStatus.code
| sort cnt desc
| limit 1000
```

Puoi aggiungere filtri aggiuntivi per restringere il campo, ad esempio concentrandoti su tutte le richieste di elenchi da `foo`.

```
filter @logStream like "kube-apiserver-audit"
| filter verb = "list"
| filter user.username like "foo"
| stats count(*) as cnt by objectRef.apiGroup, objectRef.apiVersion,
 objectRef.resource, responseStatus.code
| sort cnt desc
| limit 1000
```

Teoria del ridimensionamento di Kubernetes

Nodes vs. Churn Rate

Spesso, quando parliamo della scalabilità di Kubernetes, lo facciamo in termini di quanti nodi ci sono in un singolo cluster. È interessante notare che questa è raramente la metrica più utile per comprendere la scalabilità. Ad esempio, un cluster da 5.000 nodi con un numero elevato ma fisso di pod non metterebbe molto a dura prova il piano di controllo dopo la configurazione iniziale. Tuttavia, se prendessimo un cluster da 1.000 nodi e provassimo a creare 10.000 posti di lavoro di breve durata in meno di un minuto, eserciteremmo una forte pressione sul piano di controllo.

Il semplice utilizzo del numero di nodi per comprendere la scalabilità può essere fuorviante. È meglio pensare in termini di velocità di cambiamento che si verifica in un determinato periodo di tempo (utilizziamo un intervallo di 5 minuti per questa discussione, poiché è quello che di solito usano le query di Prometheus per impostazione predefinita). Scopriamo perché inquadrare il problema in termini di tasso di cambiamento può darci un'idea migliore di cosa ottimizzare per raggiungere la scala desiderata.

Pensare in termini di domande al secondo

Kubernetes dispone di una serie di meccanismi di protezione per ogni componente (Kubelet, Scheduler, Kube Controller Manager e server API) per evitare di sovraccaricare l'anello successivo della catena Kubernetes. Ad esempio, Kubelet dispone di un flag per limitare le chiamate al server API a una certa velocità. Questi meccanismi di protezione sono generalmente, ma non sempre, espressi in termini di query consentite al secondo o QPS.

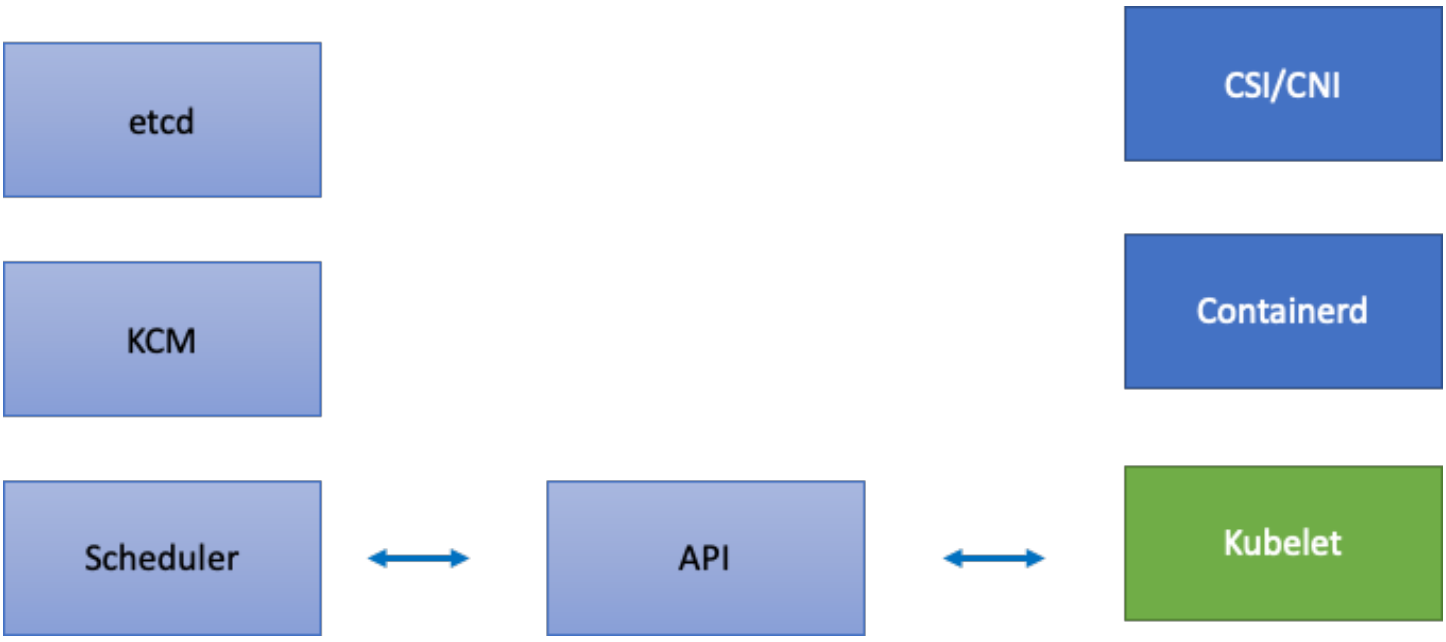
È necessario prestare molta attenzione quando si modificano queste impostazioni QPS. La rimozione di un collo di bottiglia, ad esempio le query al secondo su un Kubelet, avrà un impatto sugli altri componenti a valle. Ciò può sovraccaricare il sistema oltre una certa velocità, e lo farà, quindi la comprensione e il monitoraggio di ogni parte della catena di servizi sono fondamentali per scalare con successo i carichi di lavoro su Kubernetes.

> **Note**
>
> Il server API ha un sistema più complesso con l'introduzione di API Priority and Fairness, di cui parleremo separatamente.

> **Note**
>
> Attenzione, alcune metriche sembrano la soluzione giusta ma in realtà misurano qualcos'altro. Ad esempio, `kubelet_http_inflight_requests` si riferisce solo al server delle metriche in Kubelet, non al numero di richieste da Kubelet alle richieste apiserver. Ciò potrebbe indurci a configurare erroneamente il flag QPS su Kubelet. Una query sui registri di controllo per un determinato Kubelet sarebbe un modo più affidabile per controllare le metriche.

Scalabilità dei componenti distribuiti

Poiché EKS è un servizio gestito, dividiamo i componenti di Kubernetes in due categorie: componenti gestiti da AWS che includono etcd, Kube Controller Manager e Scheduler (nella parte sinistra del diagramma) e componenti configurabili dal cliente come Kubelet, Container Runtime e i vari operatori che chiamano AWS APIs come i driver di rete e archiviazione (nella parte destra del diagramma). Lasciamo il server API al centro anche se è gestito da AWS, poiché le impostazioni per la priorità e l'equità delle API possono essere configurate dai clienti.



Colli di bottiglia a monte e a valle

Mentre monitoriamo ogni servizio, è importante esaminare le metriche in entrambe le direzioni per individuare eventuali rallentamenti. Impariamo come farlo usando Kubelet come esempio. Kubelet

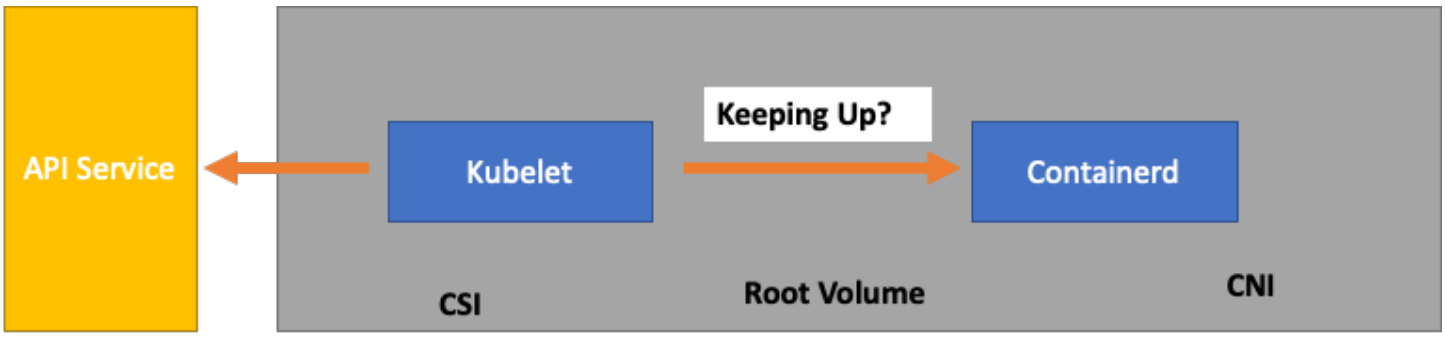
parla sia con il server API che con il runtime del contenitore; come e cosa dobbiamo monitorare per rilevare se uno dei componenti presenta un problema?

Quanti Pod per nodo

Quando esaminiamo i numeri in scala, ad esempio quanti pod possono essere eseguiti su un nodo, potremmo prendere alla lettera i 110 pod per nodo supportati da upstream.

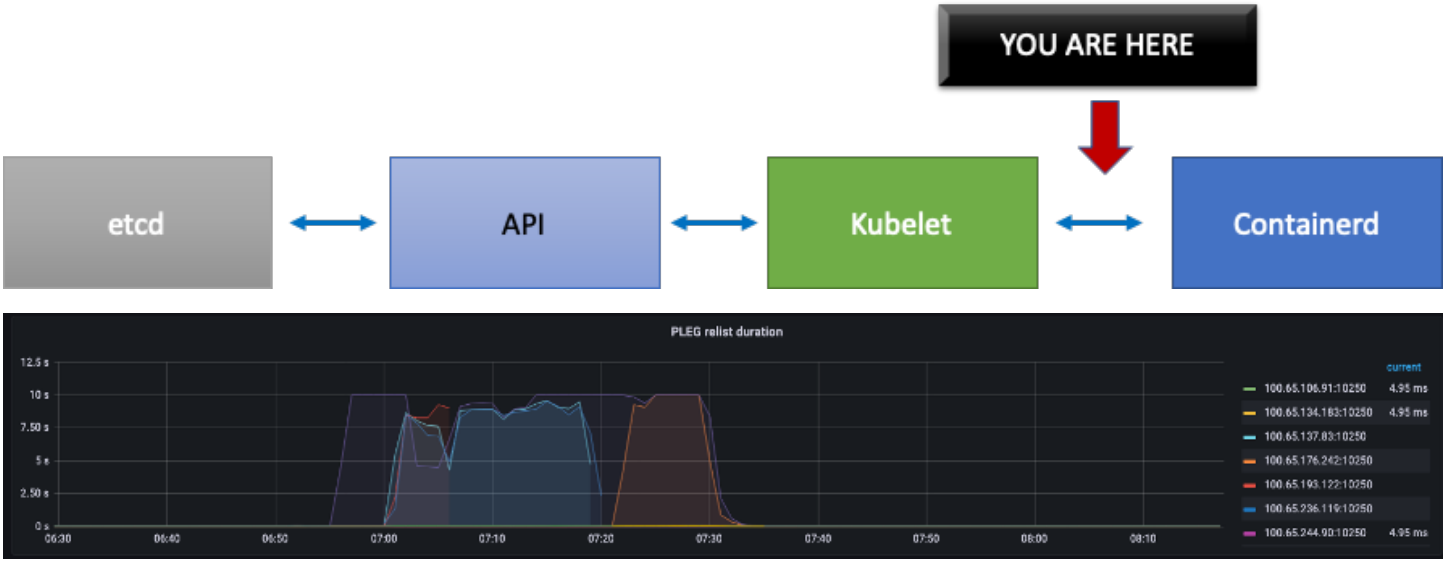
> **Note**
>
> https://kubernetes. io/docs/setup/best-practices/cluster-grande/

Tuttavia, il carico di lavoro è probabilmente più complesso di quello testato in un test di scalabilità in Upstream. Per assicurarci di poter gestire il numero di pod che vogliamo utilizzare in produzione, assicuriamoci che Kubelet «stia al passo» con il runtime di Containerd.



Per semplificare eccessivamente, Kubelet sta ottenendo lo stato dei pod dal runtime del contenitore (nel nostro caso Containerd). E se troppi pod cambiassero lo stato troppo velocemente? Se la velocità di modifica è troppo alta, le richieste [al runtime del contenitore] possono scadere.

> **Note**
>
> Kubernetes è in continua evoluzione, questo sottosistema è attualmente sottoposto a modifiche. https://github.com/kubernetes/miglioramenti/problemi/3386



Nel grafico precedente, vediamo una linea piatta che indica che abbiamo appena raggiunto il valore di timeout per la metrica della durata della generazione degli eventi del ciclo di vita del pod. Se desideri visualizzarlo nel tuo cluster, puoi utilizzare la seguente sintassi PromQL.

```
increase(kubelet_pleg_relist_duration_seconds_bucket{instance="$instance"}
[$__rate_interval])
```

Se assistiamo a questo comportamento di timeout, sappiamo di aver spinto il nodo oltre il limite di cui era capace. Dobbiamo correggere la causa del timeout prima di procedere ulteriormente. Ciò può essere ottenuto riducendo il numero di pod per nodo o cercando errori che potrebbero causare un elevato volume di nuovi tentativi (con conseguente impatto sul tasso di abbandono). La conclusione importante è che le metriche sono il modo migliore per capire se un nodo è in grado di gestire il tasso di abbandono dei pod assegnati anziché utilizzare un numero fisso.

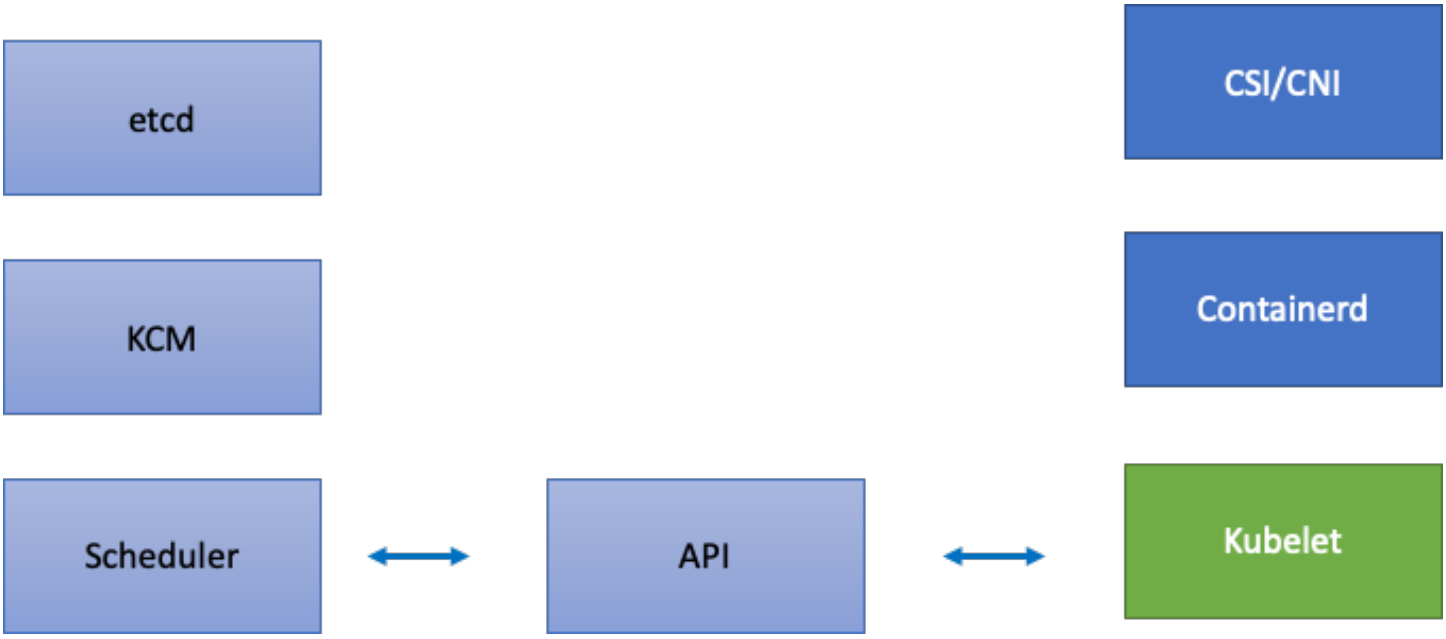
Scala in base alle metriche

Sebbene il concetto di utilizzo delle metriche per ottimizzare i sistemi sia vecchio, viene spesso trascurato quando le persone iniziano il loro percorso verso Kubernetes. Invece di concentrarci su numeri specifici (ad esempio 110 pod per nodo), concentriamo i nostri sforzi sulla ricerca di metriche che ci aiutino a individuare i punti deboli nel nostro sistema. La comprensione delle soglie corrette per queste metriche può darci un alto grado di sicurezza che il nostro sistema sia configurato in modo ottimale.

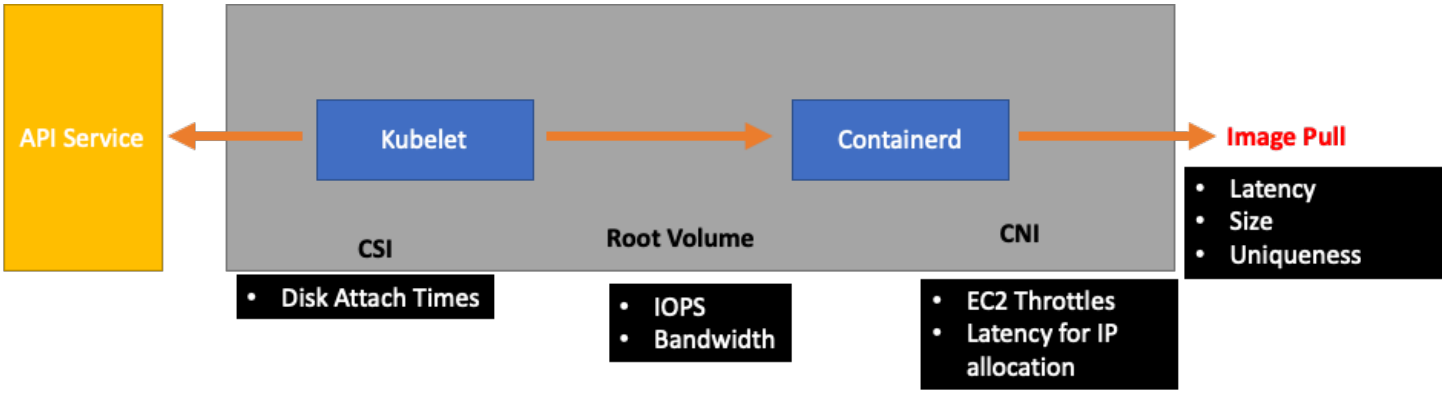
L'impatto delle modifiche

Uno schema comune che potrebbe metterci nei guai è quello di concentrarci sul primo errore metrico o di registro che sembra sospetto. Quando abbiamo visto che il Kubelet stava scadendo prima, abbiamo potuto provare cose casuali, come aumentare la frequenza al secondo che Kubelet può inviare, ecc. Tuttavia, è consigliabile esaminare il quadro completo di tutto ciò che si trova a valle dell'errore che riscontriamo per primo. Apporta ogni modifica in modo mirato e supportata dai dati.

A valle di Kubelet ci sarebbe il runtime di Containerd (errori dei pod), DaemonSets come il driver di archiviazione (CSI) e il driver di rete (CNI) che comunicano con l'API EC2, ecc.



Continuiamo il nostro esempio precedente di Kubelet che non riesce a tenere il passo con il runtime. Ci sono diversi punti in cui potremmo impacchettare un nodo in modo così denso da generare errori.



Quando progettiamo la giusta dimensione dei nodi per i nostri carichi di lavoro, si tratta di easy-to-overlook segnali che potrebbero esercitare una pressione inutile sul sistema, limitando così sia la nostra scalabilità che le prestazioni.

Il costo degli errori non necessari

I controller Kubernetes eccellono nel riprovare quando si verificano condizioni di errore, ma ciò ha un costo. Questi nuovi tentativi possono aumentare la pressione su componenti come Kube Controller Manager. Il monitoraggio di tali errori è un requisito importante dei test di scala.

Quando si verificano meno errori, è più facile individuare i problemi nel sistema. Garantendo periodicamente che i nostri cluster siano privi di errori prima delle operazioni più importanti (come gli aggiornamenti), possiamo semplificare i log di risoluzione dei problemi quando si verificano eventi imprevisti.

Espandere la nostra visione

Nei cluster su larga scala con migliaia di nodi non vogliamo cercare i colli di bottiglia singolarmente. In PromQL possiamo trovare i valori più alti in un set di dati usando una funzione chiamata topk; essendo K una variabile, inseriamo il numero di elementi che vogliamo. Qui utilizziamo tre nodi per avere un'idea se tutti i Kubelet del cluster sono saturi. Fino a questo punto abbiamo esaminato la latenza, ora vediamo se Kubelet sta scartando gli eventi.

```
topk(3, increase(kubelet_pleg_discard_events{}[$__rate_interval]))
```

Analizzando questa affermazione.

- Utilizziamo la variabile Grafana `$__rate_interval` per assicurarci che riceva i quattro campioni necessari. Questo aggira un argomento complesso del monitoraggio con una variabile semplice.
- `topk`dacci solo i migliori risultati e il numero 3 limita quei risultati a tre. Questa è una funzione utile per le metriche a livello di cluster.
- `{}`dicci che non ci sono filtri, normalmente inseriresti il nome del lavoro qualunque sia la regola di scraping, tuttavia poiché questi nomi variano lo lasceremo vuoto.

Dividere il problema a metà

Per risolvere un problema nel sistema, adotteremo l'approccio di trovare una metrica che ci indichi l'esistenza di un problema a monte o a valle, in quanto ciò ci consente di dividere il problema a metà. Sarà anche un principio fondamentale del modo in cui mostriamo i dati delle nostre metriche.

Un buon punto di partenza per questo processo è il server API, in quanto ci consente di vedere se c'è un problema con un'applicazione client o con il Control Plane.

Monitoraggio del piano di controllo

Server API

Quando esaminiamo il nostro server API, è importante ricordare che una delle sue funzioni è quella di limitare le richieste in entrata per evitare di sovraccaricare il piano di controllo. Ciò che può sembrare un collo di bottiglia a livello di server API potrebbe in realtà proteggerlo da problemi più gravi. Dobbiamo tenere conto dei pro e dei contro dell'aumento del volume di richieste che transitano attraverso il sistema. Per determinare se i valori del server API debbano essere aumentati, ecco un piccolo esempio degli aspetti a cui dobbiamo prestare attenzione:

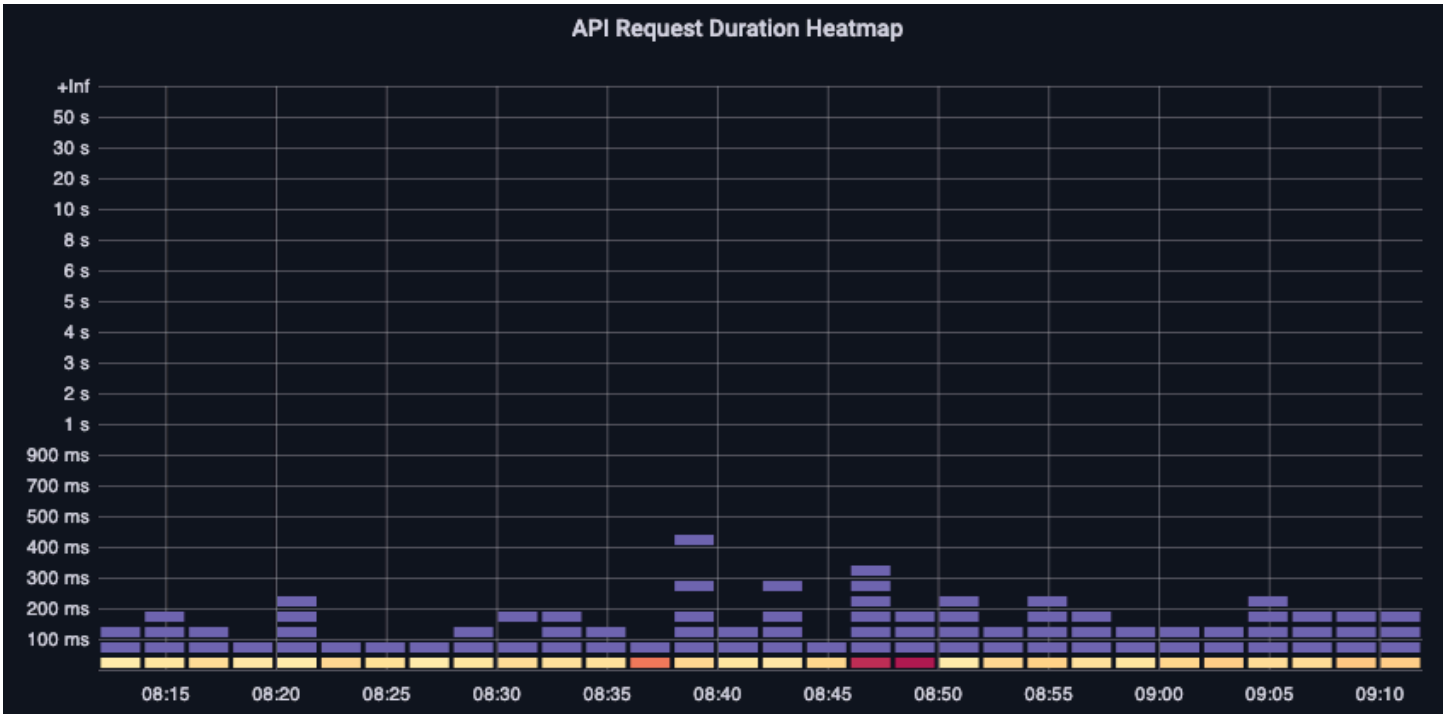
1. Qual è la latenza delle richieste che transitano attraverso il sistema?
2. Questa latenza è il server API stesso o qualcosa di «derivato» come etcd?
3. La profondità della coda del server API è un fattore di questa latenza?
4. Le code API Priority and Fairness (APF) sono configurate correttamente per i modelli di chiamata API che vogliamo?

Dov'è il problema?

Per iniziare, possiamo utilizzare la metrica della latenza delle API per darci un'idea del tempo impiegato dal server API per soddisfare le richieste di servizio. Usiamo la mappa di calore ProMQL e Grafana di seguito per visualizzare questi dati.

```
max(increase(apiserver_request_duration_seconds_bucket{subresource!
="status",subresource!="token",subresource!="scale",subresource!="/
healthz",subresource!="binding",subresource!="proxy",verb!="WATCH"}[$__rate_interval]))
 by (le)
```

> **Note**
>
> Per una descrizione approfondita su come monitorare il server API con la dashboard delle API utilizzata in questo articolo, consulta il seguente blog



Queste richieste sono tutte inferiori a un secondo, il che è una buona indicazione del fatto che il piano di controllo sta gestendo le richieste in modo tempestivo. E se invece non fosse così?

Il formato che stiamo usando nella precedente API Request Duration è una heatmap. La cosa bella del formato heatmap è che ci indica il valore di timeout per l'API di default (60 sec). Tuttavia, ciò di cui abbiamo veramente bisogno è sapere a quale soglia questo valore deve destare preoccupazione prima di raggiungere la soglia di timeout. Per una guida approssimativa su quali siano le soglie accettabili, possiamo utilizzare lo SLO di Kubernetes a monte, che può essere trovato qui

> **Note**
>
> Notate la funzione max in questa dichiarazione? Quando si utilizzano metriche che aggregano più server (per impostazione predefinita due server API su EKS) è importante non calcolare la media di tali server.

Modelli di traffico asimmetrici

E se un server API [pod] fosse caricato leggermente e l'altro pesantemente? Se calcoliamo la media di questi due numeri, potremmo fraintendere ciò che sta accadendo. Ad esempio, qui abbiamo tre server API ma tutto il carico è su uno di questi server API. Di norma, tutto ciò che ha più server,

come etcd e server API, dovrebbe essere interrotto quando si investono in problemi di scalabilità e prestazioni.

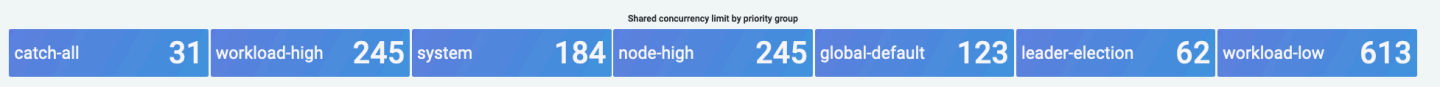
Con il passaggio a API Priority and Fairness, il numero totale di richieste sul sistema è solo uno dei fattori da verificare per verificare se il server API ha un numero di sottoscrizioni eccessivo. Poiché il sistema ora utilizza una serie di code, dobbiamo verificare se qualcuna di queste code è piena e se il traffico per quella coda sta diminuendo.

Diamo un'occhiata a queste code con la seguente domanda:

```
max without(instance)(apiserver_flowcontrol_nominal_limit_seats{})
```

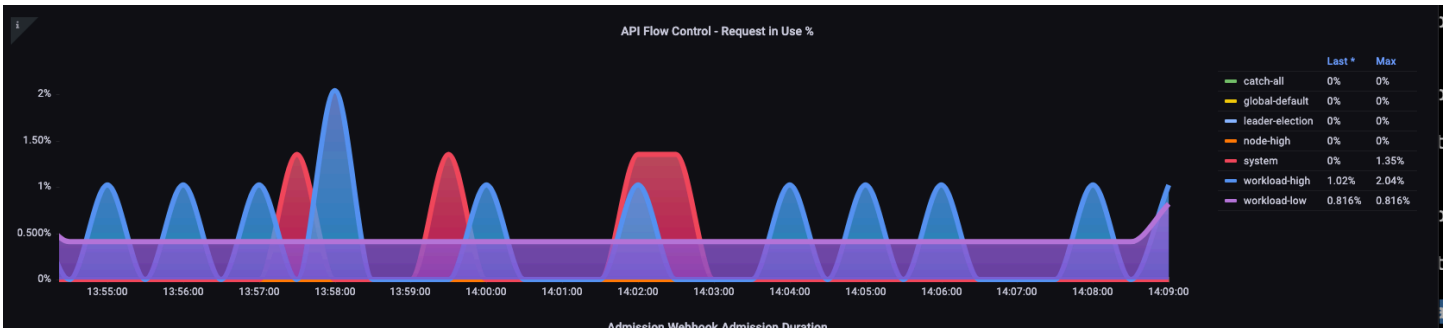
> **Note**
>
> Per ulteriori informazioni su come funziona l'API A&F, consulta la seguente guida alle migliori pratiche

Qui vediamo i sette diversi gruppi di priorità presenti di default nel cluster



Successivamente vogliamo vedere quale percentuale di quel gruppo di priorità viene utilizzata, in modo da poter capire se un determinato livello di priorità è saturo. Potrebbe essere auspicabile limitare le richieste al livello di carico di lavoro più basso, ma non lo sarebbe un calo del livello di elezione dei leader.

Il sistema API Priority and Fairness (APF) offre una serie di opzioni complesse, alcune delle quali possono avere conseguenze indesiderate. Un problema comune che riscontriamo sul campo è l'aumento della profondità della coda al punto da iniziare ad aggiungere latenze non necessarie. Possiamo monitorare questo problema utilizzando la `apiserver_flowcontrol_current_inqueue_request` metrica. Possiamo verificare la presenza di cadute usando il`apiserver_flowcontrol_rejected_requests_total`. Queste metriche avranno un valore diverso da zero se un bucket supera la concorrenza.



L'aumento della profondità della coda può rendere l'API Server una fonte significativa di latenza e deve essere fatto con cautela. Ti consigliamo di usare cautela con il numero di code create. Ad esempio, il numero di condivisioni su un sistema EKS è 600. Se creiamo troppe code, ciò può ridurre le condivisioni nelle code importanti che richiedono il throughput, come la coda per le elezioni dei leader o la coda di sistema. La creazione di troppe code aggiuntive può rendere più difficile il corretto dimensionamento di tali code.

Per concentrarci su una semplice modifica efficace che potete apportare in APF, prendiamo semplicemente le azioni dai bucket sottoutilizzati e aumentiamo le dimensioni dei bucket che raggiungono il loro massimo utilizzo. Ridistribuendo in modo intelligente le azioni tra questi gruppi, è possibile ridurre il rischio di perdite.

Per ulteriori informazioni, consulta le impostazioni di priorità e correttezza delle API nella Guida alle migliori pratiche EKS.

Latenza API vs. etcd

Come possiamo usare il metrics/logs server API per determinare se c'è un problema con il server API, o un problema relativo al server API o una combinazione upstream/downstream di entrambi. Per capirlo meglio, esaminiamo come API Server ed etcd possono essere correlati e quanto può essere facile risolvere i problemi del sistema sbagliato.

Nel grafico seguente vediamo la latenza del server API, ma vediamo anche che gran parte di questa latenza è correlata al server etcd a causa delle barre nel grafico che mostrano la maggior parte della

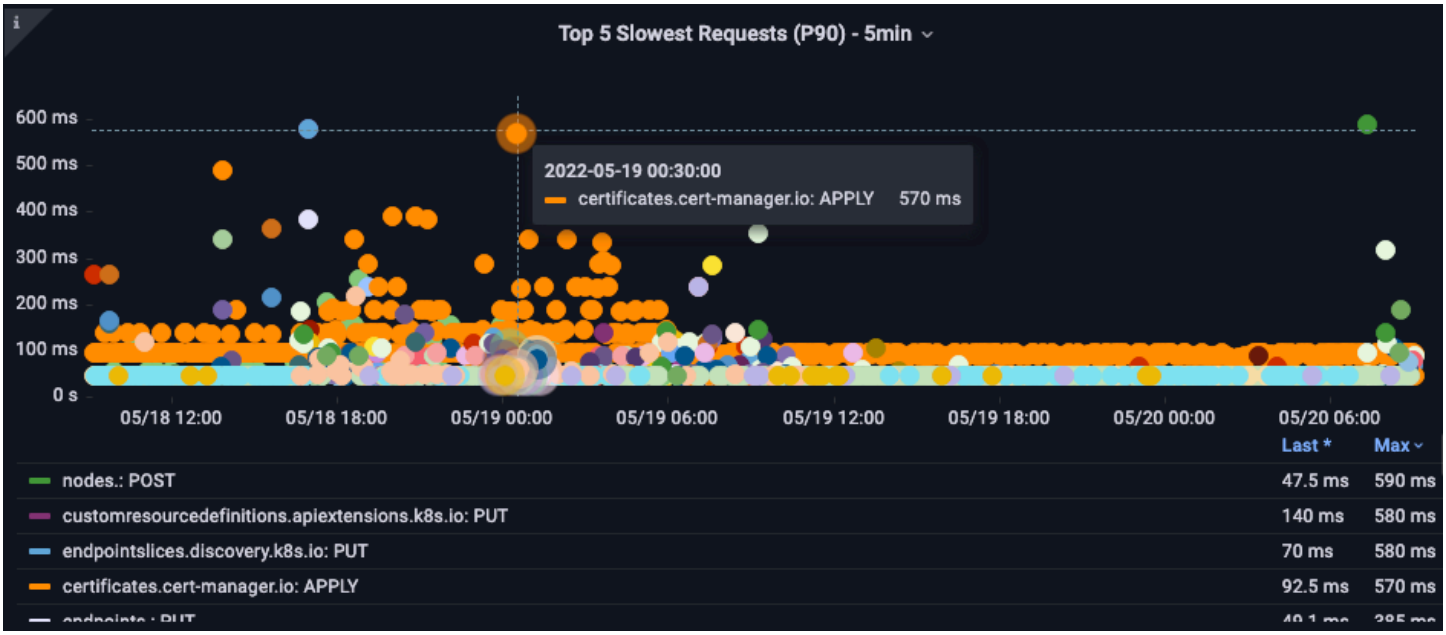
latenza a livello di etcd. Se ci sono 15 secondi di latenza etcd contemporaneamente, ci sono 20 secondi di latenza del server API, allora la maggior parte della latenza è effettivamente a livello etcd.

Osservando l'intero flusso, vediamo che è consigliabile non concentrarsi esclusivamente sull'API Server, ma anche cercare segnali che indichino che etcd è in difficoltà (ad esempio un lento aumento dei contatori di applicazioni). La possibilità di passare rapidamente all'area problematica giusta con un semplice colpo d'occhio è ciò che rende potente una dashboard.

> **Note**
>
> La dashboard nella sezione Troubleshooter.json è disponibile in https://github.com/RiskyAdventure/ Troubleshooter.json Dashboards/blob/main/api

Problemi relativi al piano di controllo e al lato client

In questo grafico cerchiamo le chiamate API che hanno richiesto più tempo per essere completate in quel periodo. In questo caso vediamo che una risorsa personalizzata (CRD) sta chiamando una funzione APPLY che è la chiamata più latente nell'intervallo di tempo delle 05:40.



Grazie a questi dati, possiamo utilizzare una query Ad-Hoc ProMQL o CloudWatch Insights per estrarre le richieste LIST dal registro di controllo durante quel periodo di tempo e vedere quale potrebbe essere l'applicazione.

Trovare la fonte con CloudWatch

Le metriche sono utilizzate al meglio per trovare l'area problematica che vogliamo esaminare e restringere sia l'intervallo di tempo che i parametri di ricerca del problema. Una volta che disponiamo di questi dati, vogliamo passare ai log per tempi ed errori più dettagliati. Per fare ciò, trasformeremo i nostri log in metriche utilizzando CloudWatch Logs Insights.

Ad esempio, per esaminare il problema di cui sopra, utilizzeremo la seguente query di CloudWatch Logs Insights per estrarre UserAgent e RequestURI in modo da poter individuare quale applicazione causa questa latenza.

> **Note**
>
> È necessario utilizzare un Count appropriato per non generare un List/Resync comportamento normale su un Watch.

```
fields @timestamp, @message
| filter @logStream like "kube-apiserver-audit"
```

```
| filter ispresent(requestURI)
| filter verb = "list"
| parse requestReceivedTimestamp /\d+-\d+-(?<StartDay>\d+)T(?<StartHour>\d+):(?
<StartMinute>\d+):(?<StartSec>\d+).(?<StartMsec>\d+)Z/
| parse stageTimestamp /\d+-\d+-(?<EndDay>\d+)T(?<EndHour>\d+):(?<EndMinute>\d+):(?
<EndSec>\d+).(?<EndMsec>\d+)Z/
| fields (StartHour * 3600 + StartMinute * 60 + StartSec + StartMsec / 1000000) as
 StartTime, (EndHour * 3600 + EndMinute * 60 + EndSec + EndMsec / 1000000) as EndTime,
 (EndTime - StartTime) as DeltaTime
| stats avg(DeltaTime) as AverageDeltaTime, count(*) as CountTime by requestURI,
 userAgent
| filter CountTime >=50
| sort AverageDeltaTime desc
```

Utilizzando questa query abbiamo trovato due agenti diversi che eseguono un gran numero di operazioni con elenchi ad alta latenza. Splunk e agent. CloudWatch Grazie ai dati, possiamo decidere di rimuovere, aggiornare o sostituire questo controller con un altro progetto.

requestURI	AverageDeltaTime	AverageD	CountTime
/api/v1/pods	rest-client/2.1.0 (linux-gnu x86_64) ruby/2.5.5p157	9.1395	87
/api/v1/pods?limit=500&resourceVersion=0	amazon-cloudwatch-agent/v0.0.0 (linux/amd64) kubernetes/$Format	5.3976	929

> **Note**
>
> Per maggiori dettagli su questo argomento, consulta il seguente blog

Pianificatore

Poiché le istanze del piano di controllo EKS vengono eseguite in un account AWS separato, non saremo in grado di raccogliere tali componenti per le metriche (il server API è l'eccezione). Tuttavia, poiché abbiamo accesso ai log di controllo di questi componenti, possiamo trasformarli in metriche per vedere se qualcuno dei sottosistemi sta causando un collo di bottiglia di scalabilità. Utilizziamo CloudWatch Logs Insights per vedere quanti pod non pianificati ci sono nella coda dello scheduler.

Pods non programmati nel registro dello scheduler

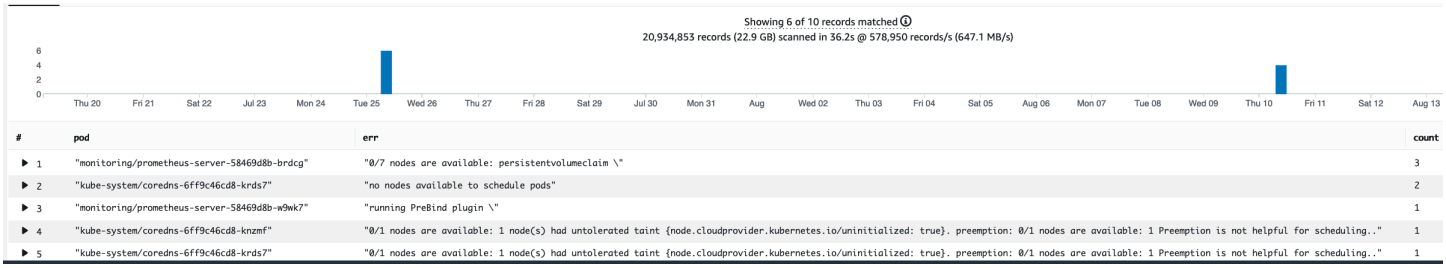
Se avessimo accesso per analizzare le metriche dello scheduler direttamente su un Kubernetes autogestito (come Kops), utilizzeremmo il seguente ProMQL per comprendere il backlog dello scheduler.

```
max without(instance)(scheduler_pending_pods)
```

Poiché non abbiamo accesso alla metrica di cui sopra in EKS, utilizzeremo la seguente query di CloudWatch Logs Insights per visualizzare il backlog verificando quanti pod non sono stati annullati durante un determinato periodo di tempo. Potremmo quindi approfondire i messaggi ricevuti nei momenti di maggiore affluenza per comprendere la natura del problema. Ad esempio, i nodi non si avviano abbastanza velocemente o il limitatore di velocità presente nello scheduler stesso.

```
fields timestamp, pod, err, @message
| filter @logStream like "scheduler"
| filter @message like "Unable to schedule pod"
| parse @message  /^.(?<date>\d{4})\s+(?<timestamp>\d+:\d+:\d+\.\d+)\s+\S*\s+\S+\]\s
\"(.*?)\"\s+pod=(?<pod>\"(.*?)\")\s+err=(?<err>\"(.*?)\")/
| count(*) as count by pod, err
| sort count desc
```

Qui vediamo gli errori dello scheduler che indicano che il pod non è stato distribuito perché lo storage PVC non era disponibile.



> **Note**
>
> La registrazione di controllo deve essere attivata sul piano di controllo per abilitare questa funzione. È inoltre consigliabile limitare la conservazione dei registri per non aumentare inutilmente i costi nel tempo. Di seguito è riportato un esempio per attivare tutte le funzioni di registrazione utilizzando lo strumento EKSCTL.

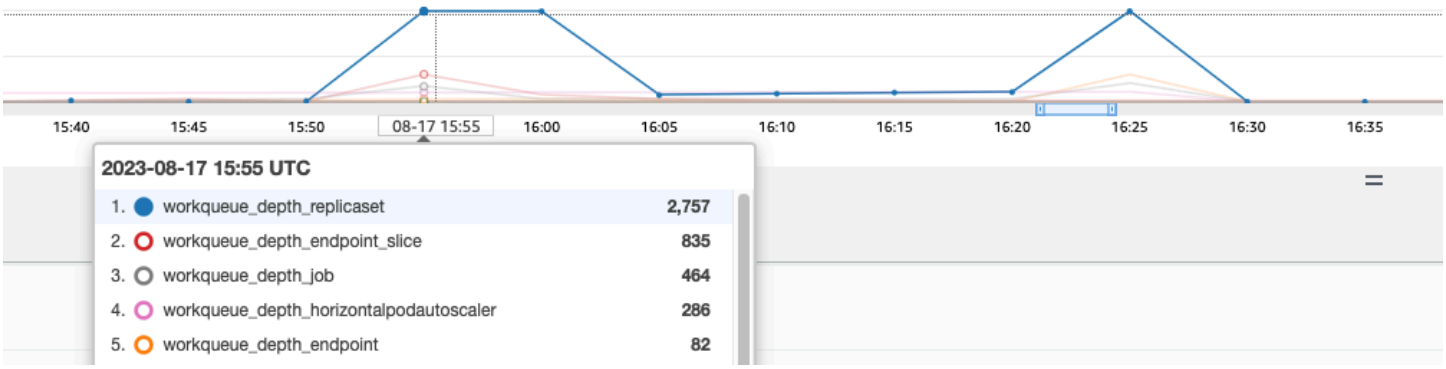
```
cloudWatch:
  clusterLogging:
    enableTypes: ["*"]
    logRetentionInDays: 10
```

Kube Controller Manager

Kube Controller Manager, come tutti gli altri controller, ha dei limiti sul numero di operazioni che può eseguire contemporaneamente. Esaminiamo quali sono alcuni di questi flag esaminando una configurazione KOPS in cui possiamo impostare questi parametri.

```
kubeControllerManager:
  concurrentEndpointSyncs: 5
  concurrentReplicasetSyncs: 5
  concurrentNamespaceSyncs: 10
  concurrentServiceaccountTokenSyncs: 5
  concurrentServiceSyncs: 5
  concurrentResourceQuotaSyncs: 5
  concurrentGcSyncs: 20
  kubeAPIBurst: 30
  kubeAPIQPS: "20"
```

Questi controller hanno code che si riempiono durante i periodi di elevato tasso di abbandono su un cluster. In questo caso vediamo che il controller del set replicaset ha un grosso backlog nella sua coda.



Abbiamo due modi diversi per affrontare una situazione del genere. Se eseguissimo la gestione autonoma, potremmo semplicemente aumentare le goroutine simultanee, tuttavia ciò avrebbe un impatto su etcd elaborando più dati nel KCM. L'altra opzione sarebbe quella di ridurre il numero di oggetti replicaset utilizzati nella distribuzione per ridurre il numero di oggetti replicaset che è possibile ripristinare, riducendo così la pressione `.spec.revisionHistoryLimit` su questo controller.

```
spec:
  revisionHistoryLimit: 2
```

Altre funzionalità di Kubernetes possono essere regolate o disattivate per ridurre la pressione nei sistemi ad alto tasso di abbandono. Ad esempio, se l'applicazione contenuta nei nostri pod non ha bisogno di comunicare direttamente con l'API k8s, la disattivazione del segreto proiettato in quei pod ridurrebbe il carico. ServiceaccountTokenSyncs Questo è il modo più auspicabile per risolvere tali problemi, se possibile.

```
kind: Pod
spec:
  automountServiceAccountToken: false
```

Nei sistemi in cui non riusciamo ad accedere alle metriche, possiamo nuovamente esaminare i log per rilevare eventuali contese. Se volessimo vedere il numero di richieste elaborate per controller o a livello aggregato, utilizzeremmo la seguente CloudWatch Logs Insights Query.

Volume totale elaborato dal KCM

```
# Query to count API qps coming from kube-controller-manager, split by controller type.
# If you're seeing values close to 20/sec for any particular controller, it's most
 likely seeing client-side API throttling.
fields @timestamp, @logStream, @message
| filter @logStream like /kube-apiserver-audit/
| filter userAgent like /kube-controller-manager/
# Exclude lease-related calls (not counted under kcm qps)
| filter requestURI not like "apis/coordination.k8s.io/v1/namespaces/kube-system/
leases/kube-controller-manager"
# Exclude API discovery calls (not counted under kcm qps)
| filter requestURI not like "?timeout=32s"
# Exclude watch calls (not counted under kcm qps)
| filter verb != "watch"
# If you want to get counts of API calls coming from a specific controller, uncomment
 the appropriate line below:
# | filter user.username like "system:serviceaccount:kube-system:job-controller"
# | filter user.username like "system:serviceaccount:kube-system:cronjob-controller"
# | filter user.username like "system:serviceaccount:kube-system:deployment-controller"
# | filter user.username like "system:serviceaccount:kube-system:replicaset-controller"
# | filter user.username like "system:serviceaccount:kube-system:horizontal-pod-
autoscaler"
# | filter user.username like "system:serviceaccount:kube-system:persistent-volume-
binder"
# | filter user.username like "system:serviceaccount:kube-system:endpointslice-
controller"
# | filter user.username like "system:serviceaccount:kube-system:endpoint-controller"
```

```
# | filter user.username like "system:serviceaccount:kube-system:generic-garbage-
controller"
| stats count(*) as count by user.username
| sort count desc
```

Quando si esaminano i problemi di scalabilità, è fondamentale esaminare ogni fase del percorso (API, scheduler, KCM, ecc.) prima di passare alla fase dettagliata della risoluzione dei problemi. Spesso, in produzione, è necessario apportare modifiche a più di una parte di Kubernetes per consentire al sistema di funzionare al massimo delle prestazioni. È facile risolvere inavvertitamente quello che è solo un sintomo (come il timeout di un nodo) di un collo di bottiglia molto più grande.

ECCETERA

etcd utilizza un file mappato in memoria per memorizzare in modo efficiente le coppie chiave-valore. Esiste un meccanismo di protezione per impostare la dimensione di questo spazio di memoria disponibile, generalmente ai limiti di 2, 4 e 8 GB. Un numero inferiore di oggetti nel database significa meno operazioni di pulizia da eseguire su etcd quando gli oggetti vengono aggiornati e le versioni precedenti devono essere pulite. Questo processo di pulizia delle vecchie versioni di un oggetto viene chiamato compattazione. Dopo una serie di operazioni di compattazione, viene seguito un processo che recupera lo spazio utilizzabile chiamato deframmentazione, che avviene al di sopra di una certa soglia o in base a un programma di tempo prestabilito.

Esistono un paio di elementi relativi all'utente che possiamo fare per limitare il numero di oggetti in Kubernetes e quindi ridurre l'impatto del processo di compattazione e deframmentazione. Ad esempio, Helm mantiene un livello elevato. `revisionHistoryLimit` In questo modo gli oggetti più vecchi, come quelli ReplicaSets presenti nel sistema, possono eseguire i rollback. Impostando i limiti della cronologia su 2, possiamo ridurre il numero di oggetti (ad esempio ReplicaSets) da dieci a due, il che a sua volta graverebbe meno carico sul sistema.

```
apiVersion: apps/v1
kind: Deployment
spec:
  revisionHistoryLimit: 2
```

Dal punto di vista del monitoraggio, se i picchi di latenza del sistema si verificano secondo uno schema prestabilito separato da ore, può essere utile verificare se questo processo di deframmentazione ne è l'origine. Possiamo vederlo usando Logs. CloudWatch

Se vuoi vedere gli start/end orari di deframmentazione usa la seguente query:

```
fields @timestamp, @message
| filter @logStream like /etcd-manager/
| filter @message like /defraging|defraged/
| sort @timestamp asc
```

@message	@logStream
Oct 6 19:19:52 ip-172-16-176-9 etcdnanny: {"level":"info","ts":"2022-10-06T19:19:52.826Z","caller":"defrag/defrag.go:312","msg":"defraging","…	etcd-manager-i-0101a20d88e74fa5b
Oct 6 19:20:28 ip-172-16-176-9 etcdnanny: {"level":"info","ts":"2022-10-06T19:20:28.836Z","caller":"defrag/defrag.go:322","msg":"defraged"}	etcd-manager-i-0101a20d88e74fa5b
Oct 6 19:25:57 ip-172-16-101-191 etcdnanny: {"level":"info","ts":"2022-10-06T19:25:57.650Z","caller":"defrag/defrag.go:312","msg":"defraging"…	etcd-manager-i-0e3aa5c2ae801af44
Oct 6 19:26:34 ip-172-16-101-191 etcdnanny: {"level":"info","ts":"2022-10-06T19:26:34.457Z","caller":"defrag/defrag.go:322","msg":"defraged"}	etcd-manager-i-0e3aa5c2ae801af44
Oct 6 19:32:26 ip-172-16-50-136 etcdnanny: {"level":"info","ts":"2022-10-06T19:32:26.592Z","caller":"defrag/defrag.go:312","msg":"defraging",…	etcd-manager-i-025ebb6b463ee0f21
Oct 6 19:33:09 ip-172-16-50-136 etcdnanny: {"level":"info","ts":"2022-10-06T19:33:09.575Z","caller":"defrag/defrag.go:322","msg":"defraged"}	etcd-manager-i-025ebb6b463ee0f21

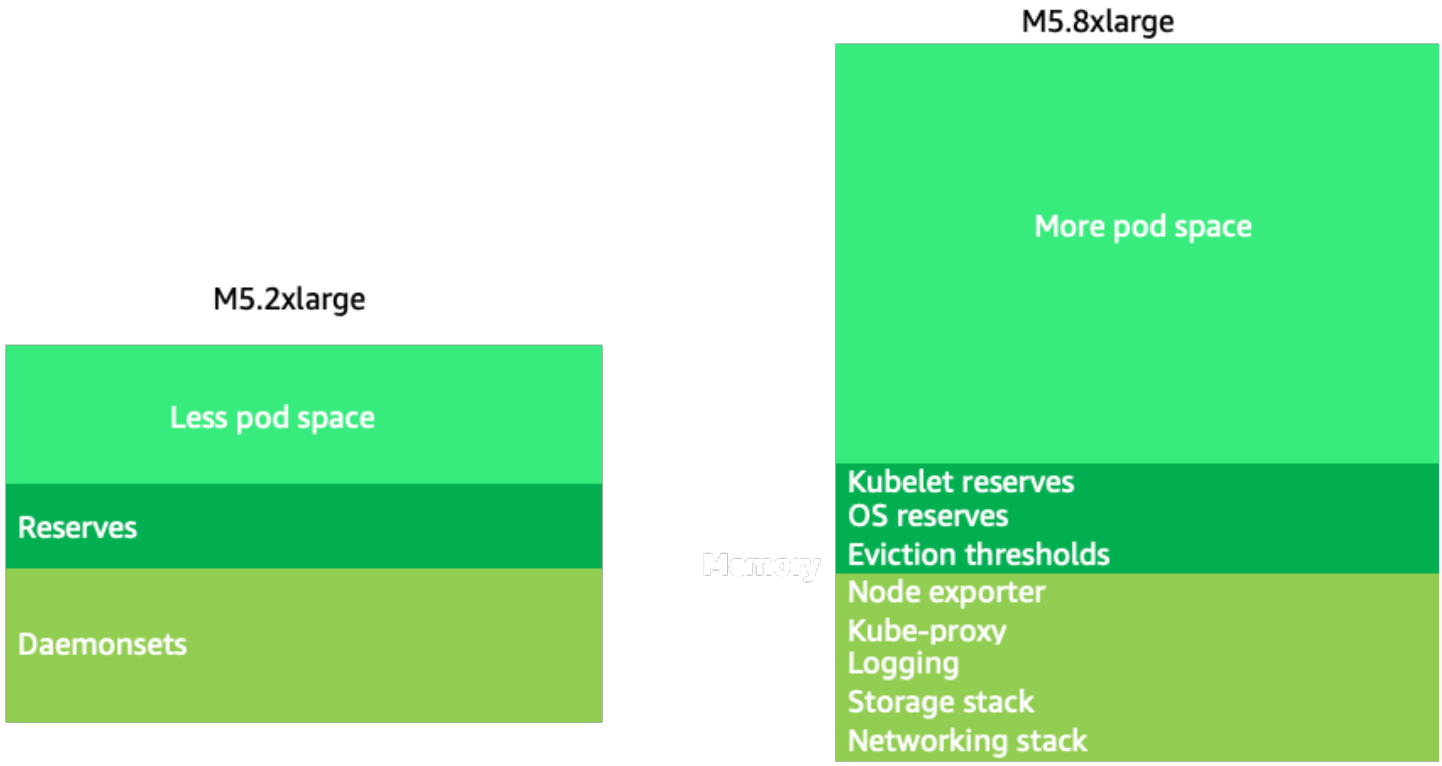
Efficienza dei nodi e dei carichi di lavoro

Essere efficienti con i nostri carichi di lavoro e nodi riduce al complexity/cost contempo le prestazioni e la scalabilità. Ci sono molti fattori da considerare quando si pianifica questa efficienza, ed è più facile pensare in termini di compromessi rispetto a un'impostazione basata sulle best practice per ogni funzionalità. Esploriamo questi compromessi in modo approfondito nella sezione seguente.

Selezione del nodo

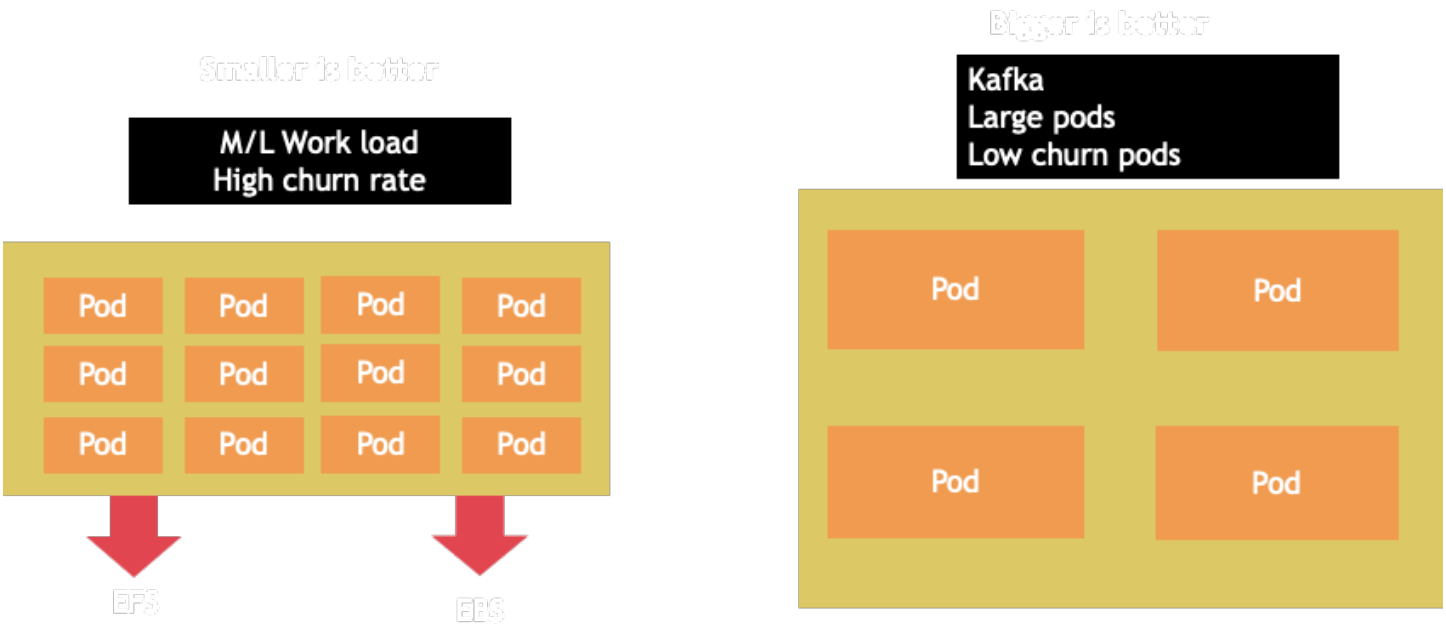
L'uso di nodi di dimensioni leggermente più grandi (4-12 volte più grandi) aumenta lo spazio disponibile per far funzionare i pod, poiché riduce la percentuale del nodo utilizzata per «sovraccarichi», ad esempio DaemonSetse Riserve per i componenti di sistema. Nel diagramma seguente vediamo la differenza tra lo spazio utilizzabile su un sistema 2xlarge e uno 8xlarge con solo un numero moderato di. DaemonSets

> **Note**
>
> Poiché k8s è scalabile orizzontalmente come regola generale, per la maggior parte delle applicazioni non ha senso prendere in considerazione l'impatto delle dimensioni dei nodi NUMA sulle prestazioni, da cui la raccomandazione di un intervallo inferiore a tale dimensione del nodo.



Le grandi dimensioni dei nodi ci consentono di avere una percentuale più elevata di spazio utilizzabile per nodo. Tuttavia, questo modello può essere portato all'estremo comprimendo il nodo con così tanti pod da causare errori o saturare il nodo. Il monitoraggio della saturazione dei nodi è fondamentale per utilizzare con successo nodi di dimensioni maggiori.

La selezione dei nodi è raramente una one-size-fits-all proposta. Spesso è meglio suddividere i carichi di lavoro con tassi di abbandono notevolmente diversi in diversi gruppi di nodi. I carichi di lavoro in batch di piccole dimensioni con un tasso di abbandono elevato sarebbero più adatti alla famiglia di istanze 4xlarge, mentre un'applicazione su larga scala come Kafka, che richiede 8 vCPU e ha un tasso di abbandono basso, sarebbe meglio gestita dalla famiglia 12xlarge.



> **Note**
>
> Un altro fattore da considerare con nodi di dimensioni molto grandi è dato dal fatto che i CGROUPS non nascondono il numero totale di vCPU all'applicazione containerizzata. I runtime dinamici possono spesso generare un numero involontario di thread del sistema operativo, creando una latenza difficile da risolvere. Per queste applicazioni si consiglia il pinning della CPU. Per un'esplorazione più approfondita dell'argomento, guarda il seguente https://www.youtube.com video/watch? v= _ NqtfDy KAqg

Node Bin-packing

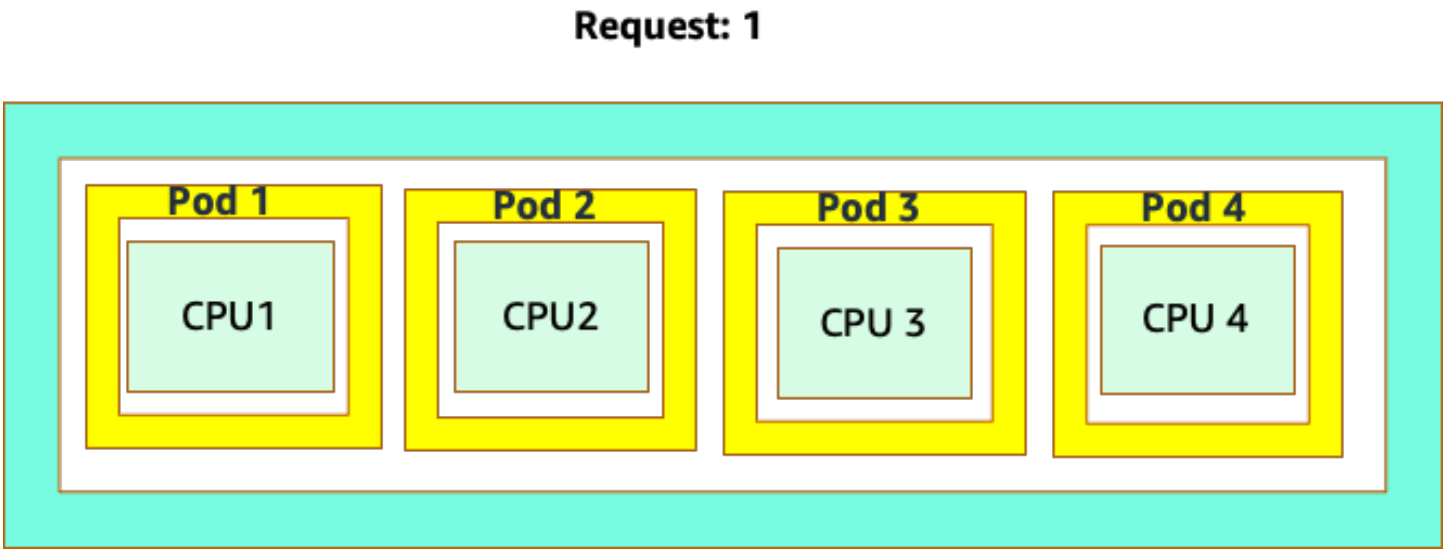
Kubernetes e regole Linux

Esistono due serie di regole a cui dobbiamo prestare attenzione quando gestiamo carichi di lavoro su Kubernetes. Le regole di Kubernetes Scheduler, che utilizza il valore di richiesta per pianificare i pod su un nodo, e poi cosa succede dopo la pianificazione del pod, che è l'ambito di Linux, non di Kubernetes.

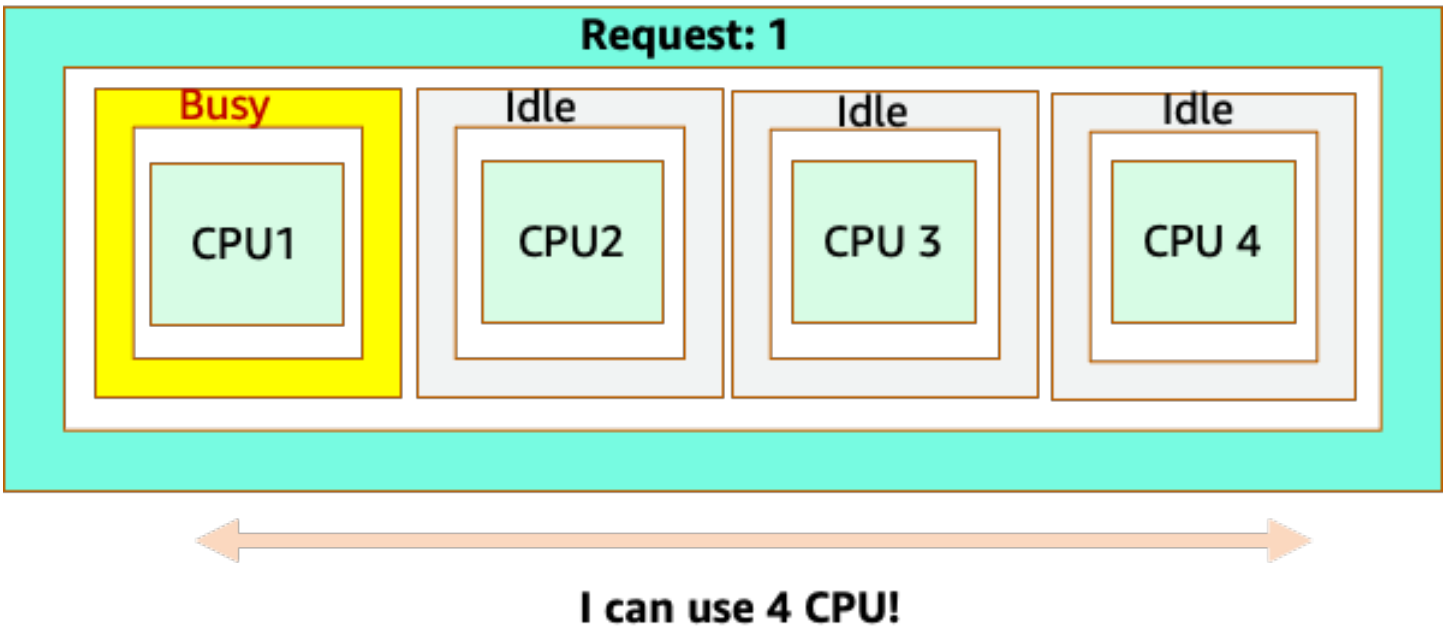
Al termine dello scheduler Kubernetes, subentra un nuovo set di regole, Linux Completely Fair Scheduler (CFS). Il punto fondamentale è che Linux CFS non ha il concetto di core. Spiegheremo perché pensare in base ai core può portare a gravi problemi di ottimizzazione dei carichi di lavoro in termini di scalabilità.

Pensare in base ai nuclei

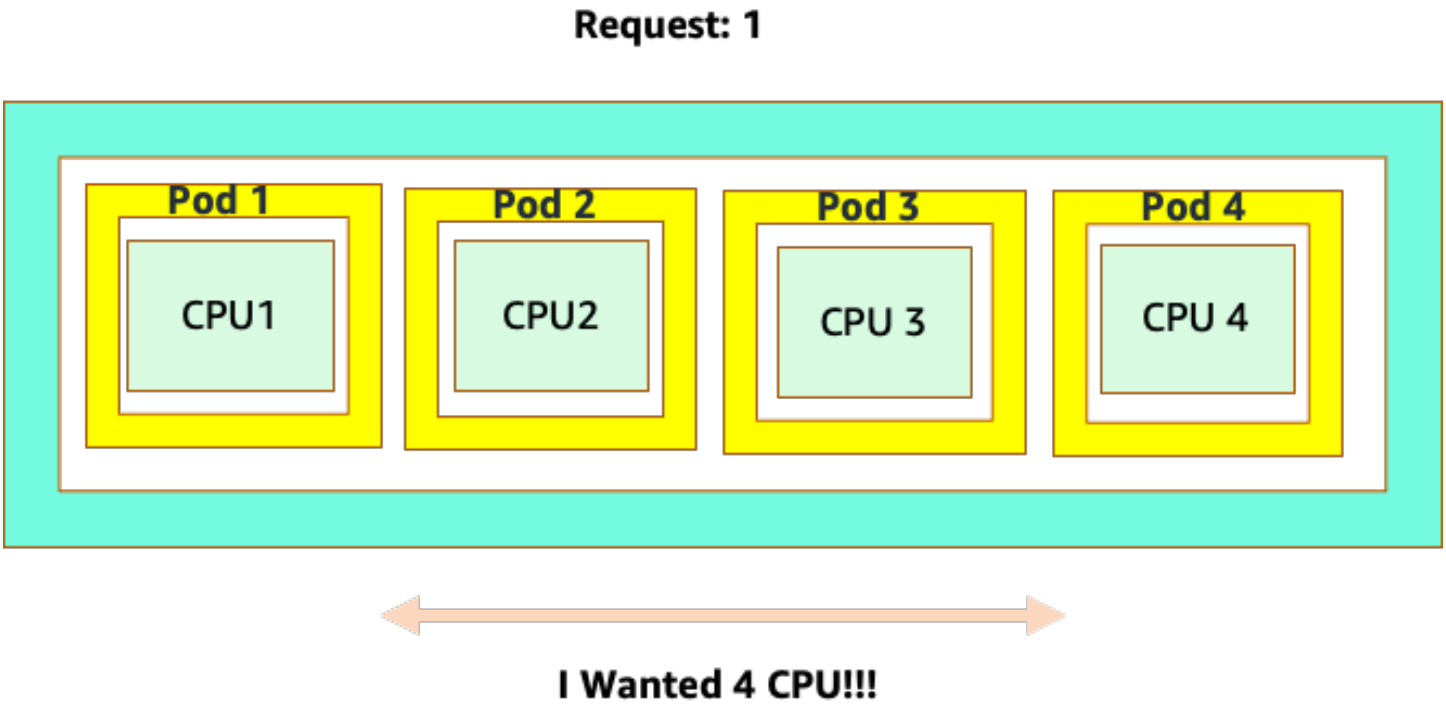
La confusione inizia perché lo scheduler Kubernetes ha il concetto di core. Dal punto di vista dello scheduler Kubernetes, se considerassimo un nodo con 4 pod NGINX, ciascuno con la richiesta di un set principale, il nodo avrebbe questo aspetto.



Tuttavia, facciamo un esperimento mentale su come questo aspetto sia diverso dal punto di vista di Linux CFS. La cosa più importante da ricordare quando si utilizza il sistema Linux CFS è che i container occupati (CGROUPS) sono gli unici contenitori che contano ai fini del sistema di condivisione. In questo caso, solo il primo contenitore è occupato, quindi è consentito utilizzare tutti e 4 i core del nodo.



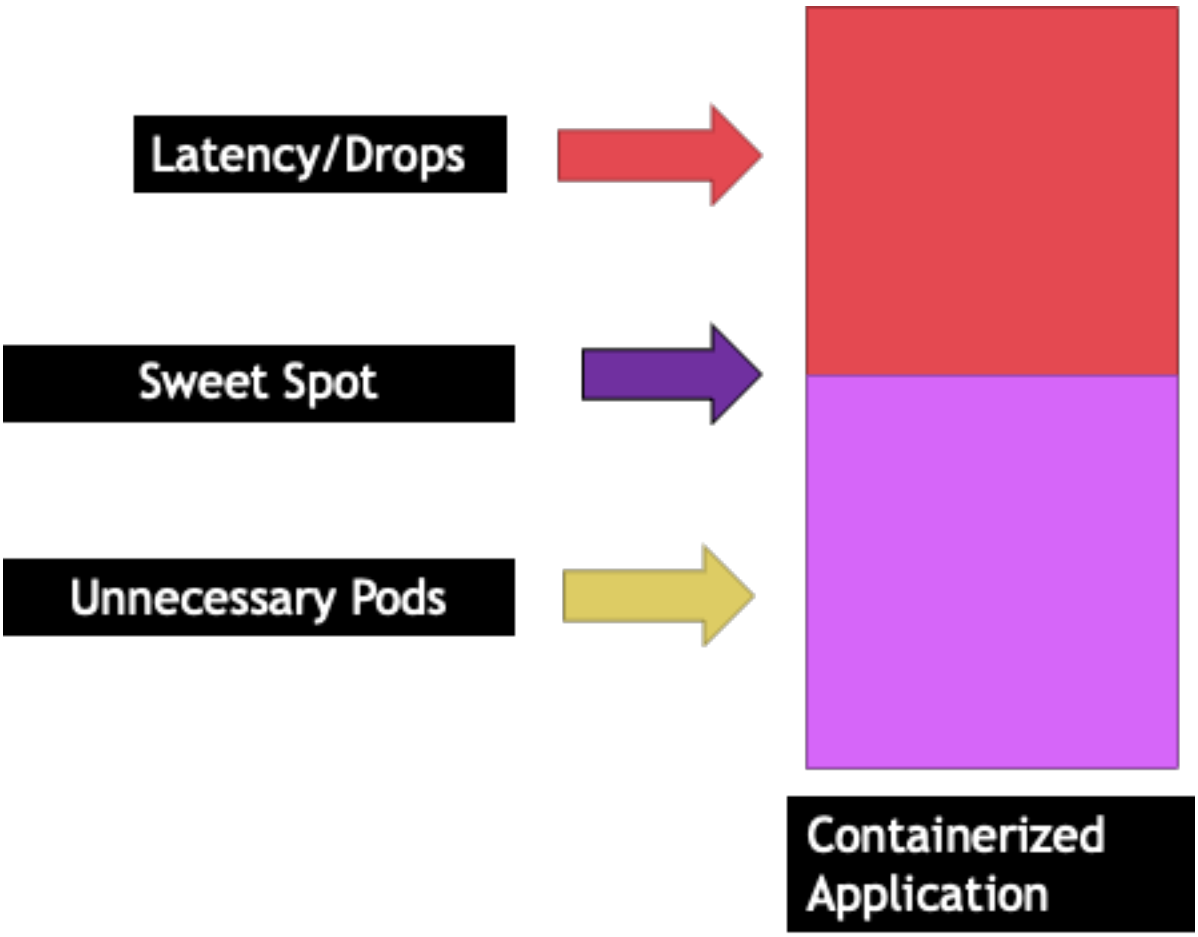
Perché questo è importante? Supponiamo di aver eseguito i nostri test delle prestazioni in un cluster di sviluppo in cui un'applicazione NGINX era l'unico contenitore occupato su quel nodo. Quando spostiamo l'app in produzione, accade quanto segue: l'applicazione NGINX richiede 4 vCPU di risorse, tuttavia, poiché tutti gli altri pod sul nodo sono occupati, le prestazioni della nostra app sono limitate.



Questa situazione ci porterebbe ad aggiungere inutilmente altri contenitori perché non consentivamo alle nostre applicazioni di scalare fino al loro «`punto dolce"`. Esploriamo questo importante concetto di a `"sweet spot"` in modo un po' più dettagliato.

Dimensionamento corretto dell'applicazione

Ogni applicazione ha un certo punto in cui non può più ricevere traffico. Superare questo limite può aumentare i tempi di elaborazione e persino ridurre il traffico se viene spinto ben oltre questo punto. Questo è noto come punto di saturazione dell'applicazione. Per evitare problemi di scalabilità, dovremmo tentare di scalare l'applicazione prima che raggiunga il punto di saturazione. Chiamiamo questo punto il punto debole.



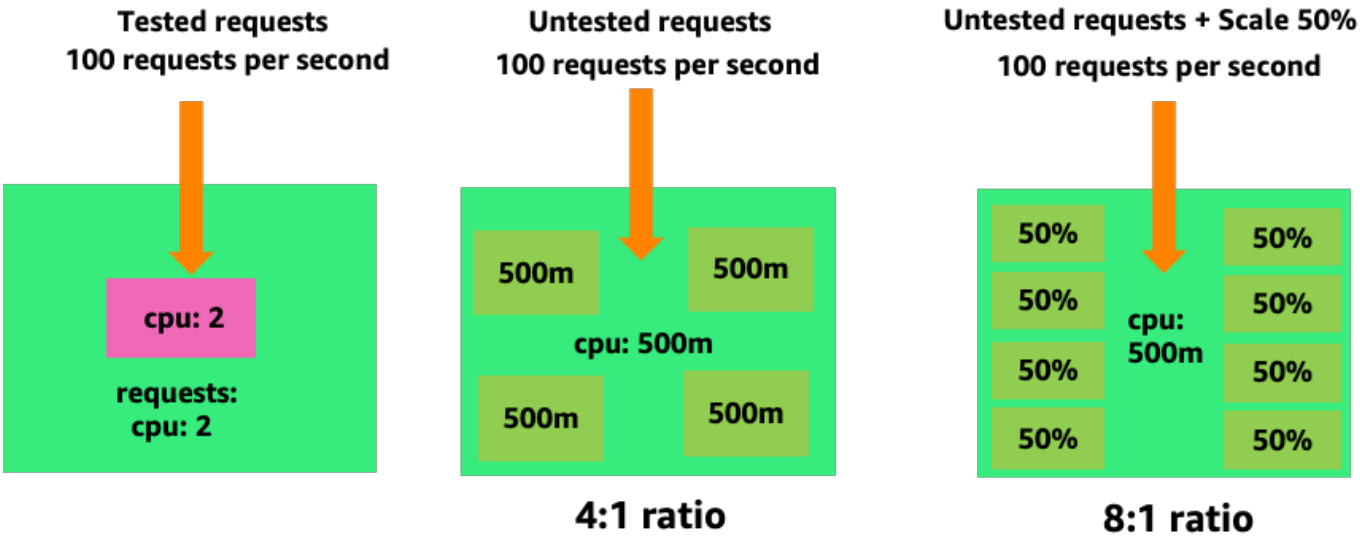
Dobbiamo testare ciascuna delle nostre applicazioni per comprenderne il punto debole. Non ci sarà una guida universale in questo caso poiché ogni applicazione è diversa. Durante questo test stiamo cercando di capire la metrica migliore che mostri il punto di saturazione delle nostre applicazioni. Spesso, le metriche di utilizzo vengono utilizzate per indicare che un'applicazione è satura, ma ciò può portare rapidamente a problemi di scalabilità (esploreremo questo argomento in dettaglio in una sezione successiva). Una volta raggiunto questo «`punto dolce"`, possiamo utilizzarlo per scalare in modo efficiente i nostri carichi di lavoro.

Al contrario, cosa succederebbe se aumentassimo ben prima del punto debole e creassimo pod non necessari? Esploriamolo nella prossima sezione.

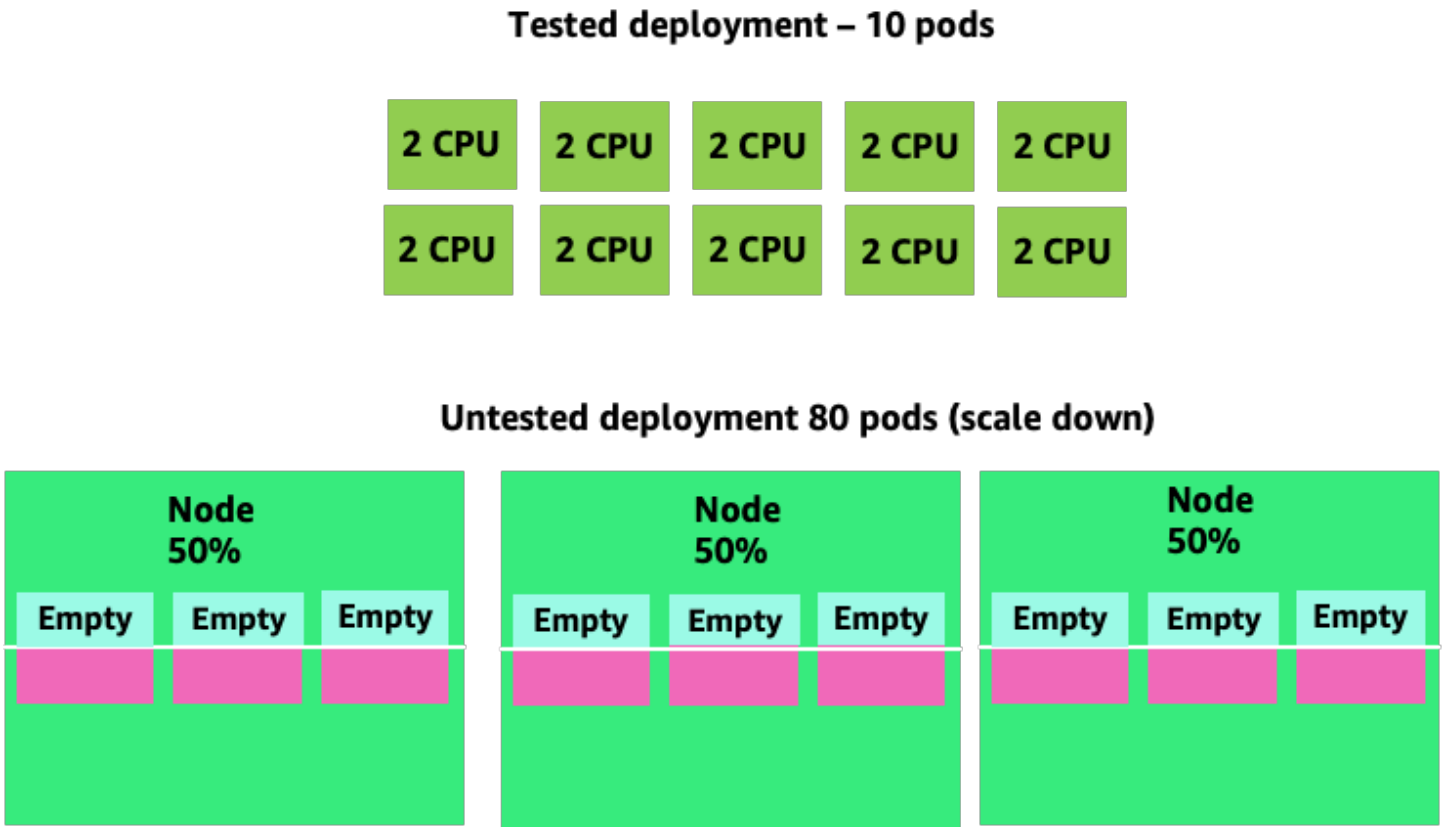
Espansione dei baccelli

Per vedere come la creazione di pod inutili possa rapidamente sfuggire di mano, diamo un'occhiata al primo esempio a sinistra. La scala verticale corretta di questo contenitore richiede circa due volte di utilizzo quando si gestiscono 100 richieste al secondo. CPUs Tuttavia, se dovessimo sottodimensionare il valore delle richieste impostando le richieste su mezzo core, ora avremmo

bisogno di 4 pod per ogni pod di cui abbiamo effettivamente bisogno. Ad aggravare ulteriormente il problema, se il nostro HPA fosse impostato sul valore predefinito del 50% di CPU, quei pod verrebbero ridimensionati a metà vuoto, creando un rapporto di 8:1.

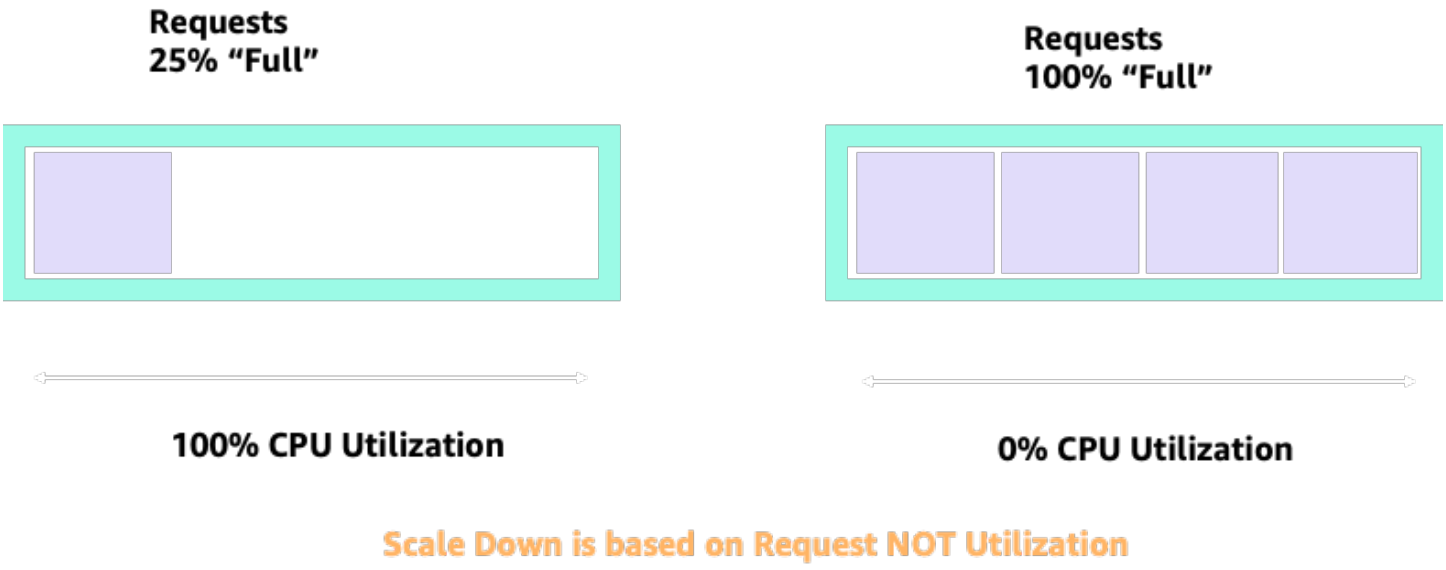


Aumentando questo problema possiamo vedere rapidamente come questo possa sfuggire di mano. Un'implementazione di dieci pod il cui punto ottimale è stato impostato in modo errato potrebbe rapidamente arrivare a 80 pod e all'infrastruttura aggiuntiva necessaria per gestirli.



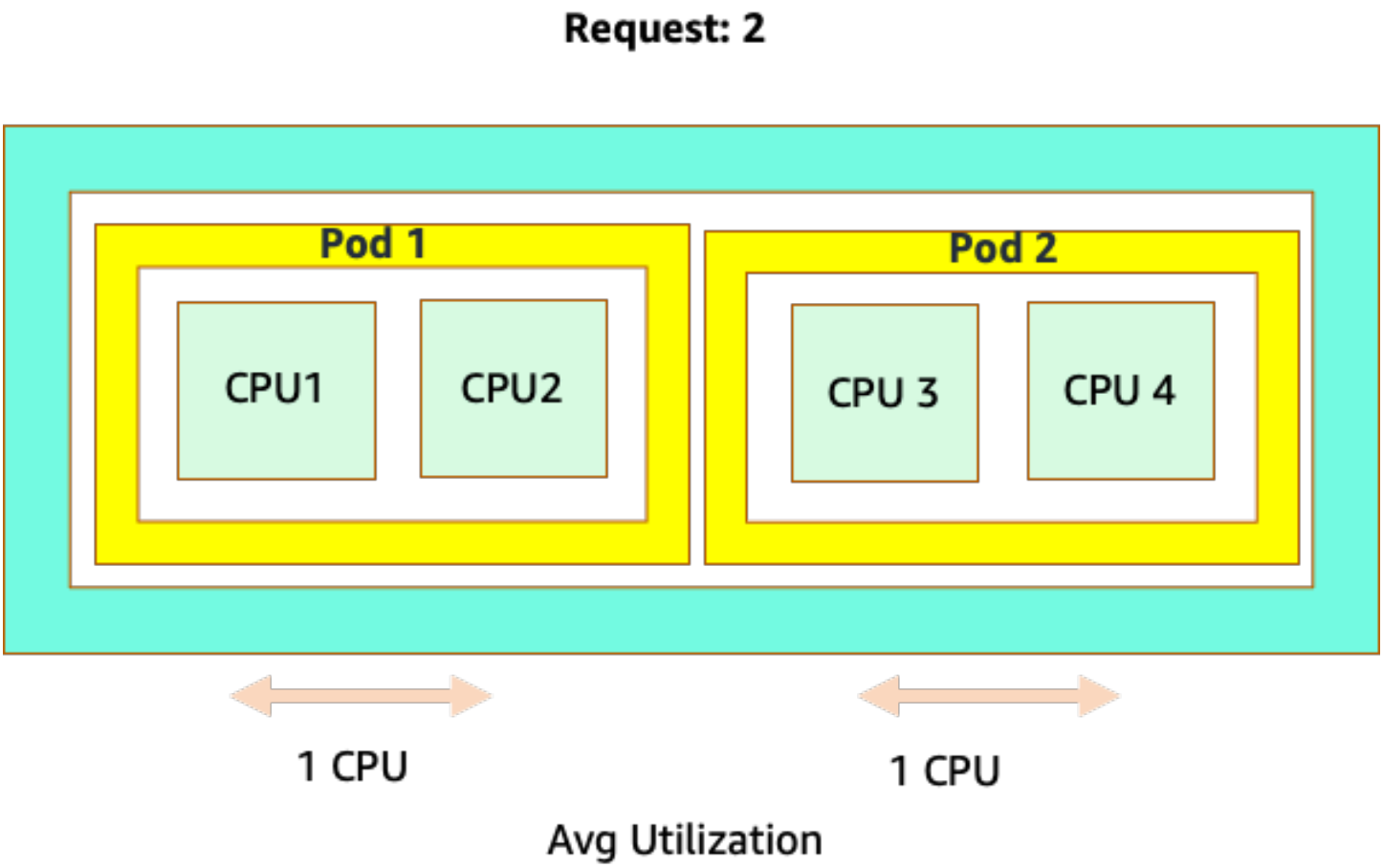
Ora che comprendiamo l'impatto di non consentire alle applicazioni di funzionare nel loro punto ideale, torniamo al livello dei nodi e chiediamoci perché questa differenza tra lo scheduler Kubernetes e Linux CFS è così importante?

In caso di scalabilità verso l'alto e verso il basso con HPA, possiamo avere uno scenario in cui abbiamo molto spazio per allocare più pod. Questa sarebbe una decisione sbagliata perché il nodo raffigurato a sinistra utilizza già la CPU al 100%. In uno scenario irrealistico ma teoricamente possibile, potremmo avere l'altro estremo in cui il nostro nodo è completamente pieno, ma l'utilizzo della CPU è pari a zero.

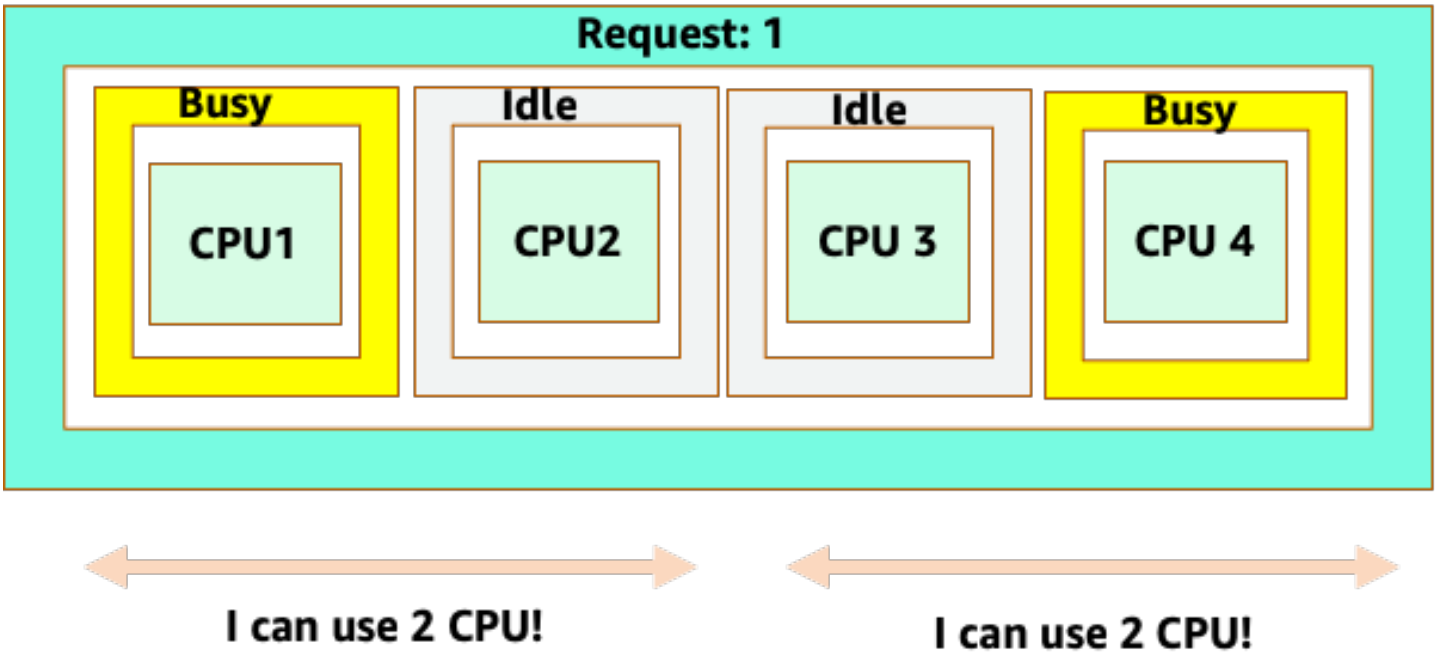


Impostazione delle richieste

Si sarebbe tentati di impostare la richiesta sul valore «ottimale» per quell'applicazione, tuttavia ciò causerebbe inefficienze, come illustrato nel diagramma seguente. Qui abbiamo impostato il valore di richiesta su 2 vCPU, tuttavia l'utilizzo medio di questi pod richiede solo 1 CPU per la maggior parte del tempo. Questa impostazione ci farebbe sprecare il 50% dei cicli della CPU, il che sarebbe inaccettabile.



Questo ci porta alla risposta complessa al problema. L'utilizzo dei container non può essere concepito in modo isolato; è necessario tenere conto delle altre applicazioni in esecuzione sul nodo. Nell'esempio seguente, i contenitori di natura esplosiva vengono mescolati con due contenitori a basso utilizzo della CPU che potrebbero avere limiti di memoria. In questo modo consentiamo ai container di raggiungere il loro punto di forza senza gravare sul nodo.



Il concetto importante da trarre da tutto ciò è che l'utilizzo del concetto di core di pianificazione Kubernetes per comprendere le prestazioni dei container Linux può portare a decisioni sbagliate in quanto non sono correlati.

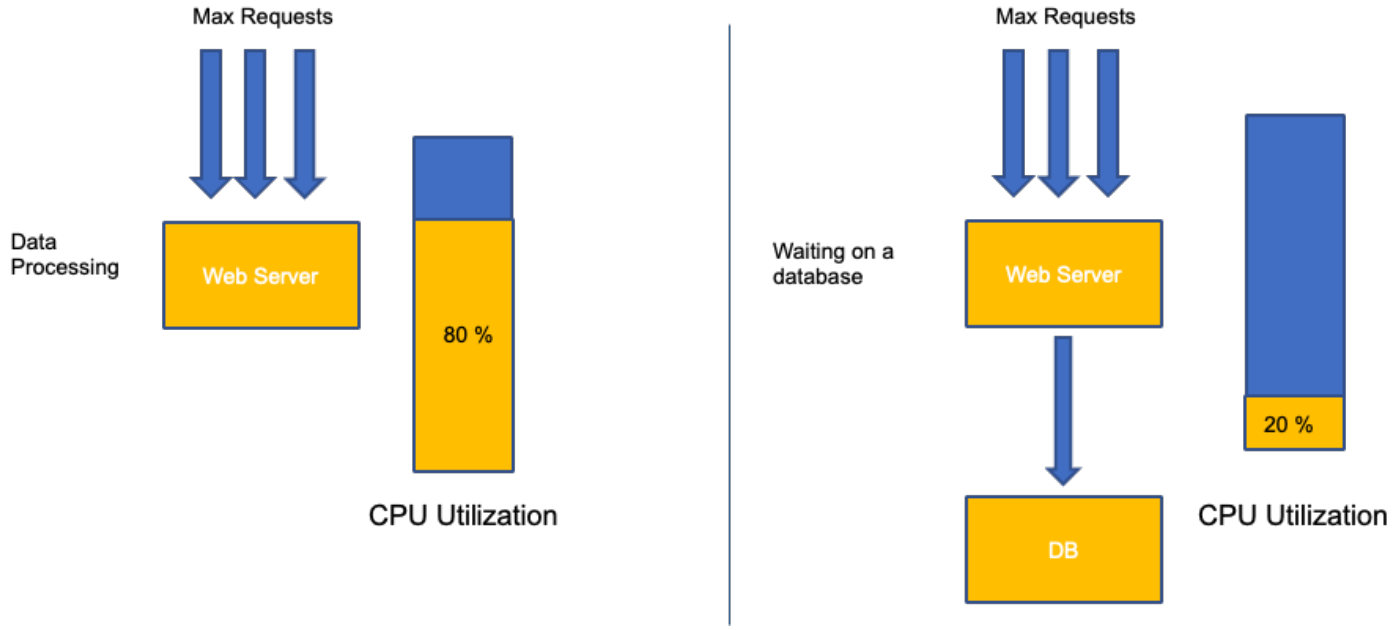
> **Note**
>
> Linux CFS ha i suoi punti di forza. Ciò è particolarmente vero per i carichi di lavoro I/O basati su sistemi basati. Tuttavia, se l'applicazione utilizza core completi senza sidecar e non ha I/O requisiti, il pinning della CPU può eliminare una notevole complessità da questo processo ed è consigliato con questi avvertimenti.

Utilizzo vs. saturazione

Un errore comune nella scalabilità delle applicazioni consiste nell'utilizzare solo l'utilizzo della CPU per la metrica di scalabilità. Nelle applicazioni complesse questo è quasi sempre un indicatore insufficiente del fatto che un'applicazione sia effettivamente satura di richieste. Nell'esempio a sinistra, vediamo che tutte le nostre richieste arrivano effettivamente al server web, quindi l'utilizzo della CPU segue bene la saturazione.

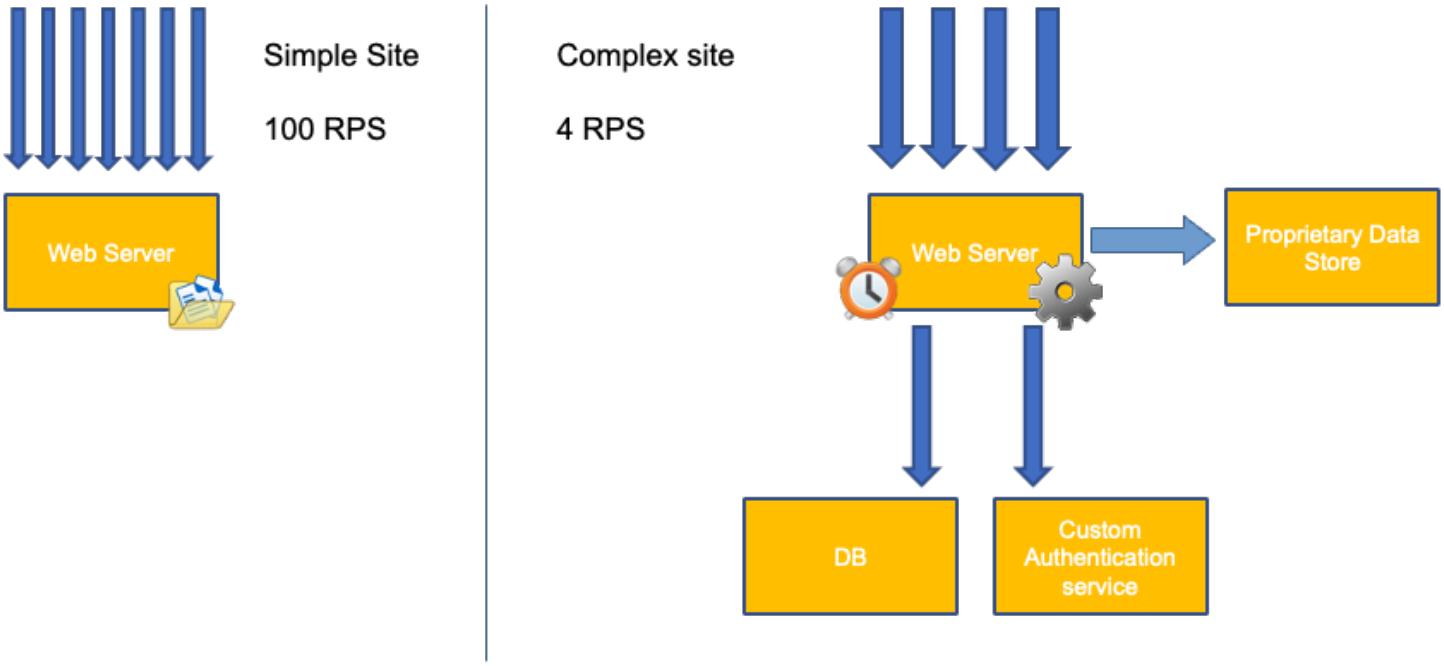
Nelle applicazioni del mondo reale, è probabile che alcune di queste richieste vengano gestite da un livello di database o da un livello di autenticazione, ecc. In questo caso più comune, notate che la

CPU non esegue il tracciamento con saturazione poiché la richiesta viene gestita da altre entità. In questo caso la CPU è un indicatore di saturazione molto scarso.



L'utilizzo di una metrica errata nelle prestazioni delle applicazioni è il motivo principale per cui il ridimensionamento non necessario e imprevedibile in Kubernetes. È necessario prestare molta attenzione nella scelta della metrica di saturazione corretta per il tipo di applicazione che si sta utilizzando. È importante notare che non è possibile fornire una raccomandazione valida per tutti. A seconda della lingua utilizzata e del tipo di applicazione in questione, esiste una serie diversificata di metriche per la saturazione.

Potremmo pensare che questo problema riguardi solo l'utilizzo della CPU, tuttavia anche altre metriche comuni come la richiesta al secondo possono rientrare nello stesso problema discusso in precedenza. Nota che la richiesta può anche essere indirizzata a livelli DB, livelli di autenticazione, non essendo gestita direttamente dal nostro server web, quindi è una metrica scadente per la vera saturazione del server web stesso.



Sfortunatamente non ci sono risposte facili quando si tratta di scegliere la giusta metrica di saturazione. Ecco alcune linee guida da tenere in considerazione:

- Comprendete il tempo di esecuzione del linguaggio: le lingue con più thread del sistema operativo reagiranno in modo diverso rispetto alle applicazioni a thread singolo, influendo quindi in modo diverso sul nodo.
- Comprendi la scala verticale corretta: quanto buffer vuoi inserire nella scala verticale delle tue applicazioni prima di scalare un nuovo pod?
- Quali metriche riflettono realmente la saturazione della tua applicazione - La metrica di saturazione per un Kafka Producer sarebbe molto diversa da quella di un'applicazione web complessa.
- In che modo tutte le altre applicazioni sul nodo si influenzano a vicenda? Le prestazioni delle applicazioni non avvengono in modo isolato, gli altri carichi di lavoro sul nodo hanno un impatto importante.

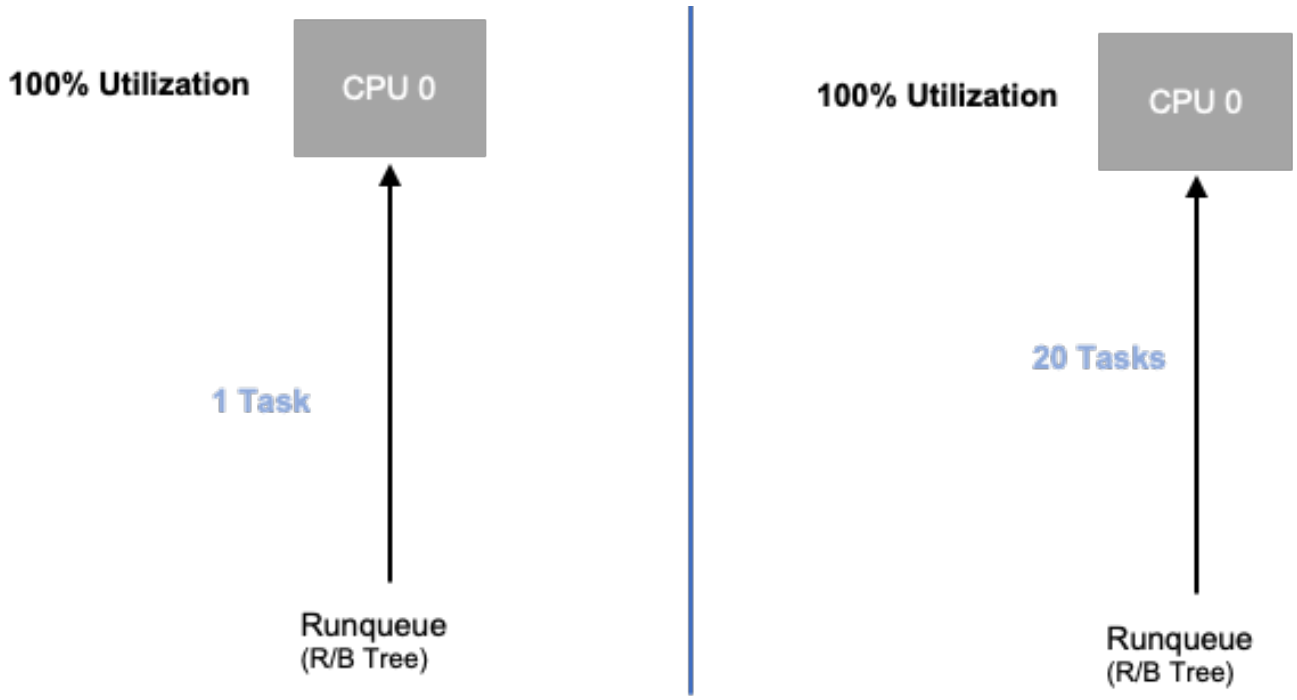
Per chiudere questa sezione, sarebbe facile considerare quanto sopra riportato come eccessivamente complesso e non necessario. Spesso può succedere che si verifichi un problema, ma non siamo consapevoli della vera natura del problema perché stiamo esaminando le metriche sbagliate. Nella prossima sezione vedremo come ciò potrebbe accadere.

Saturazione dei nodi

Ora che abbiamo esplorato la saturazione delle applicazioni, esaminiamo lo stesso concetto dal punto di vista dei nodi. Prendiamone due CPUs utilizzati al 100% per vedere la differenza tra utilizzo e saturazione.

La vCPU a sinistra è utilizzata al 100%, tuttavia non ci sono altre attività in attesa di essere eseguite su questa vCPU, quindi in senso puramente teorico, è abbastanza efficiente. Nel frattempo, nel secondo esempio abbiamo 20 applicazioni a thread singolo in attesa di essere elaborate da una vCPU. Tutte le 20 applicazioni ora sperimenteranno un certo tipo di latenza mentre attendono il loro turno per essere elaborate dalla vCPU. In altre parole, la vCPU sulla destra è satura.

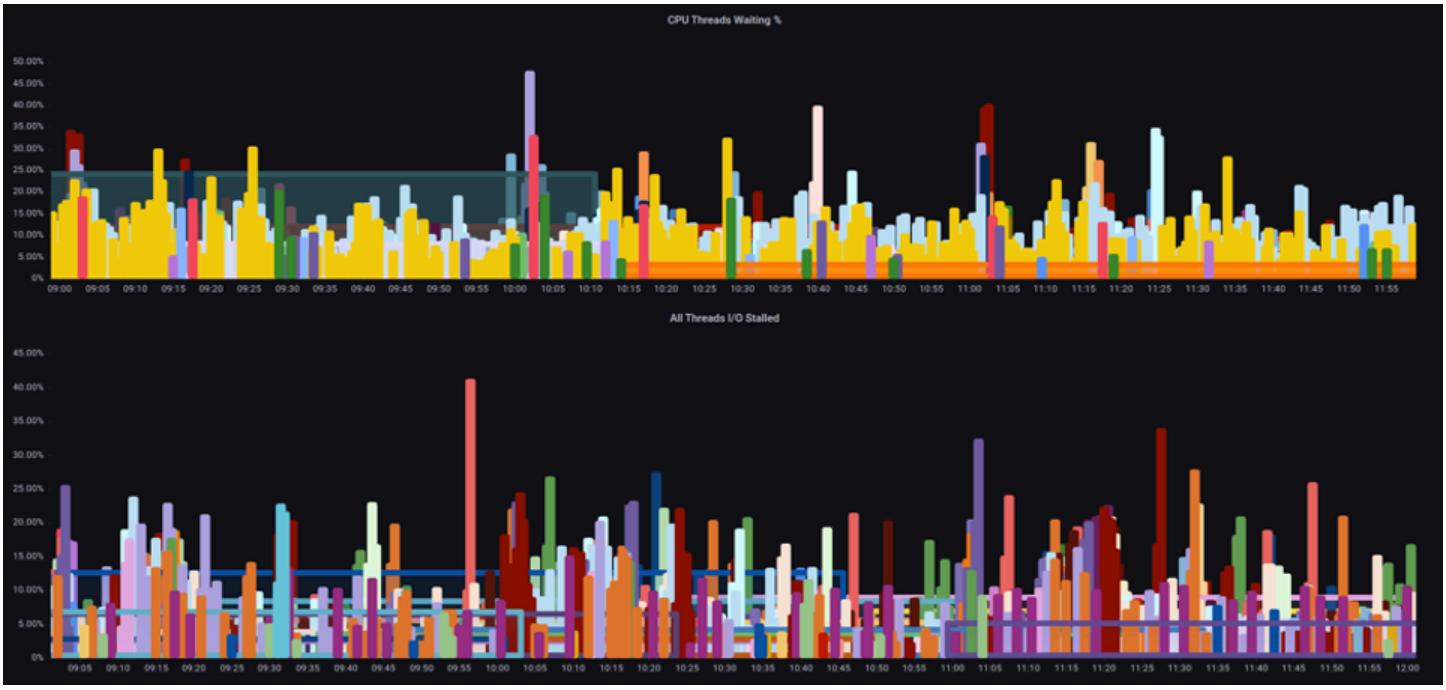
Non solo non vedremmo questo problema se ci limitassimo a guardare all'utilizzo, ma potremmo attribuire questa latenza a qualcosa di non correlato, come la rete, che ci condurrebbe sulla strada sbagliata.



È importante visualizzare le metriche di saturazione, non solo le metriche di utilizzo, quando si aumenta il numero totale di pod in esecuzione su un nodo in un dato momento, poiché è facile non notare che un nodo è troppo saturo. Per questo compito possiamo utilizzare le metriche relative alle informazioni sullo stallo della pressione, come illustrato nella tabella seguente.

ProMQL - I/O bloccato

```
topk(3, ((irate(node_pressure_io_stalled_seconds_total[1m])) * 100))
```



> **Note**
>
> Per ulteriori informazioni sulle metriche di stallo di Pressure, consulta https://facebookmicrosites.github. io/psi/docs/overview*

Con queste metriche possiamo capire se i thread sono in attesa sulla CPU o anche se tutti i thread sulla scatola sono bloccati in attesa di risorse come la memoria o I/O. For example, we could see what percentage every thread on the instance was stalled waiting on I/O per un periodo di 1 minuto.

```
topk(3, ((irate(node_pressure_io_stalled_seconds_total[1m])) * 100))
```

Usando questa metrica, possiamo vedere nel grafico qui sopra che ogni thread sulla console è rimasto bloccato il 45% del tempo di attesa I/O al picco massimo, il che significa che in quel minuto stavamo buttando via tutti i cicli della CPU. Capire che ciò sta accadendo può aiutarci a recuperare una notevole quantità di tempo vCPU, rendendo così la scalabilità più efficiente.

HPA V2

Si consiglia di utilizzare la versione autoscaling/v2 dell'API HPA. Le versioni precedenti dell'API HPA potrebbero bloccare la scalabilità in alcuni casi limite. Inoltre, i pod si limitavano a raddoppiare solo durante ogni fase di scalabilità, il che creava problemi per le piccole implementazioni che dovevano scalare rapidamente.

AutoScaling/v2 ci consente una maggiore flessibilità nell'includere più criteri di scalabilità e ci consente una grande flessibilità nell'utilizzo di metriche personalizzate ed esterne (metriche diverse da K8s).

Ad esempio, possiamo scalare in base al più alto dei tre valori (vedi sotto). Scaliamo se l'utilizzo medio di tutti i pod è superiore al 50%, se i parametri personalizzati indicano che i pacchetti al secondo di ingresso superano la media di 1.000 o l'oggetto in ingresso supera le 10.000 richieste al secondo.

> **Note**
>
> Questo è solo per dimostrare la flessibilità dell'API di auto-scaling, che consigliamo di evitare regole eccessivamente complesse che possono essere difficili da risolvere in produzione.

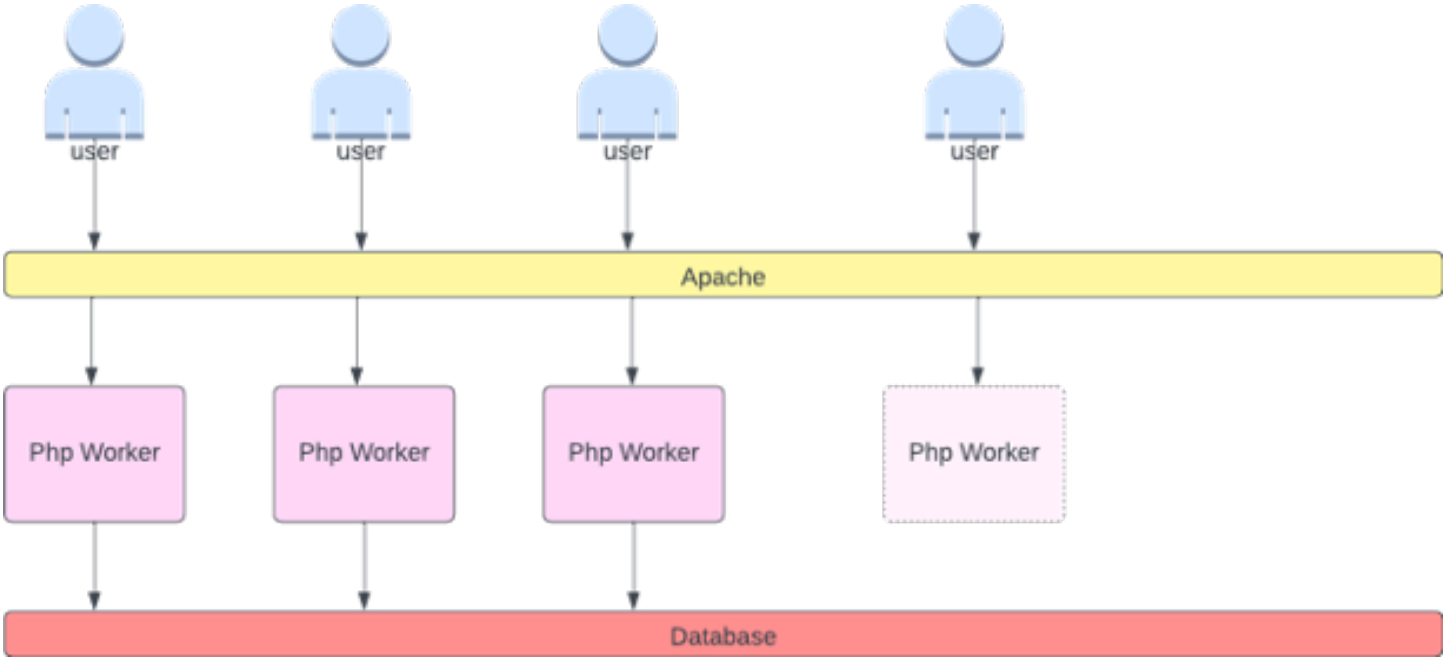
```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: php-apache
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: php-apache
  minReplicas: 1
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 50
  - type: Pods
```

```
    pods:
      metric:
        name: packets-per-second
      target:
        type: AverageValue
        averageValue: 1k
  - type: Object
    object:
      metric:
        name: requests-per-second
      describedObject:
        apiVersion: networking.k8s.io/v1
        kind: Ingress
        name: main-route
      target:
        type: Value
        value: 10k
```

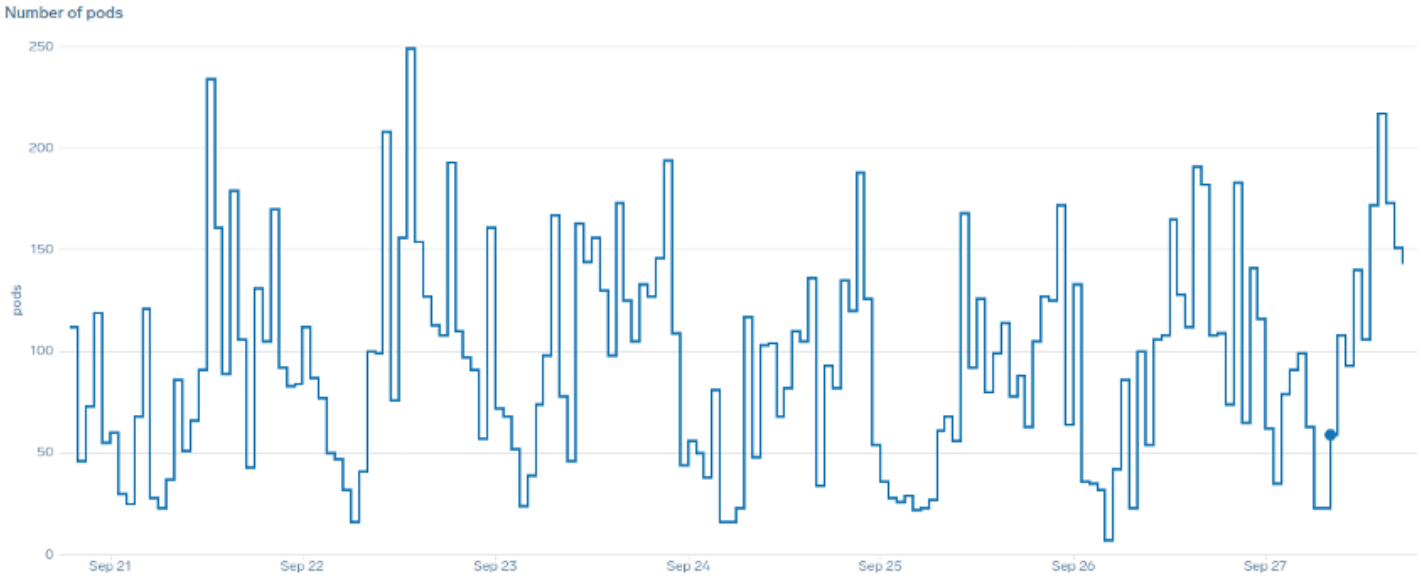
Tuttavia, abbiamo appreso il pericolo dell'utilizzo di tali metriche per applicazioni web complesse. In questo caso sarebbe meglio utilizzare metriche personalizzate o esterne che riflettano accuratamente la saturazione della nostra applicazione rispetto all'utilizzo. HPAv2 lo consente grazie alla possibilità di scalare in base a qualsiasi metrica, tuttavia dobbiamo comunque trovare ed esportare quella metrica su Kubernetes per utilizzarla.

Ad esempio, possiamo esaminare il numero di thread attivi in Apache. Questo spesso crea un profilo di scalabilità «più fluido» (maggiori informazioni su questo termine presto). Se un thread è attivo, non importa se quel thread è in attesa su un livello di database o sta gestendo una richiesta localmente, se vengono utilizzati tutti i thread dell'applicazione, è un'ottima indicazione che l'applicazione è satura.

Possiamo usare questo esaurimento dei thread come segnale per creare un nuovo pod con un pool di thread completamente disponibile. Questo ci dà anche il controllo sulla dimensione del buffer che vogliamo assorbire nell'applicazione durante i periodi di traffico intenso. Ad esempio, se avessimo un pool di thread totale di 10, il ridimensionamento a 4 thread utilizzati rispetto agli 8 thread utilizzati avrebbe un impatto notevole sul buffer disponibile per il ridimensionamento dell'applicazione. Un'impostazione di 4 sarebbe utile per un'applicazione che deve scalare rapidamente in presenza di carichi pesanti, mentre un'impostazione di 8 sarebbe più efficiente con le nostre risorse se avessimo molto tempo per scalare a causa del numero di richieste che aumenta lentamente anziché bruscamente nel tempo.

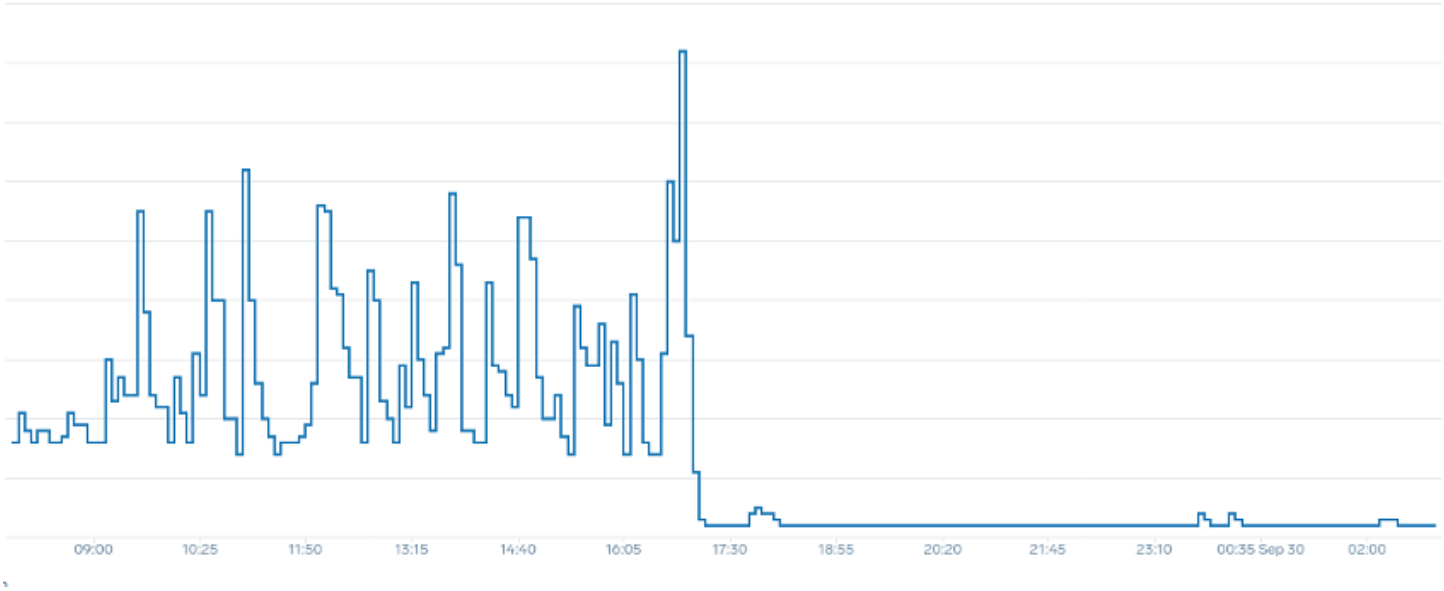


Cosa intendiamo con il termine «fluido» quando si parla di scalabilità? Notate il grafico seguente in cui utilizziamo la CPU come metrica. I pod utilizzati in questa implementazione aumenteranno in breve tempo, passando da 50 a 250 pod, per poi ridurli immediatamente di nuovo. Si tratta di un fenomeno altamente inefficiente: la scalabilità è la causa principale del tasso di abbandono dei cluster.



Notate come, dopo aver scelto una metrica che rifletta lo sweet spot corretto della nostra applicazione (la parte centrale del grafico), siamo in grado di scalare senza problemi. La nostra scalabilità è ora efficiente e i nostri pod possono scalare completamente con lo spazio di crescita che abbiamo fornito modificando le impostazioni delle richieste. Ora un gruppo più ristretto di pod sta facendo il lavoro che

centinaia di pod facevano prima. I dati del mondo reale mostrano che questo è il fattore numero uno nella scalabilità dei cluster Kubernetes.

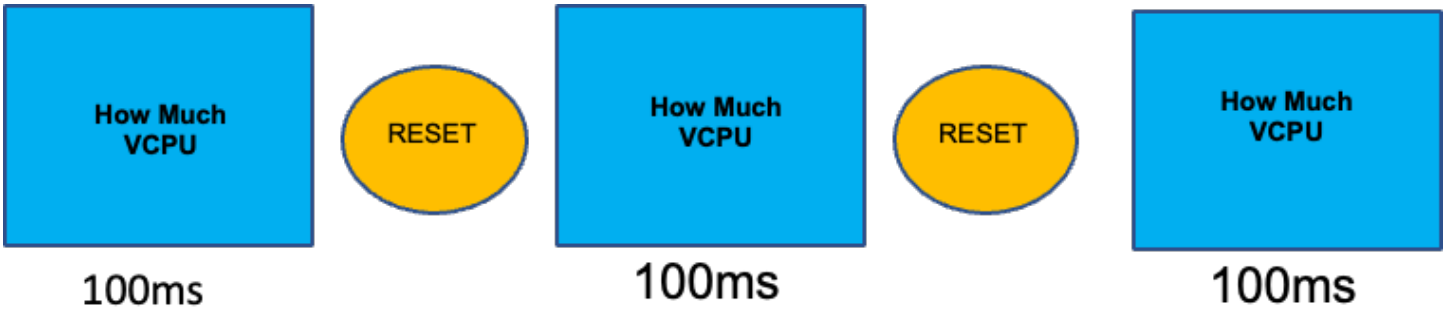


Il punto fondamentale è che l'utilizzo della CPU è solo una dimensione delle prestazioni delle applicazioni e dei nodi. L'utilizzo della CPU come unico indicatore dello stato di salute dei nostri nodi e delle nostre applicazioni crea problemi di scalabilità, prestazioni e costi, tutti concetti strettamente collegati. Più l'applicazione e i nodi sono performanti, meno è necessario scalare, il che a sua volta riduce i costi.

L'individuazione e l'utilizzo delle metriche di saturazione corrette per la scalabilità di una particolare applicazione consente inoltre di monitorare e segnalare eventuali problemi di tale applicazione. Se questo passaggio fondamentale viene saltato, le segnalazioni di problemi di prestazioni saranno difficili, se non impossibili, da comprendere.

Impostazione dei limiti della CPU

Per completare questa sezione su argomenti fraintesi, tratteremo i limiti della CPU. In breve, i limiti sono i metadati associati al contenitore che ha un contatore che si resetta ogni 100 ms. Questo aiuta Linux a tenere traccia di quante risorse CPU vengono utilizzate a livello di nodo da un contenitore specifico in un periodo di tempo di 100 ms.



Un errore comune nell'impostazione dei limiti consiste nel presupporre che l'applicazione sia a thread singolo e che sia in esecuzione solo sulla vCPU «`assigned"`. Nella sezione precedente abbiamo appreso che CFS non assegna core, e in realtà un container che esegue pool di thread di grandi dimensioni eseguirà la pianificazione su tutte le vCPU disponibili sulla confezione.

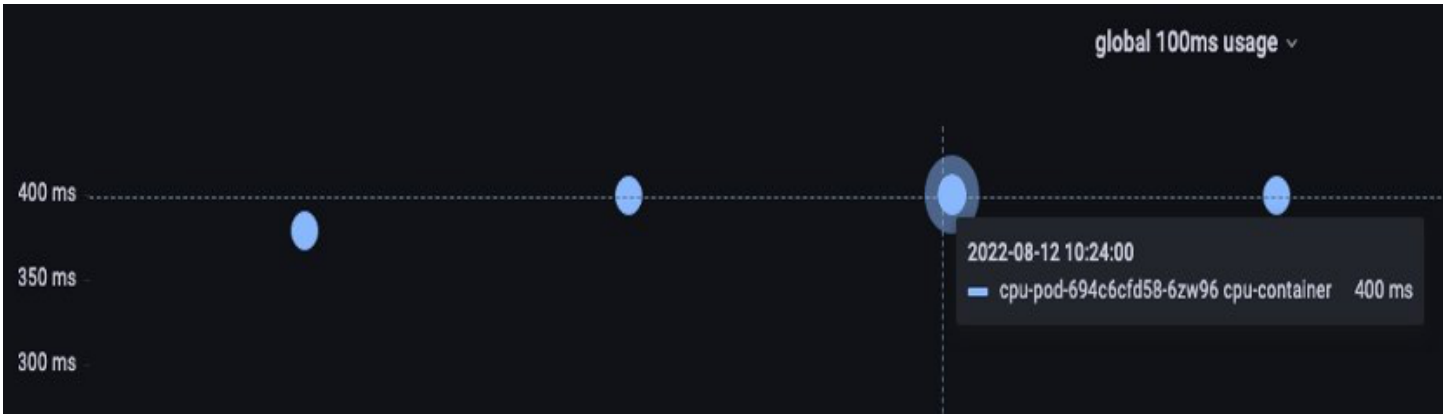
Se 64 thread del sistema operativo sono in esecuzione su 64 core disponibili (dal punto di vista di un nodo Linux), sommando il tempo di esecuzione su tutti i 64 core, il totale del tempo impiegato dalla CPU in un periodo di 100 ms sarà abbastanza grande. Poiché ciò può verificarsi solo durante un processo di raccolta dei rifiuti, può essere abbastanza facile non accorgersi di qualcosa del genere. Questo è il motivo per cui è necessario utilizzare metriche per garantire l'utilizzo corretto nel tempo prima di tentare di impostare un limite.

Fortunatamente, abbiamo un modo per vedere esattamente quanta vCPU viene utilizzata da tutti i thread di un'applicazione. Useremo la metrica `container_cpu_usage_seconds_total` per questo scopo.

Poiché la logica di throttling si verifica ogni 100 ms e questa metrica è una metrica al secondo, utilizzeremo ProMQL in modo che corrisponda a questo periodo di 100 ms. Se desideri approfondire questo lavoro sull'istruzione PromQL, consulta il seguente blog.

Interrogazione PromQL:

```
topk(3, max by (pod, container)(rate(container_cpu_usage_seconds_total{image!="",
 instance="$instance"}[$__rate_interval]))) / 10
```



Una volta che sentiamo di avere il giusto valore, possiamo porre il limite alla produzione. Diventa quindi necessario verificare se la nostra applicazione viene limitata a causa di qualcosa di imprevisto. Possiamo farlo guardando `container_cpu_throttled_seconds_total`

```
topk(3, max by (pod, container)(rate(container_cpu_cfs_throttled_seconds_total{image!
=``""``, instance=``"$instance"``}[$__rate_interval]))) / 10
```

Memoria

L'allocazione della memoria è un altro esempio in cui è facile confondere il comportamento di pianificazione di Kubernetes con il comportamento di Linux. CGroup Si tratta di un argomento più articolato, in quanto sono state apportate importanti modifiche al modo in cui la versione CGroup 2 gestisce la memoria in Linux e Kubernetes ha modificato la sua sintassi in base a ciò; leggi questo blog per ulteriori dettagli.

A differenza delle richieste della CPU, le richieste di memoria non vengono utilizzate dopo il completamento del processo di pianificazione. Questo perché non possiamo comprimere la memoria nella CGroup v1 nello stesso modo in cui possiamo comprimere la CPU. Questo ci lascia solo

dei limiti di memoria, progettati per fungere da protezione contro le perdite di memoria chiudendo completamente il pod. Si tratta di una proposta in stile «tutto o niente», tuttavia ora ci sono stati dati nuovi modi per affrontare questo problema.

Innanzitutto, è importante capire che impostare la giusta quantità di memoria per i contenitori non è così semplice come sembra. Il file system di Linux utilizzerà la memoria come cache per migliorare le prestazioni. Questa cache crescerà nel tempo e può essere difficile sapere quanta memoria sia utile per la cache, ma può essere recuperata senza un impatto significativo sulle prestazioni delle applicazioni. Ciò si traduce spesso in un'interpretazione errata dell'utilizzo della memoria.

La capacità di «comprimere» la memoria era uno dei fattori principali alla base della v2. CGroup Per saperne di più sui motivi per cui la CGroup V2 era necessaria, consulta la presentazione di Chris Down, in LISA21 cui spiega perché l'impossibilità di impostare correttamente la memoria minima è stata una delle ragioni che lo hanno spinto a creare le metriche di CGroup v2 e di stallo della pressione.

Fortunatamente, Kubernetes ora ha il concetto di e sotto. `memory.min` `memory.high` `requests.memory` Questo ci dà la possibilità di rilasciare in modo aggressivo questa memoria cache per utilizzarla in altri contenitori. Una volta che il contenitore raggiunge il limite massimo di memoria, il kernel può recuperare in modo aggressivo la memoria del contenitore fino al valore impostato su. `memory.min` In questo modo possiamo contare su una maggiore flessibilità quando un nodo è sottoposto a pressioni sulla memoria.

La domanda chiave diventa: a quale valore `memory.min` impostare? È qui che entrano in gioco le metriche di stallo della pressione della memoria. Possiamo usare queste metriche per rilevare il «danneggiamento» della memoria a livello di contenitore. Quindi possiamo usare controller come fbtax per rilevare i valori corretti cercando questo blocco della memoria e impostare dinamicamente il valore su questa impostazione. `memory.min` `memory.min`

Riepilogo

Per riassumere la sezione, è facile confondere i seguenti concetti:

- Utilizzo e saturazione
- Regole prestazionali di Linux con logica Kubernetes Scheduler

È necessario prestare molta attenzione a tenere separati questi concetti. Prestazioni e scalabilità sono legate a un livello profondo. Un ridimensionamento non necessario crea problemi di prestazioni, che a loro volta creano problemi di scalabilità.

Kubernetes Upstream SLOs

Amazon EKS esegue lo stesso codice delle versioni upstream di Kubernetes e garantisce che i cluster EKS operino entro i limiti SLOs definiti dalla community Kubernetes. Il Kubernetes Scalability Special Interest Group (SIG) definisce gli obiettivi di scalabilità e analizza i punti deboli nelle prestazioni attraverso e. SLIs SLOs

SLIs sono il modo in cui misuriamo un sistema, ad esempio metriche o misure che possono essere utilizzate per determinare il «buon» funzionamento del sistema, ad esempio la latenza o il conteggio delle richieste. SLOs definiamo i valori previsti per quando il sistema funziona «bene», ad esempio la latenza delle richieste rimane inferiore a 3 secondi. I Kubernetes si SLOs SLIs concentrano sulle prestazioni dei componenti Kubernetes e sono completamente indipendenti dal servizio Amazon EKS, SLAs che si concentra sulla disponibilità dell'endpoint del cluster EKS.

Kubernetes offre una serie di funzionalità che consentono agli utenti di estendere il sistema con componenti aggiuntivi o driver personalizzati, come driver CSI, webhook di ammissione e scaler automatici. Queste estensioni possono influire drasticamente sulle prestazioni di un cluster Kubernetes in diversi modi, ad esempio un webhook di ammissione che potrebbe aggiungere latenza alle richieste API K8s se la destinazione del webhook non è disponibile. `failurePolicy=Ignore` Il Kubernetes Scalability SIG definisce la scalabilità utilizzando un framework «you promise, we promise»:

> Se prometti di: - configurare correttamente il cluster - utilizzare le funzionalità di estensibilità «ragionevolmente» - mantieni il carico nel cluster entro i limiti consigliati
>
> quindi promettiamo che il tuo cluster sia scalabile, ad esempio: - tutti sono soddisfatti. SLOs

Kubernetes SLOs

Kubernetes SLOs non tiene conto di tutti i plugin e delle limitazioni esterne che potrebbero influire su un cluster, come la scalabilità dei nodi di lavoro o i webhook di ammissione. Questi SLOs si concentrano sui componenti di Kubernetes e garantiscono che le azioni e le risorse di Kubernetes funzionino secondo le aspettative. SLOs Aiutano gli sviluppatori Kubernetes a garantire che le modifiche al codice Kubernetes non riducano le prestazioni dell'intero sistema.

Il Kuberntes Scalability SIG definisce il seguente SLO/ ufficiale. SLIs Il team di Amazon EKS esegue regolarmente test di scalabilità sui cluster EKS per monitorare il degrado delle prestazioni man mano che vengono apportate modifiche e vengono rilasciate nuove versioni. SLOs/SLIs

Obiettivo	Definizione	SLO
Latenza della richiesta API (mutante)	Latenza di elaborazione delle chiamate API mutanti per singoli oggetti per ogni coppia (risorsa, verbo), misurata come 99° percentile negli ultimi 5 minuti	Nell'installazione predefinita di Kubernetes, per ogni coppia (risorsa, verbo), escluse le risorse virtuali e aggregate e le definizioni di risorse personalizzate, 99° percentile per giorno del cluster <= 1s
Latenza delle richieste API (sola lettura)	Latenza di elaborazione delle chiamate API di sola lettura non in streaming per ogni coppia (risorsa, ambito), misurata come 99° percentile negli ultimi 5 minuti	Nell'installazione predefinita di Kubernetes, per ogni coppia (risorsa, ambito), escluse le risorse virtuali e aggregate e le definizioni di risorse personalizzate, 99° percentile per giorno del cluster: (a) <= 1s se (b) <= 30s altrimenti (if o) `scope=resource scope=namespace scope=cluster`
Latenza di avvio del pod	Latenza di avvio dei pod stateless pianificabili, escluso il tempo necessario per estrarre le immagini ed eseguire contenitori init, misurata dal timestamp di creazione del pod al momento in cui tutti i contenitori vengono segnalati come avviati e osservati tramite watch, misurata come 99° percentile negli ultimi 5 minuti	Nell'installazione predefinita di Kubernetes, 99° percentile per giorno di cluster <= 5s

Latenza delle richieste API

L'`kube-apiserver`ha `--request-timeout` definita come `1m0s` predefinita, il che significa che una richiesta può essere eseguita per un massimo di un minuto (60 secondi) prima di essere scaduta e annullata. I SLOs valori definiti per Latency sono suddivisi in base al tipo di richiesta che viene effettuata, che può essere mutante o di sola lettura:

Mutante

Le richieste mutanti in Kubernetes apportano modifiche a una risorsa, ad esempio creazioni, eliminazioni o aggiornamenti. Queste richieste sono costose perché tali modifiche devono essere scritte nel backend etcd prima che venga restituito l'oggetto aggiornato. Etcd è un archivio chiave-valore distribuito che viene utilizzato per tutti i dati del cluster Kubernetes.

Questa latenza viene misurata come 99° percentile su 5 minuti per coppie (risorse, verbi) di risorse Kubernetes, ad esempio misurerebbe la latenza per le richieste Create Pod e le richieste Update Node. La latenza della richiesta deve essere <= 1 secondo per soddisfare lo SLO.

Sola lettura

Le richieste di sola lettura recuperano una singola risorsa (come Get Pod X) o una raccolta (come «Get all Pods from Namespace X»). `kube-apiserver`Mantiene una cache di oggetti, quindi le risorse richieste possono essere restituite dalla cache o potrebbe essere necessario recuperarle prima da etcd. Queste latenze vengono misurate anche dal 99° percentile nell'arco di 5 minuti, tuttavia le richieste di sola lettura possono avere ambiti separati. Lo SLO definisce due obiettivi diversi:

- Per le richieste effettuate per una singola risorsa (ad esempio`kubectl get pod -n mynamespace my-controller-xxx`), la latenza della richiesta deve rimanere <= 1 secondo.
- Per le richieste effettuate per più risorse in un namespace o in un cluster (ad esempio,`kubectl get pods -A`) la latenza deve rimanere <= 30 secondi

Lo SLO ha valori di destinazione diversi per diversi ambiti di richiesta perché le richieste effettuate per un elenco di risorse Kubernetes prevedono che i dettagli di tutti gli oggetti della richiesta vengano restituiti all'interno dello SLO. Su cluster di grandi dimensioni o grandi raccolte di risorse, ciò può comportare grandi dimensioni di risposta, la cui restituzione può richiedere del tempo. Ad esempio, in un cluster che esegue decine di migliaia di Pod con ogni Pod di circa 1 KB se codificato in JSON, la restituzione di tutti i Pod nel cluster consisterebbe in almeno 10 MB. I client Kubernetes possono

contribuire a ridurre questa dimensione di risposta utilizzando APIList Chunking per recuperare grandi raccolte di risorse.

Latenza di avvio del Pod

Questo SLO riguarda principalmente il tempo impiegato dalla creazione del Pod a quando i contenitori in quel Pod iniziano effettivamente l'esecuzione. Per misurarlo, viene calcolata la differenza tra il timestamp di creazione registrato sul Pod e il momento in cui un WATCH su quel Pod riporta che i contenitori sono stati avviati (escluso il tempo di estrazione dell'immagine del contenitore e l'inizio dell'esecuzione del contenitore). Per soddisfare lo SLO, il 99° percentile per giorno di cluster di questo Pod Startup Latency deve rimanere <=5 secondi.

Nota che questo SLO presuppone che i nodi di lavoro esistano già in questo cluster in uno stato pronto per la pianificazione del Pod. Questo SLO non tiene conto degli image pull o delle esecuzioni di init container e limita inoltre il test ai «pod stateless» che non sfruttano i plug-in di archiviazione persistenti.

Metriche SLI di Kubernetes

Kubernetes sta inoltre migliorando l'osservabilità in tutto il mondo SLIs aggiungendo le metriche Prometheus ai componenti Kubernetes che le tengono traccia nel tempo. SLIs Utilizzando Prometheus Query Language (PromQL) possiamo creare query che mostrano le prestazioni SLI nel tempo in strumenti come le dashboard Prometheus o Grafana, di seguito sono riportati alcuni esempi di quanto sopra. SLOs

Latenza delle richieste del server API

Metrica	Definizione
apiserver_request_sli_duration_seconds	Distribuzione della latenza di risposta (senza contare la durata dei webhook e i tempi di attesa nella coda di priorità e equità) in secondi per ogni verbo, gruppo, versione, risorsa, sottorisorsa, ambito e componente.
apiserver_request_duration_seconds	Distribuzione della latenza di risposta in secondi per ogni verbo, valore di dry run, gruppo, versione, risorsa, sottorisorsa, ambito e componente.

> **Note**
>
> La `apiserver_request_sli_duration_seconds` metrica è disponibile a partire da Kubernetes 1.27.

Puoi utilizzare queste metriche per analizzare i tempi di risposta del server API e verificare se ci sono colli di bottiglia nei componenti di Kubernetes o in altri plugin/componenti. Le domande seguenti si basano sulla dashboard SLO della community.

API Request latency SLI (variabile): questa volta non include l'esecuzione dei webhook o il tempo di attesa in coda. `histogram_quantile(0.99, sum(rate(apiserver_request_sli_duration_seconds_bucket{verb=~"CREATE|DELETE|PATCH|POST|PUT", subresource!~"proxy|attach|log|exec|portforward"}[5m])) by (resource, subresource, verb, scope, le)) > 0`

Latenza totale della richiesta API (variabile): questo è il tempo totale impiegato dalla richiesta sul server API, questo tempo può essere più lungo del tempo SLI perché include l'esecuzione dei webhook e i tempi di attesa per priorità e correttezza delle API. `histogram_quantile(0.99, sum(rate(apiserver_request_duration_seconds_bucket{verb=~"CREATE|DELETE|PATCH|POST|PUT", subresource!~"proxy|attach|log|exec|portforward"}[5m])) by (resource, subresource, verb, scope, le)) > 0`

In queste query escludiamo le richieste API di streaming che non vengono restituite immediatamente, come `kubectl port-forward` o `kubectl exec` requests (`subresource!~"proxy|attach|log|exec|portforward"`), e stiamo filtrando solo i verbi Kubernetes che modificano gli oggetti (). `verb=~"CREATE|DELETE|PATCH|POST|PUT"` Stiamo quindi calcolando il 99° percentile di tale latenza negli ultimi 5 minuti.

Possiamo usare una query simile per le richieste API di sola lettura, modifichiamo semplicemente i verbi che stiamo filtrando per includere le azioni di sola lettura e. `LIST` `GET` Esistono anche diverse soglie SLO a seconda dell'ambito della richiesta, ad esempio ottenere una singola risorsa o elencare più risorse.

API Request latency SLI (sola lettura): questo periodo non include l'esecuzione dei webhook o il tempo di attesa in coda. Per una singola risorsa (scope=resource, threshold = 1s) `histogram_quantile(0.99, sum(rate(apiserver_request_sli_duration_seconds_bucket{verb=~"GET", scope=~"resource"}[5m])) by (resource, subresource, verb, scope, le))`

Per una raccolta di risorse nello stesso spazio dei nomi (scope=namespace, threshold = 5s)
```
histogram_quantile(0.99,
sum(rate(apiserver_request_sli_duration_seconds_bucket{verb=~"LIST",
scope=~"namespace"}[5m])) by (resource, subresource, verb, scope, le))
```

Per una raccolta di risorse nell'intero cluster (scope=cluster, threshold = 30s)
```
histogram_quantile(0.99,
sum(rate(apiserver_request_sli_duration_seconds_bucket{verb=~"LIST",
scope=~"cluster"}[5m])) by (resource, subresource, verb, scope, le))
```

Latenza totale delle richieste API (sola lettura): questo è il tempo totale impiegato dalla richiesta sul server API, questo periodo può essere più lungo del tempo SLI perché include i tempi di esecuzione e attesa dei webhook. Per una singola risorsa (scope=resource, threshold = 1s) `histogram_quantile(0.99, sum(rate(apiserver_request_duration_seconds_bucket{verb=~"GET", scope=~"resource"}[5m])) by (resource, subresource, verb, scope, le))`

Per una raccolta di risorse nello stesso spazio dei nomi (scope=namespace, threshold = 5s)
```
histogram_quantile(0.99,
sum(rate(apiserver_request_duration_seconds_bucket{verb=~"LIST",
scope=~"namespace"}[5m])) by (resource, subresource, verb, scope, le))
```

Per una raccolta di risorse nell'intero cluster (scope=cluster, threshold = 30s)
```
histogram_quantile(0.99,
sum(rate(apiserver_request_duration_seconds_bucket{verb=~"LIST",
scope=~"cluster"}[5m])) by (resource, subresource, verb, scope, le))
```

Le metriche SLI forniscono informazioni dettagliate sulle prestazioni dei componenti di Kubernetes, escludendo il tempo impiegato dalle richieste in attesa nelle code API Priority e Fairness, utilizzando i webhook di ammissione o altre estensioni Kubernetes. Le metriche totali forniscono una visione più olistica in quanto riflettono il tempo che le applicazioni aspetterebbero per ricevere una risposta dal server API. Il confronto di queste metriche può fornire informazioni su dove vengono introdotti i ritardi nell'elaborazione delle richieste.

Latenza di avvio del pod

Metrica	Definizione
kubelet_pod_start_sli_duration_seconds	Durata in secondi per avviare un pod, escluso il tempo necessario per estrarre le immagini ed eseguire init containers, misurata dal timestamp di creazione del pod al momento in cui tutti i contenitori vengono segnalati come avviati e osservati tramite watch
kubelet_pod_start_duration_seconds	Durata in secondi dal momento in cui Kubelet vede un pod per la prima volta all'avvio del pod. Ciò non include il tempo necessario per pianificare il pod o aumentare la capacità del nodo di lavoro.

> **Note**
>
> `kubelet_pod_start_sli_duration_seconds`è disponibile a partire da Kubernetes 1.27.

Analogamente alle domande precedenti, puoi utilizzare queste metriche per ottenere informazioni dettagliate su quanto tempo il ridimensionamento dei nodi, l'image pull e gli init container ritardino il lancio del pod rispetto alle azioni di Kubelet.

Pod startup latency SLI: questo è il periodo che intercorre tra la creazione del pod e il momento in cui i contenitori dell'applicazione sono stati segnalati come in esecuzione. Ciò include il tempo necessario alla disponibilità della capacità del nodo di lavoro e alla pianificazione del pod, ma non include il tempo necessario per estrarre le immagini o per l'esecuzione dei contenitori init. `histogram_quantile(0.99, sum(rate(kubelet_pod_start_sli_duration_seconds_bucket[5m])) by (le))`

Latenza totale di avvio del pod: questo è il tempo impiegato dal kubelet per avviare il pod per la prima volta. Viene misurato a partire dal momento in cui il kubelet riceve il pod tramite WATCH, il che non include il tempo necessario per la scalabilità o la

pianificazione del nodo di lavoro. Ciò include il tempo necessario per estrarre le immagini e avviare i contenitori per l'esecuzione. `histogram_quantile(0.99, sum(rate(kubelet_pod_start_duration_seconds_bucket[5m])) by (le))`

SLOs sul tuo cluster

Se stai raccogliendo le metriche di Prometheus dalle risorse Kubernetes nel tuo cluster EKS, puoi ottenere informazioni più approfondite sulle prestazioni dei componenti del piano di controllo Kubernetes.

Il repository perf-tests include dashboard Grafana che mostrano le latenze e le metriche critiche delle prestazioni per il cluster durante i test. La configurazione perf-tests sfrutta il kube-prometheus-stack, un progetto open source configurato per raccogliere i parametri di Kubernetes, ma puoi anche usare Amazon Managed Prometheus e Amazon Managed Grafana.

Se utilizzi la soluzione Prometheus `kube-prometheus-stack` o una soluzione simile, puoi installare la stessa dashboard per osservarli sul tuo cluster in tempo SLOs reale.

1. Dovrai prima installare le Prometheus Rules utilizzate nelle dashboard con. `kubectl apply -f prometheus-rules.yaml` Puoi scaricare una copia delle regole qui: perf- -rules.yaml https://github.com/kubernetes/ tests/blob/master/clusterloader2/pkg/prometheus/manifests/prometheus
   a. Assicurati di controllare che lo spazio dei nomi nel file corrisponda al tuo ambiente
   b. Verifica che le etichette corrispondano al valore di `prometheus.prometheusSpec.ruleSelector` helm se stai utilizzando `kube-prometheus-stack`
2. È quindi possibile installare i dashboard in Grafana. I dashboard json e gli script python per generarli sono disponibili qui: perf- https://github.com/kubernetes/ tests/tree/master/clusterloader2/pkg/prometheus/manifests/dashboards
   a. la `slo.json` dashboard mostra le prestazioni del cluster in relazione a Kubernetes SLOs

Tieni presente che si SLOs concentrano sulle prestazioni dei componenti Kubernetes nei tuoi cluster, ma ci sono metriche aggiuntive che puoi esaminare che forniscono diverse prospettive o approfondimenti sul tuo cluster. I progetti della community Kubernetes come K ube-state-metrics possono aiutarti ad analizzare rapidamente le tendenze nel tuo cluster. I plugin e i driver più comuni della community Kubernetes emettono anche le metriche di Prometheus, che ti consentono di esaminare aspetti come scaler automatici o pianificatori personalizzati.

La Observability Best Practices Guide contiene esempi di altre metriche di Kubernetes che puoi utilizzare per ottenere ulteriori informazioni.

Limiti noti e Service Quotas

> **Tip**
>
> Esplora le best practice tramite i workshop Amazon EKS.

Amazon EKS può essere utilizzato per una varietà di carichi di lavoro e può interagire con un'ampia gamma di servizi AWS, e abbiamo visto che i carichi di lavoro dei clienti riscontrano una gamma simile di quote di servizio AWS e altri problemi che ostacolano la scalabilità.

Il tuo account AWS ha quote predefinite (un limite massimo al numero di ogni risorsa AWS che il tuo team può richiedere). Ogni servizio AWS definisce la propria quota e le quote sono generalmente specifiche della regione. Puoi richiedere aumenti per alcune quote (limiti flessibili) e altre quote non possono essere aumentate (limiti rigidi). È necessario considerare questi valori durante l'architettura delle applicazioni. Prendi in considerazione la possibilità di rivedere periodicamente questi limiti di servizio e di incorporarli durante la progettazione dell'applicazione.

Puoi esaminare l'utilizzo nel tuo account e aprire una richiesta di aumento delle quote nella console AWS Service Quotas o utilizzando l'AWS CLI. Consulta la documentazione AWS del rispettivo servizio AWS per maggiori dettagli sulle Service Quotas e su eventuali ulteriori restrizioni o avvisi relativi al loro aumento.

> **Note**
>
> Amazon EKS Service Quotas elenca le quote di servizio e contiene link per richiedere aumenti, laddove disponibili.

Altri AWS Service Quotas

Abbiamo riscontrato che i clienti EKS sono stati colpiti dalle quote elencate di seguito per altri servizi AWS. Alcune di queste possono applicarsi solo a casi d'uso o configurazioni specifici, tuttavia potresti valutare se la tua soluzione presenterà uno di questi problemi man mano che sarà scalabile. Le

Quote sono organizzate per servizio e ogni Quota ha un ID nel formato L-XXXXXXXX che puoi usare per cercarlo nella console AWS Service Quotas

Servizio	Quota (L-xxxxx)	Impatto	ID (L-xxxxx)	default
IAM	Ruoli per account	Può limitare il numero di cluster o ruoli IRSA in un account.	L-D64 FE177	1.000
IAM	OpenId connetti i provider per account	Può limitare il numero di cluster per account, OpenID Connect viene utilizzato da IRSA	L-858F3967	100
IAM	Lunghezza della policy di attendibilità del ruolo	Può limitare il numero di cluster a cui è associato un ruolo IAM per IRSA	L-C07B4B0D	2.048
VPC	Gruppi di sicurezza per interfaccia di rete	Può limitare il controllo o la connettività della rete per il cluster	L-2 AFB9258	5
VPC	IPv4 Blocchi CIDR per VPC	Può limitare il numero di EKS Worker Nodes	L-83 A9D CA0	5
VPC	Route per tabella di routing	Può limitare il controllo o la connettività della rete per il cluster	L-93826ACB	50

Servizio	Quota (L-xxxxx)	Impatto	ID (L-xxxxx)	default
VPC	Connessioni VPC in peering per VPC	Può limitare il controllo o la connettività della rete per il cluster	L-7E9ECCDB	50
VPC	Regole in entrata o in uscita per gruppo di sicurezza.	Possono limitare il controllo o la connettività della rete per il cluster, alcuni controlle r in EKS creano nuove regole	L-0 F EA8095	50
VPC	VPCs per regione	Può limitare il numero di cluster per account o il controllo o la connettività della rete per il cluster	L-F678F1CE	5
VPC	Gateway Internet per regione	Può limitare il numero di cluster per account o il controllo o la connettività della rete per il cluster	L-A4707A72	5
VPC	Interfacce di rete per regione	Può limitare il numero di nodi EKS Worker o le attività del piano di controllo Impact EKS. scaling/update	DF5L-E4 CA3	5.000

Servizio	Quota (L-xxxxx)	Impatto	ID (L-xxxxx)	default
VPC	Utilizzo degli indirizzi di rete	Può limitare il numero di cluster per account o il controllo o la connettività della rete per il cluster	L- BB24 F6E5	64.000
VPC	Utilizzo degli indirizzi di rete con peering	Può limitare il numero di cluster per account o il controllo o la connettività della rete per il cluster	LB CD17 FD4	128.000
ELB	Listener per Network Load Balancer	Può limitare il controllo dell'ingresso del traffico al cluster.	L-57A373D6	50
ELB	Gruppi di destinazione per regione	Può limitare il controllo dell'ingresso del traffico al cluster.	L-B22855CB	3.000
ELB	Destinazioni per Application Load Balancer	Può limitare il controllo dell'ingresso del traffico al cluster.	L-7E6692B2	1.000
ELB	Destinazioni per Network Load Balancer	Può limitare il controllo dell'ingresso del traffico al cluster.	L- EEF1 AD04	3.000

Servizio	Quota (L-xxxxx)	Impatto	ID (L-xxxxx)	default
ELB	Destinazioni per zona di disponibilità per Network Load Balancer	Può limitare il controllo dell'ingresso del traffico al cluster.	L-B211E961	500
ELB	Destinazioni per il gruppo di destinazione per regione	Può limitare il controllo dell'ingresso del traffico al cluster.	L-A0D0B863	1.000
ELB	Application Load Balancer per regione	Può limitare il controllo dell'ingresso del traffico al cluster.	L-53 DA6 B97	50
ELB	Classic Load Balancer per regione	Può limitare il controllo dell'ingresso del traffico al cluster.	L-E9E9831D	20
ELB	Network Load Balancer per regione	Può limitare il controllo dell'ingresso del traffico al cluster.	L-69A177A2	50
EC2	Esecuzione di istanze On-Demand Standard (A, C, D, H, I, M, R, T, Z) (come numero massimo di vCPU)	Può limitare il numero di EKS Worker Nodes	L-1216C47A	5

Servizio	Quota (L-xxxxx)	Impatto	ID (L-xxxxx)	default
EC2	Tutte le richieste di istanze Spot standard (A, C, D, H, I, M, R, T, Z) (come numero massimo di vCPU)	Può limitare il numero di EKS Worker Nodes	L-34B43A08	5
EC2	Elastico EC2-VPC IPs	Può limitare il numero di NAT GWs (e quindi VPCs), il che può limitare il numero di cluster in una regione	L-0263D0A3	5
EBS	Snapshot per regione	Può limitare la strategia di backup per carichi di lavoro con stato	L-309 BACF6	100.000
EBS	Archiviazione per volumi SSD a scopo generico (gp3) in TiB	Può limitare il numero di EKS Worker Nodes o Persisten tVolume di storage	L-7A658B76	50
EBS	Archiviazione per volumi SSD a scopo generico (gp2) in TiB	Può limitare il numero di EKS Worker Nodes o di storage Persisten tVolume	LD-18D FCD1	50

Servizio	Quota (L-xxxxx)	Impatto	ID (L-xxxxx)	default
ECR	Repository registrati	Può limitare il numero di carichi di lavoro nei cluster	L- E8D CFEB8	100.000
ECR	Immagine per repository	Può limitare il numero di carichi di lavoro nei cluster	L-03A36 CE1	20.000
SecretsManager	Segreti per regione	Può limitare il numero di carichi di lavoro nei cluster	L-2F66C23C	500.000

Limitazione delle richieste AWS

I servizi AWS implementano anche la limitazione delle richieste per garantire che rimangano performanti e disponibili per tutti i clienti. Analogamente a Service Quotas, ogni servizio AWS mantiene le proprie soglie di limitazione delle richieste. Prendi in considerazione la possibilità di rivedere la documentazione del rispettivo servizio AWS se i tuoi carichi di lavoro dovranno emettere rapidamente un gran numero di chiamate API o se noti errori di limitazione delle richieste nella tua applicazione.

Le richieste API EC2 relative al provisioning di interfacce di rete o indirizzi IP EC2 possono subire una limitazione delle richieste in cluster di grandi dimensioni o quando i cluster si ridimensionano drasticamente. La tabella seguente mostra alcune delle azioni API da cui abbiamo riscontrato la limitazione delle richieste da parte dei clienti. Puoi consultare i limiti di frequenza predefiniti di EC2 e i passaggi per richiedere un aumento del limite di velocità nella documentazione EC2 sul Rate Throttling.

Azioni mutanti	Azioni di sola lettura
AssignPrivateIpAddresses	DescribeDhcpOptions

Azioni mutanti	Azioni di sola lettura
AttachNetworkInterface	DescribeInstances
CreateNetworkInterface	DescribeNetworkInterfaces
DeleteNetworkInterface	DescribeSecurityGroups
DeleteTags	DescribeTags
DetachNetworkInterface	DescribeVpcs
ModifyNetworkInterfaceAttribute	DescribeVolumes
UnassignPrivateIpAddresses	

Altri limiti noti

- Route 53 ha anche un limite di velocità piuttosto basso di 5 richieste al secondo all'API Route 53. Se hai un gran numero di domini da aggiornare con un progetto come External DNS, potresti riscontrare una limitazione della velocità e ritardi nell'aggiornamento dei domini.
  - Alcuni tipi di istanze Nitro hanno un limite di 28 allegati di volume condiviso tra volumi Amazon EBS, interfacce di rete e volumi di NVMe instance store. Se i tuoi carichi di lavoro montano numerosi volumi EBS, potresti incontrare dei limiti alla densità dei pod che puoi raggiungere con questi tipi di istanze
  - Esiste un numero massimo di connessioni che possono essere tracciate per istanza Ec2. Se i tuoi carichi di lavoro gestiscono un gran numero di connessioni, potresti riscontrare errori o errori di comunicazione perché è stato raggiunto questo numero massimo. È possibile utilizzare le `conntrack_allowance_available` metriche relative alle prestazioni di `conntrack_allowance_exceeded` rete per monitorare il numero di connessioni tracciate sui nodi di lavoro EKS.
  - In ambiente EKS, il limite di archiviazione etcd è di 8 GiB secondo le indicazioni upstream. Monitora la metrica `apiserver_storage_size_bytes` per tracciare la dimensione del db di etcd. Puoi fare riferimento alle regole di avviso `etcdBackendQuotaLowSpace` e `etcdExcessiveDatabaseGrowth` configurare questo monitoraggio.

Procedure consigliate per gli aggiornamenti dei cluster

> **Tip**
>
> Esplora le best practice tramite i workshop Amazon EKS.

Questa guida mostra agli amministratori di cluster come pianificare ed eseguire la propria strategia di upgrade di Amazon EKS. Descrive inoltre come aggiornare i nodi autogestiti, i gruppi di nodi gestiti, i nodi Karpenter e i nodi Fargate. Non include indicazioni su EKS Anywhere, Kubernetes autogestito, AWS Outposts o AWS Local Zones.

Panoramica di

Una versione di Kubernetes comprende sia il piano di controllo che il piano dati. Per garantire un funzionamento regolare, sia il piano di controllo che il piano dati devono eseguire la stessa versione secondaria di Kubernetes, ad esempio 1.24. Sebbene AWS gestisca e aggiorni il piano di controllo, l'aggiornamento dei nodi di lavoro nel piano dati è responsabilità dell'utente.

- Piano di controllo: la versione del piano di controllo è determinata dal server API Kubernetes. Nei cluster Amazon EKS, AWS si occupa della gestione di questo componente. Gli aggiornamenti del piano di controllo possono essere avviati tramite l'API AWS.
- Piano dati: la versione del piano dati è associata alle versioni di Kubelet in esecuzione sui singoli nodi. È possibile avere nodi nello stesso cluster che eseguono versioni diverse. Puoi controllare le versioni di tutti i nodi eseguendo`kubectl get nodes`.

Prima dell'aggiornamento

Se hai intenzione di aggiornare la tua versione di Kubernetes in Amazon EKS, ci sono alcune politiche, strumenti e procedure importanti da mettere in atto prima di iniziare un aggiornamento.

- Comprendi le politiche di deprecazione: acquisisci una comprensione approfondita di come funziona la politica di deprecazione di Kubernetes. Tieni presente le eventuali modifiche imminenti che potrebbero influire sulle tue applicazioni esistenti. Le versioni più recenti di Kubernetes spesso eliminano gradualmente alcune APIs funzionalità, causando potenzialmente problemi all'esecuzione delle applicazioni.

- Esamina il registro delle modifiche di Kubernetes: esamina attentamente il registro delle modifiche di Kubernetes insieme alle versioni di Amazon EKS Kubernetes per comprendere qualsiasi possibile impatto sul cluster, ad esempio modifiche interrotte che potrebbero influire sui carichi di lavoro.
- Valuta la compatibilità dei componenti aggiuntivi del cluster: Amazon EKS non aggiorna automaticamente un componente aggiuntivo quando vengono rilasciate nuove versioni o dopo aver aggiornato il cluster a una nuova versione secondaria di Kubernetes. Consulta la sezione Aggiornamento di un componente aggiuntivo per comprendere la compatibilità di eventuali componenti aggiuntivi del cluster esistenti con la versione del cluster a cui intendi eseguire l'aggiornamento.
- Abilita la registrazione del piano di controllo: abilita la registrazione del piano di controllo per acquisire registri, errori o problemi che possono verificarsi durante il processo di aggiornamento. Valuta la possibilità di esaminare questi registri per eventuali anomalie. Testa gli aggiornamenti dei cluster in un ambiente non di produzione o integra i test automatici nel flusso di lavoro di integrazione continua per valutare la compatibilità delle versioni con le tue applicazioni, i controller e le integrazioni personalizzate.
- Esplora eksctl per la gestione dei cluster: prendi in considerazione l'utilizzo di eksctl per gestire il tuo cluster EKS. Ti offre la possibilità di aggiornare il piano di controllo, gestire i componenti aggiuntivi e gestire gli aggiornamenti dei nodi di lavoro. out-of-the-box
- Scegli la modalità automatica o i gruppi di nodi gestiti: semplifica e automatizza gli aggiornamenti dei nodi di lavoro utilizzando Auto Mode o i gruppi di nodi gestiti da EKS. Queste opzioni semplificano il processo e riducono l'intervento manuale.
- Utilizza kubectl Convert Plugin: sfrutta il plugin kubectl convert per facilitare la conversione dei file manifest di Kubernetes tra diverse versioni dell'API. Questo può aiutare a garantire che le configurazioni rimangano compatibili con la nuova versione di Kubernetes.

Mantieni il tuo cluster up-to-date

Rimanere aggiornati sugli aggiornamenti di Kubernetes è fondamentale per un ambiente EKS sicuro ed efficiente, che rifletta il modello di responsabilità condivisa di Amazon EKS. Integrando queste strategie nel tuo flusso di lavoro operativo, ti stai posizionando per mantenere up-to-date cluster sicuri che sfruttano appieno le funzionalità e i miglioramenti più recenti. Tattiche:

- Politica sulle versioni supportate: in linea con la community Kubernetes, Amazon EKS fornisce in genere tre versioni attive di Kubernetes. Una versione secondaria di Kubernetes è supportata come

standard in Amazon EKS per i primi 14 mesi dopo il suo rilascio. Una volta superata la data di fine del supporto standard, la versione passa al supporto esteso per i 12 mesi successivi. Gli avvisi di obsolescenza vengono emessi almeno 60 giorni prima della fine della data di supporto standard di una versione. Per ulteriori dettagli, consulta i documenti EKS Version Lifecycle.

- Politica di aggiornamento automatico: consigliamo vivamente di rimanere sincronizzati con gli aggiornamenti di Kubernetes nel cluster EKS. I cluster in esecuzione su una versione di Kubernetes che ha completato il ciclo di vita di 26 mesi (14 mesi di supporto standard più 12 mesi di supporto esteso) verranno aggiornati automaticamente alla versione successiva. Tieni presente che puoi disabilitare il supporto esteso. Il mancato aggiornamento proattivo prima di una versione end-of-life attiva un aggiornamento automatico, che potrebbe interrompere i carichi di lavoro e i sistemi. Per ulteriori informazioni, consulta la versione EKS. FAQs
- Create Upgrade Runbook: stabilite un processo ben documentato per la gestione degli aggiornamenti. Come parte del vostro approccio proattivo, sviluppate runbook e strumenti specializzati su misura per il vostro processo di aggiornamento. Ciò non solo migliora la preparazione, ma semplifica anche le transizioni complesse. Trasformate l'aggiornamento dei cluster almeno una volta all'anno come prassi standard. Questa pratica ti allinea ai continui progressi tecnologici, aumentando così l'efficienza e la sicurezza del tuo ambiente.

Consulta il calendario delle release di EKS

Consulta il calendario dei rilasci di EKS Kubernetes per sapere quando arriveranno nuove versioni e quando terminerà il supporto per versioni specifiche. In genere, EKS rilascia tre versioni minori di Kubernetes ogni anno e ciascuna versione secondaria è supportata per circa 14 mesi.

Inoltre, consulta le informazioni sulla versione originale di Kubernetes.

Scopri come si applica il modello di responsabilità condivisa agli aggiornamenti del cluster

Sei responsabile dell'avvio dell'aggiornamento sia per il piano di controllo del cluster che per il piano dati. Scopri come avviare un aggiornamento. Quando si avvia un aggiornamento del cluster, AWS gestisce l'aggiornamento del piano di controllo del cluster. L'utente è responsabile dell'aggiornamento del piano dati, inclusi i pod e gli addon Fargate. È necessario convalidare e pianificare gli aggiornamenti per i carichi di lavoro in esecuzione sul cluster per garantire che la loro disponibilità e le operazioni non siano influenzate dopo l'aggiornamento del cluster

Aggiorna i cluster sul posto

EKS supporta una strategia di aggiornamento dei cluster in loco. Ciò mantiene le risorse del cluster e mantiene coerente la configurazione del cluster (ad esempio, endpoint API, OIDC e sistemi di bilanciamento del carico ENIs). Ciò comporta meno interruzioni per gli utenti del cluster e utilizzerà i carichi di lavoro e le risorse esistenti nel cluster senza richiedere la ridistribuzione dei carichi di lavoro o la migrazione di risorse esterne (ad esempio DNS, storage).

Quando si esegue un aggiornamento del cluster sul posto, è importante notare che è possibile eseguire solo un aggiornamento di versione minore alla volta (ad esempio, da 1,24 a 1,25).

Ciò significa che se è necessario aggiornare più versioni, sarà necessaria una serie di aggiornamenti sequenziali. La pianificazione degli aggiornamenti sequenziali è più complicata e comporta un rischio maggiore di tempi di inattività. In questa situazione, vedi. the section called “Valuta Blue/Green i cluster come alternativa agli aggiornamenti dei cluster in loco”

Aggiorna il piano di controllo e il piano dati in sequenza

Per aggiornare un cluster è necessario eseguire le seguenti azioni:

1. Consulta le note di rilascio di Kubernetes ed EKS.
2. Effettua un backup del cluster. (opzionale)
3. Identifica e correggi l'utilizzo delle API obsolete e rimosse nei tuoi carichi di lavoro.
4. Assicurati che i gruppi di nodi gestiti, se utilizzati, siano sulla stessa versione di Kubernetes del piano di controllo. I gruppi di nodi gestiti da EKS e i nodi creati da EKS Fargate Profiles supportano 2 versioni secondarie asimmetriche tra il piano di controllo e il piano dati per Kubernetes versione 1.27 e precedenti. A partire dalla versione 1.28 e successive, i gruppi di nodi gestiti da EKS e i nodi creati da EKS Fargate Profiles supportano 3 versioni minori tra il piano di controllo e il piano dati. Ad esempio, se la versione del piano di controllo EKS è la 1.28, è possibile utilizzare in sicurezza versioni di kubelet precedenti alla 1.25. Se la tua versione EKS è 1.27, la versione di kubelet più vecchia che puoi usare è la 1.25.
5. Aggiorna il piano di controllo del cluster utilizzando la console o l'interfaccia a riga di comando AWS.
6. Verifica la compatibilità dei componenti aggiuntivi. Aggiorna i componenti aggiuntivi e i controller personalizzati di Kubernetes, se necessario.
7. Aggiorna kubectl.

8. Aggiorna il piano dati del cluster. Aggiorna i tuoi nodi alla stessa versione secondaria di Kubernetes del cluster aggiornato.

> **Tip**
>
> Se il cluster è stato creato utilizzando EKS Auto Mode, non è necessario aggiornare il piano dati del cluster. Dopo aver aggiornato il piano di controllo, EKS Auto Mode inizierà ad aggiornare in modo incrementale i nodi gestiti rispettando tutti i budget previsti per l'interruzione dei pod. Assicuratevi di monitorare questi aggiornamenti per verificare la conformità ai requisiti operativi.

Utilizza la documentazione EKS per creare una lista di controllo per gli aggiornamenti

La documentazione della versione EKS Kubernetes include un elenco dettagliato delle modifiche per ciascuna versione. Crea una lista di controllo per ogni aggiornamento.

Per indicazioni specifiche sull'aggiornamento della versione EKS, consulta la documentazione per le modifiche e le considerazioni più importanti per ciascuna versione.

- Consulta le note di rilascio per le versioni di Kubernetes sul supporto standard
- Consulta le note di rilascio per le versioni di Kubernetes sul supporto esteso

Aggiorna componenti aggiuntivi e componenti utilizzando l'API Kubernetes

Prima di aggiornare un cluster, dovresti capire quali versioni dei componenti Kubernetes stai utilizzando. Inventaria i componenti del cluster e identifica i componenti che utilizzano direttamente l'API Kubernetes. Ciò include componenti critici del cluster come agenti di monitoraggio e registrazione, scaler automatici del cluster, driver di storage per container (ad esempio EBS CSI, EFS CSI), controller di ingresso e qualsiasi altro carico di lavoro o componente aggiuntivo che si basa direttamente sull'API Kubernetes.

> **Tip**
>
> I componenti critici del cluster vengono spesso installati in un namespace `*-system`

```
kubectl get ns | grep '-system'
```

Dopo aver identificato i componenti che si basano sull'API Kubernetes, consulta la relativa documentazione per verificare la compatibilità delle versioni e i requisiti di aggiornamento. Ad esempio, consulta la documentazione di AWS Load Balancer Controller per la compatibilità delle versioni. Potrebbe essere necessario aggiornare alcuni componenti o modificare la configurazione prima di procedere con un aggiornamento del cluster. Alcuni componenti critici da controllare includono CoredNS, kube-proxy, VPC CNI e driver di archiviazione.

I cluster spesso contengono molti carichi di lavoro che utilizzano l'API Kubernetes e sono necessari per funzionalità di carico di lavoro come controller di ingresso, sistemi di distribuzione continua e strumenti di monitoraggio. Quando si aggiorna un cluster EKS, è necessario aggiornare anche i componenti aggiuntivi e gli strumenti di terze parti per assicurarsi che siano compatibili.

Consultate i seguenti esempi di componenti aggiuntivi comuni e la relativa documentazione di aggiornamento:

- Amazon VPC CNI: per la versione consigliata del componente aggiuntivo Amazon VPC CNI per ogni versione del cluster, consulta Aggiornamento del plug-in Amazon VPC CNI per il componente aggiuntivo autogestito Kubernetes. Se installato come componente aggiuntivo Amazon EKS, può essere aggiornato solo una versione minore alla volta.
- kube-proxy: vedi Aggiornamento del componente aggiuntivo autogestito Kubernetes kube-proxy.
- CoredNS: vedere Aggiornamento del componente aggiuntivo CoredNS autogestito.
- AWS Load Balancer Controller: il controller AWS Load Balancer deve essere compatibile con la versione EKS che hai distribuito. Per ulteriori informazioni, consulta la guida all'installazione.
- Driver Amazon Elastic Block Store (Amazon EBS) Container Storage Interface (CSI): per informazioni sull'installazione e l'aggiornamento, consulta Gestione del driver CSI di Amazon EBS come componente aggiuntivo Amazon EKS.
- Driver Amazon Elastic File System (Amazon EFS) Container Storage Interface (CSI): per informazioni sull'installazione e l'aggiornamento, consulta il driver CSI di Amazon EFS.
- Kubernetes Metrics Server: per ulteriori informazioni, consulta metrics-server on. GitHub

- Kubernetes Cluster Autoscaler: per aggiornare la versione di Kubernetes Cluster Autoscaler, modifica la versione dell'immagine nella distribuzione. Cluster Autoscaler è strettamente associato allo scheduler Kubernetes. Dovrai sempre aggiornarlo quando aggiorni il cluster. Consulta le GitHub versioni per trovare l'indirizzo dell'ultima versione corrispondente alla tua versione secondaria di Kubernetes.
- Karpenter: per informazioni sull'installazione e l'aggiornamento, consulta la documentazione di Karpenter.

> **Tip**
>
> Non è necessario aggiornare manualmente nessuna delle funzionalità di Amazon EKS Auto Mode, incluse le funzionalità di scalabilità automatica di calcolo, storage a blocchi e bilanciamento del carico.

Verifica i requisiti EKS di base prima dell'aggiornamento

AWS richiede determinate risorse nel tuo account per completare il processo di upgrade. Se queste risorse non sono presenti, il cluster non può essere aggiornato. Un aggiornamento del piano di controllo richiede le seguenti risorse:

1. Indirizzi IP disponibili: Amazon EKS richiede fino a cinque indirizzi IP disponibili dalle sottoreti specificate al momento della creazione del cluster per aggiornare il cluster. In caso contrario, aggiorna la configurazione del cluster per includere nuove sottoreti del cluster prima di eseguire l'aggiornamento della versione.
2. Ruolo IAM EKS: il ruolo IAM del piano di controllo è ancora presente nell'account con le autorizzazioni necessarie.
3. Se il tuo cluster ha la crittografia segreta abilitata, assicurati che il ruolo IAM del cluster sia autorizzato a utilizzare la chiave AWS Key Management Service (AWS KMS).

Verifica gli indirizzi IP disponibili

Per aggiornare il cluster, Amazon EKS richiede fino a cinque indirizzi IP disponibili dalle sottoreti specificate al momento della creazione del cluster.

Per verificare che le sottoreti dispongano di indirizzi IP sufficienti per aggiornare il cluster, puoi eseguire il seguente comando:

```
CLUSTER=<cluster name>
aws ec2 describe-subnets --subnet-ids \
  $(aws eks describe-cluster --name ${CLUSTER} \
  --query 'cluster.resourcesVpcConfig.subnetIds' \
  --output text) \
  --query 'Subnets[*].[SubnetId,AvailabilityZone,AvailableIpAddressCount]' \
  --output table

-------------------------------------------------
|                DescribeSubnets                |
+--------------------------+--------------+-------+
|  subnet-067fa8ee8476abbd6 |  us-east-1a  |  8184 |
|  subnet-0056f7403b17d2b43 |  us-east-1b  |  8153 |
|  subnet-09586f8fb3addbc8c |  us-east-1a  |  8120 |
|  subnet-047f3d276a22c6bce |  us-east-1b  |  8184 |
+--------------------------+--------------+-------+
```

Il VPC CNI Metrics Helper può essere utilizzato per creare una dashboard CloudWatch per le metriche VPC. Amazon EKS consiglia di aggiornare le sottoreti del cluster utilizzando l'API "UpdateClusterConfiguration" prima di iniziare un aggiornamento della versione Kubernetes se stai esaurendo gli indirizzi IP nelle sottoreti inizialmente specificate durante la creazione del cluster. Verifica che le nuove sottoreti che ti verranno fornite:

- appartengono allo stesso insieme di AZs quelli selezionati durante la creazione del cluster.
- appartengono allo stesso VPC fornito durante la creazione del cluster

Prendi in considerazione l'associazione di blocchi CIDR aggiuntivi se gli indirizzi IP nel blocco CIDR VPC esistente si esauriscono. AWS consente l'associazione di blocchi CIDR aggiuntivi con il VPC del cluster esistente, ampliando efficacemente il pool di indirizzi IP. Questa espansione può essere realizzata introducendo intervalli IP privati aggiuntivi (RFC 1918) o, se necessario, intervalli IP pubblici (non RFC 1918). È necessario aggiungere nuovi blocchi VPC CIDR e consentire il completamento dell'aggiornamento del VPC prima che Amazon EKS possa utilizzare il nuovo CIDR. Successivamente, puoi aggiornare le sottoreti in base ai blocchi CIDR appena configurati sul VPC.

Verifica il ruolo EKS IAM

Per verificare che il ruolo IAM sia disponibile e abbia la corretta politica di assunzione del ruolo nel tuo account, puoi eseguire i seguenti comandi:

```
CLUSTER=<cluster name>
ROLE_ARN=$(aws eks describe-cluster --name ${CLUSTER} \
  --query 'cluster.roleArn' --output text)
aws iam get-role --role-name ${ROLE_ARN##*/} \
  --query 'Role.AssumeRolePolicyDocument'

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "eks.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Esegui la migrazione ai componenti aggiuntivi EKS

Amazon EKS installa automaticamente componenti aggiuntivi come il plug-in Amazon VPC CNI per Kubernetes e CoredNS per ogni cluster. `kube-proxy` I componenti aggiuntivi possono essere gestiti automaticamente o installati come componenti aggiuntivi Amazon EKS. Amazon EKS Add-ons è un modo alternativo per gestire i componenti aggiuntivi utilizzando l'API EKS.

Puoi utilizzare i componenti aggiuntivi di Amazon EKS per aggiornare le versioni con un solo comando. Ad esempio:

```
aws eks update-addon –cluster-name my-cluster –addon-name vpc-cni –addon-version
 version-number \
--service-account-role-arn arn:aws:iam::111122223333:role/role-name –configuration-
values '{}' –resolve-conflicts PRESERVE
```

Verifica se disponi di componenti aggiuntivi EKS con:

```
aws eks list-addons --cluster-name <cluster name>
```

> **Warning**
>
> I componenti aggiuntivi EKS non vengono aggiornati automaticamente durante un aggiornamento del piano di controllo. È necessario avviare gli aggiornamenti dei componenti aggiuntivi EKS e selezionare la versione desiderata.

- L'utente è responsabile della selezione di una versione compatibile tra tutte le versioni disponibili. Consulta le linee guida sulla compatibilità delle versioni aggiuntive.
- I componenti aggiuntivi di Amazon EKS possono essere aggiornati solo una versione secondaria alla volta.

Scopri di più su quali componenti sono disponibili come componenti aggiuntivi EKS e su come iniziare.

Scopri come fornire una configurazione personalizzata a un componente aggiuntivo EKS.

Identifica e correggi l'utilizzo delle API rimosse prima di aggiornare il piano di controllo

È necessario identificare l'utilizzo dell'API rimosso APIs prima di aggiornare il piano di controllo EKS. A tale scopo, consigliamo di utilizzare strumenti in grado di controllare un cluster in esecuzione o file manifest di Kubernetes statici e renderizzati.

L'esecuzione del controllo rispetto ai file manifest statici è generalmente più accurata. Se eseguiti su cluster attivi, questi strumenti possono restituire falsi positivi.

Un'API Kubernetes obsoleta non significa che l'API sia stata rimossa. Dovresti controllare la politica di deprecazione di Kubernetes per capire in che modo la rimozione delle API influisce sui tuoi carichi di lavoro.

Cluster Insights

Cluster Insights è una funzionalità che fornisce risultati sui problemi che possono influire sulla capacità di aggiornare un cluster EKS alle versioni più recenti di Kubernetes. Questi risultati sono

curati e gestiti da Amazon EKS e offrono consigli su come porvi rimedio. Sfruttando Cluster Insights, puoi ridurre al minimo lo sforzo speso per l'aggiornamento alle versioni più recenti di Kubernetes.

Per visualizzare informazioni dettagliate su un cluster EKS, puoi eseguire il comando:

```
aws eks list-insights --region <region-code> --cluster-name <my-cluster>

{
    "insights": [
        {
            "category": "UPGRADE_READINESS",
            "name": "Deprecated APIs removed in Kubernetes v1.29",
            "insightStatus": {
                "status": "PASSING",
                "reason": "No deprecated API usage detected within the last 30 days."
            },
            "kubernetesVersion": "1.29",
            "lastTransitionTime": 1698774710.0,
            "lastRefreshTime": 1700157422.0,
            "id": "123e4567-e89b-42d3-a456-579642341238",
            "description": "Checks for usage of deprecated APIs that are scheduled for
 removal in Kubernetes v1.29. Upgrading your cluster before migrating to the updated
 APIs supported by v1.29 could cause application impact."
        }
    ]
}
```

Per un output più descrittivo sulle informazioni ricevute, puoi eseguire il comando:

```
aws eks describe-insight --region <region-code> --id <insight-id> --cluster-name <my-
cluster>
```

Hai anche la possibilità di visualizzare gli approfondimenti nella console Amazon EKS. Dopo aver selezionato il cluster dall'elenco dei cluster, i risultati degli approfondimenti si trovano nella `Upgrade Insights` scheda.

Se trovi una panoramica del cluster con`"status": ERROR`, devi risolvere il problema prima di eseguire l'aggiornamento del cluster. Esegui il `aws eks describe-insight` comando che condividerà i seguenti consigli di riparazione:

Risorse interessate:

```
"resources": [
      {
        "insightStatus": {
          "status": "ERROR"
        },
        "kubernetesResourceUri": "/apis/policy/v1beta1/podsecuritypolicies/null"
      }
]
```

APIs obsoleto:

```
"deprecationDetails": [
      {
        "usage": "/apis/flowcontrol.apiserver.k8s.io/v1beta2/flowschemas",
        "replacedWith": "/apis/flowcontrol.apiserver.k8s.io/v1beta3/flowschemas",
        "stopServingVersion": "1.29",
        "clientStats": [],
        "startServingReplacementVersion": "1.26"
      }
]
```

Azione consigliata da intraprendere:

```
"recommendation": "Update manifests and API clients to use newer Kubernetes APIs if
 applicable before upgrading to Kubernetes v1.26."
```

L'utilizzo di cluster insights tramite la console EKS o la CLI aiuta a velocizzare il processo di aggiornamento corretto delle versioni del cluster EKS. Scopri di più con le seguenti risorse: * Documenti ufficiali EKS * Blog di lancio di Cluster Insights.

K ube-no-trouble

K ube-no-trouble è un'utilità da riga di comando open source con il comando`kubent`. Quando viene eseguito `kubent` senza argomenti, utilizzerà il KubeConfig contesto corrente, scansionerà il cluster e stamperà un rapporto con ciò che APIs verrà obsoleto e rimosso.

```
kubent

4:17PM INF >>> Kube No Trouble `kubent` <<<
4:17PM INF version 0.7.0 (git sha d1bb4e5fd6550b533b2013671aa8419d923ee042)
4:17PM INF Initializing collectors and retrieving data
```

```
4:17PM INF Target K8s version is 1.24.8-eks-ffeb93d
4:l INF Retrieved 93 resources from collector name=Cluster
4:17PM INF Retrieved 16 resources from collector name="Helm v3"
4:17PM INF Loaded ruleset name=custom.rego.tmpl
4:17PM INF Loaded ruleset name=deprecated-1-16.rego
4:17PM INF Loaded ruleset name=deprecated-1-22.rego
4:17PM INF Loaded ruleset name=deprecated-1-25.rego
4:17PM INF Loaded ruleset name=deprecated-1-26.rego
4:17PM INF Loaded ruleset name=deprecated-future.rego
_____________________________________________________________________________________
>>> Deprecated APIs removed in 1.25 <<<
-------------------------------------------------------------------------------------
KIND                NAMESPACE     NAME             API_VERSION      REPLACE_WITH
 (SINCE)
PodSecurityPolicy   <undefined>   eks.privileged   policy/v1beta1   <removed> (1.21.0)
```

Può anche essere usato per scansionare file manifest statici e pacchetti helm. Si consiglia di eseguirlo `kubent` come parte di un processo di integrazione continua (CI) per identificare i problemi prima della distribuzione dei manifesti. La scansione dei manifesti è inoltre più accurata rispetto alla scansione dei cluster live.

Kube-no-trouble fornisce un esempio di account e ruolo di servizio con le autorizzazioni appropriate per la scansione del cluster.

Plutone

Un'altra opzione è pluto, che è simile `kubent` perché supporta la scansione di un cluster live, di file manifest, di grafici helm e ha un' GitHub azione che puoi includere nel tuo processo CI.

```
pluto detect-all-in-cluster

NAME             KIND                VERSION          REPLACEMENT   REMOVED
 DEPRECATED   REPL AVAIL
eks.privileged   PodSecurityPolicy   policy/v1beta1                 false     true
    true
```

Resources

Per verificare che il cluster non utilizzi la versione deprecata APIs prima dell'aggiornamento, è necessario monitorare:

- metrica a partire `apiserver_requested_deprecated_apis` da Kubernetes v1.19:

```
kubectl get --raw /metrics | grep apiserver_requested_deprecated_apis

apiserver_requested_deprecated_apis{group="policy",removed_release="1.25",resource="podsecurity
 1
```

- eventi nei log di controllo impostati su: `k8s.io/deprecated true`

```
CLUSTER="<cluster_name>"
QUERY_ID=$(aws logs start-query \
 --log-group-name /aws/eks/${CLUSTER}/cluster \
 --start-time $(date -u --date="-30 minutes" "+%s") # or date -v-30M "+%s" on MacOS \
 --end-time $(date "+%s") \
 --query-string 'fields @message | filter `annotations.k8s.io/deprecated`="true"' \
 --query queryId --output text)

echo "Query started (query id: $QUERY_ID), please hold ..." && sleep 5 # give it some
 time to query

aws logs get-query-results --query-id $QUERY_ID
```

Che produrrà le righe se sono in uso quelle APIs obsolete:

```
{
    "results": [
        [
            {
                "field": "@message",
                "value": "{\"kind\":\"Event\",\"apiVersion\":\"audit.k8s.io/v1\",
\"level\":\"Request\",\"auditID\":\"8f7883c6-b3d5-42d7-967a-1121c6f22f01\",\"stage
\":\"ResponseComplete\",\"requestURI\":\"/apis/policy/v1beta1/podsecuritypolicies?
allowWatchBookmarks=true\\u0026resourceVersion=4131\\u0026timeout=9m19s\
\u0026timeoutSeconds=559\\u0026watch=true\",\"verb\":\"watch\",\"user\":{\"username
\":\"system:apiserver\",\"uid\":\"8aabfade-da52-47da-83b4-46b16cab30fa\",
\"groups\":[\"system:masters\"]},\"sourceIPs\":[\"::1\"],\"userAgent\":\"kube-
apiserver/v1.24.16 (linux/amd64) kubernetes/af930c1\",\"objectRef\":{\"resource
\":\"podsecuritypolicies\",\"apiGroup\":\"policy\",\"apiVersion\":\"v1beta1\"},
\"responseStatus\":{\"metadata\":{},\"code\":200},\"requestReceivedTimestamp\":
\"2023-10-04T12:36:11.849075Z\",\"stageTimestamp\":\"2023-10-04T12:45:30.850483Z\",
\"annotations\":{\"authorization.k8s.io/decision\":\"allow\",\"authorization.k8s.io/
reason\":\"\",\"k8s.io/deprecated\":\"true\",\"k8s.io/removed-release\":\"1.25\"}}"
            },
```

```
[...]
```

Aggiorna i carichi di lavoro Kubernetes. Usa kubectl-convert per aggiornare i manifesti

Dopo aver identificato quali carichi di lavoro e manifesti devono essere aggiornati, potrebbe essere necessario cambiare il tipo di risorsa nei file manifest (ad esempio to). PodSecurityPolicies PodSecurityStandards Ciò richiederà l'aggiornamento delle specifiche delle risorse e ulteriori ricerche a seconda della risorsa che verrà sostituita.

Se il tipo di risorsa rimane lo stesso ma la versione dell'API deve essere aggiornata, puoi usare il `kubectl-convert` comando per convertire automaticamente i tuoi file manifest. Ad esempio, per convertire una versione precedente di Deployment in`apps/v1`. Per ulteriori informazioni, consulta Installare il plugin kubectl convert sul sito web di Kubernetes.

```
kubectl-convert -f <file> --output-version <group>/<version>
```

Configura PodDisruptionBudgets e garantisci topologySpreadConstraints la disponibilità dei tuoi carichi di lavoro durante l'aggiornamento del piano dati

Assicurati che i carichi di lavoro siano adeguati PodDisruptionBudgetse garantisci topologySpreadConstraintsla disponibilità dei carichi di lavoro durante l'aggiornamento del piano dati. Non tutti i carichi di lavoro richiedono lo stesso livello di disponibilità, quindi è necessario convalidare la scalabilità e i requisiti del carico di lavoro.

Assicurati che i carichi di lavoro siano distribuiti in più zone di disponibilità e su più host. Gli spread topologici offriranno un maggiore livello di fiducia nella migrazione automatica dei carichi di lavoro al nuovo piano dati senza incidenti.

Ecco un esempio di carico di lavoro in cui l'80% delle repliche sarà sempre disponibile e distribuirà le repliche tra zone e host

```
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: myapp
```

```
spec:
  minAvailable: "80%"
  selector:
    matchLabels:
      app: myapp
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
spec:
  replicas: 10
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
      - image: public.ecr.aws/eks-distro/kubernetes/pause:3.2
        name: myapp
        resources:
          requests:
            cpu: "1"
            memory: 256M
      topologySpreadConstraints:
      - labelSelector:
          matchLabels:
            app: host-zone-spread
        maxSkew: 2
        topologyKey: kubernetes.io/hostname
        whenUnsatisfiable: DoNotSchedule
      - labelSelector:
          matchLabels:
            app: host-zone-spread
        maxSkew: 2
        topologyKey: topology.kubernetes.io/zone
        whenUnsatisfiable: DoNotSchedule
```

AWS Resilience Hub ha aggiunto Amazon Elastic Kubernetes Service (Amazon EKS) come risorsa supportata. Resilience Hub offre un unico posto per definire, convalidare e tracciare la resilienza

delle applicazioni in modo da evitare inutili tempi di inattività causati da interruzioni del software, dell'infrastruttura o delle operazioni.

Utilizza Managed Node Groups o Karpenter per semplificare gli aggiornamenti del piano dati

Managed Node Groups e Karpenter semplificano entrambi gli aggiornamenti dei nodi, ma adottano approcci diversi.

I gruppi di nodi gestiti automatizzano il provisioning e la gestione del ciclo di vita dei nodi. Ciò significa che è possibile creare, aggiornare automaticamente o terminare i nodi con una singola operazione.

Nella configurazione predefinita, Karpenter crea automaticamente nuovi nodi utilizzando l'ultima AMI ottimizzata EKS compatibile. Man mano che EKS rilascia EKS Optimized aggiornato AMIs o il cluster viene aggiornato, Karpenter inizierà automaticamente a utilizzare queste immagini. Karpenter implementa anche Node Expiry per aggiornare i nodi.

Karpenter può essere configurato per l'utilizzo personalizzato. AMIs Se usi custom AMIs con Karpenter, sei responsabile della versione di kubelet.

Conferma la compatibilità della versione con i nodi esistenti e il piano di controllo

Prima di procedere con un aggiornamento di Kubernetes in Amazon EKS, è fondamentale garantire la compatibilità tra i gruppi di nodi gestiti, i nodi autogestiti e il piano di controllo. La compatibilità è determinata dalla versione di Kubernetes in uso e varia in base a diversi scenari. Tattiche:

- Kubernetes v1.28+ — * * A partire dalla versione 1.28 di Kubernetes e successive, esiste una politica di versione più favorevole per i componenti principali. In particolare, l'inclinazione supportata tra il server API Kubernetes e Kubelet è stata estesa con una versione minore, che va da n-2 a n-3. Ad esempio, se la versione del piano di controllo EKS è la 1.28, puoi tranquillamente utilizzare versioni di kubelet precedenti alla 1.25. Questa versione skew è supportata in AWS Fargate, gruppi di nodi gestiti e nodi autogestiti. Consigliamo vivamente di conservare le versioni di Amazon Machine Image (AMI) up-to-date per motivi di sicurezza. Le versioni precedenti di kubelet potrebbero comportare rischi per la sicurezza a causa di potenziali vulnerabilità ed esposizioni comuni (CVEs), che potrebbero superare i vantaggi dell'utilizzo di versioni precedenti di kubelet.

- Kubernetes < v1.28 — Se utilizzi una versione precedente alla v1.28, l'inclinazione supportata tra il server API e il kubelet è n-2. Ad esempio, se la tua versione EKS è 1.27, la versione di kubelet più vecchia che puoi utilizzare è la 1.25. Questa versione asimmetrica è applicabile a AWS Fargate, gruppi di nodi gestiti e nodi autogestiti.

Abilita la scadenza dei nodi per i nodi gestiti da Karpenter

Un modo in cui Karpenter implementa gli aggiornamenti dei nodi consiste nell'utilizzare il concetto di scadenza dei nodi. Ciò riduce la pianificazione richiesta per gli aggiornamenti dei nodi. Quando si imposta un valore per ttlSecondsUntilExpired nel provider, viene attivata la scadenza del nodo. Dopo che i nodi hanno raggiunto l'età definita in pochi secondi, vengono svuotati ed eliminati in modo sicuro. Questo vale anche se sono in uso e consente di sostituire i nodi con istanze aggiornate di nuova generazione. Quando un nodo viene sostituito, Karpenter utilizza l'ultima versione ottimizzata per EKS. AMIs Per ulteriori informazioni, consulta Deprovisioning sul sito Web di Karpenter.

Karpenter non aggiunge automaticamente il jitter a questo valore. Per evitare un'interruzione eccessiva del carico di lavoro, definisci un budget per le interruzioni del pod, come mostrato nella documentazione di Kubernetes.

Se configuri ttlSecondsUntilExpired su un provisioner, ciò si applica ai nodi esistenti associati al provisioner.

Usa la funzione Drift per i nodi gestiti da Karpenter

La funzione Drift di Karpenter può aggiornare automaticamente i nodi forniti da Karpenter per rimanere sincronizzati con il piano di controllo EKS. Karpenter Drift attualmente deve essere abilitato utilizzando un feature gate. La configurazione predefinita di Karpenter utilizza l'ultima AMI ottimizzata per EKS per la stessa versione principale e secondaria del piano di controllo del cluster EKS.

Una volta completato un aggiornamento del cluster EKS, la funzione Drift di Karpenter rileverà che i nodi forniti da Karpenter utilizzano l'ottimizzazione EKS AMIs per la versione precedente del cluster e isolerà, drenerà e sostituirà automaticamente tali nodi. Per supportare il trasferimento dei pod verso nuovi nodi, segui le best practice di Kubernetes impostando le quote di risorse appropriate per i pod e utilizzando i pod disruption budget (PDB). Il deprovisioning di Karpenter preavvierà i nodi sostitutivi in base alle richieste di risorse del pod e rispetterà i nodi al momento del deprovisioning. PDBs

Usa eksctl per automatizzare gli aggiornamenti per i gruppi di nodi autogestiti

I gruppi di nodi autogestiti sono istanze EC2 che sono state distribuite nel tuo account e collegate al cluster all'esterno del servizio EKS. Di solito vengono implementati e gestiti da una qualche forma di strumenti di automazione. Per aggiornare i gruppi di nodi autogestiti, è necessario fare riferimento alla documentazione degli strumenti.

Ad esempio, eksctl supporta l'eliminazione e il drenaggio dei nodi autogestiti.

Alcuni strumenti comuni includono:

- eksctl
- KOPs
- Progetti EKS

Backup del cluster prima dell'aggiornamento

Le nuove versioni di Kubernetes introducono modifiche significative al tuo cluster Amazon EKS. Dopo aver aggiornato un cluster, non puoi effettuarne il downgrade.

Velero è uno strumento open source supportato dalla community che può essere utilizzato per eseguire backup di cluster esistenti e applicarli a un nuovo cluster.

Tieni presente che puoi creare nuovi cluster solo per le versioni di Kubernetes attualmente supportate da EKS. Se la versione attualmente in esecuzione sul cluster è ancora supportata e un aggiornamento non riesce, puoi creare un nuovo cluster con la versione originale e ripristinare il piano dati. Tieni presente che le risorse AWS, incluso IAM, non sono incluse nel backup di Velero. Queste risorse dovrebbero essere ricreate.

Riavvia gli schieramenti di Fargate dopo aver aggiornato il piano di controllo

Per aggiornare i nodi del piano dati Fargate è necessario ridistribuire i carichi di lavoro. È possibile identificare quali carichi di lavoro sono in esecuzione sui nodi fargate elencando tutti i pod con l'opzione. `-o wide` Qualsiasi nome di nodo che inizia con `fargate-` dovrà essere ridistribuito nel cluster.

Valuta Blue/Green i cluster come alternativa agli aggiornamenti dei cluster in loco

Alcuni clienti preferiscono adottare una blue/green strategia di aggiornamento. Ciò può comportare vantaggi, ma include anche aspetti negativi che devono essere considerati.

I vantaggi includono:

- Possibilità di modificare più versioni EKS contemporaneamente (ad esempio da 1.23 a 1.25)
- In grado di tornare al vecchio cluster
- Crea un nuovo cluster che può essere gestito con sistemi più recenti (ad esempio terraform)
- I carichi di lavoro possono essere migrati individualmente

Alcuni aspetti negativi includono:

- Modifica dell'endpoint dell'API e dell'OIDC che richiede l'aggiornamento dei consumatori (ad esempio kubectl e CI/CD)
- Richiede l'esecuzione di 2 cluster in parallelo durante la migrazione, il che può essere costoso e limitare la capacità della regione
- È necessario un maggiore coordinamento se i carichi di lavoro dipendono l'uno dall'altro per essere migrati insieme
- I sistemi di bilanciamento del carico e il DNS esterno non possono estendersi facilmente su più cluster

Sebbene questa strategia sia possibile, è più costosa di un aggiornamento sul posto e richiede più tempo per il coordinamento e le migrazioni dei carichi di lavoro. In alcune situazioni può essere necessaria e deve essere pianificata con attenzione.

Con alti gradi di automazione e sistemi dichiarativi come GitOps, questo potrebbe essere più facile da fare. Dovrai adottare ulteriori precauzioni per i carichi di lavoro con stato, in modo che i dati vengano sottoposti a backup e migrati in nuovi cluster.

Per ulteriori informazioni, consulta i post di questi blog:

- Aggiornamento del cluster Kubernetes: la strategia di implementazione blu-verde
- Migrazione di cluster Amazon EKS Blue/Green o Canary per carichi di lavoro ArgoCD stateless

Tieni traccia delle principali modifiche pianificate nel progetto Kubernetes: pensa al futuro

Non guardate solo alla prossima versione. Rivedi le nuove versioni di Kubernetes non appena vengono rilasciate e identifica le modifiche principali. Ad esempio, alcune applicazioni utilizzavano direttamente l'API docker e il supporto per Container Runtime Interface (CRI) per Docker (nota anche come Dockershim) è stato rimosso in Kubernetes. `1.24` Questo tipo di modifica richiede più tempo per prepararsi.

Esamina tutte le modifiche documentate per la versione a cui stai eseguendo l'aggiornamento e annota eventuali passaggi di aggiornamento richiesti. Inoltre, prendi nota di eventuali requisiti o procedure specifici per i cluster gestiti di Amazon EKS.

- Registro delle modifiche di Kubernetes

Linee guida specifiche sulla rimozione delle funzionalità

Rimozione di Dockershim nella versione 1.25 - Usa Detector for Docker Socket (DDS)

L'AMI ottimizzata EKS per 1.25 non include più il supporto per Dockershim. Se hai una dipendenza da Dockershim, ad esempio stai montando il socket Docker, dovrai rimuovere tali dipendenze prima di aggiornare i nodi di lavoro alla versione 1.25.

Trova i casi in cui hai una dipendenza dal socket Docker prima di eseguire l'aggiornamento alla 1.25. Ti consigliamo di usare Detector for Docker Socket (DDS), un plugin kubectl. .

Rimozione della PodSecurityPolicy versione 1.25 - Migrazione agli standard di sicurezza Pod o a una soluzione policy-as-code

`PodSecurityPolicy`era obsoleto in Kubernetes 1.21 ed è stato rimosso in Kubernetes 1.25. Se lo utilizzi PodSecurityPolicy nel tuo cluster, devi migrare agli standard di sicurezza Kubernetes Pod Security (PSS) integrati o a una soluzione prima di aggiornare il cluster alla versione 1.25 per evitare interruzioni dei carichi di lavoro. policy-as-code

AWS ha pubblicato una FAQ dettagliata nella documentazione EKS.

Consulta le best practice Pod Security Standards (PSS) e Pod Security Admission (PSA).

Leggi il post del blog PodSecurityPolicy Deprecation sul sito web di Kubernetes.

Deprecazione del driver di archiviazione In-Tree nella versione 1.23 - Migrazione ai driver CSI (Container Storage Interface)

La Container Storage Interface (CSI) è stata progettata per aiutare Kubernetes a sostituire i meccanismi esistenti dei driver di storage in-tree. La funzionalità di migrazione dell'interfaccia di archiviazione dei container (CSI) di Amazon EBS è abilitata per impostazione predefinita su cluster Amazon EKS `1.23` e successivi. Se disponi di pod in esecuzione su una versione `1.22` o su un cluster precedente, devi installare il driver CSI di Amazon EBS prima di aggiornare il cluster alla versione per `1.23` evitare interruzioni del servizio.

Consulta le domande frequenti sulla migrazione ad Amazon EBS CSI.

Risorse aggiuntive

ClowdHaus Guida all'aggiornamento di EKS

ClowdHaus EKS Upgrade Guidance è una CLI per facilitare l'aggiornamento dei cluster Amazon EKS. Può analizzare un cluster per individuare eventuali problemi potenziali da risolvere prima dell'aggiornamento.

GoNoGo

GoNoGoè uno strumento in fase alfa per determinare l'affidabilità degli aggiornamenti dei componenti aggiuntivi del cluster.

Le migliori pratiche per l'ottimizzazione dei costi

L'ottimizzazione dei costi significa raggiungere i risultati aziendali al prezzo più basso. Seguendo la documentazione in questa guida ottimizzerai i tuoi carichi di lavoro Amazon EKS.

Linee guida generali

Nel cloud, esistono diverse linee guida generali che possono aiutarti a ottimizzare i costi dei tuoi microservizi:

- Assicurati che i carichi di lavoro in esecuzione su Amazon EKS siano indipendenti dai tipi di infrastruttura specifici per l'esecuzione dei container, questo offrirà una maggiore flessibilità per quanto riguarda l'esecuzione sui tipi di infrastruttura meno costosi. Durante l'utilizzo di Amazon EKS con EC2, possono esserci delle eccezioni quando abbiamo carichi di lavoro che richiedono tipi specifici di EC2 istanze, come una GPU o altri tipi di istanze, a causa della natura del carico di lavoro.
- Seleziona istanze di container con profili ottimali: profila i tuoi ambienti di produzione o di preproduzione e monitora parametri critici come CPU e memoria, utilizzando servizi come Amazon Container CloudWatch Insights per Amazon EKS o strumenti di terze parti disponibili nell'ecosistema Kubernetes. In questo modo potremo allocare la giusta quantità di risorse ed evitare lo spreco di risorse.
- Sfrutta le diverse opzioni di acquisto disponibili in AWS per eseguire EKS con EC2, ad esempio On-Demand, Spot e Savings Plan.

Best practice EKS per l'ottimizzazione dei costi

Esistono tre aree di best practice generali per l'ottimizzazione dei costi nel cloud:

- Risorse convenienti (Auto Scaling, Ridimensionamento, politiche e opzioni di acquisto)
- Consapevolezza della spesa (utilizzo di AWS e strumenti di terze parti)
- Ottimizzazione nel tempo (Right Sizing)

Come per qualsiasi guida, ci sono dei compromessi. Assicuratevi di collaborare con la vostra organizzazione per comprendere le priorità di questo carico di lavoro e quali sono le migliori pratiche più importanti.

Come utilizzare questa guida

Questa guida è destinata ai team DevOps responsabili dell'implementazione e della gestione dei cluster EKS e dei carichi di lavoro che supportano. La guida è organizzata in diverse aree di best practice per facilitarne l'utilizzo. Ogni argomento contiene un elenco di consigli, strumenti da utilizzare e best practice per l'ottimizzazione dei costi dei cluster EKS. Non è necessario leggere gli argomenti in un ordine particolare.

Caratteristiche principali dei servizi AWS e Kubernetes

L'ottimizzazione dei costi è supportata dai seguenti servizi e funzionalità AWS:

- EC2 Tipi di istanze, Savings Plan (e istanze riservate) e istanze Spot, a prezzi diversi.
- Auto Scaling insieme alle politiche di Auto Scaling native di Kubernetes. Prendi in considerazione Savings Plan (istanze precedentemente riservate) per carichi di lavoro prevedibili. Utilizza archivi dati gestiti come EBS ed EFS, per l'elasticità e la durabilità dei dati delle applicazioni.
- La dashboard della console Billing and Cost Management insieme ad AWS Cost Explorer fornisce una panoramica dell'utilizzo di AWS. Usa AWS Organizations per dettagli di fatturazione granulari. Sono stati inoltre condivisi i dettagli di diversi strumenti di terze parti.
- Amazon CloudWatch Container Metrics fornisce parametri sull'utilizzo delle risorse da parte del cluster EKS. Oltre alla dashboard Kubernetes, nell'ecosistema Kubernetes sono disponibili diversi strumenti che possono essere utilizzati per ridurre gli sprechi.

Questa guida include una serie di consigli che puoi utilizzare per migliorare l'ottimizzazione dei costi del tuo cluster Amazon EKS.

Feedback

Questa guida è stata pubblicata GitHub per raccogliere feedback e suggerimenti diretti dalla più ampia community EKS/Kubernetes. Se hai una best practice che ritieni debba essere inclusa nella guida, segnala un problema o invia un PR nell'archivio. GitHub La nostra intenzione è quella di aggiornare periodicamente la guida man mano che vengono aggiunte nuove funzionalità al servizio o quando si evolve una nuova best practice.

Framework di ottimizzazione dei costi

AWS Cloud Economics è una disciplina che aiuta i clienti ad aumentare l'efficienza e ridurre i costi attraverso l'adozione di moderne tecnologie di elaborazione come Amazon EKS. La disciplina consiglia di seguire una metodologia chiamata «Cloud Financial Management (CFM) framework» che consiste di 4 pilastri:

Il pilastro See: misurazione e responsabilità

Il pilastro See è un insieme fondamentale di attività e tecnologie che definiscono come misurare, monitorare e rendere conto della spesa per il cloud. Viene spesso definita «osservabilità», «strumentazione» o «telemetria». Le capacità e i limiti dell'infrastruttura «Observability» determinano cosa può essere ottimizzato. Ottenere un quadro chiaro dei costi è un primo passo fondamentale per l'ottimizzazione dei costi, in quanto è necessario sapere da dove si parte. Questo tipo di visibilità guiderà anche i tipi di attività da svolgere per ottimizzare ulteriormente l'ambiente.

Ecco una breve panoramica delle nostre migliori pratiche per il pilastro See:

- Definisci e mantieni una strategia di tagging per i tuoi carichi di lavoro.
  - Utilizza Instance Tagging, l'etichettatura dei cluster EKS consente di visualizzare i costi dei singoli cluster e di allocarli nei report su costi e utilizzo.
- Stabilisci la reportistica e il monitoraggio dell'utilizzo di EKS utilizzando tecnologie come Kubecost.
  - Abilita Cloud Intelligence Dashboards, etichettando correttamente le risorse e utilizzando le visualizzazioni, puoi misurare e stimare i costi.

- Alloca i costi del cloud alle applicazioni, alle linee di business (LoBs) e ai flussi di entrate.
- Definisci, misura e comunica efficiency/value KPIs con gli stakeholder aziendali. Ad esempio, crea un KPI «metrico unitario» che misuri il costo per transazione, ad esempio un servizio di ride sharing potrebbe avere un KPI per «costo per corsa».

Per maggiori dettagli sulle tecnologie e sulle attività consigliate associate a questo pilastro, consulta la sezione Ottimizzazione dei costi - Osservabilità di questa guida.

Il pilastro Save: ottimizzazione dei costi

Questo pilastro si basa sulle tecnologie e sulle capacità sviluppate nel pilastro «See». Le seguenti attività rientrano in genere in questo pilastro:

- Identifica ed elimina gli sprechi nel tuo ambiente.
- Architettura e progettazione per l'efficienza dei costi.
- Scegli l'opzione di acquisto migliore, ad esempio istanze on demand o istanze Spot.
- Adattati all'evoluzione dei servizi: man mano che i servizi AWS si evolvono, il modo di utilizzarli in modo efficiente potrebbe cambiare. Sii disposto ad adattarti per tenere conto di questi cambiamenti.

Poiché queste attività sono operative, dipendono in larga misura dalle caratteristiche dell'ambiente. Chiediti: quali sono i principali fattori che determinano i costi? Quale valore aziendale offrono i diversi ambienti? Quali opzioni di acquisto e scelte di infrastruttura, ad esempio i tipi di famiglie di istanze, sono più adatte per ogni ambiente?

Di seguito è riportato un elenco prioritario dei fattori di costo più comuni per i cluster EKS:

1. Costi di calcolo: la combinazione di più tipi di famiglie di istanze, le opzioni di acquisto e il bilanciamento della scalabilità con la disponibilità richiedono un'attenta considerazione. Per ulteriori informazioni, consulta i consigli nella sezione Ottimizzazione dei costi - Elaborazione di questa guida.
2. Costi di rete: l'utilizzo di 3 AZs for EKS cluster può potenzialmente aumentare i costi del traffico inter-AZ. Per i nostri consigli su come bilanciare i requisiti HA con il contenimento dei costi del traffico di rete, consulta la sezione Ottimizzazione dei costi - Rete di questa guida.
3. Costi di storage: a seconda della stateful/stateless natura dei carichi di lavoro nei cluster EKS e di come vengono utilizzati i diversi tipi di storage, lo storage può essere considerato parte del carico

di lavoro. Per considerazioni relative ai costi di storage EKS, consulta la sezione Ottimizzazione dei costi - Archiviazione di questa guida.

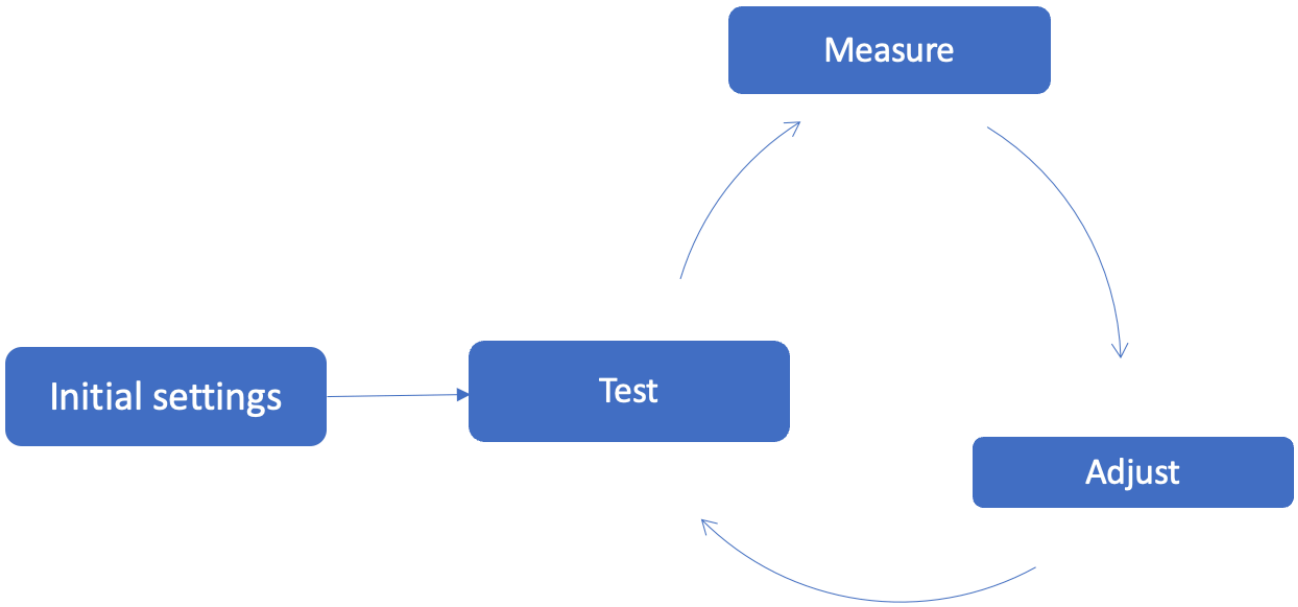
Il pilastro del piano: pianificazione e previsione

Una volta implementate le raccomandazioni del pilastro See, i cluster vengono ottimizzati su base continuativa. Grazie all'esperienza acquisita nella gestione efficiente dei cluster, le attività di pianificazione e previsione possono concentrarsi su:

- Pianificazione e previsione dei costi del cloud in modo dinamico.
- Quantificazione del valore aziendale fornito dai servizi container EKS.
- Integrazione della gestione dei costi dei cluster EKS con la pianificazione della gestione finanziaria IT.

Il pilastro Run

L'ottimizzazione dei costi è un processo continuo e comporta una serie di miglioramenti incrementali:



Garantire la sponsorizzazione dei dirigenti per questo tipo di attività è fondamentale per integrare l'ottimizzazione dei cluster EKS negli sforzi "" dell'organizzazione. FinOps Consente l'allineamento delle parti interessate attraverso una comprensione condivisa dei costi dei cluster EKS,

l'implementazione di protezioni sui costi dei cluster EKS e la garanzia che gli strumenti, l'automazione e le attività si evolvano in base alle esigenze dell'organizzazione.

Riferimenti

- AWS Cloud Economics, gestione finanziaria nel cloud

Consapevolezza della spesa

La consapevolezza delle spese consiste nel capire chi, dove e cosa causa le spese nel cluster EKS. Ottenere un quadro preciso di questi dati contribuirà a sensibilizzare l'opinione pubblica sulla spesa e a evidenziare le aree da correggere.

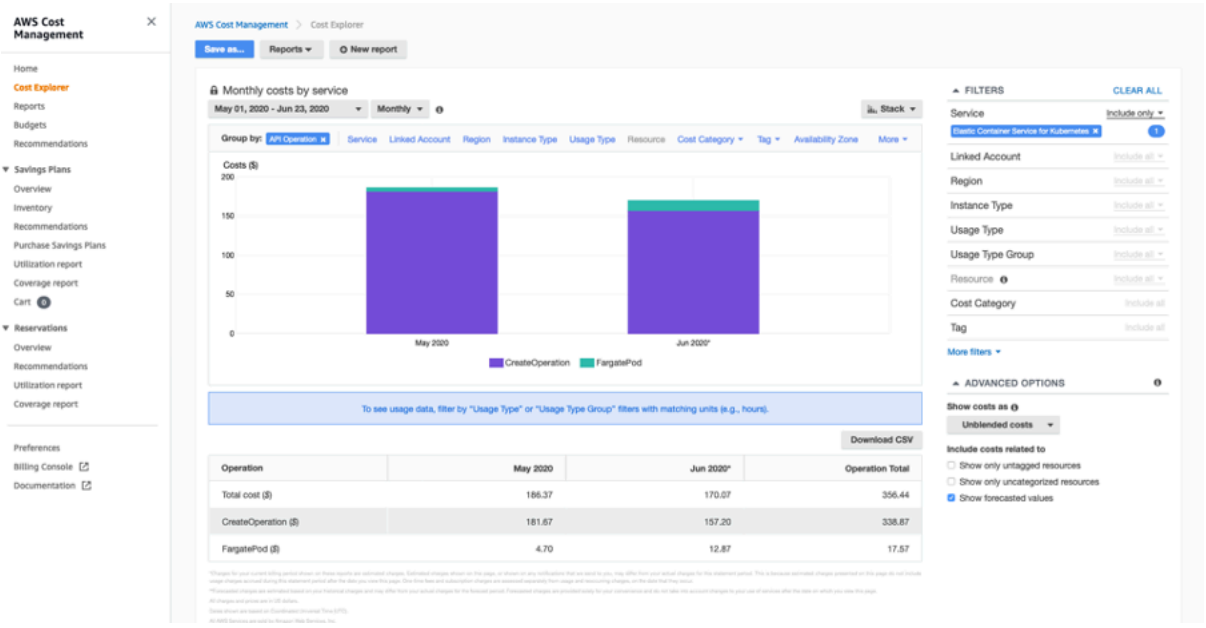
Raccomandazioni

Usa Cost Explorer

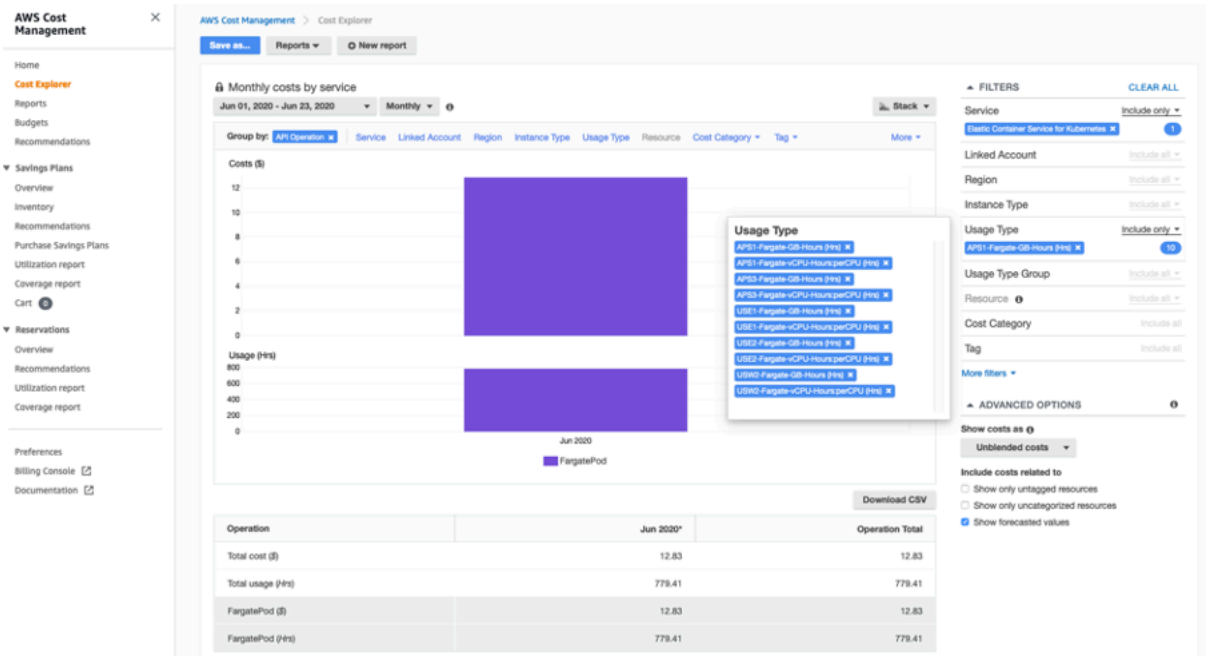
AWS Cost Explorer ha un' easy-to-useinterfaccia che consente di visualizzare, comprendere e gestire i costi e l'utilizzo di AWS nel tempo. È possibile analizzare i dati di costi e utilizzo a vari livelli utilizzando i filtri disponibili in Cost Explorer.

Costi di EKS Control Plane e EKS Fargate

Utilizzando i filtri, possiamo interrogare i costi sostenuti per i costi EKS presso Control Plane e Fargate Pod come mostrato nello schema seguente:



Utilizzando i filtri, possiamo interrogare i costi aggregati sostenuti per i Fargate Pod nelle diverse regioni di EKS, che includono sia le ore vCPU per CPU che i GB/ora, come mostrato nel diagramma seguente:



Etichettatura delle risorse

Amazon EKS supporta l'aggiunta di tag AWS ai cluster Amazon EKS. Ciò semplifica il controllo dell'accesso all'API EKS per la gestione dei cluster. I tag aggiunti a un cluster EKS sono specifici della risorsa del cluster AWS EKS, non si propagano ad altre risorse AWS utilizzate dal cluster come EC2 istanze o sistemi di bilanciamento del carico. Oggi, il tagging dei cluster è supportato per tutti i cluster EKS nuovi ed esistenti tramite l'API AWS, la console e. SDKs

AWS Fargate è una tecnologia che fornisce capacità di elaborazione on-demand e della giusta dimensione per i container. Prima di poter pianificare i pod su Fargate nel cluster, è necessario definire almeno un profilo Fargate che specifichi quali pod devono utilizzare Fargate al momento del lancio.

Aggiungere ed elencare tag a un cluster EKS:

```
$ aws eks tag-resource --resource-arn arn:aws:eks:us-west-2:xxx:cluster/ekscluster1 --
tags team=devops,env=staging,bu=cio,costcenter=1234
$ aws eks list-tags-for-resource --resource-arn arn:aws:eks:us-west-2:xxx:cluster/
ekscluster1
{
    "tags": {
        "bu": "cio",
```

```
        "env": "staging",
        "costcenter": "1234",
        "team": "devops"
    }
}
```

Dopo aver attivato i tag di allocazione dei costi in AWS Cost Explorer, AWS utilizza i tag di allocazione dei costi per organizzare i costi delle risorse nel report di allocazione dei costi, per semplificare la categorizzazione e il monitoraggio dei costi AWS.

I tag non hanno alcun significato semantico per Amazon EKS e vengono interpretati rigorosamente come una stringa di caratteri. Ad esempio, puoi definire un set di tag per i cluster Amazon EKS per monitorare il proprietario di ogni cluster e il livello di stack.

Usa AWS Trusted Advisor

AWS Trusted Advisor offre una vasta gamma di controlli e raccomandazioni sulle best practice in cinque categorie: ottimizzazione dei costi, sicurezza, tolleranza agli errori, prestazioni e limiti del servizio.

Per l'ottimizzazione dei costi, Trusted Advisor aiuta a eliminare le risorse inutilizzate e inattive e consiglia di impegnarsi per la capacità riservata. Le azioni chiave che aiuteranno Amazon EKS riguarderanno, tra le altre cose, EC2 le istanze poco utilizzate, gli indirizzi IP elastici non associati, i sistemi Idle Load Balancer, i volumi EBS sottoutilizzati. L'elenco completo dei controlli è disponibile su -practice-checklist/. https://aws.amazon.com/premiumsupport/ technology/trusted-advisor/best

Il Trusted Advisor fornisce anche consigli su Savings Plans e Reserved EC2 Instances per le istanze e Fargate, che consente di impegnarsi a utilizzare un importo costante in cambio di tariffe scontate.

> **Note**
>
> Le raccomandazioni di Trusted Advisor sono raccomandazioni generiche e non specifiche per EKS.

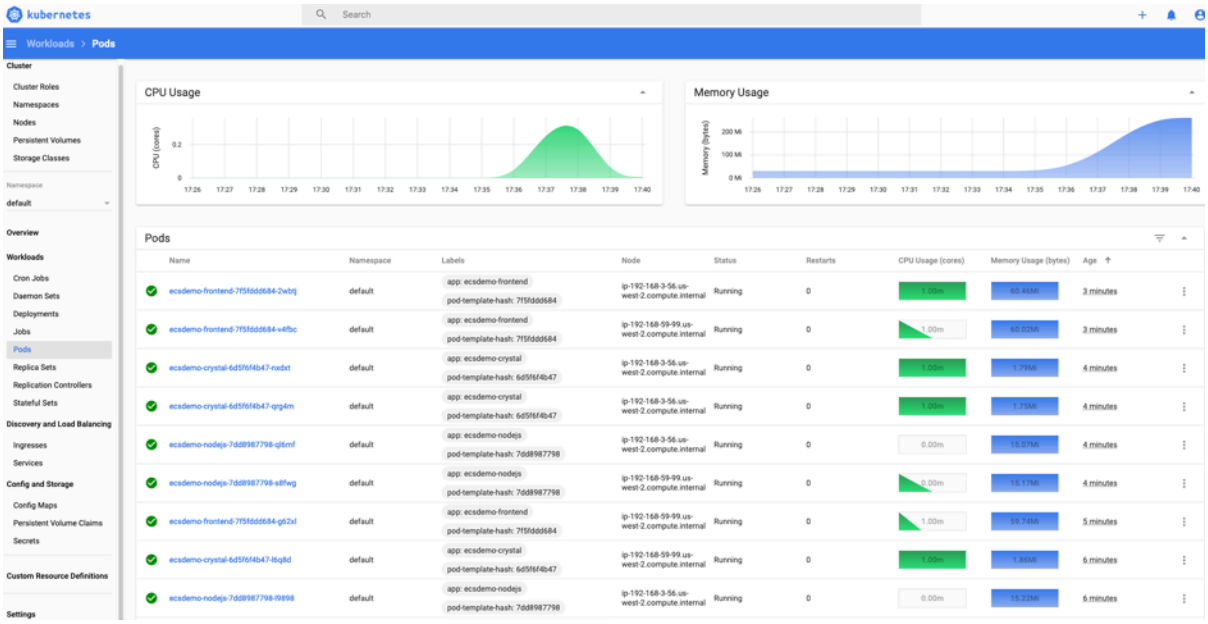
Usa la dashboard di Kubernetes

Dashboard Kubernetes

Kubernetes Dashboard è un'interfaccia utente generica basata sul Web per i cluster Kubernetes, che fornisce informazioni sul cluster Kubernetes, incluso l'utilizzo delle risorse a livello di cluster,

nodo e pod. La distribuzione della dashboard Kubernetes su un cluster Amazon EKS è descritta nella documentazione di Amazon EKS.

La dashboard fornisce informazioni dettagliate sull'utilizzo delle risorse per ogni nodo e pod, oltre a metadati dettagliati su pod, servizi, implementazioni e altri oggetti Kubernetes. Queste informazioni consolidate forniscono visibilità sull'ambiente Kubernetes.



kubectl ferma e descrive i comandi

Visualizzazione delle metriche sull'utilizzo delle risorse con i comandi kubectl top e kubectl descrizione. kubectl top mostrerà l'uso corrente di CPU e memoria per i pod o i nodi del cluster o per un pod o un nodo specifico. Il comando kubectl describe fornirà informazioni più dettagliate su un nodo o un pod specifico.

```
$ kubectl top pods
$ kubectl top nodes
$ kubectl top pod pod-name --namespace mynamespace --containers
```

Utilizzando il comando top, l'output mostrerà la quantità totale di CPU (in core) e memoria (in MiB) utilizzata dal nodo e le percentuali della capacità allocabile del nodo rappresentate da tali numeri. È quindi possibile eseguire il drill-down fino al livello successivo, il livello del contenitore all'interno dei pod, aggiungendo il flag --containers.

```
$ kubectl describe node <node>
$ kubectl describe pod <pod>
```

kubectl describe restituisce la percentuale della capacità totale disponibile rappresentata da ogni richiesta o limite di risorse.

kubectl top and describe, monitora l'utilizzo e la disponibilità di risorse critiche come CPU, memoria e storage su pod, nodi e contenitori Kubernetes. Questa consapevolezza aiuterà a comprendere l'utilizzo delle risorse e a controllare i costi.

Usa CloudWatch Container Insights

Usa CloudWatch Container Insights per raccogliere, aggregare e riepilogare metriche e log delle tue applicazioni e microservizi containerizzati. Container Insights è disponibile per Amazon Elastic Kubernetes Service EC2 su e piattaforme Kubernetes su Amazon. EC2 I parametri includono l'utilizzo di risorse come CPU, memoria, dischi e rete.

L'installazione di Insights è riportata nella documentazione.

CloudWatch crea metriche aggregate a livello di cluster, nodo, pod, task e servizio come CloudWatch metriche.

La seguente query mostra un elenco di nodi, ordinati in base all'utilizzo medio della CPU del nodo

```
STATS avg(node_cpu_utilization) as avg_node_cpu_utilization by NodeName
| SORT avg_node_cpu_utilization DESC
```

Utilizzo della CPU per nome del contenitore

```
stats pct(container_cpu_usage_total, 50) as CPUPercMedian by kubernetes.container_name
| filter Type="Container"
```

Utilizzo del disco in base al nome del contenitore

```
stats floor(avg(container_filesystem_usage/1024)) as container_filesystem_usage_avg_kb
 by InstanceId, kubernetes.container_name, device
| filter Type="ContainerFS"
| sort container_filesystem_usage_avg_kb desc
```

Altre interrogazioni di esempio sono fornite nella documentazione di Container Insights

Questa consapevolezza aiuterà a comprendere l'utilizzo delle risorse e a controllare i costi.

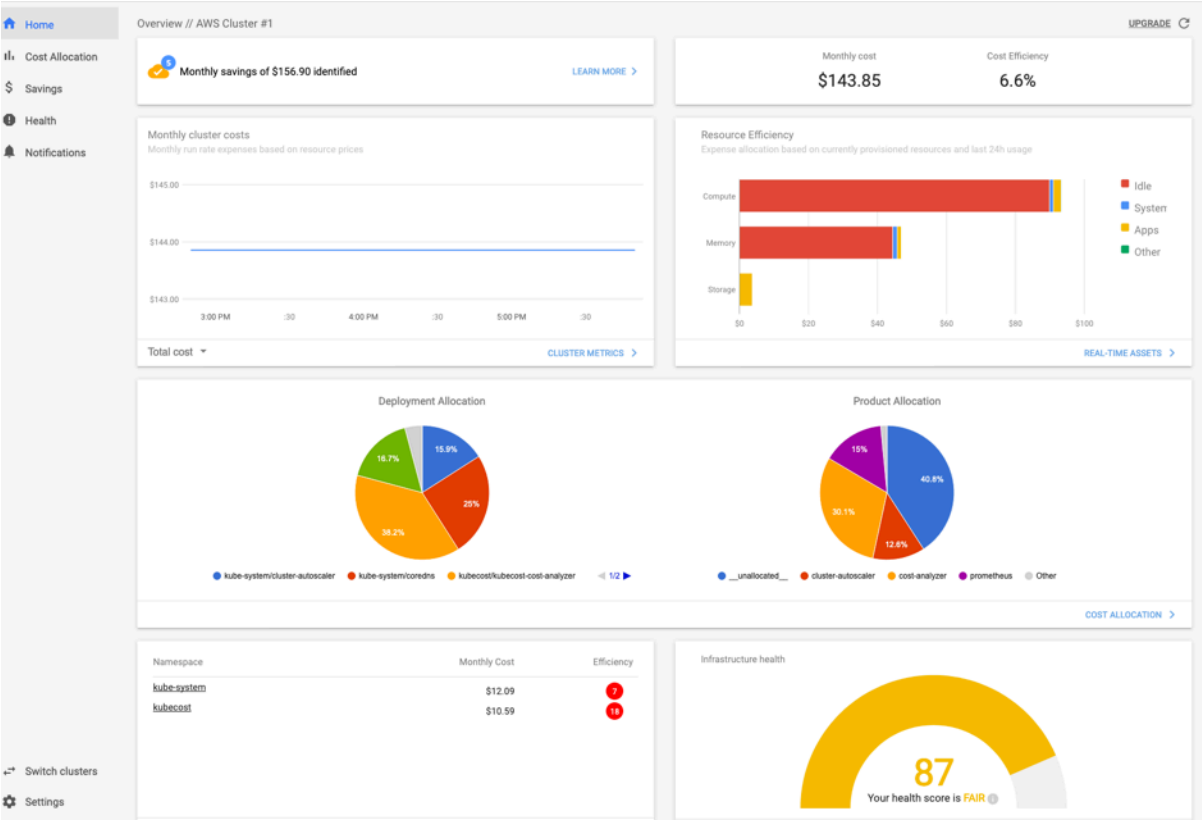
Utilizzo di Kubecost per la consapevolezza e la guida alla spesa

Strumenti di terze parti come kubecost possono essere implementati anche su Amazon EKS per ottenere visibilità sui costi di gestione del cluster Kubernetes. Fai riferimento a questo blog di AWS per tracciare i costi con Kubecost

Implementazione di kubecost con Helm 3:

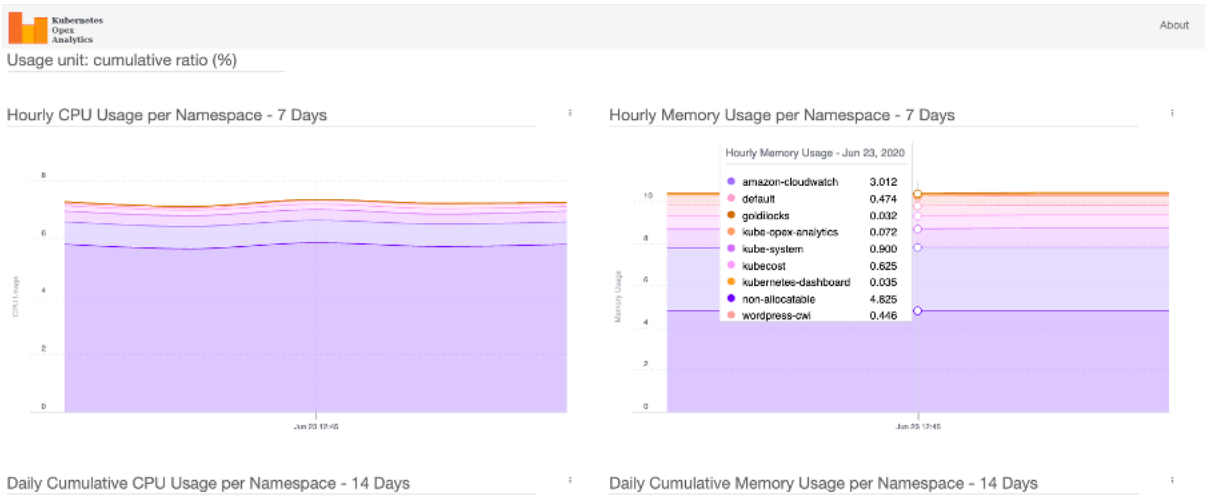
```
$ curl -sSL https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3 |
 bash
$ helm version --short
v3.2.1+gfe51cd1
$ helm repo add stable https://kubernetes-charts.storage.googleapis.com/
$ helm repo add stable https://kubernetes-charts.storage.googleapis.com/c^C
$ kubectl create namespace kubecost
namespace/kubecost created
$ helm repo add kubecost https://kubecost.github.io/cost-analyzer/
"kubecost" has been added to your repositories

$ helm install kubecost kubecost/cost-analyzer --namespace kubecost --set
 kubecostToken="aGRoZEBqc2pzLmNvbQ==xm343yadf98"
NAME: kubecost
LAST DEPLOYED: Mon May 18 08:49:05 2020
NAMESPACE: kubecost
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
----------------------------------------------------Kubecost has been successfully
 installed. When pods are Ready, you can enable port-forwarding with the following
 command:

    kubectl port-forward --namespace kubecost deployment/kubecost-cost-analyzer 9090

Next, navigate to http://localhost:9090 in a web browser.
$ kubectl port-forward --namespace kubecost deployment/kubecost-cost-analyzer 9090

NOTE: If you are using Cloud 9 or have a need to forward it to a different port like
 8080, issue the following command
$ kubectl port-forward --namespace kubecost deployment/kubecost-cost-analyzer 8080:9090
```

Dashboard Kubecost -



Utilizza lo strumento di analisi dell'allocazione dei costi e della pianificazione della capacità Kubernetes

Kubernetes Opex Analytics è uno strumento che aiuta le organizzazioni a tenere traccia delle risorse consumate dai propri cluster Kubernetes per evitare di pagare più del dovuto. A tal fine, genera report di utilizzo a breve (7 giorni), medio (14 giorni) e a lungo termine (12 mesi) che mostrano informazioni pertinenti sulla quantità di risorse impiegate da ciascun progetto nel tempo.



Yotascale

Yotascale aiuta ad allocare con precisione i costi di Kubernetes. La funzionalità di allocazione dei costi di Yotascale Kubernetes utilizza i dati sui costi effettivi, che includono gli sconti sulle istanze riservate e i prezzi delle istanze spot anziché le stime generiche dei tassi di mercato, per determinare l'impatto totale dei costi di Kubernetes

Maggiori dettagli sono disponibili sul loro sito Web.

Alcide Advisor

Alcide è un partner tecnologico avanzato di AWS Partner Network (APN). Alcide Advisor aiuta a garantire che la configurazione di cluster, nodi e pod Amazon EKS sia ottimizzata per funzionare secondo le migliori pratiche di sicurezza e le linee guida interne. Alcide Advisor è un servizio senza agenti per l'audit e la conformità di Kubernetes, progettato per garantire un flusso sicuro e senza attriti, rafforzando la fase di sviluppo prima di passare alla produzione. DevSecOps

Maggiori dettagli sono disponibili in questo post del blog.

Altri strumenti

Raccolta dei rifiuti Kubernetes

Il ruolo del Garbage Collector di Kubernetes consiste nell'eliminare determinati oggetti che prima avevano un proprietario, ma che ora non lo hanno più.

Contea di Fargate

Fargatecount è uno strumento utile che consente ai clienti AWS di tracciare, con una CloudWatch metrica personalizzata, il numero totale di pod EKS che sono stati distribuiti su Fargate in una regione specifica di un account specifico. Questo aiuta a tenere traccia di tutti i pod Fargate in esecuzione su un cluster EKS.

Popeye: un disinfettante per cluster Kubernetes

Popeye - A Kubernetes Cluster Sanitizer è un'utilità che esegue la scansione del cluster Kubernetes in tempo reale e segnala potenziali problemi con le risorse e le configurazioni distribuite. Disinfetta il cluster in base a ciò che è distribuito e non a ciò che è memorizzato sul disco. Scansionando il cluster, rileva le configurazioni errate e aiuta a garantire l'adozione delle migliori pratiche

Resources

Consulta le seguenti risorse per saperne di più sulle migliori pratiche per l'ottimizzazione dei costi.

Documentazione e blog

- Amazon EKS supporta l'etichettatura

Tools (Strumenti)

- Cos'è AWS Billing and Cost Management?
- Amazon CloudWatch Container Insights
- Come tenere traccia dei costi nei cluster Amazon EKS multi-tenant utilizzando Kubecost
- Kubecost
- Kube Opsview
- Analisi Kubernetes Opex

Calcolo e scalabilità automatica

In qualità di sviluppatore, farai delle stime sui requisiti di risorse della tua applicazione, ad esempio CPU e memoria, ma se non li aggiusti continuamente potrebbero diventare obsoleti, il che potrebbe aumentare i costi e peggiorare le prestazioni e l'affidabilità. Adeguare continuamente i requisiti di risorse di un'applicazione è più importante che soddisfarli correttamente al primo tentativo.

Le migliori pratiche menzionate di seguito vi aiuteranno a creare e gestire carichi di lavoro attenti ai costi che consentano di raggiungere risultati aziendali riducendo al minimo i costi e consentendo all'organizzazione di massimizzare il ritorno sull'investimento. Un ordine di importanza elevato per l'ottimizzazione dei costi di elaborazione del cluster è:

1. Carichi di lavoro della giusta dimensione
2. Riduci la capacità inutilizzata
3. Ottimizza i tipi di capacità di elaborazione (ad esempio Spot) e gli acceleratori (ad es.) GPUs

Dimensiona correttamente i tuoi carichi di lavoro

Nella maggior parte dei cluster EKS, la maggior parte dei costi proviene dalle EC2 istanze utilizzate per eseguire i carichi di lavoro containerizzati. Non sarete in grado di dimensionare correttamente le risorse di elaborazione senza comprendere i requisiti dei vostri carichi di lavoro. Ecco perché è essenziale utilizzare le richieste e i limiti appropriati e apportare le modifiche necessarie a tali impostazioni. Inoltre, le dipendenze, come la dimensione dell'istanza e la selezione dello storage, possono influire sulle prestazioni del carico di lavoro, con una serie di conseguenze indesiderate su costi e affidabilità.

Le richieste devono essere in linea con l'utilizzo effettivo. Se le richieste di un container sono troppo elevate, ci sarà una capacità inutilizzata, che è un fattore importante nei costi totali del cluster. Ogni contenitore in un pod, ad esempio l'applicazione e i sidecar, dovrebbe avere le proprie richieste e limiti impostati per garantire che i limiti aggregati dei pod siano il più precisi possibile.

Utilizza strumenti come Goldilocks, KRR e Kubecost che stimano le richieste di risorse e i limiti per i tuoi contenitori. A seconda della natura delle applicazioni, dei performance/cost requisiti e della complessità, è necessario valutare quali sono le metriche migliori su cui basare la scalabilità, in che momento le prestazioni dell'applicazione peggiorano (punto di saturazione) e come modificare di conseguenza le richieste e i limiti. Per ulteriori informazioni su questo argomento, consulta Application right sizing.

Ti consigliamo di utilizzare Horizontal Pod Autoscaler (HPA) per controllare quante repliche dell'applicazione devono essere in esecuzione, Vertical Pod Autoscaler (VPA) per regolare il numero di richieste e limiti necessari all'applicazione per replica e uno scaler automatico a nodi come Karpenter o Cluster Autoscaler per regolare continuamente il numero totale di nodi nel cluster. Le tecniche di ottimizzazione dei costi che utilizzano Karpenter e Cluster Autoscaler sono documentate in una sezione successiva di questo documento.

Il Vertical Pod Autoscaler può regolare le richieste e i limiti assegnati ai container in modo che i carichi di lavoro funzionino in modo ottimale. È necessario eseguire il VPA in modalità di controllo in modo che non apporti automaticamente modifiche e non riavvii i pod. Suggerirà modifiche in base alle metriche osservate. In caso di modifiche che influiscono sui carichi di lavoro di produzione, è consigliabile esaminarle e testarle prima in un ambiente non di produzione, poiché possono avere un impatto sull'affidabilità e sulle prestazioni dell'applicazione.

Ridurre i consumi

Il modo migliore per risparmiare è fornire meno risorse. Un modo per farlo è adattare i carichi di lavoro in base ai requisiti attuali. Dovresti iniziare qualsiasi iniziativa di ottimizzazione dei costi assicurandoti che i carichi di lavoro definiscano i requisiti e si scalino dinamicamente. Ciò richiederà l'acquisizione di metriche dalle applicazioni e l'impostazione di configurazioni come Pod Readiness Gates per garantire che l'applicazione possa scalare verso l'alto `PodDisruptionBudgets`e verso il basso in modo sicuro e dinamico. È importante considerare che un approccio restrittivo PodDisruptionBudgets può impedire a Cluster Autoscaler e Karpenter di ridimensionare i nodi, poiché sia Cluster Autoscaler che Karpenter lo rispettano. PodDisruptionBudgets Il valore 'minAvailable' in PodDisruptionBudget deve essere sempre inferiore al numero di pod nella distribuzione e dovresti mantenere un buon buffer tra i due, ad esempio in una distribuzione di 6 pod in cui desideri che siano sempre attivi almeno 4 pod, imposta 'minAvailable' nel tuo PodDisruptionBidget su 4. Ciò consentirà a Cluster Autoscaler e Karpenter di drenare ed eliminare in sicurezza i pod dai nodi sottoutilizzati durante un evento di scale-down del nodo. Consulta il documento delle domande frequenti su Cluster Autoscaler.

Horizontal Pod Autoscaler è un autoscaler flessibile per carichi di lavoro in grado di regolare il numero di repliche necessarie per soddisfare i requisiti di prestazioni e affidabilità dell'applicazione. Dispone di un modello flessibile per definire quando scalare verso l'alto e verso il basso in base a varie metriche come CPU, memoria o metriche personalizzate, ad esempio profondità della coda, numero di connessioni a un pod, ecc.

Il Kubernetes Metrics Server consente la scalabilità in risposta a metriche integrate come l'utilizzo di CPU e memoria, ma se desideri scalare in base ad altre metriche, come la profondità della coda di Amazon CloudWatch o SQS, dovresti prendere in considerazione progetti di scalabilità automatica basati su eventi come KEDA. Consulta questo post del blog su come utilizzare KEDA con le metriche. CloudWatch Se non sei sicuro delle metriche da monitorare e su cui basare la scalabilità, consulta le migliori pratiche per monitorare le metriche più importanti.

La riduzione del consumo del carico di lavoro crea un eccesso di capacità in un cluster e, con una corretta configurazione di scalabilità automatica, è possibile ridimensionare automaticamente i nodi e ridurre la spesa totale. Ti consigliamo di non cercare di ottimizzare la capacità di elaborazione manualmente. Lo scheduler Kubernetes e gli autoscaler dei nodi sono stati progettati per gestire questo processo al posto tuo.

Riduci la capacità inutilizzata

Dopo aver determinato la dimensione corretta per le applicazioni, riducendo le richieste in eccesso, è possibile iniziare a ridurre la capacità di elaborazione assegnata. Dovresti essere in grado di eseguire questa operazione in modo dinamico se hai impiegato il tempo necessario per dimensionare correttamente i carichi di lavoro illustrato nelle sezioni precedenti. Esistono due scaler automatici di nodi principali utilizzati con Kubernetes in AWS.

Karpenter e Cluster Autoscaler

Sia Karpenter che Kubernetes Cluster Autoscaler ridimensioneranno il numero di nodi del cluster man mano che i pod vengono creati o rimossi e i requisiti di elaborazione cambiano. L'obiettivo principale di entrambi è lo stesso, ma Karpenter adotta un approccio diverso per la gestione dei nodi, il provisioning e il de-provisioning, che può aiutare a ridurre i costi e ottimizzare l'utilizzo a livello di cluster.

Con l'aumentare delle dimensioni dei cluster e l'aumento della varietà di carichi di lavoro, diventa sempre più difficile preconfigurare gruppi di nodi e istanze. Proprio come per le richieste di carichi di lavoro, è importante impostare una linea di base iniziale e adattarla continuamente in base alle esigenze.

Se si utilizza Cluster Autoscaler, rispetterà i valori «minimo» e «massimo» di ciascun gruppo di Auto Scaling (ASG) e regolerà solo il valore «desiderato». È importante prestare attenzione durante l'impostazione di questi valori per l'ASG sottostante poiché Cluster Autoscaler non sarà in grado di ridimensionare un ASG oltre il numero «minimo». Imposta il conteggio «desiderato» come il numero di nodi necessari durante il normale orario lavorativo e «minimo» come il numero di nodi necessari durante le ore non lavorative. Consulta il documento delle domande frequenti su Cluster Autoscaler.

Cluster Autoscaler Priority Expander

Kubernetes Cluster Autoscaler funziona scalando gruppi di nodi, chiamati gruppi di nodi, verso l'alto e verso il basso man mano che le applicazioni si scalano verso l'alto e verso il basso. Se non stai scalando dinamicamente i carichi di lavoro, Cluster Autoscaler non ti aiuterà a risparmiare denaro.

Cluster Autoscaler richiede che un amministratore del cluster crei in anticipo gruppi di nodi per l'utilizzo dei carichi di lavoro. I gruppi di nodi devono essere configurati per utilizzare istanze con lo stesso «profilo», ovvero all'incirca la stessa quantità di CPU e memoria.

È possibile avere più gruppi di nodi e Cluster Autoscaler può essere configurato per impostare livelli di scalabilità prioritari e ogni gruppo di nodi può contenere nodi di dimensioni diverse. I gruppi di nodi possono avere diversi tipi di capacità e l'espansore di priorità può essere utilizzato innanzitutto per scalare gruppi meno costosi.

Di seguito è riportato un esempio di un frammento di configurazione del cluster che utilizza `ConfigMap` a per dare priorità alla capacità riservata prima di utilizzare istanze su richiesta. È possibile utilizzare la stessa tecnica per dare priorità alle istanze Graviton o Spot rispetto ad altri tipi.

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata:
  name: my-cluster
managedNodeGroups:
  - name: managed-ondemand
    minSize: 1
    maxSize: 7
    instanceType: m5.xlarge
  - name: managed-reserved
    minSize: 2
    maxSize: 10
    instanceType: c5.2xlarge
```

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-autoscaler-priority-expander
  namespace: kube-system
data:
  priorities: |-
    10:
      - .*ondemand.*
    50:
      - .*reserved.*
```

L'uso dei gruppi di nodi può aiutare le risorse di elaborazione sottostanti a eseguire le operazioni previste per impostazione predefinita, ad esempio distribuendo i nodi tra i vari carichi di lavoro

AZs, ma non tutti i carichi di lavoro hanno gli stessi requisiti o aspettative ed è meglio lasciare che le applicazioni dichiarino i propri requisiti in modo esplicito. Per ulteriori informazioni su Cluster Autoscaler, consulta la sezione sulle migliori pratiche.

Descheduler

Cluster Autoscaler può aggiungere e rimuovere la capacità dei nodi da un cluster in base alla necessità di pianificare nuovi pod o al sottoutilizzo dei nodi. Non fornisce una visione completa del posizionamento dei pod dopo che sono stati pianificati su un nodo. Se utilizzi Cluster Autoscaler, dovresti anche dare un'occhiata al descheduler Kubernetes per evitare di sprecare capacità nel tuo cluster.

Se hai 10 nodi in un cluster e ogni nodo è utilizzato al 60%, non stai utilizzando il 40% della capacità assegnata nel cluster. Con Cluster Autoscaler è possibile impostare la soglia di utilizzo per nodo al 60%, ma in questo modo si cercherà di ridurre un singolo nodo solo dopo che l'utilizzo è sceso al di sotto del 60%.

Con il descheduler può esaminare la capacità e l'utilizzo del cluster dopo che i pod sono stati pianificati o i nodi sono stati aggiunti al cluster. Tenta di mantenere la capacità totale del cluster al di sopra di una soglia specificata. Può anche rimuovere i pod in base a problemi di nodo o nuovi nodi che si uniscono al cluster per assicurarsi che i pod funzionino nel loro ambiente di calcolo ottimale. Nota che, descheduler non pianifica la sostituzione dei pod eliminati, ma a tale scopo si affida allo scheduler predefinito.

Consolidamento Karpenter

Karpenter adotta un approccio «senza gruppi» alla gestione dei nodi. Questo approccio è più flessibile per diversi tipi di carico di lavoro e richiede meno configurazioni iniziali per gli amministratori del cluster. Invece di predefinire i gruppi e scalare ogni gruppo in base alle esigenze dei carichi di lavoro, Karpenter utilizza i provisioner e i modelli di nodi per definire in generale il tipo di EC2 istanze che possono essere create e le impostazioni relative alle istanze man mano che vengono create.

Il bin packing è la pratica di utilizzare una maggiore quantità di risorse dell'istanza impacchettando più carichi di lavoro su un numero inferiore di istanze di dimensioni ottimali. Sebbene ciò contribuisca a ridurre i costi di elaborazione fornendo solo le risorse utilizzate dai carichi di lavoro, presenta un compromesso. L'avvio di nuovi carichi di lavoro può richiedere più tempo perché è necessario aggiungere capacità al cluster, specialmente durante eventi di scalabilità su larga scala. Considerate l'equilibrio tra ottimizzazione dei costi, prestazioni e disponibilità quando impostate l'imballaggio dei contenitori.

Karpenter può monitorare e comprimere continuamente per migliorare l'utilizzo delle risorse delle istanze e ridurre i costi di elaborazione. Karpenter può anche selezionare un nodo di lavoro più efficiente in termini di costi per il tuo carico di lavoro. Ciò può essere ottenuto attivando il flag «consolidation» su true nel provisioner (frammento di codice di esempio riportato di seguito). L'esempio seguente mostra un provider di esempio che consente il consolidamento. Al momento della stesura di questa guida, Karpenter non sostituirà un'istanza Spot in esecuzione con un'istanza Spot più economica. Per ulteriori dettagli sul consolidamento di Karpenter, consulta questo blog.

```
apiVersion: karpenter.sh/v1
kind: Provisioner
metadata:
  name: enable-binpacking
spec:
  consolidation:
    enabled: true
```

Per carichi di lavoro che potrebbero non essere interrompibili, ad esempio lavori in batch di lunga durata senza checkpoint, prendi in considerazione l'idea di annotare i pod con l'annotazione. `do-not-evict` Disattivando lo sfratto dai pod, stai dicendo a Karpenter che non deve rimuovere volontariamente i nodi che contengono questo pod. Tuttavia, se un `do-not-evict` pod viene aggiunto a un nodo mentre il nodo si sta esaurendo, i pod rimanenti verranno comunque espulsi, ma quel pod bloccherà la terminazione finché non viene rimosso. In entrambi i casi, il nodo verrà isolato per evitare che vengano pianificati lavori aggiuntivi sul nodo. Di seguito è riportato un esempio che mostra come impostare l'annotazione:

```
apiVersion: v1
kind: Pod
metadata:
  name: label-demo
  labels:
    environment: production
  annotations: +
    "karpenter.sh/do-not-evict": "true"
spec:
  containers:

* name: nginx
image: nginx
ports:
 ** containerPort: 80
```

Rimuovi i nodi sottoutilizzati regolando i parametri di Cluster Autoscaler

L'utilizzo dei nodi è definito come la somma delle risorse richieste divisa per la capacità. Per impostazione predefinita, `scale-down-utilization-threshold` è impostato al 50%. Questo parametro può essere utilizzato insieme a and`scale-down-unneeded-time`, che determina per quanto tempo un nodo non deve essere necessario prima che sia idoneo per la riduzione. L'impostazione predefinita è 10 minuti. I pod ancora in esecuzione su un nodo ridimensionato verranno programmati su altri nodi da kube-scheduler. La modifica di queste impostazioni può aiutare a rimuovere i nodi sottoutilizzati, ma è importante testare prima questi valori in modo da non forzare il cluster a ridimensionarsi prematuramente.

È possibile evitare il ridimensionamento assicurandosi che i pod costosi da rimuovere siano protetti da un'etichetta riconosciuta da Cluster Autoscaler. A tale scopo, assicuratevi che i pod costosi da rimuovere abbiano l'annotazione. `cluster-autoscaler.kubernetes.io/safe-to-evict=false` Di seguito è riportato un esempio di yaml per impostare l'annotazione:

```
apiVersion: v1
kind: Pod
metadata:
  name: label-demo
  labels:
    environment: production
  annotations: +
    "cluster-autoscaler.kubernetes.io/safe-to-evict": "false"
spec:
  containers:

* name: nginx
image: nginx
ports:
 ** containerPort: 80
```

Etichetta i nodi con Cluster Autoscaler e Karpenter

I tag delle risorse AWS vengono utilizzati per organizzare le risorse e tenere traccia dei costi AWS a livello dettagliato. Non sono direttamente correlati alle etichette Kubernetes per il monitoraggio dei costi. Si consiglia di iniziare con l'etichettatura delle risorse Kubernetes e utilizzare strumenti come Kubecost per ottenere report sui costi dell'infrastruttura basati sulle etichette Kubernetes su pod, namespace ecc.

I nodi di lavoro devono avere tag per mostrare le informazioni di fatturazione in AWS Cost Explorer. Con Cluster Autoscaler, tagga i nodi di lavoro all'interno di un gruppo di nodi gestito utilizzando il modello di avvio. Per i gruppi di nodi autogestiti, tagga le tue istanze utilizzando il gruppo di scalabilità EC2 automatica. Per le istanze fornite da Karpenter, etichettale usando spec.tags nel modello di nodo.

Cluster multi-tenant

Quando lavori su cluster condivisi da team diversi, potresti non avere visibilità sugli altri carichi di lavoro in esecuzione sullo stesso nodo. Sebbene le richieste di risorse possano contribuire a isolare alcuni problemi legati al problema dei «vicini rumorosi», come la condivisione della CPU, potrebbero non isolare tutti i limiti delle risorse, come l'esaurimento del disco. I/O Non tutte le risorse consumabili in base a un carico di lavoro possono essere isolate o limitate. I carichi di lavoro che consumano risorse condivise a ritmi più elevati rispetto ad altri carichi di lavoro devono essere isolati mediante contaminazioni e tolleranze dei nodi. Un'altra tecnica avanzata per questo tipo di carico di lavoro è il pinning della CPU, che garantisce una CPU esclusiva anziché una CPU condivisa per il contenitore.

L'isolamento dei carichi di lavoro a livello di nodo può essere più costoso, ma potrebbe essere possibile pianificare i BestEffortlavori o sfruttare ulteriori risparmi utilizzando istanze riservate, processori Graviton o Spot.

I cluster condivisi possono anche avere vincoli di risorse a livello di cluster, come l'esaurimento degli indirizzi IP, i limiti dei servizi Kubernetes o le richieste di scalabilità delle API. È necessario consultare la guida alle migliori pratiche di scalabilità per assicurarsi che i cluster evitino questi limiti.

È possibile isolare le risorse a livello di namespace o di provider Karpenter. Le quote di risorse forniscono un modo per impostare limiti al numero di risorse che i carichi di lavoro possono consumare in un namespace. Questa può essere una buona barriera iniziale, ma dovrebbe essere valutata continuamente per assicurarsi che non limiti artificialmente la scalabilità dei carichi di lavoro.

I provisioner di Karpenter possono impostare limiti su alcune delle risorse consumabili in un cluster (ad esempio CPU, GPU), ma sarà necessario configurare le applicazioni tenant per utilizzare il provisioner appropriato. Ciò può impedire a un singolo provider di creare troppi nodi in un cluster, ma deve essere continuamente valutato per assicurarsi che il limite non sia troppo basso e, a sua volta, impedire la scalabilità dei carichi di lavoro.

Scalabilità automatica pianificata

Potrebbe essere necessario ridimensionare i cluster nei fine settimana e fuori orario. Ciò è particolarmente importante per i cluster di test e non di produzione in cui si desidera ridurli a zero

quando non sono in uso. Soluzioni come cluster-turndown possono ridurre a zero le repliche in base a una pianificazione cron. Puoi farlo anche con Karpenter, descritto nel seguente blog AWS.

Ottimizza i tipi di capacità di elaborazione

Dopo aver ottimizzato la capacità totale di elaborazione nel cluster e aver utilizzato il bin packing, è necessario esaminare il tipo di elaborazione a cui è stato assegnato il provisioning nei cluster e il modo in cui si paga per tali risorse. AWS offre piani Compute Savings in grado di ridurre i costi di elaborazione, che classificheremo nei seguenti tipi di capacità:

- Spot
- Savings Plans
- On demand
- Fargate

Ogni tipo di capacità presenta compromessi diversi in termini di sovraccarico di gestione, disponibilità e impegni a lungo termine e dovrai decidere qual è la soluzione più adatta al tuo ambiente. Nessun ambiente deve basarsi su un singolo tipo di capacità ed è possibile combinare più tipi di esecuzione in un unico cluster per ottimizzare requisiti e costi specifici dei carichi di lavoro.

Spot

Il tipo di capacità spot fornisce EC2 istanze dalla capacità inutilizzata in una zona di disponibilità. Spot offre gli sconti più elevati, fino al 90%, ma tali istanze possono essere interrotte quando sono necessarie altrove. Inoltre, potrebbe non essere sempre possibile effettuare il provisioning di nuove istanze Spot e le istanze Spot esistenti possono essere recuperate con un avviso di interruzione di 2 minuti. Se l'applicazione ha un processo di avvio o arresto lungo, le istanze Spot potrebbero non essere l'opzione migliore.

Spot Compute dovrebbe utilizzare un'ampia varietà di tipi di istanze per ridurre la probabilità che non disponga di capacità spot disponibile. Le interruzioni delle istanze devono essere gestite per chiudere i nodi in modo sicuro. I nodi forniti con Karpenter o facenti parte di un Managed Node Group supportano automaticamente le notifiche di interruzione delle istanze. Se utilizzi nodi autogestiti, dovrai eseguire il gestore di terminazione dei nodi separatamente per chiudere correttamente le istanze spot.

È possibile bilanciare istanze spot e on-demand in un unico cluster. Con Karpenter è possibile creare provisioner ponderati per raggiungere un equilibrio tra diversi tipi di capacità. Con Cluster Autoscaler è possibile creare gruppi di nodi misti con istanze spot, on-demand o riservate.

Ecco un esempio di utilizzo di Karpenter per dare priorità alle istanze Spot rispetto alle istanze On-Demand. Quando si crea un provisioner, è possibile specificare Spot, On-Demand o entrambi (come illustrato di seguito). Quando specificate entrambi, e se il pod non specifica esplicitamente se deve utilizzare Spot o On-Demand, Karpenter dà la priorità a Spot quando fornisce a un nodo una strategia di allocazione. price-capacity-optimization

```
apiVersion: karpenter.sh/v1
kind: Provisioner
metadata:
  name: spot-prioritized
spec:
  requirements:
    - key: "karpenter.sh/capacity-type"
      operator: In
        values: ["spot", "on-demand"]
```

Savings Plans, istanze riservate e AWS EDP

Puoi ridurre la spesa di elaborazione utilizzando un piano di risparmio di calcolo. I piani di risparmio offrono prezzi ridotti per un periodo di 1 o 3 anni di utilizzo dell'elaborazione. L'utilizzo può essere applicato alle EC2 istanze in un cluster EKS ma si applica anche a qualsiasi utilizzo di elaborazione come Lambda e Fargate. Con i piani di risparmio puoi ridurre i costi e continuare a scegliere qualsiasi tipo di EC2 istanza durante il periodo di impegno.

Il piano Compute Savings può ridurre i EC2 costi fino al 66% senza richiedere impegni sui tipi di istanze, sulle famiglie o sulle aree geografiche che desideri utilizzare. I risparmi vengono applicati automaticamente alle istanze man mano che le utilizzi.

EC2 Instance Savings Plans offre risparmi fino al 72% sull'elaborazione con un impegno di utilizzo in una regione e una EC2 famiglia specifiche, ad esempio istanze della famiglia C. È possibile spostare l'utilizzo in qualsiasi zona della regione, utilizzare qualsiasi generazione della famiglia di istanze, ad esempio c5 o c6, e utilizzare istanze di qualsiasi dimensione all'interno della famiglia. Lo sconto verrà applicato automaticamente a qualsiasi istanza del tuo account che soddisfi i criteri del piano di risparmio.

Le istanze riservate sono simili a EC2 Instance Savings Plan ma garantiscono anche la capacità in una zona o regione di disponibilità e riducono i costi, fino al 72%, rispetto alle istanze on demand. Una volta calcolata la capacità riservata necessaria, puoi selezionare per quanto tempo desideri prenotarle (1 anno o 3 anni). Gli sconti verranno applicati automaticamente man mano che eseguirai tali EC2 istanze nel tuo account.

I clienti hanno anche la possibilità di sottoscrivere un Enterprise Agreement con AWS. Gli accordi aziendali consentono ai clienti di personalizzare i contratti per adattarli alle loro esigenze. I clienti possono usufruire di sconti sui prezzi basati su AWS EDP (Enterprise Discount Program). Per ulteriori informazioni sugli accordi aziendali, contatta il tuo rappresentante di vendita AWS.

On demand

Rispetto ai piani di risparmio, EC2 le istanze On-Demand offrono il vantaggio di una disponibilità senza interruzioni, rispetto alle istanze spot, e di nessun impegno a lungo termine. Se stai cercando di ridurre i costi in un cluster, dovresti ridurre l'utilizzo di istanze on-demand. EC2

Dopo aver ottimizzato i requisiti del carico di lavoro, dovresti essere in grado di calcolare la capacità minima e massima per i tuoi cluster. Questo numero può cambiare nel tempo, ma raramente diminuisce. Prendi in considerazione l'idea di utilizzare un Savings Plan per tutto ciò che è al di sotto del minimo e una capacità spot che non influisca sulla disponibilità dell'applicazione. Qualsiasi altra cosa che potrebbe non essere utilizzata continuamente o che è necessaria per la disponibilità può essere utilizzata su richiesta.

Come indicato in questa sezione, il modo migliore per ridurre l'utilizzo è consumare meno risorse e utilizzare le risorse fornite nella massima misura possibile. Con Cluster Autoscaler è possibile rimuovere i nodi sottoutilizzati con l'impostazione. `scale-down-utilization-threshold` Con Karpenter si consiglia di abilitare il consolidamento.

Per identificare manualmente i tipi di EC2 istanze che possono essere utilizzati con i carichi di lavoro, è necessario utilizzare ec2-instance-selector, che può mostrare le istanze disponibili in ogni regione e le istanze compatibili con EKS. Esempio di utilizzo per istanze con architettura di processo x86, 4 GB di memoria, 2 v CPUs e disponibili nella regione us-east-1.

```
ec2-instance-selector --memory 4 --vcpus 2 --cpu-architecture x86_64 \
  -r us-east-1 --service eks
c5.large
c5a.large
c5ad.large
```

```
c5d.large
c6a.large
c6i.large
t2.medium
t3.medium
t3a.medium
```

Per gli ambienti non di produzione, è possibile ridimensionare automaticamente i cluster durante le ore non utilizzate, come la notte e i fine settimana. Il progetto kubecost cluster-turndown è un esempio di controller in grado di ridimensionare automaticamente il cluster in base a una pianificazione prestabilita.

Calcolo Fargate

Fargate Compute è un'opzione di elaborazione completamente gestita per i cluster EKS. Fornisce l'isolamento dei pod pianificando un pod per nodo in un cluster Kubernetes. Consente di dimensionare i nodi di elaborazione in base ai requisiti di CPU e RAM del carico di lavoro per controllare in modo rigoroso l'utilizzo del carico di lavoro in un cluster.

Fargate è in grado di scalare carichi di lavoro fino a 0,25 vCPU con 0,5 GB di memoria e fino a 16 vCPU con 120 GB di memoria. Esistono dei limiti al numero di varianti di dimensione dei pod disponibili e dovrai capire come il tuo carico di lavoro si adatta meglio a una configurazione Fargate. Ad esempio, se il carico di lavoro richiede 1 vCPU con 0,5 GB di memoria, il pod Fargate più piccolo sarà 1 vCPU con 2 GB di memoria.

Sebbene Fargate offra molti vantaggi, come l'assenza di gestione delle EC2 istanze o del sistema operativo, può richiedere una maggiore capacità di elaborazione rispetto alle EC2 istanze tradizionali, poiché ogni pod distribuito è isolato come nodo separato nel cluster. Ciò richiede una maggiore duplicazione per cose come Kubelet, gli agenti di registrazione e tutti quelli che normalmente si distribuiscono su DaemonSets un nodo. DaemonSets non sono supportati in Fargate e dovranno essere convertiti in pod «`sidecars"` ed eseguiti insieme all'applicazione.

Fargate non può trarre vantaggio dal bin packing o dall'overprovisioning della CPU perché il limite del carico di lavoro è un nodo che non è espandibile o condivisibile tra carichi di lavoro. Fargate consente di risparmiare tempo nella gestione delle EC2 istanze, il che di per sé ha un costo, ma i costi di CPU e memoria possono essere più costosi rispetto ad altri tipi di EC2 capacità. I pod Fargate possono sfruttare il piano di risparmio di calcolo per ridurre i costi on-demand.

Ottimizza l'utilizzo dell'elaborazione

Un altro modo per risparmiare sull'infrastruttura di elaborazione consiste nell'utilizzare un'elaborazione più efficiente per il carico di lavoro. Questo può derivare da elaborazioni generiche più performanti come i processori Graviton, che sono fino al 20% più economici e il 60% più efficienti dal punto di vista energetico rispetto all'x86, o da acceleratori specifici per carichi di lavoro come and. GPUs FPGAs Dovrai creare contenitori che possano funzionare sull'architettura ARM e configurare nodi con gli acceleratori giusti per i tuoi carichi di lavoro.

EKS ha la capacità di eseguire cluster con architettura mista (ad esempio amd64 e arm64) e se i contenitori sono compilati per più architetture, puoi sfruttare i processori Graviton con Karpenter, abilitando entrambe le architetture nel tuo provisioner. Per mantenere costanti le prestazioni, tuttavia, si consiglia di mantenere ogni carico di lavoro su un'unica architettura di elaborazione e di utilizzare un'architettura diversa solo se non è disponibile capacità aggiuntiva.

I provider possono essere configurati con più architetture e i carichi di lavoro possono anche richiedere architetture specifiche nelle specifiche del carico di lavoro.

```
apiVersion: karpenter.sh/v1
kind: Provisioner
metadata:
  name: default
spec:
  requirements:
  - key: "kubernetes.io/arch"
    operator: In
    values: ["arm64", "amd64"]
```

Con Cluster Autoscaler dovrai creare un gruppo di nodi per le istanze Graviton e impostare le tolleranze dei nodi sul tuo carico di lavoro per utilizzare la nuova capacità.

GPUs e FPGAs può aumentare notevolmente le prestazioni del carico di lavoro, ma il carico di lavoro dovrà essere ottimizzato per utilizzare l'acceleratore. Molti tipi di carico di lavoro per l'apprendimento automatico e l'intelligenza artificiale possono essere utilizzati GPUs per l'elaborazione e le istanze possono essere aggiunte a un cluster e montate in un carico di lavoro utilizzando richieste di risorse.

```
spec:
  template:
    spec:
    - containers:
```

```
    ...
    resources:
        limits:
          nvidia.com/gpu: "1"
```

Alcuni hardware GPU possono essere condivisi tra più carichi di lavoro in modo da poter effettuare il provisioning e l'utilizzo di una singola GPU. Per scoprire come configurare la condivisione della GPU del carico di lavoro, consulta il plug-in del dispositivo GPU virtuale per ulteriori informazioni. Puoi anche fare riferimento ai seguenti blog:

- Condivisione di GPU su Amazon EKS con istanze NVIDIA time-slicing e accelerate EC2
- Massimizzazione dell'utilizzo della GPU con la GPU multi-istanza (MIG) di NVIDIA su Amazon EKS: esecuzione di più pod per GPU per prestazioni migliorate

Ottimizzazione dei costi - Rete

Progettare sistemi per l'alta disponibilità (HA) è una best practice per raggiungere la resilienza e la tolleranza ai guasti. In pratica, ciò significa distribuire i carichi di lavoro e l'infrastruttura sottostante su più zone di disponibilità (AZs) in una determinata regione AWS. Garantire che queste caratteristiche siano presenti per il tuo ambiente Amazon EKS migliorerà l'affidabilità complessiva del tuo sistema. Inoltre, gli ambienti EKS saranno probabilmente composti anche da una varietà di costrutti (ad esempio VPCs), componenti (ad esempio) e integrazioni (ad esempio ECR e altri registri di contenitori ELBs).

La combinazione di sistemi ad alta disponibilità e altri componenti specifici dei casi d'uso può svolgere un ruolo significativo nel modo in cui i dati vengono trasferiti ed elaborati. Ciò avrà a sua volta un impatto sui costi sostenuti a causa del trasferimento e dell'elaborazione dei dati.

Le pratiche descritte di seguito vi aiuteranno a progettare e ottimizzare gli ambienti EKS al fine di ottenere un rapporto costo-efficacia per diversi domini e casi d'uso.

Comunicazione da Pod a Pod

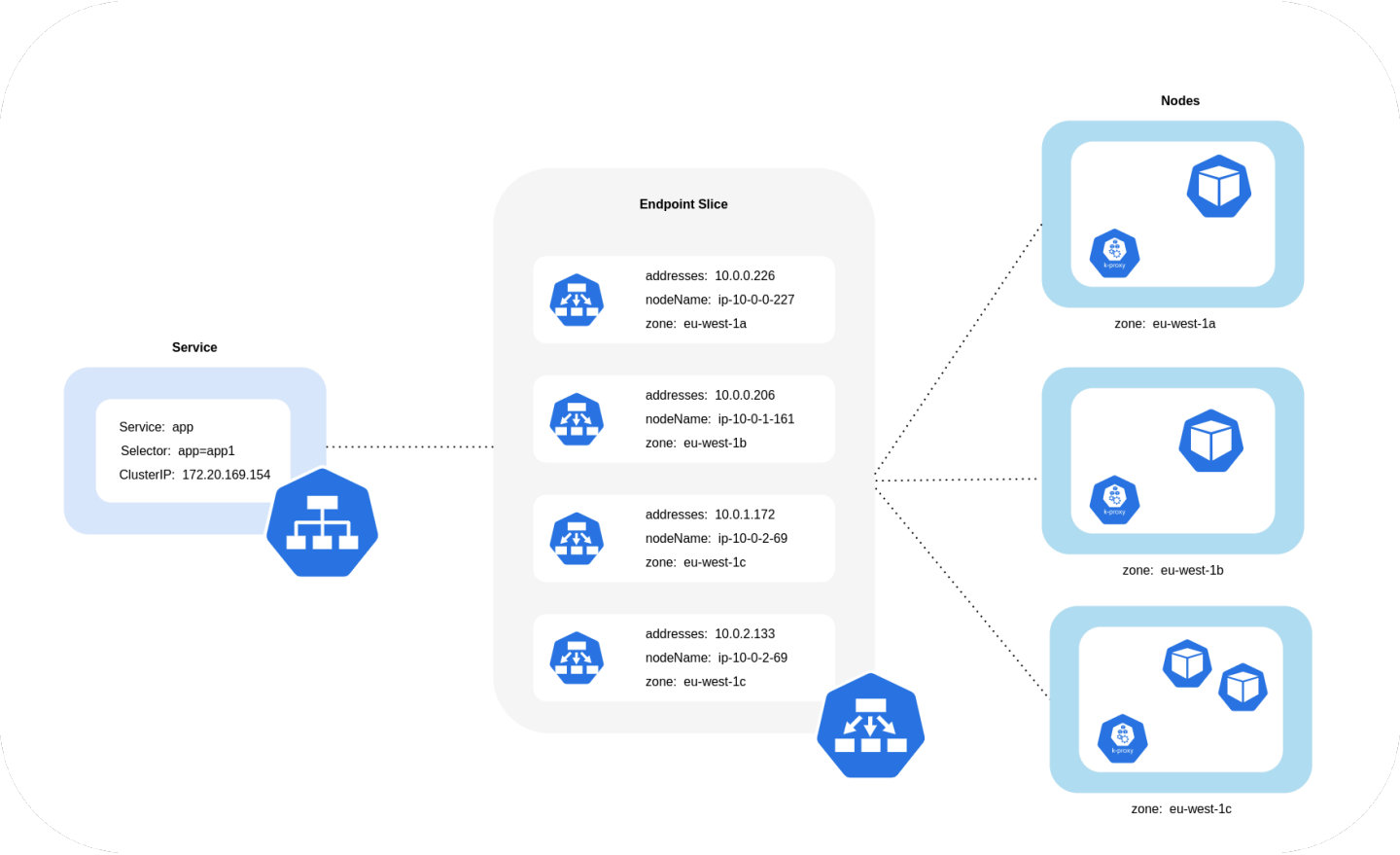
A seconda della configurazione, la comunicazione di rete e il trasferimento di dati tra i pod possono avere un impatto significativo sul costo complessivo di esecuzione dei carichi di lavoro Amazon EKS. Questa sezione tratterà diversi concetti e approcci per mitigare i costi legati alla comunicazione tra pod, considerando al contempo le architetture ad alta disponibilità (HA), le prestazioni delle applicazioni e la resilienza.

Limitazione del traffico verso una zona di disponibilità

Il progetto Kubernetes ha iniziato fin dall'inizio a sviluppare costrutti compatibili con la topologia, tra cui etichette come kubernetes. io/hostname, topology.kubernetes.io/region, and topology.kubernetes.io/zoneassegnati ai nodi per abilitare funzionalità come la distribuzione del carico di lavoro tra domini di errore e fornitori di volumi con riconoscimento della topologia. Dopo la laurea in Kubernetes 1.17, le etichette sono state utilizzate anche per abilitare funzionalità di routing basate sulla topologia per la comunicazione da Pod a Pod.

Di seguito sono riportate alcune strategie su come controllare la quantità di traffico Cross-AZ tra i Pod del cluster EKS per ridurre i costi e minimizzare la latenza.

Se desideri una visibilità granulare sulla quantità di traffico Cross-AZ tra i Pod del tuo cluster (ad esempio la quantità di dati trasferiti in byte), consulta questo post.



Come illustrato nel diagramma precedente, i Servizi sono lo strato di astrazione stabile della rete che riceve il traffico destinato ai tuoi Pod. Quando viene creato un servizio, ne vengono creati più di uno. EndpointSlices Ciascuno EndpointSlice ha un elenco di endpoint contenente un sottoinsieme di indirizzi Pod insieme ai nodi su cui sono in esecuzione e qualsiasi altra informazione sulla topologia.

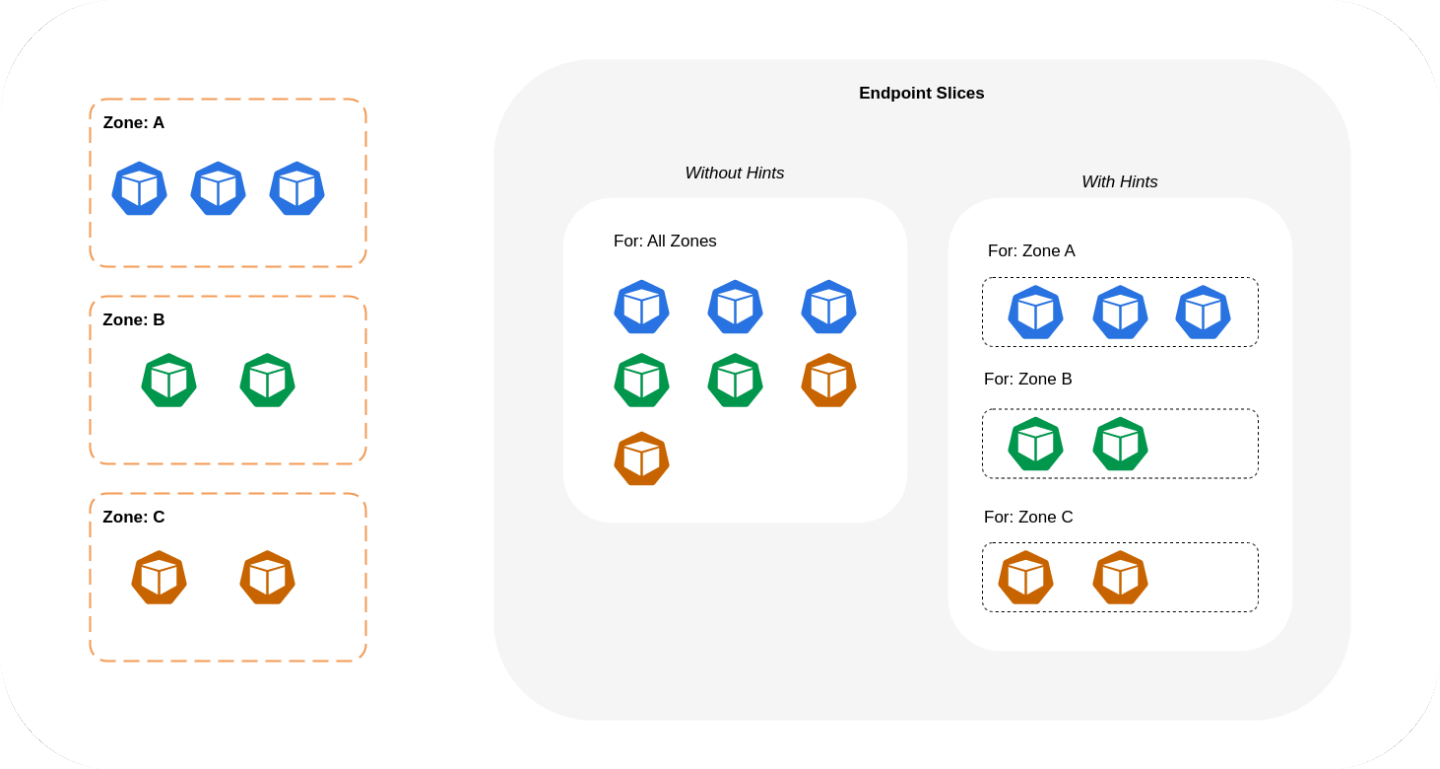
Quando si utilizza Amazon VPC CNI, kube-proxy, un daemset in esecuzione su ogni nodo, mantiene le regole di rete per consentire la comunicazione Pod e il rilevamento dei servizi (un'alternativa basata su EBPF CNIs potrebbe non utilizzare kube-proxy ma fornire un comportamento equivalente). Svolge il ruolo di routing interno, ma lo fa in base a ciò che consuma dalla creazione. EndpointSlices

Su EKS, kube-proxy utilizza principalmente le regole NAT di iptables (o IPVS, in alternativa) per la distribuzione del traffico su tutti i pod del cluster, NFTablesindipendentemente dal nodo o dalla posizione in cui sono posizionati. Questa distribuzione predefinita può portare al routing del traffico Cross-AZ, causando potenzialmente un aumento della latenza per le applicazioni sensibili e dei costi di trasferimento dati tra le AZ nelle implementazioni di grandi dimensioni.

Utilizzo del Topology Aware Routing (precedentemente noto come Topology Aware Hints)

Quando il routing con riconoscimento della topologia è abilitato e implementato su un servizio Kubernetes, il EndpointSlice controller allocherà proporzionalmente gli endpoint alle diverse zone su cui è distribuito il cluster. Per ciascuno di questi endpoint, il EndpointSlice controller imposterà anche un suggerimento per la zona. I suggerimenti descrivono per quale zona un endpoint deve servire il traffico. `kube-proxy`indirizzerà quindi il traffico da una zona a un endpoint in base ai suggerimenti applicati.

Il diagramma seguente mostra come EndpointSlices i suggerimenti siano organizzati in modo tale da `kube-proxy` poter sapere a quale destinazione devono andare in base al punto di origine zonale. Senza i suggerimenti, non esiste tale allocazione o organizzazione e il traffico verrà inoltrato a diverse destinazioni zonali indipendentemente dalla sua provenienza.



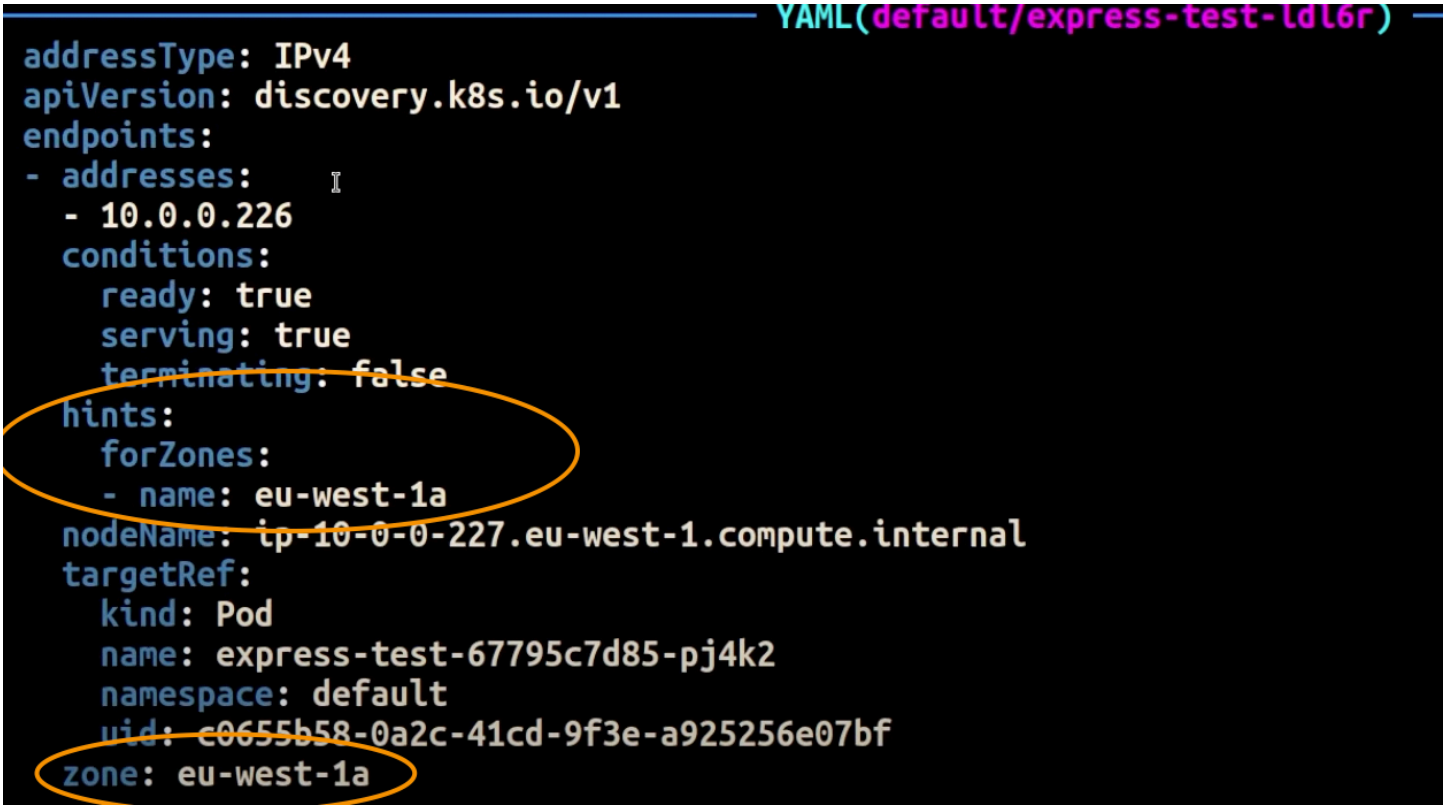
In alcuni casi, il EndpointSlice controller può applicare un suggerimento per una zona diversa, il che significa che l'endpoint potrebbe finire per servire il traffico proveniente da una zona diversa. La ragione di ciò è cercare di mantenere una distribuzione uniforme del traffico tra gli endpoint in zone diverse.

Di seguito è riportato un frammento di codice su come abilitare il routing con riconoscimento della topologia per un servizio.

```
apiVersion: v1
kind: Service
metadata:
  name: orders-service
  namespace: ecommerce
  annotations:
    service.kubernetes.io/topology-mode: Auto
spec:
  selector:
    app: orders
  type: ClusterIP
  ports:

* protocol: TCP
```

```
port: 3003
targetPort: 3003
```

La schermata seguente mostra il risultato ottenuto dal EndpointSlice controller che ha applicato con successo un suggerimento a un endpoint per una replica Pod in esecuzione nell'AZ. `eu-west-1a`



> **Note**
>
> È importante notare che il routing con riconoscimento della topologia è ancora in versione beta. Questa funzionalità offre prestazioni più prevedibili con carichi di lavoro distribuiti uniformemente sulla topologia del cluster, poiché il controller alloca gli endpoint in modo proporzionale tra le zone, ma può saltare l'assegnazione dei suggerimenti quando le risorse dei nodi in una zona sono troppo squilibrate per evitare un sovraccarico eccessivo. Pertanto, si consiglia vivamente di utilizzarla insieme a vincoli di pianificazione che aumentano la disponibilità di un'applicazione come i vincoli di diffusione della topologia dei pod. Tieni presente che i suggerimenti potrebbero non essere assegnati anche quando la capacità oscilla tra le zone, ad esempio quando si utilizzano istanze Spot Amazon EC2, poiché le interruzioni o le sostituzioni non vengono rilevate in tempo reale durante il calcolo della distribuzione proporzionale.

Utilizzo della distribuzione del traffico

Introdotto in Kubernetes 1.30 e reso disponibile a livello generale nella versione 1.33, Traffic Distribution offre un'alternativa più semplice al Topology Aware Routing per la preferenza del traffico nella stessa zona. Sebbene Topology Aware Routing tenti di utilizzare un approccio intelligente al routing del traffico per evitare il sovraccarico degli endpoint, ha determinato un comportamento imprevedibile. La distribuzione del traffico dà invece priorità alla prevedibilità. L' PreferClose opzione ordina a kube-proxy di creare regole che indirizzino innanzitutto il traffico verso gli endpoint della stessa zona in base al suggerimento zonale impostato dal Controller. EndpointSlice Quando non sono disponibili endpoint della stessa zona, si torna alla distribuzione del traffico su qualsiasi endpoint del cluster per il Servizio. Questa funzionalità è progettata per carichi di lavoro che accettano il compromesso di ottimizzare per la prossimità piuttosto che tentare di distribuire uniformemente il carico fornita da Topology Aware Routing.

Di seguito è riportato un frammento di codice su come abilitare la distribuzione del traffico per un servizio.

```
apiVersion: v1
kind: Service
metadata:
  name: orders-service
  namespace: ecommerce
spec:
  trafficDistribution: PreferClose
  selector:
    app: orders
  type: ClusterIP
  ports:

* protocol: TCP
port: 3003
targetPort: 3003
```

Quando si abilita la distribuzione del traffico, emerge una sfida comune: gli endpoint all'interno di una singola AZ possono sovraccaricarsi se la maggior parte del traffico proviene dalla stessa zona. Questo sovraccarico può creare problemi significativi:

- Un singolo Horizontal Pod Autoscaler (HPA) che gestisce un'implementazione Multi-AZ può rispondere scalando i pod tra diversi. AZs Tuttavia, questa azione non riesce a far fronte in modo efficace all'aumento del carico nella zona interessata.

- Questa situazione a sua volta può portare a un'inefficienza delle risorse. Quando gli autoscaler dei cluster come Karpenter rilevano lo scale-out del pod tra diversi nodi AZs, possono fornire nodi aggiuntivi nell'area inalterata, con conseguente allocazione non necessaria delle risorse. AZs

Per superare questa sfida:

- Crea implementazioni separate per zona, con una scalabilità indipendente HPAs l'una dall'altra.
- Sfrutta i vincoli di diffusione della topologia per garantire la distribuzione del carico di lavoro nel cluster, il che aiuta a prevenire il sovraccarico degli endpoint nelle zone ad alto traffico.

Utilizzo di Autoscaler: assegna nodi a una zona di disponibilità specifica

Consigliamo vivamente di eseguire i carichi di lavoro in ambienti ad alta disponibilità su più piattaforme. AZs Ciò migliora l'affidabilità delle applicazioni, soprattutto quando si verifica un incidente o un problema con una zona di disponibilità. Se siete disposti a sacrificare l'affidabilità per ridurre i costi relativi alla rete, potete limitare i nodi a una singola AZ.

Per far funzionare tutti i tuoi Pod nella stessa AZ, esegui il provisioning dei nodi di lavoro nella stessa AZ o pianifica i Pod sui nodi di lavoro in esecuzione sulla stessa AZ. Per effettuare il provisioning dei nodi all'interno di una singola AZ, definisci un gruppo di nodi con sottoreti appartenenti alla stessa AZ con Cluster Autoscaler (CA). Per Karpenter, usa `topology.kubernetes.io/zone` e specifica l'AZ in cui desideri creare i nodi di lavoro. Ad esempio, lo snippet di Karpenter provisioner riportato di seguito esegue il provisioning dei nodi nella zona AZ us-west-2a.

Karpenter

```
apiVersion: karpenter.sh/v1
kind: Provisioner
metadata:
name: single-az
spec:
  requirements:

* key: "topology.kubernetes.io/zone"`
operator: In
values: ["us-west-2a"]
```

Cluster Autoscaler (CA)

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata:
  name: my-ca-cluster
  region: us-east-1
  version: "1.21"
availabilityZones:

* us-east-1a
managedNodeGroups:
* name: managed-nodes
labels:
  role: managed-nodes
instanceType: t3.medium
minSize: 1
maxSize: 10
desiredCapacity: 1
...
```

Utilizzo di Pod Assignment e Node Affinity

In alternativa, se hai nodi di lavoro in esecuzione in più nodi AZs, ogni nodo avrebbe l'etichetta topology.kubernetes.io/zone con il valore della relativa AZ (come us-west-2a o us-west-2b). Puoi utilizzare o programmare i Pod per i nodi in una singola AZ. `nodeSelector` `nodeAffinity` Ad esempio, il seguente file manifest pianificherà il Pod all'interno di un nodo in esecuzione in AZ us-west-2a.

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    env: test
spec:
  nodeSelector:
    topology.kubernetes.io/zone: us-west-2a
  containers:

* name: nginx
image: nginx
imagePullPolicy: IfNotPresent
```

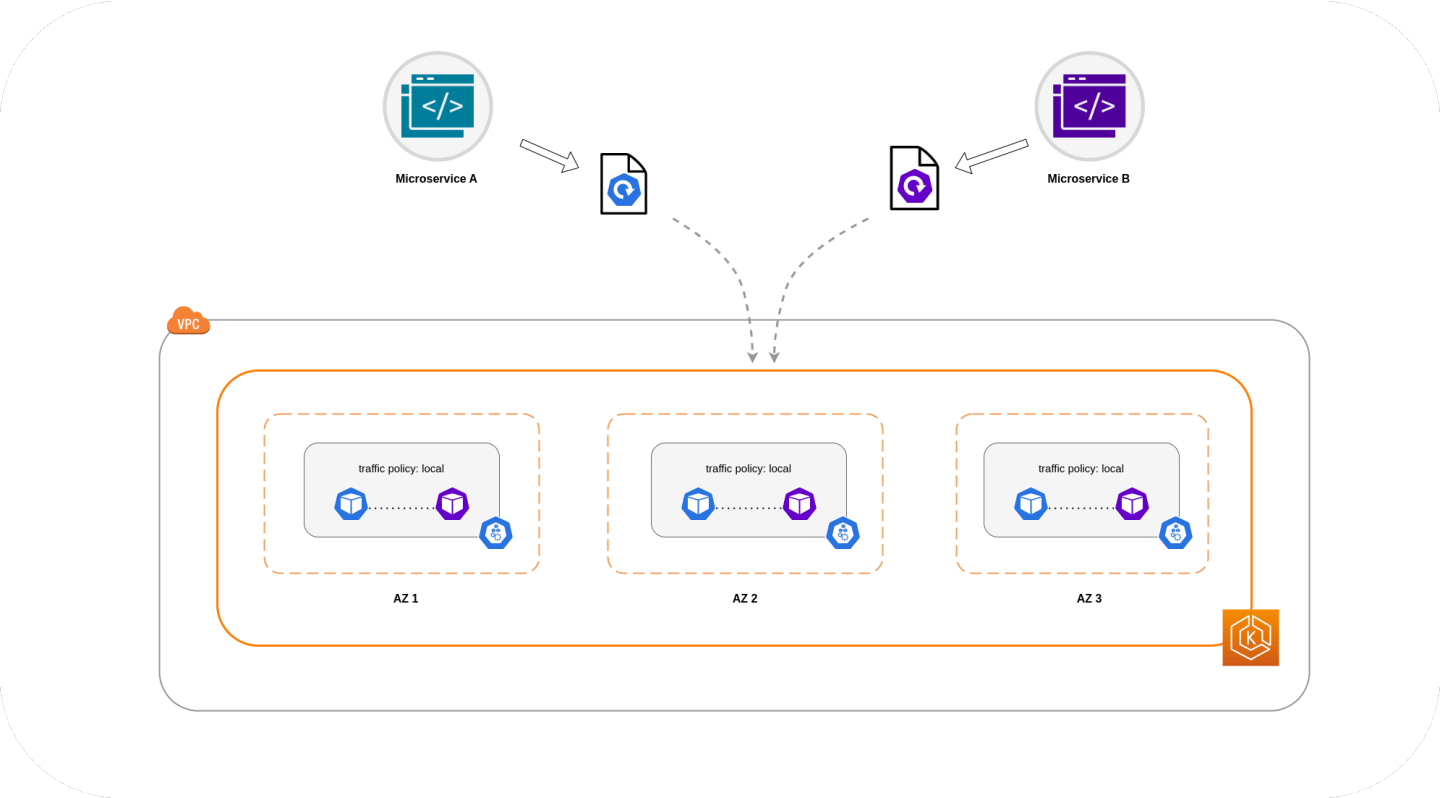
Limitazione del traffico verso un nodo

Ci sono casi in cui limitare il traffico a livello zonale non è sufficiente. Oltre alla riduzione dei costi, potrebbe essere necessario ridurre la latenza di rete tra determinate applicazioni che hanno frequenti intercomunicazioni. Per ottenere prestazioni di rete ottimali e ridurre i costi, è necessario un modo per limitare il traffico a un nodo specifico. Ad esempio, il microservizio A dovrebbe sempre comunicare con il microservizio B sul nodo 1, anche nelle configurazioni ad alta disponibilità (HA). La comunicazione tra il Microservizio A sul Nodo 1 e il Microservizio B sul Nodo 2 può avere un impatto negativo sulle prestazioni desiderate per applicazioni di questo tipo, specialmente se il Nodo 2 si trova in una zona di zona completamente separata.

Utilizzo della politica sul traffico interno del servizio

Per limitare il traffico di rete Pod a un nodo, puoi utilizzare la politica sul traffico interno del Servizio. Per impostazione predefinita, il traffico inviato al servizio di un carico di lavoro verrà distribuito in modo casuale tra i diversi endpoint generati. Quindi, in un'architettura HA, ciò significa che il traffico proveniente dal Microservizio A potrebbe essere indirizzato a qualsiasi replica del Microservizio B su un determinato nodo tra i diversi nodi. AZs Tuttavia, con la politica sul traffico interno del Servizio impostata su`Local`, il traffico sarà limitato agli endpoint sul nodo da cui proviene il traffico. Questa politica impone l'uso esclusivo degli endpoint locali del nodo. Di conseguenza, i costi relativi al traffico di rete per quel carico di lavoro saranno inferiori rispetto a quelli derivanti dalla distribuzione a livello di cluster. Inoltre, la latenza sarà inferiore, il che renderà l'applicazione più performante.

> **Note**
>
> È importante notare che questa funzionalità non può essere combinata con il routing con riconoscimento della topologia in Kubernetes.



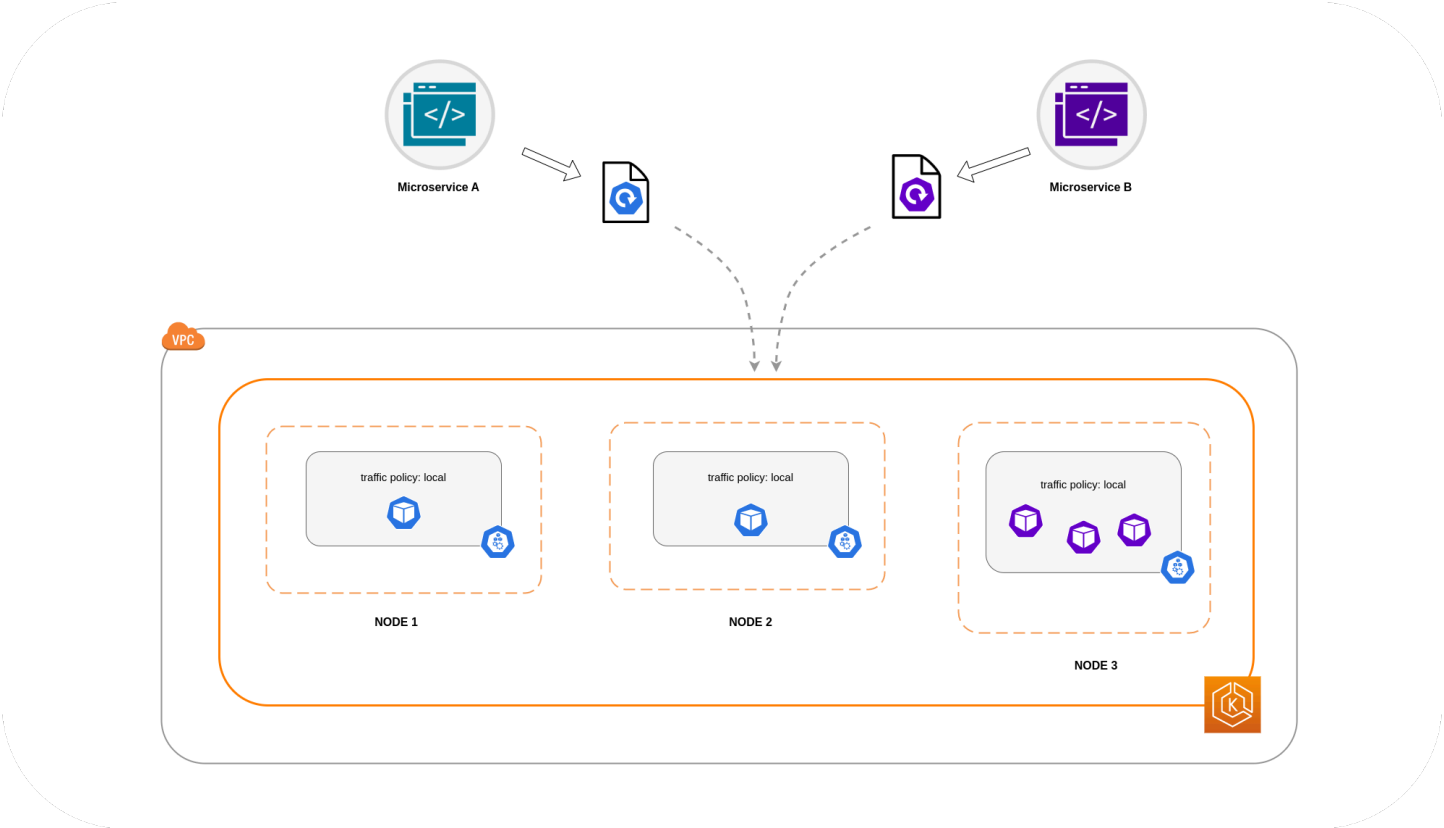
Di seguito è riportato un frammento di codice su come impostare la politica interna sul traffico per un servizio.

```
apiVersion: v1
kind: Service
metadata:
  name: orders-service
  namespace: ecommerce
spec:
  selector:
    app: orders
  type: ClusterIP
  ports:

* protocol: TCP
port: 3003
targetPort: 3003
  internalTrafficPolicy: Local
```

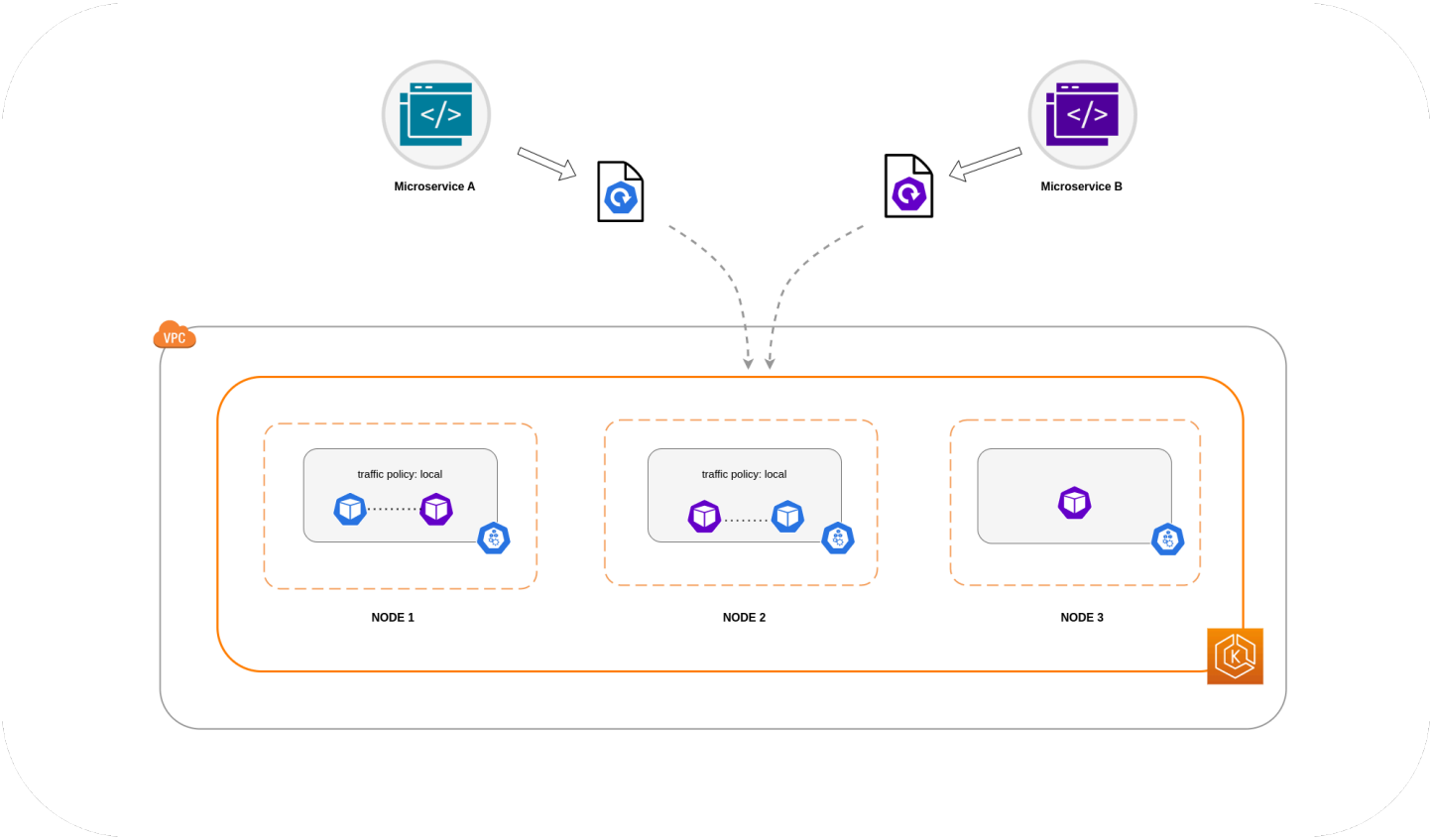
Per evitare comportamenti imprevisti dell'applicazione dovuti a cali di traffico, è necessario prendere in considerazione i seguenti approcci:

- Eseguite un numero sufficiente di repliche per ciascuno dei Pod comunicanti
- Ottieni una distribuzione relativamente uniforme dei Pod utilizzando i vincoli di diffusione della topologia
- Utilizza le regole di affinità dei pod per la co-localizzazione dei Pod comunicanti

In questo esempio, avete 2 repliche del Microservizio A e 3 repliche del Microservizio B. Se il Microservizio A ha le sue repliche distribuite tra i Nodi 1 e 2 e il Microservizio B ha tutte e 3 le sue repliche sul Nodo 3, non saranno in grado di comunicare a causa della politica interna sul traffico. `Local` Quando non sono disponibili endpoint locali a livello di nodo, il traffico diminuisce.



Se Microservice B ha 2 delle sue 3 repliche sui nodi 1 e 2, allora ci sarà comunicazione tra le applicazioni peer. Tuttavia, avresti comunque una replica isolata del Microservice B senza alcuna replica peer con cui comunicare.



> **Note**
>
> In alcuni scenari, una replica isolata come quella illustrata nel diagramma precedente potrebbe non essere motivo di preoccupazione se serve ancora a uno scopo (ad esempio soddisfare le richieste provenienti dal traffico esterno in entrata).

Utilizzo della policy sul traffico interno del servizio con vincoli di diffusione della topologia

L'utilizzo della politica interna sul traffico insieme ai vincoli di diffusione della topologia può essere utile per garantire il giusto numero di repliche per la comunicazione dei microservizi su nodi diversi.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: express-test
spec:
  replicas: 6
  selector:
    matchLabels:
```

```
      app: express-test
  template:
    metadata:
      labels:
        app: express-test
        tier: backend
    spec:
      topologySpreadConstraints:
      - maxSkew: 1
        topologyKey: "topology.kubernetes.io/zone"
        whenUnsatisfiable: ScheduleAnyway
        labelSelector:
          matchLabels:
            app: express-test
```

Utilizzo della policy sul traffico interno del servizio con Pod Affinity Rules

Un altro approccio consiste nell'utilizzare le regole di affinità di Pod quando si utilizza la politica del traffico interno del Servizio. Con Pod affinity, puoi influenzare lo scheduler affinché collochino determinati Pod a causa della loro frequente comunicazione. Applicando rigidi vincoli di pianificazione (`requiredDuringSchedulingIgnoredDuringExecution`) su determinati Pod, otterrete risultati migliori per la co-locazione dei Pod quando lo Scheduler posiziona i Pod sui nodi.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: graphql
  namespace: ecommerce
  labels:
    app.kubernetes.io/version: "0.1.6"
    ...
    spec:
      serviceAccountName: graphql-service-account
      affinity:
        podAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
              - key: app
                operator: In
                values:
                - orders
```

```
            topologyKey: "kubernetes.io/hostname"
```

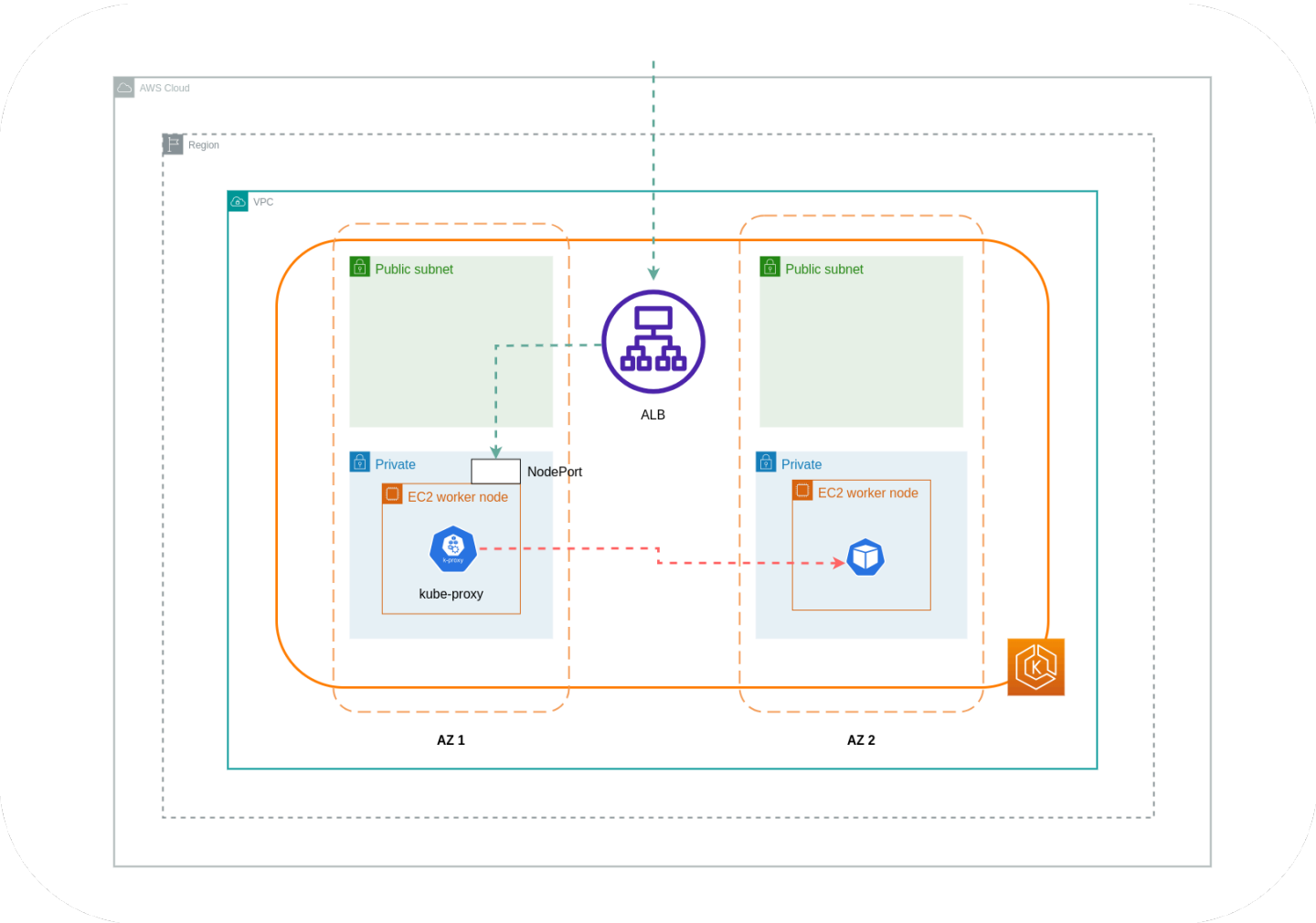
Comunicazione da Load Balancer a Pod

I carichi di lavoro EKS sono in genere gestiti da un sistema di bilanciamento del carico che distribuisce il traffico ai Pod pertinenti del cluster EKS. L'architettura può comprendere bilanciatori di carico interni ed esterni. and/or A seconda dell'architettura e delle configurazioni del traffico di rete, la comunicazione tra i sistemi di bilanciamento del carico e i Pod può contribuire in modo significativo ai costi di trasferimento dei dati.

Puoi utilizzare AWS Load Balancer Controller per gestire automaticamente la creazione di risorse ELB (ALB e NLB). I costi di trasferimento dei dati sostenuti in tali configurazioni dipenderanno dal percorso seguito dal traffico di rete. Il controller AWS Load Balancer supporta due modalità di traffico di rete, la modalità istanza e la modalità ip.

Quando si utilizza la modalità di istanza, NodePort verrà aperto un nodo su ogni nodo del cluster EKS. Il load balancer eseguirà quindi il proxy del traffico in modo uniforme tra i nodi. Se su un nodo è in esecuzione il Pod di destinazione, non verranno sostenuti costi di trasferimento dei dati. Tuttavia, se il Pod di destinazione si trova su un nodo separato e in una AZ diversa da quella che NodePort riceve il traffico, ci sarà un ulteriore hop di rete dal kube-proxy al Pod di destinazione. In tale scenario, verranno applicati costi di trasferimento dati Cross-AZ. A causa della distribuzione uniforme del traffico tra i nodi, è molto probabile che vengano applicati costi aggiuntivi per il trasferimento dei dati associati agli spostamenti del traffico di rete tra zone dai kube-proxy ai relativi Pod di destinazione.

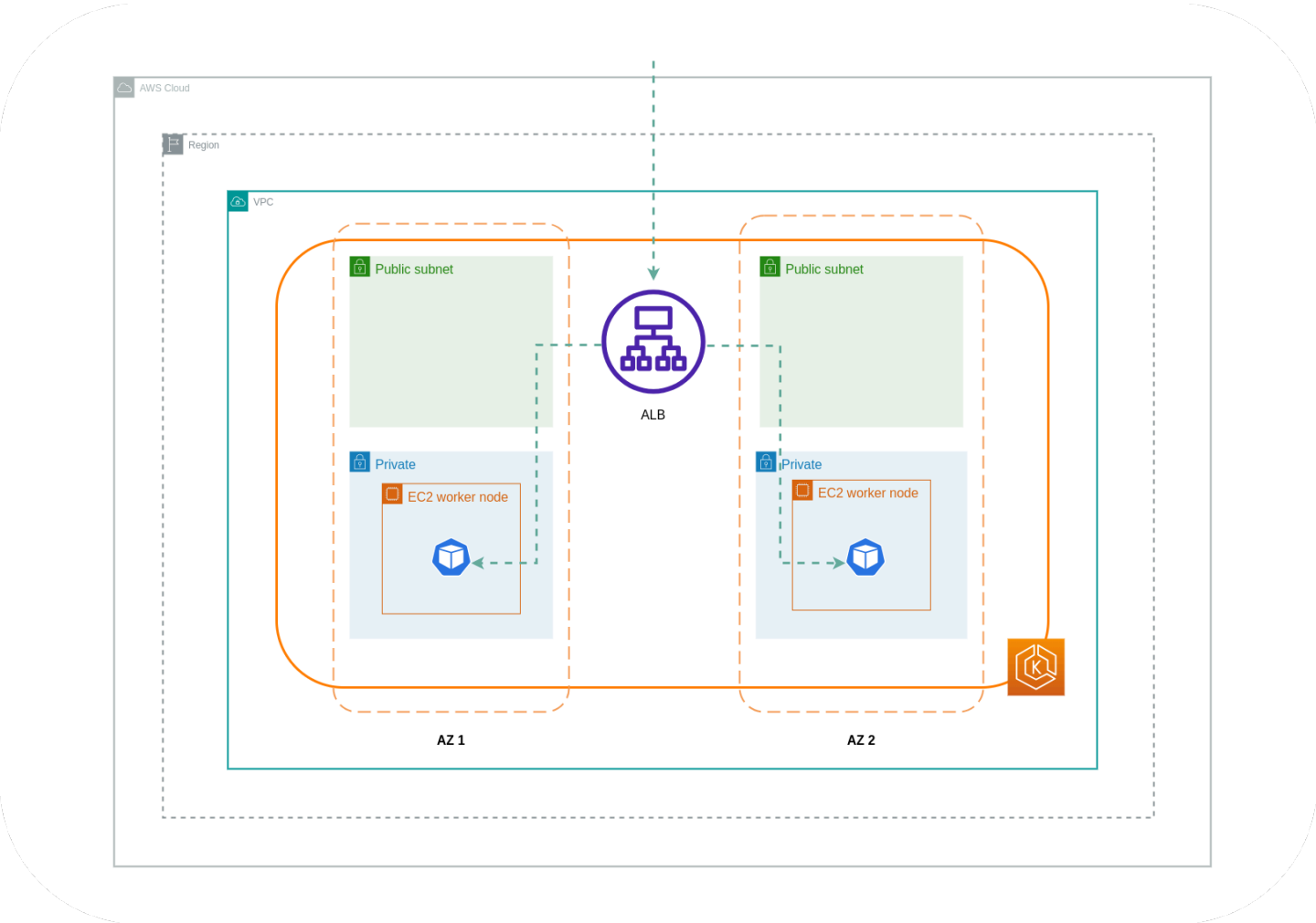
Il diagramma seguente mostra un percorso di rete per il traffico che fluisce dal sistema di bilanciamento del carico al Pod di destinazione e successivamente dal Pod di destinazione su un NodePort nodo separato in una AZ `kube-proxy` diversa. Questo è un esempio dell'impostazione della modalità di istanza.



Quando si utilizza la modalità ip, il traffico di rete viene inoltrato dal load balancer direttamente al Pod di destinazione. Di conseguenza, questo approccio non prevede costi per il trasferimento dei dati.

> **Note**
>
> Si consiglia di impostare il sistema di bilanciamento del carico in modalità traffico IP per ridurre i costi di trasferimento dei dati. Per questa configurazione, è anche importante assicurarsi che il sistema di bilanciamento del carico sia distribuito su tutte le sottoreti del VPC.

Il diagramma seguente mostra i percorsi di rete per il traffico che scorre dal sistema di bilanciamento del carico ai Pods in modalità IP di rete.



Trasferimento dati dal registro dei contenitori

Amazon ECR

Il trasferimento dei dati nel registro privato di Amazon ECR è gratuito. Il trasferimento di dati all'interno della regione non comporta alcun costo, ma il trasferimento di dati verso Internet e tra le regioni verrà addebitato alle tariffe di trasferimento dati Internet su entrambi i lati del trasferimento.

È necessario utilizzare la funzionalità di replica delle immagini ECRs integrata per replicare le immagini dei container pertinenti nella stessa regione dei carichi di lavoro. In questo modo la replica verrebbe addebitata una sola volta e tutte le immagini recuperate dalla stessa regione (intra-regione) sarebbero gratuite.

È possibile ridurre ulteriormente i costi di trasferimento dei dati associati all'estrazione di immagini da ECR (trasferimento dati in uscita) utilizzando Interface VPC Endpoints per connettersi agli archivi ECR locali. L'approccio alternativo di connessione all'endpoint AWS pubblico di ECR (tramite un

gateway NAT e un Internet Gateway) comporterà costi di elaborazione e trasferimento dei dati più elevati. La prossima sezione tratterà la riduzione dei costi di trasferimento dei dati tra i carichi di lavoro e i servizi AWS in modo più dettagliato.

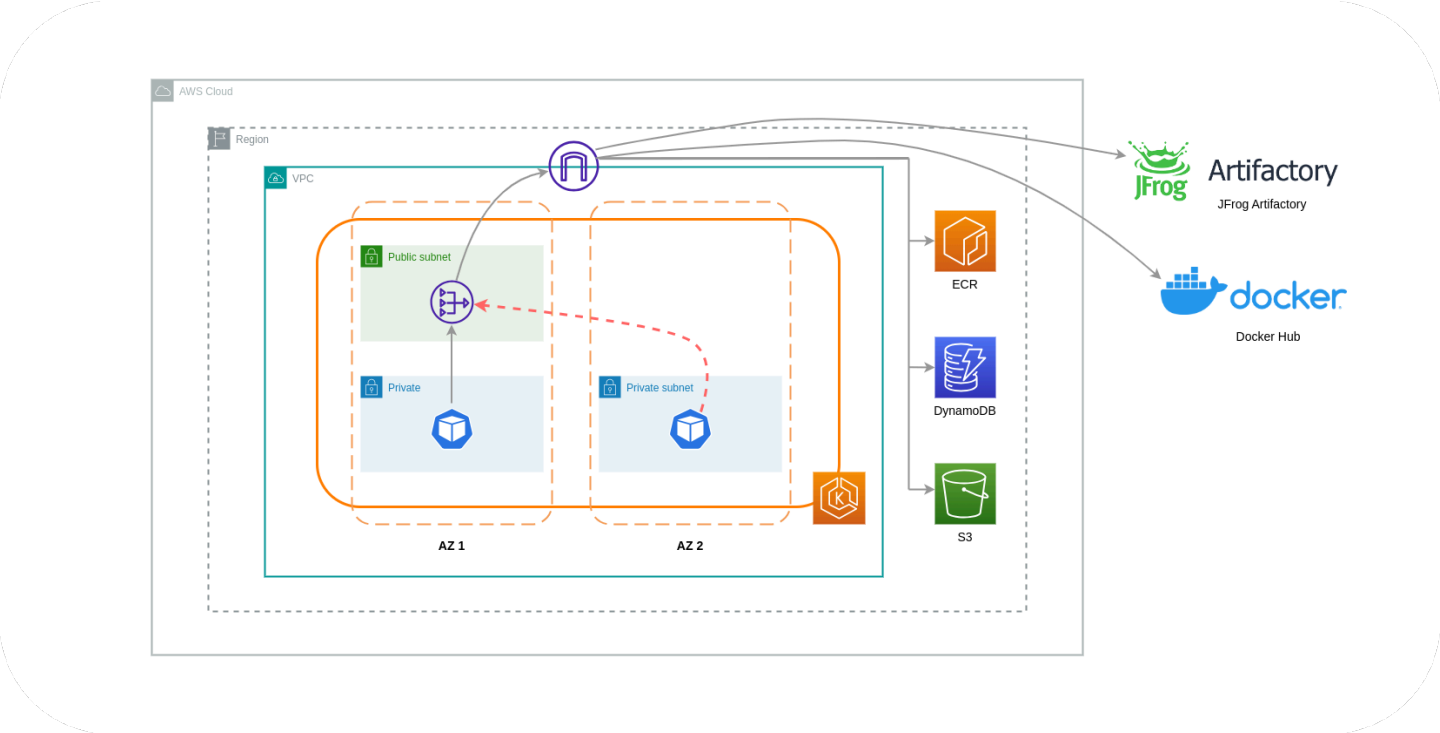
Se esegui carichi di lavoro con immagini particolarmente grandi, puoi creare Amazon Machine Images (AMIs) personalizzate con immagini di container prememorizzate nella cache. Ciò può ridurre il tempo iniziale di recupero dell'immagine e i potenziali costi di trasferimento dei dati da un registro di container ai nodi di lavoro EKS.

Trasferimento di dati verso Internet e servizi AWS

È pratica comune integrare i carichi di lavoro Kubernetes con altri servizi AWS o strumenti e piattaforme di terze parti tramite Internet. L'infrastruttura di rete sottostante utilizzata per instradare il traffico da e verso la destinazione pertinente può influire sui costi sostenuti nel processo di trasferimento dei dati.

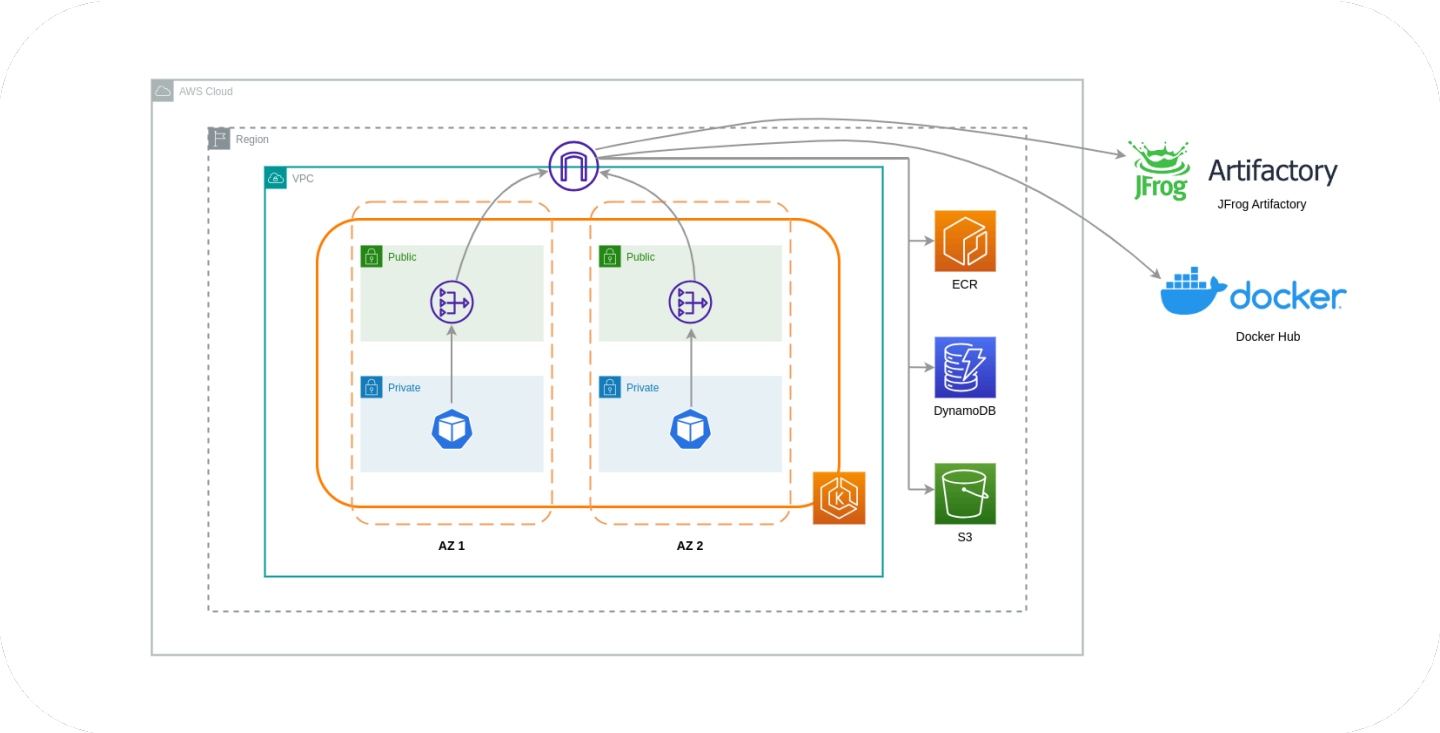
Utilizzo dei gateway NAT

I gateway NAT sono componenti di rete che eseguono la traduzione degli indirizzi di rete (NAT). Il diagramma seguente mostra i pod in un cluster EKS che comunicano con altri servizi AWS (Amazon ECR, DynamoDB e S3) e piattaforme di terze parti. In questo esempio, i Pod vengono eseguiti in sottoreti private separate. AZs Per inviare e ricevere traffico da Internet, un gateway NAT viene distribuito nella sottorete pubblica di una AZ, che consente a qualsiasi risorsa con indirizzi IP privati di condividere un unico indirizzo IP pubblico per accedere a Internet. Questo gateway NAT a sua volta comunica con il componente Internet Gateway, consentendo l'invio dei pacchetti alla destinazione finale.



Quando si utilizzano i gateway NAT per questi casi d'uso, è possibile ridurre al minimo i costi di trasferimento dei dati implementando un gateway NAT in ogni AZ. In questo modo, il traffico indirizzato a Internet passerà attraverso il gateway NAT nella stessa zona, evitando il trasferimento di dati tra le AZ. Tuttavia, anche se risparmierete sul costo del trasferimento dei dati tra AZ, questa configurazione implica che dovrete sostenere il costo di un gateway NAT aggiuntivo nella vostra architettura.

Questo approccio consigliato è illustrato nel diagramma seguente.



Utilizzo di endpoint VPC

Per ridurre ulteriormente i costi in tali architetture, è necessario utilizzare gli endpoint VPC per stabilire la connettività tra i carichi di lavoro e i servizi AWS. Gli endpoint VPC consentono di accedere ai servizi AWS dall'interno di un VPC senza data/network che i pacchetti attraversino Internet. Tutto il traffico è interno e rimane all'interno della rete AWS. Esistono due tipi di endpoint VPC: gli endpoint VPC di interfaccia (supportati da molti servizi AWS) e gli endpoint VPC Gateway (supportati solo da S3 e DynamoDB).

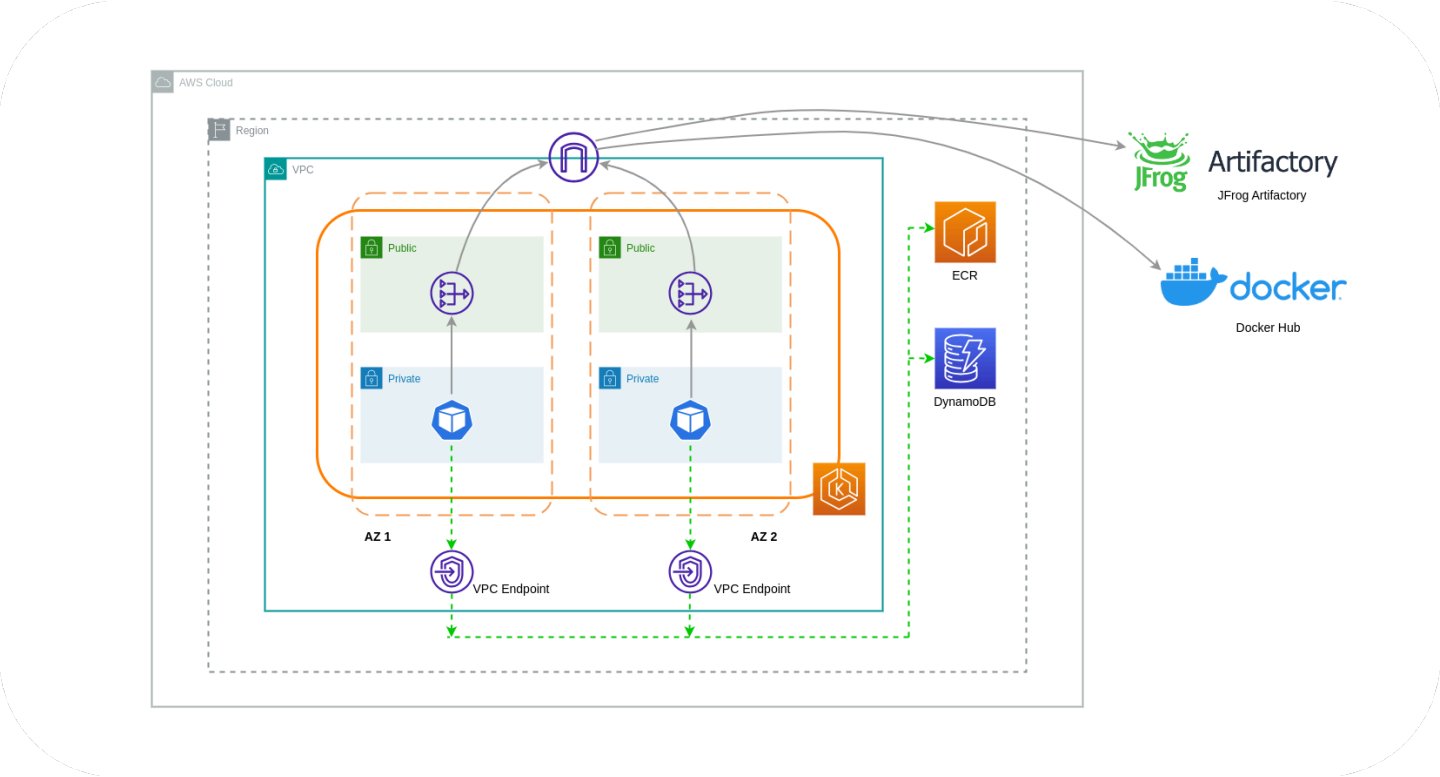
Endpoint VPC gateway

Non ci sono costi orari o di trasferimento dati associati agli endpoint VPC Gateway. Quando si utilizzano gli endpoint VPC Gateway, è importante notare che non sono estendibili oltre i limiti del VPC. Non possono essere utilizzati nel peering VPC, nelle reti VPN o tramite Direct Connect.

Endpoint VPC di interfaccia

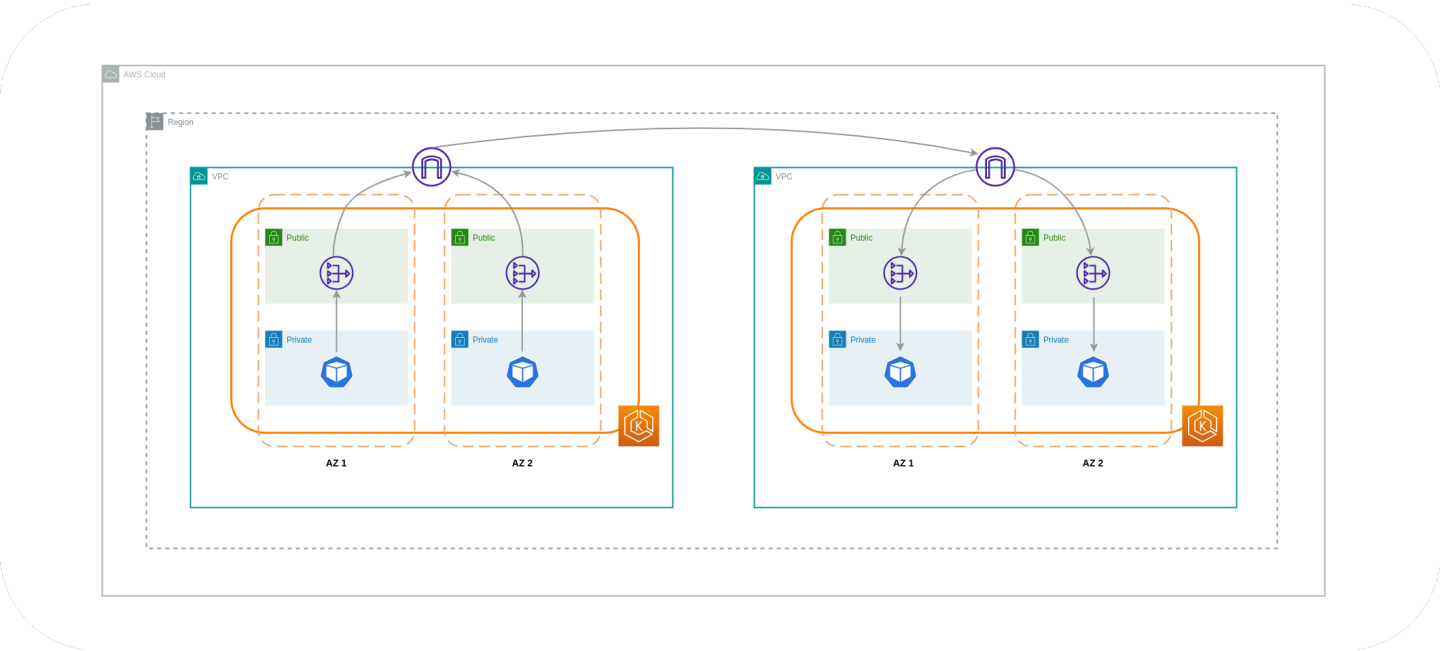
Gli endpoint VPC hanno una tariffa oraria e un costo aggiuntivo associato all'elaborazione dei dati tramite l'ENI sottostante. Tieni presente che il trasferimento dati tra AZ non viene addebitato.

Il diagramma seguente mostra i pod che comunicano con i servizi AWS tramite endpoint VPC.



Trasferimento di dati tra VPCs

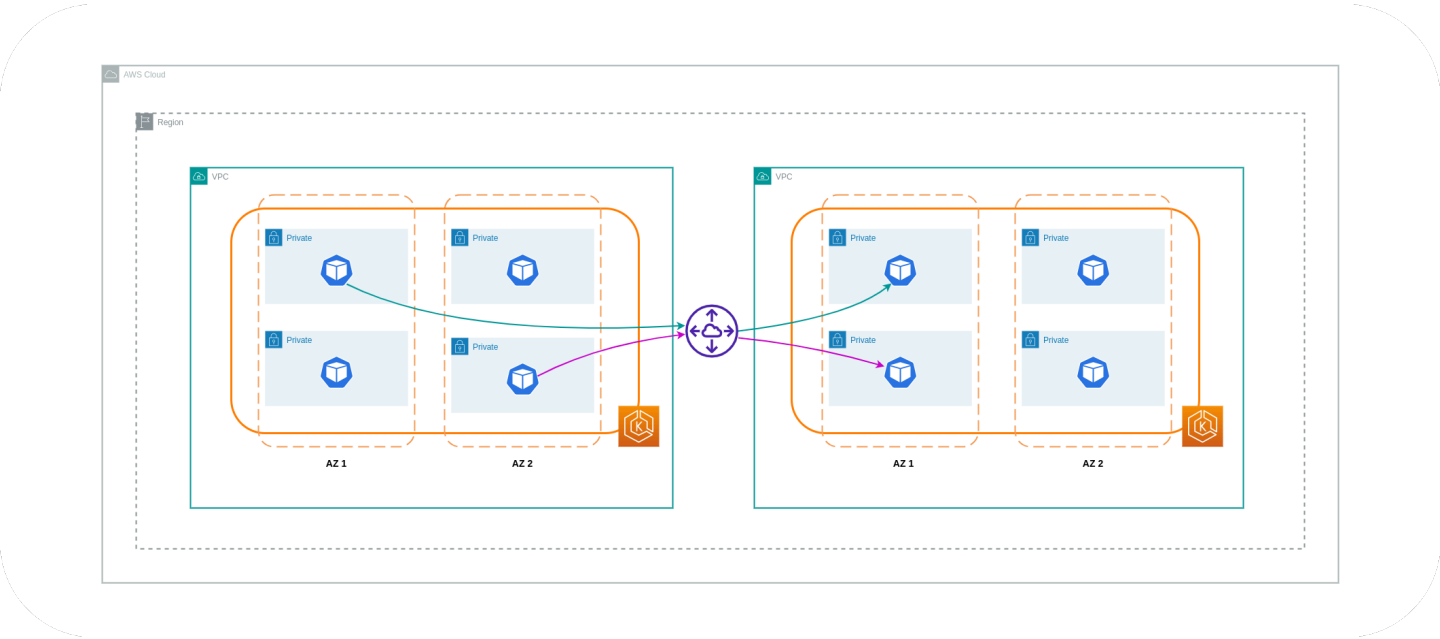
In alcuni casi, potresti avere carichi di lavoro in VPC distinti (all'interno della stessa regione AWS) che devono comunicare tra loro. Ciò può essere ottenuto consentendo al traffico di attraversare la rete Internet pubblica tramite Internet Gateway collegati ai rispettivi VPC. Tale comunicazione può essere abilitata implementando componenti di infrastruttura come istanze EC2, gateway NAT o istanze NAT in sottoreti pubbliche. Tuttavia, una configurazione che include questi componenti comporterà costi per i dati in entrata e in uscita da. processing/transferring VPCs Se il traffico da e verso la zona separata si VPCs sposta AZs, verrà addebitato un costo aggiuntivo per il trasferimento dei dati. Il diagramma seguente illustra una configurazione che utilizza i gateway NAT e i gateway Internet per stabilire la comunicazione tra carichi di lavoro diversi. VPCs



Connessioni peering VPC

Per ridurre i costi per questi casi d'uso, puoi utilizzare il peering VPC. Con una connessione peering VPC, non ci sono costi di trasferimento dati per il traffico di rete che rimane all'interno della stessa AZ. In caso di incrocio di traffico AZs, verrà addebitato un costo. Tuttavia, l'approccio VPC Peering è consigliato per una comunicazione economica tra carichi di lavoro separati all'interno della stessa regione AWS. VPCs Tuttavia, è importante notare che il peering VPC è efficace principalmente per la connettività VPC 1:1 perché non consente il networking transitivo.

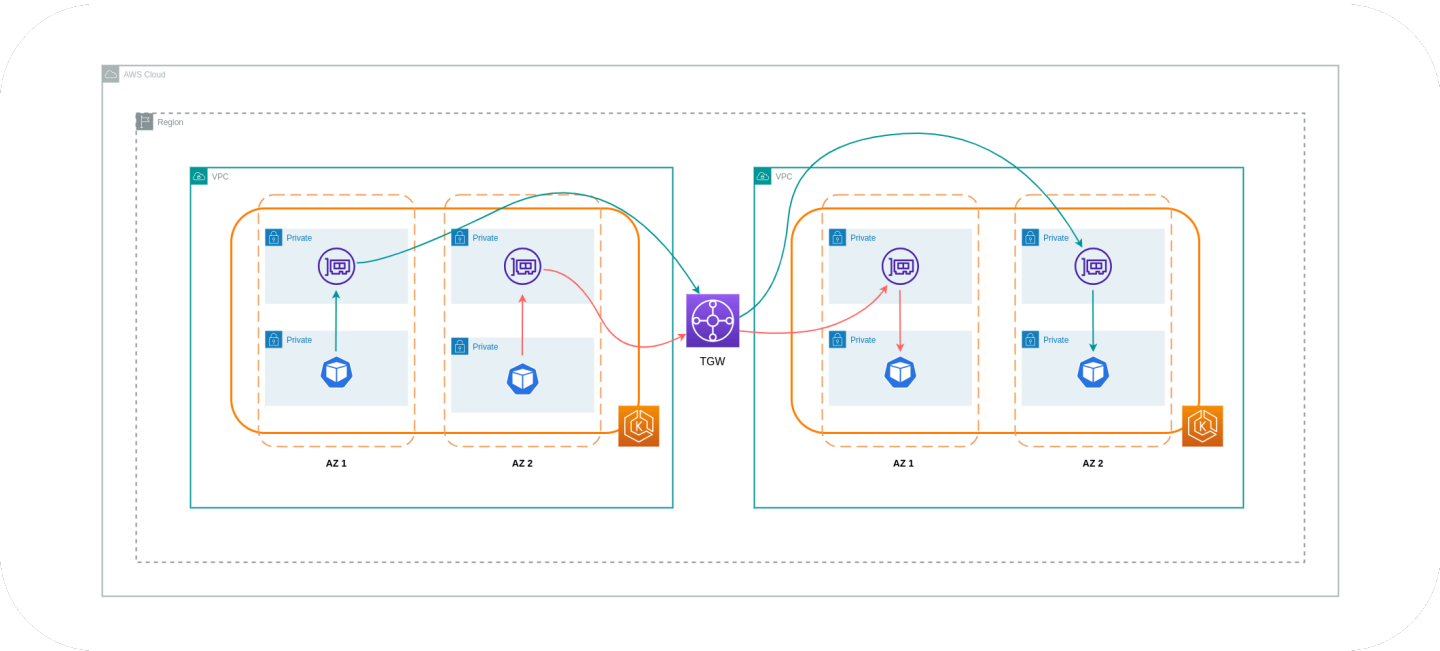
Il diagramma seguente è una rappresentazione di alto livello della comunicazione dei carichi di lavoro tramite una connessione peering VPC.



Connessioni di rete transitive

Come indicato nella sezione precedente, le connessioni peering VPC non consentono la connettività di rete transitiva. Se desideri connetterne 3 o più VPCs con requisiti di rete transitivi, allora dovresti usare un Transit Gateway (TGW). Ciò ti consentirà di superare i limiti del peering VPC o qualsiasi sovraccarico operativo associato alla presenza di più connessioni peering VPC tra più connessioni. VPCs La fatturazione viene effettuata su base oraria e per i dati inviati al TGW. Non vi è alcun costo di destinazione associato al traffico inter-AZ che attraversa il TGW.

Il diagramma seguente mostra il traffico inter-AZ che scorre attraverso un TGW tra carichi di lavoro in diverse ma VPCs all'interno della stessa regione AWS.



Utilizzo di una Service Mesh

Le service mesh offrono potenti funzionalità di rete che possono essere utilizzate per ridurre i costi relativi alla rete negli ambienti cluster EKS. Tuttavia, è necessario considerare attentamente le attività operative e la complessità che una service mesh introdurrà nel proprio ambiente se ne adotterà una.

Limitazione del traffico alle zone di disponibilità

Utilizzo della distribuzione ponderata per località di Istio

Istio consente di applicare le politiche di rete al traffico dopo il routing. Questa operazione viene eseguita utilizzando le regole di destinazione, come la distribuzione ponderata per località. Utilizzando questa funzione, puoi controllare il peso (espresso in percentuale) del traffico che può raggiungere una determinata destinazione in base alla sua origine. L'origine di questo traffico può provenire da un sistema di bilanciamento del carico esterno (o rivolto al pubblico) o da un Pod all'interno del cluster stesso. Quando tutti gli endpoint Pod sono disponibili, la località verrà selezionata in base a un algoritmo ponderato di bilanciamento del carico round-robin. Nel caso in cui alcuni endpoint non siano integri o non disponibili, il peso della località verrà regolato automaticamente per riflettere questo cambiamento negli endpoint disponibili.

> **Note**
>
> Prima di implementare la distribuzione ponderata per località, è necessario iniziare a comprendere i modelli di traffico di rete e le implicazioni che la politica della regola di

destinazione può avere sul comportamento dell'applicazione. Pertanto, è importante disporre di meccanismi di tracciamento distribuiti con strumenti come AWS X-Ray o Jaeger.

Le regole di destinazione Istio descritte sopra possono essere applicate anche per gestire il traffico da un sistema di bilanciamento del carico ai Pods nel cluster EKS. Le regole di distribuzione ponderate in base alla località possono essere applicate a un servizio che riceve traffico da un sistema di bilanciamento del carico ad alta disponibilità (in particolare Ingress Gateway). Queste regole consentono di controllare la quantità di traffico destinata a destinazione in base alla sua origine zonale, in questo caso il sistema di bilanciamento del carico. Se configurato correttamente, si verificherà una riduzione del traffico in uscita tra le zone rispetto a un sistema di bilanciamento del carico che distribuisce il traffico in modo uniforme o casuale su repliche Pod diverse. AZs

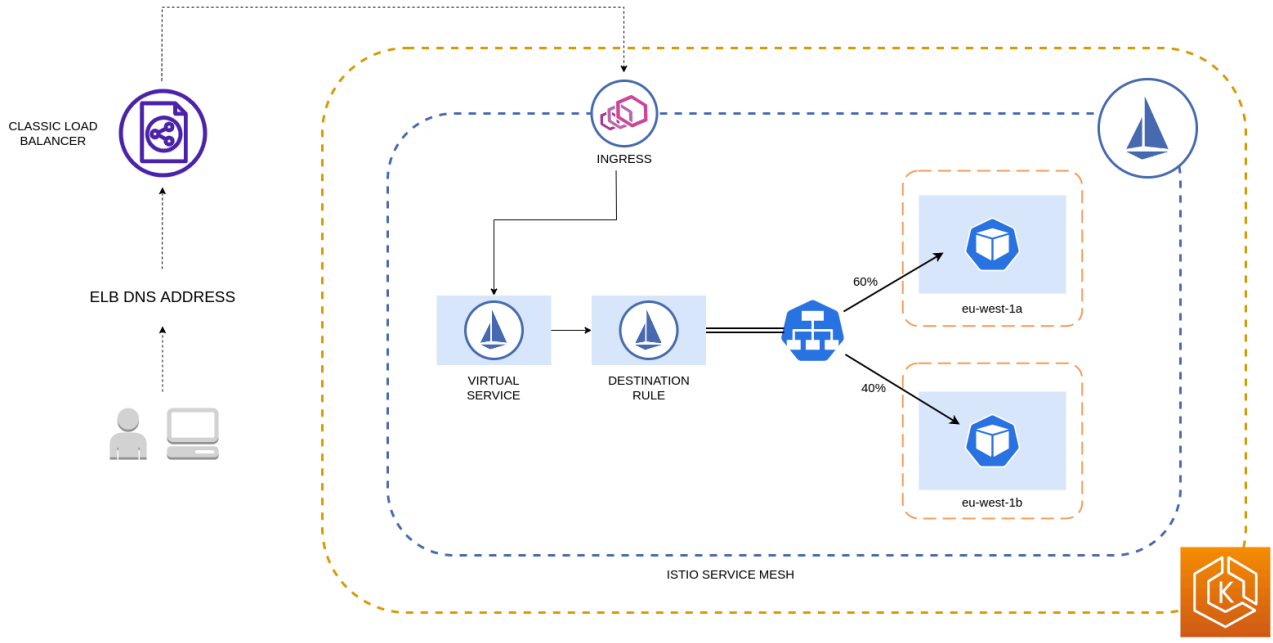
Di seguito è riportato un esempio di blocco di codice di una risorsa Destination Rule in Istio. Come si può vedere di seguito, questa risorsa specifica configurazioni ponderate per il traffico in entrata da 3 diverse località della regione. AZs `eu-west-1` Queste configurazioni dichiarano che la maggior parte del traffico in entrata (il 70% in questo caso) da una determinata AZ deve essere inviato tramite proxy a una destinazione nella stessa AZ da cui proviene.

```
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: express-test-dr
spec:
  host: express-test.default.svc.cluster.local
  trafficPolicy:
    loadBalancer:                      +
      localityLbSetting:
        distribute:
        - from: eu-west-1/eu-west-1a/  +
          to:
            "eu-west-1/eu-west-1a/_": 70
            "eu-west-1/eu-west-1b/_": 20
            "eu-west-1/eu-west-1c/_": 10
        - from: eu-west-1/eu-west-1b/_  +
          to:
            "eu-west-1/eu-west-1a/_": 20
            "eu-west-1/eu-west-1b/_": 70
            "eu-west-1/eu-west-1c/_": 10
        - from: eu-west-1/eu-west-1c/_  +
          to:
```

```
            "eu-west-1/eu-west-1a/_": 20
            "eu-west-1/eu-west-1b/_": 10
            "eu-west-1/eu-west-1c/*": 70**
    connectionPool:
      http:
        http2MaxRequests: 10
        maxRequestsPerConnection: 10
    outlierDetection:
      consecutiveGatewayErrors: 1
      interval: 1m
      baseEjectionTime: 30s
```

> **Note**
>
> Il peso minimo che può essere distribuito alla destinazione è dell'1%. Il motivo è quello di mantenere le regioni e le zone di failover nel caso in cui gli endpoint nella destinazione principale diventino non integri o non disponibili.

Il diagramma seguente illustra uno scenario in cui esiste un sistema di bilanciamento del carico ad alta disponibilità nella regione eu-west-1 e viene applicata la distribuzione ponderata per località. La politica relativa alla regola di destinazione di questo diagramma è configurata in modo da inviare il 60% del traffico proveniente da eu-west-1a ai Pod nella stessa AZ, mentre il 40% del traffico proveniente da eu-west-1a dovrebbe essere indirizzato ai Pod nell'eu-west-1b.



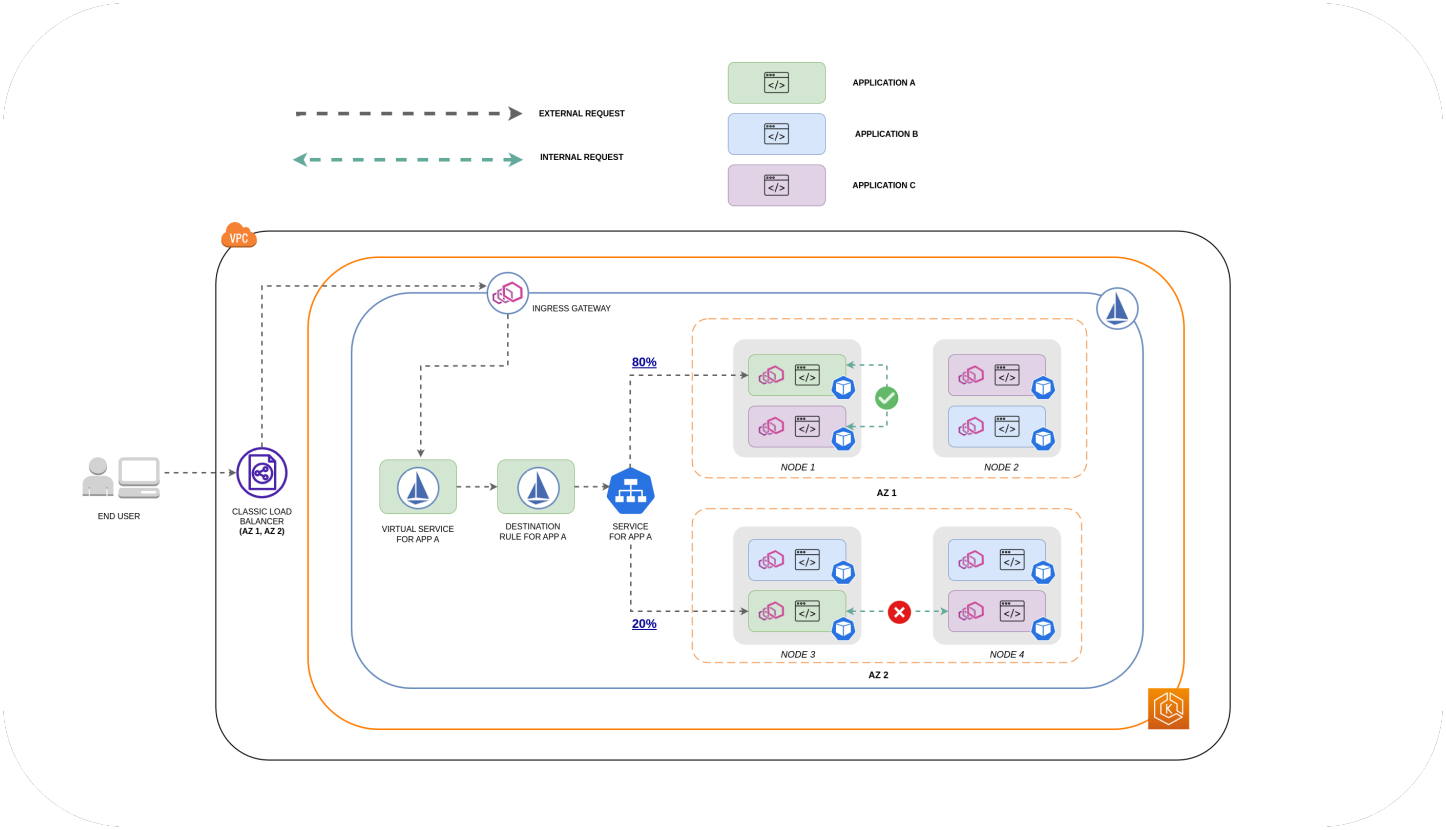
Limitazione del traffico alle zone e ai nodi di disponibilità

Utilizzo della politica sul traffico interno del servizio con Istio

Per mitigare i costi di rete associati al traffico esterno in entrata e al traffico interno tra i pod, puoi combinare le regole di destinazione di Istio e la politica sul traffico interno del servizio Kubernetes. Il modo di combinare le regole di destinazione di Istio con la politica sul traffico interno del servizio dipenderà in gran parte da 3 fattori:

- Il ruolo dei microservizi
- Modelli di traffico di rete tra i microservizi
- Come devono essere distribuiti i microservizi nella topologia del cluster Kubernetes

Il diagramma seguente mostra come sarebbe il flusso di rete nel caso di una richiesta annidata e come le politiche sopra menzionate controllerebbero il traffico.

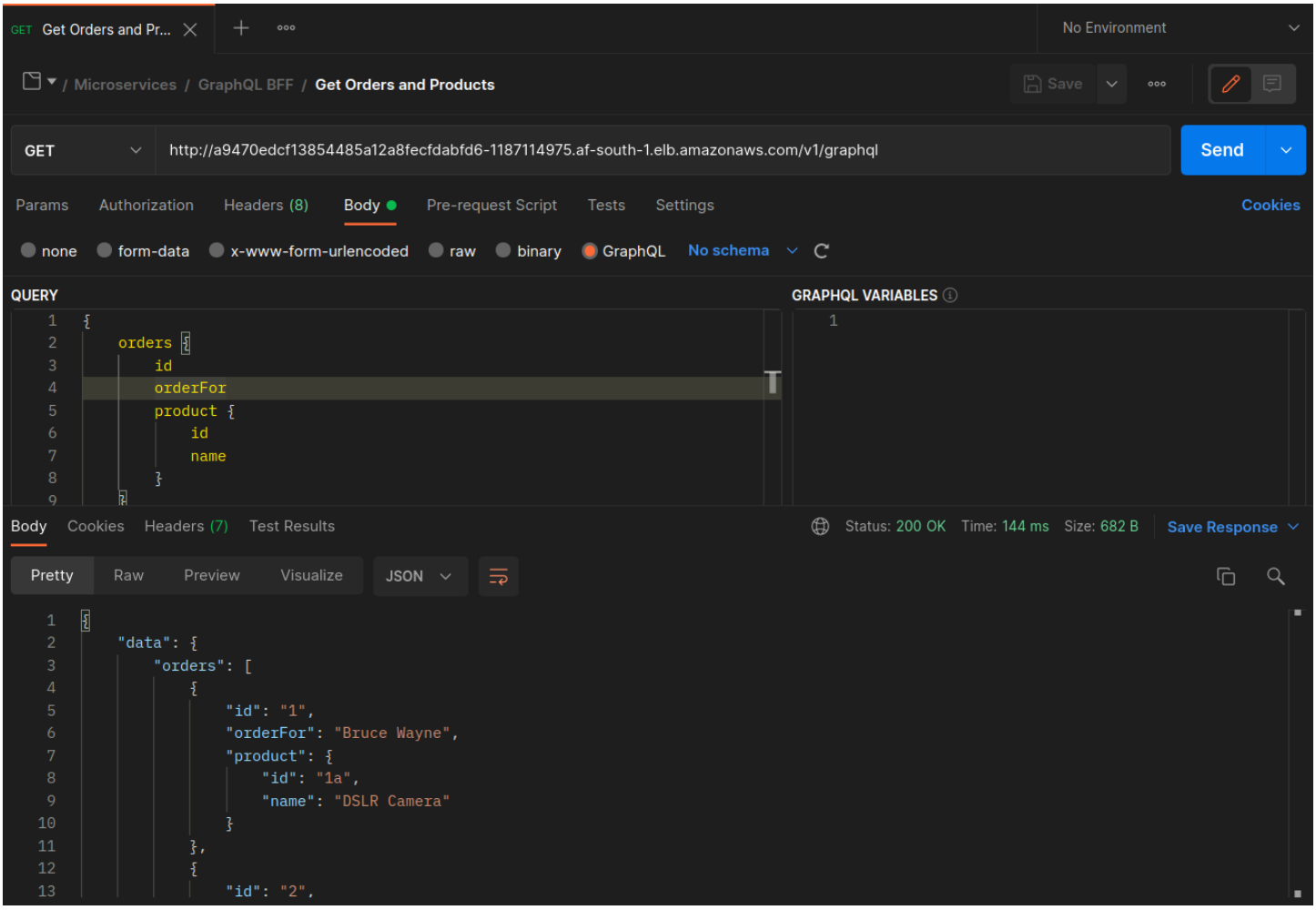


1. L'utente finale invia una richiesta all'APP A, che a sua volta invia una richiesta annidata all'APP C. Questa richiesta viene prima inviata a un sistema di bilanciamento del carico ad alta disponibilità, che dispone di istanze in AZ 1 e AZ 2, come mostra il diagramma precedente.
2. La richiesta esterna in entrata viene quindi indirizzata alla destinazione corretta dal servizio virtuale Istio.
3. Dopo che la richiesta è stata instradata, la regola di destinazione Istio controlla la quantità di traffico destinata alla rispettiva richiesta in AZs base alla sua provenienza (AZ 1 o AZ 2).
4. Il traffico passa quindi al Servizio per l'APP A e viene quindi inviato tramite proxy ai rispettivi endpoint Pod. Come mostrato nel diagramma, l'80% del traffico in entrata viene inviato agli endpoint Pod in AZ 1 e il 20% del traffico in entrata viene inviato a AZ 2.
5. L'APP A effettua quindi una richiesta interna all'APP C. Il servizio APP C ha una politica interna sul traffico abilitata (`internalTrafficPolicy`: Local`).
6. La richiesta interna dall'APP A (sul NODO 1) all'APP C ha esito positivo grazie all'endpoint locale del nodo disponibile per l'APP C.
7. La richiesta interna dall'APP A (su NODE 3) all'APP C ha esito negativo perché non ci sono endpoint locali del nodo disponibili per l'APP C. Come mostra il diagramma, l'APP C non ha repliche su NODE 3. * *

Le schermate seguenti sono tratte da un esempio dal vivo di questo approccio. La prima serie di schermate mostra una richiesta esterna riuscita a un `graphql` e una richiesta annidata riuscita da una `orders` replica co-localizzata sul `graphql` nodo. `ip-10-0-0-151.af-south-1.compute.internal`

test-results.txt

```
kubectl get pods -n ecommerce -o wide --field-selector spec.nodeName=ip-10-0-0-147.af-south-1.compute.i
NAME                        READY   STATUS    RESTARTS   AGE   IP          NODE
products-8567b458c8-brgdp   2/2     Running   0          64m   10.0.0.83   ip-10-0-0-147.af-south-1.com
--------------------------------------------------------------------------------
kubectl get pods -n ecommerce -o wide --field-selector spec.nodeName=ip-10-0-1-153.af-south-1.compute.i
NAME                        READY   STATUS    RESTARTS   AGE   IP          NODE
products-8567b458c8-5shq8   2/2     Running   0          65m   10.0.1.4    ip-10-0-1-153.af-south-1.comp
--------------------------------------------------------------------------------
kubectl get pods -n ecommerce -o wide --field-selector spec.nodeName=ip-10-0-1-68.af-south-1.compute.in
NAME                      READY   STATUS    RESTARTS   AGE   IP           NODE
orders-684bfbd6d9-5dxcf   2/2     Running   0          22m   10.0.1.145   ip-10-0-1-68.af-south-1.compu
--------------------------------------------------------------------------------
kubectl get pods -n ecommerce -o wide --field-selector spec.nodeName=ip-10-0-0-151.af-south-1.compute.i
NAME                      READY   STATUS    RESTARTS   AGE   IP           NODE
graphql-8f877c686-k7k9b   2/2     Running   0          18m   10.0.0.239   ip-10-0-0-151.af-south-1.comp
orders-684bfbd6d9-rc2zp   2/2     Running   0          18m   10.0.0.182   ip-10-0-0-151.af-south-1.comp
```



Con Istio, puoi verificare ed esportare le statistiche di tutti i cluster ed endpoint upstream di cui i tuoi proxy sono a conoscenza. Questo può aiutare a fornire un quadro del flusso di rete e della quota di distribuzione tra i servizi di un carico di lavoro. Continuando con lo stesso esempio, gli `orders` endpoint di cui è a conoscenza il `graphql` proxy possono essere ottenuti utilizzando il seguente comando:

```
kubectl exec -it deploy/graphql -n ecommerce -c istio-proxy -- curl localhost:15000/
clusters | grep orders
```

```
...
orders-service.ecommerce.svc.cluster.local::10.0.1.33:3003::**rq_error::0**
orders-service.ecommerce.svc.cluster.local::10.0.1.33:3003::**rq_success::119**
orders-service.ecommerce.svc.cluster.local::10.0.1.33:3003::**rq_timeout::0**
orders-service.ecommerce.svc.cluster.local::10.0.1.33:3003::**rq_total::119**
orders-service.ecommerce.svc.cluster.local::10.0.1.33:3003::**health_flags::healthy**
orders-service.ecommerce.svc.cluster.local::10.0.1.33:3003::**region::af-south-1**
orders-service.ecommerce.svc.cluster.local::10.0.1.33:3003::**zone::af-south-1b**
...
```

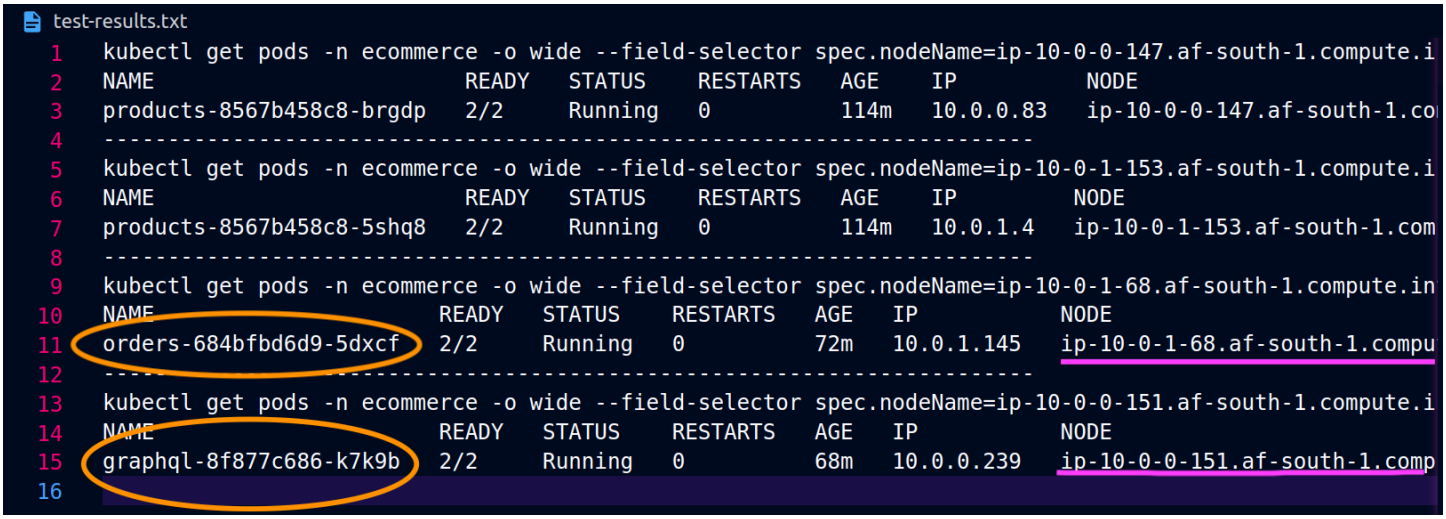
In questo caso, il `graphql` proxy conosce solo l'`orders`endpoint della replica con cui condivide un nodo. Se rimuovi l'`internalTrafficPolicy: Local`impostazione dal servizio ordini ed esegui nuovamente un comando come quello precedente, i risultati restituiranno tutti gli endpoint delle repliche distribuiti sui diversi nodi. Inoltre, esaminando `rq_total` i rispettivi endpoint, noterai una condivisione relativamente uniforme nella distribuzione della rete. Di conseguenza, se gli endpoint sono associati a servizi upstream che funzionano in modo diverso AZs, questa distribuzione di rete tra le zone comporterà costi più elevati.

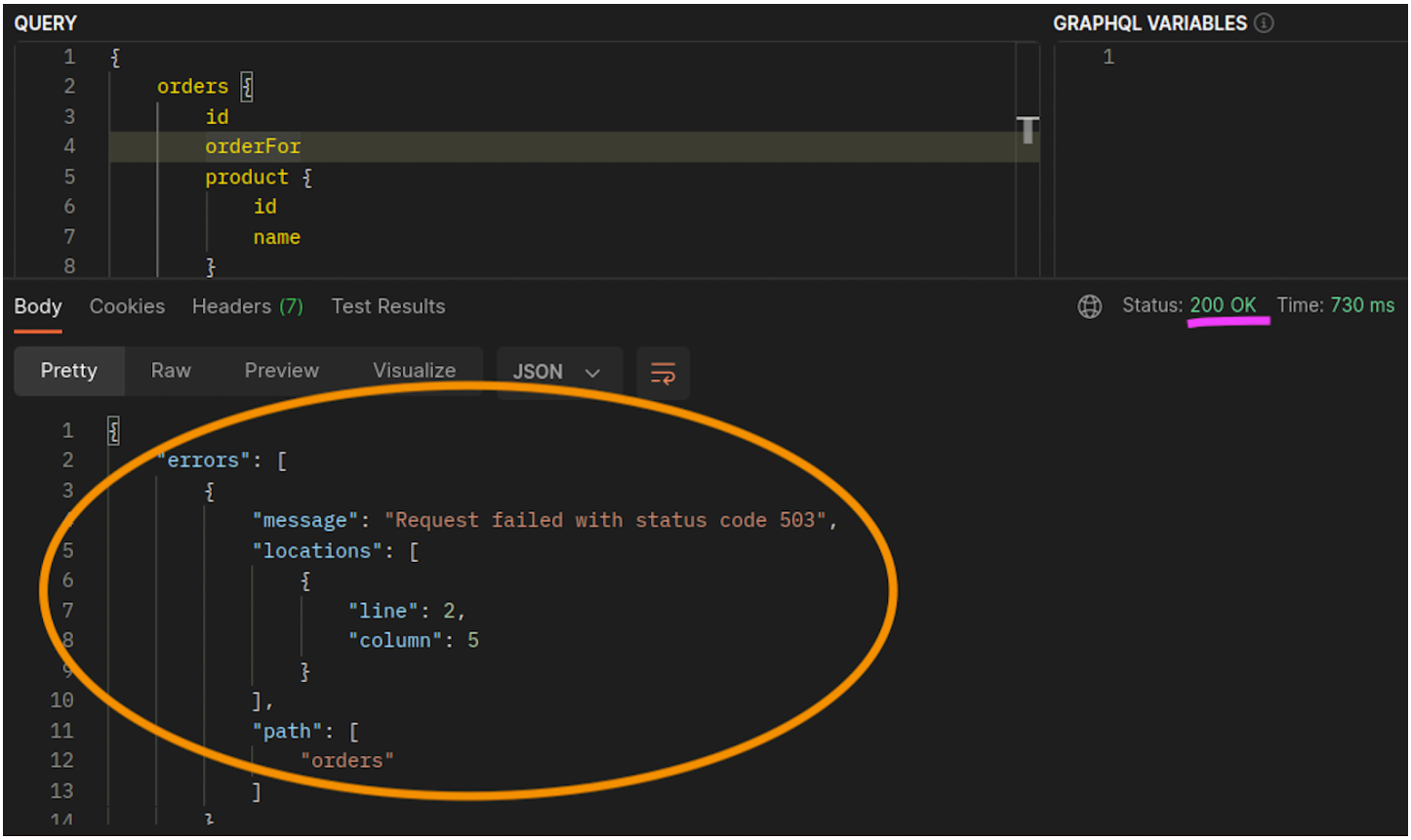
Come indicato nella sezione precedente, è possibile collocare insieme i Pod che comunicano frequentemente utilizzando l'affinità tra i pod.

```
...
spec:
...
  template:
    metadata:
      labels:
        app: graphql
        role: api
        workload: ecommerce
    spec:
```

```
      affinity:
        podAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
              - key: app
                operator: In
                values:
                - orders
            topologyKey: "kubernetes.io/hostname"
      nodeSelector:
        managedBy: karpenter
        billing-team: ecommerce
...
```

Quando le `orders` repliche `graphql` and non coesistono sullo stesso nodo (`ip-10-0-0-151.af-south-1.compute.internal`), la prima richiesta a `graphql` ha esito positivo, come indicato nella schermata di Postman riportata di seguito, mentre la `200 response code` seconda richiesta annidata da a ha esito negativo con a. `graphql` `orders` `503 response code`





Risorse aggiuntive

- Gestione della latenza e dei costi di trasferimento dei dati su EKS utilizzando Istio
- Esplorazione dell'effetto di Topology Aware Hints sul traffico di rete in Amazon Elastic Kubernetes Service
- Ottieni visibilità sui byte di rete da pod a pod di Amazon EKS Cross-AZ

- Ottimizza il traffico AZ con Istio
- Ottimizza il traffico AZ con Topology Aware Routing
- Ottimizza i costi e le prestazioni di Kubernetes con Service Internal Traffic Policy
- Ottimizza i costi e le prestazioni di Kubernetes con Istio e Service Internal Traffic Policy
- Overview of Data Transfer Costs for Common Architectures
- Comprensione dei costi di trasferimento dei dati per i servizi container AWS

Storage

Panoramica

Esistono scenari in cui è possibile eseguire applicazioni che richiedono la conservazione dei dati a breve o lungo termine. In questi casi d'uso, i volumi possono essere definiti e montati dai Pod in modo che i relativi contenitori possano attingere a diversi meccanismi di archiviazione. Kubernetes supporta diversi tipi di volumi per lo storage temporaneo e persistente. La scelta dello storage dipende in gran parte dai requisiti dell'applicazione. Per ogni approccio, esistono implicazioni in termini di costi e le pratiche descritte di seguito che consentono di ridurre i costi per i carichi di lavoro che richiedono una qualche forma di storage negli ambienti EKS.

Volumi effimeri

I volumi effimeri sono destinati ad applicazioni che richiedono volumi locali transitori ma non richiedono che i dati rimangano persistenti dopo il riavvio. Ne sono un esempio i requisiti per lo spazio di memoria virtuale, la memorizzazione nella cache e i dati di input di sola lettura come dati di configurazione e segreti. Puoi trovare maggiori dettagli sui volumi effimeri di Kubernetes qui. La maggior parte dei volumi temporanei (ad esempio emptyDir, ConfigMap, DownwardAPI, secret, hostpath) è supportata da dispositivi scrivibili collegati localmente (di solito il disco root) o RAM, quindi è importante scegliere il volume host più efficiente in termini di costi e prestazioni.

Utilizzo di volumi EBS

Consigliamo di iniziare con gp3 come volume root dell'host. È l'ultimo volume SSD generico offerto da Amazon EBS e offre anche un prezzo inferiore (fino al 20%) per GB rispetto ai volumi gp2.

Utilizzo di Amazon EC2 Instance Stores

Gli EC2 instance store di Amazon forniscono uno storage temporaneo a livello di blocco per le tue EC2 istanze. Lo storage fornito dagli EC2 instance store è accessibile tramite dischi fisicamente collegati agli host. A differenza di Amazon EBS, puoi collegare solo volumi di Instance Store all'avvio dell'istanza e questi volumi esistono solo per la durata dell'istanza. Non possono essere scollegati e ricollegati ad altre istanze. Puoi trovare ulteriori informazioni sugli Amazon EC2 Instance Store qui. Non sono previsti costi aggiuntivi associati al volume di un instance store. Ciò le rende (Instance Store Volume) più efficienti in termini di costi rispetto alle EC2 istanze generiche con grandi volumi EBS.

Per utilizzare i volumi di archiviazione locali in Kubernetes, devi partizionare, configurare e formattare i dischi utilizzando i dati utente di EC2 Amazon in modo che i volumi possano essere montati secondo le specifiche del pod. HostPath In alternativa, puoi sfruttare Local Persistent Volume Static Provisioner per semplificare la gestione dello storage locale. Il provisioner statico Local Persistent Volume consente di accedere ai volumi di archiviazione delle istanze locali tramite l'interfaccia PersistentVolumeClaim Kubernetes (PVC) standard. Inoltre, fornirà PersistentVolumes (PVs) che contiene informazioni sull'affinità dei nodi per pianificare i Pod sui nodi corretti. Sebbene utilizzi Kubernetes PersistentVolumes, i volumi di EC2 instance store sono di natura effimera. I dati scritti su dischi temporanei sono disponibili solo durante la vita dell'istanza. Quando l'istanza viene terminata, lo sono anche i dati. Consulta questo blog per maggiori dettagli.

Tieni presente che quando usi i volumi di Amazon EC2 Instance Store, il limite totale di IOPS viene condiviso con l'host e associa i Pod a un host specifico. È necessario esaminare attentamente i requisiti del carico di lavoro prima di adottare i volumi di Amazon EC2 Instance Store.

Volumi persistenti

Kubernetes è in genere associato all'esecuzione di applicazioni stateless. Tuttavia, esistono scenari in cui potresti voler eseguire microservizi che richiedono la conservazione di dati o informazioni persistenti da una richiesta all'altra. I database sono un esempio comune di questi casi d'uso. Tuttavia, i Pod e i contenitori o i processi al loro interno sono di natura effimera. Per conservare i dati oltre la durata di vita di un Pod, è possibile definire l' PVs accesso all'archiviazione in una posizione specifica indipendente dal Pod. I costi associati PVs dipendono in larga misura dal tipo di storage utilizzato e dal modo in cui le applicazioni lo utilizzano.

Di seguito sono elencati diversi tipi di opzioni di storage che supportano Kubernetes su PVs Amazon EKS. Le opzioni di storage descritte di seguito sono Amazon EBS, Amazon EFS, Amazon FSx for Lustre, Amazon FSx for NetApp ONTAP.

Volumi Amazon Elastic Block Store (EBS)

I volumi Amazon EBS possono essere utilizzati come Kubernetes PVs per fornire volumi di storage a livello di blocco. Sono ideali per database che si basano su letture e scritture casuali e applicazioni ad alta velocità di trasmissione che eseguono letture e scritture lunghe e continue. Il driver Amazon Elastic Block Store Container Storage Interface (CSI) consente ai cluster Amazon EKS di gestire il ciclo di vita dei volumi Amazon EBS per volumi persistenti. La Container Storage Interface abilita e facilita l'interazione tra Kubernetes e un sistema di storage. Quando un driver CSI viene distribuito nel cluster EKS, è possibile accedere alle sue funzionalità tramite le risorse di storage Kubernetes native come Persistent Volumes (PVs), Persistent Volume Claims () e Storage Classes (). PVCs SCs Questo collegamento fornisce esempi pratici di come interagire con i volumi Amazon EBS con il driver Amazon EBS CSI.

Scelta del volume giusto

Consigliamo di utilizzare lo storage a blocchi di ultima generazione (gp3) in quanto offre il giusto equilibrio tra prezzo e prestazioni. Consente inoltre di scalare gli IOPS e la velocità effettiva del volume indipendentemente dalle dimensioni del volume senza dover fornire ulteriore capacità di storage a blocchi. Se attualmente utilizzi volumi gp2, ti consigliamo vivamente di migrare ai volumi gp3. Il post sul blog La migrazione dei cluster Amazon EKS dai volumi EBS gp2 a gp3 spiega come migrare da gp2 a gp3 su cluster Amazon EKScon backup e ripristino utilizzando la funzionalità CSI Volume Snapshots, che richiede tempi di inattività delle applicazioni.

Amazon EBS consente di modificare le caratteristiche del volume come dimensione del volume, IOPS e velocità effettiva online. Utilizzando questa funzionalità è possibile migrare da gp2 a gp3 senza tempi di inattività delle applicazioni utilizzando le annotazioni PVC come descritto in questo blog, che richiede il driver CSI EBS v1.19.0+, o iniziando con Amazon EKS v1.31 e il driver EBS CSI 1.35 utilizzando l'API come descritto qui. VolumeAttributesClass

Quando hai applicazioni che richiedono prestazioni più elevate e richiedono volumi più grandi di quelli che un singolo volume gp3 può supportare, dovresti prendere in considerazione l'utilizzo di io2 block express. Questo tipo di storage è ideale per le implementazioni più grandi, I/O intensive e mission critical come SAP HANA o altri database di grandi dimensioni con requisiti di bassa latenza. Tieni presente che le prestazioni EBS di un'istanza sono limitate dai limiti di prestazioni dell'istanza, quindi non tutte le istanze supportano i volumi io2 block express. Puoi controllare i tipi di istanza supportati e altre considerazioni in questo documento.

Un singolo volume gp3 può supportare fino a 16.000 IOPS massimi, 1.000 velocità effettiva massima, MiB/s massimo 16 TiB. Volume SSD Provisioned IOPS di ultima generazione che fornisce fino a 256.000 IOPS, 4.000 MiB/s, throughput e 64 TiB.

Tra queste opzioni, è consigliabile personalizzare al meglio le prestazioni e i costi di storage in base alle esigenze delle applicazioni.

Monitora e ottimizza nel tempo

È importante comprendere le prestazioni di base dell'applicazione e monitorarla per determinati volumi per verificare se soddisfa i requisiti richiesti requirements/expectations o se il provisioning è eccessivo (ad esempio, uno scenario in cui gli IOPS assegnati non vengono utilizzati appieno).

Invece di allocare un volume elevato dall'inizio, è possibile aumentare gradualmente le dimensioni del volume man mano che si accumulano dati. Puoi ridimensionare dinamicamente i volumi utilizzando la funzionalità di ridimensionamento del volume nel driver CSI di Amazon Elastic Block Store (). aws-ebs-csi-driver Tieni presente che puoi solo aumentare la dimensione del volume EBS.

Per identificare e rimuovere eventuali volumi EBS sospesi, puoi utilizzare la categoria di ottimizzazione dei costi di AWS trusted advisor. Questa funzionalità consente di identificare volumi non collegati o volumi con un'attività di scrittura molto bassa per un periodo di tempo. Esiste uno strumento di sola lettura, open source e nativo del cloud, chiamato Popeye, che analizza i cluster Kubernetes attivi e segnala potenziali problemi con le risorse e le configurazioni distribuite. Ad esempio, può eseguire una scansione alla ricerca di elementi inutilizzati PVs e verificare se sono associati o se sono presenti errori di montaggio del PVCs volume.

Per un'analisi approfondita del monitoraggio, consulta la guida all'osservabilità dell'ottimizzazione dei costi di EKS.

Un'altra opzione da prendere in considerazione sono le raccomandazioni sul volume di AWS Compute Optimizer Amazon EBS. Questo strumento identifica automaticamente la configurazione ottimale del volume e il corretto livello di prestazioni richiesto. Ad esempio, può essere utilizzato per impostazioni ottimali relative agli IOPS assegnati, alle dimensioni dei volumi e ai tipi di volumi EBS in base all'utilizzo massimo negli ultimi 14 giorni. Quantifica inoltre i potenziali risparmi mensili sui costi derivanti dalle sue raccomandazioni. Puoi consultare questo blog per maggiori dettagli.

Politica di conservazione dei backup

Puoi eseguire il backup dei dati sui tuoi volumi Amazon EBS scattando point-in-time istantanee. Il driver Amazon EBS CSI supporta gli snapshot dei volumi. Puoi imparare a creare un'istantanea e ripristinare un PV EBS utilizzando i passaggi descritti qui.

Le istantanee successive sono backup incrementali, il che significa che vengono salvati solo i blocchi sul dispositivo che sono stati modificati dopo l'istantanea più recente. Ciò consente di ridurre il tempo necessario per creare lo snapshot e risparmiare sui costi di archiviazione in quanto i dati non vengono duplicati. Tuttavia, l'aumento del numero di vecchie istantanee EBS senza una politica di conservazione adeguata può causare costi imprevisti quando si opera su larga scala. Se esegui il backup diretto dei volumi Amazon EBS tramite l'API AWS, puoi sfruttare Amazon Data Lifecycle Manager (DLM) che fornisce una soluzione di gestione del ciclo di vita automatizzata e basata su policy per Amazon Elastic Block Store (EBS) Snapshots e Amazon Machine Images () con supporto EBS. AMIs La console semplifica l'automazione della creazione, della conservazione e dell'eliminazione di istantanee EBS e. AMIs

> **Note**
>
> Al momento non è possibile utilizzare Amazon DLM tramite il driver Amazon EBS CSI.

In un ambiente Kubernetes, puoi sfruttare uno strumento open source chiamato Velero per eseguire il backup dei tuoi EBS Persistent Volumes. È possibile impostare un flag TTL quando si pianifica la scadenza dei backup del processo. Ecco un esempio di guida di Velero.

Amazon Elastic File System (EFS)

Amazon Elastic File System (EFS) è un file system serverless e completamente elastico che consente di condividere i dati dei file utilizzando l'interfaccia di file system e la semantica del file system standard per un ampio spettro di carichi di lavoro e applicazioni. Esempi di carichi di lavoro e applicazioni includono Wordpress e Drupal, strumenti per sviluppatori come JIRA e Git e sistemi notebook condivisi come Jupyter e home directory.

Uno dei principali vantaggi di Amazon EFS è che può essere montato da più container distribuiti su più nodi e più zone di disponibilità. Un altro vantaggio è che paghi solo per lo storage che utilizzi. I file system EFS cresceranno e si ridurranno automaticamente man mano che si aggiungono e rimuovono file, eliminando la necessità di pianificare la capacità.

Per utilizzare Amazon EFS in Kubernetes, devi utilizzare il driver Amazon Elastic File System Container Storage Interface (CSI),. aws-efs-csi-driver Attualmente, il driver può creare punti di accesso in modo dinamico. Tuttavia, il file system Amazon EFS deve essere prima fornito e fornito come input per il parametro della classe di storage Kubernetes.

Scelta della classe di storage EFS giusta

Amazon EFS offre quattro classi di storage.

Due classi di storage standard:

- Standard Amazon EFS
- Amazon EFS Standard-Accesso non frequente (EFS Standard-IA)

Due classi di storage a zona singola:

- Amazon EFS Una zona
- Amazon EFS One Zone-Accesso non frequente (EFS One Zone-IA)

Le classi di storage Infrequent Access (IA) sono ottimizzate in termini di costi per i file a cui non si accede tutti i giorni. Con la gestione del ciclo di vita di Amazon EFS, puoi spostare file a cui non è stato effettuato l'accesso per la durata della politica del ciclo di vita (7, 14, 30, 60 o 90 giorni) nelle classi di storage IA, il che può ridurre i costi di storage fino al 92 percento rispetto rispettivamente alle classi di storage EFS Standard ed EFS One Zone.

Con EFS Intelligent-Tiering, la gestione del ciclo di vita monitora i modelli di accesso del file system e sposta automaticamente i file nella classe di storage più ottimale.

> **Note**
>
> aws-efs-csi-driver attualmente non ha il controllo sulla modifica delle classi di storage, sulla gestione del ciclo di vita o su Intelligent-Tiering. Questi devono essere configurati manualmente nella console AWS o tramite EFS APIs.

> **Note**
>
> aws-efs-csi-driver non è compatibile con le immagini dei contenitori basate su Windows.

> **Note**
>
> Esiste un problema di memoria noto quando vol-metrics-opt-in(per l'emissione di parametri di volume) è abilitata a causa della DiskUsagefunzione che consuma una quantità di memoria proporzionale alla dimensione del file system. Attualmente, si consiglia di disabilitare l'opzione `-- vol-metrics-opt-in `su file system di grandi dimensioni per evitare di consumare troppa memoria. Ecco un link alla pubblicazione di github per maggiori dettagli.

Amazon FSx per Lustre

Lustre è un file system parallelo ad alte prestazioni comunemente utilizzato nei carichi di lavoro che richiedono un throughput fino a centinaia GB/s e latenze inferiori al millisecondo per operazione. Viene utilizzato per scenari come la formazione sull'apprendimento automatico, la modellazione finanziaria, l'HPC e l'elaborazione video. Amazon FSx for Lustre offre uno storage condiviso completamente gestito con scalabilità e prestazioni, perfettamente integrato con Amazon S3.

Puoi utilizzare i volumi di storage persistente Kubernetes supportati da FSx for Lustre utilizzando il driver CSI for Lustre di Amazon EKS o il FSx tuo cluster Kubernetes autogestito su AWS. Consulta la documentazione di Amazon EKS per ulteriori dettagli ed esempi.

Collegamento ad Amazon S3

Si consiglia di collegare un repository di dati a lungo termine altamente durevole che risiede su Amazon S3 al file system for FSx Lustre. Una volta collegati, i set di dati di grandi dimensioni vengono caricati lentamente in base alle esigenze da Amazon S3 ai file system for Lustre. FSx Puoi anche eseguire le analisi e i risultati su S3 e quindi eliminare il file system Lustre.

Scelta delle opzioni di implementazione e archiviazione corrette

FSx for Lustre offre diverse opzioni di implementazione. La prima opzione si chiama scratch e non replica i dati, mentre la seconda opzione è chiamata persistent e, come suggerisce il nome, persiste i dati.

La prima opzione (scratch) può essere utilizzata per ridurre il costo dell'elaborazione temporanea dei dati a breve termine. L'opzione di distribuzione persistente è progettata per lo storage a lungo termine che replica automaticamente i dati all'interno di una zona di disponibilità AWS. Supporta anche lo storage SSD e HDD.

Puoi configurare il tipo di implementazione desiderato in base ai parametri di Kubernetes del filesystem FSx for lustre. StorageClass Ecco un link che fornisce modelli di esempio.

> **Note**
>
> Per carichi di lavoro sensibili alla latenza o carichi di lavoro che richiedono i massimi livelli di IOPS/throughput, dovresti scegliere lo storage SSD. Per carichi di lavoro incentrati sul throughput e non sensibili alla latenza, dovresti scegliere lo storage su HDD.

Abilita la compressione dei dati

È inoltre possibile abilitare la compressione dei dati sul file system specificando "LZ4" come tipo di compressione dei dati. Una volta abilitata, tutti i file appena scritti verranno automaticamente compressi su Lustre prima FSx di essere scritti su disco e decompressi quando vengono letti. LZ4 l'algoritmo di compressione dei dati è senza perdite, quindi i dati originali possono essere completamente ricostruiti dai dati compressi.

Puoi configurare il tipo di compressione dei dati come indicato nei parametri LZ4 di Kubernetes del file system FSx for lustre. StorageClass La compressione è disabilitata quando il valore è impostato su NONE, che è l'impostazione predefinita. Questo collegamento fornisce modelli di esempio.

> **Note**
>
> Amazon FSx for Lustre non è compatibile con le immagini di container basate su Windows.

Amazon FSx per NetApp ONTAP

Amazon FSx for NetApp ONTAP è uno storage condiviso completamente gestito basato sul NetApp file system ONTAP. FSx for ONTAP offre uno storage di file condiviso ricco di funzionalità, veloce e flessibile, ampiamente accessibile dalle istanze di calcolo Linux, Windows e macOS in esecuzione in AWS o in locale.

Amazon FSx for NetApp ONTAP supporta due livelli di storage: 1/livello primario e 2/livello di pool di capacità.

Il livello principale è un livello predisposto basato su SSD ad alte prestazioni per dati attivi e sensibili alla latenza. Il livello di pool di capacità completamente elastico è ottimizzato in termini di costi per

i dati a cui si accede raramente, si ridimensiona automaticamente man mano che i dati vengono suddivisi su più livelli e offre petabyte di capacità praticamente illimitati. È possibile abilitare la compressione e la deduplicazione dei dati sullo storage con pool di capacità e ridurre ulteriormente la quantità di capacità di storage consumata dai dati. NetAppla FabricPool funzionalità nativa basata su policy monitora continuamente i modelli di accesso ai dati, trasferendo automaticamente i dati in modo bidirezionale tra i livelli di storage per ottimizzare prestazioni e costi.

NetAppAstra Trident fornisce un'orchestrazione dinamica dello storage utilizzando un driver CSI che consente ai cluster Amazon EKS di gestire il ciclo di vita dei volumi persistenti supportati PVs da Amazon per i file system ONTAP. FSx NetApp Per iniziare, consulta Use Astra Trident with Amazon FSx for NetApp ONTAP nella documentazione di Astra Trident.

Altre considerazioni

Riduci al minimo le dimensioni dell'immagine del contenitore

Una volta implementati i contenitori, le immagini dei contenitori vengono memorizzate nella cache dell'host come livelli multipli. Riducendo le dimensioni delle immagini, è possibile ridurre la quantità di storage richiesta sull'host.

Utilizzando fin dall'inizio immagini di base ridotte, come immagini scratch o immagini di container senza distribuzione (che contengono solo l'applicazione e le relative dipendenze di runtime), è possibile ridurre i costi di storage oltre ad altri vantaggi accessori, come la riduzione della superficie di attacco e tempi di recupero delle immagini più brevi.

È inoltre consigliabile prendere in considerazione l'utilizzo di strumenti open source, come Slim.ai, che offre un modo semplice e sicuro per creare immagini minime.

Più livelli di pacchetti, strumenti, dipendenze delle applicazioni e librerie possono facilmente aumentare le dimensioni dell'immagine del contenitore. Utilizzando build in più fasi, potete copiare selettivamente gli artefatti da una fase all'altra, escludendo tutto ciò che non è necessario dall'immagine finale. Puoi consultare altre best practice per la creazione di immagini qui.

Un'altra cosa da considerare è per quanto tempo conservare le immagini memorizzate nella cache. Potresti voler pulire le immagini obsolete dalla cache delle immagini quando viene utilizzata una certa quantità di disco. In questo modo sarà possibile assicurarsi di disporre di spazio sufficiente per il funzionamento dell'host. Per impostazione predefinita, il kubelet esegue la raccolta dei rifiuti sulle immagini non utilizzate ogni cinque minuti e sui contenitori inutilizzati ogni minuto.

Per configurare le opzioni per il contenitore inutilizzato e la raccolta dei rifiuti di immagini, ottimizza il kubelet utilizzando un file di configurazione e modifica i parametri relativi alla raccolta dei rifiuti utilizzando il tipo di risorsa. *KubeletConfiguration*

Puoi saperne di più nella documentazione di Kubernetes.

Osservabilità

Introduzione

Gli strumenti di osservabilità ti aiutano a rilevare, correggere e analizzare in modo efficiente i tuoi carichi di lavoro. Il costo dei dati di telemetria aumenta naturalmente con l'aumento dell'utilizzo di EKS. A volte può essere difficile bilanciare le esigenze operative e misurare ciò che conta per l'azienda e tenere sotto controllo i costi di osservabilità. Questa guida si concentra sulle strategie di ottimizzazione dei costi per i tre pilastri dell'osservabilità: log, metriche e tracce. Ognuna di queste best practice può essere applicata indipendentemente per soddisfare gli obiettivi di ottimizzazione dell'organizzazione.

Registrazione

La registrazione svolge un ruolo fondamentale nel monitoraggio e nella risoluzione dei problemi delle applicazioni del cluster. Esistono diverse strategie che possono essere utilizzate per ottimizzare i costi di registrazione. Le strategie di best practice elencate di seguito includono l'esame delle politiche di conservazione dei log per implementare controlli granulari sulla durata di conservazione dei dati di registro, l'invio dei dati di log a diverse opzioni di archiviazione in base all'importanza e l'utilizzo del filtro dei log per restringere i tipi di messaggi di log archiviati. Una gestione efficiente della telemetria dei log può portare a risparmi sui costi per gli ambienti.

Piano di controllo EKS

Ottimizza i log del tuo Control Plane

Il piano di controllo Kubernetes è un insieme di componenti che gestiscono i cluster e questi componenti inviano diversi tipi di informazioni come flussi di log a un gruppo di log in Amazon. CloudWatch Sebbene l'abilitazione di tutti i tipi di log del piano di controllo comporti dei vantaggi, è necessario essere consapevoli delle informazioni contenute in ogni registro e dei costi associati all'archiviazione di tutta la telemetria dei log. Ti vengono addebitati i costi standard di acquisizione

e archiviazione dei dati di CloudWatch Logs per i log inviati ad Amazon CloudWatch Logs dai tuoi cluster. Prima di abilitarli, valuta se ogni flusso di log è necessario.

Ad esempio, nei cluster non di produzione, abilita selettivamente tipi di log specifici, come i log del server api, solo per l'analisi e disattivali in seguito. Tuttavia, per i cluster di produzione, in cui potrebbe non essere possibile riprodurre gli eventi e la risoluzione dei problemi richiede più informazioni di registro, è possibile abilitare tutti i tipi di log. Ulteriori dettagli sull'implementazione dell'ottimizzazione dei costi del piano di controllo sono disponibili in questo post del blog.

Trasmetti i log su S3

Un'altra best practice per l'ottimizzazione dei costi è lo streaming dei log del piano di controllo su S3 tramite abbonamenti Logs. CloudWatch L'utilizzo CloudWatch degli abbonamenti Logs consente di inoltrare selettivamente i log a S3, il che offre uno storage a lungo termine più efficiente in termini di costi rispetto alla conservazione dei log a tempo indeterminato. CloudWatch Ad esempio, per i cluster di produzione, puoi creare un gruppo di log critico e sfruttare gli abbonamenti per trasmettere questi log a S3 dopo 15 giorni. Ciò garantirà un accesso rapido ai log per l'analisi, ma anche un risparmio sui costi spostando i log in uno storage più efficiente in termini di costi.

> **Important**
>
> A partire dal 5/09/2023, i log EKS sono classificati come Vended Logs in Amazon Logs. CloudWatch I Vended Logs sono log di servizio AWS specifici pubblicati nativamente dai servizi AWS per conto del cliente e disponibili a prezzi scontati in base al volume. Visita la pagina CloudWatch dei prezzi di Amazon per ulteriori informazioni sui prezzi di Vended Logs.

Piano dati EKS

Retention dei log

La politica CloudWatch di conservazione predefinita di Amazon prevede la conservazione dei log a tempo indeterminato e senza scadenza, con costi di storage applicabili alla tua regione AWS. Per ridurre i costi di archiviazione, puoi personalizzare la politica di conservazione per ogni gruppo di log in base ai requisiti del carico di lavoro.

In un ambiente di sviluppo, potrebbe non essere necessario un lungo periodo di conservazione. In un ambiente di produzione, tuttavia, è possibile impostare una politica di conservazione più lunga

per soddisfare i requisiti di risoluzione dei problemi, conformità e pianificazione della capacità. Ad esempio, se utilizzate un'applicazione di e-commerce durante l'alta stagione delle festività natalizie, il sistema è sottoposto a un carico maggiore e possono insorgere problemi che potrebbero non essere immediatamente evidenti, vi consigliamo di impostare una conservazione dei log più lunga per una risoluzione dettagliata dei problemi e un'analisi post-evento.

Puoi configurare i periodi di conservazione nella CloudWatch console AWS o nell'API AWS con una durata da 1 giorno a 10 anni in base a ciascun gruppo di log. Un periodo di conservazione flessibile può far risparmiare sui costi di archiviazione dei log, mantenendo allo stesso tempo i log critici.

Opzioni di archiviazione dei log

Lo storage è un fattore importante dei costi di osservabilità, pertanto è fondamentale ottimizzare la strategia di archiviazione dei log. Le strategie devono essere in linea con i requisiti dei carichi di lavoro, mantenendo al contempo prestazioni e scalabilità. Una strategia per ridurre i costi di archiviazione dei log consiste nell'utilizzare i bucket AWS S3 e i diversi livelli di storage.

Inoltra i log direttamente a S3

Valuta la possibilità di inoltrare i log meno critici, come gli ambienti di sviluppo, direttamente a S3 anziché a Cloudwatch. Ciò può avere un impatto immediato sui costi di archiviazione dei log. Un'opzione è inoltrare i log direttamente a S3 utilizzando Fluentbit. Lo definisci nella `[OUTPUT]` sezione, la destinazione in cui FluentBit trasmette i log del contenitore per la conservazione. Consulta i parametri di configurazione aggiuntivi qui.

```
[OUTPUT]
        Name eks_to_s3
        Match application.*
        bucket $S3_BUCKET name
        region us-east-2
        store_dir /var/log/fluentbit
        total_file_size 30M
        upload_timeout 3m
```

Inoltra i log CloudWatch solo per analisi a breve termine

Per i log più critici, ad esempio ambienti di produzione in cui potrebbe essere necessario eseguire un'analisi immediata dei dati, è consigliabile inoltrare i log a. CloudWatch Lo definisci nella `[OUTPUT]` sezione, la destinazione in cui FluentBit trasmette i log dei contenitori per la conservazione. Consulta i parametri di configurazione aggiuntivi qui.

```
[OUTPUT]
        Name eks_to_cloudwatch_logs
        Match application.*
        region us-east-2
        log_group_name fluent-bit-cloudwatch
        log_stream_prefix from-fluent-bit-
        auto_create_group On
```

Tuttavia, ciò non avrà un impatto immediato sui risparmi sui costi. Per ulteriori risparmi, dovrai esportare questi log in Amazon S3.

Esporta in Amazon S3 da CloudWatch

Per archiviare CloudWatch i log di Amazon a lungo termine, consigliamo di esportare i CloudWatch log di Amazon EKS su Amazon Simple Storage Service (Amazon S3). Puoi inoltrare i log al bucket Amazon S3 creando un'attività di esportazione tramite la console o l'API. Dopo averlo fatto, Amazon S3 presenta molte opzioni per ridurre ulteriormente i costi. Puoi definire le tue regole del ciclo di vita di Amazon S3 per spostare i log in una classe di storage più adatta alle tue esigenze o sfruttare la classe di storage Amazon S3 Intelligent-Tiering per fare in modo che AWS sposti automaticamente i dati in uno storage a lungo termine in base al tuo modello di utilizzo. Consulta questo blog per maggiori dettagli. Ad esempio, per il tuo ambiente di produzione, i log rimangono archiviati CloudWatch per più di 30 giorni, quindi vengono esportati nel bucket Amazon S3. Puoi quindi utilizzare Amazon Athena per interrogare i dati nel bucket Amazon S3 se devi fare riferimento ai log in un secondo momento.

Riduci i livelli di log

Pratica la registrazione selettiva per la tua applicazione. Per impostazione predefinita, sia le applicazioni che i nodi generano i log. Per i log delle applicazioni, regola i livelli di registro in base alla criticità del carico di lavoro e dell'ambiente. Ad esempio, l'applicazione java riportata di seguito genera `INFO` log, che è la tipica configurazione predefinita dell'applicazione e, a seconda del codice, può generare un volume elevato di dati di registro.

```
import org.apache.log4j.*;

public class LogClass {
   private static org.apache.log4j.Logger log = Logger.getLogger(LogClass.class);

public static void main(String[] args) {
      log.setLevel(Level.INFO);
```

```
   log.debug("This is a DEBUG message, check this out!");
   log.info("This is an INFO message, nothing to see here!");
   log.warn("This is a WARN message, investigate this!");
   log.error("This is an ERROR message, check this out!");
   log.fatal("This is a FATAL message, investigate this!");    } }
```

In un ambiente di sviluppo, imposta il livello di log in questo modo puoi eseguire DEBUG il debug dei problemi o individuarne di potenziali prima che entrino in produzione.

```
      log.setLevel(Level.DEBUG);
```

In un ambiente di produzione, valuta la possibilità di modificare il livello di registro in ERROR o. FATAL Ciò genererà il log solo quando l'applicazione presenta errori, riducendo l'output del registro e aiutandovi a concentrarvi sui dati importanti sullo stato dell'applicazione.

```
      log.setLevel(Level.ERROR);
```

Puoi ottimizzare i vari livelli di log dei componenti di Kubernetes. Ad esempio, se si utilizza Bottlerocket come sistema operativo EKS Node, esistono impostazioni di configurazione che consentono di regolare il livello di registro del processo Kubelet. Di seguito è riportato un frammento di questa impostazione di configurazione. Notate il livello di registro predefinito pari a 2, che regola la verbosità di registrazione del processo. `kubelet`

```
[settings.kubernetes]
log-level = "2"
image-gc-high-threshold-percent = "85"
image-gc-low-threshold-percent = "80"
```

Per un ambiente di sviluppo, è possibile impostare un livello di registro maggiore di 2 per visualizzare eventi aggiuntivi, utile per il debug. Per un ambiente di produzione, è possibile impostare il livello su 0 per visualizzare solo gli eventi critici.

Sfrutta i filtri

Quando si utilizza una configurazione EKS Fluentbit predefinita per inviare i log dei container a Cloudwatch, FluentBit acquisisce e invia TUTTI i log dei contenitori delle applicazioni arricchiti con metadati Kubernetes a Cloudwatch, come mostrato nel riquadro di configurazione riportato di seguito. `[INPUT]`

```
 [INPUT]
     Name                tail
     Tag                 application.*
     Exclude_Path        /var/log/containers/cloudwatch-agent*, /var/log/containers/
fluent-bit*, /var/log/containers/aws-node*, /var/log/containers/kube-proxy*
     Path                /var/log/containers/*.log
     Docker_Mode         On
     Docker_Mode_Flush   5
     Docker_Mode_Parser  container_firstline
     Parser              docker
     DB                  /var/fluent-bit/state/flb_container.db
     Mem_Buf_Limit       50MB
     Skip_Long_Lines     On
     Refresh_Interval    10
     Rotate_Wait         30
     storage.type        filesystem
     Read_from_Head      ${READ_FROM_HEAD}
```

La sezione precedente `[INPUT]` sta importando tutti i log del contenitore. Questo può generare una grande quantità di dati che potrebbero non essere necessari. Il filtraggio di questi dati può ridurre la quantità di dati di registro inviati e CloudWatch quindi ridurre i costi. È possibile applicare un filtro ai log prima che vengano emessi. CloudWatch Fluentbit lo definisce nella sezione. `[FILTER]` Ad esempio, filtrare i metadati di Kubernetes per evitare che vengano aggiunti agli eventi di registro può ridurre il volume dei log.

```
    [FILTER]
        Name                nest
        Match               application.*
        Operation           lift
        Nested_under        kubernetes
        Add_prefix          Kube.

    [FILTER]
        Name                modify
        Match               application.*
        Remove              Kube.<Metadata_1>
        Remove              Kube.<Metadata_2>
        Remove              Kube.<Metadata_3>

    [FILTER]
        Name                nest
        Match               application.*
```

```
        Operation           nest
        Wildcard            Kube.*
        Nested_under        kubernetes
        Remove_prefix       Kube.
```

Metriche

Le metriche forniscono informazioni preziose sulle prestazioni del sistema. Consolidando tutte le metriche delle risorse relative al sistema o disponibili in una posizione centralizzata, si ottiene la capacità di confrontare e analizzare i dati sulle prestazioni. Questo approccio centralizzato consente di prendere decisioni strategiche più informate, come aumentare o ridurre le risorse. Inoltre, le metriche svolgono un ruolo cruciale nella valutazione dello stato delle risorse, consentendoti di adottare misure proattive quando necessario. In genere, i costi di osservabilità scalano con la raccolta e la conservazione dei dati di telemetria. Di seguito sono riportate alcune strategie che è possibile implementare per ridurre il costo della telemetria metrica: raccogliere solo le metriche importanti, ridurre la cardinalità dei dati di telemetria e ottimizzare la granularità della raccolta dei dati di telemetria.

Monitora ciò che conta e raccogli solo ciò di cui hai bisogno

La prima strategia di riduzione dei costi consiste nel ridurre il numero di metriche raccolte e, a sua volta, ridurre i costi di conservazione.

1. Iniziate a lavorare a ritroso rispetto ai vostri requisiti e/o a quelli dei vostri stakeholder per determinare le metriche più importanti. Le metriche di successo sono diverse per tutti! Scopri com'è bello e misuralo.
2. Valuta la possibilità di approfondire i carichi di lavoro che stai supportando e identificandone i Key Performance Indicators (KPIs), noti anche come «Golden Signals». Questi dovrebbero essere in linea con i requisiti aziendali e delle parti interessate. Il calcolo e SLIs l' SLOs SLAs utilizzo di Amazon CloudWatch e Metric Math sono fondamentali per la gestione dell'affidabilità del servizio. Segui le migliori pratiche descritte in questa guida per monitorare e mantenere efficacemente le prestazioni del tuo ambiente EKS.
3. Quindi continua attraverso i diversi livelli dell'infrastruttura per connettere e correlare i parametri del cluster, del nodo e dell'infrastruttura EKS aggiuntivi al tuo carico di lavoro. KPIs Archivia le metriche aziendali e le metriche operative in un sistema in cui è possibile correlarle tra loro e trarre conclusioni sulla base degli impatti osservati su entrambe.
4. EKS espone le metriche provenienti dal piano di controllo, dal cluster kube-state-metrics, dai pod e dai nodi. La rilevanza di tutte queste metriche dipende dalle tue esigenze, tuttavia è probabile

che non ti serva ogni singola metrica nei diversi livelli. Puoi utilizzare questa guida alle metriche essenziali di EKS come base per monitorare lo stato generale di un cluster EKS e dei tuoi carichi di lavoro.

Ecco un esempio di configurazione di prometheus scrape in cui utilizziamo le metriche per mantenere solo le metriche di Kubelet e `relabel_config` per eliminare tutte le metriche dei container. `metric_relabel_config`

```
  kubernetes_sd_configs:
  - role: endpoints
    namespaces:
      names:
      - kube-system
  bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
  tls_config:
    insecure_skip_verify: true
  relabel_configs:
  - source_labels: [__meta_kubernetes_service_label_k8s_app]
    regex: kubelet
    action: keep

  metric_relabel_configs:
  - source_labels: [__name__]
    regex: container_(network_tcp_usage_total|network_udp_usage_total|tasks_state|
cpu_load_average_10s)
    action: drop
```

Riduci la cardinalità ove applicabile

La cardinalità si riferisce all'unicità dei valori dei dati in combinazione con le relative dimensioni (ad esempio le etichette di Prometheus) per uno specifico set di metriche. Le metriche ad alta cardinalità hanno molte dimensioni e ogni combinazione di metriche dimensionali ha una maggiore unicità. Una cardinalità più elevata comporta un aumento delle dimensioni dei dati di telemetria metrica e delle esigenze di archiviazione, con un conseguente aumento dei costi.

Nell'esempio ad alta cardinalità riportato di seguito, vediamo che Metric, Latency, ha Dimensions, RequestId, CustomerId e Service e ogni dimensione ha molti valori univoci. La cardinalità è la misura della combinazione del numero di valori possibili per dimensione. In Prometheus, ogni set di dimensioni/etichette uniche viene considerato come una nuova metrica, quindi una cardinalità elevata significa più metriche.

Negli ambienti EKS con molte metriche e dimensioni/etichette per metrica (Cluster, Namespace, Service, Pod, Container, ecc.), la cardinalità tende a crescere. Per ottimizzare i costi, considera attentamente la cardinalità delle metriche che stai raccogliendo. Ad esempio, se stai aggregando una metrica specifica per la visualizzazione a livello di cluster, puoi eliminare etichette aggiuntive che si trovano su un livello inferiore, come l'etichetta del namespace.

Per identificare le metriche ad alta cardinalità in Prometheus, puoi eseguire la seguente query PROMQL per determinare quali destinazioni di scrape hanno il maggior numero di metriche (cardinalità):

```
topk_max(5, max_over_time(scrape_samples_scraped[1h]))
```

e la seguente query PROMQL può aiutarti a determinare quali target di scrape hanno i tassi di abbandono delle metriche più elevati (quante nuove serie di metriche sono state create in un determinato scrape):

```
topk_max(5, max_over_time(scrape_series_added[1h]))
```

Se utilizzi Grafana, puoi utilizzare Mimirtool di Grafana Lab per analizzare i dashboard di Grafana e le regole di Prometheus per identificare metriche ad alta cardinalità non utilizzate. Segui questa guida su come utilizzare i comandi and per identificare le metriche attive a cui non viene fatto riferimento nelle `mimirtool analyze` dashboard`mimirtool analyze prometheus`.

Prendi in considerazione la granularità delle metriche

La raccolta di metriche con una granularità più elevata, ad esempio ogni secondo anziché ogni minuto, può avere un grande impatto sulla quantità di telemetria raccolta e archiviata, con un conseguente aumento dei costi. Stabilisci intervalli ragionevoli per la raccolta dei dati o delle metriche che offrano un equilibrio tra una granularità sufficiente per rilevare problemi transitori e un livello sufficientemente basso da garantire un ottimo rapporto qualità-prezzo. Riduci la granularità delle metriche utilizzate per la pianificazione della capacità e l'analisi di finestre temporali più ampie.

Di seguito è riportato un frammento della configurazione predefinita di AWS Distro for Opentelemetry (ADOT) EKS Addon Collector.

> **Important**
>
> l'intervallo globale di scrape di Prometheus è impostato su 15 secondi. Questo intervallo di scraping può essere aumentato con conseguente diminuzione della quantità di dati metrici raccolti in Prometheus.

```
apiVersion: opentelemetry.io/v1alpha1
kind: OpenTelemetryCollector
metadata:
  name: my-collector-amp

...

config: |
    extensions:
      sigv4auth:
        region: "+++<YOUR_AWS_REGION>+++" service: "aps"+++</YOUR_AWS_REGION>+++

 receivers:
   #
   # Scrape configuration for the Prometheus Receiver
   # This is the same configuration used when Prometheus is installed using the
 community Helm chart
   #
   prometheus:
     config:
       global:   scrape_interval: 15s
         scrape_timeout: 10s
```

Tracciamento

Il costo principale associato al tracciamento deriva dalla generazione dello storage delle tracce. Con il tracciamento, l'obiettivo è raccogliere dati sufficienti per diagnosticare e comprendere gli aspetti prestazionali. Tuttavia, poiché i costi delle tracce a raggi X si basano sui dati inoltrati a X-Ray, la cancellazione delle tracce dopo l'inoltro non ridurrà i costi. Esaminiamo i modi per ridurre i costi di tracciamento conservando al contempo i dati per consentirvi di eseguire un'analisi adeguata.

Applica le regole di campionamento

La frequenza di campionamento dei raggi X è conservativa per impostazione predefinita. Definite le regole di campionamento che consentono di controllare la quantità di dati raccolti. Ciò migliorerà l'efficienza delle prestazioni riducendo al contempo i costi. Riducendo la frequenza di campionamento, è possibile raccogliere tracce dalla richiesta solo per ciò di cui hanno bisogno i carichi di lavoro, mantenendo al contempo una struttura a costi inferiori.

Ad esempio, avete un'applicazione java di cui volete eseguire il debug delle tracce di tutte le richieste per 1 route problematica.

Configura tramite l'SDK per caricare le regole di campionamento da un documento JSON

```
{
"version": 2,
  "rules": [
    {
"description": "debug-eks",
      "host": "*",
      "http_method": "PUT",
      "url_path": "/history/*",
      "fixed_target": 0,
      "rate": 1,
      "service_type": "debug-eks"
    }
  ],
  "default": {
"fixed_target": 1,
    "rate": 0.1
  }
}
```

Tramite la console

Applica Tail Sampling con AWS Distro for OpenTelemetry (ADOT)

ADOT Tail Sampling consente di controllare il volume di tracce inserite nel servizio. Tuttavia, Tail Sampling consente di definire le politiche di campionamento dopo che tutti gli intervalli della richiesta sono stati completati anziché all'inizio. Ciò limita ulteriormente la quantità di dati grezzi trasferiti CloudWatch, riducendo così i costi.

Ad esempio, se stai campionando l'1% del traffico verso una pagina di destinazione e il 10% delle richieste verso una pagina di pagamento, potresti lasciare 300 tracce per un periodo di 30 minuti. Con una regola ADOT Tail Sampling che filtra errori specifici, potresti rimanere con 200 tracce, il che riduce il numero di tracce memorizzate.

```
processors:
  groupbytrace:
    wait_duration: 10s
    num_traces: 300
    tail_sampling:
    decision_wait: 1s # This value should be smaller than wait_duration
    policies:
      - ..... # Applicable policies**
  batch/tracesampling:
    timeout: 0s # No need to wait more since this will happen in previous processors
    send_batch_max_size: 8196 # This will still allow us to limit the size of the
 batches sent to subsequent exporters

service:
  pipelines:
    traces/tailsampling:
      receivers: [otlp]
      processors: [groupbytrace, tail_sampling, batch/tracesampling]
      exporters: [awsxray]
```

Sfrutta le opzioni di storage di Amazon S3

È necessario sfruttare il bucket AWS S3 e le sue diverse classi di storage per archiviare le tracce. Esporta le tracce su S3 prima della scadenza del periodo di conservazione. Utilizza le regole del ciclo di vita di Amazon S3 per spostare i dati di traccia nella classe di storage che soddisfa i tuoi requisiti.

Ad esempio, se hai tracce che risalgono a 90 giorni fa, Amazon S3 Intelligent-Tiering può spostare automaticamente i dati in uno storage a lungo termine in base al tuo modello di utilizzo. Puoi usare Amazon Athena per interrogare i dati in Amazon S3 se devi fare riferimento alle tracce in un secondo momento. Ciò può ridurre ulteriormente i costi per la tracciabilità distribuita.

Risorse aggiuntive:

- Guida alle migliori pratiche in materia di osservabilità
- Raccolta di metriche sulle migliori pratiche

- AWS re:Invent 2022 - Best practice di osservabilità su Amazon () COP343
- AWS re:Invent 2022 - Osservabilità: best practice per applicazioni moderne () COP344

Procedure consigliate per Windows

Questa guida fornisce consigli sull'esecuzione di contenitori e nodi Windows.

Argomenti

- Gestione delle AMI Windows ottimizzate per Amazon EKS
- Configurare gMSA per Windows Pods e contenitori
- Rafforzamento dei nodi di lavoro di Windows
- Scansione delle immagini dei contenitori
- Versione e licenza di Windows Server
- Registrazione dei log
- Monitoraggio
- Rete Windows
- Evitare gli errori OOM
- Applicazione di patch a server e contenitori Windows
- Esecuzione di carichi di lavoro eterogenei
- Contesti di sicurezza Pod
- Opzioni di archiviazione persistente
- Rafforzamento delle immagini dei contenitori Windows

Gestione delle AMI Windows ottimizzate per Amazon EKS

Windows Amazon EKS AMIs optimized si basa su Windows Server 2019 e Windows Server 2022. Sono configurate per fungere da immagine di base per i nodi Amazon EKS. Per impostazione predefinita, AMIs includono i seguenti componenti:

- kubelet
- kube-proxy
- AWS IAM Authenticator per Kubernetes
- csi-proxy
- containerd

Puoi recuperare a livello di codice l'ID Amazon Machine Image (AMI) per Amazon EKS ottimizzato AMIs interrogando l'API AWS Systems Manager Parameter Store. Questo parametro elimina la necessità di cercare manualmente le AMI ottimizzate per Amazon EKS IDs. Per ulteriori informazioni sull'API Systems Manager Parameter Store, vedere GetParameter. Il tuo account utente deve disporre dell'autorizzazione ssm: GetParameter IAM per recuperare i metadati AMI ottimizzati per Amazon EKS.

L'esempio seguente recupera l'ID AMI per l'ultima AMI ottimizzata Amazon EKS per Windows Server 2019 LTSC Core. Il numero di versione riportato nel nome dell'AMI si riferisce alla build Kubernetes corrispondente per cui è stato preparato.

```
aws ssm get-parameter --name /aws/service/ami-windows-latest/Windows_Server-2019-
English-Core-EKS_Optimized-1.21/image_id --region us-east-1 --query "Parameter.Value"
 --output text
```

Output di esempio:

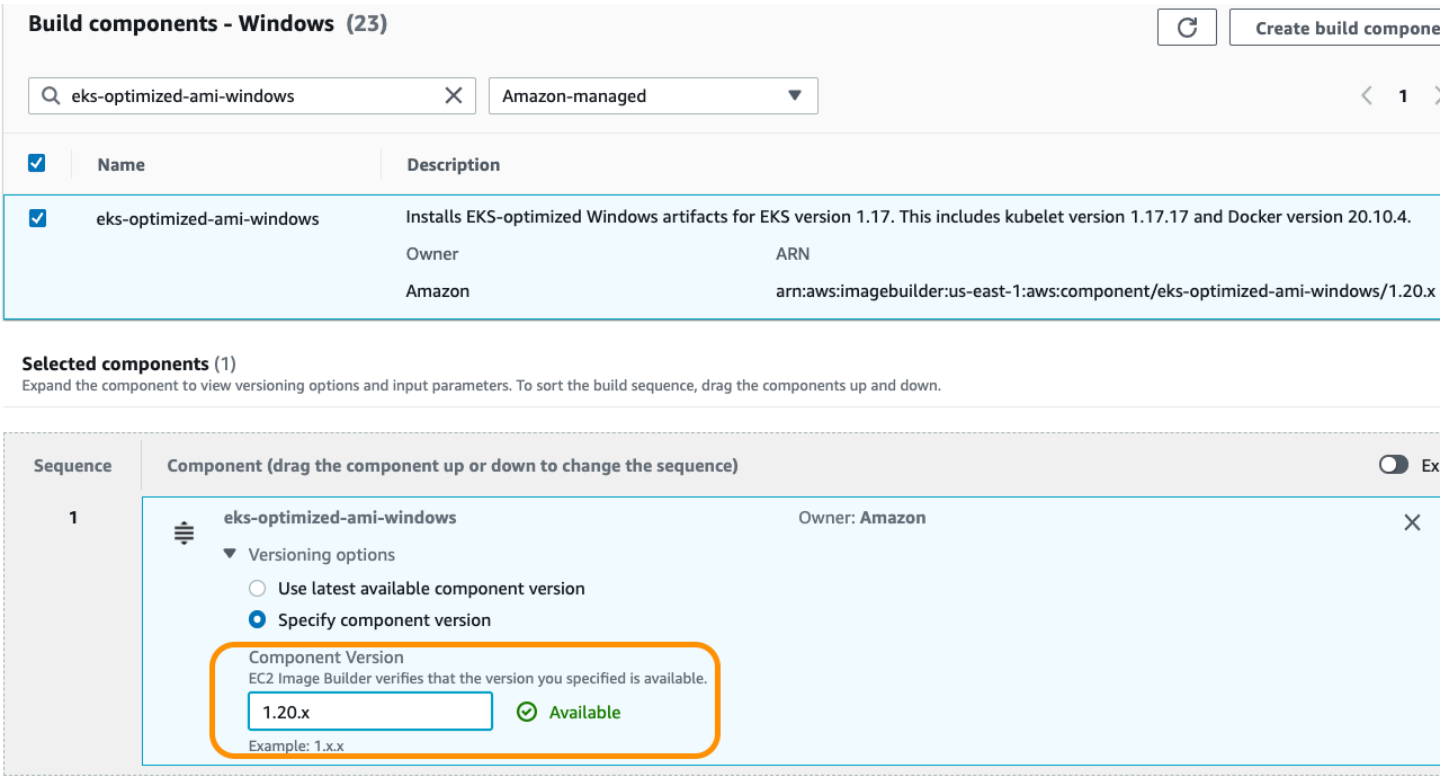
```
ami-09770b3eec4552d4e
```

Gestione della propria AMI Windows ottimizzata per Amazon EKS

Un passo essenziale verso gli ambienti di produzione è mantenere la stessa AMI Windows ottimizzata per Amazon EKS e la stessa versione kubelet in tutto il cluster Amazon EKS.

L'utilizzo della stessa versione in tutto il cluster Amazon EKS riduce i tempi di risoluzione dei problemi e aumenta la coerenza del cluster. Amazon EC2 Image Builder aiuta a creare e gestire AMIs finestre personalizzate ottimizzate per Amazon EKS da utilizzare in un cluster Amazon EKS.

Usa Amazon EC2 Image Builder per scegliere tra le versioni di Windows Server, le date and/or di rilascio dell'AMI AWS Windows Server e la versione di build del sistema operativo. La fase di compilazione dei componenti consente di scegliere tra gli artefatti Windows ottimizzati per EKS esistenti e le versioni kubelet. Per ulteriori informazioni: - .html https://docs.aws.amazon.com/eks/latest/userguide/eks custom-ami-windows



NOTA: prima di selezionare un'immagine di base, consulta la sezione Versione e licenza di Windows Server per dettagli importanti relativi agli aggiornamenti del canale di rilascio.

Configurazione di un avvio più rapido per EKS personalizzato ottimizzato AMIs

Quando si utilizza un'AMI personalizzata ottimizzata per Windows Amazon EKS, i nodi di lavoro Windows possono essere avviati fino al 65% più velocemente abilitando la funzionalità Fast Launch. Questa funzionalità mantiene una serie di istantanee preconfigurate che prevedono la specializzazione in Sysprep, i passaggi Windows Out of Box Experience (OOBE) e i riavvii richiesti già completati. Queste istantanee vengono quindi utilizzate negli avvii successivi, riducendo il tempo necessario per la scalabilità orizzontale o la sostituzione dei nodi. Fast Launch può essere abilitato solo per conto AMIs tuo tramite la console EC2 o nella CLI di AWS e il numero di istantanee gestite è configurabile.

NOTA: Fast Launch non è compatibile con l'AMI ottimizzata EKS predefinita fornita da Amazon, crea un'AMI personalizzata come sopra prima di tentare di abilitarla.

Per ulteriori informazioni: AWS Windows AMIs : configura la tua AMI per un avvio più rapido

Memorizzazione nella cache dei livelli base di Windows su livelli personalizzati AMIs

Le immagini dei contenitori Windows sono più grandi delle loro controparti Linux. Se si esegue un'applicazione containerizzata basata su .NET Framework, la dimensione media dell'immagine è di circa 8,24 GB. Durante la pianificazione del pod, l'immagine del contenitore deve essere completamente estratta ed estratta dal disco prima che il pod raggiunga lo stato In esecuzione.

Durante questo processo, il runtime del contenitore (containerd) recupera ed estrae l'intera immagine del contenitore nel disco. L'operazione pull è un processo parallelo, il che significa che il runtime del contenitore attira i livelli dell'immagine del contenitore in parallelo. Al contrario, l'operazione di estrazione avviene in un processo sequenziale ed è I/O intensivo. Per questo motivo, l'immagine del contenitore può impiegare più di 8 minuti per essere completamente estratta e pronta per essere utilizzata dal runtime del contenitore (containerd) e, di conseguenza, l'avvio del pod può richiedere diversi minuti.

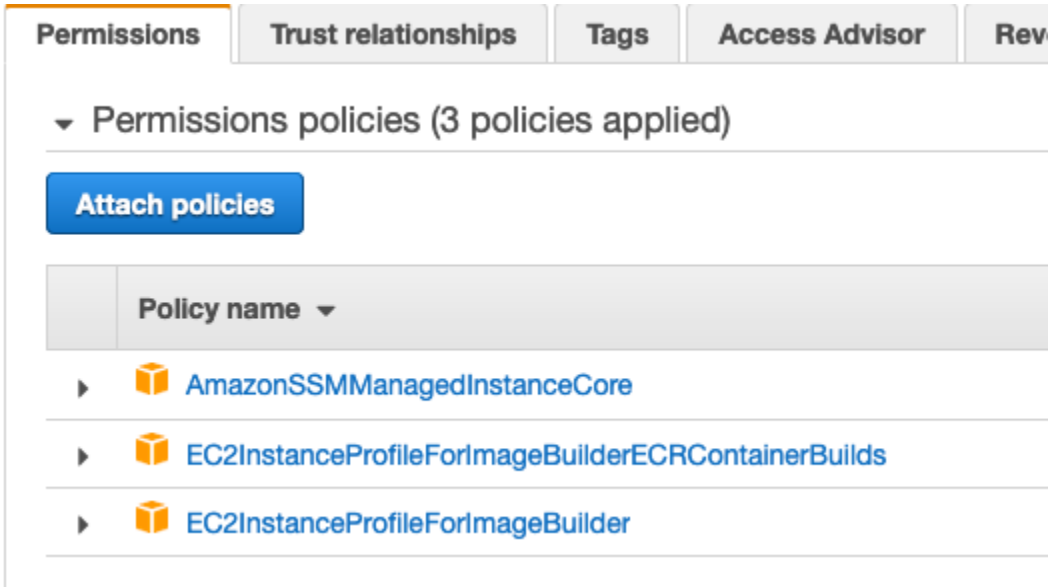
Come indicato nell'argomento Applicazione di patch a Windows Server and Container, esiste un'opzione per creare un'AMI personalizzata con EKS. Durante la preparazione dell'AMI, puoi aggiungere un componente aggiuntivo EC2 Image Builder per estrarre localmente tutte le immagini dei container Windows necessarie e quindi generare l'AMI. Questa strategia ridurrà drasticamente il tempo in cui un pod raggiunge lo stato In esecuzione.

Su Amazon EC2 Image Builder, crea un componente per scaricare le immagini necessarie e allegalo alla ricetta Image. L'esempio seguente estrae un'immagine specifica da un repository ECR.

```
name: ContainerdPull
description: This component pulls the necessary containers images for a cache strategy.
schemaVersion: 1.0

phases:
  - name: build
    steps:
      - name: containerdpull
        action: ExecutePowerShell
        inputs:
          commands:
            - Set-ExecutionPolicy Unrestricted -Force
            - (Get-ECRLoginCommand).Password | docker login --username AWS --password-
stdin 111000111000.dkr.ecr.us-east-1.amazonaws.com
            - ctr image pull mcr.microsoft.com/dotnet/framework/aspnet:latest
```

```
            - ctr image pull 111000111000.dkr.ecr.us-east-1.amazonaws.com/
myappcontainerimage:latest
```

Per assicurarti che il seguente componente funzioni come previsto, controlla se il ruolo IAM utilizzato da EC2 Image builder (EC2InstanceProfileForImageBuilder) ha le policy allegate:



Post del blog

Nel seguente post del blog, troverai istruzioni dettagliate su come implementare la strategia di caching per Windows AMIs Amazon EKS personalizzati:

Accelerazione dei tempi di avvio dei container Windows con EC2 Image Builder e la strategia Image Cache

Configurare gMSA per Windows Pods e contenitori

Cos'è un account gMSA

Le applicazioni basate su Windows, ad esempio le applicazioni.NET, utilizzano spesso Active Directory come provider di identità, authorization/authentication utilizzando il protocollo NTLM o Kerberos.

Un server di applicazioni per lo scambio di ticket Kerberos con Active Directory richiede l'aggiunta a un dominio. I contenitori Windows non supportano i join di dominio e non avrebbero molto senso in quanto i contenitori sono risorse effimere, che gravano sul pool RID di Active Directory.

Tuttavia, gli amministratori possono sfruttare gli account gMSA Active Directory per negoziare un'autenticazione Windows per risorse come contenitori Windows, NLB e server farm.

Caso d'uso di Windows container e gMSA

Le applicazioni che sfruttano l'autenticazione Windows e vengono eseguite come contenitori Windows traggono vantaggio da gMSA perché il nodo Windows viene utilizzato per lo scambio del ticket Kerberos per conto del contenitore. Sono disponibili due opzioni per configurare il nodo di lavoro Windows per supportare l'integrazione gMSA:

1 - Nodi di lavoro Windows aggiunti al dominio

In questa configurazione, il nodo di lavoro Windows viene aggiunto al dominio Active Directory e l'account AD Computer dei nodi di lavoro Windows viene utilizzato per l'autenticazione con Active Directory e recuperare l'identità gMSA da utilizzare con il pod.

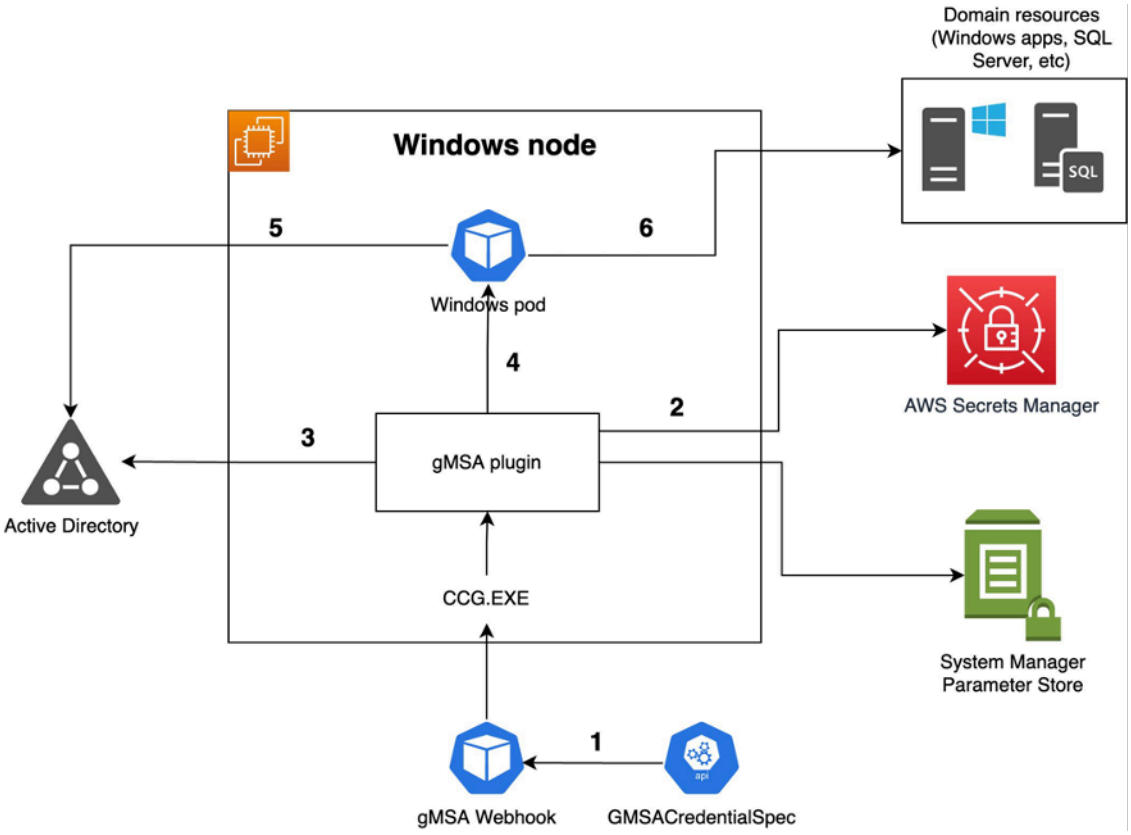
Nell'approccio integrato al dominio, è possibile gestire e potenziare facilmente i nodi di lavoro Windows utilizzando Active Directory esistente GPOs; tuttavia, genera ulteriori sovraccarichi operativi e ritardi durante l'aggiunta del nodo di lavoro di Windows al cluster Kubernetes, poiché richiede riavvii aggiuntivi durante l'avvio del nodo e la pulizia del garage di Active Directory dopo la chiusura dei nodi da parte del cluster Kubernetes.

Nel seguente post del blog, troverai informazioni dettagliate step-by-step su come implementare l'approccio del nodo di lavoro Windows unito al dominio:

Autenticazione Windows sui pod Windows di Amazon EKS

2 - Nodi di lavoro Windows senza dominio

In questa configurazione, il nodo di lavoro di Windows non fa parte del dominio Active Directory e viene utilizzata un'identità «portatile» (utente/password) per l'autenticazione con Active Directory e recuperare l'identità gMSA da utilizzare con il pod.



L'identità portatile è un utente di Active Directory; l'identità (user/password) è archiviata su AWS Secrets Manager o AWS System Manager Parameter Store e un plug-in sviluppato da AWS chiamato ccg_plugin verrà utilizzato per recuperare questa identità da AWS Secrets Manager o AWS System Manager Parameter Store e passarla a containerd per recuperare l'identità gMSA e renderla disponibile per il pod.

In questo approccio senza dominio, puoi trarre vantaggio dall'assenza di alcuna interazione con Active Directory durante l'avvio del nodo di lavoro di Windows quando usi gMSA e dalla riduzione del sovraccarico operativo per gli amministratori di Active Directory.

Nel seguente post del blog, troverai informazioni dettagliate step-by-step su come implementare l'approccio del nodo di lavoro Windows senza dominio:

Autenticazione Windows senza dominio per i pod Windows di Amazon EKS

Nota importante

Nonostante il pod sia in grado di utilizzare un account gMSA, è necessario configurare anche l'applicazione o il servizio di conseguenza per supportare l'autenticazione di Windows, ad esempio,

per configurare Microsoft IIS per supportare l'autenticazione di Windows, è necessario prepararlo tramite dockerfile:

```
RUN Install-WindowsFeature -Name Web-Windows-Auth -IncludeAllSubFeature
RUN Import-Module WebAdministration; Set-ItemProperty 'IIS:\AppPools\SiteName' -name
 processModel.identityType -value 2
RUN Import-Module WebAdministration; Set-WebConfigurationProperty -Filter '/
system.webServer/security/authentication/anonymousAuthentication' -Name Enabled -Value
 False -PSPath 'IIS:\' -Location 'SiteName'
RUN Import-Module WebAdministration; Set-WebConfigurationProperty -Filter '/
system.webServer/security/authentication/windowsAuthentication' -Name Enabled -Value
 True -PSPath 'IIS:\' -Location 'SiteName'
```

Rafforzamento dei nodi di lavoro di Windows

Il rafforzamento del sistema operativo è una combinazione di configurazione del sistema operativo, applicazione di patch e rimozione di pacchetti software non necessari, che mira a bloccare un sistema e ridurre la superficie di attacco. È consigliabile preparare una propria AMI Windows ottimizzata per EKS con le configurazioni di rafforzamento richieste dalla propria azienda.

AWS fornisce ogni mese una nuova AMI Windows ottimizzata per EKS contenente le ultime patch di sicurezza di Windows Server. Tuttavia, è ancora responsabilità dell'utente rafforzare la propria AMI applicando le configurazioni del sistema operativo necessarie indipendentemente dal fatto che utilizzi gruppi di nodi autogestiti o gestiti.

Microsoft offre una gamma di strumenti come Microsoft Security Compliance Toolkit e Security Baselines che ti aiutano a ottenere un rafforzamento in base alle tue esigenze di policy di sicurezza. Sono disponibili anche benchmark CIS che devono essere implementati su un'AMI Windows ottimizzata per Amazon EKS per ambienti di produzione.

Riduzione della superficie di attacco con Windows Server Core

Windows Server Core è un'opzione di installazione minima disponibile come parte dell'AMI Windows ottimizzata EKS. La distribuzione di Windows Server Core offre un paio di vantaggi. Innanzitutto, ha un ingombro su disco relativamente ridotto: 6 GB su Server Core contro 10 GB su Windows Server con esperienza Desktop. In secondo luogo, ha una superficie di attacco più piccola grazie alla sua base di codice più piccola e disponibile. APIs

AWS fornisce ai clienti nuove finestre ottimizzate per Amazon EKS AMIs ogni mese, contenenti le ultime patch di sicurezza Microsoft, indipendentemente dalla versione supportata da Amazon EKS.

Come best practice, i nodi di lavoro Windows devono essere sostituiti con nuovi nodi basati sulla più recente AMI ottimizzata per Amazon EKS. Qualsiasi nodo in esecuzione per più di 45 giorni senza un aggiornamento o una sostituzione del nodo non dispone delle migliori pratiche di sicurezza.

Evitare le connessioni RDP

Remote Desktop Protocol (RDP) è un protocollo di connessione sviluppato da Microsoft per fornire agli utenti un'interfaccia grafica per connettersi a un altro computer Windows tramite una rete.

Come procedura consigliata, è consigliabile trattare i nodi di lavoro di Windows come se fossero host temporanei. Ciò significa nessuna connessione di gestione, nessun aggiornamento e nessuna risoluzione dei problemi. Qualsiasi modifica e aggiornamento deve essere implementato come una nuova AMI personalizzata e sostituita dall'aggiornamento di un gruppo Auto Scaling. Vedi Patching di server e contenitori Windows e gestione delle AMI Windows ottimizzate per Amazon EKS.

Disabilita le connessioni RDP sui nodi Windows durante la distribuzione passando il valore false sulla proprietà ssh, come nell'esempio seguente:

```
nodeGroups:
- name: windows-ng
  instanceType: c5.xlarge
  minSize: 1
  volumeSize: 50
  amiFamily: WindowsServer2019CoreContainer
  ssh:
    allow: false
```

Se è necessario accedere al nodo Windows, utilizza AWS System Manager Session Manager per stabilire una PowerShell sessione sicura tramite la console AWS e l'agente SSM. Per scoprire come implementare la soluzione, guarda Accesso sicuro alle istanze Windows utilizzando AWS Systems Manager Session Manager.

Per utilizzare System Manager Session Manager, è necessario applicare una policy IAM aggiuntiva al ruolo IAM utilizzato per avviare il nodo di lavoro Windows. Di seguito è riportato un esempio in cui Amazon SSMManaged InstanceCore è specificato nel manifesto del `eksctl` cluster:

```
 nodeGroups:
- name: windows-ng
  instanceType: c5.xlarge
```

```
  minSize: 1
  volumeSize: 50
  amiFamily: WindowsServer2019CoreContainer
  ssh:
    allow: false
  iam:
    attachPolicyARNs:
      - arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy
      - arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy
      - arn:aws:iam::aws:policy/ElasticLoadBalancingFullAccess
      - arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly
      - arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore
```

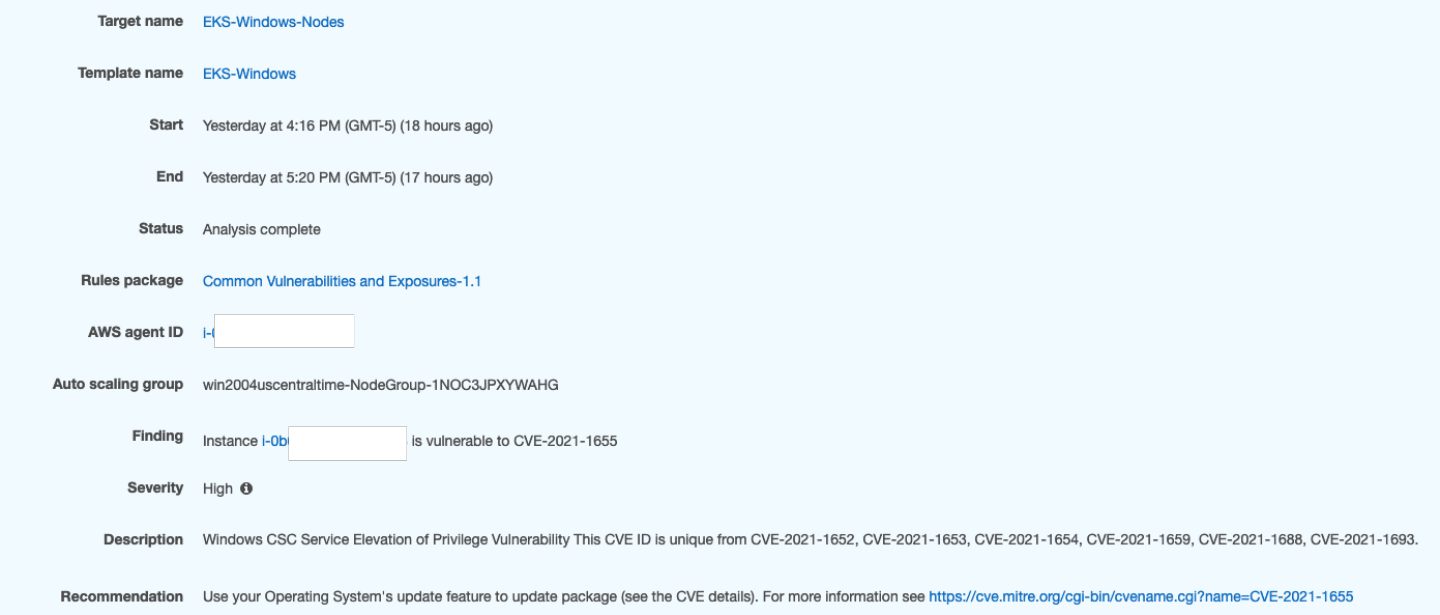
Amazon Inspector

> Amazon Inspector è un servizio di valutazione della sicurezza automatizzato che aiuta a migliorare la sicurezza e la conformità delle applicazioni distribuite su AWS. Amazon Inspector valuta automaticamente le applicazioni in base all'esposizione, alle vulnerabilità e alle deviazioni dalle best practice. Dopo aver eseguito una valutazione, Amazon Inspector produce un elenco dettagliato di risultati di sicurezza con priorità in base al livello di gravità. Questi risultati possono essere esaminati direttamente o come parte di report di valutazione dettagliati disponibili tramite la console o l'API di Amazon Inspector.

Amazon Inspector può essere utilizzato per eseguire la valutazione CIS Benchmark sul nodo di lavoro di Windows e può essere installato su un Windows Server Core eseguendo le seguenti attività:

1. Scarica il seguente file.exe: .exe https://inspector-agent.amazonaws.com/windows/ installer/latest/ AWSAgentInstall
2. Trasferisci l'agente nel nodo di lavoro di Windows.
3. Esegui il seguente comando PowerShell per installare l'agente Amazon Inspector: `. \AWSAgentInstall.exe /install`

Di seguito è riportato l'output dopo la prima esecuzione. Come puoi vedere, ha generato risultati basati sul database CVE. Puoi usarlo per rafforzare i tuoi nodi Worker o creare un'AMI basata sulle configurazioni rinforzate.



Per ulteriori informazioni su Amazon Inspector, incluso come installare gli agenti Amazon Inspector, configurare la valutazione CIS Benchmark e generare report, guarda il video Miglioramento della sicurezza e della conformità dei carichi di lavoro di Windows con Amazon Inspector.

Amazon GuardDuty

> Amazon GuardDuty è un servizio di rilevamento delle minacce che monitora continuamente attività dannose e comportamenti non autorizzati per proteggere gli account AWS, i carichi di lavoro e i dati archiviati in Amazon S3. Con il cloud, la raccolta e l'aggregazione delle attività di account e di rete è semplificata, ma per i team di sicurezza può richiedere molto tempo per analizzare continuamente i dati dei registri degli eventi alla ricerca di potenziali minacce.

Utilizzando Amazon GuardDuty hai visibilità sulle attività dannose contro i nodi di lavoro di Windows, come gli attacchi RDP brute force e Port Probe.

Guarda il GuardDuty video sul rilevamento delle minacce per i carichi di lavoro Windows con Amazon per scoprire come implementare ed eseguire benchmark CIS sull'AMI Windows EKS ottimizzata

Sicurezza in Amazon EC2 per Windows

Leggi le best practice di sicurezza per le istanze Windows di Amazon EC2 per implementare i controlli di sicurezza a ogni livello.

Scansione delle immagini dei contenitori

La scansione delle immagini è una funzionalità di valutazione automatica delle vulnerabilità che aiuta a migliorare la sicurezza delle immagini dei contenitori delle applicazioni scansionandole per rilevare un'ampia gamma di vulnerabilità del sistema operativo.

Attualmente, Amazon Elastic Container Registry (ECR) è in grado di scansionare solo l'immagine del contenitore Linux alla ricerca di vulnerabilità. Tuttavia, esistono strumenti di terze parti che possono essere integrati con una CI/CD pipeline esistente per la scansione delle immagini dei contenitori Windows.

- Anchore
- PaloAlto Nuvola Prisma
- Trend Micro - Controllo intelligente della sicurezza approfondita

Per ulteriori informazioni su come integrare queste soluzioni con Amazon Elastic Container Repository (ECR), consulta:

- Anchore, scansione di immagini su Amazon Elastic Container Registry (ECR)
- PaloAlto, scansione di immagini su Amazon Elastic Container Registry (ECR)
- TrendMicro, scansione di immagini su Amazon Elastic Container Registry (ECR)

Versione e licenza di Windows Server

Versione di Windows Server

Un'AMI Windows ottimizzata per Amazon EKS si basa sulle edizioni Windows Server 2019 e 2022 Datacenter sul Long-Term Servicing Channel (LTSC). La versione Datacenter non ha limiti al numero di contenitori in esecuzione su un nodo di lavoro. Per ulteriori informazioni: https://docs.microsoft.com/en-us/ virtualization/windowscontainers/about/faq

Long-Term Servicing Channel (LTSC)

Precedentemente chiamato «Long-Term Servicing Branch», questo è il modello di rilascio che già conoscete, in cui una nuova versione principale di Windows Server viene rilasciata ogni 2-3 anni. Gli utenti hanno diritto a 5 anni di supporto mainstream e 5 anni di supporto esteso.

Licenze

Quando avvii un'istanza Amazon EC2 con un'AMI basata su Windows Server, Amazon copre i costi di licenza e la conformità delle licenze per te.

Registrazione dei log

Le applicazioni containerizzate in genere indirizzano i log delle applicazioni a STDOUT. Il runtime del contenitore intercetta questi registri e li utilizza, in genere li scrive su un file. La posizione in cui vengono archiviati questi file dipende dal runtime e dalla configurazione del contenitore.

Una differenza fondamentale con i pod Windows è che non generano STDOUT. È possibile eseguire l'operazione LogMonitorper recuperare i registri ETW (Event Tracing for Windows), i registri degli eventi di Windows e altri registri specifici dell'applicazione dall'esecuzione di contenitori Windows e reindirizza l'output di registro formattato a STDOUT. Questi log possono quindi essere trasmessi in streaming utilizzando fluent-bit o fluentd verso la destinazione desiderata, ad esempio Amazon. CloudWatch

Il meccanismo di raccolta dei log recupera i log dai pod Kubernetes. STDOUT/STDERR A DaemonSetè un modo comune per raccogliere i log dai contenitori. Ti dà la possibilità di gestire i log routing/filtering/enrichment indipendentemente dall'applicazione. DaemonSet È possibile utilizzare un fluentd per trasmettere questi registri e qualsiasi altro registro generato dall'applicazione all'aggregatore di log desiderato.

Informazioni più dettagliate sullo streaming dei log dai carichi di lavoro Windows a sono spiegate qui CloudWatch

Raccomandazioni per la registrazione

Le migliori pratiche generali di registrazione non sono diverse quando si utilizzano carichi di lavoro Windows in Kubernetes.

- Registra sempre le voci di registro strutturate (JSON/SYSLOG), il che semplifica la gestione delle voci di registro in quanto esistono molti parser predefiniti per tali formati strutturati.
- Centralizza i log: i contenitori di registrazione dedicati possono essere utilizzati specificamente per raccogliere e inoltrare messaggi di log da tutti i contenitori a una destinazione
- Mantieni bassa la verbosità dei log tranne durante il debug. La verbosità pone molto stress sull'infrastruttura di registrazione e nel rumore si possono perdere eventi significativi.

- Registra sempre le informazioni sull'applicazione insieme all'ID della transazione/richiesta per la tracciabilità. Gli oggetti Kubernetes non hanno il nome dell'applicazione, quindi, ad esempio, il nome di un pod potrebbe non avere alcun significato durante il `windows-twryrqyw` debug dei log. Ciò facilita la tracciabilità e la risoluzione dei problemi delle applicazioni con i log aggregati.

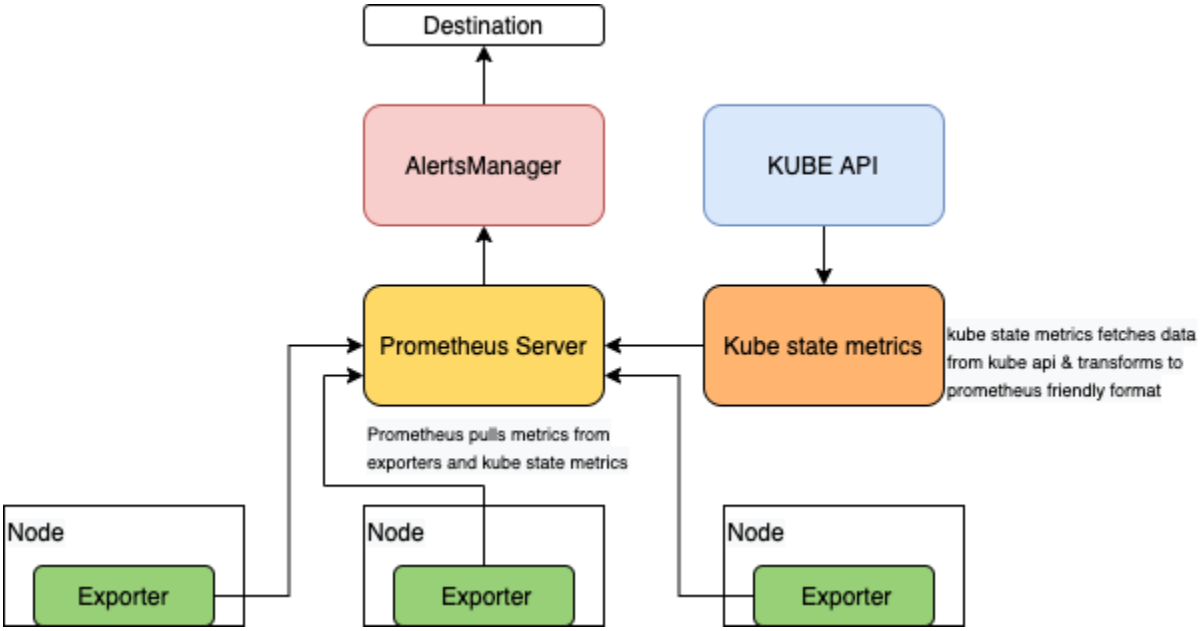
  Il modo in cui vengono generati questi transaction/correlation ID dipende dal costrutto di programmazione. Ma uno schema molto comune consiste nell'utilizzare un Aspect/Interceptor di registrazione, che può utilizzare MDC (Mapped diagnostic context) per iniettare un ID univoco a ogni richiesta in arrivo, in questo modo: transaction/correlation

```
import org.slf4j.MDC;
import java.util.UUID;
Class LoggingAspect { //interceptor

    @Before(value = "execution(* *.*(..))")
    func before(...) {
        transactionId = generateTransactionId();
        MDC.put(CORRELATION_ID, transactionId);
    }

    func generateTransactionId() {
        return UUID.randomUUID().toString();
    }
}
```

Monitoraggio

Prometheus, un progetto CNCF laureato, è di gran lunga il sistema di monitoraggio più popolare con integrazione nativa in Kubernetes. Prometheus raccoglie metriche relative a contenitori, pod, nodi e cluster. Inoltre, Prometheus AlertsManager sfrutta i vantaggi che consentono di programmare avvisi per avvisare l'utente se qualcosa nel cluster sta andando storto. Prometheus memorizza i dati metrici come dati di serie temporali identificati dal nome e dalle coppie di metriche. key/value Prometheus include un modo per eseguire interrogazioni utilizzando un linguaggio chiamato PromQL, che è l'abbreviazione di Prometheus Query Language.

L'architettura di alto livello della raccolta di metriche di Prometheus è mostrata di seguito:



Prometheus utilizza un meccanismo pull e raccoglie le metriche dagli obiettivi utilizzando gli esportatori e dall'API Kubernetes utilizzando le metriche dello stato di kube. Ciò significa che le applicazioni e i servizi devono esporre un endpoint HTTP (S) contenente metriche in formato Prometheus. Prometheus quindi, in base alla sua configurazione, estrarrà periodicamente le metriche da questi endpoint HTTP (S).

Un esportatore ti consente di utilizzare metriche di terze parti come metriche in formato Prometheus. Un esportatore Prometheus viene in genere distribuito su ogni nodo. Per un elenco completo degli esportatori, consulta gli esportatori di Prometheus. Sebbene node exporter sia adatto per esportare l'hardware dell'host e le metriche del sistema operativo per i nodi Linux, non funzionerà per i nodi Windows.

In un cluster EKS a nodi misti con nodi Windows, quando si utilizza la tabella di comando stabile Prometheus, verranno visualizzati i pod non funzionanti sui nodi Windows, poiché questo programma di esportazione non è destinato a Windows. Dovrete trattare il pool di lavoro di Windows separatamente e installare invece Windows Exporter nel gruppo di nodi di lavoro di Windows.

Per configurare il monitoraggio di Prometheus per i nodi Windows, è necessario scaricare e installare l'esportatore WMI sul server Windows stesso e quindi configurare i target all'interno della configurazione scrape del file di configurazione Prometheus. La pagina delle versioni fornisce tutti i programmi di installazione.msi disponibili, con i rispettivi set di funzionalità e correzioni di bug. Il programma di installazione configurerà windows_exporter come servizio Windows, oltre a creare un'eccezione nel firewall di Windows. Se il programma di installazione viene eseguito senza

parametri, l'esportatore verrà eseguito con le impostazioni predefinite per i raccoglitori, le porte e così via abilitati.

Puoi consultare la sezione sulle migliori pratiche di pianificazione di questa guida che suggerisce l'uso taints/tolerations o RuntimeClass la distribuzione selettiva di node Exporter solo su nodi Linux, mentre l'esportatore di Windows viene installato sui nodi Windows durante l'avvio del nodo o utilizzando uno strumento di gestione della configurazione a tua scelta (ad esempio chef, Ansible, SSM ecc.).

Si noti che, a differenza dei nodi Linux in cui l'esportatore di nodi è installato come demonset, nei nodi Windows l'esportatore WMI è installato sull'host stesso. L'esportatore esporterà metriche come l'utilizzo della CPU, della memoria e del disco I/O e può essere utilizzato anche per monitorare i siti e le applicazioni IIS, le interfacce e i servizi di rete.

Per impostazione predefinita, windows_exporter esporrà tutte le metriche dei raccoglitori abilitati. Questo è il metodo consigliato per raccogliere le metriche per evitare errori. Tuttavia, per un uso avanzato, a windows_exporter può essere passato un elenco opzionale di raccoglitori per filtrare le metriche. Il parametro collect [], nella configurazione di Prometheus, consente di farlo.

I passaggi di installazione predefiniti per Windows includono il download e l'avvio dell'esportatore come servizio durante il processo di avvio con argomenti, ad esempio i raccoglitori da filtrare.

```
> Powershell Invoke-WebRequest https://github.com/prometheus-community/
windows_exporter/releases/download/v0.13.0/windows_exporter-0.13.0-amd64.msi -OutFile
 <DOWNLOADPATH>

> msiexec /i <DOWNLOADPATH>
 ENABLED_COLLECTORS="cpu,cs,logical_disk,net,os,system,container,memory"
```

Per impostazione predefinita, è possibile eseguire lo scraping delle metriche nell'endpoint /metrics sulla porta 9182. A questo punto, Prometheus può utilizzare le metriche aggiungendo il seguente scrape_config alla configurazione di Prometheus

```
scrape_configs:
    - job_name: "prometheus"
      static_configs:
        - targets: ['localhost:9090']
    ...
    - job_name: "wmi_exporter"
      scrape_interval: 10s
      static_configs:
```

```
        - targets: ['<windows-node1-ip>:9182', '<windows-node2-ip>:9182', ...]
```

La configurazione di Prometheus viene ricaricata utilizzando

```
> ps aux | grep prometheus
> kill HUP <PID>
```

Un modo migliore e consigliato per aggiungere obiettivi consiste nell'utilizzare una definizione di risorsa personalizzata chiamata ServiceMonitor, che fa parte dell'operatore Prometheus] che fornisce la definizione di ServiceMonitor un oggetto e un controller che attiverà la configurazione di Prometheus richiesta e creerà automaticamente ServiceMonitors la configurazione Prometheus richiesta.

Il ServiceMonitor, che specifica in modo dichiarativo come devono essere monitorati i gruppi di servizi Kubernetes, viene utilizzato per definire un'applicazione da cui desideri acquisire le metriche dall'interno di Kubernetes. All'interno ServiceMonitor specifichiamo le etichette Kubernetes che l'operatore può utilizzare per identificare il servizio Kubernetes, che a sua volta identifica i Pod che desideriamo monitorare.

Per sfruttare il ServiceMonitor, crea un oggetto Endpoint che punti a obiettivi Windows specifici, un servizio headless e un per i nodi Windows. ServiceMontor

```
apiVersion: v1
kind: Endpoints
metadata:
  labels:
    k8s-app: wmiexporter
  name: wmiexporter
  namespace: kube-system
subsets:
- addresses:
  - ip: NODE-ONE-IP
    targetRef:
      kind: Node
      name: NODE-ONE-NAME
  - ip: NODE-TWO-IP
    targetRef:
      kind: Node
      name: NODE-TWO-NAME
  - ip: NODE-THREE-IP
    targetRef:
```

```
      kind: Node
      name: NODE-THREE-NAME
  ports:
  - name: http-metrics
    port: 9182
    protocol: TCP

---
apiVersion: v1
kind: Service ##Headless Service
metadata:
  labels:
    k8s-app: wmiexporter
  name: wmiexporter
  namespace: kube-system
spec:
  clusterIP: None
  ports:
  - name: http-metrics
    port: 9182
    protocol: TCP
    targetPort: 9182
  sessionAffinity: None
  type: ClusterIP

---
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor ##Custom ServiceMonitor Object
metadata:
  labels:
    k8s-app: wmiexporter
  name: wmiexporter
  namespace: monitoring
spec:
  endpoints:
  - interval: 30s
    port: http-metrics
  jobLabel: k8s-app
  namespaceSelector:
    matchNames:
    - kube-system
  selector:
    matchLabels:
```

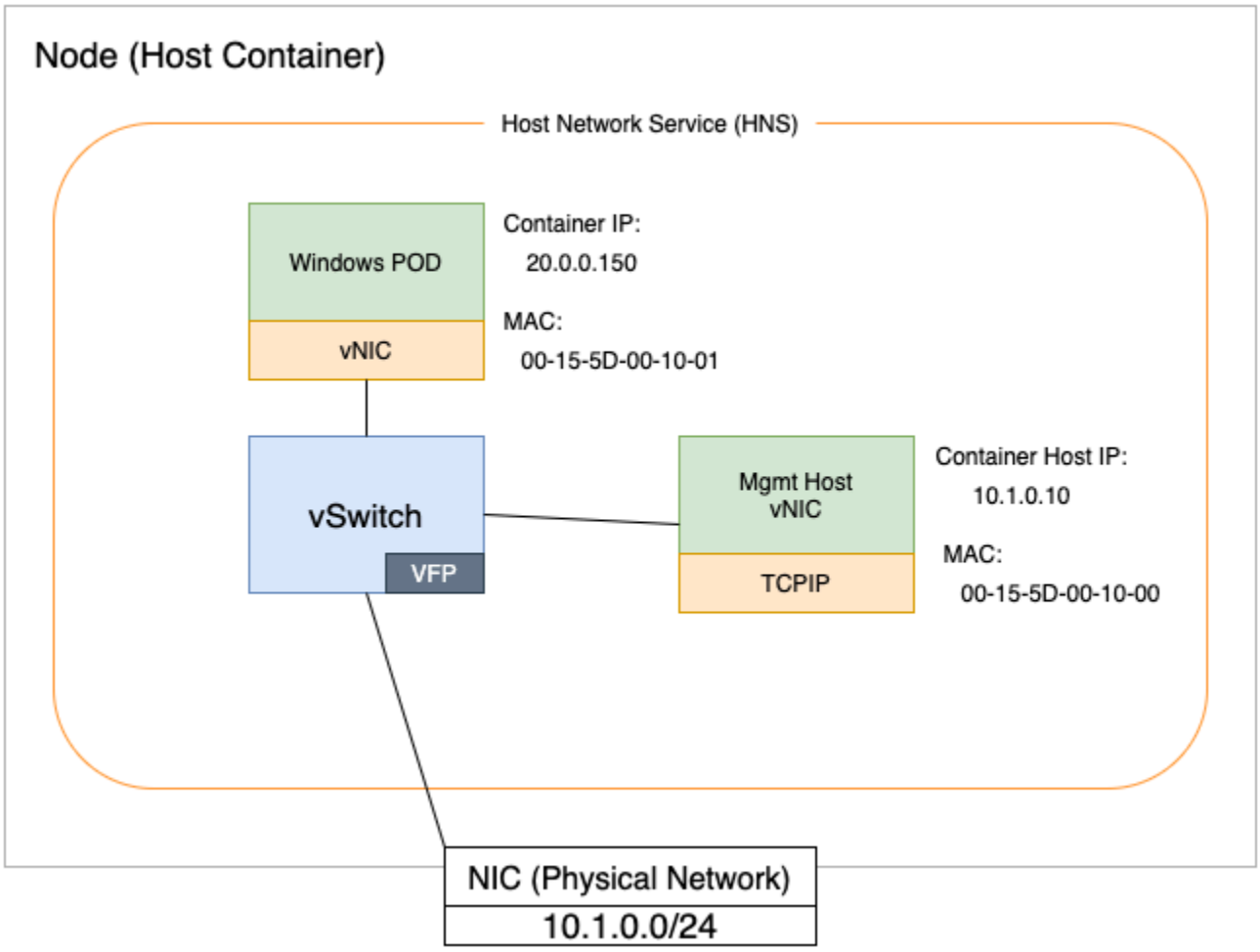
```
      k8s-app: wmiexporter
```

Per maggiori dettagli sull'operatore e sull'utilizzo di ServiceMonitor, consulta la documentazione ufficiale dell'operatore. Nota che Prometheus supporta il rilevamento dinamico degli obiettivi utilizzando molte opzioni di rilevamento dei servizi.

Rete Windows

Panoramica di Windows Container Networking

I contenitori Windows sono fondamentalmente diversi dai contenitori Linux. I contenitori Linux utilizzano costrutti Linux come namespace, file system union e cgroups. In Windows, questi costrutti vengono astratti dai containerd dall'Host Compute Service (HCS). HCS funge da livello API che si colloca al di sopra dell'implementazione del contenitore su Windows. I contenitori Windows sfruttano anche l'Host Network Service (HNS) che definisce la topologia di rete su un nodo.



Dal punto di vista della rete, HCS e HNS fanno funzionare i contenitori Windows come macchine virtuali. Ad esempio, ogni contenitore dispone di un adattatore di rete virtuale (vNIC) collegato a uno switch virtuale Hyper-V (vSwitch) come mostrato nel diagramma precedente.

Gestione degli indirizzi IP

Un nodo in Amazon EKS utilizza la sua Elastic Network Interface (ENI) per connettersi a una rete AWS VPC. Attualmente, è supportato un solo ENI per nodo di lavoro Windows. La gestione degli indirizzi IP per i nodi Windows viene eseguita da VPC Resource Controller che viene eseguito nel piano di controllo. Ulteriori dettagli sul flusso di lavoro per la gestione degli indirizzi IP dei nodi Windows sono disponibili qui.

Il numero di pod che un nodo di lavoro di Windows può supportare dipende dalla dimensione del nodo e dal numero di indirizzi disponibili IPv4 . È possibile calcolare l' IPv4 indirizzo disponibile sul nodo come segue:

- Per impostazione predefinita, all'ENI vengono assegnati solo IPv4 gli indirizzi secondari. In tal caso:

  ```
  Total IPv4 addresses available for Pods = Number of supported IPv4 addresses in the
   primary interface - 1
  ```

  Ne sottraiamo uno dal conteggio totale poiché un IPv4 indirizzo verrà utilizzato come indirizzo principale dell'ENI e quindi non può essere assegnato ai Pod.
- Se il cluster è stato configurato per un'alta densità di pod abilitando la funzione di delega dei prefissi, allora-

  ```
  Total IPv4 addresses available for Pods = (Number of supported IPv4 addresses in the
   primary interface - 1) * 16
  ```

  Qui, invece di allocare IPv4 indirizzi secondari, VPC Resource Controller effettuerà l'`/28` `prefixes`allocazione e, pertanto, il numero complessivo di indirizzi IPv4 disponibili verrà incrementato di 16 volte.

Utilizzando la formula precedente, possiamo calcolare il numero massimo di pod per un nodo di lavoro Windows basato su un'istanza m5.large come segue:

- Per impostazione predefinita, quando si esegue in modalità IP secondaria-

```
10 secondary IPv4 addresses per ENI - 1 = 9 available IPv4 addresses
```

- Quando si utilizza `prefix delegation` -

```
(10 secondary IPv4 addresses per ENI - 1) * 16 = 144 available IPv4 addresses
```

Per ulteriori informazioni sul numero di indirizzi IP che un tipo di istanza può supportare, consulta Indirizzi IP per interfaccia di rete per tipo di istanza.

Un'altra considerazione fondamentale è il flusso del traffico di rete. Con Windows esiste il rischio di esaurimento delle porte sui nodi con più di 100 servizi. Quando si verifica questa condizione, i nodi inizieranno a generare errori con il seguente messaggio:

«Creazione della policy non riuscita: hcnCreateLoad Balancer non riuscito in Win32: la porta specificata esiste già».

Per risolvere questo problema, utilizziamo Direct Server Return (DSR). DSR è un'implementazione della distribuzione asimmetrica del carico di rete. In altre parole, il traffico di richiesta e risposta utilizza percorsi di rete diversi. Questa funzionalità accelera la comunicazione tra i pod e riduce il rischio di esaurimento delle porte. Si consiglia pertanto di abilitare il DSR sui nodi Windows.

Il DSR è abilitato per impostazione predefinita in Windows Server SAC EKS Optimized. AMIs Per Windows Server 2019 LTSC EKS Optimized AMIs, sarà necessario abilitarlo durante il provisioning dell'istanza utilizzando lo script riportato di seguito e utilizzando Windows Server 2019 Full o Core come AMIFamily nel NodeGroup. `eksctl` Vedi eksctl custom AMI per ulteriori informazioni.

```
nodeGroups:
- name: windows-ng
  instanceType: c5.xlarge
  minSize: 1
  volumeSize: 50
  amiFamily: WindowsServer2019CoreContainer
  ssh:
    allow: false
```

Per utilizzare DSR in Windows Server 2019 e versioni successive, è necessario specificare i seguenti flag kube-proxy durante l'avvio dell'istanza. È possibile farlo modificando lo script userdata associato ai gruppi di nodi autogestiti Launch Template.

```
<powershell>
[string]$EKSBinDir = "$env:ProgramFiles\Amazon\EKS"
[string]$EKSBootstrapScriptName = 'Start-EKSBootstrap.ps1'
[string]$EKSBootstrapScriptFile = "$EKSBinDir\$EKSBootstrapScriptName"
(Get-Content $EKSBootstrapScriptFile).replace('"--proxy-mode=kernelspace",', '"--proxy-
mode=kernelspace", "--feature-gates WinDSR=true", "--enable-dsr",') | Set-Content
 $EKSBootstrapScriptFile
& $EKSBootstrapScriptFile -EKSClusterName "eks-windows" -APIServerEndpoint "https://
<REPLACE-EKS-CLUSTER-CONFIG-API-SERVER>" -Base64ClusterCA "<REPLACE-EKSCLUSTER-
CONFIG-DETAILS-CA>" -DNSClusterIP "172.20.0.10" -KubeletExtraArgs "--node-
labels=alpha.eksctl.io/cluster-name=eks-windows,alpha.eksctl.io/nodegroup-name=windows-
ng-ltsc2019 --register-with-taints=" 3>&1 4>&1 5>&1 6>&1
</powershell>
```

L'abilitazione del DSR può essere verificata seguendo le istruzioni nel blog di Microsoft Networking e in Windows Containers on AWS Lab.

```
PS C:\temp> (Get-Content C:\temp\pmsx1eau.0ia\policy.txt | ConvertFrom-JSON).Policies


ExternalPort : 53
InternalPort : 53
IsDSR        : True
Protocol     : 17
Type         : ELB
VIPs         : {172.20.0.10}

ExternalPort : 53
InternalPort : 53
IsDSR        : True
Protocol     : 6
Type         : ELB
VIPs         : {172.20.0.10}

ExternalPort : 443
InternalPort : 443
IsDSR        : True
Protocol     : 6
Type         : ELB
VIPs         : {172.20.0.1}
```

Se preservare IPv4 gli indirizzi disponibili e ridurre al minimo gli sprechi è fondamentale per la sottorete, in genere si consiglia di evitare di utilizzare la modalità di delega dei prefissi, come indicato in Prefix Mode for Windows - When to avoid. Se si desidera continuare a utilizzare la delega dei prefissi, è possibile adottare misure per ottimizzare l'utilizzo degli indirizzi nella sottorete. IPv4 Vedi Configurazione dei parametri per la delega dei prefissi per istruzioni dettagliate su come ottimizzare la richiesta e il processo di allocazione degli indirizzi. IPv4 La regolazione di queste configurazioni

può aiutarti a trovare un equilibrio tra la conservazione degli IPv4 indirizzi e i vantaggi della delega dei prefissi in termini di densità dei pod.

Quando si utilizza l'impostazione predefinita di assegnazione di IPv4 indirizzi secondari, attualmente non esistono configurazioni supportate per manipolare il modo in cui il VPC Resource Controller richiede e alloca gli indirizzi. IPv4 Più specificamente, `warm-ip-target` sono supportate solo per `minimum-ip-target` la modalità di delega dei prefissi. Tieni inoltre presente che in modalità IP secondaria, a seconda degli indirizzi IP disponibili sull'interfaccia, il VPC Resource Controller in genere alloca 3 IPv4 indirizzi inutilizzati sul nodo per tuo conto per mantenerlo caldo IPs per tempi di avvio più rapidi dei pod. Se desideri ridurre al minimo lo spreco di indirizzi IP non utilizzati, potresti pianificare più pod su un determinato nodo Windows in modo da utilizzare quanta più capacità possibile di indirizzi IP dell'ENI. Più esplicitamente, è possibile evitare che non vengano utilizzati a caldo IPs se tutti gli indirizzi IP dell'ENI sono già utilizzati dal nodo e dai pod in esecuzione. Un'altra soluzione alternativa per aiutarti a risolvere i vincoli relativi alla disponibilità degli indirizzi IP nelle tue sottoreti potrebbe essere quella di aumentare le dimensioni della sottorete o separare i nodi Windows nelle rispettive sottoreti dedicate.

Inoltre, è importante notare che al momento non IPv6 è supportato sui nodi Windows.

Opzioni CNI (Container Network Interface)

Il AWSVPC CNI è il plugin CNI de facto per i nodi di lavoro Windows e Linux. Sebbene il AWSVPC CNI soddisfi le esigenze di molti clienti, a volte è necessario prendere in considerazione alternative come una rete overlay per evitare l'esaurimento dell'IP. In questi casi, il CNI Calico può essere utilizzato al posto del CNI. AWSVPC Project Calico è un software open source sviluppato da Tigera. Tale software include un CNI che funziona con EKS. Le istruzioni per l'installazione di Calico CNI in EKS sono disponibili nella pagina di installazione di Project Calico EKS.

Politiche di rete

È considerata una buona pratica passare dalla modalità predefinita di comunicazione aperta tra i pod del cluster Kubernetes alla limitazione dell'accesso in base alle policy di rete. L'open source Project Calico offre un forte supporto per le policy di rete che funzionano con i nodi Linux e Windows. Questa funzionalità è separata e non dipende dall'uso del CNI di Calico. Consigliamo quindi di installare Calico e di utilizzarlo per la gestione delle politiche di rete.

Le istruzioni per l'installazione di Calico in EKS sono disponibili nella pagina Installazione di Calico su Amazon EKS.

Inoltre, i consigli forniti nella sezione Amazon EKS Best Practices Guide for Security - Network si applicano anche ai cluster EKS con nodi di lavoro Windows, tuttavia alcune funzionalità come «Security Groups for Pods» non sono supportate da Windows al momento.

Evitare gli errori OOM

Windows non ha un out-of-memory process killer come Linux. Windows considera sempre tutte le allocazioni di memoria in modalità utente come virtuali e i file di pagina sono obbligatori. L'effetto finale è che Windows non raggiungerà le condizioni di memoria esaurite allo stesso modo di Linux. I processi passeranno da una pagina all'altra su disco anziché essere soggetti alla chiusura della memoria (OOM) in esaurimento della memoria. Se la memoria è sovradimensionata e tutta la memoria fisica è esaurita, il paging può rallentare le prestazioni.

Sistema di prenotazione e memoria Kubelet

A differenza di Linux, dove `--kubelet-reserve` acquisisce la prenotazione delle risorse per i demoni di sistema Kubernetes come kubelet, container runtime, ecc.; e `--system-reserve` acquisisce la prenotazione delle risorse per i demoni di sistema operativo come sshd, udev e così via. In Windows questi flag non acquisiscono e non impostano limiti di memoria su kubelet o sui processi in esecuzione sul nodo.

Tuttavia, puoi combinare questi flag per ridurre la capacità sul nodo con il limite delle risorse di memoria Pod manifest per controllare l'allocazione della memoria per pod. NodeAllocatable Utilizzando questa strategia si ha un migliore controllo dell'allocazione della memoria e un meccanismo di riduzione al minimo out-of-memory (OOM) sui nodi Windows.

Nei nodi Windows, è consigliabile riservare almeno 2 GB di memoria per il sistema operativo e il processo. Utilizzare `--kubelet-reserve` and/or `--system-reserve` per ridurre NodeAllocatable.

Seguendo la documentazione sui nodi Windows autogestiti di Amazon EKS, utilizza il CloudFormation modello per avviare un nuovo gruppo di nodi Windows con personalizzazioni della configurazione kubelet. CloudFormation Ha un elemento chiamato `BootstrapArguments` che è lo stesso di. `KubeletExtraArgs` Da utilizzare con i seguenti flag e valori:

```
--kube-reserved memory=0.5Gi,ephemeral-storage=1Gi --system-
reserved memory=1.5Gi,ephemeral-storage=1Gi --eviction-hard
 memory.available<200Mi,nodefs.available<10%"
```

Se eksctl è lo strumento di distribuzione, consulta la seguente documentazione per personalizzare la configurazione di kubelet https://eksctl. io/usage/customizing-the-kubelet/

Requisiti di memoria per contenitori Windows

Secondo la documentazione Microsoft, un'immagine di base di Windows Server per NANO richiede almeno 30 MB, mentre Server Core richiede 45 MB. Questi numeri aumentano man mano che si aggiungono componenti di Windows come .NET Framework, Web Services come IIS e applicazioni.

È essenziale conoscere la quantità minima di memoria richiesta dall'immagine del contenitore Windows, ovvero l'immagine di base più i relativi livelli di applicazione, e impostarla come quella del contenitore resources/requests nelle specifiche del pod. È inoltre necessario impostare un limite per evitare che i pod consumino tutta la memoria disponibile del nodo in caso di problemi con l'applicazione.

Nell'esempio seguente, quando lo scheduler Kubernetes tenta di posizionare un pod su un nodo, le richieste del pod vengono utilizzate per determinare quale nodo dispone di risorse sufficienti per la pianificazione.

```
spec:
 - name: iis
   image: mcr.microsoft.com/windows/servercore/iis:windowsservercore-ltsc2019
   resources:
     limits:
       cpu: 1
       memory: 800Mi
     requests:
       cpu: .1
       memory: 128Mi
```

Conclusioni

L'utilizzo di questo approccio riduce al minimo i rischi di esaurimento della memoria ma non impedisce che ciò accada. Utilizzando Amazon CloudWatch Metrics, puoi impostare avvisi e correzioni in caso di esaurimento della memoria.

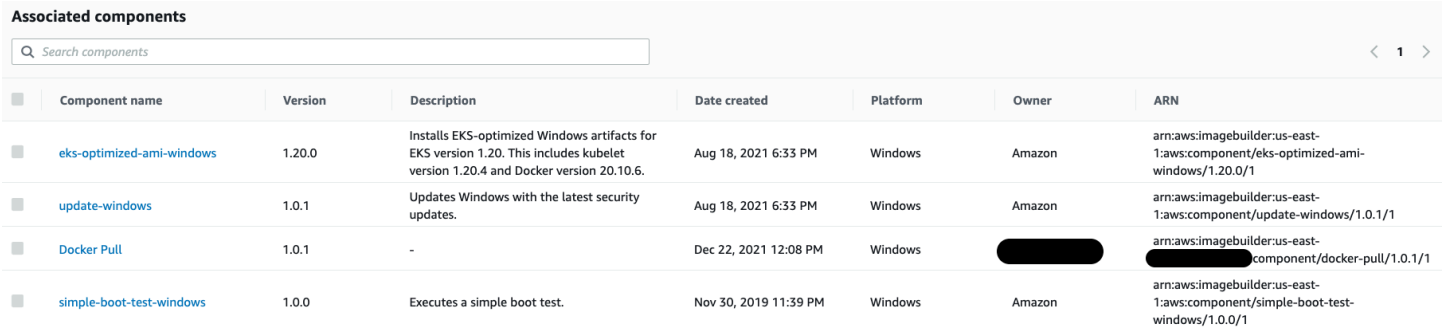
Applicazione di patch a server e contenitori Windows

L'applicazione di patch a Windows Server è un'attività di gestione standard per gli amministratori di Windows. Ciò può essere ottenuto utilizzando diversi strumenti come Amazon System Manager -

Patch Manager, WSUS, System Center Configuration Manager e molti altri. Tuttavia, i nodi Windows in un cluster Amazon EKS non devono essere trattati come normali server Windows. Devono essere trattati come server immutabili. In poche parole, evita di aggiornare un nodo esistente, basta avviarne uno nuovo basato su una nuova AMI aggiornata.

Utilizzando EC2 Image Builder è AMIs possibile automatizzare la compilazione, creando ricette e aggiungendo componenti.

L'esempio seguente mostra i componenti, che possono essere preesistenti creati da AWS (gestiti da Amazon) e i componenti creati da te (di mia proprietà). Presta particolare attenzione al componente gestito da Amazon chiamato update-windows, che aggiorna Windows Server prima di generare l'AMI tramite la pipeline EC2 Image Builder.



Associated components

Search components

1

Component name	Version	Description	Date created	Platform	Owner	ARN
eks-optimized-ami-windows	1.20.0	Installs EKS-optimized Windows artifacts for EKS version 1.20. This includes kubelet version 1.20.4 and Docker version 20.10.6.	Aug 18, 2021 6:33 PM	Windows	Amazon	arn:aws:imagebuilder:us-east-1:aws:component/eks-optimized-ami-windows/1.20.0/1
update-windows	1.0.1	Updates Windows with the latest security updates.	Aug 18, 2021 6:33 PM	Windows	Amazon	arn:aws:imagebuilder:us-east-1:aws:component/update-windows/1.0.1/1
Docker Pull	1.0.1	-	Dec 22, 2021 12:08 PM	Windows	[redacted]	arn:aws:imagebuilder:us-east-[redacted]component/docker-pull/1.0.1/1
simple-boot-test-windows	1.0.0	Executes a simple boot test.	Nov 30, 2019 11:39 PM	Windows	Amazon	arn:aws:imagebuilder:us-east-1:aws:component/simple-boot-test-windows/1.0.0/1

EC2 Image Builder ti consente di creare AMI basate su Amazon AMIs Managed Public e personalizzarle per soddisfare i tuoi requisiti aziendali. Puoi quindi AMIs associarli a Launch Templates che ti consentono di collegare una nuova AMI all'Auto Scaling Group creato da EKS Nodegroup. Al termine, puoi iniziare a terminare i nodi Windows esistenti e ne verranno lanciati di nuovi in base alla nuova AMI aggiornata.

Spingere e tirare immagini Windows

Amazon pubblica EKS ottimizzato AMIs che include due immagini di container Windows memorizzate nella cache.

```
mcr.microsoft.com/windows/servercore
mcr.microsoft.com/windows/nanoserver
```

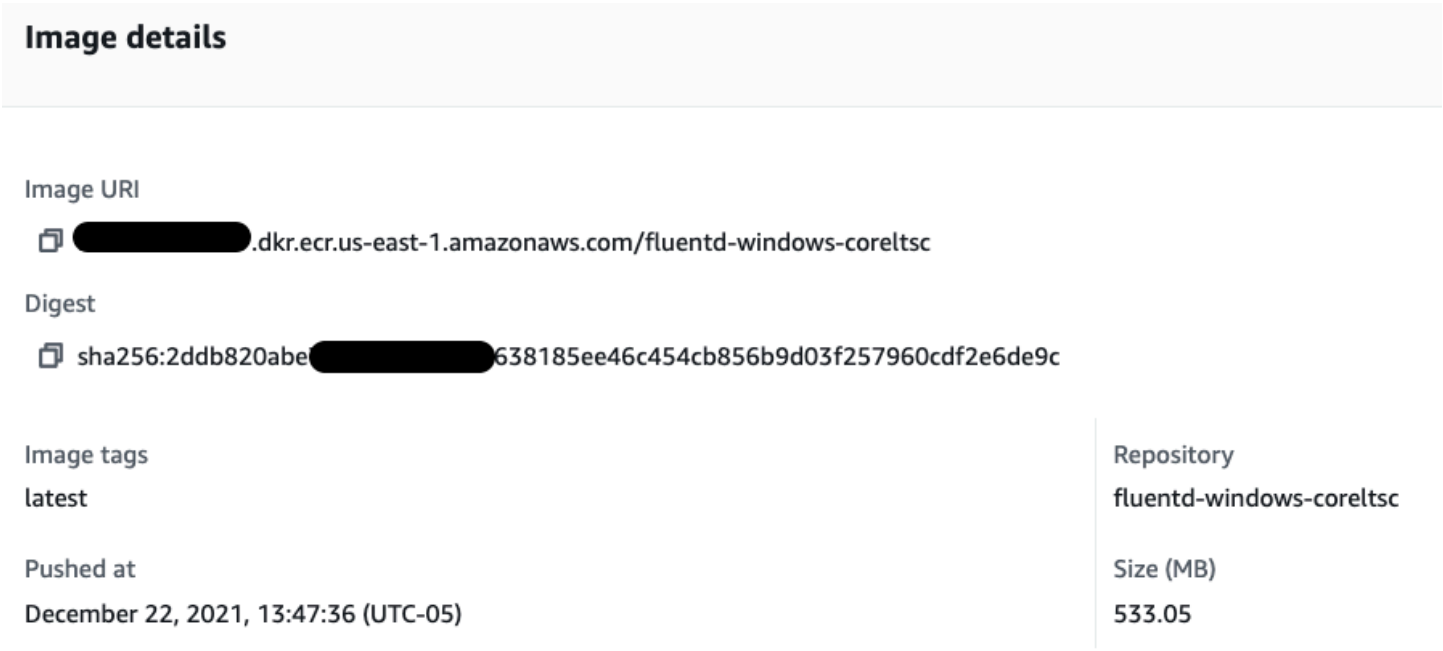
```
REPOSITORY                                                      TAG                     IMAGE ID       CREATED         SIZE
mcr.microsoft.com/windows/servercore                            ltsc2019                617dd5ead5b8   6 weeks ago     5.74GB
mcr.microsoft.com/windows/nanoserver                            1809                    8334af2fa7ba   6 weeks ago     257MB
[redacted].dkr.ecr.us-east-1.amazonaws.com/fluentd-windows-coreltsc   latest            721afca2c725   7 weeks ago     6.96GB
fluent/fluentd                                                  v1.14-windows-ltsc2019-1   721afca2c725   7 weeks ago     6.96GB
amazonaws.com/eks/pause-windows                                 latest                  6392f69ae6e7   10 months ago   255MB
```

Le immagini memorizzate nella cache vengono aggiornate in seguito agli aggiornamenti sul sistema operativo principale. Quando Microsoft rilascia un nuovo aggiornamento di Windows che influisce direttamente sull'immagine base del contenitore Windows, l'aggiornamento verrà avviato come un normale Windows Update sul sistema operativo principale. Mantenere l'ambiente up-to-date offre un ambiente più sicuro a livello di nodo e contenitore.

Le dimensioni dell'immagine di un contenitore Windows influiscono push/pull sulle operazioni, il che può rallentare i tempi di avvio del contenitore. La memorizzazione nella cache delle immagini dei contenitori Windows consente di I/O eseguire operazioni costose (estrazione dei file) sulla creazione della build AMI anziché sull'avvio del contenitore. Di conseguenza, tutti i livelli di immagine necessari verranno estratti dall'AMI e saranno pronti per essere utilizzati, accelerando il tempo di avvio di un contenitore Windows e la possibilità di iniziare ad accettare traffico. Durante un'operazione push, nel repository vengono caricati solo i livelli che compongono l'immagine.

L'esempio seguente mostra che su Amazon ECR le immagini del fluentd-windows-sac2004 hanno solo 390,18 MB. Questa è la quantità di upload avvenuta durante l'operazione push.

L'esempio seguente mostra un'immagine fluentd Windows ltsc inviata a un repository Amazon ECR. La dimensione del layer memorizzato in ECR è 533,05 MB.



L'output seguente di`docker image ls`, la dimensione del fluentd v1.14-windows-ltsc2019-1 è di 6,96 GB su disco, ma ciò non significa che abbia scaricato ed estratto quella quantità di dati.

In pratica, durante l'operazione pull verranno scaricati ed estratti solo i 533,05 MB compressi.

```
REPOSITORY                                                                  TAG
            IMAGE ID       CREATED         SIZE
111122223333.dkr.ecr.us-east-1.amazonaws.com/fluentd-windows-coreltsc   latest
            721afca2c725   7 weeks ago     6.96GB
fluent/fluentd                                                              v1.14-windows-
ltsc2019-1   721afca2c725   7 weeks ago     6.96GB
amazonaws.com/eks/pause-windows                                             latest
            6392f69ae6e7   10 months ago   255MB
```

La colonna delle dimensioni mostra la dimensione complessiva dell'immagine, 6,96 GB. Scomponendolo:

- Immagine di base LTSC di Windows Server Core 2019 = 5,74 GB
- Immagine di base non compressa Fluentd = 6,96 GB
- Differenza su disco = 1,2 GB
- Immagine finale compressa Fluentd ECR = 533,05 MB

L'immagine di base esiste già sul disco locale, per cui la quantità totale su disco è di 1,2 GB aggiuntivi. La prossima volta che vedrete la quantità di GBs nella colonna delle dimensioni, non preoccupatevi troppo, probabilmente più del 70% è già presente su disco come immagine del contenitore memorizzata nella cache.

Documentazione di riferimento

Accelerazione dei tempi di avvio dei container Windows con EC2 Image Builder e la strategia Image Cache

Esecuzione di carichi di lavoro eterogenei

Kubernetes supporta cluster eterogenei in cui è possibile avere una combinazione di nodi Linux e Windows nello stesso cluster. All'interno di quel cluster, puoi avere una combinazione di Pod che funzionano su Linux e Pod che funzionano su Windows. Puoi persino eseguire più versioni di Windows nello stesso cluster. Tuttavia, ci sono diversi fattori (come indicato di seguito) che dovranno essere presi in considerazione quando si prende questa decisione.

Assegnazione PODs ai nodi: best practice

Per mantenere i carichi di lavoro Linux e Windows sui rispettivi nodi specifici del sistema operativo, è necessario utilizzare una combinazione di selettori di nodi e contaminazioni/tolleranze. L'obiettivo principale della pianificazione dei carichi di lavoro in un ambiente eterogeneo è evitare di compromettere la compatibilità per i carichi di lavoro Linux esistenti.

Garantire che i carichi di lavoro specifici del sistema operativo arrivino sull'host container appropriato

Gli utenti possono garantire che i contenitori Windows possano essere pianificati sull'host appropriato utilizzando NodeSelectors. Oggi tutti i nodi Kubernetes hanno le seguenti etichette predefinite:

```
kubernetes.io/os = [windows|linux]
kubernetes.io/arch = [amd64|arm64|...]
```

Se una specifica Pod non include un NodeSelector come`"kubernetes.io/os": windows`, il Pod può essere pianificato su qualsiasi host, Windows o Linux. Questo può essere problematico poiché un contenitore Windows può essere eseguito solo su Windows e un contenitore Linux può essere eseguito solo su Linux.

Negli ambienti aziendali, non è raro disporre di un gran numero di implementazioni preesistenti per contenitori Linux, nonché di un ecosistema di off-the-shelf configurazioni, come Helm charts. In queste situazioni, potresti essere restio ad apportare modifiche ai NodeSelectors di una distribuzione. L'alternativa è usare Taints.

Ad esempio: `--register-with-taints='os=windows:NoSchedule'`

Se stai usando EKS, eksctl offre modi per applicare i taint tramite ClusterConfig:

```
NodeGroups:
  - name: windows-ng
    amiFamily: WindowsServer2022FullContainer
    ...
    labels:
      nodeclass: windows2022
    taints:
      os: "windows:NoSchedule"
```

Aggiungendo una macchia a tutti i nodi di Windows, lo scheduler non pianificherà i pod su quei nodi a meno che non tollerino la contaminazione. Esempio di manifesto Pod:

```
nodeSelector:
    kubernetes.io/os: windows
tolerations:
    - key: "os"
      operator: "Equal"
      value: "windows"
      effect: "NoSchedule"
```

Gestione di più build di Windows nello stesso cluster

L'immagine di base del contenitore Windows utilizzata da ogni pod deve corrispondere alla stessa versione di build del kernel del nodo. Se desideri utilizzare più build di Windows Server nello stesso cluster, devi impostare etichette di nodo aggiuntive, NodeSelectors o utilizzare un'etichetta chiamata windows-build.

Kubernetes 1.17 aggiunge automaticamente una nuova etichetta node.kubernetes.io/windows-build per semplificare la gestione di più build di Windows nello stesso cluster. Se utilizzi una versione precedente, ti consigliamo di aggiungere questa etichetta manualmente ai nodi Windows.

Questa etichetta riflette il numero principale, secondario e di build di Windows che devono corrispondere per motivi di compatibilità. Di seguito sono riportati i valori utilizzati oggi per ogni versione di Windows Server.

È importante notare che Windows Server sta passando al Long-Term Servicing Channel (LTSC) come canale di rilascio principale. Il Windows Server Semi-Annual Channel (SAC) è stato ritirato il 9 agosto 2022. Non ci saranno future versioni SAC di Windows Server.

Nome prodotto	Numeri di build
Server completo 2022 LTSC	10.0.20348
Server core 2019 LTSC	10.0.17763

È possibile verificare la versione di build del sistema operativo tramite il seguente comando:

```
kubectl get nodes -o wide
```

L'output KERNEL-VERSION corrisponde alla versione build del sistema operativo Windows.

```
NAME                          STATUS   ROLES    AGE   VERSION               INTERNAL-
IP   EXTERNAL-IP     OS-IMAGE                         KERNEL-VERSION
 CONTAINER-RUNTIME
ip-10-10-2-235.ec2.internal   Ready    <none>   23m   v1.24.7-eks-fb459a0
 10.10.2.235   3.236.30.157    Windows Server 2022 Datacenter   10.0.20348.1607
         containerd://1.6.6
ip-10-10-31-27.ec2.internal   Ready    <none>   23m   v1.24.7-eks-fb459a0
 10.10.31.27   44.204.218.24   Windows Server 2019 Datacenter   10.0.17763.4131
         containerd://1.6.6
ip-10-10-7-54.ec2.internal    Ready    <none>   31m   v1.24.11-eks-a59e1f0   10.10.7.54
    3.227.8.172     Amazon Linux 2                  5.10.173-154.642.amzn2.x86_64
 containerd://1.6.19
```

L'esempio seguente applica un NodeSelector aggiuntivo al manifesto del pod in modo che corrisponda alla versione Windows-build corretta quando si eseguono diverse versioni del sistema operativo di gruppi di nodi Windows.

```
nodeSelector:
    kubernetes.io/os: windows
    node.kubernetes.io/windows-build: '10.0.20348'
tolerations:
    - key: "os"
    operator: "Equal"
    value: "windows"
    effect: "NoSchedule"
```

Semplificazione e tolleranza nei manifesti Pod utilizzando NodeSelector RuntimeClass

Puoi anche utilizzarlo per RuntimeClass semplificare il processo di utilizzo delle macchie e delle tolleranze. Ciò può essere ottenuto creando un RuntimeClass oggetto che viene utilizzato per incapsulare queste macchie e tolleranze.

Crea un RuntimeClass eseguendo il seguente manifesto:

```
apiVersion: node.k8s.io/v1beta1
kind: RuntimeClass
metadata:
  name: windows-2022
```

```
handler: 'docker'
scheduling:
  nodeSelector:
    kubernetes.io/os: 'windows'
    kubernetes.io/arch: 'amd64'
    node.kubernetes.io/windows-build: '10.0.20348'
  tolerations:
  - effect: NoSchedule
    key: os
    operator: Equal
    value: "windows"
```

Una volta creata la classe Runtime, assegnala utilizzando come Spec nel manifesto del Pod:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: iis-2022
  labels:
    app: iis-2022
spec:
  replicas: 1
  template:
    metadata:
      name: iis-2022
      labels:
        app: iis-2022
    spec:
      runtimeClassName: windows-2022
      containers:
      - name: iis
```

Managed Node Group Support

Per aiutare i clienti a eseguire le proprie applicazioni Windows in modo più semplificato, AWS ha lanciato il supporto per il supporto Amazon EKS Managed Node Group (MNG) per i container Windows il 15 dicembre 2022. Per aiutare ad allineare i team operativi, Windows è abilitato a utilizzare MNGs gli stessi flussi di lavoro e gli stessi strumenti di Linux. MNGs Sono supportate le versioni complete e principali della famiglia AMI (Amazon Machine Image) di Windows Server 2019 e 2022.

Le seguenti famiglie AMI sono supportate per i Managed Node Groups (MNG).

Famiglia AMI
Windows_Core_2019_x86_64
Windows_Full_2019_x86_64
Windows_Core_2022_x86_64
Windows_Full_2022_x86_64

Documentazione aggiuntiva

Documentazione ufficiale AWS: https://docs.aws.amazon.com/eks/ latest/userguide/windows - support.html

Per capire meglio come funziona Pod Networking (CNI), controlla il seguente link: -networking.html https://docs.aws.amazon.com/eks/ latest/userguide/pod

Blog AWS sulla distribuzione di gruppi di nodi gestiti per Windows su EKS: https://aws.amazon.com/ blogs/ contenitori/ -/deploying-amazon-eks-windowsmanaged-node-groups

Contesti di sicurezza Pod

Pod Security Policies (PSP) e Pod Security Standards (PSS) sono due modi principali per rafforzare la sicurezza in Kubernetes. Tieni presente che PodSecurityPolicy è obsoleto a partire da Kubernetes v1.21 e verrà rimosso nella v1.25 e Pod Security Standard (PSS) è l'approccio consigliato da Kubernetes per far rispettare la sicurezza in futuro.

Una Pod Security Policy (PSP) è una soluzione nativa di Kubernetes per implementare le politiche di sicurezza. PSP è una risorsa a livello di cluster che controlla gli aspetti sensibili alla sicurezza delle specifiche Pod. Utilizzando Pod Security Policy è possibile definire una serie di condizioni che i Pod devono soddisfare per essere accettati dal cluster. La funzionalità PSP è disponibile sin dai primi giorni di Kubernetes ed è progettata per impedire la creazione di pod mal configurati su un determinato cluster.

Per ulteriori informazioni sulle politiche di sicurezza dei pod, consulta la documentazione di Kubernetes. In base alla politica di deprecazione di Kubernetes, le versioni precedenti smetteranno di ricevere supporto nove mesi dopo l'obsolescenza della funzionalità.

D'altra parte, i Pod Security Standards (PSS), che è l'approccio di sicurezza consigliato e in genere implementato utilizzando i contesti di sicurezza, sono definiti come parte delle specifiche del Pod e del contenitore nel manifesto del Pod. PSS è lo standard ufficiale che il team di progetto Kubernetes ha definito per soddisfare le migliori pratiche relative alla sicurezza per i Pods. Definisce politiche come quelle di base (minimamente restrittive, predefinite), privilegiate (non restrittive) e limitate (le più restrittive).

Ti consigliamo di iniziare con il profilo di base. Il profilo di base PSS offre un solido equilibrio tra sicurezza e potenziale attrito, richiede un elenco minimo di eccezioni e rappresenta un buon punto di partenza per la sicurezza dei carichi di lavoro. Se stai utilizzando PSPs attualmente, ti consigliamo di passare a PSS. Maggiori dettagli sulle politiche PSS sono disponibili nella documentazione di Kubernetes. Queste politiche possono essere applicate con diversi strumenti, inclusi quelli di OPA e Kyverno. Ad esempio, Kyverno fornisce qui la raccolta completa delle politiche PSS.

Le impostazioni del contesto di sicurezza consentono di concedere privilegi a determinati processi, utilizzare i profili di programma per limitare le funzionalità ai singoli programmi, consentire l'aumento dei privilegi, filtrare le chiamate di sistema, tra le altre cose.

I pod Windows in Kubernetes presentano alcune limitazioni e caratteristiche distintive rispetto ai carichi di lavoro standard basati su Linux per quanto riguarda i contesti di sicurezza.

Windows utilizza un oggetto Job per contenitore con un filtro dello spazio dei nomi di sistema per contenere tutti i processi in un contenitore e fornire l'isolamento logico dall'host. Non è possibile eseguire un contenitore Windows senza il filtro dello spazio dei nomi. Ciò significa che i privilegi di sistema non possono essere rivendicati nel contesto dell'host e quindi i contenitori privilegiati non sono disponibili in Windows.

Le seguenti `windowsOptions` sono le uniche opzioni documentate del contesto di sicurezza di Windows, mentre le altre sono opzioni generali del contesto di sicurezza

Per un elenco degli attributi del contesto di sicurezza supportati in Windows e Linux, consulta la documentazione ufficiale disponibile qui.

Le impostazioni specifiche del Pod vengono applicate a tutti i contenitori. Se non specificato, PodSecurityContext verranno utilizzate le opzioni di. Se impostato in entrambi SecurityContext e PodSecurityContext, il valore specificato in ha la SecurityContext precedenza.

Ad esempio, l'impostazione runAsUser del nome per contenitori e contenitori, che è un'opzione di Windows, è un equivalente approssimativo dell' runAsUser impostazione specifica di Linux e nel seguente manifesto, il contesto di sicurezza specifico del pod viene applicato a tutti i contenitori

```
apiVersion: v1
kind: Pod
metadata:
  name: run-as-username-pod-demo
spec:
  securityContext:
    windowsOptions:
      runAsUserName: "ContainerUser"
  containers:
  - name: run-as-username-demo

  nodeSelector:
    kubernetes.io/os: windows
```

Di seguito, invece, il contesto di sicurezza a livello di contenitore ha la precedenza sul contesto di sicurezza a livello di pod.

```
apiVersion: v1
kind: Pod
metadata:
  name: run-as-username-container-demo
spec:
  securityContext:
    windowsOptions:
      runAsUserName: "ContainerUser"
  containers:
  - name: run-as-username-demo
    ..
    securityContext:
        windowsOptions:
            runAsUserName: "ContainerAdministrator"
  nodeSelector:
    kubernetes.io/os: windows
```

Esempi di valori accettabili per il campo runAsUser Name: ContainerAdministrator,, NT AUTHORITY\ NETWORK SERVICE ContainerUser, NT AUTHORITY\ LOCAL SERVICE

In genere è consigliabile eseguire i contenitori con pod ContainerUser per Windows. Gli utenti non sono condivisi tra il contenitore e l' ContainerAdministrator host, ma dispongono di privilegi aggiuntivi all'interno del contenitore. Tieni presente che ci sono delle limitazioni relative al nome utente di cui devi essere a conoscenza.

Un buon esempio di quando usare ContainerAdministrator è impostare PATH. Puoi usare la direttiva USER per farlo, in questo modo:

```
USER ContainerAdministrator
RUN setx /M PATH "%PATH%;C:/your/path"
USER ContainerUser
```

Nota inoltre che i segreti sono scritti in testo non crittografato sul volume del nodo (rispetto a tmpfs/in-memory su linux). Ciò significa che devi fare due cose

- Usa file ACLs per proteggere la posizione segreta del file
- Usa la crittografia a livello di volume utilizzando BitLocker

Opzioni di archiviazione persistente

Che cos'è un plug-in in-tree e un plug-in di out-of-tree volume?

Prima dell'introduzione della Container Storage Interface (CSI), tutti i plugin di volume erano integrati nell'albero, ossia erano creati, collegati, compilati e forniti con i binari principali di Kubernetes ed estendevano l'API principale di Kubernetes. Ciò significava che l'aggiunta di un nuovo sistema di storage a Kubernetes (un plug-in di volume) richiedeva il controllo del codice nel repository di codice principale di Kubernetes.

Out-of-tree i plugin di volume sono sviluppati indipendentemente dalla codebase di Kubernetes e vengono distribuiti (installati) nei cluster Kubernetes come estensioni. Ciò offre ai fornitori la possibilità di aggiornare i driver, vale a dire separatamente dal ciclo di rilascio di Kubernetes. out-of-band Ciò è in gran parte possibile perché Kubernetes ha creato un'interfaccia di archiviazione o CSI che offre ai fornitori un modo standard di interfacciarsi con k8s.

Puoi trovare ulteriori informazioni sulle classi di storage Amazon Elastic Kubernetes Services (EKS) e sui driver CSI su .html https://docs.aws.amazon.com/eks/ latest/userguide/storage

Plugin In-tree Volume per Windows

I volumi Kubernetes consentono di distribuire applicazioni con requisiti di persistenza dei dati su Kubernetes. La gestione dei volumi persistenti consiste in singoli contenitori in un pod. provisioning/ de-provisioning/resizing of volumes, attaching/detaching a volume to/from a Kubernetes node, and mounting/dismounting a volume to/from Il codice per l'implementazione di queste azioni di gestione

dei volumi per uno specifico back-end o protocollo di archiviazione viene fornito sotto forma di un plug-in di volume Kubernetes (In-tree Volume Plugins). Su Amazon Elastic Kubernetes Services (EKS), la seguente classe di plugin di volume Kubernetes è supportata su Windows:

awsElasticBlockPlugin In-tree Volume: Store

Per utilizzare il plug-in di volume In-tree sui nodi Windows, è necessario crearne uno aggiuntivo StorageClass per utilizzare NTFS come fsType. Su EKS, l'impostazione predefinita StorageClass utilizza ext4 come FSType predefinito.

A StorageClass fornisce agli amministratori un modo per descrivere le «classi» di storage che offrono. Classi diverse potrebbero corrispondere a quality-of-service livelli, politiche di backup o politiche arbitrarie determinate dagli amministratori del cluster. Kubernetes non ha opinioni su cosa rappresentino le classi. Questo concetto viene talvolta chiamato «profili» in altri sistemi di storage.

È possibile verificarlo eseguendo il seguente comando:

```
kubectl describe storageclass gp2
```

Output:

```
Name:            gp2
IsDefaultClass:  Yes
Annotations:     kubectl.kubernetes.io/last-applied-
configuration={"apiVersion":"storage.k8s.io/v1","kind":"StorageClas
","metadata":{"annotations":{"storageclass.kubernetes.io/is-default-
class":"true"},"name":"gp2"},"parameters":{"fsType"
"ext4","type":"gp2"},"provisioner":"kubernetes.io/aws-
ebs","volumeBindingMode":"WaitForFirstConsumer"}
,storageclass.kubernetes.io/is-default-class=true
Provisioner:           kubernetes.io/aws-ebs
Parameters:            fsType=ext4,type=gp2
AllowVolumeExpansion:  <unset>
MountOptions:          <none>
ReclaimPolicy:         Delete
VolumeBindingMode:     WaitForFirstConsumer
Events:                <none>
```

Per creare una nuova versione StorageClass che supporti NTFS, utilizzate il seguente manifesto:

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
```

```
metadata:
  name: gp2-windows
provisioner: kubernetes.io/aws-ebs
parameters:
  type: gp2
  fsType: ntfs
volumeBindingMode: WaitForFirstConsumer
```

Createlo StorageClass eseguendo il comando seguente:

```
kubectl apply -f NTFSStorageClass.yaml
```

Il passaggio successivo consiste nel creare un Persistent Volume Claim (PVC).

Un PersistentVolume (PV) è una parte di storage nel cluster che è stata fornita da un amministratore o fornita dinamicamente utilizzando PVC. È una risorsa del cluster proprio come un nodo è una risorsa del cluster. Questo oggetto API acquisisce i dettagli dell'implementazione dello storage, che si tratti di NFS, iSCSI o di un sistema di storage. cloud-provider-specific

A PersistentVolumeClaim (PVC) è una richiesta di archiviazione da parte di un utente. I claim possono richiedere dimensioni e modalità di accesso specifiche (ad esempio, possono essere montati ReadWriteOnce ReadOnlyMany o ReadWriteMany).

Gli utenti devono disporre PersistentVolumes di attributi diversi, ad esempio le prestazioni, per diversi casi d'uso. Gli amministratori dei cluster devono essere in grado di offrire una varietà di soluzioni PersistentVolumes che si differenziano non solo in termini di dimensioni e modalità di accesso, senza esporre gli utenti ai dettagli di come tali volumi vengono implementati. Per queste esigenze, c'è la StorageClass risorsa.

Nell'esempio seguente, il PVC è stato creato all'interno delle finestre del namespace.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: ebs-windows-pv-claim
  namespace: windows
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: gp2-windows
  resources:
```

```
    requests:
      storage: 1Gi
```

Create il PVC eseguendo il seguente comando:

```
kubectl apply -f persistent-volume-claim.yaml
```

Il seguente manifesto crea un Windows Pod, configura l' VolumeMount as `C:\Data` e utilizza il PVC come storage collegato`C:\Data`.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: windows-server-ltsc2019
  namespace: windows
spec:
  selector:
    matchLabels:
      app: windows-server-ltsc2019
      tier: backend
      track: stable
  replicas: 1
  template:
    metadata:
      labels:
        app: windows-server-ltsc2019
        tier: backend
        track: stable
    spec:
      containers:
      - name: windows-server-ltsc2019
        image: mcr.microsoft.com/windows/servercore:ltsc2019
        ports:
        - name: http
          containerPort: 80
        imagePullPolicy: IfNotPresent
        volumeMounts:
        - mountPath: "C:\\data"
          name: test-volume
      volumes:
        - name: test-volume
          persistentVolumeClaim:
            claimName: ebs-windows-pv-claim
```

```
      nodeSelector:
        kubernetes.io/os: windows
        node.kubernetes.io/windows-build: '10.0.17763'
```

Verifica i risultati accedendo al pod Windows tramite PowerShell:

```
kubectl exec -it podname powershell -n windows
```

All'interno del Windows Pod, esegui: `ls`

Output:

```
PS C:\> ls


    Directory: C:\


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----          3/8/2021   1:54 PM                data
d-----          3/8/2021   3:37 PM                inetpub
d-r---          1/9/2021   7:26 AM                Program Files
d-----          1/9/2021   7:18 AM                Program Files (x86)
d-r---          1/9/2021   7:28 AM                Users
d-----          3/8/2021   3:36 PM                var
d-----          3/8/2021   3:36 PM                Windows
-a----         12/7/2019   4:20 AM           5510 License.txt
```

La directory dei dati è fornita dal volume EBS.

Out-of-tree per Windows

Il codice associato ai plugin CSI viene fornito come out-of-tree script e file binari che in genere vengono distribuiti come immagini di container e distribuiti utilizzando costrutti Kubernetes standard come e. DaemonSets StatefulSets I plugin CSI gestiscono un'ampia gamma di azioni di gestione dei volumi in Kubernetes. I plugin CSI in genere sono costituiti da plug-in di nodo (che vengono eseguiti su ciascun nodo come plug-in di controllo) e plug-in di controllo. DaemonSet

I plugin dei nodi CSI (specialmente quelli associati a volumi persistenti esposti come dispositivi a blocchi o su un file system condiviso) devono eseguire diverse operazioni privilegiate come la

scansione dei dispositivi a disco, il montaggio di file system, ecc. Queste operazioni differiscono per ogni sistema operativo host. Per i nodi di lavoro Linux, i plug-in dei nodi CSI containerizzati vengono in genere distribuiti come contenitori privilegiati. Per i nodi di lavoro Windows, le operazioni privilegiate per i plug-in dei nodi CSI containerizzati sono supportate utilizzando csi-proxy, un file binario autonomo gestito dalla comunità che deve essere preinstallato su ogni nodo Windows.

L'AMI Windows ottimizzata per Amazon EKS include il proxy CSI a partire da aprile 2022. I clienti possono utilizzare il driver CSI SMB sui nodi Windows per accedere ad Amazon for Windows File Server, Amazon FSx FSx for NetApp ONTAP SMB Shares, and/or AWS Storage Gateway — File Gateway.

Il seguente blog contiene dettagli di implementazione su come configurare SMB CSI Driver per utilizzare Amazon FSx for Windows File Server come storage persistente per Windows Pods.

File server Amazon FSx per Windows

Un'opzione consiste nell'utilizzare Amazon FSx for Windows File Server tramite una funzionalità SMB denominata SMB Global Mapping che consente di montare una condivisione SMB sull'host, quindi trasferire le directory di tale condivisione in un contenitore. Il contenitore non deve essere configurato con un server, una condivisione, un nome utente o una password specifici: tutto viene invece gestito sull'host. Il contenitore funzionerà come se avesse una memoria locale.

> La mappatura globale SMB è trasparente per l'orchestratore ed è montata, il HostPath che può implicare problemi di sicurezza.

Nell'esempio seguente, il percorso `G:\Directory\app-state` è una condivisione SMB sul nodo Windows.

```
apiVersion: v1
kind: Pod
metadata:
  name: test-fsx
spec:
  containers:
  - name: test-fsx
    image: mcr.microsoft.com/windows/servercore:ltsc2019
    command:
      - powershell.exe
      - -command
      - "Add-WindowsFeature Web-Server; Invoke-WebRequest -UseBasicParsing
 -Uri 'https://dotnetbinaries.blob.core.windows.net/servicemonitor/2.0.1.6/
```

```
ServiceMonitor.exe' -OutFile 'C:\\ServiceMonitor.exe'; echo '<html><body><br/
><br/><marquee><H1>Hello EKS!!!<H1><marquee></body><html>' > C:\\inetpub\\wwwroot\
\default.html; C:\\ServiceMonitor.exe 'w3svc'; "
    volumeMounts:
      - mountPath: C:\dotnetapp\app-state
        name: test-mount
  volumes:
    - name: test-mount
      hostPath:
        path: G:\Directory\app-state
        type: Directory
  nodeSelector:
      beta.kubernetes.io/os: windows
      beta.kubernetes.io/arch: amd64
```

Il seguente blog contiene dettagli di implementazione su come configurare Amazon FSx for Windows File Server come storage persistente per Windows Pods.

Rafforzamento delle immagini dei contenitori Windows

Stai rafforzando le immagini dei contenitori Windows? Nel corso degli anni, ho lavorato con clienti a livello globale per aiutarli a migrare i carichi di lavoro legacy verso i container, in particolare i carichi di lavoro Windows. Con oltre 20 anni di esperienza, ho visto organizzazioni dedicare notevoli sforzi e risorse al rafforzamento dei propri server Windows, implementando qualsiasi cosa, dai benchmark CIS alla protezione antivirus in fase di esecuzione per salvaguardare i dati sensibili.

Tuttavia, è emersa una tendenza preoccupante. Man mano che queste macchine virtuali altamente sicure vengono modernizzate in contenitori, molte pratiche critiche di rafforzamento vengono trascurate. Le migliori pratiche di sicurezza di Windows, dall'immagine di base (OS) ai servizi Web come IIS, vengono spesso trascurate e la maggior parte dell'attenzione si concentra esclusivamente sulla protezione dell'host del contenitore. È fondamentale riconoscere che, sebbene i container operino in namespace isolati, condividono comunque le primitive del kernel con l'host. Gli aggressori sono in genere più interessati ai movimenti laterali piuttosto che a colpire direttamente l'host del container, il che consente loro di sfruttare le impostazioni di sicurezza deboli dei container e accedere ai dati sensibili.

L'obiettivo delle documentazioni è evidenziare alcune impostazioni di sicurezza essenziali da implementare specificamente per i contenitori Windows che ospitano siti Web su IIS. `ASP.NET` Ci concentreremo su quattro aree chiave:

- Politiche di sicurezza dell'account
- Policy di audit
- Le migliori pratiche di sicurezza IIS
- Principio del privilegio minimo

Inizieremo approfondendo il motivo per cui ciascuna di queste configurazioni di sicurezza è fondamentale per proteggere i contenitori Windows, esaminando i rischi specifici che mitigano e i vantaggi in termini di sicurezza che offrono. Successivamente, esamineremo un frammento di codice che dimostra come implementare correttamente queste configurazioni nel tuo Dockerfile, assicurando che il contenitore sia protetto da potenziali minacce. Infine, analizzeremo ogni impostazione in dettaglio, offrendo una spiegazione esaustiva della sua funzione, dell'impatto sulla sicurezza dei container e di come contribuisce alla salvaguardia delle applicazioni. Questo approccio non solo ti mostrerà come applicare queste best practice, ma ti fornirà anche le informazioni necessarie per capire perché sono essenziali per mantenere un solido livello di sicurezza negli ambienti containerizzati.

1. Configura le politiche dell'account (password o blocco) utilizzando le politiche di sicurezza locali e il registro

Windows Server Core è un'opzione di installazione minima disponibile come parte dell' [AMI Windows ottimizzata EKS] (https://docs.aws.amazon.com/eks/latest/userguide/eksoptimized-windows-ami-.html). La configurazione delle politiche dell'account (password o blocco) utilizzando le politiche di sicurezza locali e il registro rafforza la sicurezza del sistema applicando solide regole di password e blocco. Queste politiche richiedono agli utenti di creare password complesse con una lunghezza e una complessità minime definite, proteggendole dai comuni attacchi relativi alle password.

Impostando un'età massima per le password, agli utenti viene richiesto di aggiornare regolarmente le proprie password, riducendo la probabilità che le credenziali vengano compromesse. Le politiche di blocco aggiungono un ulteriore livello di protezione bloccando temporaneamente gli account dopo un determinato numero di tentativi di accesso falliti, contribuendo a prevenire gli attacchi di forza bruta. La configurazione di queste impostazioni tramite il registro di Windows consente agli amministratori di applicare queste misure di sicurezza a livello di sistema, garantendo uniformità e conformità in tutta l'organizzazione. L'applicazione di queste politiche di account in un contenitore Windows è essenziale per mantenere la coerenza della sicurezza, anche se i contenitori sono spesso effimeri e destinati a carichi di lavoro isolati:

Coerenza della sicurezza

- Conformità: l'applicazione di politiche coerenti in materia di password e regole di blocco nei container aiuta a mantenere la conformità in materia di sicurezza, specialmente in ambienti che richiedono controlli di accesso rigorosi (ad esempio, conformità normativa come HIPAA, PCI-DSS).
- Contenitori rinforzati: l'applicazione di queste impostazioni garantisce che il contenitore Windows sia protetto dagli accessi non autorizzati o dagli attacchi basati su password, allineando il livello di sicurezza del contenitore alle più ampie politiche di sicurezza del sistema.

Protezione contro gli attacchi di forza bruta

- Blocco dell'account: queste impostazioni aiutano a difendersi dai tentativi di accesso con forza bruta bloccando gli account dopo un numero specifico di tentativi di accesso non riusciti. Ciò impedisce agli aggressori di provare un numero illimitato di password.
- Complessità delle password: la richiesta di password complesse con una lunghezza sufficiente riduce la probabilità che vengano sfruttate password deboli, anche in ambienti containerizzati isolati.

Scenari multiutente

- Se l'applicazione containerizzata è progettata per gestire più utenti o richiede l'autenticazione degli utenti, l'applicazione delle politiche in materia di password garantisce che gli account utente all'interno del contenitore aderiscano a rigide regole di sicurezza, limitando l'accesso solo agli utenti autorizzati.

Contenitori Windows persistenti

- Sebbene i contenitori siano generalmente considerati effimeri, alcuni contenitori Windows possono eseguire servizi a lungo termine o gestire gli utenti, il che rende importante applicare politiche di sicurezza adeguate in modo simile a un normale server Windows.

Coerenza negli ambienti ibridi

- Se utilizzi sia macchine virtuali che contenitori nella tua infrastruttura, l'applicazione delle stesse politiche di sicurezza (ad esempio, le password/lockout politiche) in tutti gli ambienti garantisce standard di sicurezza uniformi, semplificando la governance e la gestione.

In sintesi, l'applicazione di queste politiche relative agli account all'interno dei contenitori Windows garantisce che i contenitori non rappresentino un punto debole della strategia di sicurezza, proteggendo dagli attacchi di password e rafforzando la coerenza nell'intero ambiente.

File Docker:

```
# Configure account policies for password complexity and lockout
RUN powershell -Command \
      "Write-Output 'Configuring Account Policies (Password/Lockout)...'; \
      NET ACCOUNTS /MINPWLEN:14 /MAXPWAGE:60 /MINPWAGE:14 /LOCKOUTTHRESHOLD:5
```

Spiegazione:

Questa sezione configura le politiche dell'account per le impostazioni di password e blocco tramite il registro di Windows. Queste politiche aiutano a far rispettare la sicurezza controllando i requisiti relativi alle password e le soglie di blocco degli account.

1. MinimumPasswordLength (MINPWLEN) = 14 Questa impostazione definisce il numero minimo di caratteri per una password. L'intervallo è 0-14 caratteri; il valore predefinito è sei caratteri.
2. MaximumPasswordAge (MAXPWAGE) = 60 Questa impostazione definisce il numero massimo di giorni di validità di una password. Non viene specificato alcun limite utilizzando UNLIMITED. / MAXPWAGE non può essere inferiore a /MINPWAGE. L'intervallo è 1-999; il valore predefinito è 90 giorni
3. Lockout Threshold (LOCKOUTTHRESHOLD) = 5 Questa impostazione definisce la soglia per i tentativi di accesso non riusciti. Dopo 5 tentativi errati, l'account verrà bloccato.

Queste impostazioni aiutano a migliorare la sicurezza delle password e a prevenire gli attacchi di forza bruta applicando politiche rigorose in materia di password e bloccando gli account dopo un certo numero di tentativi di accesso falliti.

2. Policy di audit

Le policy di controllo sono importanti per Windows Containers perché forniscono una visibilità fondamentale sugli eventi di sicurezza, come i tentativi di accesso e l'uso dei privilegi, aiutando a rilevare accessi non autorizzati, monitorare l'attività degli utenti e garantire la conformità agli standard normativi. Anche nella natura effimera dei container, i registri di controllo sono essenziali per le indagini sugli incidenti, il rilevamento proattivo delle minacce e il mantenimento di un livello di sicurezza coerente negli ambienti containerizzati.

Monitoraggio della sicurezza e conformità:

- Tieni traccia delle attività degli utenti: le politiche di controllo consentono agli amministratori di monitorare le attività degli utenti, come i tentativi di accesso e l'uso dei privilegi, all'interno del contenitore. Questo è fondamentale per rilevare accessi non autorizzati o comportamenti sospetti.
- Conformità alle normative: molte organizzazioni sono tenute a registrare gli eventi di sicurezza per garantire la conformità a normative come HIPAA, PCI-DSS e GDPR. L'attivazione di politiche di controllo nei container garantisce il rispetto di questi requisiti, anche in ambienti containerizzati.

Indagine sugli incidenti:

- Analisi forense: se un'applicazione o un servizio containerizzato viene compromesso, i registri di controllo possono fornire informazioni preziose per l'analisi post-incidente. Aiutano i team di sicurezza a tracciare le azioni intraprese dagli aggressori o a identificare come si è verificata una violazione.
- Rilevamento in tempo reale: i registri di controllo consentono agli amministratori di impostare avvisi in tempo reale per eventi critici (ad esempio, tentativi di accesso falliti, escalation dei privilegi). Questo monitoraggio proattivo aiuta a rilevare tempestivamente gli attacchi e consente tempi di risposta più rapidi.

Coerenza tra gli ambienti:

- Posizione di sicurezza uniforme: applicando politiche di controllo nei container tramite il registro, garantite pratiche di sicurezza coerenti in ambienti containerizzati e non containerizzati. In questo modo si evita che i container diventino un punto morto per il monitoraggio della sicurezza.
- Visibilità negli ambienti ibridi: per le organizzazioni che utilizzano sia server che container Windows tradizionali, le policy di controllo offrono visibilità e controllo simili su tutte le piattaforme, rendendo la gestione più semplice ed efficace.

Monitoraggio delle operazioni privilegiate:

- Controllo dell'utilizzo dei privilegi: negli ambienti container in cui le applicazioni vengono eseguite con privilegi elevati o in cui vengono eseguite attività amministrative, il controllo delle operazioni privilegiate garantisce la responsabilità. È possibile registrare chi ha avuto accesso a risorse riservate o ha eseguito attività critiche all'interno del contenitore.

- Prevenzione dell'abuso dei privilegi: monitorando l'uso dei privilegi, è possibile rilevare quando utenti non autorizzati cercano di elevare i propri privilegi o di accedere ad aree riservate all'interno del contenitore, il che aiuta a prevenire attacchi interni o esterni.

Rilevamento dei tentativi di accesso non autorizzati:

- Tentativi di accesso non riusciti: l'abilitazione delle politiche di controllo per i tentativi di accesso non riusciti aiuta a identificare gli attacchi di forza bruta o i tentativi non autorizzati di accedere alle applicazioni containerizzate. Ciò fornisce visibilità su chi sta cercando di accedere al sistema e con quale frequenza.
- Monitoraggio del blocco dell'account: il controllo degli eventi di blocco degli account consente agli amministratori di rilevare e indagare su potenziali blocchi causati da attività sospette o dannose.

Sicurezza persistente anche in ambienti effimeri:

- Effimeri ma sicuri: sebbene i container siano effimeri, il che significa che possono essere eliminati e ricreati frequentemente, il controllo svolge comunque un ruolo chiave nel garantire che gli eventi di sicurezza vengano acquisiti mentre il contenitore è in esecuzione. Ciò garantisce che gli eventi di sicurezza critici vengano registrati per tutta la durata del ciclo di vita del contenitore.

Registrazione centralizzata:

- Inoltro dei log a sistemi centralizzati: i container possono essere integrati con sistemi di registrazione centralizzati (ad esempio, ELK stack, AWS CloudWatch) per acquisire i log di controllo da più istanze di container. Ciò consente una migliore analisi e correlazione degli eventi di sicurezza nell'infrastruttura.

File Docker:

```
# Configure audit policies for logging security events
RUN powershell -Command \
    "Write-Host 'Configuring Audit Policy..'; \
    Set-ItemProperty -Path 'HKLM:\\SYSTEM\\CurrentControlSet\\Control\\Lsa' -Name
 'SCENoApplyLegacyAuditPolicy' -Value 0; \
    auditpol /set /category:"Logon/Logoff" /subcategory:"Logon" /failure:enable

# Creates STDOUT on Windows Containers (check GitHub LogMonitor:: https://github.com/
microsoft/windows-container-tools/blob/main/LogMonitor/README.md)
```

```
COPY LogMonitor.exe LogMonitorConfig.json 'C:\\LogMonitor\\'
WORKDIR /LogMonitor
```

Spiegazione:

Questa sezione configura le politiche di controllo utilizzando le modifiche del registro. Le politiche di controllo controllano quali eventi di sicurezza vengono registrati da Windows, il che aiuta a monitorare e rilevare i tentativi di accesso non autorizzati.

1. SCENoApplyLegacyAuditPolicy = 0 Ciò disabilita il formato dei criteri di controllo legacy, abilitando politiche di controllo più granulari introdotte nelle versioni successive di Windows. Questo è importante per le configurazioni di controllo moderne.
2. Sottocategoria Auditpol: «Logon» Questa impostazione consente il controllo degli eventi di accesso riusciti e non riusciti. Il valore 3 indica che Windows registrerà sia i tentativi di accesso riusciti che quelli non riusciti. Questo aiuta a monitorare chi accede al sistema e a rilevare i tentativi di accesso non riusciti.

Queste politiche di controllo sono fondamentali per il monitoraggio della sicurezza e la conformità, in quanto forniscono registri dettagliati di importanti eventi di sicurezza come i tentativi di accesso e l'uso di operazioni privilegiate.

3. Procedure consigliate di sicurezza IIS per contenitori Windows

L'implementazione delle migliori pratiche IIS nei contenitori Windows è importante per diversi motivi, garantendo che le applicazioni siano sicure, ad alte prestazioni e scalabili. Sebbene i container offrano isolamento e un ambiente leggero, richiedono comunque una configurazione corretta per evitare vulnerabilità e problemi operativi. Ecco perché è fondamentale seguire le migliori pratiche per IIS nei contenitori Windows:

Sicurezza

- Prevenzione delle vulnerabilità comuni: IIS è spesso oggetto di attacchi come il cross-site scripting (XSS), il clickjacking e la divulgazione di informazioni. L'implementazione di header di sicurezza (ad esempio X-Content-Type-Options, e Strict-Transport-Security) aiuta a proteggere l'applicazione da X-Frame-Options queste minacce.
- L'isolamento non è sufficiente: i contenitori sono isolati, ma un'istanza IIS non configurata correttamente può esporre informazioni riservate, come dettagli sulla versione del server, elenchi di directory o comunicazioni non crittografate. Disattivando funzionalità come l'esplorazione delle

directory e la rimozione dell'intestazione della versione di IIS, si riduce al minimo la superficie di attacco.

- Crittografia e HTTPS: le migliori pratiche, come l'applicazione di connessioni solo HTTPS, garantiscono che i dati in transito siano crittografati, proteggendo le informazioni sensibili dall'intercettazione.

Prestazioni

- Uso efficiente delle risorse: le migliori pratiche IIS, come l'abilitazione della compressione dinamica e statica, riducono l'utilizzo della larghezza di banda e migliorano i tempi di caricamento. Queste ottimizzazioni sono particolarmente importanti negli ambienti containerizzati, in cui le risorse sono condivise tra i container e il sistema host.
- Registrazione ottimizzata: la corretta configurazione della registrazione (ad esempio, includendo l' X-Forwarded-Forintestazione) garantisce la possibilità di tracciare l'attività del client riducendo al minimo il sovraccarico di registrazione non necessario. Ciò consente di raccogliere dati pertinenti per la risoluzione dei problemi senza compromettere le prestazioni.

Scalabilità e manutenibilità

- Coerenza tra gli ambienti: seguendo le migliori pratiche, assicuri che la configurazione IIS sia coerente tra più istanze di container. Ciò semplifica la scalabilità e garantisce che, quando vengono implementati nuovi contenitori, rispettino le stesse linee guida in materia di sicurezza e prestazioni.
- Configurazioni automatizzate: le migliori pratiche in Dockerfiles, come l'impostazione delle autorizzazioni per le cartelle e la disabilitazione di funzionalità non necessarie, garantiscono che ogni nuovo contenitore sia configurato automaticamente correttamente. Ciò riduce l'intervento manuale e riduce il rischio di errore umano.

Conformità

- Soddisfazione dei requisiti normativi: molti settori hanno requisiti normativi rigorosi (ad esempio, PCI-DSS, HIPAA) che impongono misure di sicurezza specifiche, come le comunicazioni crittografate (HTTPS) e la registrazione delle richieste dei client. Seguire le migliori pratiche IIS nei contenitori aiuta a garantire la conformità a questi standard.
- Verificabilità: l'implementazione di politiche di audit e la registrazione sicura consentono la tracciabilità degli eventi, che è fondamentale negli audit. Ad esempio, la registrazione dell' X-

Forwarded-Forintestazione garantisce che gli indirizzi IP dei client vengano registrati correttamente nelle architetture basate su proxy.

Riduzione al minimo dei rischi negli ambienti condivisi

- Evitare configurazioni errate: i container condividono il kernel dell'host e, sebbene siano isolati l'uno dall'altro, un'istanza IIS mal configurata potrebbe esporre vulnerabilità o creare problemi di prestazioni. Le best practice garantiscono che ogni istanza IIS funzioni in modo ottimale, riducendo il rischio di problemi tra container.
- Accesso con privilegi minimi: l'impostazione delle autorizzazioni appropriate per cartelle e file all'interno del contenitore (ad esempio, l'utilizzo di Set-Acl in PowerShell) garantisce che gli utenti e i processi all'interno del contenitore abbiano solo l'accesso necessario, riducendo il rischio di aumento dei privilegi o di manomissione dei dati.

Resilienza in ambienti effimeri

- Natura effimera dei contenitori: i contenitori sono spesso di breve durata e vengono ricostruiti frequentemente. L'applicazione delle migliori pratiche IIS garantisce che ogni contenitore sia configurato in modo sicuro e coerente, indipendentemente dal numero di volte in cui viene ridistribuito. Ciò impedisce l'introduzione di configurazioni errate nel tempo.
- Mitigazione delle potenziali configurazioni errate: applicando automaticamente le migliori pratiche (ad esempio, disabilitando protocolli o intestazioni deboli), il rischio di una configurazione errata durante il riavvio o l'aggiornamento dei container è ridotto al minimo.

File Docker:

```
# Enforce HTTPS (disable HTTP) -- Only if container is target for SSL termination
RUN powershell -Command \
    "$httpBinding = Get-WebBinding -Name 'Default Web Site' -Protocol http | Where-
Object { $_.bindingInformation -eq '*:80:' }; \
    if ($httpBinding) { Remove-WebBinding -Name 'Default Web Site' -Protocol http -Port
 80; } \
    $httpsBinding = Get-WebBinding -Name 'Default Web Site' -Protocol https | Where-
Object { $_.bindingInformation -eq '*:443:' }; \
    if (-not $httpsBinding) { New-WebBinding -Name 'Default Web Site' -Protocol https -
Port 443 -IPAddress '*'; }"

# Use secure headers
```

```
RUN powershell -Command \
    "Write-Host 'Adding security headers...'; \
    Add-WebConfigurationProperty -pspath 'MACHINE/WEBROOT/APPHOST' -filter
 'system.applicationHost/sites/siteDefaults/logFile/customFields' -name
 "." -value @{logFieldName='X-Forwarded-For';sourceName='X-Forwarded-
For';sourceType='RequestHeader'}; \
    Add-WebConfigurationProperty -pspath 'MACHINE/WEBROOT/APPHOST' -filter
 "system.webServer/httpProtocol/customHeaders" -name "." -value @{name='Strict-
Transport-Security';value='max-age=31536000; includeSubDomains'}; \
    Add-WebConfigurationProperty -pspath 'MACHINE/WEBROOT/APPHOST' -filter
 "system.webServer/httpProtocol/customHeaders" -name "." -value @{name='X-Content-Type-
Options';value='nosniff'}; \
    Add-WebConfigurationProperty -pspath 'MACHINE/WEBROOT/APPHOST' -filter
 "system.webServer/httpProtocol/customHeaders" -name "." -value @{name='X-XSS-
Protection';value='1; mode=block'}; \
    Add-WebConfigurationProperty -pspath 'MACHINE/WEBROOT/APPHOST' -filter
 "system.webServer/httpProtocol/customHeaders" -name "." -value @{name='X-Frame-
Options';value='DENY'};"

# Disable IIS version disclosure
RUN powershell -Command \
    "Write-Host 'Disabling IIS version disclosure...'; \
    Import-Module WebAdministration; \
    Set-WebConfigurationProperty -pspath 'MACHINE/WEBROOT/APPHOST' -filter
 "system.webServer/security/requestFiltering" -name "removeServerHeader" -value
 "true";"

# Set IIS Logging Best Practices
RUN powershell -Command \
    Set-WebConfigurationProperty -pspath 'MACHINE/WEBROOT/APPHOST' -filter
 "system.webServer/directoryBrowse" -name "enabled" -value "false"; \
    Set-WebConfigurationProperty -pspath 'MACHINE/WEBROOT/APPHOST' -filter
 "system.webServer/httpErrors" -name "existingResponse" -value "PassThrough"; \

# Enable IIS dynamic and static compression to optimize performance
RUN powershell -Command \
    "Write-Host 'Enabling IIS compression...'; \
    Enable-WindowsOptionalFeature -Online -FeatureName IIS-HttpCompressionDynamic; \
    Import-Module WebAdministration; \
    Set-WebConfigurationProperty -pspath 'MACHINE/WEBROOT/APPHOST' -filter
 "system.webServer/urlCompression" -name "doDynamicCompression" -value "true"; \
    Set-WebConfigurationProperty -pspath 'MACHINE/WEBROOT/APPHOST' -filter
 "system.webServer/urlCompression" -name "doStaticCompression" -value "true"
```

```
# Ensure proper folder permissions using PowerShell's Set-Acl

RUN powershell -Command \
    "Write-Host 'Setting folder permissions for IIS...'; \
    $path = 'C:\\inetpub\\wwwroot'; \
    $acl = Get-Acl $path; \
    $iusr = New-Object System.Security.Principal.NTAccount('IIS_IUSRS'); \
    $rule = New-Object System.Security.AccessControl.FileSystemAccessRule($iusr,
 'ReadAndExecute', 'ContainerInherit, ObjectInherit', 'None', 'Allow'); \
    $acl.SetAccessRule($rule); \
    $users = New-Object System.Security.Principal.NTAccount('Users'); \
    $rule2 = New-Object System.Security.AccessControl.FileSystemAccessRule($users,
 'ReadAndExecute', 'ContainerInherit, ObjectInherit', 'None', 'Allow'); \
    $acl.SetAccessRule($rule2); \
    Set-Acl -Path $path -AclObject $acl"
```

Spiegazione:

Questo comando configura IIS per registrare l' X-Forwarded-Forintestazione, che viene comunemente utilizzata per acquisire l'indirizzo IP del client originale quando una richiesta passa attraverso un proxy o un sistema di bilanciamento del carico. Per impostazione predefinita, IIS registra solo l'indirizzo IP del load balancer o del reverse proxy, quindi l'aggiunta di questo campo di registro personalizzato consente di tenere traccia del vero IP del client per il controllo della sicurezza, l'analisi e la risoluzione dei problemi.

1. X-Forwarded-For intestazione che viene comunemente utilizzata per acquisire l'indirizzo IP del client originale quando una richiesta passa attraverso un proxy o un sistema di bilanciamento del carico. Per impostazione predefinita, IIS registra solo l'indirizzo IP del load balancer o del reverse proxy, quindi l'aggiunta di questo campo di registro personalizzato consente di tenere traccia del vero IP del client per il controllo della sicurezza, l'analisi e la risoluzione dei problemi.
2. Strict-Transport-Security (HSTS) Assicura che i browser comunichino solo tramite HTTPS. Il max-age=31536000 specifica che questa politica viene applicata per 1 anno e la applica a tutti i sottodomini. includeSubDomains
3. X-Content-Type-Options Impedisce ai browser di «sniffare MIME» una risposta diversa dal Content-Type dichiarato. Questo aiuta a prevenire alcuni tipi di attacchi.
4. X-XSS-Protection Abilita la protezione Cross-Site Scripting (XSS) nei browser.
5. X-Frame-Options Impedisce che la pagina venga incorporata negli iframe, proteggendola dagli attacchi di clickjacking.

6. Disabilita la divulgazione della versione di IIS Questo comando disabilita l'intestazione Server nelle risposte HTTP, che per impostazione predefinita rivela la versione di IIS utilizzata. Nascondere queste informazioni aiuta a ridurre il rischio che gli aggressori identifichino e prendano di mira le vulnerabilità specifiche della versione IIS.
7. Abilita connessioni solo HTTPS Questa sezione (commentata) applica le connessioni HTTPS e disabilita HTTP. Se non è commentato, il Dockerfile configurerà IIS in modo che ascolti solo sulla porta 443 (HTTPS) e rimuoverà il binding HTTP predefinito sulla porta 80. Ciò è utile per terminare SSL all'interno del contenitore e garantisce che tutto il traffico sia crittografato.
8. Disabilita la navigazione nelle directory Impedisce a IIS di mostrare un elenco di directory quando non è presente alcun documento predefinito. In questo modo si evita di esporre la struttura interna dei file agli utenti.
9. Passa attraverso pagine di errore personalizzate Assicura che, se l'applicazione dispone di una propria gestione degli errori, IIS lascerà passare le pagine di errore dell'applicazione anziché mostrare le pagine di errore IIS predefinite.
10. La modalità di errore dettagliata configura IIS per visualizzare messaggi di errore dettagliati solo per le richieste locali, aiutando gli sviluppatori a diagnosticare i problemi senza esporre informazioni riservate agli utenti esterni.
11. Garantire le autorizzazioni corrette per le cartelle Questo blocco configura le autorizzazioni delle cartelle per la radice web IIS (C:\inetpub\wwwroot). Imposta le autorizzazioni di lettura ed esecuzione per i gruppi IIS_IUSRS e Users, assicurando che questi utenti possano accedere alla cartella ma non modificare i file. L'impostazione delle autorizzazioni corrette riduce al minimo il rischio di accesso non autorizzato o di manomissione dei file ospitati dal server Web.

L'applicazione delle best practice IIS in Windows Containers garantisce che le applicazioni containerizzate siano sicure, ad alte prestazioni e scalabili. Queste pratiche aiutano a prevenire le vulnerabilità, ottimizzare l'utilizzo delle risorse, garantire la conformità e mantenere la coerenza tra le istanze dei container. Anche se i contenitori sono progettati per essere isolati, è necessaria una configurazione corretta per ridurre al minimo i rischi e garantire l'affidabilità dell'applicazione negli ambienti di produzione.

4. Principio del privilegio minimo

Il Principle of Least Privilege (PoLP) è fondamentale per i contenitori Windows per diversi motivi importanti, in particolare per migliorare la sicurezza e ridurre al minimo i rischi all'interno degli ambienti containerizzati. Questo principio impone che un sistema o un'applicazione funzionino con il

livello minimo di autorizzazioni necessario per funzionare correttamente. Ecco perché è importante nei contenitori Windows:

Riduzione al minimo della superficie di attacco

- I container spesso eseguono applicazioni che interagiscono con vari componenti del sistema e, maggiore è il numero di privilegi di cui dispone un'applicazione, maggiore è il suo accesso a tali componenti. Limitando le autorizzazioni del contenitore solo a ciò che è necessario, PoLP riduce significativamente la superficie di attacco, rendendo più difficile per un utente malintenzionato sfruttare il contenitore in caso di compromissione.

Limitazione dell'impatto dei contenitori compromessi

- Se un contenitore Windows è compromesso, l'esecuzione di applicazioni con privilegi eccessivi (ad esempio, accesso di amministratore o a livello di root) potrebbe consentire a un utente malintenzionato di ottenere il controllo sui file di sistema critici o di aumentare i privilegi sull'host del contenitore. Applicando PoLP, anche in caso di violazione di un container, l'aggressore è limitato a ciò che può fare, impedendo un'ulteriore escalation e l'accesso a risorse sensibili o ad altri contenitori.

Protezione in ambienti multi-tenant

- In ambienti cloud o aziendali, è possibile eseguire più container sulla stessa infrastruttura fisica o virtuale. PoLP garantisce che un contenitore compromesso non abbia la possibilità di accedere a risorse o dati appartenenti ad altri tenant. Questo isolamento è fondamentale per mantenere la sicurezza in ambienti condivisi e multitenant, proteggendo dai movimenti laterali tra i contenitori.

Mitigare l'escalation dei privilegi

- I contenitori che funzionano con privilegi elevati possono essere utilizzati dagli aggressori per aumentare i privilegi all'interno del sistema. PoLP mitiga questo rischio limitando l'accesso del contenitore alle risorse di sistema, prevenendo in tal modo azioni non autorizzate o escalation di privilegi oltre l'ambiente del container.

Conformità e controllo

- Molti standard normativi e quadri di sicurezza (ad esempio PCI DSS, HIPAA, GDPR) richiedono che i sistemi aderiscano al PoLP per limitare l'accesso ai dati sensibili. L'esecuzione di contenitori Windows con privilegi limitati aiuta le organizzazioni a rispettare queste normative e garantisce che alle applicazioni venga concesso l'accesso solo alle risorse di cui hanno specificamente bisogno.

Riduzione del rischio di errori di configurazione

- Quando i container vengono utilizzati con privilegi non necessari, anche un piccolo errore di configurazione può portare a gravi vulnerabilità di sicurezza. Ad esempio, se un container che funziona come amministratore viene accidentalmente esposto a Internet, un utente malintenzionato potrebbe ottenere il controllo del sistema. PoLP aiuta a prevenire tali rischi impostando di default privilegi limitati, rendendo meno pericolose le configurazioni errate.

Posizione di sicurezza dei container migliorata

- Seguendo PoLP, i contenitori vengono isolati meglio dal sistema host sottostante e gli uni dagli altri. Ciò garantisce che l'applicazione containerizzata abbia meno probabilità di accedere o modificare file o processi di sistema al di fuori dell'ambito definito, preservando l'integrità del sistema operativo host e di altri carichi di lavoro.

File Docker:

```
# Strongly recommended that when deploying a Windows server container to any multi-
tenant environment that your application runs via the ContainerUser account
USER ContainerUser
```

Spiegazione:

In questa sezione, il ContainerUser comando USER specifica che l'applicazione all'interno del contenitore Windows deve essere eseguita con l' ContainerUser account anziché con l'account amministratore predefinito.

Ecco perché è importante, specialmente in un ambiente multitenant:

1. Principio del privilegio minimo: l' ContainerUser account è un utente non amministrativo con privilegi limitati. L'esecuzione dell'applicazione con questo account rispetta il principio del privilegio minimo, che aiuta a ridurre al minimo il rischio di sfruttamento. Se un utente malintenzionato

dovesse compromettere l'applicazione, avrebbe un accesso limitato al sistema, riducendo i danni potenziali.

2. Sicurezza avanzata: in ambienti multitenant, i container possono condividere la stessa infrastruttura sottostante. Running as ContainerUser garantisce che, anche se un container viene compromesso, non disporrà dei privilegi amministrativi per accedere o modificare file di sistema critici o altri contenitori. Ciò riduce notevolmente la superficie di attacco.
3. Evitare l'accesso root: per impostazione predefinita, i contenitori potrebbero funzionare con autorizzazioni elevate (simili all'accesso root nei contenitori Linux), il che può essere pericoloso se sfruttato. L'utilizzo ContainerUser garantisce che l'applicazione non venga eseguita con diritti amministrativi non necessari, rendendo più difficile per gli aggressori aumentare i privilegi.
4. Best practice per ambienti multi-tenant: negli ambienti in cui più utenti o organizzazioni condividono la stessa infrastruttura (ad esempio nel cloud), la sicurezza è fondamentale. L'esecuzione di applicazioni con autorizzazioni limitate impedisce all'applicazione di un tenant di influire sulle altre, proteggendo i dati e le risorse sensibili su tutta la piattaforma.

Il ContainerUser comando USER garantisce che l'applicazione venga eseguita con privilegi minimi, migliorando la sicurezza negli ambienti multitenant limitando i danni che potrebbero verificarsi in caso di compromissione del contenitore. Si tratta di una procedura ottimale per prevenire accessi non autorizzati o l'escalation dei privilegi in un ambiente containerizzato.

Il Principle of Least Privilege è essenziale per i contenitori Windows perché limita il potenziale impatto delle violazioni della sicurezza, riduce la superficie di attacco e impedisce l'accesso non autorizzato ai componenti critici del sistema. Eseguendo applicazioni containerizzate con solo le autorizzazioni necessarie, le organizzazioni possono migliorare in modo significativo la sicurezza e la stabilità dei propri ambienti container, specialmente nelle infrastrutture multitenant e condivise.

Considerazioni finali: perché proteggere i contenitori Windows è fondamentale nell'odierno panorama delle minacce

Nel mondo digitale odierno in rapida evoluzione, in cui le minacce stanno diventando sempre più sofisticate e abbondanti, proteggere i contenitori Windows non è solo una raccomandazione, è una necessità assoluta. I container offrono un modo leggero e flessibile per impacchettare e distribuire applicazioni, ma non sono immuni dalle vulnerabilità di sicurezza. Man mano che sempre più aziende adottano i container per semplificare la propria infrastruttura, se non adeguatamente protetti, diventano anche un potenziale bersaglio di attacchi informatici.

Internet è invaso da varie minacce, che vanno da attori malintenzionati che prendono di mira vulnerabilità prive di patch a bot automatizzati che scansionano configurazioni errate. Senza le giuste misure di sicurezza, i container possono essere sfruttati per esporre dati sensibili, aumentare i privilegi o fungere da punti di ingresso per attacchi che possono compromettere l'infrastruttura più ampia. Ciò rende la sicurezza dei container tanto importante quanto la protezione di qualsiasi altra parte dell'ambiente.

Quando si utilizzano contenitori Windows, si applicano ancora molte best practice di sicurezza tradizionali. L'implementazione di solide politiche di account, la protezione delle configurazioni IIS, l'applicazione di HTTPS, l'utilizzo di rigide regole firewall e l'applicazione dell'accesso con privilegi minimi ai file critici sono tutte misure chiave che garantiscono la resilienza del contenitore agli attacchi. Inoltre, controlli e registrazioni regolari forniscono visibilità su ciò che accade all'interno del container, consentendoti di catturare attività sospette prima che si trasformino in un vero e proprio incidente.

La protezione dei contenitori Windows è inoltre conforme ai requisiti normativi che impongono la protezione dei dati sensibili e la garanzia dell'integrità delle applicazioni. Man mano che le architetture cloud-native e containerizzate diventano sempre più diffuse, garantire la sicurezza a ogni livello, dall'immagine di base al container in esecuzione, contribuirà a salvaguardare le operazioni e a mantenere la fiducia dei clienti.

In sintesi, l'ascesa delle applicazioni containerizzate, unita al crescente numero di minacce informatiche, rende la sicurezza dei container un aspetto non negoziabile della moderna gestione dell'infrastruttura. Aderendo alle migliori pratiche e monitorando continuamente le vulnerabilità, le aziende possono sfruttare l'agilità e l'efficienza dei contenitori Windows senza compromettere la sicurezza. In questo ambiente ricco di minacce, proteggere i contenitori Windows non è solo un'opzione, è un must.

Le migliori pratiche per le implementazioni ibride

Questa guida fornisce indicazioni sull'esecuzione di implementazioni in ambienti on-premise o edge con EKS Hybrid Nodes o EKS Anywhere.

Al momento abbiamo pubblicato guide per i seguenti argomenti:

- Best practice per i nodi ibridi EKS e le disconnessioni di rete

Nodi ibridi EKS e disconnessioni di rete

L'architettura EKS Hybrid Nodes può essere una novità per i clienti abituati a gestire cluster Kubernetes locali interamente nei propri data center o edge location. Con EKS Hybrid Nodes, il piano di controllo Kubernetes viene eseguito in una regione AWS e solo i nodi vengono eseguiti in locale, con il risultato di un'architettura cluster Kubernetes «estesa» o «estesa».

Ciò porta a una domanda comune: «Cosa succede se i miei nodi vengono disconnessi dal piano di controllo di Kubernetes?»

In questa guida, risponderemo a questa domanda esaminando i seguenti argomenti. Si consiglia di convalidare la stabilità e l'affidabilità delle applicazioni tramite disconnessioni di rete, poiché ogni applicazione può comportarsi in modo diverso in base alle dipendenze, alla configurazione e all'ambiente. Consulta il eks-hybrid-examples GitHub repository aws-samples/ per la configurazione dei test, le procedure e i risultati a cui puoi fare riferimento per testare le disconnessioni di rete con EKS Hybrid Nodes e le tue applicazioni. Il GitHub repository contiene anche dettagli aggiuntivi sui test utilizzati per convalidare il comportamento spiegato in questa guida.

- Le migliori pratiche per la stabilità attraverso le disconnessioni di rete
- Comportamento del failover del pod Kubernetes tramite disconnessioni di rete
- Traffico di rete delle applicazioni tramite disconnessioni di rete
- Credenziali di hosting tramite disconnessioni di rete

Le migliori pratiche per la stabilità tramite le disconnessioni di rete

Rete ad alta disponibilità

L'approccio migliore per evitare disconnessioni di rete tra i nodi ibridi e il piano di controllo Kubernetes consiste nell'utilizzare connessioni ridondanti e resilienti dall'ambiente locale da e verso AWS. Consulta la documentazione di AWS Direct Connect Resiliency Toolkit e AWS Site-to-Site VPN per ulteriori informazioni sull'architettura di reti ibride ad alta disponibilità con tali soluzioni.

Applicazioni ad alta disponibilità

Quando progettate le applicazioni, prendete in considerazione i domini di errore e gli effetti dei diversi tipi di interruzioni. Kubernetes fornisce meccanismi integrati per distribuire e gestire le repliche delle applicazioni su domini di nodi, zone e regioni. L'uso di questi meccanismi dipende dall'architettura dell'applicazione, dagli ambienti e dai requisiti di disponibilità. Ad esempio, le applicazioni stateless possono spesso essere distribuite con più repliche e possono spostarsi tra host e capacità di infrastruttura arbitrari, inoltre è possibile utilizzare selettori di nodi e vincoli di diffusione della topologia per eseguire istanze dell'applicazione su domini diversi. Per i dettagli sulle tecniche a livello di applicazione per creare applicazioni resilienti su Kubernetes, consulta la EKS Best Practices Guide.

Kubernetes valuta le informazioni zonali per i nodi che sono disconnessi dal piano di controllo di Kubernetes per determinare se spostare i pod su altri nodi. Se tutti i nodi di una zona non sono raggiungibili, Kubernetes annulla gli sgomberi dei pod per i nodi di quella zona. Come best practice, se disponi di un'implementazione con nodi in esecuzione in più data center o sedi fisiche, assegna una zona a ciascun nodo in base al relativo data center o alla posizione fisica. Quando esegui EKS con nodi nel cloud, questa etichetta di zona viene applicata automaticamente da AWS cloud-controller-manager. Tuttavia, a non cloud-controller-manager viene utilizzato con i nodi ibridi, quindi puoi passare queste informazioni attraverso la configurazione di kubelet. Di seguito è riportato un esempio di come configurare una zona nella configurazione del nodo per i nodi ibridi. La configurazione viene passata quando si connettono i nodi ibridi al cluster con la CLI dei nodi ibridi ()`nodeadm`. Per ulteriori informazioni sull'`topology.kubernetes.io/zone`etichetta, consulta la documentazione di Kubernetes. Per ulteriori informazioni sulla CLI dei nodi ibridi, vedere il riferimento nodeadm di Hybrid Nodes.

```
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
    name: my-cluster
```

```
    region: my-region
  kubelet:
    flags:
       - --node-labels=topology.kubernetes.io/zone=dc1
  hybrid:
    ...
```

Monitoraggio della rete

Se utilizzi AWS Direct Connect o AWS Site-to-Site VPN per la tua connettività ibrida, puoi sfruttare CloudWatch allarmi, log e metriche per osservare lo stato della tua connessione ibrida e diagnosticare i problemi. Per ulteriori informazioni, consulta Monitoraggio delle risorse AWS Direct Connect e Monitoraggio di una connessione AWS Site-to-Site VPN.

Si consiglia di creare allarmi per `NodeNotReady` gli eventi segnalati dall' node-lifecycle-controlleresecuzione sul piano di controllo EKS, che segnalano che un nodo ibrido potrebbe subire una disconnessione dalla rete. È possibile creare questo allarme abilitando la registrazione del piano di controllo EKS per Controller Manager e creando un filtro metrico CloudWatch per il messaggio «Recording status change event message for node» con status= «». NodeNotReady Dopo aver creato un filtro metrico, puoi creare un allarme per questo filtro in base alle soglie desiderate. Per ulteriori informazioni, consulta Alarming for logs nella documentazione. CloudWatch

Puoi utilizzare le metriche integrate Transit Gateway (TGW) e Virtual Private Gateway (VGW) per osservare il traffico di rete in entrata e in uscita dal TGW o VGW. È possibile creare allarmi per queste metriche per rilevare scenari in cui il traffico di rete scende al di sotto dei livelli normali, indicando un potenziale problema di rete tra i nodi ibridi e il piano di controllo EKS. Le metriche TGW e VGW sono descritte nella tabella seguente.

Gateway	Metrica	Description
Gateway di transito	BytesIn	I byte ricevuti da TGW dall'allegato (dal piano di controllo EKS ai nodi ibridi)
Gateway di transito	BytesOut	I byte inviati da TGW all'allegato (nodi ibridi al piano di controllo EKS)
Gateway virtuale privato	TunnelDataIn	I byte inviati dal lato AWS della connessione attraverso il tunnel VPN al gateway del cliente (dal piano di controllo EKS ai nodi ibridi)

Gateway	Metrica	Description
Gateway virtuale privato	TunnelDataOut	I byte ricevuti sul lato AWS della connessione attraverso il tunnel VPN dal gateway del cliente (nodi ibridi al piano di controllo EKS)

Puoi anche utilizzare CloudWatch Network Monitor per ottenere informazioni più approfondite sulle tue connessioni ibride per ridurre il tempo medio di ripristino e determinare se i problemi di rete hanno origine in AWS o nel tuo ambiente. CloudWatch Network Monitor può essere utilizzato per visualizzare la perdita di pacchetti e la latenza nelle connessioni di rete ibride, impostare avvisi e soglie e quindi intervenire per migliorare le prestazioni di rete. Per ulteriori informazioni, consulta Utilizzo di Amazon CloudWatch Network Monitor.

EKS offre diverse opzioni per monitorare lo stato dei cluster e delle applicazioni. Per quanto riguarda lo stato dei cluster, è possibile utilizzare la dashboard di osservabilità nella console EKS per rilevare, risolvere e risolvere rapidamente i problemi. Puoi anche utilizzare Amazon Managed Service per Prometheus, AWS Distro for Open Telemetry (ADOT) CloudWatch e per il monitoraggio di cluster, applicazioni e infrastrutture. Per ulteriori informazioni sulle opzioni di osservabilità EKS, consulta Monitora le prestazioni del cluster e visualizza i log.

Risoluzione dei problemi locali

Per prepararti alle disconnessioni di rete tra i nodi ibridi e il piano di controllo EKS, puoi configurare backend di monitoraggio e registrazione secondari per mantenere l'osservabilità delle applicazioni quando i servizi AWS regionali non sono raggiungibili. Ad esempio, puoi configurare il raccoglitore AWS Distro for Open Telemetry (ADOT) per inviare metriche e log a più backend. Puoi anche utilizzare strumenti locali, come la `crictl` CLI, per interagire localmente con pod e contenitori in sostituzione di `kubectl` o altri client compatibili con l'API Kubernetes che in genere interrogano l'endpoint del server API Kubernetes. Per ulteriori informazioni, consulta la documentazione in cri-tools. `crictl` `crictl` GitHub Di seguito sono elencati alcuni `crictl` comandi utili.

Elenca i pod in esecuzione sull'host:

```
crictl pods
```

Elenca i contenitori in esecuzione sull'host:

```
crictl ps
```

Elenca le immagini in esecuzione sull'host:

```
crictl images
```

Ottieni i log di un contenitore in esecuzione sull'host:

```
crictl logs CONTAINER_NAME
```

Ottieni statistiche sui pod in esecuzione sull'host:

```
crictl statsp
```

Traffico di rete dell'applicazione

Quando si utilizzano nodi ibridi, è importante considerare e comprendere i flussi di rete del traffico delle applicazioni e le tecnologie utilizzate per esporre le applicazioni esternamente al cluster. Diverse tecnologie per il bilanciamento del carico delle applicazioni e l'ingresso si comportano in modo diverso durante le disconnessioni di rete. Ad esempio, se si utilizza la funzionalità BGP Control Plane di Cilium per il bilanciamento del carico delle applicazioni, la sessione BGP per i pod e i servizi potrebbe essere inattiva durante le disconnessioni di rete. Ciò accade perché la funzionalità degli altoparlanti BGP è integrata con l'agente Cilium e l'agente Cilium si riavvierà continuamente quando viene disconnesso dal piano di controllo Kubernetes. Il motivo del riavvio è dovuto al fallimento del controllo dello stato di Cilium perché il suo stato di salute è associato all'accesso al piano di controllo di Kubernetes (vedi CFP: #31702 con un miglioramento opt-in in Cilium v1.17). Allo stesso modo, se utilizzi Application Load Balancers (ALB) o Network Load Balancers (NLB) per il traffico applicativo originato dalla regione AWS, tale traffico potrebbe essere temporaneamente inattivo se l'ambiente locale perde la connettività alla regione AWS. Si consiglia di verificare che le tecnologie utilizzate per il bilanciamento del carico e l'ingresso rimangano stabili durante le disconnessioni di rete prima di passare alla produzione. L'esempio nel eks-hybrid-examples GitHub repo aws-samples/ utilizza MetallB per il bilanciamento del carico in modalità L2, che rimane stabile durante le disconnessioni di rete tra i nodi ibridi e il piano di controllo EKS.

Verifica le dipendenze dai servizi AWS remoti

Quando usi nodi ibridi, fai attenzione alle dipendenze che assumi dai servizi AWS regionali esterni al tuo ambiente locale o perimetrale. Gli esempi includono l'accesso ad Amazon S3 o Amazon RDS per

i dati delle applicazioni, l'utilizzo di Amazon Managed Service for Prometheus o per metriche e log, l'utilizzo di Application and Network Load Balancer CloudWatch per il traffico originato dalla regione e il recupero di contenitori da Amazon Elastic Container Registry. Questi servizi non saranno accessibili durante le disconnessioni di rete tra l'ambiente locale e AWS. Se il tuo ambiente locale è soggetto a disconnessioni di rete con AWS, esamina l'utilizzo dei servizi AWS e assicurati che la perdita di una connessione a tali servizi non comprometta la stabilità statica delle tue applicazioni.

Ottimizza il comportamento di failover del pod Kubernetes

Esistono opzioni per ottimizzare il comportamento di failover dei pod durante le disconnessioni di rete per applicazioni che non sono portabili tra host o per ambienti con risorse limitate che non dispongono di capacità di riserva per il failover dei pod. In genere, è importante considerare i requisiti di risorse delle applicazioni e disporre di una capacità sufficiente per consentire il failover di una o più istanze dell'applicazione su un host diverso in caso di guasto di un nodo.

- Opzione 1 - Utilizzo DaemonSets: questa opzione si applica alle applicazioni che possono e devono essere eseguite su tutti i nodi del cluster. DaemonSets sono configurati automaticamente per tollerare la contaminazione irraggiungibile, che mantiene i DaemonSet pod legati ai rispettivi nodi attraverso le disconnessioni di rete.
- Opzione 2 - Ottimizzazione in caso `tolerationSeconds` di contaminazione irraggiungibile: puoi regolare la quantità di tempo in cui i pod rimangono legati ai nodi durante le disconnessioni di rete. A tale scopo, configurate i pod dell'applicazione in modo che tollerino la contaminazione irraggiungibile con l'`NoExecute`effetto per una durata specificata (nelle specifiche dell'applicazione). `tolerationSeconds` Con questa opzione, in caso di disconnessioni di rete, i pod delle applicazioni rimangono associati ai nodi fino alla scadenza. `tolerationSeconds` Consideratela con attenzione, perché se aumentate `tolerationSeconds` per `NoExecute` via della contaminazione irraggiungibile, i pod in esecuzione su host irraggiungibili potrebbero impiegare più tempo per essere trasferiti su altri host funzionanti e raggiungibili.
- Opzione 3: Controller personalizzato: puoi creare ed eseguire un controller personalizzato (o altro software) che monitora Kubernetes per rilevare la contaminazione irraggiungibile che causa l'effetto. `NoExecute` Quando viene rilevata questa contaminazione, il controller personalizzato può controllare le metriche specifiche dell'applicazione per valutare lo stato dell'applicazione. Se l'applicazione è integra, il controller personalizzato può rimuovere la contaminazione irraggiungibile, impedendo l'eliminazione dei pod dai nodi durante le disconnessioni di rete.

Di seguito è riportato un esempio di come configurare un Deployment with `tolerationSeconds` for the unreachable taint. Nell'esempio, `tolerationSeconds` è impostato su `1800` (30 minuti), il che

significa che i pod in esecuzione su nodi irraggiungibili verranno rimossi solo se la disconnessione dalla rete dura più di 30 minuti.

```
apiVersion: apps/v1
kind: Deployment
metadata:
...
spec:
...
      tolerations:
      - key: "node.kubernetes.io/unreachable"
        operator: "Exists"
        effect: "NoExecute"
        tolerationSeconds: 1800
```

Failover dei pod Kubernetes tramite disconnessioni di rete

Iniziamo con una rassegna dei concetti, dei componenti e delle impostazioni chiave che influenzano il comportamento di Kubernetes durante le disconnessioni di rete tra i nodi e il piano di controllo Kubernetes. EKS è conforme a Kubernetes a monte, quindi tutti i concetti, i componenti e le impostazioni di Kubernetes descritti qui si applicano alle implementazioni EKS ed EKS Hybrid Nodes.

Sono stati apportati miglioramenti a EKS specificamente per migliorare il comportamento di failover dei pod durante le disconnessioni di rete. Per ulteriori informazioni, consulta i numeri #131294 e #131481 nell'archivio Kubernetes upstream. GitHub

Concetti

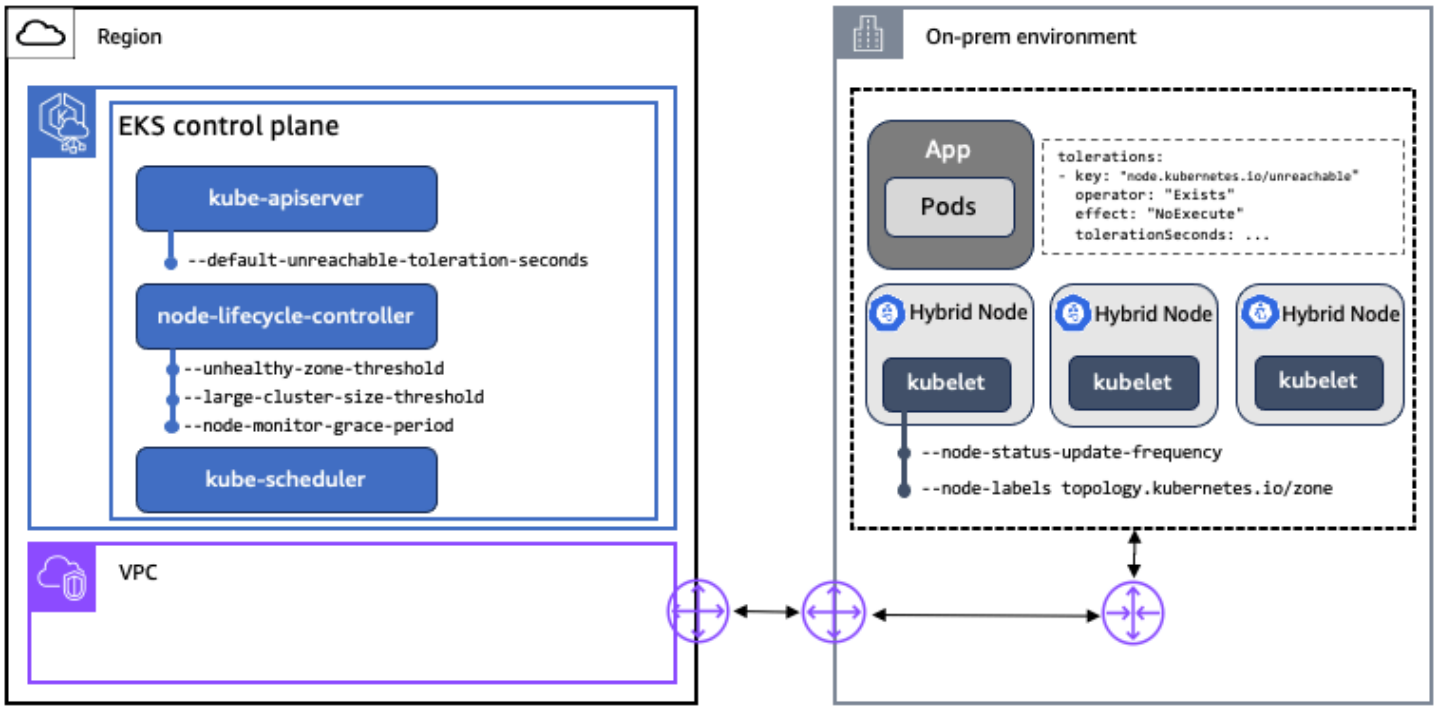
Tinte e tolleranze: le tinte e le tolleranze vengono utilizzate in Kubernetes per controllare la pianificazione dei pod sui nodi. I taint vengono impostati da node-lifecycle-controller per indicare che i nodi non sono idonei alla pianificazione o che i pod su tali nodi devono essere rimossi. Quando i nodi non sono raggiungibili a causa di una disconnessione dalla rete, applica node.kubernetes. node-lifecycle-controller io/unreachable taint with a NoSchedule effect, and with a NoExecute effect if certain conditions are met. The node.kubernetes.io/unreachabletaint NodeCondition corrisponde a Ready being Unknown. Gli utenti possono specificare le tolleranze per le macchie a livello di applicazione in. PodSpec

- NoSchedule: Non sono programmati nuovi Pod sul nodo contaminato a meno che non abbiano una tolleranza corrispondente. I pod già in esecuzione sul nodo non vengono eliminati.

- NoExecute: I pod che non tollerano la contaminazione vengono eliminati immediatamente. I pod che tollerano la contaminazione (senza specificare TolerationSeconds) rimangono legati per sempre. I pod che tollerano la contaminazione con un valore di tolleranza specificato rimangono vincolati per il periodo di tempo specificato. Trascorso tale periodo, il controller del ciclo di vita del nodo rimuove i Pod dal nodo.

Node Leases: Kubernetes utilizza l'API Lease per comunicare gli heartbeat dei nodi Kubelet al server dell'API Kubernetes. Per ogni nodo, esiste un oggetto Lease con un nome corrispondente. Internamente, ogni heartbeat kubelet aggiorna il campo spec.RenewTime dell'oggetto Lease. Il piano di controllo Kubernetes utilizza il timestamp di questo campo per determinare la disponibilità del nodo. Se i nodi sono disconnessi dal piano di controllo di Kubernetes, non possono aggiornare SPEC.RenewTime per il relativo Lease e il piano di controllo lo interpreta come Ready come Unknown. NodeCondition

Componenti



Componente	Sottocomponente	Description
Piano di controllo Kubernetes	kube-api-server	Il server API è un componente fondamentale del piano di controllo Kubernetes che espone l'API Kubernetes.

Componente	Sottocomponente	Description
Piano di controllo Kubernetes	node-lifecycle-controller	Uno dei controller che esegue. kube-controller-manager È responsabile del rilevamento e della risposta ai problemi dei nodi.
Piano di controllo Kubernetes	kube-scheduler	Un componente del piano di controllo che controlla i Pod appena creati senza un nodo assegnato e seleziona un nodo su cui eseguirli.
Nodi Kubernetes	kubelet	Un agente che viene eseguito su ogni nodo del cluster. Il kubelet controlla PodSpecs e garantisce che i contenitori descritti in essi PodSpecs siano funzionanti e integri.

Impostazioni di configurazione

Compone	Impostazione	Description	K8s (impostazione predefinita)	EKS predefinito	Configurabile in EKS
kube-api-server	default-unreachable-toleration-seconds	Indica la tolleranza `unreachable:NoExecute` che viene aggiunta `tolerationSeconds` di default a ogni pod che non ha già tale tolleranza.	300	300	No
node-lifecycle-controller	node-monitor-grace-period	Il periodo di tempo in cui un nodo può non rispondere prima di essere contrassegnato come non integro. Deve essere N volte superiore a quello di kubelet`nodeStatusUpdateFrequency` , dove N è il numero di tentativi consentiti affinché kubelet pubblichi lo stato del nodo.	40	40	No

Compor e	Impostazione	Description	K8s (impost ione predefi ta)	EKS predefi to	Configura bile in EKS
node-life cycle-con troller	large-cluster-size-threshold	Il numero di nodi in cui node-lifecycle-con troller considera il cluster come grande ai fini della logica di sfratto. `--secondary-node-eviction-rate` viene sostituit o a 0 per i cluster di queste dimensioni o inferiori.	50	100.000	No
node-life cycle-con troller	unhealthy-zone-threshold	La percentuale di nodi in una zona che deve essere Non pronta affinché tale zona venga considerata non integra.	55%	55%	No
kubelet	node-stat us-update-frequency	Con che frequenza Kubelet invia lo stato del nodo al piano di controllo. Deve essere compatibile con `nodeMonitorGracePe riod` in. node-lifecycle-controller	10	10	Sì
kubelet	etichette per nodi	Etichette da aggiungere durante la registrazione del nodo nel cluster. L'etichet ta `topology.kubernetes.io/zone` può essere specificata con nodi ibridi per raggruppare i nodi in zone.	Nessun	Nessun	Sì

Failover del pod Kubernetes tramite disconnessioni di rete

Il comportamento qui descritto presuppone che i pod funzionino come implementazioni Kubernetes con impostazioni predefinite e che EKS venga utilizzato come provider Kubernetes. Il comportamento effettivo potrebbe variare in base all'ambiente, al tipo di disconnessione della rete, alle applicazioni, alle dipendenze e alla configurazione del cluster. Il contenuto di questa guida è stato convalidato utilizzando un'applicazione specifica, una configurazione del cluster e un sottoinsieme di plugin. Si

consiglia vivamente di testare il comportamento nel proprio ambiente e con le proprie applicazioni prima di passare alla produzione.

In caso di disconnessioni di rete tra i nodi e il piano di controllo Kubernetes, il kubelet su ciascun nodo disconnesso non può comunicare con il piano di controllo Kubernetes. Di conseguenza, il kubelet non può rimuovere i pod su quei nodi finché la connessione non viene ripristinata. Ciò significa che i pod in esecuzione su quei nodi prima della disconnessione della rete continuano a funzionare durante la disconnessione, supponendo che nessun altro guasto li provochi la chiusura. In sintesi, è possibile raggiungere la stabilità statica durante le disconnessioni di rete tra i nodi e il piano di controllo di Kubernetes, ma non è possibile eseguire operazioni di modifica sui nodi o sui carichi di lavoro finché la connessione non viene ripristinata.

Esistono cinque scenari principali che producono diversi comportamenti di failover dei pod in base alla natura della disconnessione della rete. In tutti gli scenari, il cluster torna a funzionare senza l'intervento dell'operatore una volta che i nodi si riconnettono al piano di controllo di Kubernetes. Gli scenari seguenti descrivono i risultati attesi sulla base delle nostre osservazioni, ma questi risultati potrebbero non essere applicabili a tutte le possibili configurazioni di applicazioni e cluster.

Scenario 1: interruzione completa del cluster

Risultato previsto: i pod sui nodi irraggiungibili non vengono rimossi e continuano a funzionare su tali nodi.

Un'interruzione completa del cluster significa che tutti i nodi del cluster vengono disconnessi dal piano di controllo di Kubernetes. In questo scenario, il node-lifecycle-controller piano di controllo rileva che tutti i nodi del cluster sono irraggiungibili e annulla qualsiasi rimozione dei pod.

Gli amministratori del cluster vedranno tutti i nodi con stato durante la disconnessione. `Not Ready` Lo stato dei pod non cambia e non sono previsti nuovi pod su nessun nodo durante la disconnessione e la successiva riconnessione.

Scenario 2: interruzione dell'intera zona

Risultato previsto: i pod sui nodi irraggiungibili non vengono rimossi e continuano a funzionare su tali nodi.

Un'interruzione completa della zona significa che tutti i nodi della zona vengono disconnessi dal piano di controllo di Kubernetes. In questo scenario, il node-lifecycle-controller piano di controllo rileva che tutti i nodi della zona sono irraggiungibili e annulla qualsiasi sfratto dei pod.

Gli amministratori del cluster vedranno tutti i nodi con stato durante la disconnessione. `Not Ready` Lo stato dei pod non cambia e non sono previsti nuovi pod su nessun nodo durante la disconnessione e la successiva riconnessione.

Scenario 3: interruzione della zona maggioritaria

Risultato previsto: i pod sui nodi irraggiungibili non vengono rimossi e continuano a funzionare su tali nodi.

Un'interruzione della zona di maggioranza significa che la maggior parte dei nodi di una determinata zona viene disconnessa dal piano di controllo di Kubernetes. Le zone in Kubernetes sono definite da nodi con la stessa etichetta. `topology.kubernetes.io/zone` Se non è definita alcuna zona nel cluster, un'interruzione di maggiore entità significa che la maggior parte dei nodi dell'intero cluster viene disconnessa. Per impostazione predefinita, la maggioranza è definita da `sunhealthy-zone-threshold`, che node-lifecycle-controller è impostata al 55% sia in Kubernetes che in EKS. Poiché in EKS `large-cluster-size-threshold` è impostato su 100.000, se il 55% o più dei nodi di una zona non sono raggiungibili, lo sgombero dei pod viene annullato (dato che la maggior parte dei cluster è molto più piccola di 100.000 nodi).

Gli amministratori del cluster vedranno la maggior parte dei nodi della zona con lo stato `Not Ready` durante la disconnessione, ma lo stato dei pod non cambierà e non verranno riprogrammati su altri nodi.

Tieni presente che il comportamento sopra riportato si applica solo ai cluster più grandi di tre nodi. Nei cluster di tre nodi o meno, è previsto lo sgombero dei pod sui nodi non raggiungibili e i nuovi pod sui nodi integri.

Durante i test, abbiamo osservato occasionalmente che i pod venivano rimossi esattamente da un nodo irraggiungibile durante le disconnessioni di rete, anche quando la maggior parte dei nodi della zona era irraggiungibile. Stiamo ancora esaminando una possibile condizione di gara in Kubernetes come causa di questo comportamento. node-lifecycle-controller

Scenario 4: distruzione delle zone di minoranza

Risultato previsto: i pod vengono rimossi dai nodi irraggiungibili e i nuovi pod vengono programmati sui nodi idonei e disponibili.

Un'interruzione minoritaria significa che una percentuale inferiore di nodi in una zona viene disconnessa dal piano di controllo di Kubernetes. Se non è definita alcuna zona nel cluster, un'interruzione di minoranza significa che la minoranza di nodi dell'intero cluster viene disconnessa.

Come indicato, la minoranza è definita dall'impostazione di node-lifecycle-controller, che per `unhealthy-zone-threshold` impostazione predefinita è 55%. In questo scenario, se la disconnessione dalla rete dura più a lungo di `default-unreachable-toleration-seconds` (5 minuti) e `node-monitor-grace-period` (40 secondi) e meno del 55% dei nodi di una zona sono irraggiungibili, i nuovi pod vengono programmati sui nodi integri mentre i pod sui nodi irraggiungibili sono contrassegnati per l'eliminazione.

Gli amministratori dei cluster vedranno i nuovi pod creati su nodi integri, mentre i pod sui nodi disconnessi verranno visualizzati come. `Terminating` Ricorda che, anche se i pod sui nodi disconnessi hanno uno `Terminating` stato, non vengono completamente rimossi finché il nodo non si riconnette al piano di controllo di Kubernetes.

Scenario 5: riavvio del nodo durante un'interruzione della rete

Risultato previsto: i pod sui nodi irraggiungibili non vengono avviati finché i nodi non si riconnettono al piano di controllo di Kubernetes. Il failover dei pod segue la logica descritta negli Scenari 1—3, a seconda del numero di nodi non raggiungibili.

Un riavvio del nodo durante un'interruzione della rete significa che un altro errore (ad esempio un ciclo di alimentazione, un out-of-memory evento o un altro problema) si è verificato su un nodo contemporaneamente alla disconnessione della rete. I pod in esecuzione su quel nodo all'inizio della disconnessione di rete non vengono riavviati automaticamente durante la disconnessione, se anche il kubelet è stato riavviato. Kubelet interroga il server dell'API Kubernetes durante l'avvio per sapere quali pod deve eseguire. Se il kubelet non riesce a raggiungere il server API a causa di una disconnessione dalla rete, non può recuperare le informazioni necessarie per avviare i pod.

In questo scenario, gli strumenti di risoluzione dei problemi locali come la `crictl` CLI non possono essere utilizzati per avviare i pod manualmente come misura «break-glass». Kubernetes in genere rimuove i pod falliti e ne crea di nuovi anziché riavviare i pod esistenti (vedi #10213 nel repository containerd per i dettagli). GitHub I pod statici sono l'unico oggetto del carico di lavoro Kubernetes controllato dal kubelet e che possono essere riavviati durante questi scenari. Tuttavia, in genere non è consigliabile utilizzare pod statici per le distribuzioni di applicazioni. Invece, distribuisci più repliche su host diversi per garantire la disponibilità delle applicazioni in caso di più errori simultanei, come un guasto del nodo più una disconnessione di rete tra i tuoi nodi e il piano di controllo di Kubernetes.

Traffico di rete delle applicazioni tramite disconnessioni di rete

Gli argomenti di questa pagina sono relativi alla rete di cluster Kubernetes e al traffico delle applicazioni durante le disconnessioni di rete tra i nodi e il piano di controllo Kubernetes.

Cilium

Cilium dispone di diverse modalità per la gestione degli indirizzi IP (IPAM), l'incapsulamento, il bilanciamento del carico e il routing dei cluster. Le modalità convalidate in questa guida utilizzavano Cluster Scope IPAM, VXLAN overlay, BGP load balancing e kube-proxy. Cilium è stato utilizzato anche senza il bilanciamento del carico BGP, sostituendolo con il bilanciamento del carico MetalLB L2.

La base dell'installazione Cilium è costituita dall'operatore Cilium e dagli agenti Cilium. L'operatore Cilium funziona come distribuzione e registra le Cilium Custom Resource Definitions (CRDs), gestisce IPAM e sincronizza gli oggetti del cluster con il server API Kubernetes, tra le altre funzionalità. Gli agenti Cilium vengono eseguiti su ciascun nodo DaemonSet e gestiscono i programmi eBPF per controllare le regole di rete per i carichi di lavoro in esecuzione sul cluster.

In genere, il routing interno al cluster configurato da Cilium rimane disponibile e attivo durante le disconnessioni di rete, il che può essere confermato osservando i flussi di traffico all'interno del cluster e le regole della tabella IP (iptables) per la rete di pod.

```
ip route show table all | grep cilium
```

```
10.86.2.0/26 via 10.86.3.16 dev cilium_host proto kernel src 10.86.3.16 mtu 1450
10.86.2.64/26 via 10.86.3.16 dev cilium_host proto kernel src 10.86.3.16 mtu 1450
10.86.2.128/26 via 10.86.3.16 dev cilium_host proto kernel src 10.86.3.16 mtu 1450
10.86.2.192/26 via 10.86.3.16 dev cilium_host proto kernel src 10.86.3.16 mtu 1450
10.86.3.0/26 via 10.86.3.16 dev cilium_host proto kernel src 10.86.3.16
10.86.3.16 dev cilium_host proto kernel scope link
...
```

Tuttavia, durante le disconnessioni di rete, l'operatore Cilium e gli agenti Cilium si riavviano a causa dell'associazione dei controlli di integrità con l'integrità della connessione con il server API Kubernetes. Si prevede di vedere quanto segue nei registri dell'operatore Cilium e degli agenti Cilium durante le disconnessioni di rete. Durante le disconnessioni di rete, è possibile utilizzare strumenti come la `crictl` CLI per osservare i riavvii di questi componenti, compresi i relativi registri.

```
msg="Started gops server" address="127.0.0.1:9890" subsys=gops
msg="Establishing connection to apiserver" host="https://<k8s-cluster-ip>:443"
 subsys=k8s-client
msg="Establishing connection to apiserver" host="https://<k8s-cluster-ip>:443"
 subsys=k8s-client
```

```
msg="Unable to contact k8s api-server" error="Get \"https://<k8s-cluster-ip>:443/
api/v1/namespaces/kube-system\": dial tcp <k8s-cluster-ip>:443: i/o timeout"
 ipAddr="https://<k8s-cluster-ip>:443" subsys=k8s-client
msg="Start hook failed" function="client.(*compositeClientset).onStart
 (agent.infra.k8s-client)" error="Get \"https://<k8s-cluster-ip>:443/api/v1/namespaces/
kube-system\": dial tcp <k8s-cluster-ip>:443: i/o timeout"
msg="Start failed" error="Get \"https://<k8s-cluster-ip>:443/api/v1/namespaces/kube-
system\": dial tcp <k8s-cluster-ip>:443: i/o timeout" duration=1m5.003834026s
msg=Stopping
msg="Stopped gops server" address="127.0.0.1:9890" subsys=gops
msg="failed to start: Get \"https://<k8s-cluster-ip>:443/api/v1/namespaces/kube-system
\": dial tcp <k8s-cluster-ip>:443: i/o timeout" subsys=daemon
```

Se utilizzi la funzionalità BGP Control Plane di Cilium per il bilanciamento del carico delle applicazioni, la sessione BGP per i tuoi pod e servizi potrebbe non funzionare durante le disconnessioni di rete perché la funzionalità degli altoparlanti BGP è integrata con l'agente Cilium e l'agente Cilium si riavvierà continuamente quando viene disconnesso dal piano di controllo Kubernetes. Per ulteriori informazioni, consulta la Cilium BGP Control Plane Operation Guide nella documentazione di Cilium. Inoltre, se si verifica un errore simultaneo durante una disconnessione di rete, ad esempio un ciclo di alimentazione o il riavvio della macchina, le rotte Cilium non verranno preservate attraverso queste azioni, sebbene le rotte vengano ricreate quando il nodo si riconnette al piano di controllo di Kubernetes e Cilium si riavvia.

Calico

Presto in arrivo

Metallo B

MetallB ha due modalità per il bilanciamento del carico: modalità L2 e modalità BGP. Fai riferimento alla documentazione di MetallB per i dettagli sul funzionamento di queste modalità di bilanciamento del carico e sui relativi limiti. La convalida di questa guida ha utilizzato MetallB in modalità L2, in cui una macchina del cluster assume la proprietà del servizio Kubernetes e utilizza ARP per rendere gli indirizzi IP del sistema di bilanciamento del carico IPv4 raggiungibili sulla rete locale. Quando si esegue MetallB, c'è un controller responsabile dell'assegnazione degli IP e degli altoparlanti, in esecuzione su ciascun nodo, che sono responsabili dei servizi pubblicitari con indirizzi IP assegnati. Il controller MetallB funziona come Deployment e gli altoparlanti MetallB funzionano come un. DaemonSet Durante le disconnessioni di rete, il controller MetallB e gli altoparlanti non riescono a controllare le risorse del cluster nel server dell'API Kubernetes, ma continuano a funzionare.

Soprattutto, i Servizi che utilizzano MetallB per la connettività esterna rimangono disponibili e accessibili durante le disconnessioni di rete.

kube-proxy

Nei cluster EKS, kube-proxy funziona come un DaemonSet unico nodo ed è responsabile della gestione delle regole di rete per consentire la comunicazione tra servizi e pod traducendo gli indirizzi IP dei servizi negli indirizzi IP dei pod sottostanti. Le regole delle tabelle IP (iptables) configurate da kube-proxy vengono mantenute durante le disconnessioni di rete e il routing all'interno del cluster continua a funzionare e i pod kube-proxy continuano a funzionare.

È possibile osservare le regole kube-proxy con i seguenti comandi iptables. Il primo comando mostra che i pacchetti che attraversano la catena vengono indirizzati verso la `PREROUTING` catena. `KUBE-SERVICES`

```
iptables -t nat -L PREROUTING
```

```
Chain PREROUTING (policy ACCEPT)
target         prot opt source      destination
KUBE-SERVICES  all  --  anywhere    anywhere      /* kubernetes service portals */
```

Ispezionando la `KUBE-SERVICES` catena possiamo vedere le regole per i vari servizi del cluster.

```
Chain KUBE-SERVICES (2 references)
target                     prot opt source      destination
KUBE-SVL-NZTS37XDTDNXGCKJ  tcp  --  anywhere    172.16.189.136  /* kube-system/hubble-
peer:peer-service cluster IP /
KUBE-SVC-2BINP2AXJOTI3HJ5  tcp  --  anywhere    172.16.62.72    / default/metallb-
webhook-service cluster IP /
KUBE-SVC-LRNEBRA3Z5YGJ4QC  tcp  --  anywhere    172.16.145.111  / default/redis-leader
 cluster IP /
KUBE-SVC-I7SKRZYQ7PWYV5X7  tcp  --  anywhere    172.16.142.147  / kube-system/eks-
extension-metrics-api:metrics-api cluster IP /
KUBE-SVC-JD5MR3NA4I4DYORP  tcp  --  anywhere    172.16.0.10     / kube-system/kube-
dns:metrics cluster IP /
KUBE-SVC-TCOU7JCQXEZGVUNU  udp  --  anywhere    172.16.0.10     / kube-system/kube-
dns:dns cluster IP /
KUBE-SVC-ERIFXISQEP7F7OF4  tcp  --  anywhere    172.16.0.10     / kube-system/kube-
dns:dns-tcp cluster IP /
KUBE-SVC-ENODL3HWJ5BZY56Q  tcp  --  anywhere    172.16.7.26     / default/frontend
 cluster IP /
```

```
KUBE-EXT-ENODL3HWJ5BZY56Q  tcp  --  anywhere    <LB-IP>    / default/frontend
 loadbalancer IP /
KUBE-SVC-NPX46M4PTMTKRN6Y  tcp  --  anywhere    172.16.0.1      / default/
kubernetes:https cluster IP /
KUBE-SVC-YU5RV2YQWHLZ5XPR  tcp  --  anywhere    172.16.228.76   / default/redis-
follower cluster IP /
KUBE-NODEPORTS             all  --  anywhere    anywhere        / kubernetes service
 nodeports; NOTE: this must be the last rule in this chain */
```

Ispezionando la catena del servizio di frontend per l'applicazione possiamo vedere gli indirizzi IP dei pod che supportano il servizio.

```
iptables -t nat -L KUBE-SVC-ENODL3HWJ5BZY56Q
```

```
Chain KUBE-SVC-ENODL3HWJ5BZY56Q (2 references)
target                     prot opt source    destination
KUBE-SEP-EKXE7ASH7Y74BGBO  all  --  anywhere  anywhere    /* default/frontend ->
 10.86.2.103:80 / statistic mode random probability 0.33333333349
KUBE-SEP-GCY3OUXWSVMSEAR6  all  --  anywhere  anywhere    / default/frontend ->
 10.86.2.179:80 / statistic mode random probability 0.50000000000
KUBE-SEP-6GJJR3EF5AUP2WBU  all  --  anywhere  anywhere    / default/frontend ->
 10.86.3.47:80 */
```

I seguenti messaggi di log kube-proxy sono previsti durante le disconnessioni di rete mentre tenta di controllare il server dell'API Kubernetes per verificare la presenza di aggiornamenti alle risorse del nodo e dell'endpoint.

```
"Unhandled Error" err="k8s.io/client-go/informers/factory.go:160: Failed to watch
 *v1.Node: failed to list *v1.Node: Get \"https://<k8s-endpoint>/api/v1/nodes?
fieldSelector=metadata.name%3D<node-name>&resourceVersion=2241908\": dial tcp <k8s-
ip>:443: i/o timeout" logger="UnhandledError"
"Unhandled Error" err="k8s.io/client-go/informers/factory.go:160: Failed to watch
 *v1.EndpointSlice: failed to list *v1.EndpointSlice: Get \"https://<k8s-endpoint>/
apis/discovery.k8s.io/v1/endpointslices?labelSelector=%21service.kubernetes.io
%2Fheadless%2C%21service.kubernetes.io%2Fservice-proxy-name&resourceVersion=2242090\":
 dial tcp <k8s-ip>:443: i/o timeout" logger="UnhandledError"
```

CoreDNS

Per impostazione predefinita, i pod nei cluster EKS utilizzano l'indirizzo IP del cluster CoredNS come name server per le query DNS interne al cluster. Nei cluster EKS, CoredNS viene eseguito

come implementazione sui nodi. Con i nodi ibridi, i pod sono in grado di continuare a comunicare con CoreDNS durante le disconnessioni di rete quando ci sono repliche CoreDNS in esecuzione localmente su nodi ibridi. Se disponi di un cluster EKS con nodi nel cloud e nodi ibridi nell'ambiente locale, si consiglia di avere almeno una replica CoredNS in ogni ambiente. CoredNS continua a fornire query DNS per i record creati prima della disconnessione della rete e continua a funzionare durante la riconnessione di rete per la stabilità statica.

I seguenti messaggi di log CoredNS sono previsti durante le disconnessioni di rete mentre tenta di elencare gli oggetti dal server dell'API Kubernetes.

```
Failed to watch *v1.Namespace: failed to list *v1.Namespace: Get "https://<k8s-cluster-
ip>:443/api/v1/namespaces?resourceVersion=2263964": dial tcp <k8s-cluster-ip>:443: i/o
 timeout
Failed to watch *v1.Service: failed to list *v1.Service: Get "https://<k8s-cluster-
ip>:443/api/v1/services?resourceVersion=2263966": dial tcp <k8s-cluster-ip>:443: i/o
 timeout
Failed to watch *v1.EndpointSlice: failed to list *v1.EndpointSlice: Get "https://<k8s-
cluster-ip>:443/apis/discovery.k8s.io/v1/endpointslices?resourceVersion=2263896": dial
 tcp <k8s-cluster-ip>: i/o timeout
```

Credenziali dell'host tramite disconnessioni di rete

EKS Hybrid Nodes è integrato con le attivazioni ibride di AWS Systems Manager (SSM) e AWS IAM Roles Anywhere per le credenziali IAM temporanee utilizzate per autenticare il nodo con il piano di controllo EKS. Sia SSM che IAM Roles Anywhere aggiornano automaticamente le credenziali temporanee che gestiscono sugli host locali. Si consiglia di utilizzare un unico provider di credenziali tra i nodi ibridi del cluster: attivazioni ibride SSM o IAM Roles Anywhere, ma non entrambi.

Attivazioni ibride SSM

Le credenziali temporanee fornite da SSM sono valide per un'ora. Non è possibile modificare la durata di validità delle credenziali quando si utilizza SSM come fornitore di credenziali. Le credenziali temporanee vengono ruotate automaticamente da SSM prima della scadenza e la rotazione non influisce sullo stato dei nodi o delle applicazioni. Tuttavia, in caso di disconnessioni di rete tra l'agente SSM e l'endpoint regionale SSM, SSM non è in grado di aggiornare le credenziali e le credenziali potrebbero scadere.

SSM utilizza il backoff esponenziale per i nuovi tentativi di aggiornamento delle credenziali se non è in grado di connettersi agli endpoint regionali SSM. Nella versione dell'agente SSM `3.3.808.0` e successive (rilasciate ad agosto 2024), il backoff esponenziale è limitato a 30 minuti. A seconda

della durata della disconnessione di rete, SSM potrebbe impiegare fino a 30 minuti per aggiornare le credenziali e i nodi ibridi non si riconnetteranno al piano di controllo EKS finché le credenziali non verranno aggiornate. In questo scenario, puoi riavviare l'agente SSM per forzare l'aggiornamento delle credenziali. Come effetto collaterale dell'attuale comportamento di aggiornamento delle credenziali SSM, i nodi potrebbero riconnettersi in momenti diversi a seconda di quando l'agente SSM su ciascun nodo riesce ad aggiornare le proprie credenziali. Per questo motivo, è possibile che si verifichi il failover dei pod da nodi che non sono ancora stati ricollegati a nodi già ricollegati.

Scarica la versione dell'agente SSM. Puoi anche controllare la sezione Fleet Manager della console SSM:

```
# AL2023, RHEL
yum info amazon-ssm-agent
# Ubuntu
snap list amazon-ssm-agent
```

Riavvia l'agente SSM:

```
# AL2023, RHEL
systemctl restart amazon-ssm-agent
# Ubuntu
systemctl restart snap.amazon-ssm-agent.amazon-ssm-agent
```

Visualizza i log dell'agente SSM:

```
tail -f /var/log/amazon/ssm/amazon-ssm-agent.log
```

Messaggi di registro previsti durante le disconnessioni di rete:

```
INFO [CredentialRefresher] Credentials ready
INFO [CredentialRefresher] Next credential rotation will be in 29.995040663666668
 minutes
ERROR [CredentialRefresher] Retrieve credentials produced error: RequestError: send
 request failed
INFO [CredentialRefresher] Sleeping for 35s before retrying retrieve credentials
ERROR [CredentialRefresher] Retrieve credentials produced error: RequestError: send
 request failed
INFO [CredentialRefresher] Sleeping for 56s before retrying retrieve credentials
ERROR [CredentialRefresher] Retrieve credentials produced error: RequestError: send
 request failed
```

```
INFO [CredentialRefresher] Sleeping for 1m24s before retrying retrieve credentials
```

IAM Roles Anywhere

Le credenziali temporanee fornite da IAM Roles Anywhere sono valide per un'ora per impostazione predefinita. Puoi configurare la durata di validità delle credenziali con IAM Roles Anywhere tramite il `durationSeconds`campo del tuo profilo IAM Roles Anywhere. La durata massima di validità delle credenziali è di 12 ore. L'`MaxSessionDuration`impostazione sul tuo ruolo IAM Hybrid Nodes deve essere maggiore dell'`durationSeconds`impostazione sul tuo profilo IAM Roles Anywhere.

Quando si utilizza IAM Roles Anywhere come fornitore di credenziali per i nodi ibridi, la riconnessione al piano di controllo EKS dopo le disconnessioni di rete avviene in genere entro pochi secondi dal ripristino della rete, poiché il kubelet chiama `aws_signing_helper credential-process` per ottenere le credenziali su richiesta. Sebbene non sia direttamente correlato ai nodi ibridi o alle disconnessioni di rete, puoi configurare notifiche e avvisi per la scadenza dei certificati quando utilizzi IAM Roles Anywhere. Per ulteriori informazioni, consulta Personalizzare le impostazioni di notifica in IAM Roles Anywhere.

Best practice per l'esecuzione dei AI/ML carichi di lavoro

> **Tip**
>
> Esplora le best practice tramite i workshop Amazon EKS.

L'implementazione delle migliori pratiche per l'esecuzione AI/ML di carichi di lavoro su EKS può garantire che tali carichi di lavoro siano performanti, convenienti, resilienti e dotati di risorse adeguate. Le best practice sono suddivise nelle seguenti sezioni generali: elaborazione, rete, archiviazione, osservabilità e prestazioni.

Feedback

Questa guida viene pubblicata GitHub per raccogliere feedback e suggerimenti diretti dalla comunità più ampia EKS/Kubernetes . Se hai una best practice che ritieni debba essere inclusa nella guida, segnala un problema o invia un PR nell' GitHub archivio. La nostra intenzione è quella di aggiornare periodicamente la guida man mano che vengono aggiunte nuove funzionalità al servizio o quando si evolve una nuova best practice.

Calcolo e scalabilità automatica

> **Tip**
>
> Esplora le best practice tramite i workshop Amazon EKS.

Ottimizzazione delle risorse GPU e gestione dei costi

Pianifica i carichi di lavoro con i requisiti della GPU utilizzando etichette Known

Per i AI/ML carichi di lavoro sensibili alle diverse caratteristiche della GPU (ad esempio GPU, memoria GPU), consigliamo di specificare i requisiti della GPU utilizzando etichette di pianificazione note supportate dai tipi di nodi utilizzati con Karpenter e i gruppi di nodi gestiti. La mancata definizione di questi parametri può comportare la pianificazione dei pod su istanze con risorse GPU inadeguate, con conseguenti guasti o un peggioramento delle prestazioni. Ti consigliamo di

utilizzare NodeSelector o Node affinity per specificare su quale nodo deve essere eseguito un pod e di impostare le risorse di calcolo (CPU, memoria, ecc.) nella sezione delle risorse del pod. GPUs

Esempio

Ad esempio, utilizzando il selettore di nodi di nomi GPU quando si utilizza Karpenter:

```
apiVersion: v1
kind: Pod
metadata:
  name: gpu-pod-example
spec:
  containers:
  - name: ml-workload
    image: <image>
    resources:
      limits:
        nvidia.com/gpu: 1  # Request one NVIDIA GPU
  nodeSelector:
    karpenter.k8s.aws/instance-gpu-name: "l40s"  # Run on nodes with NVIDIA L40S GPUs
```

Usa Kubernetes Device Plugin per esporre GPUs

Per eseguire l'esposizione GPUs sui nodi, il driver GPU NVIDIA deve essere installato sul sistema operativo del nodo e il runtime del contenitore deve essere configurato per consentire allo scheduler Kubernetes di assegnare i pod ai nodi se disponibili. GPUs Il processo di configurazione per il NVIDIA Kubernetes Device Plugin dipende dall'AMI EKS Accelerated che stai utilizzando:

- AMI Bottlerocket Accelerated: questa AMI include il driver GPU NVIDIA e il NVIDIA Kubernetes Device Plugin è preinstallato e pronto all'uso, abilitando il supporto GPU immediatamente. Non è richiesta alcuna configurazione aggiuntiva per l'esposizione allo scheduler Kubernetes. GPUs
- AL2023 AMI accelerata: questa AMI include il driver GPU NVIDIA ma il plugin per dispositivi NVIDIA Kubernetes non è preinstallato. È necessario installare e configurare il plug-in del dispositivo separatamente, in genere tramite un. DaemonSet Nota che se usi eksctl per creare il tuo cluster e specifichi un tipo di istanza GPU (ad esempio`g5.xlarge`) nel tuo ClusterConfig, `eksctl` selezionerà automaticamente l'AMI appropriata e installerà il NVIDIA Kubernetes Device Plugin. Per saperne di più, consulta il supporto per le GPU nella documentazione di eksctl.

Se invece decidi di utilizzare l'operatore EKS Accelerated AMIs e NVIDIA GPU per gestire componenti come il plug-in per dispositivi NVIDIA Kubernetes, prendi nota di disabilitare la gestione

del driver GPU NVIDIA e del toolkit NVIDIA Container come indicato nei driver GPU NVIDIA preinstallati e nella documentazione NVIDIA Container Toolkit NVIDIA.

Per verificare che il GPUs NVIDIA Device Plugin sia attivo e sia esposto correttamente, esegui:

```
kubectl describe node | grep nvidia.com/gpu
```

Questo comando verifica se la `nvidia.com/gpu` risorsa rientra nella capacità del nodo e nelle risorse allocabili. Ad esempio, dovrebbe essere visualizzato un nodo con una GPU. `nvidia.com/gpu: 1` Per ulteriori informazioni, consulta la Kubernetes GPU Scheduling Guide.

Utilizza molti tipi di istanze EC2 diversi

L'utilizzo del maggior numero possibile di tipi di istanze EC2 diversi è una best practice importante per la scalabilità su Amazon EKS, come indicato nella sezione. the section called “Piano dei dati” Questa raccomandazione si applica anche alle istanze con hardware accelerato (ad es.). GPUs Se crei un cluster che utilizza un solo tipo di istanza e provi a scalare il numero di nodi oltre la capacità della regione, potresti ricevere un errore di capacità insufficiente (ICE), che indica che non sono disponibili istanze. È importante comprendere le caratteristiche uniche dei AI/ML carichi di lavoro prima di diversificare arbitrariamente. Esamina i tipi di istanze disponibili utilizzando lo strumento EC2 Instance Type Explorer per generare un elenco di tipi di istanze che soddisfano i tuoi requisiti di elaborazione specifici ed evita di limitare arbitrariamente il tipo di istanze che possono essere utilizzate nel tuo cluster.

Le istanze di elaborazione accelerata sono offerte in diversi modelli di acquisto per soddisfare carichi di lavoro a breve, medio termine e stazionari. Per carichi di lavoro a breve termine, flessibili e tolleranti ai guasti, in cui vorresti evitare di effettuare una prenotazione, dai un'occhiata alle istanze Spot. I blocchi di capacità, le istanze On-Demand e i piani di risparmio consentono di fornire istanze di elaborazione accelerate per carichi di lavoro a medio e lungo termine. Per aumentare le possibilità di accedere correttamente alla capacità richiesta nell'opzione di acquisto preferita, si consiglia di utilizzare un elenco diversificato di tipi di istanze e zone di disponibilità. In alternativa, se ICEs riscontri un modello di acquisto specifico, riprova a utilizzare un modello diverso.

Esempio L'esempio seguente mostra come consentire a un Karpenter di effettuare il provisioning NodePool di istanze G e P superiori alla terza generazione (ad esempio, p3). Per ulteriori informazioni, consulta la sezione Scalabilità.

```
- key: karpenter.k8s.aws/instance-category
```

```
  operator: In
  values: ["g", "p"] # Diversifies across G-series and P-series
- key: karpenter.k8s.aws/instance-generation
  operator: Gt
  values: ["3"] # Selects instance generations greater than 3
```

Per i dettagli sull'uso delle istanze Spot per le GPU, consulta «Prendi in considerazione l'utilizzo delle istanze Spot di Amazon EC2 GPUs per Karpenter» di seguito.

Prendi in considerazione l'utilizzo di istanze Spot Amazon EC2 per Karpenter GPUs

Le istanze Spot di Amazon EC2 consentono di sfruttare la capacità EC2 inutilizzata nel cloud AWS e sono disponibili con uno sconto fino al 90% rispetto ai prezzi On-Demand. Le istanze Spot di Amazon EC2 possono essere interrotte con un preavviso di due minuti quando EC2 necessita del ripristino della capacità. Per ulteriori informazioni, consulta le istanze Spot nella Guida per l'utente di Amazon EC2. Amazon EC2 Spot può essere un'ottima scelta per carichi di lavoro con tolleranza ai guasti, stateless e flessibili (tempo e tipo di istanza). Per ulteriori informazioni su quando utilizzare le istanze Spot, consulta le best practice per le istanze Spot EC2. Puoi anche utilizzare le istanze Spot per AI/ML carichi di lavoro se sono compatibili con Spot.

Casi d'uso

I carichi di lavoro compatibili con Spot possono essere big data, carichi di lavoro containerizzati, CI/CD, server web stateless, High Performance Computing (HPC) e carichi di lavoro di rendering. Le istanze Spot non sono adatte per carichi di lavoro rigidi, stateful, intolleranti ai guasti o strettamente accoppiati tra i nodi di istanza (ad esempio, carichi di lavoro con processi paralleli che dipendono fortemente l'uno dall'altro per il calcolo, che richiedono una comunicazione costante tra i nodi, come applicazioni di calcolo ad alte prestazioni basate su MPI come la fluidodinamica computazionale o i database distribuiti con interdipendenze complesse). Ecco i casi d'uso specifici che consigliamo (in nessun ordine particolare):

- Inferenza online in tempo reale: utilizza le istanze Spot per una scalabilità ottimizzata in termini di costi per i carichi di lavoro di inferenza in tempo reale, a condizione che i carichi di lavoro siano adatti allo spot. In altre parole, il tempo di inferenza è inferiore a due minuti, l'applicazione è tollerante ai guasti alle interruzioni e può essere eseguita su diversi tipi di istanze. Garantisci l'elevata disponibilità attraverso la diversità delle istanze (ad esempio, tra più tipi di istanze e zone di disponibilità) o le prenotazioni, implementando al contempo la tolleranza agli errori a livello di applicazione per gestire potenziali interruzioni Spot.

- Ottimizzazione iperparametrica: utilizza le istanze Spot per eseguire lavori di ottimizzazione esplorativa in modo opportuno, poiché le interruzioni possono essere tollerate senza perdite significative, soprattutto per esperimenti di breve durata.
- Aumento dei dati: utilizza le istanze Spot per eseguire attività di preelaborazione e aumento dei dati che possono essere riavviate dai checkpoint se interrotte, il che le rende ideali per la disponibilità variabile di Spot.
- Modelli di ottimizzazione: utilizza le istanze Spot per la messa a punto con solidi meccanismi di checkpoint per riprendere dall'ultimo stato salvato, riducendo al minimo l'impatto delle interruzioni delle istanze.
- Inferenza in batch: utilizza le istanze Spot per elaborare grandi batch di richieste di inferenza offline in non-real-time modo da mettere in pausa e riprendere i lavori, offrendo il miglior allineamento con i risparmi sui costi di Spot e gestendo potenziali interruzioni attraverso nuovi tentativi o diversificazione.
- Sottoinsiemi di formazione opportunistici: utilizza le istanze Spot per carichi di lavoro di formazione marginali o sperimentali (ad esempio, modelli più piccoli con meno di 10 milioni di parametri), dove le interruzioni sono accettabili e possono essere applicate ottimizzazioni dell'efficienza come la diversificazione tra tipi di istanze o regioni, sebbene non sia consigliabile per la formazione su scala di produzione a causa di potenziali interruzioni.

Considerazioni

Per utilizzare le istanze Spot per carichi di lavoro accelerati su Amazon EKS, ci sono una serie di considerazioni chiave (in nessun ordine particolare):

- Usa Karpenter per gestire le istanze Spot con il consolidamento avanzato abilitato. Specificando karpenter.sh/capacity-type come «spot» in Karpenter, Karpenter eseguirà il provisioning delle istanze Spot per impostazione predefinita senza alcuna configurazione aggiuntiva. NodePool Tuttavia, per abilitare il Spot-to-Spot consolidamento avanzato, che sostituisce i nodi Spot sottoutilizzati con alternative Spot più economiche, è necessario abilitare il feature gate impostando --feature-gates =true negli argomenti del controller Karpenter o tramite la variabile di ambiente FEATURE_GATES. SpotToSpotConsolidation SpotToSpotConsolidation Karpenter utilizza la strategia di allocazione per effettuare il provisioning delle istanze EC2. price-capacity-optimized In base NodePool ai requisiti e ai vincoli dei pod, Karpenter racchiude i pod non programmabili e invia un set diversificato di tipi di istanze all'API Amazon EC2 Fleet. Puoi utilizzare lo strumento EC2 Instance Type Explorer per generare un elenco di tipi di istanze che soddisfano i tuoi requisiti di elaborazione specifici.

- Assicurati che i carichi di lavoro siano stateless, flessibili e tolleranti agli errori. I carichi di lavoro devono essere stateless, tolleranti ai guasti e flessibili in termini di dimensioni. instance/GPU Ciò consente una ripresa senza interruzioni dopo le interruzioni di Spot e la flessibilità delle istanze consente di rimanere potenzialmente su Spot più a lungo. Abilita la gestione delle interruzioni Spot in Karpenter configurando il valore settings.InterruptionQueue Helm con il nome della coda AWS SQS per catturare gli eventi di interruzione Spot. Ad esempio, durante l'installazione tramite Helm, usa --set «settings.interruptionQueue=$ {CLUSTER_NAME}». Per vedere un esempio, consulta la guida Getting Started with Karpenter. Quando Karpenter rileva un evento di interruzione Spot, cordone, contamina, drena e chiude automaticamente i nodi prima dell'evento di interruzione per massimizzare il periodo di tolleranza di terminazione dei pod. Allo stesso tempo, Karpenter avvierà immediatamente un nuovo nodo in modo che possa essere pronto il prima possibile.
- Evitate di limitare eccessivamente la selezione del tipo di istanza. Dovreste evitare il più possibile di vincolare i tipi di istanza. Non limitando i tipi di istanze, vi è una maggiore possibilità di acquisire capacità Spot su larga scala con una frequenza inferiore di interruzioni delle istanze Spot a un costo inferiore. Ad esempio, evita di limitarti a tipi specifici (ad esempio g5.xlarge). Prendi in considerazione la possibilità di specificare un insieme diversificato di categorie e generazioni di istanze utilizzando chiavi come karpenter.k8s. aws/instance-category and karpenter.k8s.aws/instance-generation. Karpenter enables easier diversification of on-demand and Spot instance capacity across multiple instance types and Availability Zones (AZs). Moreover, if your AI/MLil carico di lavoro richiede un numero specifico o limitato di acceleratori ma è flessibile tra le regioni, puoi utilizzare Spot Placement Score per identificare dinamicamente la regione ottimale in cui distribuire il carico di lavoro prima del lancio.
- Amplia NodePool i requisiti per includere un numero maggiore di famiglie di istanze EC2 simili. Ogni pool di istanze Spot è costituito da una capacità di istanze EC2 inutilizzata per un tipo di istanza specifico in una zona di disponibilità (AZ) specifica. Quando Karpenter tenta di fornire un nuovo nodo, seleziona un tipo di istanza che soddisfa i requisiti del nodo. NodePool Se nessun tipo di istanza compatibile dispone della capacità Spot in qualsiasi zona, il provisioning fallisce. Per evitare questo problema, consenti istanze della serie g più ampie (generazione 4 o superiore) di NVIDIA di diverse dimensioni e zone di disponibilità (AZs), tenendo conto delle esigenze hardware come la memoria GPU o il Ray Tracing. Poiché le istanze possono essere di diversi tipi, devi assicurarti che il tuo carico di lavoro sia in grado di funzionare su ogni tipo e che le prestazioni ottenute soddisfino le tue esigenze.
- Sfrutta tutte le zone di disponibilità in una regione. La capacità disponibile varia in base alla zona di disponibilità (AZ), un tipo di istanza specifico potrebbe non essere disponibile in una zona ma abbondante in un'altra. Ogni combinazione unica di un tipo di istanza e una zona di disponibilità costituisce un pool di capacità Spot separato. Richiedendo la capacità AZs in tutte le

aree geografiche che rientrano nei NodePool requisiti richiesti da Karpenter, è possibile cercare in modo efficace più pool contemporaneamente. Ciò massimizza il numero di pool di capacità Spot e quindi aumenta la probabilità di acquisire capacità Spot. Per ottenere ciò, nella NodePool configurazione, ometti completamente la chiave topology.kubernetes.io/zone per consentire a Karpenter di selezionare tra tutte le opzioni disponibili AZs nella regione, oppure AZs elenca esplicitamente utilizzando l'operatore: In e fornisci i valori (ad esempio, us-west-2a).

- Prendi in considerazione l'utilizzo di Spot Placement Score (SPS) per avere visibilità sulla probabilità di accedere con successo alla capacità richiesta utilizzando le istanze Spot. Spot Placement Score (SPS) è uno strumento che fornisce un punteggio per aiutarti a valutare la probabilità di successo di una richiesta Spot. Quando usi SPS, devi prima specificare i requisiti di elaborazione per le tue istanze Spot, quindi Amazon EC2 restituisce le prime 10 regioni o zone di disponibilità (AZs) in cui è probabile che la tua richiesta Spot abbia successo. Le regioni e le zone di disponibilità vengono valutate su una scala da 1 a 10. Un punteggio pari a 10 indica che è molto probabile, ma non è garantito che la richiesta Spot abbia successo. Un punteggio di 1 indica che la tua richiesta Spot ha bassissime probabilità di successo. Lo stesso punteggio potrebbe essere restituito per diverse regioni o zone di disponibilità. Per ulteriori informazioni, consulta la Guida alla creazione di una dashboard Spot Placement Score Tracker su AWS. Poiché la capacità di Spot oscilla continuamente, SPS ti aiuterà a identificare la combinazione di tipi di istanze e regioni più adatta ai tuoi vincoli di carico di lavoro (ad esempio flessibilità, prestazioni, dimensioni, ecc.). AZs Se il AI/ML carico di lavoro richiede un numero specifico o limitato di acceleratori ma è flessibile tra le regioni, puoi utilizzare il punteggio di posizionamento di Spot per identificare dinamicamente la regione ottimale in cui distribuire il carico di lavoro prima del lancio. Per aiutarti a scoprire automaticamente la probabilità di acquisire capacità Spot, forniamo una guida per la creazione di una dashboard di monitoraggio SPS. Questa soluzione monitora i punteggi SPS nel tempo utilizzando una configurazione YAML per configurazioni diversificate (ad esempio, requisiti di istanza inclusi GPUs), archivia le metriche e fornisce dashboard per confrontare le configurazioni. CloudWatch Definisci dashboard per carico di lavoro per valutare le esigenze di vCPU, memoria e GPU, garantendo configurazioni ottimali per i cluster EKS, inclusa la possibilità di utilizzare altre regioni AWS. Per ulteriori informazioni, consulta Come funziona il punteggio di posizionamento di Spot.
- Gestisci con garbo le interruzioni e i test di Spot. Per un pod con un periodo di terminazione superiore a due minuti, il vecchio nodo verrà interrotto prima della riprogrammazione dei pod, il che potrebbe influire sulla disponibilità del carico di lavoro. Prendi in considerazione l'avviso di interruzione Spot di due minuti durante la progettazione delle tue applicazioni, implementa il checkpoint nelle applicazioni a lunga esecuzione (ad esempio, salvando i progressi su uno storage persistente come Amazon S3) per riprenderlo dopo le interruzioni, estendi i terminationGracePeriod

secondi (l'impostazione predefinita è 30 secondi) nelle specifiche del Pod per consentire più tempo per uno spegnimento regolare e gestisci le interruzioni utilizzando i and/or segnali SIGTERM del ciclo di vita PreStop all'interno dell'applicazione per attività di spegnimento regolari come pulizia, salvataggio dello stato e chiusura della connessione. Per i carichi di lavoro in tempo reale, in cui il tempo di scalabilità è importante e i carichi di lavoro impiegano più di due minuti prima che l'applicazione sia pronta a servire il traffico, prendi in considerazione l'ottimizzazione dei tempi di avvio dei container e di caricamento dei modelli ML mediante revisioni the section called "Archiviazione" e the section called "Performance" best practice. Per testare un nodo sostitutivo, usa AWS Fault Injection Service (FIS) per simulare le interruzioni Spot.

Oltre a queste best practice principali di Spot, tieni conto di questi fattori nella gestione dei carichi di lavoro delle GPU su Amazon EKS. A differenza dei carichi di lavoro basati su CPU, i carichi di lavoro su GPU sono particolarmente sensibili ai dettagli hardware, come le funzionalità della GPU e la memoria GPU disponibile. I carichi di lavoro GPU potrebbero essere limitati dai tipi di istanze che possono utilizzare, con un minor numero di opzioni disponibili rispetto a. CPUs Come primo passo, valuta se il tuo carico di lavoro è flessibile a seconda delle istanze. Se non sai quanti tipi di istanze possono essere utilizzati dal tuo carico di lavoro, testali singolarmente per garantire compatibilità e funzionalità. Identifica quanto puoi essere flessibile per diversificare il più possibile, confermando al contempo che la diversificazione mantiene il carico di lavoro funzionante e comprendi eventuali impatti sulle prestazioni (ad esempio, sulla produttività o sui tempi di completamento). Nell'ambito della diversificazione dei carichi di lavoro, considera quanto segue:

- Esamina la compatibilità tra CUDA e framework. I carichi di lavoro della GPU potrebbero essere ottimizzati per hardware o tipi di GPU specifici (ad esempio, V100 in p3 rispetto a A100 in p4) o scritti per versioni CUDA specifiche per librerie come, ad esempio TensorFlow, assicurati di verificare la compatibilità per i tuoi carichi di lavoro. Questa compatibilità è fondamentale per prevenire errori di runtime, arresti anomali, guasti nell'accelerazione della GPU (ad esempio, versioni CUDA non corrispondenti con framework simili PyTorch o che TensorFlow possono impedirne l'esecuzione) o la capacità di sfruttare funzionalità hardware come FP16/INT8 precision.
- Memoria GPU. Assicurati di valutare i requisiti di memoria dei tuoi modelli e di profilare l'utilizzo della memoria del modello durante l'esecuzione utilizzando strumenti come DCGM Exporter e imposta la memoria GPU minima richiesta per il tipo di istanza in etichette note come karpenter.k8s.aws/. instance-gpu-memory La VRAM della GPU varia in base al tipo di istanza (ad esempio, NVIDIA T4 ha 16 GB, A10G ha 24 GB, V100 ha 16-32 GB) e i modelli ML (ad esempio, modelli a linguaggio di grandi dimensioni) possono superare la memoria disponibile, causando errori out-of-memory (OOM) o arresti anomali. Per le istanze Spot in EKS, ciò può limitare la

diversificazione. Ad esempio, non è possibile includere tipi di VRAM inferiori se il modello non è adatto, il che potrebbe limitare l'accesso ai pool di capacità e aumentare il rischio di interruzione. Tieni presente che per l'inferenza a singola GPU e a nodo singolo (ad esempio, più pod pianificati sullo stesso nodo per utilizzare le relative risorse GPU), ciò potrebbe limitare la diversificazione, poiché nella configurazione Spot puoi includere solo tipi di istanze con VRAM sufficiente.

- Precisione e prestazioni a virgola mobile. Non tutte le architetture GPU Nvidia hanno la stessa precisione a virgola mobile (ad esempio,/). FP16 INT8 Valuta le prestazioni dei tipi di core (CUDA/ Tensor/RT) e la precisione a virgola mobile necessarie per i tuoi carichi di lavoro. L'esecuzione su una GPU meno costosa e meno performante non significa che sia migliore, quindi valuta la possibilità di valutare le prestazioni in termini di lavoro completato entro un periodo di tempo specifico per comprendere l'impatto della diversificazione.

Scenario: diversificazione per carichi di lavoro di inferenza in tempo reale

Per un carico di lavoro di inferenza online in tempo reale sulle istanze Spot, puoi configurare un Karpenter in modo che si diversifichi tra famiglie e generazioni di istanze NodePool GPU compatibili. Questo approccio garantisce un'elevata disponibilità attingendo a più pool Spot, mantenendo al contempo le prestazioni attraverso vincoli sulle funzionalità, sulla memoria e sull'architettura della GPU. Supporta l'utilizzo di alternative quando la capacità dell'istanza è limitata, riducendo al minimo le interruzioni e ottimizzando la latenza di inferenza. Questo esempio NodePool indica che utilizza istanze delle serie g e p maggiori di 3, con più di 20 GB di memoria GPU.

Esempio

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: gpu-inference-spot
spec:
  template:
    metadata:
      labels:
        role: gpu-spot-worker
    spec:
      requirements:
        - key: karpenter.sh/capacity-type
          operator: In
          values: ["spot"] # Use Spot Instances
        - key: karpenter.k8s.aws/instance-category
          operator: In
```

```
          values: ["g", "p"] # Diversifies across G-series and P-series
        - key: karpenter.k8s.aws/instance-generation
          operator: Gt
          values: ["3"] # Selects instance generations greater than 3
        - key: kubernetes.io/arch
          operator: In
          values: ["amd64"] # Specifies AMD64 architecture, compatible with NVIDIA GPUs
        - key: karpenter.k8s.aws/instance-gpu-memory
          operator: Gt
          values: ["20480"] # Ensures more than 20GB (20480 MiB) total GPU memory
      taints:
        - key: nvidia.com/gpu
          effect: NoSchedule
      nodeClassRef:
        name: gpu-inference-ec2
        group: karpenter.k8s.aws
        kind: EC2NodeClass
      expireAfter: 720h
  limits:
    cpu: 100
    memory: 100Gi
  disruption:
    consolidationPolicy: WhenEmptyOrUnderutilized
    consolidateAfter: 5m # Enables consolidation of underutilized nodes after 5 minutes
```

Implementa il checkpointing per lavori di formazione di lunga durata

Il checkpointing è una tecnica di tolleranza ai guasti che prevede il salvataggio periodico dello stato di un processo, consentendone la ripresa dall'ultimo punto salvato in caso di interruzioni. Nell'apprendimento automatico, è comunemente associato alla formazione, in cui i lavori di lunga durata possono far risparmiare peso al modello e gli stati dell'ottimizzatore per riprendere l'addestramento dopo guasti, come problemi hardware o interruzioni delle istanze Spot.

I checkpoint vengono utilizzati per salvare lo stato dei modelli di machine learning (ML) durante l'addestramento. I checkpoint sono istantanee del modello e possono essere configurati tramite le funzioni di callback dei framework ML. È possibile utilizzare i checkpoint salvati per riavviare un job di addestramento dall'ultimo checkpoint salvato. Utilizzando i checkpoint, si salvano le istantanee del modello durante l'addestramento a causa di un'interruzione imprevista del processo o dell'istanza di formazione. Ciò consente di riprendere l'addestramento del modello in futuro da un checkpoint. Oltre a implementare un sistema di resilienza dei nodi, consigliamo di implementare il checkpoint

per mitigare l'impatto delle interruzioni, incluse quelle causate da guasti hardware o interruzioni delle istanze Spot di Amazon EC2.

Senza checkpoint, le interruzioni possono comportare uno spreco di tempo di elaborazione e una perdita di progressi, il che è costoso per i lavori di formazione di lunga durata. Il checkpointing consente ai lavori di salvare periodicamente il loro stato (ad esempio, pesi del modello e stati dell'ottimizzatore) e di riprendere dall'ultimo checkpoint (ultimo batch elaborato) dopo un'interruzione. Per implementare il checkpointing, progetta la tua applicazione in modo che elabori i dati in grandi batch e salvi i risultati intermedi su uno storage persistente, ad esempio un bucket Amazon S3 tramite il driver Mountpoint for Amazon S3 CSI mentre il processo di formazione procede.

Casi d'uso

Il checkpointing è particolarmente utile in scenari specifici per bilanciare la tolleranza agli errori con il sovraccarico delle prestazioni. Prendi in considerazione l'utilizzo del checkpoint nei seguenti casi:

- La durata del lavoro supera alcune ore: per lavori di formazione di lunga durata (ad esempio, più di 1-2 ore per modelli di piccole dimensioni o days/weeks per modelli di base di grandi dimensioni con miliardi di parametri), in cui la perdita di progressi a causa delle interruzioni è costosa. I lavori più brevi potrebbero non giustificare le spese generali. I/O
- Per istanze Spot o guasti hardware: in ambienti soggetti a interruzioni, come EC2 Spot (preavviso di 2 minuti) o guasti hardware (ad esempio errori di memoria della GPU), il checkpoint consente una ripresa rapida, rendendo Spot fattibile per ridurre i costi in carichi di lavoro con tolleranza ai guasti.
- Formazione distribuita su larga scala: per le configurazioni con hundreds/thousands acceleratori (ad esempio, >100 GPUs), in cui il tempo medio tra i guasti diminuisce linearmente con la scala. Utilizzalo per il model/data parallelismo per gestire l'accesso simultaneo ai checkpoint ed evitare riavvii completi.
- Modelli su larga scala con un elevato fabbisogno di risorse: nella formazione LLM su scala petabyte, in cui i guasti sono inevitabili a causa delle dimensioni del cluster, gli approcci su più livelli (rapido, locale ogni 5-30 minuti per i transitori, duraturo ogni ora per i guasti più gravi) ottimizzano i tempi di ripristino rispetto all'efficienza.

Utilizza ML Capacity Blocks per garantire la capacità delle istanze P e Trainium

I Capacity Blocks for ML ti consentono di prenotare istanze GPU molto richieste, in particolare istanze P (ad esempio, p6-b200, p5, p5e, p5en, p4d, p4de) e istanze Trainium (ad es. trn1, trn2), da avviare

quasi immediatamente o in date future per supportare i carichi di lavoro di machine learning (ML) di breve durata. Queste prenotazioni sono ideali per garantire la capacità di svolgere attività ad alta intensità di calcolo come la formazione dei modelli e la messa a punto. I prezzi di EC2 Capacity Blocks sono costituiti da una commissione di prenotazione e da una commissione per il sistema operativo. Per ulteriori informazioni sui prezzi, consulta i prezzi di EC2 Capacity Blocks for ML.

GPUs Per riservare i AI/ML carichi di lavoro su Amazon EKS per una garanzia di capacità prevedibile, consigliamo di utilizzare i blocchi di capacità ML per le prenotazioni di capacità a breve termine o le prenotazioni di capacità su richiesta (ODCRs) per la garanzia della capacità generica.

- Gli ODCR consentono di riservare la capacità delle istanze EC2 (ad esempio, istanze GPU come g5 o p5) in una zona di disponibilità specifica per un determinato periodo, garantendo la disponibilità anche in caso di forte domanda. ODCRs non avete alcun impegno a lungo termine, ma pagate la tariffa On-Demand per la capacità riservata, utilizzata o inattiva. In EKS, ODCRs sono supportati da tipi di nodi come Karpenter e gruppi di nodi gestiti. Per stabilire le priorità ODCRs in Karpenter, configura l'utilizzo del campo. NodeClass `capacityReservationSelectorTerms` Vedi la documentazione di Karpenter. NodePools
- I Capacity Blocks sono un meccanismo di prenotazione specializzato per istanze GPU (ad esempio p5, p4d) o Trainium (trn1, trn2), progettato per carichi di lavoro ML a breve termine come la formazione di modelli, la messa a punto o la sperimentazione. Riservi la capacità per un periodo definito (in genere da 24 ore a 182 giorni) a partire da una data futura, pagando solo per il tempo prenotato. Sono prepagati, richiedono una pianificazione preliminare per le esigenze di capacità e non supportano la scalabilità automatica, ma sono collocati in EC2 per una rete a bassa latenza. UltraClusters Fanno pagare solo per il periodo prenotato. Per ulteriori informazioni, consulta Trova e acquista blocchi di capacità oppure inizia a configurare gruppi di nodi gestiti con Capacity Blocks seguendo le istruzioni contenute in Creare un gruppo di nodi gestiti con blocchi di capacità per ML.

Riserva la capacità tramite la Console di gestione AWS e configura i tuoi nodi per utilizzare blocchi di capacità ML. Pianifica le prenotazioni in base alla pianificazione dei carichi di lavoro ed esegui i test in un cluster di staging. Per ulteriori informazioni, consulta la documentazione di Capacity Blocks.

Prendi in considerazione le prenotazioni di capacità on-demand, Amazon EC2 Spot o On-Demand Capacity Reservations (ODCR) per le istanze G Amazon EC2

Per le istanze G Amazon EC2, considera le diverse opzioni di acquisto tra On-Demand, Amazon EC2 Spot e On-Demand Capacity Reservations. ODCRsconsentono di riservare la capacità delle istanze EC2 in una zona di disponibilità specifica per un determinato periodo, garantendo la disponibilità

anche in caso di forte domanda. A differenza dei ML Capacity Block, disponibili solo per le istanze P e Trainium, gli ODCR possono essere utilizzati per una gamma più ampia di tipi di istanze, comprese le istanze G, rendendoli adatti a carichi di lavoro che richiedono funzionalità GPU diverse, come l'inferenza o la grafica. Quando si utilizzano le istanze Spot di Amazon EC2, la possibilità di diversificare tra diversi tipi di istanze, dimensioni e zone di disponibilità è fondamentale per poter rimanere su Spot più a lungo.

ODCRs non avete alcun impegno a lungo termine, ma pagate la tariffa On-Demand per la capacità riservata, utilizzata o inattiva. ODCRs possono essere creati per un uso immediato o programmati per date future, garantendo flessibilità nella pianificazione della capacità. In Amazon EKS, ODCRs sono supportati da tipi di nodi come Karpenter e gruppi di nodi gestiti. Per stabilire le priorità ODCRs in Karpenter, configura l'utilizzo del campo. NodeClass `capacityReservationSelectorTerms` Vedi la documentazione di Karpenter. NodePools Per ulteriori informazioni sulla creazione ODCRs, inclusi i comandi CLI, consulta la Guida introduttiva alla prenotazione della capacità su richiesta.

Prendi in considerazione altri tipi e dimensioni di istanze accelerate

La selezione dell'istanza accelerata e della dimensione appropriate è essenziale per ottimizzare prestazioni e costi nei carichi di lavoro ML su Amazon EKS. Ad esempio, diverse famiglie di istanze GPU hanno prestazioni e funzionalità diverse, come la memoria GPU. Per aiutarti a scegliere l'opzione con il miglior rapporto prezzo/prestazioni, consulta le istanze GPU disponibili nella pagina Tipi di istanze EC2 in Accelerated Computing. Valuta diversi tipi e dimensioni di istanze per trovare la soluzione migliore per i tuoi requisiti di carico di lavoro specifici. Prendi in considerazione fattori come il numero GPUs, la memoria e le prestazioni della rete. Selezionando attentamente il tipo e la dimensione dell'istanza GPU corretti, è possibile ottenere un migliore utilizzo delle risorse e un'efficienza dei costi nei cluster EKS.

Se si utilizza un'istanza GPU in un nodo EKS, per impostazione predefinita il `nvidia-device-plugin-daemonset` pod verrà inserito nel `kube-system` namespace. Per capire rapidamente se state utilizzando appieno le GPU della vostra istanza, potete usare nvidia-smi come mostrato di seguito:

```
kubectl exec nvidia-device-plugin-daemonset-xxxxx \
  -n kube-system -- nvidia-smi \
  --query-
gpu=index,power.draw,power.limit,temperature.gpu,utilization.gpu,utilization.memory,memory.free
 \
  --format=csv -l 5
```

- Se `utilization.memory` è vicino al 100%, è probabile che i tuoi codici siano vincolati alla memoria. Ciò significa che la GPU (memoria) è completamente utilizzata, ma potrebbe suggerire che sia necessario studiare un'ulteriore ottimizzazione delle prestazioni.
- Se `utilization.gpu` è vicino al 100%, ciò non significa necessariamente che la GPU sia completamente utilizzata. Una metrica migliore da considerare è il rapporto tra a. `power.draw` `power.limit` Se questo rapporto è pari o superiore al 100%, i tuoi codici stanno utilizzando appieno la capacità di calcolo della GPU.
- Il `-l 5` flag indica di emettere le metriche ogni 5 secondi. Nel caso di un singolo tipo di istanza GPU, il flag di interrogazione dell'indice non è necessario.

Per ulteriori informazioni, consulta le istanze GPU nella documentazione di AWS.

Ottimizza l'allocazione delle risorse GPU con Time-Slicing, MIG e l'allocazione frazionata della GPU

I limiti statici delle risorse in Kubernetes (ad esempio, CPU, memoria, numero di GPU) possono portare a un approvvigionamento eccessivo o al sottoutilizzo, in particolare per carichi di lavoro dinamici come l'inferenza. AI/ML La scelta della GPU giusta è importante. Per carichi di lavoro a basso volume o con picchi di traffico, il time-slicing consente a più carichi di lavoro di condividere una singola GPU condividendone le risorse di elaborazione, migliorando potenzialmente l'efficienza e riducendo gli sprechi. La condivisione della GPU può essere ottenuta tramite diverse opzioni:

- Sfrutta i selettori dei nodi e l'affinità dei nodi per influenzare la pianificazione: assicurati che i nodi forniti e i pod siano pianificati in modo appropriato GPUs per il carico di lavoro (ad es. `karpenter.k8s.aws/instance-gpu-name: "a100"`
- Time-Slicing: pianifica i carichi di lavoro per condividere le risorse di calcolo di una GPU nel tempo, permettendo l'esecuzione simultanea senza partizionamento fisico. È ideale per carichi di lavoro con esigenze di elaborazione variabili, ma può mancare l'isolamento della memoria.
- GPU multiistanza (MIG): MIG consente di partizionare una singola GPU NVIDIA in più istanze isolate ed è supportata con NVIDIA Ampere (ad esempio, GPU A100), NVIDIA Hopper (ad esempio, GPU H100) e NVIDIA Blackwell (ad esempio Blackwell). GPUs GPUs Ogni istanza MIG riceve risorse di calcolo e memoria dedicate, che consentono la condivisione delle risorse in ambienti multi-tenant o carichi di lavoro che richiedono garanzie di risorse, il che consente di ottimizzare l'utilizzo delle risorse GPU, inclusi scenari come la gestione di più modelli con batch di diverse dimensioni tramite time-slicing.

- Allocazione frazionata della GPU: utilizza la pianificazione basata su software per allocare parti del calcolo o della memoria di una GPU ai carichi di lavoro, offrendo flessibilità per carichi di lavoro dinamici. NVIDIA KAI Scheduler, parte della piattaforma Run:AI, consente tutto ciò consentendo ai pod di richiedere risorse GPU frazionarie.

Per abilitare queste funzionalità in EKS, puoi implementare il NVIDIA Device Plugin, che utilizza come risorse programmabili e supporta time-slicing e MIG. GPUs Per saperne di più, consulta Time-Slicing GPUs in Kubernetes e condivisione della GPU su Amazon EKS con NVIDIA time-slicing e istanze EC2 accelerate.

Esempio

Ad esempio, per abilitare il time-slicing con il plugin per dispositivi NVIDIA:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: nvidia-device-plugin-config
  namespace: kube-system
data:
  config.yaml: |
    version: v1
    sharing:
      timeSlicing:
        resources:
        - name: nvidia.com/gpu
          replicas: 4  # Allow 4 pods to share each GPU
```

Esempio

Ad esempio, per utilizzare KAI Scheduler per l'allocazione frazionata della GPU, implementalo insieme a NVIDIA GPU Operator e specifica le risorse GPU frazionarie nelle specifiche del pod:

```
apiVersion: v1
kind: Pod
metadata:
  name: fractional-gpu-pod-example
  annotations:
    gpu-fraction: "0.5"  # Annotation for 50% GPU
  labels:
    runai/queue: "default"  # Required queue assignment
```

```
spec:
  containers:
  - name: ml-workload
    image: nvcr.io/nvidia/pytorch:25.04-py3
    resources:
      limits:
        nvidia.com/gpu: 1
  nodeSelector:
    nvidia.com/gpu: "true"
  schedulerName: kai-scheduler
```

Resilienza dei nodi e gestione dei lavori di formazione

Implementa Node Health Checks con il ripristino automatico

Per i lavori di formazione distribuiti su Amazon EKS che richiedono comunicazioni frequenti tra nodi, come l'addestramento su modelli multi-GPU su più nodi, problemi hardware come guasti della GPU o dell'EFA possono causare interruzioni dei lavori di formazione. Queste interruzioni possono portare alla perdita di progressi nella formazione e all'aumento dei costi, in particolare per carichi di lavoro di lunga durata che si basano su hardware stabile. AI/ML

Per aumentare la resilienza contro i guasti hardware, come i guasti della GPU nei cluster EKS che eseguono carichi di lavoro GPU, consigliamo di utilizzare EKS Node Monitoring Agent con Auto Repair o Amazon. SageMaker HyperPod Mentre EKS Node Monitoring Agent with Auto Repair offre funzionalità come il monitoraggio dello stato dei nodi e la riparazione automatica utilizzando meccanismi Kubernetes standard, SageMaker HyperPod offre resilienza mirata e funzionalità aggiuntive progettate specificamente per la formazione ML su larga scala, come controlli approfonditi dello stato e ripresa automatica dei lavori.

- L'EKS Node Monitoring Agent con Node Auto Repair monitora continuamente lo stato dei nodi leggendo i log `Ready` e applicando NodeConditions, incluse condizioni standard come quelle specifiche dell'hardware accelerato per identificare problemi come guasti della GPU o della rete. Quando un nodo è considerato non integro, Node Auto Repair lo isola e lo sostituisce con un nuovo nodo. La riprogrammazione dei pod e il riavvio dei lavori si basano sui meccanismi standard di Kubernetes e sulla politica di riavvio del processo.
- L'agente per i controlli SageMaker HyperPodapprofonditi e il monitoraggio dello stato di salute monitora continuamente lo stato di salute delle istanze basate su GPU e Trainium. È personalizzato per i AI/ML carichi di lavoro e utilizza etichette (ad esempio) per gestire lo stato dei nodi. node-health-status Quando un nodo è considerato non integro, HyperPod attiva la

sostituzione automatica dell'hardware difettoso, ad esempio. GPUs Per impostazione predefinita, rileva i guasti relativi alla rete per l'EFA attraverso i suoi controlli sanitari di base e supporta la ripresa automatica dei lavori di formazione interrotti, consentendo di continuare i lavori dall'ultimo checkpoint e riducendo al minimo le interruzioni per le attività di machine learning su larga scala.

Sia per EKS Node Monitoring Agent con Auto Repair che per SageMaker HyperPod i cluster che utilizzano EFA, per monitorare metriche specifiche di EFA come errori di Remote Direct Memory Access (RDMA) e cadute di pacchetti, assicurati che il driver AWS EFA sia installato. Inoltre, consigliamo di implementare l'CloudWatch Observability Add-on o di utilizzare strumenti come DCGM Exporter con Prometheus e Grafana per monitorare EFA, GPU e, per esempio, metriche specifiche relative alle sue funzionalità. SageMaker HyperPod

Disabilita Karpenter Consolidation per carichi di lavoro sensibili alle interruzioni

Per i carichi di lavoro sensibili alle interruzioni, come l'elaborazione, le attività di AI/ML previsione su larga scala o la formazione, consigliamo di ottimizzare le politiche di consolidamento di Karpenter per evitare interruzioni durante l'esecuzione del lavoro. La funzionalità di consolidamento di Karpenter ottimizza automaticamente i costi del cluster chiudendo i nodi sottoutilizzati o sostituendoli con alternative più economiche. Tuttavia, anche quando un carico di lavoro utilizza completamente una GPU, Karpenter può consolidare i nodi se identifica un tipo di istanza di dimensioni adeguate a un prezzo inferiore che soddisfi i requisiti del pod, con conseguenti interruzioni del lavoro.

La politica di `WhenEmptyOrUnderutilized` consolidamento può chiudere i nodi prematuramente, con conseguenti tempi di esecuzione più lunghi. Ad esempio, le interruzioni possono ritardare la ripresa dei job a causa della riprogrammazione dei pod e del ricaricamento dei dati, il che potrebbe essere costoso per i lavori di inferenza in batch di lunga durata. Per ovviare a questo problema, puoi impostare `WhenEmpty` e configurare una `consolidateAfter` durata, ad esempio 1 ora, `consolidationPolicy` per conservare i nodi durante i picchi di carico di lavoro. Esempio:

```
disruption:
  consolidationPolicy: WhenEmpty
  consolidateAfter: 60m
```

Questo approccio migliora la latenza di avvio dei pod per carichi di lavoro di inferenza in batch complessi e altri lavori sensibili alle interruzioni, come l'elaborazione di dati di inferenza online in tempo reale o l'addestramento dei modelli, in cui il costo dell'interruzione supera i risparmi sui costi di elaborazione. NodePool Karpenter Disruption Budgets è un'altra funzionalità per la gestione delle interruzioni di Karpenter. Con i budget, puoi assicurarti che non più di un certo numero di nodi

vengano interrotti nei nodi prescelti in un determinato momento. NodePool Puoi anche utilizzare i budget per le interruzioni per evitare che tutti i nodi vengano interrotti in un determinato momento (ad esempio nelle ore di punta). Per ulteriori informazioni, consulta la documentazione di Karpenter Consolidation.

Usa ttlSecondsAfter Finished per la pulizia automatica dei lavori Kubernetes

Consigliamo di impostare `ttlSecondsAfterFinished` i job Kubernetes in Amazon EKS per eliminare automaticamente gli oggetti di lavoro completati. Gli oggetti di lavoro persistenti consumano risorse del cluster, come la memoria del server API, e complicano il monitoraggio con dashboard ingombranti (ad esempio Grafana, Amazon). CloudWatch Ad esempio, l'impostazione di un TTL di 1 ora garantisce che i lavori vengano rimossi subito dopo il completamento, mantenendo il cluster in ordine. Per ulteriori dettagli, consulta Pulizia automatica per lavori finiti.

Configurazione della priorità dei lavori a bassa priorità per lavori/carichi di lavoro con priorità più alta

Per i carichi di AI/ML lavoro a priorità mista su Amazon EKS, puoi configurare la priorità dei lavori a bassa priorità per garantire che le attività con priorità più alta (ad esempio, inferenza in tempo reale) ricevano risorse tempestivamente. Senza prelazione, i carichi di lavoro a bassa priorità come i processi batch (ad es. inferenza in batch, elaborazione dei dati), i servizi non batch (ad esempio, attività in background, cron job) o i lavori che richiedono un uso intensivo di CPU/memoria (ad esempio i servizi Web) possono ritardare i pod critici occupando i nodi. La priorità consente a Kubernetes di eliminare i pod a bassa priorità quando i pod ad alta priorità richiedono risorse, garantendo un'allocazione efficiente delle risorse sui nodi con GPUs o memoria. CPUs Ti consigliamo di utilizzare Kubernetes per assegnare priorità e controllare il comportamento di sfratto. `PriorityClass` `PodDisruptionBudget`

```
apiVersion: scheduling.k8s.io/v1
kind: PriorityClass
metadata:
  name: low-priority
value: 100
---
spec:
  priorityClassName: low-priority
```

Per ulteriori informazioni, consulta la documentazione sulla priorità e la priorità di Kubernetes.

Scalabilità e prestazioni delle applicazioni

Personalizza la capacità di calcolo per i carichi di lavoro ML con Karpenter o Static Nodes

Per garantire una capacità di calcolo efficiente in termini di costi e reattiva per i flussi di lavoro di machine learning (ML) su Amazon EKS, consigliamo di personalizzare la strategia di provisioning dei nodi in base alle caratteristiche del carico di lavoro e agli impegni di costo. Di seguito sono riportati due approcci da considerare: just-in-time scalabilità con Karpenter e gruppi di nodi statici per la capacità riservata.

- Just-in-time scaler sul piano di dati come Karpenter: per flussi di lavoro ML dinamici con richieste di calcolo variabili (ad esempio, inferenza basata su GPU seguita da grafici basati su CPU), consigliamo di utilizzare scalatori sul piano dati come Karpenter. just-in-time
- Usa gruppi di nodi statici per carichi di lavoro prevedibili: per carichi di lavoro ML prevedibili e stazionari o quando si utilizzano istanze riservate, i gruppi di nodi gestiti da EKS possono contribuire a garantire che la capacità riservata sia completamente fornita e utilizzata, massimizzando i risparmi. Questo approccio è ideale per tipi di istanze specifici eseguiti tramite o. RIs ODCRs

Esempio

Questo è un esempio di Karpenter diversificato NodePoolche consente il lancio di istanze g Amazon EC2 in cui la generazione di istanze è superiore a tre.

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: gpu-inference
spec:
  template:
    spec:
      nodeClassRef:
        group: karpenter.k8s.aws
        kind: EC2NodeClass
        name: default
      requirements:
        - key: karpenter.sh/capacity-type
          operator: In
```

```
          values: ["on-demand"]
        - key: karpenter.k8s.aws/instance-category
          operator: In
          values: ["g"]
        - key: karpenter.k8s.aws/instance-generation
          operator: Gt
          values: ["3"]
        - key: kubernetes.io/arch
          operator: In
          values: ["amd64"]
      taints:
        - key: nvidia.com/gpu
          effect: NoSchedule
  limits:
    cpu: "1000"
    memory: "4000Gi"
    nvidia.com/gpu: "10"  *# Limit the total number of GPUs to 10 for the NodePool*
  disruption:
    consolidationPolicy: WhenEmpty
    consolidateAfter: 60m
    expireAfter: 720h
```

Esempio

Esempio di utilizzo di gruppi di nodi statici per un carico di lavoro di formazione:

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata:
  name: ml-cluster
  region: us-west-2
managedNodeGroups:
  - name: gpu-node-group
    instanceType: p4d.24xlarge
    minSize: 2
    maxSize: 2
    desiredCapacity: 2
    taints:
      - key: nvidia.com/gpu
        effect: NoSchedule
```

Utilizza fattori e tolleranze per impedire che carichi di lavoro non accelerati vengano pianificati su istanze accelerate

La pianificazione di carichi di lavoro non accelerati su risorse GPU non è efficiente dal punto di vista del calcolo, pertanto consigliamo di utilizzare taints and toleration per garantire che i pod di carichi di lavoro non accelerati non siano pianificati su nodi inappropriati. Per ulteriori informazioni, consulta la documentazione di Kubernetes.

Scalabilità basata sulle prestazioni del modello

Per i carichi di lavoro di inferenza, consigliamo di utilizzare Kubernetes Event-Driven Autoscaling (KEDA) per scalare in base a metriche prestazionali del modello come le richieste di inferenza o il throughput dei token, con periodi di cooldown appropriati. Le politiche di scalabilità statica possono sovrafornire o sottodimensionare le risorse, con un impatto sui costi e sulla latenza. Scopri di più nella documentazione KEDA.

Allocazione dinamica delle risorse per la gestione avanzata delle GPU

L'allocazione dinamica delle risorse (DRA) rappresenta un progresso fondamentale nella gestione delle risorse GPU Kubernetes. DRA supera le tradizionali limitazioni dei plug-in dei dispositivi per consentire la condivisione sofisticata delle GPU, il riconoscimento della topologia e il coordinamento delle risorse tra i nodi. Disponibile nella versione 1.33 di Amazon EKS, DRA affronta le sfide critiche dei AI/ML carichi di lavoro fornendo quanto segue:

- Allocazione di GPU a grana fine
- Meccanismi di condivisione avanzati, come Multi-Process Service (MPS) e Multi-Instance GPU (MIG)
- Support per architetture hardware di nuova generazione, tra cui NVIDIA GB200 UltraServers

L'allocazione tradizionale delle GPU viene considerata GPUs come risorse intere opache, con conseguente notevole sottoutilizzo (spesso del 30-40% nei cluster di produzione). Ciò si verifica perché i carichi di lavoro ricevono l'accesso esclusivo all'intero anche quando richiedono solo risorse frazionarie. GPUs DRA trasforma questo modello introducendo un'allocazione strutturata e dichiarativa che fornisce allo scheduler Kubernetes una visibilità completa sulle caratteristiche hardware e sui requisiti del carico di lavoro. Ciò consente decisioni di posizionamento intelligenti e una condivisione efficiente delle risorse.

Vantaggi dell'utilizzo di DRA anziché del plug-in per dispositivi NVIDIA

Il plug-in per dispositivi NVIDIA (a partire dalla versione`0.12.0`) supporta meccanismi di condivisione della GPU tra cui time-slicing, MPS e MIG. Tuttavia, esistono limitazioni architettoniche che DRA affronta.

Limitazioni dei plug-in per dispositivi NVIDIA

- Configurazione statica: le configurazioni di condivisione della GPU (repliche time-slicing e impostazioni MPS) richiedono la preconfigurazione a livello di cluster. `ConfigMaps` Ciò rende difficile fornire strategie di condivisione diverse per carichi di lavoro diversi.
- Selezione granulare limitata: sebbene il plug-in del dispositivo esponga le caratteristiche della GPU tramite le etichette dei nodi, i carichi di lavoro non possono richiedere dinamicamente configurazioni GPU specifiche (dimensioni della memoria e capacità di calcolo) come parte della decisione di pianificazione.
- Nessun coordinamento delle risorse tra nodi: non è possibile gestire risorse GPU distribuite su più nodi o esprimere requisiti topologici complessi come i domini per sistemi come NVIDIA. NVLink GB200
- Vincoli dello scheduler: lo scheduler Kubernetes tratta le risorse GPU come numeri interi opachi, limitando la sua capacità di prendere decisioni basate sulla topologia o di gestire dipendenze complesse dalle risorse.
- Complessità di configurazione: l'impostazione di diverse strategie di condivisione richiede un'etichettatura multipla e attenta dei nodi, creando complessità operativa. `ConfigMaps`

Soluzioni con DRA

- Selezione dinamica delle risorse: DRA consente ai carichi di lavoro di specificare requisiti dettagliati (memoria GPU, versioni dei driver e attributi specifici) al momento della richiesta. `resourceclaims` Ciò consente una corrispondenza più flessibile delle risorse.
- Conoscenza della topologia: tramite parametri strutturati e selettori di dispositivi, DRA gestisce requisiti complessi come la comunicazione GPU tra nodi e le interconnessioni coerenti con la memoria.
- Gestione delle risorse tra nodi: `computeDomains` abilita il coordinamento delle risorse GPU distribuite su più nodi, fondamentale per sistemi come i canali IMEX. GB200

- Configurazione specifica per il carico di lavoro: ciascuna **`ResourceClaim`** specifica strategie e configurazioni di condivisione diverse, consentendo un controllo granulare per carico di lavoro anziché impostazioni a livello di cluster.
- Integrazione migliorata dello scheduler: DRA fornisce allo scheduler informazioni dettagliate sul dispositivo e consente decisioni di posizionamento più intelligenti in base alla topologia hardware e alle caratteristiche delle risorse.

Importante: DRA non sostituisce completamente il plug-in del dispositivo NVIDIA. Il driver NVIDIA DRA funziona insieme al plug-in del dispositivo per fornire funzionalità avanzate. Il plug-in del dispositivo continua a gestire l'individuazione e la gestione di base della GPU, mentre DRA aggiunge funzionalità avanzate di allocazione e pianificazione.

Istanze supportate da DRA e relative funzionalità

Il supporto DRA varia in base alla famiglia di istanze Amazon EC2 e all'architettura GPU, come illustrato nella tabella seguente.

Famiglia di istanze	Tipo GPU	Time-slicing	Supporto MIG	Supporto MPS	Supporto IMEX	Casi d’uso
G5	NVIDIA A10G	Sì	No	Sì	No	Carichi di lavoro di inferenza e grafica
G6	NVIDIA L4	Sì	No	Sì	No	Inferenza AI ed elaborazione video
G6e	NVIDIA L40S	Sì	No	Sì	No	Formazione, inferenza e grafica
P4D/ P4DE	NVIDIA A100	Sì	Sì	Sì	No	Formazione su larga scala e HPC
P5	NVIDIA H100	Sì	Sì	Sì	No	Formazione sul modello Foundation
P6	NVIDIA B200	Sì	Sì	Sì	No	Modelli con miliardi o trilioni di parametri

Famiglia di istanze	Tipo GPU	Time-slicing	Supporto MIG	Supporto MPS	Supporto IMEX	Casi d'uso
						, addestramento distribuito e inferenza
P6e	NVIDIA GB200	Sì	Sì	Sì	Sì	Modelli con miliardi o trilioni di parametri , addestramento distribuito e inferenza

Di seguito sono riportate le descrizioni di ciascuna funzionalità della tabella:

- Time-slicing: consente a più carichi di lavoro di condividere le risorse di calcolo della GPU nel tempo.
- GPU multiistanza (MIG): partizionamento a livello hardware che crea istanze GPU isolate.
- Servizio multiprocesso (MPS): consente l'esecuzione simultanea di più processi CUDA su una singola GPU.
- Internode Memory Exchange (IMEX): comunicazione coerente in termini di memoria tra i nodi per. GB200 UltraServers

Risorse aggiuntive

Per ulteriori informazioni sui driver Kubernetes DRA e NVIDIA DRA, consulta le seguenti risorse su: GitHub

- Kubernetes dynamic-resource-allocation
- Proposta di miglioramento di Kubernetes per DRA
- Driver NVIDIA DRA per GPUs
- Esempi e guida introduttiva di NVIDIA DRA

Configura l'allocazione dinamica delle risorse per la gestione avanzata delle GPU

L'argomento seguente mostra come configurare l'allocazione dinamica delle risorse (DRA) per la gestione avanzata delle GPU.

Prerequisiti

Prima di implementare DRA su Amazon EKS, assicurati che il tuo ambiente soddisfi i seguenti requisiti.

Configurazione del cluster

- Cluster Amazon EKS in esecuzione `1.33` o versione successiva
- Gruppi di nodi gestiti Amazon EKS (DRA è attualmente supportato solo da gruppi di nodi gestiti con AL2023 Bottlerocket ottimizzato per NVIDIA AMIs, non con Karpenter)
- Nodi di lavoro compatibili con GPU NVIDIA con tipi di istanze appropriati

Componenti richiesti

- Versione del plug-in per dispositivi NVIDIA `0.17.1` o successiva
- Versione del driver NVIDIA DRA o successiva `25.3.0`

Fase 1: Creare un cluster con un gruppo di nodi abilitato al DRA utilizzando eksctl

1. Crea un file di configurazione del cluster denominato: `dra-eks-cluster.yaml`

```
---
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: dra-eks-cluster
  region: us-west-2
  version: '1.33'

managedNodeGroups:
- name: gpu-dra-nodes
  amiFamily: AmazonLinux2023
  instanceType: g6.12xlarge
  desiredCapacity: 2
  minSize: 1
  maxSize: 3

  labels:
    node-type: "gpu-dra"
```

```
    nvidia.com/gpu.present: "true"

  taints:
  - key: nvidia.com/gpu
    value: "true"
    effect: NoSchedule
```

2. Crea il cluster:

```
eksctl create cluster -f dra-eks-cluster.yaml
```

Fase 2: Implementazione del plug-in per dispositivi NVIDIA

Implementa il plug-in per dispositivi NVIDIA per consentire l'individuazione di base della GPU:

1. Aggiungi il repository Helm del plug-in per dispositivi NVIDIA:

```
helm repo add nvidia https://nvidia.github.io/k8s-device-plugin
helm repo update
```

2. Crea valori personalizzati per il plug-in del dispositivo:

```
cat <<EOF > nvidia-device-plugin-values.yaml
gfd:
  enabled: true
nfd:
  enabled: true
tolerations:
  - key: nvidia.com/gpu
    operator: Exists
    effect: NoSchedule
EOF
```

3. Installa il plug-in del dispositivo NVIDIA:

```
helm install nvidia-device-plugin nvidia/nvidia-device-plugin \
 --namespace nvidia-device-plugin \
 --create-namespace \
 --version v0.17.1 \
 --values nvidia-device-plugin-values.yaml
```

Fase 3: Implementazione del driver NVIDIA DRA Helm chart

1. Crea un file di `dra-driver-values.yaml` valori per il driver DRA:

```
---
nvidiaDriverRoot: /

gpuResourcesEnabledOverride: true

resources:
  gpus:
    enabled: true
  computeDomains:
    enabled: true  # Enable for GB200 IMEX support

controller:
  tolerations:
  - key: nvidia.com/gpu
    operator: Exists
    effect: NoSchedule

kubeletPlugin:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: "nvidia.com/gpu.present"
            operator: In
            values: ["true"]
  tolerations:
  - key: nvidia.com/gpu
    operator: Exists
    effect: NoSchedule
```

2. Aggiungi il repository NVIDIA NGC Helm:

```
helm repo add nvidia https://helm.ngc.nvidia.com/nvidia
helm repo update
```

3. Installa il driver NVIDIA DRA:

```
helm install nvidia-dra-driver nvidia/nvidia-dra-driver-gpu \
```

```
--version="25.3.0-rc.2" \
--namespace nvidia-dra-driver \
--create-namespace \
--values dra-driver-values.yaml
```

Fase 4: Verificare l'installazione del DRA

1. Verificate che le risorse dell'API DRA siano disponibili:

   ```
   kubectl api-resources | grep resource.k8s.io/v1beta1
   ```

   Di seguito è riportato l'output previsto:

   ```
   deviceclasses resource.k8s.io/v1beta1 false DeviceClass
   resourceclaims resource.k8s.io/v1beta1 true ResourceClaim
   resourceclaimtemplates resource.k8s.io/v1beta1 true ResourceClaimTemplate
   resourceslices resource.k8s.io/v1beta1 false ResourceSlice
   ```

2. Controlla le classi di dispositivi disponibili:

   ```
   kubectl get deviceclasses
   ```

   Di seguito è riportato un esempio di output previsto:

   ```
   NAME                                        AGE
   compute-domain-daemon.nvidia.com            4h39m
   compute-domain-default-channel.nvidia.com   4h39m
   gpu.nvidia.com                              4h39m
   mig.nvidia.com                              4h39m
   ```

   Quando un'istanza GPU G6 appena creata si unisce al cluster Amazon EKS con DRA abilitato, si verificano le seguenti azioni:

   - Il driver NVIDIA DRA rileva automaticamente la GPU A10G e ne crea due su quel nodo. `resourceslices`
   - La `gpu.nvidia.com` slice registra il dispositivo GPU A10G fisico con le sue specifiche (memoria, capacità di elaborazione e altro).
   - Poiché A10G non supporta il partizionamento MIG, la `compute-domain.nvidia.com` slice crea un singolo dominio di calcolo che rappresenta l'intero contesto di calcolo della GPU.

- Questi `resourceslices` vengono quindi pubblicati sul server API Kubernetes, rendendo le risorse della GPU disponibili per la pianificazione. `resourceclaims`

  Lo scheduler DRA può ora allocare in modo intelligente questa GPU ai Pod che richiedono risorse GPU, fornendo una gestione delle risorse più flessibile rispetto agli approcci tradizionali basati sui plug-in dei dispositivi`resourceclaimtemplates`. Ciò avviene automaticamente senza intervento manuale. Il nodo diventa disponibile semplicemente per i carichi di lavoro della GPU una volta che il driver DRA ha completato il processo di individuazione e registrazione delle risorse.

  Quando si esegue il comando seguente:

  ```
  kubectl get resourceslices
  ```

  Di seguito è riportato un esempio di output previsto:

  ```
  NAME                                                        NODE
              DRIVER                          POOL                                  AGE
  ip-100-64-129-47.ec2.internal-compute-domain.nvidia.com-rwsts
   ip-100-64-129-47.ec2.internal    compute-domain.nvidia.com
   ip-100-64-129-47.ec2.internal    35m
  ip-100-64-129-47.ec2.internal-gpu.nvidia.com-6kndg
   ip-100-64-129-47.ec2.internal    gpu.nvidia.com
   ip-100-64-129-47.ec2.internal    35m
  ```

Continua su the section called “Pianifica un carico di lavoro GPU semplice utilizzando l'allocazione dinamica delle risorse”.

Pianifica un carico di lavoro GPU semplice utilizzando l'allocazione dinamica delle risorse

Per pianificare un carico di lavoro GPU semplice utilizzando l'allocazione dinamica delle risorse (DRA), procedi nel seguente modo. Prima di procedere, assicurati di averlo seguito. the section called “Configura l'allocazione dinamica delle risorse per la gestione avanzata delle GPU”

1. Crea una base `ResourceClaimTemplate` per l'allocazione della GPU con un file denominato: `basic-gpu-claim-template.yaml`

   ```
   ---
   ```

```
apiVersion: v1
kind: Namespace
metadata:
  name: gpu-test1

---
apiVersion: resource.k8s.io/v1beta1
kind: ResourceClaimTemplate
metadata:
  namespace: gpu-test1
  name: single-gpu
spec:
  spec:
    devices:
      requests:
      - name: gpu
        deviceClassName: gpu.nvidia.com
```

2. Applica il modello:

```
kubectl apply -f basic-gpu-claim-template.yaml
```

3. Verifica lo stato:

```
kubectl get resourceclaimtemplates -n gpu-test1
```

   Di seguito è riportato un output di esempio:

```
NAME         AGE
single-gpu   9m16s
```

4. Crea un Pod che usi il `ResourceClaimTemplate` con un file chiamato`basic-gpu-pod.yaml`:

```
---
apiVersion: v1
kind: Pod
metadata:
  namespace: gpu-test1
  name: gpu-pod
  labels:
    app: pod
spec:
```

```
  containers:
  - name: ctr0
    image: ubuntu:22.04
    command: ["bash", "-c"]
    args: ["nvidia-smi -L; trap 'exit 0' TERM; sleep 9999 & wait"]
    resources:
      claims:
      - name: gpu0
  resourceClaims:
  - name: gpu0
    resourceClaimTemplateName: single-gpu
  nodeSelector:
    NodeGroupType: gpu-dra
    nvidia.com/gpu.present: "true"
  tolerations:
  - key: "nvidia.com/gpu"
    operator: "Exists"
    effect: "NoSchedule"
```

5. Applica e monitora il Pod:

```
kubectl apply -f basic-gpu-pod.yaml
```

6. Controlla lo stato del Pod:

```
kubectl get pod -n gpu-test1
```

   Di seguito è riportato un esempio di output previsto:

```
NAME      READY   STATUS    RESTARTS   AGE
gpu-pod   1/1     Running   0          13m
```

7. Controlla lo `ResourceClaim` stato:

```
kubectl get resourceclaims -n gpu-test1
```

   Di seguito è riportato un esempio di output previsto:

```
NAME                 STATE                AGE
gpu-pod-gpu0-l76cg   allocated,reserved   9m6s
```

8. Visualizza i log dei Pod per visualizzare le informazioni sulla GPU:

```
kubectl logs gpu-pod -n gpu-test1
```

Di seguito è riportato un esempio di output previsto:

```
GPU 0: NVIDIA L4 (UUID: GPU-da7c24d7-c7e3-ed3b-418c-bcecc32af7c5)
```

Continua a leggere the section called “Tecniche di ottimizzazione della GPU con allocazione dinamica delle risorse” per tecniche di ottimizzazione della GPU più avanzate utilizzando DRA.

Tecniche di ottimizzazione della GPU con allocazione dinamica delle risorse

I carichi di lavoro GPU moderni richiedono una gestione sofisticata delle risorse per ottenere un utilizzo ottimale e un'efficienza dei costi. DRA consente diverse tecniche di ottimizzazione avanzate che rispondono a diversi casi d'uso e funzionalità hardware:

- Il time-slicing consente a più carichi di lavoro di condividere le risorse di calcolo della GPU nel tempo, rendendolo ideale per carichi di lavoro di inferenza con utilizzo sporadico della GPU. Per vedere un esempio, consulta the section called “Ottimizza i carichi di lavoro della GPU con time-slicing”.
- Il servizio Multi-Process (MPS) consente l'esecuzione simultanea di più processi CUDA su una singola GPU con un isolamento migliore rispetto al time-slicing. Per vedere un esempio, consulta the section called “Ottimizza i carichi di lavoro della GPU con MPS”.
- La GPU multiistanza (MIG) fornisce il partizionamento a livello hardware, creando istanze GPU isolate con risorse di calcolo e memoria dedicate. Per vedere un esempio, consulta the section called “Ottimizza i carichi di lavoro della GPU con GPU multiistanza”.
- Internode Memory Exchange (IMEX) consente una comunicazione coerente in termini di memoria tra i nodi per la formazione distribuita sui sistemi NVIDIA. GB200 Per vedere un esempio, consulta the section called “Ottimizza i carichi di lavoro della GPU con IMEX utilizzando istanze P6e GB200”.

Queste tecniche possono migliorare significativamente l'utilizzo delle risorse. Le organizzazioni segnalano che l'utilizzo della GPU aumenta dal 30-40% con l'allocazione tradizionale all'80-90% con strategie di condivisione ottimizzate. La scelta della tecnica dipende dalle caratteristiche del carico di lavoro, dai requisiti di isolamento e dalle capacità hardware.

Ottimizza i carichi di lavoro della GPU con time-slicing

Il time-slicing consente a più carichi di lavoro di condividere le risorse di calcolo della GPU pianificandone l'esecuzione in sequenza sulla stessa GPU fisica. È ideale per carichi di lavoro di inferenza con utilizzo sporadico della GPU.

Effettuare le seguenti operazioni.

1. Definite un file `ResourceClaimTemplate` per il time-slicing con un file denominato: `timeslicing-claim-template.yaml`

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: timeslicing-gpu

---
apiVersion: resource.k8s.io/v1beta1
kind: ResourceClaimTemplate
metadata:
  name: timeslicing-gpu-template
  namespace: timeslicing-gpu
spec:
  spec:
    devices:
      requests:
      - name: shared-gpu
        deviceClassName: gpu.nvidia.com
      config:
      - requests: ["shared-gpu"]
        opaque:
          driver: gpu.nvidia.com
          parameters:
            apiVersion: resource.nvidia.com/v1beta1
            kind: GpuConfig
            sharing:
              strategy: TimeSlicing
```

2. Definite un Pod usando il time-slicing con un file chiamato: `timeslicing-pod.yaml`

```
---
# Pod 1 - Inference workload
```

```
apiVersion: v1
kind: Pod
metadata:
  name: inference-pod-1
  namespace: timeslicing-gpu
  labels:
    app: gpu-inference
spec:
  restartPolicy: Never
  containers:
  - name: inference-container
    image: nvcr.io/nvidia/pytorch:25.04-py3
    command: ["python", "-c"]
    args:
    - |
      import torch
      import time
      import os
      print(f"=== POD 1 STARTING ===")
      print(f"GPU available: {torch.cuda.is_available()}")
      print(f"GPU count: {torch.cuda.device_count()}")
      if torch.cuda.is_available():
          device = torch.cuda.current_device()
          print(f"Current GPU: {torch.cuda.get_device_name(device)}")
          print(f"GPU Memory:
 {torch.cuda.get_device_properties(device).total_memory / 1024**3:.1f} GB")
          # Simulate inference workload
          for i in range(20):
              x = torch.randn(1000, 1000).cuda()
              y = torch.mm(x, x.t())
              print(f"Pod 1 - Iteration {i+1} completed at {time.strftime('%H:%M:
%S')}")
              time.sleep(60)
      else:
          print("No GPU available!")
          time.sleep(5)
    resources:
      claims:
      - name: shared-gpu-claim
  resourceClaims:
  - name: shared-gpu-claim
    resourceClaimTemplateName: timeslicing-gpu-template
  nodeSelector:
    NodeGroupType: "gpu-dra"
```

```
    nvidia.com/gpu.present: "true"
  tolerations:
  - key: nvidia.com/gpu
    operator: Exists
    effect: NoSchedule


---
# Pod 2 - Training workload
apiVersion: v1
kind: Pod
metadata:
  name: training-pod-2
  namespace: timeslicing-gpu
  labels:
    app: gpu-training
spec:
  restartPolicy: Never
  containers:
  - name: training-container
    image: nvcr.io/nvidia/pytorch:25.04-py3
    command: ["python", "-c"]
    args:
    - |
      import torch
      import time
      import os
      print(f"=== POD 2 STARTING ===")
      print(f"GPU available: {torch.cuda.is_available()}")
      print(f"GPU count: {torch.cuda.device_count()}")
      if torch.cuda.is_available():
          device = torch.cuda.current_device()
          print(f"Current GPU: {torch.cuda.get_device_name(device)}")
          print(f"GPU Memory:
 {torch.cuda.get_device_properties(device).total_memory / 1024**3:.1f} GB")
          # Simulate training workload with heavier compute
          for i in range(15):
              x = torch.randn(2000, 2000).cuda()
              y = torch.mm(x, x.t())
              loss = torch.sum(y)
              print(f"Pod 2 - Training step {i+1}, Loss: {loss.item():.2f} at
 {time.strftime('%H:%M:%S')}")
              time.sleep(5)
      else:
```

```
          print("No GPU available!")
          time.sleep(60)
    resources:
      claims:
      - name: shared-gpu-claim-2
  resourceClaims:
  - name: shared-gpu-claim-2
    resourceClaimTemplateName: timeslicing-gpu-template
  nodeSelector:
    NodeGroupType: "gpu-dra"
    nvidia.com/gpu.present: "true"
  tolerations:
  - key: nvidia.com/gpu
    operator: Exists
    effect: NoSchedule
```

3. Applica il modello e il Pod:

```
kubectl apply -f timeslicing-claim-template.yaml
kubectl apply -f timeslicing-pod.yaml
```

4. Monitora le richieste di risorse:

```
kubectl get resourceclaims -n timeslicing-gpu -w
```

   Di seguito è riportato un output di esempio:

```
NAME                                        STATE                AGE
inference-pod-1-shared-gpu-claim-9p97x      allocated,reserved   21s
training-pod-2-shared-gpu-claim-2-qghnb     pending              21s
inference-pod-1-shared-gpu-claim-9p97x      pending              105s
training-pod-2-shared-gpu-claim-2-qghnb     pending              105s
inference-pod-1-shared-gpu-claim-9p97x      pending              105s
training-pod-2-shared-gpu-claim-2-qghnb     allocated,reserved   105s
inference-pod-1-shared-gpu-claim-9p97x      pending              105s
```

Primo Pod (`inference-pod-1`)

- Stato: `allocated,reserved`
- Significato: DRA ha trovato una GPU disponibile e l'ha riservata per questo Pod

- Stato del pod: inizia a funzionare immediatamente

Secondo Pod (`training-pod-2`)

- Stato: `pending`
- Significato: in attesa che DRA configuri il time-slicing sulla stessa GPU
- Stato del pod: In attesa di essere programmato
- Lo stato passerà dal `pending` `allocated,reserved` al `running`

Ottimizza i carichi di lavoro della GPU con MPS

Multi-Process Service (MPS) consente l'esecuzione simultanea di più contesti CUDA su una singola GPU con un isolamento migliore rispetto al time-slicing.

Effettuare le seguenti operazioni.

1. Definite un file `ResourceClaimTemplate` per MPS con il nome`mps-claim-template.yaml`:

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: mps-gpu

---
apiVersion: resource.k8s.io/v1beta1
kind: ResourceClaimTemplate
metadata:
  name: mps-gpu-template
  namespace: mps-gpu
spec:
  spec:
    devices:
      requests:
      - name: shared-gpu
        deviceClassName: gpu.nvidia.com
      config:
      - requests: ["shared-gpu"]
        opaque:
          driver: gpu.nvidia.com
          parameters:
```

```
            apiVersion: resource.nvidia.com/v1beta1
            kind: GpuConfig
            sharing:
              strategy: MPS
```

2. Definite un Pod utilizzando MPS con un file denominato: `mps-pod.yaml`

```
---
# Single Pod with Multiple Containers sharing GPU via MPS
apiVersion: v1
kind: Pod
metadata:
  name: mps-multi-container-pod
  namespace: mps-gpu
  labels:
    app: mps-demo
spec:
  restartPolicy: Never
  containers:
  # Container 1 - Inference workload
  - name: inference-container
    image: nvcr.io/nvidia/pytorch:25.04-py3
    command: ["python", "-c"]
    args:
    - |
      import torch
      import torch.nn as nn
      import time
      import os

      print(f"=== INFERENCE CONTAINER STARTING ===")
      print(f"Process ID: {os.getpid()}")
      print(f"GPU available: {torch.cuda.is_available()}")
      print(f"GPU count: {torch.cuda.device_count()}")

      if torch.cuda.is_available():
          device = torch.cuda.current_device()
          print(f"Current GPU: {torch.cuda.get_device_name(device)}")
          print(f"GPU Memory:
 {torch.cuda.get_device_properties(device).total_memory / 1024**3:.1f} GB")

          # Create inference model
          model = nn.Sequential(
              nn.Linear(1000, 500),
```

```
            nn.ReLU(),
            nn.Linear(500, 100)
        ).cuda()

        # Run inference
        for i in range(1, 999999):
            with torch.no_grad():
                x = torch.randn(128, 1000).cuda()
                output = model(x)
                result = torch.sum(output)
                print(f"Inference Container PID {os.getpid()}: Batch {i}, Result:
{result.item():.2f} at {time.strftime('%H:%M:%S')}")
            time.sleep(2)
    else:
        print("No GPU available!")
        time.sleep(60)
  resources:
    claims:
    - name: shared-gpu-claim
      request: shared-gpu

 # Container 2 - Training workload
 - name: training-container
   image: nvcr.io/nvidia/pytorch:25.04-py3
   command: ["python", "-c"]
   args:
   - |
     import torch
     import torch.nn as nn
     import time
     import os

     print(f"=== TRAINING CONTAINER STARTING ===")
     print(f"Process ID: {os.getpid()}")
     print(f"GPU available: {torch.cuda.is_available()}")
     print(f"GPU count: {torch.cuda.device_count()}")

     if torch.cuda.is_available():
         device = torch.cuda.current_device()
         print(f"Current GPU: {torch.cuda.get_device_name(device)}")
         print(f"GPU Memory:
{torch.cuda.get_device_properties(device).total_memory / 1024**3:.1f} GB")

         # Create training model
```

```
          model = nn.Sequential(
              nn.Linear(2000, 1000),
              nn.ReLU(),
              nn.Linear(1000, 500),
              nn.ReLU(),
              nn.Linear(500, 10)
          ).cuda()

          criterion = nn.MSELoss()
          optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

          # Run training
          for epoch in range(1, 999999):
              x = torch.randn(64, 2000).cuda()
              target = torch.randn(64, 10).cuda()

              optimizer.zero_grad()
              output = model(x)
              loss = criterion(output, target)
              loss.backward()
              optimizer.step()

              print(f"Training Container PID {os.getpid()}: Epoch {epoch}, Loss:
{loss.item():.4f} at {time.strftime('%H:%M:%S')}")
              time.sleep(3)
      else:
          print("No GPU available!")
          time.sleep(60)
    resources:
      claims:
      - name: shared-gpu-claim
        request: shared-gpu

  resourceClaims:
  - name: shared-gpu-claim
    resourceClaimTemplateName: mps-gpu-template

  nodeSelector:
    NodeGroupType: "gpu-dra"
    nvidia.com/gpu.present: "true"
  tolerations:
  - key: nvidia.com/gpu
    operator: Exists
```

```
    effect: NoSchedule
```

3. Applica il modello e crea più MPS Pod:

```
kubectl apply -f mps-claim-template.yaml
kubectl apply -f mps-pod.yaml
```

4. Monitora le richieste di risorse:

```
kubectl get resourceclaims -n mps-gpu -w
```

   Di seguito è riportato un output di esempio:

```
NAME                                                STATE                AGE
mps-multi-container-pod-shared-gpu-claim-2p9kx   allocated,reserved   86s
```

Questa configurazione dimostra una vera condivisione della GPU utilizzando NVIDIA Multi-Process Service (MPS) tramite l'allocazione dinamica delle risorse (DRA). A differenza del time-slicing, in cui i carichi di lavoro utilizzano a turno la GPU in sequenza, MPS consente l'esecuzione simultanea di entrambi i container sulla stessa GPU fisica. L'intuizione principale è che la condivisione di DRA MPS richiede più contenitori all'interno di un singolo Pod, non più Pod separati. Una volta distribuito, il driver DRA ne `ResourceClaim` assegna uno al Pod e configura automaticamente MPS per consentire l'esecuzione simultanea dei contenitori di inferenza e di addestramento.

Ogni contenitore dispone del proprio spazio di memoria GPU e delle proprie risorse di calcolo isolate, con il daemon MPS che coordina l'accesso all'hardware sottostante. È possibile verificare che funzioni effettuando le seguenti operazioni:

- Verifican`vidia-smi`, che mostrerà entrambi i contenitori come processi M+C (`MPS + Compute`) che condividono lo stesso dispositivo GPU.
- Monitoraggio dei log di entrambi i contenitori, che mostreranno timestamp interlacciati che dimostrano l'esecuzione simultanea.

Questo approccio massimizza l'utilizzo della GPU consentendo ai carichi di lavoro complementari di condividere in modo efficiente il costoso hardware della GPU, anziché lasciarlo sottoutilizzato da un singolo processo.

Contenitore 1: **`inference-container`**

```
root@mps-multi-container-pod:/workspace# nvidia-smi
Wed Jul 16 21:09:30 2025
+-----------------------------------------------------------------------------------------
+
| NVIDIA-SMI 570.158.01             Driver Version: 570.158.01     CUDA Version: 12.9
   |
|-----------------------------------------+------------------------
+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr.
 ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute
 M. |
|                                         |                        |               MIG
 M. |
|=========================================+========================
+======================|
|   0  NVIDIA L4                      On  |   00000000:35:00.0 Off |
 0 |
| N/A   48C    P0             28W /   72W |     597MiB /  23034MiB |      0%   E.
 Process |
|                                         |                        |
 N/A |
+-----------------------------------------+------------------------
+----------------------+

+-----------------------------------------------------------------------------------------
+
| Processes:
   |
|  GPU   GI   CI              PID   Type   Process name                        GPU
 Memory |
|        ID   ID                                                               Usage
   |
|
=========================================================================================|
|    0   N/A  N/A               1    M+C   python
 246MiB |
+-----------------------------------------------------------------------------------------
+
```

Contenitore 2: **training-container**

```
root@mps-multi-container-pod:/workspace# nvidia-smi
Wed Jul 16 21:16:00 2025
+-----------------------------------------------------------------------------------------
+
| NVIDIA-SMI 570.158.01             Driver Version: 570.158.01     CUDA Version: 12.9
   |
|-----------------------------------------+------------------------
+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr.
 ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute
 M. |
|                                         |                        |               MIG
 M. |
|=========================================+========================
+======================|
|   0  NVIDIA L4                      On  |   00000000:35:00.0 Off |
 0 |
| N/A   51C    P0             28W /   72W |     597MiB /  23034MiB |      0%   E.
 Process |
|                                         |                        |
 N/A |
+-----------------------------------------+------------------------
+----------------------+

+-----------------------------------------------------------------------------------------
+
| Processes:
   |
|  GPU   GI   CI              PID   Type   Process name                        GPU
 Memory |
|        ID   ID                                                               Usage
   |
|
=========================================================================================|
|    0   N/A  N/A               1    M+C   python
 314MiB |
+-----------------------------------------------------------------------------------------
+
```

Ottimizza i carichi di lavoro della GPU con GPU multiistanza

La GPU multiistanza (MIG) fornisce il partizionamento a livello hardware, creando istanze GPU isolate con risorse di calcolo e memoria dedicate.

L'utilizzo del partizionamento MIG dinamico con vari profili richiede l'operatore GPU NVIDIA. L'operatore GPU NVIDIA utilizza MIG Manager per creare profili MIG e riavviare le istanze GPU come P4D, P4De, P5, P6 e altre per applicare le modifiche alla configurazione. L'operatore GPU include funzionalità complete di gestione MIG tramite il componente MIG Manager, che controlla le modifiche alle etichette dei nodi e applica automaticamente la configurazione MIG appropriata. Quando viene richiesta una modifica del profilo MIG, l'operatore spegne automaticamente tutti i client GPU, applica la nuova geometria della partizione e riavvia i servizi interessati. Questo processo richiede il riavvio del nodo per le istanze GPU per garantire transizioni di stato della GPU pulite. Ecco perché l'abilitazione `WITH0REB00T=true` nella configurazione MIG Manager è essenziale per implementazioni MIG di successo.

Sono necessari sia NVIDIA DRA Driver che NVIDIA GPU Operator per lavorare con MIG in Amazon EKS. Inoltre, non sono necessari NVIDIA Device Plugin e DCGM Exporter, poiché fanno parte di NVIDIA GPU Operator. Poiché EKS NVIDIA AMIs viene fornito con i driver NVIDIA preinstallati, abbiamo disabilitato l'implementazione dei driver da parte dell'operatore GPU per evitare conflitti e sfruttare i driver ottimizzati già presenti sulle istanze. Il driver NVIDIA DRA gestisce l'allocazione dinamica delle risorse per le istanze MIG, mentre l'operatore GPU gestisce l'intero ciclo di vita della GPU. Ciò include la configurazione MIG, la funzionalità dei plug-in dei dispositivi, il monitoraggio tramite DCGM e l'individuazione delle funzionalità dei nodi. Questo approccio integrato offre una soluzione completa per la gestione delle GPU aziendali, con isolamento a livello hardware e funzionalità di allocazione dinamica delle risorse.

Fase 1: Implementare NVIDIA GPU Operator

1. Aggiungi il repository NVIDIA GPU Operator:

```
helm repo add nvidia https://nvidia.github.io/gpu-operator
helm repo update
```

2. Crea un file: `gpu-operator-values.yaml`

```
driver:
  enabled: false

mig:
```

```
  strategy: mixed

migManager:
  enabled: true
  env:
    - name: WITH_REBOOT
      value: "true"
  config:
    create: true
    name: custom-mig-parted-configs
    default: "all-disabled"
    data:
      config.yaml: |-
        version: v1
        mig-configs:
          all-disabled:
            - devices: all
              mig-enabled: false

          # P4D profiles (A100 40GB)
          p4d-half-balanced:
            - devices: [0, 1, 2, 3]
              mig-enabled: true
              mig-devices:
                "1g.5gb": 2
                "2g.10gb": 1
                "3g.20gb": 1
            - devices: [4, 5, 6, 7]
              mig-enabled: false

          # P4DE profiles (A100 80GB)
          p4de-half-balanced:
            - devices: [0, 1, 2, 3]
              mig-enabled: true
              mig-devices:
                "1g.10gb": 2
                "2g.20gb": 1
                "3g.40gb": 1
            - devices: [4, 5, 6, 7]
              mig-enabled: false

devicePlugin:
  enabled: true
  config:
```

```
    name: ""
    create: false
    default: ""

toolkit:
  enabled: true

nfd:
  enabled: true

gfd:
  enabled: true

dcgmExporter:
  enabled: true
  serviceMonitor:
    enabled: true
    interval: 15s
    honorLabels: false
    additionalLabels:
      release: kube-prometheus-stack

nodeStatusExporter:
  enabled: false

operator:
  defaultRuntime: containerd
  runtimeClass: nvidia
  resources:
    limits:
      cpu: 500m
      memory: 350Mi
    requests:
      cpu: 200m
      memory: 100Mi

daemonsets:
  tolerations:
    - key: "nvidia.com/gpu"
      operator: "Exists"
      effect: "NoSchedule"
  nodeSelector:
    accelerator: nvidia
```

```
  priorityClassName: system-node-critical
```

3. Installa GPU Operator utilizzando il `gpu-operator-values.yaml` file:

```
helm install gpu-operator nvidia/gpu-operator \
  --namespace gpu-operator \
  --create-namespace \
  --version v25.3.1 \
  --values gpu-operator-values.yaml
```

   Questo grafico Helm utilizza i seguenti componenti e più profili MIG:

   - Device Plugin (pianificazione delle risorse GPU)
   - DCGM Exporter (metriche e monitoraggio della GPU)
   - Node Feature Discovery (NFD - etichettatura hardware)
   - GPU Feature Discovery (GFD - etichettatura specifica per GPU)
   - MIG Manager (partizionamento GPU multiistanza)
   - Container Toolkit (runtime del contenitore GPU)
   - Operator Controller (gestione del ciclo di vita)

4. Verifica i pod di distribuzione:

```
kubectl get pods -n gpu-operator
```

   Di seguito è riportato un output di esempio:

```
NAME                                                          READY   STATUS
 RESTARTS        AGE
gpu-feature-discovery-27rdq                                   1/1     Running
 0               3h31m
gpu-operator-555774698d-48brn                                 1/1     Running
 0               4h8m
nvidia-container-toolkit-daemonset-sxmh9                      1/1     Running
 1 (3h32m ago)   4h1m
nvidia-cuda-validator-qb77g                                   0/1     Completed
 0               3h31m
nvidia-dcgm-exporter-cvzd7                                    1/1     Running
 0               3h31m
nvidia-device-plugin-daemonset-5ljm5                          1/1     Running
 0               3h31m
```

```
nvidia-gpu-operator-node-feature-discovery-gc-67f66fc557-q5wkt    1/1     Running
 0               4h8m
nvidia-gpu-operator-node-feature-discovery-master-5d8ffddcsl6s6   1/1     Running
 0               4h8m
nvidia-gpu-operator-node-feature-discovery-worker-6t4w7           1/1     Running
 1 (3h32m ago)   4h1m
nvidia-gpu-operator-node-feature-discovery-worker-9w7g8           1/1     Running
 0               4h8m
nvidia-gpu-operator-node-feature-discovery-worker-k5fgs           1/1     Running
 0               4h8m
nvidia-mig-manager-zvf54                                          1/1     Running
 1 (3h32m ago)   3h35m
```

5. Crea un cluster Amazon EKS con un gruppo di nodi gestiti P4de per testare gli esempi MIG:

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: dra-eks-cluster
  region: us-east-1
  version: '1.33'

managedNodeGroups:
# P4DE MIG Node Group with Capacity Block Reservation
- name: p4de-mig-nodes
  amiFamily: AmazonLinux2023
  instanceType: p4de.24xlarge

  # Capacity settings
  desiredCapacity: 0
  minSize: 0
  maxSize: 1

  # Use specific subnet in us-east-1b for capacity reservation
  subnets:
    - us-east-1b

  # AL2023 NodeConfig for RAID0 local storage only
  nodeadmConfig:
    apiVersion: node.eks.aws/v1alpha1
    kind: NodeConfig
    spec:
      instance:
```

```
        localStorage:
          strategy: RAID0

  # Node labels for MIG configuration
  labels:
    nvidia.com/gpu.present: "true"
    nvidia.com/gpu.product: "A100-SXM4-80GB"
    nvidia.com/mig.config: "p4de-half-balanced"
    node-type: "p4de"
    vpc.amazonaws.com/efa.present: "true"
    accelerator: "nvidia"

  # Node taints
  taints:
    - key: nvidia.com/gpu
      value: "true"
      effect: NoSchedule

  # EFA support
  efaEnabled: true

  # Placement group for high-performance networking
  placementGroup:
    groupName: p4de-placement-group
    strategy: cluster

  # Capacity Block Reservation (CBR)
  # Ensure CBR ID matches the subnet AZ with the Nodegroup subnet
  spot: false
  capacityReservation:
    capacityReservationTarget:
      capacityReservationId: "cr-abcdefghij"  # Replace with your capacity
 reservation ID
```

NVIDIA GPU Operator utilizza l'etichetta aggiunta ai nodi `nvidia.com/mig.config: "p4de-half-balanced"` e partiziona la GPU con il profilo specificato.

6. Accedi all'istanza. `p4de`
7. Esegui il comando seguente:

```
nvidia-smi -L
```

Dovresti vedere il seguente output di esempio:

```
[root@ip-100-64-173-145 bin]# nvidia-smi -L
GPU 0: NVIDIA A100-SXM4-80GB (UUID: GPU-ab52e33c-be48-38f2-119e-b62b9935925a)
  MIG 3g.40gb     Device  0: (UUID: MIG-da972af8-a20a-5f51-849f-bc0439f7970e)
  MIG 2g.20gb     Device  1: (UUID: MIG-7f9768b7-11a6-5de9-a8aa-e9c424400da4)
  MIG 1g.10gb     Device  2: (UUID: MIG-498adad6-6cf7-53af-9d1a-10cfd1fa53b2)
  MIG 1g.10gb     Device  3: (UUID: MIG-3f55ef65-1991-571a-ac50-0dbf50d80c5a)
GPU 1: NVIDIA A100-SXM4-80GB (UUID: GPU-0eabeccc-7498-c282-0ac7-d3c09f6af0c8)
  MIG 3g.40gb     Device  0: (UUID: MIG-80543849-ea3b-595b-b162-847568fe6e0e)
  MIG 2g.20gb     Device  1: (UUID: MIG-3af1958f-fac4-59f1-8477-9f8d08c55029)
  MIG 1g.10gb     Device  2: (UUID: MIG-401088d2-716f-527b-a970-b1fc7a4ac6b2)
  MIG 1g.10gb     Device  3: (UUID: MIG-8c56c75e-5141-501c-8f43-8cf22f422569)
GPU 2: NVIDIA A100-SXM4-80GB (UUID: GPU-1c7a1289-243f-7872-a35c-1d2d8af22dd0)
  MIG 3g.40gb     Device  0: (UUID: MIG-e9b44486-09fc-591a-b904-0d378caf2276)
  MIG 2g.20gb     Device  1: (UUID: MIG-ded93941-9f64-56a3-a9b1-a129c6edf6e4)
  MIG 1g.10gb     Device  2: (UUID: MIG-6c317d83-a078-5c25-9fa3-c8308b379aa1)
  MIG 1g.10gb     Device  3: (UUID: MIG-2b070d39-d4e9-5b11-bda6-e903372e3d08)
GPU 3: NVIDIA A100-SXM4-80GB (UUID: GPU-9a6250e2-5c59-10b7-2da8-b61d8a937233)
  MIG 3g.40gb     Device  0: (UUID: MIG-20e3cd87-7a57-5f1b-82e7-97b14ab1a5aa)
  MIG 2g.20gb     Device  1: (UUID: MIG-04430354-1575-5b42-95f4-bda6901f1ace)
  MIG 1g.10gb     Device  2: (UUID: MIG-d62ec8b6-e097-5e99-a60c-abf8eb906f91)
  MIG 1g.10gb     Device  3: (UUID: MIG-fce20069-2baa-5dd4-988a-cead08348ada)
GPU 4: NVIDIA A100-SXM4-80GB (UUID: GPU-5d09daf0-c2eb-75fd-3919-7ad8fafa5f86)
GPU 5: NVIDIA A100-SXM4-80GB (UUID: GPU-99194e04-ab2a-b519-4793-81cb2e8e9179)
GPU 6: NVIDIA A100-SXM4-80GB (UUID: GPU-c1a1910f-465a-e16f-5af1-c6aafe499cd6)
GPU 7: NVIDIA A100-SXM4-80GB (UUID: GPU-c2cfafbc-fd6e-2679-e955-2a9e09377f78)
```

NVIDIA GPU Operator ha applicato con successo il profilo `p4de-half-balanced` MIG all'istanza P4DE, creando partizioni GPU a livello hardware come configurato. Ecco come funziona il partizionamento:

L'operatore GPU ha applicato questa configurazione dal profilo MIG incorporato:

```
p4de-half-balanced:
  - devices: [0, 1, 2, 3]        # First 4 GPUs: MIG enabled
    mig-enabled: true
    mig-devices:
      "1g.10gb": 2               # 2x small instances (10GB each)
      "2g.20gb": 1               # 1x medium instance (20GB)
      "3g.40gb": 1               # 1x large instance (40GB)
```

```
  - devices: [4, 5, 6, 7]        # Last 4 GPUs: Full GPUs
    mig-enabled: false
```

In base al tuo `nvidia-smi -L` output, ecco cosa ha creato l'operatore GPU:

- Compatibile con MIG GPUs (0-3): hardware partizionato
  - GPU 0: NVIDIA A100- 80 GB SXM4
    - MIG 3g.40gb Device 0 — Carichi di lavoro di grandi dimensioni (40 GB di memoria, 42) SMs
    - MIG 2g.20gb Device 1 — Carichi di lavoro medi (20 GB di memoria, 28) SMs
    - MIG 1g.10gb Device 2 — Piccoli carichi di lavoro (10 GB di memoria, 14) SMs
    - MIG 1g.10gb Device 3 — Piccoli carichi di lavoro (10 GB di memoria, 14) SMs
  - GPU 1: NVIDIA A100- -80 GB SXM4
    - MIG 3g.40gb Device 0 — Layout di partizione identico
    - MIG 2g.20gb Dispositivo 1
    - MIG 1g.10gb Dispositivo 2
    - MIG 1g.10gb Dispositivo 3
  - GPU 2 e GPU 3: stesso schema di GPU 0 e GPU 1
- Completo GPUs (4-7): nessun partizionamento MIG
  - GPU 4: NVIDIA A100- -80 GB — GPU completa da 80 GB SXM4
  - GPU 5: NVIDIA A100- 80 GB — GPU completa da 80 GB SXM4
  - GPU 6: NVIDIA A100- 80 GB — GPU completa da 80 GB SXM4
  - GPU 7: NVIDIA A100- 80 GB — GPU completa da 80 GB SXM4

Una volta che l'operatore GPU NVIDIA crea le partizioni MIG, il driver NVIDIA DRA rileva automaticamente queste istanze isolate dall'hardware e le rende disponibili per l'allocazione dinamica delle risorse in Kubernetes. Il driver DRA rileva ogni istanza MIG con il suo profilo specifico (1g.10gb, 2g.20gb, 3g.40gb) e le espone come risorse pianificabili attraverso la classe del dispositivo. `mig.nvidia.com`

Il driver DRA monitora continuamente la topologia MIG e mantiene un inventario delle istanze disponibili in tutte. GPUs Quando un Pod richiede un profilo MIG specifico tramite a`ResourceClaimTemplate`, il driver DRA seleziona in modo intelligente l'istanza MIG appropriata da qualsiasi GPU disponibile, abilitando una vera multi-tenancy a livello hardware. Questa

allocazione dinamica consente l'esecuzione simultanea di più carichi di lavoro isolati sulla stessa GPU fisica, mantenendo rigidi limiti di risorse e garanzie prestazionali.

Fase 2: Verificare l'allocazione delle risorse MIG

Eseguiamo ora alcuni esempi per dimostrare come DRA alloca dinamicamente le istanze MIG a diversi carichi di lavoro. Implementate `resourceclaimtemplates` e testate i pod per vedere come il driver DRA colloca i carichi di lavoro tra le partizioni MIG disponibili, consentendo a più container di condividere le risorse GPU con isolamento a livello di hardware.

1. Crea `resourceclaimtemplates` per contenere il MIG: `mig-claim-template.yaml`

```
apiVersion: v1
kind: Namespace
metadata:
  name: mig-gpu

---
# Template for 3g.40gb MIG instance (Large training)
apiVersion: resource.k8s.io/v1beta1
kind: ResourceClaimTemplate
metadata:
  name: mig-large-template
  namespace: mig-gpu
spec:
  spec:
    devices:
      requests:
      - name: mig-large
        deviceClassName: mig.nvidia.com
        selectors:
        - cel:
            expression: |
              device.attributes['gpu.nvidia.com'].profile == '3g.40gb'

---
# Template for 2g.20gb MIG instance (Medium training)
apiVersion: resource.k8s.io/v1beta1
kind: ResourceClaimTemplate
metadata:
  name: mig-medium-template
  namespace: mig-gpu
spec:
```

```
  spec:
    devices:
      requests:
      - name: mig-medium
        deviceClassName: mig.nvidia.com
        selectors:
        - cel:
            expression: |
              device.attributes['gpu.nvidia.com'].profile == '2g.20gb'

---
# Template for 1g.10gb MIG instance (Small inference)
apiVersion: resource.k8s.io/v1beta1
kind: ResourceClaimTemplate
metadata:
  name: mig-small-template
  namespace: mig-gpu
spec:
  spec:
    devices:
      requests:
      - name: mig-small
        deviceClassName: mig.nvidia.com
        selectors:
        - cel:
            expression: |
              device.attributes['gpu.nvidia.com'].profile == '1g.10gb'
```

2. Applica i tre modelli:

```
kubectl apply -f mig-claim-template.yaml
```

3. Esegui il comando seguente:

```
kubectl get resourceclaimtemplates -n mig-gpu
```

Di seguito è riportato un output di esempio:

```
NAME                 AGE
mig-large-template   71m
mig-medium-template  71m
mig-small-template   71m
```

4. Crea `mig-pod.yaml` per pianificare più lavori per sfruttare questo `resourceclaimtemplates` vantaggio:

```
---
# ConfigMap containing Python scripts for MIG pods
apiVersion: v1
kind: ConfigMap
metadata:
  name: mig-scripts-configmap
  namespace: mig-gpu
data:
  large-training-script.py: |
    import torch
    import torch.nn as nn
    import torch.optim as optim
    import time
    import os

    print(f"=== LARGE TRAINING POD (3g.40gb) ===")
    print(f"Process ID: {os.getpid()}")
    print(f"GPU available: {torch.cuda.is_available()}")
    print(f"GPU count: {torch.cuda.device_count()}")

    if torch.cuda.is_available():
        device = torch.cuda.current_device()
        print(f"Using GPU: {torch.cuda.get_device_name(device)}")
        print(f"GPU Memory: {torch.cuda.get_device_properties(device).total_memory /
 1e9:.1f} GB")

        # Large model for 3g.40gb instance
        model = nn.Sequential(
            nn.Linear(2048, 1024),
            nn.ReLU(),
            nn.Linear(1024, 512),
            nn.ReLU(),
            nn.Linear(512, 256),
            nn.ReLU(),
            nn.Linear(256, 10)
        ).cuda()

        optimizer = optim.Adam(model.parameters())
        criterion = nn.CrossEntropyLoss()
```

```
        print(f"Model parameters: {sum(p.numel() for p in model.parameters())}")

        # Training loop
        for epoch in range(100):
            # Large batch for 3g.40gb
            x = torch.randn(256, 2048).cuda()
            y = torch.randint(0, 10, (256,)).cuda()

            optimizer.zero_grad()
            output = model(x)
            loss = criterion(output, y)
            loss.backward()
            optimizer.step()

            if epoch % 10 == 0:
                print(f"Large Training - Epoch {epoch}, Loss: {loss.item():.4f}, GPU
Memory: {torch.cuda.memory_allocated()/1e9:.2f}GB")
            time.sleep(3)

        print("Large training completed on 3g.40gb MIG instance")

 medium-training-script.py: |
   import torch
   import torch.nn as nn
   import torch.optim as optim
   import time
   import os

   print(f"=== MEDIUM TRAINING POD (2g.20gb) ===")
   print(f"Process ID: {os.getpid()}")
   print(f"GPU available: {torch.cuda.is_available()}")
   print(f"GPU count: {torch.cuda.device_count()}")

   if torch.cuda.is_available():
       device = torch.cuda.current_device()
       print(f"Using GPU: {torch.cuda.get_device_name(device)}")
       print(f"GPU Memory: {torch.cuda.get_device_properties(device).total_memory /
1e9:.1f} GB")

       # Medium model for 2g.20gb instance
       model = nn.Sequential(
           nn.Linear(1024, 512),
           nn.ReLU(),
           nn.Linear(512, 256),
```

```
            nn.ReLU(),
            nn.Linear(256, 10)
        ).cuda()

        optimizer = optim.Adam(model.parameters())
        criterion = nn.CrossEntropyLoss()

        print(f"Model parameters: {sum(p.numel() for p in model.parameters())}")

        # Training loop
        for epoch in range(100):
            # Medium batch for 2g.20gb
            x = torch.randn(128, 1024).cuda()
            y = torch.randint(0, 10, (128,)).cuda()

            optimizer.zero_grad()
            output = model(x)
            loss = criterion(output, y)
            loss.backward()
            optimizer.step()

            if epoch % 10 == 0:
                print(f"Medium Training - Epoch {epoch}, Loss: {loss.item():.4f}, GPU
Memory: {torch.cuda.memory_allocated()/1e9:.2f}GB")
            time.sleep(4)

        print("Medium training completed on 2g.20gb MIG instance")

 small-inference-script.py: |
   import torch
   import torch.nn as nn
   import time
   import os

   print(f"=== SMALL INFERENCE POD (1g.10gb) ===")
   print(f"Process ID: {os.getpid()}")
   print(f"GPU available: {torch.cuda.is_available()}")
   print(f"GPU count: {torch.cuda.device_count()}")

   if torch.cuda.is_available():
       device = torch.cuda.current_device()
       print(f"Using GPU: {torch.cuda.get_device_name(device)}")
       print(f"GPU Memory: {torch.cuda.get_device_properties(device).total_memory /
1e9:.1f} GB")
```

```
        # Small model for 1g.10gb instance
        model = nn.Sequential(
            nn.Linear(512, 256),
            nn.ReLU(),
            nn.Linear(256, 10)
        ).cuda()

        print(f"Model parameters: {sum(p.numel() for p in model.parameters())}")

        # Inference loop
        for i in range(200):
            with torch.no_grad():
                # Small batch for 1g.10gb
                x = torch.randn(32, 512).cuda()
                output = model(x)
                prediction = torch.argmax(output, dim=1)

                if i % 20 == 0:
                    print(f"Small Inference - Batch {i}, Predictions:
 {prediction[:5].tolist()}, GPU Memory: {torch.cuda.memory_allocated()/1e9:.2f}GB")
            time.sleep(2)

        print("Small inference completed on 1g.10gb MIG instance")

---
# Pod 1: Large training workload (3g.40gb)
apiVersion: v1
kind: Pod
metadata:
  name: mig-large-training-pod
  namespace: mig-gpu
  labels:
    app: mig-large-training
    workload-type: training
spec:
  restartPolicy: Never
  containers:
  - name: large-training-container
    image: nvcr.io/nvidia/pytorch:25.04-py3
    command: ["python", "/scripts/large-training-script.py"]
    volumeMounts:
    - name: script-volume
      mountPath: /scripts
```

```
      readOnly: true
    resources:
      claims:
      - name: mig-large-claim
  resourceClaims:
  - name: mig-large-claim
    resourceClaimTemplateName: mig-large-template
  nodeSelector:
    node.kubernetes.io/instance-type: p4de.24xlarge
    nvidia.com/gpu.present: "true"
  tolerations:
  - key: nvidia.com/gpu
    operator: Exists
    effect: NoSchedule
  volumes:
  - name: script-volume
    configMap:
      name: mig-scripts-configmap
      defaultMode: 0755

---
# Pod 2: Medium training workload (2g.20gb) - can run on SAME GPU as Pod 1
apiVersion: v1
kind: Pod
metadata:
  name: mig-medium-training-pod
  namespace: mig-gpu
  labels:
    app: mig-medium-training
    workload-type: training
spec:
  restartPolicy: Never
  containers:
  - name: medium-training-container
    image: nvcr.io/nvidia/pytorch:25.04-py3
    command: ["python", "/scripts/medium-training-script.py"]
    volumeMounts:
    - name: script-volume
      mountPath: /scripts
      readOnly: true
    resources:
      claims:
      - name: mig-medium-claim
  resourceClaims:
```

```
  - name: mig-medium-claim
    resourceClaimTemplateName: mig-medium-template
  nodeSelector:
    node.kubernetes.io/instance-type: p4de.24xlarge
    nvidia.com/gpu.present: "true"
  tolerations:
  - key: nvidia.com/gpu
    operator: Exists
    effect: NoSchedule
  volumes:
  - name: script-volume
    configMap:
      name: mig-scripts-configmap
      defaultMode: 0755

---
# Pod 3: Small inference workload (1g.10gb) - can run on SAME GPU as Pod 1 & 2
apiVersion: v1
kind: Pod
metadata:
  name: mig-small-inference-pod
  namespace: mig-gpu
  labels:
    app: mig-small-inference
    workload-type: inference
spec:
  restartPolicy: Never
  containers:
  - name: small-inference-container
    image: nvcr.io/nvidia/pytorch:25.04-py3
    command: ["python", "/scripts/small-inference-script.py"]
    volumeMounts:
    - name: script-volume
      mountPath: /scripts
      readOnly: true
    resources:
      claims:
      - name: mig-small-claim
  resourceClaims:
  - name: mig-small-claim
    resourceClaimTemplateName: mig-small-template
  nodeSelector:
    node.kubernetes.io/instance-type: p4de.24xlarge
    nvidia.com/gpu.present: "true"
```

```
  tolerations:
  - key: nvidia.com/gpu
    operator: Exists
    effect: NoSchedule
  volumes:
  - name: script-volume
    configMap:
      name: mig-scripts-configmap
      defaultMode: 0755
```

5. Applica questa specifica, che dovrebbe implementare tre Pod:

```
kubctl apply -f mig-pod.yaml
```

   Questi Pod devono essere programmati dal driver DRA.

6. Controllate i registri del Pod del driver DRA e vedrete un risultato simile a questo:

```
I0717 21:50:22.925811 1 driver.go:87] NodePrepareResource is called: number of
 claims: 1
I0717 21:50:22.932499 1 driver.go:129] Returning newly prepared devices
 for claim '933e9c72-6fd6-49c5-933c-a896407dc6d1': [&Device{RequestNames:
[mig-large],PoolName:ip-100-64-173-145.ec2.internal,DeviceName:gpu-0-
mig-9-4-4,CDIDeviceIDs:[k8s.gpu.nvidia.com/device=**gpu-0-mig-9-4-4**],}]
I0717 21:50:23.186472 1 driver.go:87] NodePrepareResource is called: number of
 claims: 1
I0717 21:50:23.191226 1 driver.go:129] Returning newly prepared devices
 for claim '61e5ddd2-8c2e-4c19-93ae-d317fecb44a4': [&Device{RequestNames:
[mig-medium],PoolName:ip-100-64-173-145.ec2.internal,DeviceName:gpu-2-
mig-14-0-2,CDIDeviceIDs:[k8s.gpu.nvidia.com/device=**gpu-2-mig-14-0-2**],}]
I0717 21:50:23.450024 1 driver.go:87] NodePrepareResource is called: number of
 claims: 1
I0717 21:50:23.455991 1 driver.go:129] Returning newly prepared devices
 for claim '1eda9b2c-2ea6-401e-96d0-90e9b3c111b5': [&Device{RequestNames:
[mig-small],PoolName:ip-100-64-173-145.ec2.internal,DeviceName:gpu-1-
mig-19-2-1,CDIDeviceIDs:[k8s.gpu.nvidia.com/device=**gpu-1-mig-19-2-1**],}]
```

7. Verifica il `resourceclaims` per vedere lo stato del Pod:

```
kubectl get resourceclaims -n mig-gpu -w
```

   Di seguito è riportato un output di esempio:

```
NAME                                                    STATE                AGE
mig-large-training-pod-mig-large-claim-6dpn8     pending              0s
mig-large-training-pod-mig-large-claim-6dpn8     pending              0s
mig-large-training-pod-mig-large-claim-6dpn8     allocated,reserved   0s
mig-medium-training-pod-mig-medium-claim-bk596   pending              0s
mig-medium-training-pod-mig-medium-claim-bk596   pending              0s
mig-medium-training-pod-mig-medium-claim-bk596   allocated,reserved   0s
mig-small-inference-pod-mig-small-claim-d2t58    pending              0s
mig-small-inference-pod-mig-small-claim-d2t58    pending              0s
mig-small-inference-pod-mig-small-claim-d2t58    allocated,reserved   0s
```

Come puoi vedere, tutti i Pod sono stati spostati dalla posizione in sospeso al `allocated,reserved` driver DRA.

8. Esegui `nvidia-smi` dal nodo. Noterai che sono in esecuzione tre processori Python:

```
root@ip-100-64-173-145 bin]# nvidia-smi
+-----------------------------------------------------------------------------------------
+
| NVIDIA-SMI 570.158.01 Driver Version: 570.158.01 CUDA Version: 12.8 |
|-----------------------------------------+------------------------
+----------------------+
| GPU Name Persistence-M | Bus-Id Disp.A | Volatile Uncorr. ECC |
| Fan Temp Perf Pwr:Usage/Cap | Memory-Usage | GPU-Util Compute M. |
| | | MIG M. |
|=========================================+========================
+======================|
| 0 NVIDIA A100-SXM4-80GB On | 00000000:10:1C.0 Off | On |
| N/A 63C P0 127W / 400W | 569MiB / 81920MiB | N/A Default |
| | | Enabled |
+-----------------------------------------+------------------------
+----------------------+
| 1 NVIDIA A100-SXM4-80GB On | 00000000:10:1D.0 Off | On |
| N/A 56C P0 121W / 400W | 374MiB / 81920MiB | N/A Default |
| | | Enabled |
+-----------------------------------------+------------------------
+----------------------+
| 2 NVIDIA A100-SXM4-80GB On | 00000000:20:1C.0 Off | On |
| N/A 63C P0 128W / 400W | 467MiB / 81920MiB | N/A Default |
| | | Enabled |
+-----------------------------------------+------------------------
+----------------------+
```

```
| 3 NVIDIA A100-SXM4-80GB On | 00000000:20:1D.0 Off | On |
| N/A 57C P0 118W / 400W | 249MiB / 81920MiB | N/A Default |
| | | Enabled |
+-----------------------------------------+-----------------------
+----------------------+
| 4 NVIDIA A100-SXM4-80GB On | 00000000:90:1C.0 Off | 0 |
| N/A 51C P0 77W / 400W | 0MiB / 81920MiB | 0% Default |
| | | Disabled |
+-----------------------------------------+-----------------------
+----------------------+
| 5 NVIDIA A100-SXM4-80GB On | 00000000:90:1D.0 Off | 0 |
| N/A 46C P0 69W / 400W | 0MiB / 81920MiB | 0% Default |
| | | Disabled |
+-----------------------------------------+-----------------------
+----------------------+
| 6 NVIDIA A100-SXM4-80GB On | 00000000:A0:1C.0 Off | 0 |
| N/A 52C P0 74W / 400W | 0MiB / 81920MiB | 0% Default |
| | | Disabled |
+-----------------------------------------+-----------------------
+----------------------+
| 7 NVIDIA A100-SXM4-80GB On | 00000000:A0:1D.0 Off | 0 |
| N/A 47C P0 72W / 400W | 0MiB / 81920MiB | 0% Default |
| | | Disabled |
+-----------------------------------------+-----------------------
+----------------------+



+-----------------------------------------------------------------------------------------------------------
+
| MIG devices: |
+------------------+--------------------------------+-----------
+----------------------+
| GPU GI CI MIG | Memory-Usage | Vol| Shared |
| ID ID Dev | BAR1-Usage | SM Unc| CE ENC DEC OFA JPG |
| | | ECC| |
|==================+================================+===========
+======================|
| 0 2 0 0 | 428MiB / 40192MiB | 42 0 | 3 0 2 0 0 |
| | 2MiB / 32767MiB | | |
+------------------+--------------------------------+-----------
+----------------------+
| 0 3 0 1 | 71MiB / 19968MiB | 28 0 | 2 0 1 0 0 |
| | 0MiB / 16383MiB | | |
```

```
+------------------+-----------------------------------+-----------
+-----------------------+
| 0 9 0 2 | 36MiB / 9728MiB | 14 0 | 1 0 0 0 0 |
| | 0MiB / 8191MiB | | |
+------------------+-----------------------------------+-----------
+-----------------------+
| 0 10 0 3 | 36MiB / 9728MiB | 14 0 | 1 0 0 0 0 |
| | 0MiB / 8191MiB | | |
+------------------+-----------------------------------+-----------
+-----------------------+
| 1 1 0 0 | 107MiB / 40192MiB | 42 0 | 3 0 2 0 0 |
| | 0MiB / 32767MiB | | |
+------------------+-----------------------------------+-----------
+-----------------------+
| 1 5 0 1 | 71MiB / 19968MiB | 28 0 | 2 0 1 0 0 |
| | 0MiB / 16383MiB | | |
+------------------+-----------------------------------+-----------
+-----------------------+
| 1 13 0 2 | 161MiB / 9728MiB | 14 0 | 1 0 0 0 0 |
| | 2MiB / 8191MiB | | |
+------------------+-----------------------------------+-----------
+-----------------------+
| 1 14 0 3 | 36MiB / 9728MiB | 14 0 | 1 0 0 0 0 |
| | 0MiB / 8191MiB | | |
+------------------+-----------------------------------+-----------
+-----------------------+
| 2 1 0 0 | 107MiB / 40192MiB | 42 0 | 3 0 2 0 0 |
| | 0MiB / 32767MiB | | |
+------------------+-----------------------------------+-----------
+-----------------------+
| 2 5 0 1 | 289MiB / 19968MiB | 28 0 | 2 0 1 0 0 |
| | 2MiB / 16383MiB | | |
+------------------+-----------------------------------+-----------
+-----------------------+
| 2 13 0 2 | 36MiB / 9728MiB | 14 0 | 1 0 0 0 0 |
| | 0MiB / 8191MiB | | |
+------------------+-----------------------------------+-----------
+-----------------------+
| 2 14 0 3 | 36MiB / 9728MiB | 14 0 | 1 0 0 0 0 |
| | 0MiB / 8191MiB | | |
+------------------+-----------------------------------+-----------
+-----------------------+
| 3 1 0 0 | 107MiB / 40192MiB | 42 0 | 3 0 2 0 0 |
| | 0MiB / 32767MiB | | |
```

```
+-----------------+-----------------------------------+-----------
+----------------------+
| 3 5 0 1 | 71MiB / 19968MiB | 28 0 | 2 0 1 0 0 |
| | 0MiB / 16383MiB | | |
+-----------------+-----------------------------------+-----------
+----------------------+
| 3 13 0 2 | 36MiB / 9728MiB | 14 0 | 1 0 0 0 0 |
| | 0MiB / 8191MiB | | |
+-----------------+-----------------------------------+-----------
+----------------------+
| 3 14 0 3 | 36MiB / 9728MiB | 14 0 | 1 0 0 0 0 |
| | 0MiB / 8191MiB | | |
+-----------------+-----------------------------------+-----------
+----------------------+



+-----------------------------------------------------------------------------------------
+
| Processes: |
| GPU GI CI PID Type Process name GPU Memory |
| ID ID Usage |
|
=========================================================================================|
**| 0 2 0 64080 C python 312MiB |
| 1 13 0 64085 C python 118MiB |
| 2 5 0 64073 C python 210MiB |**
+-----------------------------------------------------------------------------------------
+
```

Ottimizza i carichi di lavoro della GPU con IMEX utilizzando istanze P6e GB200

IMEX (Internode Memory Exchange) consente una comunicazione coerente in termini di memoria tra i nodi per la formazione distribuita su NVIDIA. GB200 UltraServers

Effettuare le seguenti operazioni.

1. Definite un programma `ComputeDomain` per l'addestramento multinodo con un file denominato`imex-compute-domain.yaml`:

```
apiVersion: resource.nvidia.com/v1beta1
kind: ComputeDomain
metadata:
```

```
  name: distributed-training-domain
  namespace: default
spec:
  numNodes: 2
  channel:
    resourceClaimTemplate:
      name: imex-channel-template
```

2. Definisci un Pod utilizzando i canali IMEX con un file denominato: `imex-pod.yaml`

```
apiVersion: v1
kind: Pod
metadata:
  name: imex-distributed-training
  namespace: default
  labels:
    app: imex-training
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: nvidia.com/gpu.clique
            operator: Exists
  containers:
  - name: distributed-training
    image: nvcr.io/nvidia/pytorch:25.04-py3
    command: ["bash", "-c"]
    args:
    - |
      echo "=== IMEX Channel Verification ==="
      ls -la /dev/nvidia-caps-imex-channels/
      echo ""

      echo "=== GPU Information ==="
      nvidia-smi
      echo ""

      echo "=== NCCL Test (if available) ==="
      python -c "
      import torch
      import torch.distributed as dist
      import os
```

```
    print(f'CUDA available: {torch.cuda.is_available()}')
    print(f'CUDA device count: {torch.cuda.device_count()}')

    if torch.cuda.is_available():
        for i in range(torch.cuda.device_count()):
            print(f'GPU {i}: {torch.cuda.get_device_name(i)}')

    # Check for IMEX environment variables
    imex_vars = [k for k in os.environ.keys() if 'IMEX' in k or 'NVLINK' in k]
    if imex_vars:
        print('IMEX Environment Variables:')
        for var in imex_vars:
            print(f'  {var}={os.environ[var]}')

    print('IMEX channel verification completed')
    "

    # Keep container running for inspection
    sleep 3600
  resources:
    claims:
    - name: imex-channel-0
    - name: imex-channel-1
resourceClaims:
- name: imex-channel-0
  resourceClaimTemplateName: imex-channel-template
- name: imex-channel-1
  resourceClaimTemplateName: imex-channel-template
tolerations:
- key: nvidia.com/gpu
  operator: Exists
  effect: NoSchedule
```

> **Note**
>
> Ciò richiede istanze GB200 P6e.

3. Implementa IMEX applicando i modelli and: `ComputeDomain`

```
kubectl apply -f imex-claim-template.yaml
kubectl apply -f imex-compute-domain.yaml
```

```
kubectl apply -f imex-pod.yaml
```

4. Controlla lo stato. `ComputeDomain`

```
kubectl get computedomain distributed-training-domain
```

5. Monitora la distribuzione del demone IMEX.

```
kubectl get pods -n nvidia-dra-driver -l resource.nvidia.com/computeDomain
```

6. Controlla i canali IMEX nel Pod:

```
kubectl exec imex-distributed-training -- ls -la /dev/nvidia-caps-imex-channels/
```

7. Visualizza i log del Pod:

```
kubectl logs imex-distributed-training
```

   Di seguito è riportato un esempio di output previsto:

```
=== IMEX Channel Verification ===
total 0
drwxr-xr-x. 2 root root 80 Jul 8 10:45 .
drwxr-xr-x. 6 root root 380 Jul 8 10:45 ..
crw-rw-rw-. 1 root root 241, 0 Jul 8 10:45 channel0
crw-rw-rw-. 1 root root 241, 1 Jul 8 10:45 channel1
```

Per ulteriori informazioni, vedere l'esempio di NVIDIA su GitHub.

Rete

> **Tip**
>
> Esplora le best practice tramite i workshop Amazon EKS.

Prendi in considerazione una maggiore larghezza di banda di rete o un adattatore Elastic Fabric per applicazioni con un'elevata comunicazione tra i nodi

Per carichi di lavoro di formazione distribuiti su Amazon EKS con elevate esigenze di comunicazione tra nodi, prendi in considerazione la possibilità di selezionare istanze con una larghezza di banda di rete maggiore o Elastic Fabric Adapter (EFA). Prestazioni di rete insufficienti possono ostacolare il trasferimento dei dati, rallentando le attività di machine learning come l'addestramento distribuito su più GPU. Tieni presente che i carichi di lavoro di inferenza in genere non hanno un'elevata comunicazione tra i nodi.

Esempio

Ad esempio, utilizzando Karpenter:

```
apiVersion: v1
kind: Pod
metadata:
  name: ml-workload
spec:
  nodeSelector:
    karpenter.k8s.aws/instance-network-bandwidth: "100000"  # 100 Gbps in Mbps
    node.kubernetes.io/instance-type: p5.48xlarge  # EFA-enabled instance
  containers:
  - name: training-job
    image: `763104351884.dkr.ecr.us-west-2.amazonaws.com/pytorch-inference:2.6.0-gpu-
py312-cu124-ubuntu22.04-ec2-v1.6`
    resources:
      limits:
        vpc.amazonaws.com/efa: 1  # Requires EFA device plugin
```

Assicurati che strumenti come MPI e NCCL siano installati nell'immagine del tuo container per sfruttare EFA per i lavori di formazione.

Aumenta il numero di indirizzi IP disponibili per consentire tempi di avvio dei pod più rapidi

In EKS, ogni pod necessita di un indirizzo IP dal blocco CIDR VPC. Man mano che il cluster si espande con più nodi e pod, si rischia l'esaurimento degli indirizzi IP o il rallentamento delle

prestazioni, ma l'abilitazione della delega dei prefissi può mitigare questi problemi preallocando gli intervalli IP e riducendo le chiamate API EC2, con conseguenti tempi di avvio dei pod più rapidi e una migliore scalabilità.

L'abilitazione della delega dei prefissi dopo la creazione del cluster consente alla VPC Container Network Interface (CNI) di assegnare prefissi IP (/28, ciascuno con 16 indirizzi IP) alle interfacce di rete sulle istanze EC2. Ciò significa che ogni nodo può supportare più pod, riducendo il rischio di carenze di IP. Ad esempio, in un'`c5.4xlarge`istanza, puoi supportare fino a 110 pod con delega di prefisso.

Sebbene la delega dei prefissi sia fondamentale per ottimizzare l'utilizzo dell'IP in ambienti con molti pod di piccole dimensioni, i AI/ML carichi di lavoro spesso utilizzano un numero inferiore di pod più grandi (ad esempio, un pod per GPU). L'abilitazione della delega dei prefissi consente a VPC CNI di preallocare un prefisso per un avvio più rapido del pod mantenendo un pool caldo. Ciò significa che gli indirizzi IP sono immediatamente disponibili, riducendo il tempo necessario per l'inizializzazione dei pod rispetto all'allocazione su richiesta in modalità senza prefisso. In questi casi, il risparmio di IP derivante dall'abilitazione della delega dei prefissi offre vantaggi in termini di prestazioni per i carichi di lavoro. AI/ML Riducendo il numero di chiamate API EC2 necessarie per la configurazione degli indirizzi IP e la preallocazione degli intervalli IP, l'utilizzo della delega di prefissi consente tempi di avvio dei pod più rapidi, il che è particolarmente utile per scalare rapidamente i carichi di lavoro. AI/ML

Per abilitare la delega dei prefissi:

```
kubectl set env daemonset/aws-node -n kube-system ENABLE_PREFIX_DELEGATION=true
```

Garantisci una pianificazione adeguata delle sottoreti VPC per evitare l'esaurimento degli indirizzi IP, specialmente nelle implementazioni di grandi dimensioni, e gestisci i blocchi CIDR per evitare sovrapposizioni. VPCs Per ulteriori informazioni, consulta Ottimizzazione dell'utilizzo degli indirizzi IP e Assegnazione di più indirizzi IP ai nodi Amazon EKS con prefissi.

Sicurezza

> **Tip**
>
> Esplora le best practice tramite i workshop Amazon EKS.

Sicurezza e conformità

Prendi in considerazione S3 con KMS per uno storage conforme alla crittografia

Salvo diversa indicazione, tutti i bucket S3 utilizzano SSE-S3 per impostazione predefinita per crittografare gli oggetti inattivi. Tuttavia, puoi scegliere di configurare i bucket per utilizzare invece la crittografia lato server con le chiavi AWS Key Management Service (AWS KMS) (SSE-KMS). I controlli della sicurezza in AWS KMS possono essere d'aiuto per soddisfare i requisiti di conformità correlati alla crittografia. Puoi utilizzare queste chiavi KMS per proteggere i dati nei bucket Amazon S3. Quando utilizzi la crittografia SSE-KMS con un bucket S3, le chiavi AWS KMS devono trovarsi nella stessa regione del bucket.

Configura i tuoi bucket generici per utilizzare S3 Bucket Keys per SSE-KMS, per ridurre i costi delle richieste AWS KMS fino al 99 percento diminuendo il traffico delle richieste da Amazon S3 ad AWS KMS. Le S3 Bucket Keys sono sempre abilitate e funzionano in un bucket di directory e non possono essere disabilitate. `GET` `PUT`

Tieni presente che Amazon S3 Express One Zone utilizza un tipo specifico di bucket chiamato bucket di directory S3. I bucket di directory sono destinati esclusivamente alla classe di storage S3 Express One Zone e consentono un accesso ad alte prestazioni e bassa latenza. Per configurare la crittografia dei bucket predefinita su un bucket di directory S3, utilizza la CLI di AWS e specifica l'ID della chiave KMS o l'ARN, non l'alias, come nell'esempio seguente:

Example

```
aws s3api put-bucket-encryption --bucket my-directory-bucket --server-side-encryption-
configuration \
   '{"Rules": [{"ApplyServerSideEncryptionByDefault": {"SSEAlgorithm": "aws:kms",
 "KMSMasterKeyID": "1234abcd-12ab-34cd-56ef-1234567890ab"}}]}'
```

Assicurati che il ruolo IAM del tuo pod EKS disponga delle autorizzazioni KMS (ad esempio) per accedere agli oggetti crittografati. `kms:Decrypt` Provalo in un ambiente di staging caricando un modello di esempio nel bucket, montandolo in un pod (ad esempio, tramite il driver CSI Mountpoint S3) e verificando che il pod sia in grado di leggere i dati crittografati senza errori. Registri di controllo tramite AWS CloudTrail per confermare la conformità ai requisiti di crittografia. Consulta la documentazione KMS per i dettagli di configurazione e la gestione delle chiavi.

Archiviazione

> **Tip**
>
> Esplora le best practice tramite i workshop Amazon EKS.

Gestione e archiviazione dei dati

Implementa i modelli AI sui pod utilizzando un driver CSI

I carichi di lavoro AI/ML richiedono spesso l'accesso ad artefatti di modelli di grandi dimensioni (ad esempio pesi addestrati, configurazioni) e i pod necessitano di un modo affidabile e scalabile per accedervi senza incorporarli nelle immagini dei container, il che può aumentare le dimensioni delle immagini e i tempi di recupero del registro dei container. Per ridurre il sovraccarico operativo legato alla gestione dei montaggi di volume, consigliamo di implementare modelli AI sui pod montando i servizi di storage Amazon (ad esempio, S3, FSx for Lustre, per FSx OpenZFS, EFS) come Persistent Volumes () utilizzando i rispettivi driver CSI. PVs Per i dettagli sull'implementazione, consulta gli argomenti successivi in questa sezione.

Ottimizza lo storage per le cache dei modelli ML su EKS

L'utilizzo di una soluzione di archiviazione ottimale è fondamentale per ridurre al minimo la latenza di avvio dei pod e delle applicazioni, ridurre l'utilizzo della memoria, ottenere i livelli di prestazioni desiderati per accelerare i carichi di lavoro e garantire la scalabilità dei carichi di lavoro ML. I carichi di lavoro ML si basano spesso su file modello (pesi), che possono essere di grandi dimensioni e richiedere l'accesso condiviso ai dati tra pod o nodi. La scelta della soluzione di storage ottimale dipende dalle caratteristiche del carico di lavoro, come l'efficienza a nodo singolo, l'accesso a più nodi, i requisiti di latenza, i vincoli di costo e anche i requisiti di integrazione dei dati (ad esempio con un repository di dati Amazon S3). Ti consigliamo di confrontare diverse soluzioni di storage con i tuoi carichi di lavoro per capire quale soddisfa i tuoi requisiti e abbiamo fornito le seguenti opzioni per aiutarti a valutare in base ai requisiti del tuo carico di lavoro.

Il driver EKS CSI supporta i seguenti servizi di storage AWS, ognuno con il proprio driver CSI e presenta i propri punti di forza per i flussi di lavoro AI e ML:

- Mountpoint per Amazon S3
- Amazon FSx per Lustre

- Amazon FSx per OpenZFS
- Amazon EFS
- Amazon EBS

La scelta del servizio di storage AWS dipende dall'architettura di distribuzione, dalla scalabilità, dai requisiti di prestazioni e dalla strategia di costo. I driver Storage CSI devono essere installati sul cluster EKS, il che consente al driver CSI di creare e gestire volumi persistenti (PV) al di fuori del ciclo di vita di un Pod. Utilizzando il driver CSI, puoi creare definizioni PV dei servizi di storage AWS supportati come risorse del cluster EKS. I pod possono quindi accedere a questi volumi di storage per i rispettivi volumi di dati creando un Persistent Volume Claim (PVC) per il PV. A seconda del servizio di storage AWS e dello scenario di implementazione, un singolo PVC (e il relativo PV associato) può essere collegato a più Pod per un carico di lavoro. Ad esempio, per l'allenamento ML, i dati di allenamento condivisi vengono archiviati su un PV e vi si accede da più Pod; per l'inferenza online in tempo reale, i modelli LLM vengono memorizzati nella cache su un PV e vi si accede da più Pod. Di seguito sono riportati esempi di file YAML PV e PVC per i servizi di storage AWS per aiutarti a iniziare.

Monitoraggio delle prestazioni Le scarse prestazioni del disco possono ritardare la lettura delle immagini dei container, aumentare la latenza di avvio dei pod e ridurre la velocità di inferenza o di addestramento. Usa Amazon CloudWatch per monitorare i parametri prestazionali per i tuoi servizi di storage AWS. Quando identifichi problemi di prestazioni, modifica i parametri di configurazione dello storage per ottimizzare le prestazioni.

Scenario: carico di lavoro di più istanze GPU

Amazon FSx for Lustre: negli scenari in cui disponi di più istanze di calcolo GPU EC2 con carichi di lavoro dinamici sensibili alla latenza e con throughput a larghezza di banda elevata, come formazione distribuita e gestione di modelli, e richiedi l'integrazione nativa del repository di dati Amazon S3, ti consigliamo Amazon for Lustre. FSx FSx for Lustre fornisce un file system parallelo ad alte prestazioni completamente gestito, progettato per carichi di lavoro ad alta intensità di calcolo come HPC (High Performance Computing) e Machine Learning.

È possibile installare il driver CSI FSx for Lustre per montare i FSx filesystem su EKS come volume persistente (PV), quindi implementarlo FSx per il file system Lustre come cache autonoma ad alte prestazioni o come file system collegato a S3 per fungere da cache ad alte prestazioni per i dati S3, fornendo un throughput rapido I/O e elevato per l'accesso ai dati tra le istanze di calcolo della GPU. FSx for Lustre può essere distribuito con opzioni di archiviazione Scratch-SSD o Persistent-SSD:

- Storage Scratch-SSD: consigliato per carichi di lavoro temporanei o di breve durata (ore), con capacità di throughput fissa per TiB fornita.
- Storage SSD persistente: consigliato per carichi di lavoro critici e di lunga durata che richiedono il massimo livello di disponibilità, ad esempio simulazioni HPC, analisi dei big data o formazione sul Machine Learning. Con lo storage SSD persistente, puoi configurare sia la capacità di archiviazione che la capacità di throughput (per TiB) richieste.

Considerazioni sulle prestazioni:

- Pod amministrativo da gestire FSx per il file system Lustre: configura un pod «amministrativo» con il client lustre installato e il file system FSx montato. Ciò consentirà un punto di accesso per consentire la regolazione fine del file system FSx e anche in situazioni in cui è necessario preriscaldare il file system FSx con i dati di addestramento ML o i modelli LLM prima di avviare le istanze di calcolo GPU. Ciò è particolarmente importante se l'architettura utilizza istanze Amazon GPU/compute EC2 basate su Spot, in cui è possibile utilizzare il Pod amministrativo per «riscaldare» o «precaricare» i dati desiderati nel file system FSx, in modo che siano pronti per essere elaborati quando si eseguono le istanze Amazon EC2 basate su Spot.
- Elastic Fabric Adapter (EFA): i tipi di implementazione dello storage SSD persistente supportano Elastic Fabric Adapter (EFA), dove l'utilizzo di EFA è ideale per carichi di lavoro basati su GPU ad alte prestazioni e velocità effettiva. Tieni presente che FSx for Lustre supporta NVIDIA GPUDirect Storage (GDS), dove GDS è una tecnologia che crea un percorso dati diretto tra lo storage locale o remoto e la memoria GPU, per consentire un accesso più rapido ai dati.
- Compressione: abilita la compressione dei dati sul file system se hai tipi di file che possono essere compressi. Ciò può contribuire ad aumentare le prestazioni in quanto la compressione dei dati riduce la quantità di dati trasferiti tra FSx i file server Lustre e lo storage.
- Configurazione dello striping del file system Lustre:
  - Striping dei dati: consente FSx a Luster di distribuire i dati di un file su più Object Storage Targets (OSTs) all'interno di un file system Lustre, massimizza l'accesso parallelo e il throughput, in particolare per i lavori di formazione ML su larga scala.
  - Striping del file system autonomo: per impostazione predefinita, viene creata automaticamente una configurazione di striping Lustre a 4 componenti tramite la funzionalità Progressive file layouts (PFL) di for Lustre. FSx Nella maggior parte degli scenari non è necessario aggiornare il conto/dimensione delle strisce PFL Lustre predefinito. Se è necessario regolare lo striping dei dati Lustre, è possibile regolare manualmente lo striping Lustre facendo riferimento ai parametri di striping di un file system for Lustre. FSx

- File system collegato a S3: i file importati nel FSx file system utilizzando l'integrazione nativa di Amazon S3 (Data Repository Association o DRA) non utilizzano il layout PFL predefinito, ma utilizzano invece il layout nel parametro del file system. `ImportedFileChunkSize` I file importati da S3 più grandi di quelli `ImportedFileChunkSize` verranno archiviati su più file OSTs con un numero di stripe basato sul valore definito (impostazione predefinita 1GiB). `ImportedFileChunkSize` Se hai file di grandi dimensioni, ti consigliamo di impostare questo parametro su un valore più alto.
- Posizionamento: implementa un file system FSx for Lustre nella stessa zona di disponibilità dei nodi di elaborazione o GPU per consentire l'accesso ai dati con la latenza più bassa ed evitare modelli di accesso incrociati tra zone di disponibilità. Se disponi di più nodi GPU situati in zone di disponibilità diverse, ti consigliamo di implementare un FSx file system in ciascuna zona di disponibilità per un accesso ai dati a bassa latenza.

Esempio

Definizione del volume persistente (PV) per un file system FSx for Lustre, utilizzando il provisioning statico (laddove l' FSx istanza è già stato fornito).

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: fsx-pv
spec:
  capacity:
    storage: 1200Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteMany
  mountOptions:
    - flock
  persistentVolumeReclaimPolicy: Recycle
  csi:
    driver: fsx.csi.aws.com
    volumeHandle: [FileSystemId of FSx instance]
    volumeAttributes:
      dnsname: [DNSName of FSx instance]
      mountname: [MountName of FSx instance]
```

Esempio

Definizione di Persistent Volume Claim per PV denominata: `fsx-pv`

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: fsx-claim
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: ""
  resources:
    requests:
      storage: 1200Gi
  volumeName: fsx-pv
```

Esempio

Configura un pod per utilizzare un Persistent Volume Claim di`fsx-claim`:

```
apiVersion: v1
kind: Pod
metadata:
  name: fsx-app
spec:
  containers:
  - name: app
    image: amazonlinux:2023
    command: ["/bin/sh"]
    volumeMounts:
    - name: persistent-storage
      mountPath: /data
  volumes:
  - name: persistent-storage
    persistentVolumeClaim:
      claimName: fsx-claim
```

Per esempi completi, vedere gli esempi FSx di Lustre Driver in GitHub. Monitora i parametri prestazionali di Amazon FSx for Lustre utilizzando Amazon. CloudWatch Quando vengono identificati problemi di prestazioni, modifica i parametri di configurazione in base alle esigenze.

Scenario: carico di lavoro a singola istanza GPU

Mountpoint per Amazon S3 con driver CSI: puoi montare un bucket S3 come volume nei tuoi pod utilizzando il driver CSI Mountpoint per Amazon S3. Questo metodo consente un controllo granulare degli accessi su quali Pod possono accedere a bucket S3 specifici. Ogni pod dispone di una propria istanza di mountpoint e di una cache locale (5-10 GB), che isolano le prestazioni di caricamento e lettura del modello tra i pod. Questa configurazione supporta l'autenticazione a livello di pod con IAM Roles for Service Accounts (IRSA) e il controllo delle versioni dei modelli indipendenti per diversi modelli o clienti. Il compromesso è l'aumento dell'utilizzo della memoria e del traffico API, poiché ogni pod invia chiamate API S3 e mantiene la propria cache.

Esempio: esempio parziale di implementazione di un Pod (YAML) con driver CSI:

```
# CSI driver dynamically mounts the S3 bucket for each pod

volumes:
  - name: s3-mount
    csi:
      driver: s3.csi.aws.com
      volumeAttributes:
        bucketName: your-s3-bucket-name
        mountOptions: "--allow-delete"  # Optional
        region: us-west-2

containers:
  - name: inference
    image: your-inference-image
    volumeMounts:
      - mountPath: /models
        name: s3-mount
volumeMounts:
  - name: model-cache
    mountPath: /models
volumes:
  - name: model-cache
    hostPath:
      path: /mnt/s3-model-cache
```

Considerazioni sulle prestazioni:

- Memorizzazione nella cache dei dati: Mountpoint per S3 può memorizzare nella cache i contenuti per ridurre i costi e migliorare le prestazioni in caso di letture ripetute sullo stesso file. Fai

riferimento alla configurazione della memorizzazione nella cache per le opzioni e i parametri di memorizzazione nella cache.

- Dimensione parziale dell'oggetto: quando memorizzi e accedi a file di dimensioni superiori a 72 GB, consulta Configurazione delle prestazioni di Mountpoint per capire come configurare i parametri della `--write-part-size` riga di comando `--read-part-size` e il profilo dei dati per soddisfare i requisiti del profilo dei dati e del carico di lavoro.
- La cache condivisa è progettata per oggetti di dimensioni fino a 1 MB. Non supporta oggetti di grandi dimensioni. Utilizzate l'opzione Cache locale per memorizzare nella cache gli oggetti nei volumi EBS NVMe o EBS sul nodo EKS.
- Costi per le richieste API: quando si eseguono un numero elevato di operazioni sui file con Mountpoint for S3, i costi delle richieste API possono diventare una parte dei costi di archiviazione. Per mitigare questo problema, se non è richiesta una forte coerenza, abilita sempre la memorizzazione nella cache dei metadati e imposta il `metadata-ttl` periodo per ridurre il numero di operazioni API su S3.

Per ulteriori dettagli, consulta il driver CSI Mountpoint for Amazon S3 nella documentazione ufficiale di Amazon EKS. Ti consigliamo di monitorare i parametri prestazionali di Amazon S3 con i parametri di CloudWatch Amazon in caso di colli di bottiglia e di modificare la configurazione dove necessario.

Amazon FSx per lo storage condiviso persistente OpenZFS

Per scenari che coinvolgono più istanze di calcolo GPU EC2 con carichi di lavoro sensibili alla latenza che richiedono alta disponibilità, alte prestazioni, sensibilità ai costi e implementazioni di più pod per diverse applicazioni, consigliamo Amazon per OpenZFS. FSx Alcuni esempi di carichi di lavoro includono l'inferenza in tempo reale, l'apprendimento per rinforzo e l'addestramento di reti antagoniste generative. FSx per OpenZFS è particolarmente utile per i carichi di lavoro che richiedono un accesso ad alte prestazioni a una struttura di directory mirata con file di piccole dimensioni che utilizzano modelli di accesso ai dati IO di piccole dimensioni. Inoltre, FSx per OpenZFS offre la flessibilità necessaria per scalare le prestazioni indipendentemente dalla capacità di archiviazione, aiutando a raggiungere un'efficienza ottimale in termini di costi adattando le dimensioni dello storage alle esigenze effettive e mantenendo i livelli di prestazioni richiesti

Il driver CSI nativo FSx per OpenZFS consente la creazione di più PVCs file system in un unico file creando più volumi. Ciò riduce il sovraccarico di gestione e massimizza l'utilizzo del throughput e degli IOPS del file system attraverso implementazioni consolidate di pod di applicazioni su un singolo file system. Inoltre, include funzionalità aziendali come istantanee a copia zero, cloni a copia zero e quote di utenti e gruppi, che possono essere fornite dinamicamente tramite il driver CSI.

FSx for OpenZFS supporta tre diversi tipi di distribuzione al momento della creazione:

- Single-AZ: opzione dal costo più basso con latenze inferiori al millisecondo, ma non offre un'elevata disponibilità a livello di file system o di zona di disponibilità. Consigliato per carichi di lavoro di sviluppo e test o per carichi di lavoro con elevata disponibilità a livello di applicazione.
- Single-AZ (HA): offre un'elevata disponibilità a livello di file system con latenze inferiori al millisecondo. Consigliato per carichi di lavoro dalle prestazioni più elevate che richiedono un'elevata disponibilità.
- Multi-AZ: offre un'elevata disponibilità a livello di file system e tra le zone di disponibilità. Consigliato per carichi di lavoro ad alte prestazioni che richiedono una disponibilità aggiuntiva tra le zone di disponibilità.

Considerazioni sulle prestazioni:

- Tipo di implementazione: se la disponibilità aggiuntiva tra le zone di disponibilità non è un requisito, prendi in considerazione l'utilizzo del tipo di implementazione Single-AZ (HA). Questo tipo di implementazione fornisce fino al 100% del throughput per le scritture, mantiene latenze inferiori al millisecondo e i file system Gen2 dispongono di una NVMe cache aggiuntiva per archiviare fino a terabyte di dati a cui si accede di frequente. I file system Multi-AZ forniscono fino al 75% del throughput per le scritture a una latenza maggiore per adattarsi al traffico Cross-AZ.
- Throughput e IOPS: sia il throughput che l'IOPS configurati per il file system possono essere aumentati o ridotti dopo l'implementazione. È possibile fornire fino al 10% dell'accesso ai dati memorizzati nella cacheGB/s of disk throughput providing up to 21GB/s. Gli IOPS possono essere scalati fino a 400.000 dal disco e la cache può fornire oltre 1 milione di IOPS. Si noti che la scalabilità del throughput di un file system Single-AZ causa una breve interruzione del file system in quanto non esiste un'elevata disponibilità. La scalabilità della velocità effettiva di un file system Single-AZ (HA) o Multi-AZ può essere eseguita senza interruzioni. Gli IOPS SSD possono essere scalati una volta ogni sei ore.
- Classe di archiviazione: FSx per OpenZFS supporta sia la classe di archiviazione SSD che la classe di archiviazione Intelligent-Tiering. Per i AI/ML carichi di lavoro si consiglia di utilizzare la classe di archiviazione SSD che fornisce prestazioni costanti al carico di lavoro mantenendo le CPU/GPU il più occupate possibile.
- Compressione: abilita l'algoritmo di LZ4 compressione se hai un carico di lavoro che può essere compresso. Ciò riduce la quantità di dati che ogni file consuma nella cache, permettendo di fornire più dati direttamente dalla cache come throughput di rete e IOPS per ridurre il carico sul disco SSD.

- Dimensione del record: la maggior parte dei AI/ML carichi di lavoro trarrà vantaggio dal mantenimento della dimensione di record predefinita di 128 KiB. Questo valore deve essere ridotto solo se il set di dati è costituito da file di grandi dimensioni (superiori a 10 GiB) con accesso costante a piccoli blocchi inferiore a 128 KiB dall'applicazione.

Una volta creato il file system, il servizio crea automaticamente un volume root associato. È consigliabile archiviare i dati all'interno di volumi secondari del volume root del file system. Utilizzando il driver CSI FSx for OpenZFS si crea un Persistent Volume Claim associato per creare dinamicamente il volume figlio.

Esempi:

Una definizione di Storage Class (SC) per un volume FSx for OpenZFS, utilizzata per creare un volume figlio del volume root ($ROOT_VOL_ID) su un file system esistente ed esportare il volume nel VPC CIDR ($VPC_CIDR) utilizzando il protocollo NFS v4.2.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: fsxz-vol-sc
provisioner: fsx.openzfs.csi.aws.com
parameters:
  ResourceType: "volume"
  ParentVolumeId: '"$ROOT_VOL_ID"'
  CopyTagsToSnapshots: 'false'
  DataCompressionType: '"LZ4"'
  NfsExports: '[{"ClientConfigurations": [{"Clients": "$VPC_CIDR", "Options":
 ["rw","crossmnt","no_root_squash"]}]}]'
  ReadOnly: 'false'
  RecordSizeKiB: '128'
  Tags: '[{"Key": "Name", "Value": "AI-ML"}]'
  OptionsOnDeletion: '["DELETE_CHILD_VOLUMES_AND_SNAPSHOTS"]'
reclaimPolicy: Delete
allowVolumeExpansion: false
mountOptions:
  - nfsvers=4.2
  - rsize=1048576
  - wsize=1048576
  - timeo=600
  - nconnect=16
  - async
```

Un Persistent fsxz-vol-sc Volume Claim (PVC) creato dinamicamente rispetto a quello creato sopra. Nota, la capacità di archiviazione allocata è di 1Gi, necessaria FSx per i volumi OpenZFS, come indicato nelle domande frequenti sui driver CSI. Al volume verrà fornita la piena capacità fornita al file system con questa configurazione. Se è necessario limitare la capacità del volume, è possibile farlo utilizzando quote per utenti o gruppi.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: dynamic-vol-pvc
  namespace: example
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: fsxz-vol-sc
  resources:
    requests:
      storage: 1Gi
```

Configura un pod per montare un volume utilizzando il Persistent Volume Claim (PVC) di dynamic-vol-pvc:

```
kind: Pod
apiVersion: v1
metadata:
  name: fsx-app
  namespace: example
spec:
  volumes:
    - name: dynamic-vol-pv
      persistentVolumeClaim:
        claimName: dynamic-vol-pvc
  containers:
    - name: app
      image: amazonlinux:2023
      command: ["/bin/sh"]
      volumeMounts:
        - mountPath: "/mnt/fsxz"
          name: dynamic-vol-pv
```

Amazon EFS per cache di modelli condivisi

In scenari in cui disponi di un ambiente di calcolo con più istanze di calcolo GPU EC2 e carichi di lavoro dinamici che richiedono l'accesso condiviso al modello su più nodi e zone di disponibilità (ad esempio, inferenza online in tempo reale con Karpenter) con esigenze di prestazioni e scalabilità moderate, consigliamo di utilizzare un file system Amazon Elastic File System (EFS) come volume persistente tramite il driver CSI EFS. Amazon EFS è un file system NFS basato sul cloud completamente gestito, altamente disponibile e scalabile che abilita istanze e contenitori EC2 con storage di file condiviso, con prestazioni costanti e in cui non è richiesto il provisioning anticipato dello storage. Usa EFS come volume modello e monta il volume come file system condiviso definendo un volume persistente sul cluster EKS. Ogni Persistent Volume Claim (PVC) supportato da un file system EFS viene creato come punto di accesso EFS al file system EFS. EFS consente a più nodi e pod di accedere agli stessi file di modello, eliminando la necessità di sincronizzare i dati con il file system di ciascun nodo. Installa il driver EFS CSI per integrare EFS con EKS.

Puoi implementare un file system Amazon EFS con le seguenti modalità di throughput:

- Bursting Throughput: ridimensiona la velocità effettiva in base alle dimensioni del file system, adatta a carichi di lavoro diversi con picchi occasionali.
- Provisioned Throughput: throughput dedicato, ideale per lavori di formazione ML coerenti con esigenze prestazionali prevedibili entro limiti.
- Throughput elastico (consigliato per il machine learning): scalabilità automatica in base al carico di lavoro, conveniente per diversi carichi di lavoro ML.

Per visualizzare le specifiche prestazionali, consulta le specifiche prestazionali di Amazon EFS.

Considerazioni sulle prestazioni:

- Usa Elastic Throughput per diversi carichi di lavoro.
- Usa la classe di storage Standard per carichi di lavoro ML attivi.

Per esempi completi di utilizzo del file system Amazon EFS come volume persistente all'interno del cluster EKS e dei pod, consulta gli esempi di driver CSI EFS in. GitHub Monitora i parametri prestazionali di Amazon EFS utilizzando Amazon CloudWatch. Quando vengono identificati problemi di prestazioni, modifica i parametri di configurazione in base alle esigenze.

Usa S3 Express One Zone per flussi di lavoro orientati agli oggetti e sensibili alla latenza

Per i AI/ML carichi di lavoro sensibili alla latenza su Amazon EKS, come la formazione su modelli su larga scala, l'inferenza o l'analisi ad alte prestazioni, consigliamo di utilizzare S3 Express One Zone per lo storage e il recupero di modelli ad alte prestazioni. S3 Express One Zone offre uno spazio dei nomi gerarchico, simile a un file system, in cui è sufficiente caricare in un bucket di directory, adatto per «inserire tutto», mantenendo al contempo un'alta velocità. Ciò è particolarmente utile se siete abituati a flussi di lavoro orientati agli oggetti. In alternativa, se sei più abituato ai file system (ad esempio, conformi a POSIX), puoi preferire Amazon for Lustre o OpenZFS. FSx Amazon S3 Express One Zone archivia i dati in un'unica zona di disponibilità (AZ) utilizzando bucket di directory e offre una latenza inferiore rispetto ai bucket S3 standard, che distribuiscono i dati su più bucket. AZs Per ottenere i migliori risultati, assicurati di collocare le tue risorse di calcolo EKS nella stessa zona del bucket Express One Zone. Per saperne di più sulle differenze di S3 Express One Zone, consulta Differenze per i bucket di directory.

Per accedere a S3 Express One Zone con la semantica del filesystem, consigliamo di utilizzare il driver CSI Mountpoint S3, che monta i bucket S3 (incluso Express One Zone) come file system locale. Questo traduce le operazioni sui file (ad esempio, apertura, lettura, scrittura) in chiamate API S3, fornendo un accesso ad alta velocità ottimizzato per carichi di lavoro impegnativi in lettura da più client e scritture sequenziali su nuovi oggetti. Per i dettagli sulle operazioni e le limitazioni supportate (ad esempio, nessuna conformità POSIX completa, ma aggiunte e ridenominazioni supportate in Express One Zone), consulta la documentazione semantica di Mountpoint.

Vantaggi prestazionali

- Fornisce un accesso ai dati fino a 10 volte più veloce rispetto a S3 Standard, con una latenza costante di un millisecondo e costi di richiesta inferiori fino all'80%.
- È scalabile per gestire da centinaia a milioni di richieste al secondo per bucket di directory, evitando le throttling o le interruzioni tipiche di S3 standard durante carichi estremi (ad esempio, in cluster con decine o centinaia di migliaia di reti saturate). GPUs/CPUs
- Utilizza un meccanismo di autenticazione basato sulla sessione. Effettua l'autenticazione una volta per ottenere un token di sessione, quindi esegui operazioni ripetute ad alta velocità senza sovraccarico di autenticazione per richiesta. È ottimizzato per carichi di lavoro come checkpoint frequenti o caricamento di dati.

Casi d'uso consigliati

- Memorizzazione nella cache: uno dei principali casi d'uso dell'utilizzo del driver CSI Mountpoint S3 con S3 Express One Zone è la memorizzazione nella cache. La prima istanza legge i dati da S3 Standard (uso generico), memorizzandoli nella cache in Express One Zone a bassa latenza. Le letture successive da parte di altri client accedono più rapidamente ai dati memorizzati nella cache, il che è ideale per scenari a più nodi in cui più nodi EKS leggono gli stessi dati (ad esempio, set di dati di addestramento condivisi). Ciò può migliorare le prestazioni fino a 7 volte per gli accessi ripetuti e ridurre i costi di elaborazione. Per i carichi di lavoro che richiedono la piena conformità POSIX (ad esempio, blocco dei file e modifiche sul posto), considera Amazon FSx for Lustre o OpenZFS come alternative.
- AI/ML Addestramento e inferenza su larga scala: ideale per carichi di lavoro con centinaia o migliaia di nodi di elaborazione (ad esempio, nei cluster EKS) GPUs in cui la limitazione generica di S3 potrebbe causare ritardi e sprecare costose risorse di elaborazione. Ad esempio, i ricercatori LLM o le organizzazioni che utilizzano modelli quotidiani traggono vantaggio da un accesso rapido e affidabile senza interrompere S3 regionale. tests/checkpoints Per carichi di lavoro su scala ridotta (ad esempio, decine di nodi), possono essere sufficienti S3 Standard o altre classi di storage.
- Pipeline di dati: Load/prepare modelli, dati di addestramento di archivio o checkpoint di flusso. Se il tuo team preferisce lo storage di oggetti rispetto ai file system tradizionali (ad esempio, grazie alla familiarità con S3), usalo invece di apportare modifiche ingegneristiche per opzioni conformi a POSIX come Lustre. FSx

Considerazioni

- Resilienza: il design Single-AZ offre una durabilità del 99,449% (come lo standard S3, tramite ridondanza all'interno della AZ) ma una disponibilità inferiore (progettazione al 99,95%, SLA del 99,9%) rispetto alle classi Multi-AZ (disponibilità del 99,99%). È meno resiliente ai guasti AZ. Utilizzalo per dati ricreabili o memorizzati nella cache. Prendi in considerazione la replica o i backup Multi-AZ per carichi di lavoro critici.
- Supporto per API e funzionalità: supporta un sottoinsieme di S3 APIs (ad esempio, nessuna policy o replica del ciclo di vita); potrebbe richiedere modifiche minori all'app per l'autenticazione della sessione o la gestione degli oggetti.
- Integrazione EKS: colloca EKS nella stessa AZ del pods/nodes bucket di directory per ridurre al minimo la latenza di rete. Usa i driver Mountpoint per Amazon S3 o CSI per l'accesso nativo a Kubernetes.

- Test: verifica la latenza di recupero in un ambiente non di produzione per convalidare i miglioramenti delle prestazioni. Monitora il throttling negli scenari S3 standard (ad esempio, elevata saturazione della GPU) e confrontalo.

La classe di storage S3 Express One Zone è disponibile in più regioni e si integra con EKS per i carichi di lavoro che richiedono l'accesso agli oggetti senza attendere lo storage. Per ulteriori informazioni, consulta Guida introduttiva a S3 Express One Zone.

Osservabilità

> **Tip**
>
> Esplora le best practice tramite i workshop Amazon EKS.

Monitoraggio e osservabilità

Spiegazione delle metriche della GPU

La metrica sull'utilizzo della GPU mostra se la GPU ha funzionato correttamente durante la finestra di esempio. Questa metrica rileva la percentuale di tempo in cui la GPU ha eseguito almeno un'istruzione, ma non rivela l'efficienza con cui la GPU ha utilizzato il proprio hardware. Una GPU contiene più Streaming Multiprocessor (SMs), che sono le unità di elaborazione parallele che eseguono le istruzioni. Una lettura dell'utilizzo al 100% può significare che la GPU ha eseguito pesanti carichi di lavoro paralleli su tutte le sue unità SMs, oppure può significare che una singola piccola istruzione ha attivato la GPU durante il periodo di campionamento. Per comprendere l'utilizzo effettivo, è necessario esaminare le metriche della GPU a più livelli dell'architettura hardware. Ogni Streaming Multiprocessor è costruito con diversi tipi di core e ogni livello presenta caratteristiche prestazionali diverse. Le metriche di primo livello (utilizzo della GPU, utilizzo della memoria, potenza della GPU e temperatura della GPU, visibili tramite nvidia-smi) mostrano se il dispositivo è attivo. Metriche più approfondite (utilizzo di SM, SM Activity e utilizzo del tensor core) rivelano l'efficienza con cui la GPU utilizza le sue risorse.

Obiettivi a un elevato consumo energetico della GPU

Il sottoutilizzo GPUs spreca la capacità di elaborazione e aumenta i costi perché i carichi di lavoro non riescono a coinvolgere tutti i componenti della GPU contemporaneamente. Per i AI/ML carichi

di lavoro su Amazon EKS, monitora il consumo energetico della GPU come proxy per identificare l'attività effettiva della GPU. GPU Utilization riporta la percentuale di tempo in cui la GPU esegue qualsiasi kernel, ma non rivela se i multiprocessori di streaming, i controller di memoria e i tensor core siano tutti attivi contemporaneamente. Il consumo energetico colma questa lacuna, perché l'hardware completamente impegnato consuma molta più energia rispetto all'hardware che esegue kernel leggeri o rimane inattivo tra un'attività e l'altra. Confrontate l'energia assorbita con la potenza di progettazione termica (TDP) della GPU per individuare il sottoutilizzo, quindi verificate se il carico di lavoro è ostacolato dalla preelaborazione della CPU, dall'I/O di rete o dalle dimensioni inefficienti dei batch.

CloudWatch Configura Container Insights su Amazon EKS per identificare pod, nodi o carichi di lavoro con un basso consumo energetico della GPU. Questo strumento si integra direttamente con Amazon EKS e consente di monitorare il consumo energetico della GPU e di regolare la pianificazione dei pod o i tipi di istanze quando il consumo energetico scende al di sotto dei livelli target. Se hai bisogno di una visualizzazione avanzata o di dashboard personalizzate, usa DCGM-Exporter di NVIDIA con Prometheus e Grafana per il monitoraggio nativo di Kubernetes. Entrambi gli approcci evidenziano metriche chiave di NVIDIA come (consumo energetico della GPU) e (temperatura della GPU). `nvidia_smi_power_draw` `nvidia_smi_temperature_gpu` Per un elenco di metriche, consulta -Agent-NVIDIA-GPU.htm. https://docs.aws.amazon.com/AmazonCloudWatch/ latest/monitoring/CloudWatch Cerca modelli come un consumo energetico costantemente basso in orari specifici o per lavori particolari. Queste tendenze ti aiutano a identificare dove consolidare i carichi di lavoro o modificare l'allocazione delle risorse.

I limiti statici delle risorse in Kubernetes (come il numero di CPU, memoria e GPU) spesso portano a un approvvigionamento eccessivo o al sottoutilizzo, specialmente per carichi di lavoro dinamici come l'inferenza in cui la domanda oscilla. AI/ML Analizza le tendenze di utilizzo e consolida i carichi di lavoro riducendoli al minimo. GPUs Assicurati che ogni GPU raggiunga il pieno utilizzo prima di allocarne altre. Questo approccio riduce gli sprechi e abbassa i costi. Per una guida dettagliata sull'ottimizzazione della pianificazione e delle strategie di condivisione, consulta le best practice EKS Compute and Autoscaling

Osservabilità e metriche

Utilizzo di strumenti di monitoraggio e osservabilità per i carichi di lavoro AI/ML

AI/ML I servizi moderni richiedono il coordinamento tra infrastruttura, modellazione e logica applicativa. Gli ingegneri della piattaforma gestiscono l'infrastruttura e lo stack di osservabilità. Raccolgono, archiviano e visualizzano le metriche. AI/ML gli ingegneri definiscono metriche

specifiche del modello e monitorano le prestazioni in condizioni di carico e distribuzione dei dati variabili. Gli sviluppatori di applicazioni utilizzano APIs, indirizzano le richieste e tengono traccia delle metriche a livello di servizio e delle interazioni con gli utenti. Senza pratiche di osservabilità unificate, questi team lavorano in silos e perdono segnali critici sullo stato e le prestazioni del sistema. La creazione di una visibilità condivisa tra gli ambienti garantisce che tutte le parti interessate possano rilevare tempestivamente i problemi e mantenere un servizio affidabile.

L'ottimizzazione dei cluster Amazon EKS per i AI/ML carichi di lavoro presenta sfide di monitoraggio uniche, in particolare per quanto riguarda la gestione della memoria GPU. Senza un monitoraggio adeguato, le organizzazioni devono affrontare errori out-of-memory (OOM), inefficienze delle risorse e costi inutili. Un monitoraggio efficace garantisce migliori prestazioni, resilienza e costi inferiori per i clienti EKS. Utilizza un approccio olistico che combini tre livelli di monitoraggio. Innanzitutto, monitora le metriche granulari della GPU utilizzando NVIDIA DCGM Exporter per tenere traccia del consumo energetico della GPU, della temperatura della GPU, dell'attività SM, dell'occupazione SM e degli errori XID. In secondo luogo, monitora framework che servono l'inferenza come Ray e vLLm per ottenere informazioni sui carichi di lavoro distribuiti attraverso le loro metriche native. In terzo luogo, raccogli informazioni a livello di applicazione per tracciare metriche personalizzate specifiche per il tuo carico di lavoro. Questo approccio a più livelli offre visibilità dall'utilizzo dell'hardware alle prestazioni delle applicazioni.

Strumenti e framework

Diversi strumenti e framework forniscono out-of-the-box metriche native per il monitoraggio dei carichi di lavoro. AI/ML Queste metriche integrate eliminano la necessità di strumentazione personalizzata e riducono i tempi di configurazione. Le metriche si concentrano su aspetti prestazionali come latenza, velocità effettiva e generazione di token, che sono fondamentali per il servizio di inferenza e il benchmarking. L'utilizzo di metriche native consente di iniziare il monitoraggio immediatamente senza creare pipeline di raccolta personalizzate.

- VLLm: un motore di servizio ad alta velocità per modelli linguistici di grandi dimensioni (LLMs) che fornisce metriche native come la latenza delle richieste e l'utilizzo della memoria.
- Ray: un framework di elaborazione distribuito che emette metriche per carichi di lavoro AI scalabili, inclusi i tempi di esecuzione delle attività e l'utilizzo delle risorse.
- Hugging Face Text Generation Inference (TGI): un toolkit per l'implementazione e la distribuzione, con metriche integrate per le prestazioni di inferenza. LLMs
- NVIDIA genai-perf: uno strumento a riga di comando per il benchmarking di modelli di intelligenza artificiale generativa, la misurazione del throughput, della latenza e delle metriche specifiche del LLM, come le richieste completate in intervalli di tempo specifici.

Metodi di osservabilità

Si consiglia di implementare eventuali meccanismi di osservabilità aggiuntivi in uno dei seguenti modi.

CloudWatch Container Insights Se la tua organizzazione preferisce strumenti nativi di AWS con una configurazione minima, ti consigliamo CloudWatch Container Insights. Si integra con NVIDIA DCGM Exporter per raccogliere i parametri della GPU e offre dashboard predefinite per informazioni rapide. Abilitato dall'installazione del componente aggiuntivo CloudWatch Observability sul cluster, Container Insights implementa e gestisce il ciclo di vita di NVIDIA DCGM Exporter, che raccoglie le metriche della GPU dai driver di Nvidia e le espone a. CloudWatch

Dopo aver installato Container Insights, rileva automaticamente NVIDIA nel tuo ambiente e raccoglie metriche critiche su salute e CloudWatch prestazioni. GPUs Queste metriche vengono visualizzate su dashboard curate. out-of-the-box Puoi anche integrare Ray e VLLm CloudWatch utilizzando l'Unified CloudWatch Agent per inviare le relative metriche native. Questo approccio unificato semplifica l'osservabilità negli ambienti EKS e consente ai team di concentrarsi sull'ottimizzazione delle prestazioni e dei costi anziché sulla creazione di un'infrastruttura di monitoraggio.

Per un elenco completo delle metriche disponibili, consulta le metriche di Amazon EKS e Kubernetes Container Insights. Per step-by-step indicazioni sull'implementazione del monitoraggio della GPU, consulta Ottieni informazioni operative per i carichi di lavoro delle GPU NVIDIA utilizzando Amazon CloudWatch Container Insights. Per esempi pratici di ottimizzazione della latenza di inferenza, consulta Ottimizzazione della reattività dell'IA: una guida pratica all'inferenza ottimizzata per la latenza di Amazon Bedrock.

Prometheus e Grafana gestiti Se la tua organizzazione ha bisogno di dashboard personalizzati e funzionalità di visualizzazione avanzate, implementa Prometheus con NVIDIA DCGM-Exporter e Grafana per il monitoraggio nativo di Kubernetes. Prometheus acquisisce e archivia le metriche della GPU da DCGM-Exporter, mentre Grafana offre funzionalità flessibili di visualizzazione e avviso. Questo approccio offre un maggiore controllo sulla progettazione delle dashboard e sulla conservazione delle metriche rispetto a Container Insights. CloudWatch

Puoi estendere questo stack di monitoraggio integrando framework open source come Ray e VLLm Ray e vLLm per esportare le loro metriche native su Prometheus. Puoi anche connettere Grafana a una fonte di dati AWS X-Ray per visualizzare tracce distribuite e identificare i colli di bottiglia prestazionali nella pipeline di inferenza. Questa combinazione offre end-to-end visibilità dalle metriche a livello di GPU attraverso i flussi di richiesta a livello di applicazione.

Per step-by-step indicazioni sulla distribuzione di questo stack di monitoraggio, consulta Monitoraggio dei carichi di lavoro GPU su Amazon EKS utilizzando i servizi open source gestiti da AWS.

Prendi in considerazione il monitoraggio della formazione di base e le metriche di ottimizzazione

Monitora i parametri di formazione di base per monitorare lo stato e le prestazioni del tuo cluster Amazon EKS e dei carichi di lavoro di machine learning in esecuzione su di esso. I carichi di lavoro di formazione hanno requisiti di monitoraggio diversi rispetto ai carichi di lavoro di inferenza perché vengono eseguiti per periodi prolungati, consumano risorse in modo diverso e richiedono visibilità sulla convergenza dei modelli e sull'efficienza della pipeline di dati. Le metriche riportate di seguito consentono di identificare i punti critici, ottimizzare l'allocazione delle risorse e garantire il corretto completamento dei lavori di formazione. Per step-by-step indicazioni sull'implementazione di questo approccio di monitoraggio, consulta Introduzione all'osservazione dei carichi di lavoro di machine learning su Amazon EKS.

Metriche sull'utilizzo delle risorse

Monitora le metriche sull'utilizzo delle risorse per verificare che le risorse vengano consumate correttamente. Queste metriche consentono di identificare i punti deboli e le cause principali dei problemi di prestazioni.

- CPU, memoria, rete, potenza della GPU e temperatura della GPU: monitora queste metriche per garantire che le risorse allocate soddisfino le richieste del carico di lavoro e identifichino le opportunità di ottimizzazione. Tieni traccia di metriche come gpu_memory_usage_bytes per identificare i modelli di consumo della memoria e rilevare i picchi di utilizzo. Calcola percentili come il 95° percentile (P95) per comprendere le maggiori esigenze di memoria durante l'allenamento. Questa analisi consente di ottimizzare i modelli e l'infrastruttura per evitare errori OOM e ridurre i costi.
- SM Occupancy, SM FPxx Activity, Activity: monitora queste metriche per capire come viene utilizzata la risorsa sottostante sulla GPU. Obiettivo 0.8 per SM Activity come regola generale.
- Utilizzo delle risorse di nodi e pod: monitora l'utilizzo delle risorse a livello di nodo e pod per identificare il conflitto di risorse e i potenziali colli di bottiglia. Monitora se i nodi si avvicinano ai limiti di capacità, il che può ritardare la pianificazione dei pod e rallentare i lavori di formazione.
- Utilizzo delle risorse rispetto a richieste e limiti: confronta l'utilizzo effettivo delle risorse con le richieste e i limiti configurati per determinare se il cluster è in grado di gestire i carichi di lavoro attuali e di gestire quelli futuri. Questo confronto rivela se è necessario modificare le allocazioni delle risorse per evitare errori di OOM o sprechi di risorse.

- Metriche interne di framework ML: acquisisci le metriche interne di formazione e convergenza da framework ML come e. TensorFlow PyTorch Queste metriche includono le curve di perdita, il tasso di apprendimento, il tempo di elaborazione in batch e la durata delle fasi di formazione. Visualizza queste metriche utilizzando TensorBoard strumenti simili per tracciare la convergenza dei modelli e identificare le inefficienze di formazione.

Metriche delle prestazioni del modello

Monitora le metriche delle prestazioni dei modelli per verificare che il processo di formazione produca modelli che soddisfano i requisiti di precisione e aziendali. Queste metriche consentono di determinare quando interrompere l'allenamento, confrontare le versioni dei modelli e identificare il degrado delle prestazioni.

- Precisione, precisione, richiamo e punteggio F1: tieni traccia di queste metriche per comprendere le prestazioni del tuo modello sui dati di convalida. Calcola il punteggio F1 su un set di convalida dopo ogni periodo di addestramento per valutare se il modello sta migliorando e quando raggiunge livelli di prestazioni accettabili.
- Metriche specifiche per l'azienda e KPIs: definisci e monitora le metriche che misurano direttamente il valore aziendale delle tue iniziative. AI/ML Per un sistema di raccomandazioni, tieni traccia di metriche come la percentuale di clic o il tasso di conversione per assicurarti che il modello produca i risultati aziendali previsti.
- Prestazioni nel tempo: confronta le metriche delle prestazioni tra le versioni del modello e i cicli di formazione per identificare le tendenze e rilevare il degrado. Verifica se le versioni più recenti dei modelli mantengono o migliorano le prestazioni rispetto ai modelli di base. Questo confronto cronologico ti aiuta a decidere se implementare nuovi modelli o esaminare i problemi di formazione.

Metriche sulla qualità dei dati e sulla deriva

Monitora la qualità dei dati e le metriche sulla deriva per garantire che i dati di allenamento rimangano coerenti e rappresentativi. La deriva dei dati può causare un peggioramento delle prestazioni del modello nel tempo, mentre i problemi di qualità dei dati possono impedire la convergenza dei modelli o produrre risultati inaffidabili.

- Proprietà statistiche dei dati di input: monitora le proprietà statistiche come la media, la deviazione standard e la distribuzione delle funzionalità di input nel tempo per rilevare deviazioni o anomalie dei dati. Monitora se le distribuzioni delle funzionalità si discostano in modo significativo dai dati di formazione di base. Ad esempio, se la media di una funzionalità critica cambia di più di due

deviazioni standard, verifica se la pipeline di dati è cambiata o se la fonte di dati sottostante è cambiata.

- Rilevamento e avvisi della deriva dei dati: implementa meccanismi automatizzati per rilevare e avvisare i problemi di qualità dei dati prima che influiscano sulla formazione. Utilizzate test statistici come il test Kolmogorov-Smirnov o il test chi-squared per confrontare le attuali distribuzioni dei dati con i dati di allenamento originali. Imposta avvisi quando i test rilevano deviazioni significative in modo da poter riqualificare i modelli con dati aggiornati o esaminare i problemi della pipeline di dati.

Metriche di latenza e velocità effettiva

Monitora le metriche di latenza e velocità effettiva per identificare i punti deboli nella pipeline di formazione e ottimizzare l'utilizzo delle risorse. Queste metriche aiutano a capire dove viene impiegato il tempo durante la formazione e dove concentrare gli sforzi di ottimizzazione.

- End-to-End Latenza delle pipeline di formazione ML: misura il tempo totale di flusso dei dati attraverso l'intera pipeline di formazione, dall'inserimento dei dati all'aggiornamento del modello. Tieni traccia di questa metrica in tutte le sessioni di formazione per identificare se le modifiche alla pipeline migliorano o peggiorano le prestazioni. L'elevata latenza spesso indica problemi nel caricamento dei dati, nella preelaborazione o nella comunicazione di rete tra i nodi.
- Produttività e velocità di elaborazione della formazione: monitora il volume di dati elaborati dalla pipeline di formazione per unità di tempo per garantire un utilizzo efficiente delle risorse. Monitora metriche come i campioni elaborati al secondo o i batch completati al minuto. Un throughput ridotto rispetto alla capacità hardware suggerisce inefficienze nel caricamento dei dati, nella preelaborazione o nel calcolo dei modelli che sprecano i cicli della GPU.
- Checkpoint Save and Restore Latency: monitora il tempo necessario per salvare i checkpoint del modello nello storage (S3 FSx, EFS) e ripristinarli nella memoria della GPU o della CPU quando si riprendono i lavori o si ripristinano da errori. Le operazioni di checkpoint lente prolungano i tempi di ripristino dei processi e aumentano i costi. Tieni traccia delle dimensioni dei checkpoint, risparmia la durata, la durata del ripristino e il numero di errori per identificare opportunità di ottimizzazione come la compressione o livelli di storage più veloci.
- Tempo di caricamento e preelaborazione dei dati: misura il tempo impiegato per caricare i dati dallo storage e applicare le trasformazioni di preelaborazione. Confrontate questo tempo con il tempo di calcolo del modello per determinare se la formazione è legata ai dati o al calcolo. Se il caricamento dei dati richiede più del 20% del tempo di formazione totale, prendi in considerazione l'ottimizzazione della pipeline di dati con caching, prefetching o storage più veloce.

Tassi di errore e fallimenti

Monitora i tassi di errore e gli errori in tutta la pipeline di formazione per mantenere l'affidabilità e prevenire lo spreco di risorse di elaborazione. Gli errori non rilevati possono causare l'interruzione automatica dei processi di formazione, la produzione di modelli non validi o la perdita di ore di tempo della GPU prima che si notino problemi.

- Monitoraggio degli errori nella pipeline: monitora gli errori in tutte le fasi della pipeline ML, tra cui la preelaborazione dei dati, l'addestramento dei modelli e le operazioni di checkpoint. Registra i tipi di errore, le frequenze e i componenti interessati per identificare rapidamente i problemi. Gli errori più comuni includono mancate corrispondenze nel formato dei dati, out-of-memory errori durante la preelaborazione e errori di salvataggio dei checkpoint dovuti ai limiti di archiviazione. Imposta avvisi quando i tassi di errore superano le soglie di base in modo da poter indagare prima che gli errori si verifichino a cascata.
- Analisi degli errori ricorrenti: identifica e analizza i modelli negli errori ricorrenti per prevenire guasti futuri e migliorare l'affidabilità della pipeline. Analizza i log per scoprire se campioni di dati specifici, dimensioni dei batch o configurazioni di formazione causano costantemente errori. Ad esempio, se determinati tipi di dati di input generano errori di preelaborazione, aggiungi i controlli di convalida nelle fasi iniziali della pipeline o aggiorna la logica di pulizia dei dati. Tieni traccia del tempo medio tra i guasti (MTBF) per misurare se l'affidabilità della pipeline migliora nel tempo. »

Metriche specifiche per Kubernetes ed EKS

Monitora le metriche di Kubernetes ed EKS per garantire che l'infrastruttura del cluster rimanga integra e possa supportare i carichi di lavoro di formazione. Queste metriche ti aiutano a rilevare i problemi dell'infrastruttura prima che causino errori di formazione o un peggioramento delle prestazioni.

- Metriche dello stato del cluster Kubernetes: monitora lo stato e lo stato degli oggetti Kubernetes, inclusi pod, nodi, implementazioni e servizi. Tieni traccia dello stato dei pod per identificare i pod bloccati in stato di sospeso, guasto o crash loop. Monitora le condizioni dei nodi per rilevare problemi come la pressione del disco, la pressione della memoria o l'indisponibilità della rete. Usa kubectl o gli strumenti di monitoraggio per controllare continuamente queste metriche e impostare avvisi quando i pod non si avviano o i nodi diventano non programmabili.
- Metriche di esecuzione della pipeline di formazione: monitora le esecuzioni della pipeline riuscite e non riuscite, la durata dei lavori, i tempi di completamento delle fasi e gli errori di orchestrazione. Monitora se i lavori di formazione vengono completati entro le finestre temporali previste e se i

tassi di fallimento aumentano nel tempo. Tieni traccia di metriche come la percentuale di successo professionale, la durata media del lavoro e il tempo necessario per fallire. Queste metriche consentono di identificare se problemi di infrastruttura, problemi di configurazione o problemi di qualità dei dati causano errori di formazione.

- AWS Service Metrics: monitora i parametri per i servizi AWS che supportano l'infrastruttura EKS e i carichi di lavoro di formazione. Monitora i parametri di S3 come la latenza delle richieste, i tassi di errore e il throughput per garantire che le prestazioni di caricamento dei dati rimangano costanti. Tieni traccia delle metriche del volume EBS, tra cui IOPS, velocità effettiva e lunghezza della coda, per rilevare i colli di bottiglia dello storage. Monitora i log di flusso VPC e le metriche di rete per identificare problemi di connettività tra i nodi o verso servizi esterni.
- Kubernetes Control Plane Metrics: monitora il server API, lo scheduler, il controller manager e il database etcd per rilevare problemi o guasti di prestazioni che influiscono sulle operazioni del cluster. Tieni traccia della latenza, della frequenza delle richieste e del tasso di errore del server API per garantire che il piano di controllo risponda rapidamente alle richieste di pianificazione. Monitora le dimensioni del database etcd, la durata del commit e le modifiche apportate alle modifiche per rilevare problemi di stabilità. L'elevata latenza del server API o le frequenti modifiche al leader etcd possono ritardare la pianificazione dei pod e prolungare i tempi di avvio dei processi di formazione.

Registri di applicazioni e istanze

Raccogli e analizza i log delle applicazioni e delle istanze per diagnosticare problemi che le sole metriche non sono in grado di spiegare. I log forniscono un contesto dettagliato su errori, cambiamenti di stato ed eventi di sistema che aiutano a capire perché i processi di formazione falliscono o funzionano male. La correlazione dei log con le metriche consente di individuare più rapidamente le cause alla radice.

- Registri delle applicazioni: raccogli i registri delle applicazioni dai processi di formazione, dalle pipeline di dati e dai framework ML per identificare i colli di bottiglia e diagnosticare gli errori. Questi registri raccolgono informazioni dettagliate sull'esecuzione del lavoro, inclusi errori di caricamento dei dati, errori di inizializzazione del modello, errori di salvataggio dei checkpoint e avvisi specifici del framework. Correla i timestamp dei log con i picchi metrici per capire cosa ha causato il peggioramento o gli errori delle prestazioni. Ad esempio, se l'utilizzo della GPU diminuisce improvvisamente, controllate i log delle applicazioni per individuare eventuali errori che indichino lo stallo della pipeline di dati o gli errori di preelaborazione. Utilizzate strumenti di registrazione centralizzati come CloudWatch Logs o Fluent Bit per aggregare i log di tutti i pod e renderli ricercabili.

- Registri delle istanze: raccogli i log a livello di istanza, come i log del journal di sistema e l'output di dmesg, per rilevare problemi hardware e problemi a livello di kernel. Questi registri rivelano problemi come errori dei driver della GPU, errori di allocazione della memoria, errori del disco e problemi dell'interfaccia di rete che potrebbero non apparire nei registri delle applicazioni. I/O Correla i log delle istanze con i log e le metriche delle applicazioni per determinare se gli errori di formazione derivano da problemi hardware o problemi delle applicazioni. Ad esempio, se un processo di formazione fallisce con un out-of-memory errore, controllate nei log di dmesg i messaggi killer del kernel OOM che indicano se il sistema ha esaurito la memoria o se l'applicazione ha superato i limiti del contenitore. Imposta avvisi per errori hardware critici come errori XID della GPU o guasti del disco in modo da poter sostituire le istanze difettose prima che causino interruzioni diffuse della formazione.

Le seguenti sezioni mostrano come raccogliere le metriche sopra descritte utilizzando due approcci consigliati da AWS: CloudWatch Container Insights e Amazon Managed Prometheus Amazon Managed Prometheus con Amazon Managed Grafana. Scegli CloudWatch Container Insights se preferisci gli strumenti nativi di AWS con configurazione minima e dashboard predefiniti. Scegli Amazon Managed Prometheus con Amazon Managed Grafana se hai bisogno di dashboard personalizzati, funzionalità di visualizzazione avanzate o desideri integrarti con l'infrastruttura di monitoraggio esistente basata su Prometheus. Per un elenco completo delle metriche di Container Insights disponibili, consulta le metriche di Amazon EKS e Kubernetes Container Insights.

Prendi in considerazione il monitoraggio delle metriche di inferenza online in tempo reale

Nei sistemi in tempo reale, la bassa latenza è fondamentale per fornire risposte tempestive agli utenti o ad altri sistemi dipendenti. L'elevata latenza può compromettere l'esperienza utente o violare i requisiti prestazionali. I componenti che influenzano la latenza di inferenza includono il tempo di caricamento del modello, il tempo di preelaborazione, il tempo di previsione effettivo, il tempo di post-elaborazione, il tempo di trasmissione di rete. Consigliamo di monitorare la latenza di inferenza per garantire risposte a bassa latenza che soddisfino gli accordi sui livelli di servizio () e di sviluppare metriche personalizzate per quanto segue. SLAs Esegui il test in base al carico previsto, includi la latenza di rete, tieni conto delle richieste simultanee ed esegui il test con batch di dimensioni variabili.

- Time to First Token (TTFT): periodo di tempo che intercorre tra il momento in cui un utente invia una richiesta e la ricezione dell'inizio di una risposta (la prima parola, token o blocco). Ad esempio, nei chatbot, dovresti controllare quanto tempo impiega a generare il primo output (token) dopo che l'utente ha posto una domanda.

- End-to-End Latenza: è il tempo totale tra la ricezione di una richiesta e l'invio della risposta. Ad esempio, misura il tempo che intercorre tra la richiesta e la risposta.
- Token di output per secondo (TPS): indica la velocità con cui il modello genera nuovi token dopo aver iniziato a rispondere. Ad esempio, nei chatbot, monitoreresti la velocità di generazione dei modelli linguistici per un testo di base.
- Frequenza di errore: tiene traccia delle richieste non riuscite, il che può indicare problemi di prestazioni. Ad esempio, monitora le richieste non riuscite per documenti di grandi dimensioni o determinati caratteri.
- Throughput: misura il numero di richieste o operazioni che il sistema è in grado di gestire per unità di tempo. Ad esempio, tieni traccia delle richieste al secondo per gestire i carichi di picco.

K/V (Key/Value) cache can be a powerful optimization technique for inference latency, particularly relevant for transformer-based models. K/V cache stores the key and value tensors from previous transformer layer computations, reducing redundant computations during autoregressive inference, particularly in large language models (LLMs). Cache Efficiency Metrics (specifically for K/Vo una sessione (uso della cache):

- hit/miss Rapporto di cache: per le configurazioni di inferenza che sfruttano la memorizzazione nella cache (K/V o cache incorporate), misura la frequenza con cui la cache è utile. Le basse percentuali di successo possono indicare che la configurazione della cache o le modifiche del carico di lavoro non sono ottimali, entrambi fattori che possono aumentare la latenza.

Negli argomenti successivi, dimostreremo la raccolta di dati per alcune delle metriche sopra menzionate. Forniremo esempi dei due approcci consigliati da AWS: AWS-Native CloudWatch Container Insights e Amazon Managed Prometheus open source con Amazon Managed Grafana. Sceglieresti una di queste soluzioni in base alle tue esigenze complessive di osservabilità. Consulta i parametri di Amazon EKS e Kubernetes Container Insights per l'elenco completo dei parametri di Container Insights.

Monitoraggio dell'utilizzo della memoria della GPU

Come discusso nell'the section called “Prendi in considerazione il monitoraggio della formazione di base e le metriche di ottimizzazione”argomento, l'utilizzo della memoria GPU è essenziale per prevenire errori out-of-memory (OOM) e garantire un utilizzo efficiente delle risorse. Gli esempi seguenti mostrano come utilizzare l'applicazione di formazione per esporre un istogramma metrico personalizzato e calcolare l'utilizzo della memoria P95 per identificare i picchi di consumo.

`gpu_memory_usage_bytes` Assicuratevi di eseguire il test con un esempio di lavoro di formazione (ad esempio, la messa a punto di un modello di trasformatore) in un ambiente di staging.

Esempio di AWS-Native CloudWatch Container Insights

Questo esempio dimostra come strumentare l'applicazione di formazione per esporla `gpu_memory_usage_bytes` come istogramma utilizzando l'approccio nativo di AWS. Tieni presente che il AI/ML contenitore deve essere configurato per emettere log strutturati in formato CloudWatch Embedded Metrics Format (EMF). CloudWatch logs analizza EMF e pubblica le metriche. Usa aws_embedded_metrics nella tua applicazione di formazione per inviare log strutturati in formato EMF a Logs, che estrae le metriche della GPU. CloudWatch

```
from aws_embedded_metrics import metric_scope
import torch
import numpy as np

memory_usage = []

@metric_scope
def log_gpu_memory(metrics):
    # Record current GPU memory usage
    mem = torch.cuda.memory_allocated()
    memory_usage.append(mem)

    # Log as histogram metric
    metrics.set_namespace("MLTraining/GPUMemory")
    metrics.put_metric("gpu_memory_usage_bytes", mem, "Bytes", "Histogram")

    # Calculate and log P95 if we have enough data points
    if len(memory_usage) >= 10:
        p95 = np.percentile(memory_usage, 95)
        metrics.put_metric("gpu_memory_p95_bytes", p95, "Bytes")
        print(f"Current memory: {mem} bytes, P95: {p95} bytes")

# Example usage in training loop
for epoch in range(20):
    # Your model training code would go here
    log_gpu_memory()
```

Esempio di Prometheus e Grafana

Questo esempio dimostra come strumentare l'applicazione di formazione per esporla `gpu_memory_usage_bytes` come istogramma utilizzando la libreria client Prometheus in Python.

```
from prometheus_client import Histogram
from prometheus_client import start_http_server
import pynvml

start_http_server(8080)
memory_usage = Histogram(
    'gpu_memory_usage_bytes',
    'GPU memory usage during training',
    ['gpu_index'],
    buckets=[1e9, 2e9, 4e9, 8e9, 16e9, 32e9]
)

# Function to get GPU memory usage
def get_gpu_memory_usage():
    if torch.cuda.is_available():
        # Get the current GPU device
        device = torch.cuda.current_device()

        # Get memory usage in bytes
        memory_allocated = torch.cuda.memory_allocated(device)
        memory_reserved = torch.cuda.memory_reserved(device)

        # Total memory usage (allocated + reserved)
        total_memory = memory_allocated + memory_reserved

        return device, total_memory
    else:
        return None, 0

# Get GPU memory usage
gpu_index, memory_used = get_gpu_memory_usage()
```

Tieni traccia della durata della richiesta di inferenza per l'inferenza online in tempo reale

Come discusso nell'the section called “Prendi in considerazione il monitoraggio della formazione di base e le metriche di ottimizzazione”argomento, la bassa latenza è fondamentale per fornire risposte tempestive agli utenti o ad altri sistemi dipendenti. Gli esempi seguenti mostrano come tenere traccia di una metrica istogramma personalizzata`inference_request_duration_seconds`, esposta

dall'applicazione di inferenza. Calcola la latenza del 95° percentile (P95) per concentrarti sugli scenari peggiori, esegui test con richieste di inferenza sintetica (ad esempio, tramite Locust) in un ambiente di staging e imposta soglie di avviso (ad esempio, >500 ms) per rilevare le violazioni degli SLA.

Esempio di AWS-Native CloudWatch Container Insights

Questo esempio dimostra come creare una metrica di istogramma personalizzata nella tua applicazione di inferenza per inference_request_duration_seconds utilizzando AWS Embedded Metric Format. CloudWatch

```
import boto3
import time
from aws_embedded_metrics import metric_scope, MetricsLogger

cloudwatch = boto3.client('cloudwatch')

@metric_scope
def log_inference_duration(metrics: MetricsLogger, duration: float):
    metrics.set_namespace("ML/Inference")
    metrics.put_metric("inference_request_duration_seconds", duration, "Seconds",
 "Histogram")
    metrics.set_property("Buckets", [0.1, 0.5, 1, 2, 5])

@metric_scope
def process_inference_request(metrics: MetricsLogger):
    start_time = time.time()

    # Your inference processing code here
    # For example:
    # result = model.predict(input_data)

    duration = time.time() - start_time
    log_inference_duration(metrics, duration)

    print(f"Inference request processed in {duration} seconds")

# Example usage
process_inference_request()
```

Esempio di Prometheus e Grafana

Questo esempio dimostra come creare una metrica istogramma personalizzata nell'applicazione di inferenza per inference_request_duration_seconds utilizzando la libreria client Prometheus in Python:

```
from prometheus_client import Histogram
from prometheus_client import start_http_server
import time

start_http_server(8080)
request_duration = Histogram(
    'inference_request_duration_seconds',
    'Inference request latency',
    buckets=[0.1, 0.5, 1, 2, 5]
)
start_time = time.time()
# Process inference request
request_duration.observe(time.time() - start_time)
```

In Grafana, usa la query `histogram_quantile(0.95, sum(rate(inference_request_duration_seconds_bucket[5m])) by (le, pod))` per visualizzare le tendenze di latenza P95. Per ulteriori informazioni, vedere Prometheus Histogram Documentation e Prometheus Client Documentation.

Tieni traccia del throughput dei token per un'inferenza online in tempo reale

Come discusso nell'the section called “Prendi in considerazione il monitoraggio della formazione di base e le metriche di ottimizzazione”argomento, consigliamo di monitorare il tempo di elaborazione dei token per valutare le prestazioni del modello e ottimizzare le decisioni di scalabilità. Gli esempi seguenti mostrano come tenere traccia di una metrica istogramma personalizzata`token_processing_duration_seconds`, esposta dall'applicazione di inferenza. Calcola la durata del 95° percentile (P95) per analizzare l'efficienza di elaborazione, eseguire test con carichi di richieste simulati (ad esempio, da 100 a 1000 richieste/secondo) in un cluster non di produzione e regolare i trigger KEDA per ottimizzare la scalabilità.

Esempio di AWS-Native CloudWatch Container Insights

Questo esempio dimostra come creare una metrica di istogramma personalizzata nella tua applicazione di inferenza per token_processing_duration_seconds utilizzando AWS Embedded Metric Format. CloudWatch Utilizza le dimensioni (funzione `) with a custom `get_duration_bucket `set_dimension) per classificare le durate in intervalli (ad esempio, «0.01", «>1").

```
import boto3
import time
```

```
from aws_embedded_metrics import metric_scope, MetricsLogger

cloudwatch = boto3.client('cloudwatch')

@metric_scope
def log_token_processing(metrics: MetricsLogger, duration: float, token_count: int):
    metrics.set_namespace("ML/TokenProcessing")
    metrics.put_metric("token_processing_duration_seconds", duration, "Seconds")
    metrics.set_dimension("ProcessingBucket", get_duration_bucket(duration))
    metrics.set_property("TokenCount", token_count)

def get_duration_bucket(duration):
    buckets = [0.01, 0.05, 0.1, 0.5, 1]
    for bucket in buckets:
        if duration <= bucket:
            return f"<={bucket}"
    return f">{buckets[-1]}"

@metric_scope
def process_tokens(input_text: str, model, tokenizer, metrics: MetricsLogger):
    tokens = tokenizer.encode(input_text)
    token_count = len(tokens)

    start_time = time.time()
    # Process tokens (replace with your actual processing logic)
    output = model(tokens)
    duration = time.time() - start_time

    log_token_processing(metrics, duration, token_count)
    print(f"Processed {token_count} tokens in {duration} seconds")
    return output
```

Esempio di Prometheus e Grafana

Questo esempio dimostra come creare una metrica istogramma personalizzata nell'applicazione di inferenza per token_processing_duration_seconds utilizzando la libreria client Prometheus in Python.

```
from prometheus_client import Histogram
from prometheus_client import start_http_server
import time

start_http_server(8080)
token_duration = Histogram(
```

```
    'token_processing_duration_seconds',
    'Token processing time per request',
    buckets=[0.01, 0.05, 0.1, 0.5, 1]
)
start_time = time.time()
# Process tokens
token_duration.observe(time.time() - start_time)
```

In Grafana, usa la query `histogram_quantile(0.95, sum(rate(token_processing_duration_seconds_bucket[5m])) by (le, pod))` per visualizzare le tendenze dei tempi di elaborazione P95. Per ulteriori informazioni, vedere Prometheus Histogram Documentation e Prometheus Client Documentation.

Misura la latenza di Checkpoint Restore

Come discusso nell'the section called “Prendi in considerazione il monitoraggio della formazione di base e le metriche di ottimizzazione”argomento, la latenza dei checkpoint è una metrica fondamentale durante più fasi del ciclo di vita del modello. Gli esempi seguenti mostrano come tenere traccia di una metrica istogramma personalizzata, esposta dall'applicazione. `checkpoint_restore_duration_seconds` Calcola la durata del 95° percentile (P95) per monitorare le prestazioni di ripristino, eseguire test con interruzioni Spot in un cluster non di produzione e impostare soglie di avviso (ad esempio, <30 secondi) per rilevare i ritardi.

Esempio di AWS-Native CloudWatch Container Insights

Questo esempio dimostra come utilizzare l'applicazione batch per esporre checkpoint_restore_duration_seconds come istogramma utilizzando Insights: CloudWatch

```
import boto3
import time
import torch
from aws_embedded_metrics import metric_scope, MetricsLogger

@metric_scope
def log_checkpoint_restore(metrics: MetricsLogger, duration: float):
    metrics.set_namespace("ML/ModelOperations")
    metrics.put_metric("checkpoint_restore_duration_seconds", duration, "Seconds",
 "Histogram")
    metrics.set_property("Buckets", [1, 5, 10, 30, 60])
    metrics.set_property("CheckpointSource", "s3://my-bucket/checkpoint.pt")

@metric_scope
```

```
def load_checkpoint(model, checkpoint_path: str, metrics: MetricsLogger):
    start_time = time.time()

    # Load model checkpoint
    model.load_state_dict(torch.load(checkpoint_path))

    duration = time.time() - start_time
    log_checkpoint_restore(metrics, duration)

    print(f"Checkpoint restored in {duration} seconds")
```

Esempio di Prometheus e Grafana

Questo esempio dimostra come strumentare l'applicazione batch per esporla `checkpoint_restore_duration_seconds` come istogramma utilizzando la libreria client Prometheus in Python:

```
from prometheus_client import Histogram
from prometheus_client import start_http_server
import torch

start_http_server(8080)
restore_duration = Histogram(
    'checkpoint_restore_duration_seconds',
    'Time to restore checkpoint',
    buckets=[1, 5, 10, 30, 60]
)
with restore_duration.time():
    model.load_state_dict(torch.load("s3://my-bucket/checkpoint.pt"))
```

In Grafana, usa la query `histogram_quantile(0.95, sum(rate(checkpoint_restore_duration_seconds_bucket[5m]) by (le))` per visualizzare le tendenze della latenza di ripristino P95. Per ulteriori informazioni, vedere Prometheus Histogram Documentation e Prometheus Client Documentation.

Scalabilità e prestazioni delle applicazioni

> **Tip**
>
> Esplora le best practice tramite i workshop Amazon EKS.

Gestione degli artefatti ML, framework di servizio e ottimizzazione delle startup

La distribuzione di modelli di machine learning (ML) su Amazon EKS richiede un'attenta considerazione del modo in cui i modelli vengono integrati nelle immagini dei container e negli ambienti di runtime. Ciò garantisce scalabilità, riproducibilità e utilizzo efficiente delle risorse. Questo argomento descrive i diversi approcci alla gestione degli artefatti dei modelli ML, alla selezione dei framework di servizio e all'ottimizzazione dei tempi di avvio dei container attraverso tecniche come il pre-caching, tutte personalizzate per ridurre i tempi di avvio dei container.

Gestione degli artefatti del modello ML nelle distribuzioni

Una decisione chiave è come gestire autonomamente gli artefatti del modello ML (come pesi e configurazioni). La scelta influisce sulle dimensioni dell'immagine, sulla velocità di implementazione, sulla frequenza di aggiornamento del modello e sul sovraccarico operativo. Tieni presente che quando ci riferiamo alla memorizzazione del «modello», ci riferiamo agli artefatti del modello (come i parametri addestrati e i pesi del modello). Esistono diversi approcci alla gestione degli artefatti del modello ML su Amazon EKS. Ognuno ha i suoi compromessi e il migliore dipende dalle dimensioni del modello, dalla cadenza di aggiornamento e dalle esigenze di infrastruttura. Considerate i seguenti approcci, dal meno consigliato al più consigliato:

- Inserimento del modello nell'immagine del contenitore: copia i file del modello (ad es. .safetensors, .pth, .h5) nell'immagine del contenitore (ad esempio, Dockerfile) durante la creazione dell'immagine. Il modello fa parte dell'immagine immutabile. Si consiglia di utilizzare questo approccio per i modelli più piccoli con aggiornamenti poco frequenti. Ciò garantisce coerenza e riproducibilità, non prevede ritardi di caricamento e semplifica la gestione delle dipendenze, ma comporta immagini di dimensioni maggiori, rallenta le build e le push, richiede la ricostruzione e la ridistribuzione per gli aggiornamenti dei modelli e non è ideale per i modelli di grandi dimensioni a causa del throughput di pull del registro.
- Scaricamento del modello in fase di esecuzione: all'avvio del contenitore, l'applicazione scarica il modello da uno storage esterno (ad esempio, Amazon S3, supportato da S3 CRT per trasferimenti ottimizzati ad alto rendimento utilizzando metodi come il driver CSI Mountpoint per S3, AWS S3 CLI o l'OSS CLI s5cmd) tramite script in un contenitore init o entrypoint. Consigliamo di iniziare con questo approccio per modelli di grandi dimensioni con aggiornamenti frequenti. In questo modo le immagini dei container si concentrano sul codice/runtime, facilita gli aggiornamenti dei modelli senza ricostruzioni, supporta il controllo delle versioni tramite metadati di archiviazione, ma

introduce potenziali errori di rete (richiede una logica di ripetizione) e richiede l'autenticazione e la memorizzazione nella cache.

Per ulteriori informazioni, consulta Accelerare il processo di pull nel workshop AI on EKS.

AI servizio dei modelli ML

La distribuzione e l'utilizzo di modelli di machine learning (ML) su Amazon EKS richiede la selezione di un approccio di model serving appropriato per ottimizzare latenza, throughput, scalabilità e semplicità operativa. La scelta dipende dal tipo di modello (ad esempio, linguaggio, modello di visione), dalle richieste del carico di lavoro (ad esempio, inferenza in tempo reale) e dall'esperienza del team. Gli approcci comuni includono configurazioni basate su Python per la prototipazione, server modello dedicati per funzionalità di livello di produzione e motori di inferenza specializzati per alte prestazioni ed efficienza. Ogni metodo comporta compromessi in termini di complessità di configurazione, prestazioni e utilizzo delle risorse. Tieni presente che i framework di gestione possono aumentare le dimensioni (multiple GBs) delle immagini dei container a causa delle dipendenze, con un potenziale impatto sui tempi di avvio: prendi in considerazione la possibilità di disaccoppiare i container utilizzando tecniche di gestione degli artefatti per mitigare questo problema. Le opzioni sono elencate da quelle meno consigliate a quelle più consigliate:

Utilizzo di framework Python (ad esempio, FastAPI, HuggingFace Transformers with) PyTorch Sviluppa un'applicazione personalizzata utilizzando framework Python, incorporando file modello (weights, config, tokenizer) all'interno di una configurazione di nodi containerizzata.

- Vantaggi: prototipazione semplice, solo Python senza infrastruttura aggiuntiva, compatibile con tutti i HuggingFace modelli, implementazione semplice di Kubernetes.
- Svantaggi: si limita al singolo request/simple batch, la generazione lenta di token (nessun kernel ottimizzato), la memoria è inefficiente, manca di scalabilità e monitoraggio e comporta lunghi tempi di avvio.
- Raccomandazione: utilizzalo per la prototipazione iniziale o per attività a nodo singolo che richiedono un'integrazione logica personalizzata.

Utilizzo di framework di model serving dedicati (ad es. Tensorrt-LLM, TGI) Adotta server specializzati come Tensorrt-LLM o TGI per l'inferenza ML, gestendo il caricamento, il routing e l'ottimizzazione del modello. Questi supportano formati come safetensors, con compilazioni o plugin opzionali.

- Vantaggi: offre il batching (parallelismo). static/in-flight or continuous), quantization (INT8, FP8, GPTQ), hardware optimizations (NVIDIA, AMD, Intel, Inferentia), and multi-GPU support (Tensor/ Pipeline TensorRT-LLM supporta diversi modelli (Encoder-Decoder)LLMs, mentre TGI sfrutta l'integrazione. HuggingFace
- Contro: Tensorrt-LLM necessita di compilazione ed è solo per NVIDIA; TGI potrebbe essere meno efficiente nel batching; entrambi aggiungono un sovraccarico di configurazione e potrebbero non adattarsi a tutti i tipi di modello (ad esempio, non trasformatori).
- PyTorch/TensorFlow Raccomandazione: adatto per modelli che richiedono funzionalità di produzione come test o produttività elevata con hardware compatibile. A/B

Utilizzo di motori di inferenza specializzati ad alto rendimento (ad esempio, vLLM) Utilizza motori di inferenza avanzati come vLLM, ottimizzando il servizio LLM con, batch in volo e quantizzazione (, -KV, AWQ) PagedAttention, integrabili con l'autoscaling EKS. INT8 FP8

- Vantaggi: elevata produttività ed efficienza della memoria (risparmio del 40-60% di VRAM), gestione dinamica delle richieste, streaming di token, supporto multi-GPU Tensor Parallel a nodo singolo e ampia compatibilità hardware.
- Svantaggi: ottimizzato per i trasformatori che utilizzano solo decoder (ad es. LLa MA), meno efficace per i modelli senza trasformatore, richiede hardware compatibile (ad esempio, NVIDIA) e sforzi di configurazione. GPUs
- Raccomandazione: la scelta migliore per l'inferenza LLM ad alto volume e bassa latenza su EKS, che massimizza la scalabilità e le prestazioni.

Ottimizzazione dei tempi di recupero delle immagini dei contenitori

Le immagini di contenitori di grandi dimensioni possono causare ritardi di avvio a freddo che influiscono sulla latenza di avvio del pod. Per i carichi di lavoro sensibili alla latenza, come i carichi di lavoro di inferenza in tempo reale scalati orizzontalmente, l'avvio rapido dei pod è fondamentale. Considerate i seguenti approcci per ottimizzare i tempi di recupero delle immagini dei container:

Riduzione delle dimensioni delle immagini dei contenitori

Ridurre le dimensioni delle immagini dei contenitori durante l'avvio è un altro modo per rimpicciolire le immagini. È possibile effettuare riduzioni in ogni fase del processo di creazione dell'immagine del contenitore. Per iniziare, scegliete immagini di base che contengano il minor numero di dipendenze richieste. Durante la creazione delle immagini, includi solo le librerie e gli artefatti essenziali richiesti.

Durante la creazione di immagini, provate a combinare più `COPY` comandi `RUN` per creare un numero inferiore di livelli più grandi. Per quanto riguarda i AI/ML framework, utilizzate build in più fasi per separare build e runtime, copiando solo gli elementi necessari (ad esempio, tramite i registri `COPY –from=` for o i contesti locali) e selezionate varianti come immagini solo in fase di esecuzione (ad esempio, da 3,03 GB rispetto a devel a 6,66 GB). `pytorch/pytorch:2.7.1-cuda11.8-cudnn9-runtime` Per ulteriori informazioni, consulta Riduzione delle dimensioni dell'immagine del contenitore nel workshop AI on EKS.

Utilizzo dello snapshotter SOCI per preestrarre le immagini

Per immagini molto grandi che non è possibile minimizzare facilmente, è possibile utilizzare lo snapshotter open source Seekable OCI (SOCI) configurato in modalità pull and unpack parallela. Questa soluzione consente di utilizzare le immagini esistenti senza ricostruire o modificare le pipeline di compilazione. Questa opzione è particolarmente efficace quando si distribuiscono carichi di lavoro con immagini molto grandi su istanze di calcolo EC2 ad alte prestazioni. Funziona bene con configurazioni di rete ad alto throughput e storage ad alte prestazioni, come è tipico dei carichi di lavoro scalabili. AI/ML

La pull/unpack modalità parallela SOCI migliora le prestazioni di acquisizione delle end-to-end immagini attraverso strategie di parallelizzazione configurabili. L'estrazione e la preparazione delle immagini più rapide influiscono direttamente sulla rapidità con cui è possibile implementare nuovi carichi di lavoro e scalare il cluster in modo efficiente. Le acquisizioni di immagini hanno due fasi principali:

1. Recupero dei livelli dal registro al nodo

   Per l'ottimizzazione del recupero dei livelli, SOCI crea più connessioni HTTP simultanee per livello, moltiplicando la velocità di download oltre la limitazione della singola connessione. Divide i livelli di grandi dimensioni in blocchi e li scarica contemporaneamente su più connessioni. Questo approccio aiuta a saturare la larghezza di banda di rete disponibile e a ridurre significativamente i tempi di download. Ciò è particolarmente utile per i AI/ML carichi di lavoro in cui un singolo livello può pesare diversi gigabyte.

2. Disimballare e preparare quegli strati per creare contenitori

   Per l'ottimizzazione del disimballaggio dei livelli, SOCI elabora più livelli contemporaneamente. Invece di aspettare che ogni livello venga decompresso completamente prima di iniziare il successivo, utilizza i core della CPU disponibili per decomprimere ed estrarre più livelli

contemporaneamente. Questa elaborazione parallela trasforma la tradizionale fase di decompressione legata all'I/O in un'operazione ottimizzata per la CPU che si adatta ai core disponibili. Il sistema orchestra attentamente questa parallelizzazione per mantenere la coerenza del file system massimizzando al contempo la velocità di trasmissione.

La modalità pull parallela SOCI utilizza un sistema di controllo a doppia soglia con parametri configurabili sia per la concomitanza dei download che per il parallelismo di decompressione. Questo controllo granulare consente di ottimizzare il comportamento di SOCI per soddisfare i requisiti prestazionali e le condizioni ambientali specifici. La comprensione di questi parametri consente di ottimizzare il tempo di esecuzione per ottenere le migliori prestazioni di pull.

Riferimenti

- Per ulteriori informazioni sulla soluzione e sui compromessi di ottimizzazione, consulta la documentazione sulle funzionalità nell'archivio del progetto SOCI su. GitHub
- Per un esempio pratico con Karpenter su Amazon EKS, consulta Karpenter Blueprint che utilizza la modalità parallela dello snapshotter SOCI. pull/unpack
- Per informazioni sulla configurazione di Bottlerocket per il pull parallelo, vedere soci-snapshotter Parallel Pull Unpack Mode nella documentazione di Bottlerocket.

Utilizzo delle istantanee EBS per preestrarre le immagini

Puoi creare uno snapshot di Amazon Elastic Block Store (EBS) delle immagini dei container memorizzate nella cache e riutilizzarlo per i nodi di lavoro EKS. Ciò garantisce che le immagini vengano precaricate localmente all'avvio del nodo, riducendo i tempi di inizializzazione dei pod. Per ulteriori informazioni, consulta Ridurre i tempi di avvio dei container su Amazon EKS con il volume di dati Bottlerocket per ulteriori informazioni utilizzando Karpenter ed EKS Terraform Blueprints per gruppi di nodi gestiti.

Per saperne di più, consulta Using containerd snapshotter e Preload delle immagini dei container nei volumi di dati Bottlerocket con EBS Snapshots nell'AI on EKS Workshop.

Utilizzo della Container Runtime Cache per preestrarre le immagini

Puoi precaricare le immagini dei container sui nodi utilizzando le risorse Kubernetes (ad esempio, DaemonSet o Deployment) per popolare la cache di runtime del contenitore del nodo. La cache di runtime del contenitore è l'archiviazione locale gestita dal runtime del contenitore (ad esempio,

containerd), in cui le immagini vengono archiviate dopo essere state estratte da un registro. La preestrazione assicura che le immagini siano disponibili localmente, evitando ritardi nel download durante l'avvio del pod. Questo approccio è particolarmente utile quando le immagini cambiano spesso (ad esempio, aggiornamenti frequenti), quando le istantanee EBS non sono preconfigurate, quando la creazione di un volume EBS richiederebbe più tempo rispetto all'estrazione diretta da un registro di contenitori o quando i nodi sono già nel cluster e devono far girare i pod su richiesta utilizzando una delle diverse immagini possibili.

La preselezione di tutte le varianti garantisce tempi di avvio rapidi indipendentemente dall'immagine necessaria. Ad esempio, in un carico di lavoro ML estremamente parallelo che richiede 100.000 piccoli modelli creati utilizzando 10 tecniche diverse, il preupload di 10 immagini tramite un DaemonSet cluster di grandi dimensioni (ad esempio migliaia di nodi) riduce al minimo il tempo di avvio del pod, consentendone il completamento in meno di 10 secondi evitando richiami su richiesta. L'utilizzo dell'approccio Container Runtime Cache elimina la necessità di gestire le istantanee EBS e garantisce di avere sempre a disposizione la versione più recente dell'immagine del contenitore DaemonSets, ma per i carichi di lavoro di inferenza in tempo reale in cui i nodi scalano i in/out, new nodes added by tools like Cluster Autoscaler may schedule workload pods before the pre-pull DaemonSet completes image pulling. This can cause the initial pod on the new node to trigger the pull anyway, potentially delaying startup and impacting low-latency requirements. Additionally, kubelet image garbage collection can affect pre-pulled images by removing unused ones when disk usage exceeds certain thresholds or if they exceed a configured maximum unused age. In scale-in/out modelli, ciò può comportare l'eliminazione delle immagini sui nodi inattivi, con la conseguente necessità di richiamare nuovamente le immagini durante i successivi scale-up e ridurre l'affidabilità della cache per carichi di lavoro a picco.

Consulta il GitHub repository AWS per esempi di preinserimento delle immagini nella cache di runtime del contenitore.

Prendi in considerazione NVMe lo storage in kubelet e containerd

Valuta la possibilità di configurare `kubelet` e utilizzare dischi `containerd` di storage a istanze temporanee per prestazioni del disco più elevate NVMe . Il processo di estrazione del contenitore prevede lo scaricamento di un'immagine del contenitore da un registro e la decompressione dei relativi livelli in un formato utilizzabile. Per ottimizzare I/O le operazioni durante la decompressione, è necessario valutare cosa offre livelli più elevati di I/O prestazioni e velocità effettiva per il tipo di istanza dell'host del contenitore: NVMe istanze con backup con archiviazione locale rispetto a IBS Volume IOPS/throughput. Per le istanze EC2 con archiviazione NVMe locale, prendi in considerazione la configurazione del filesystem sottostante del nodo per kubelet ()`/var/lib/`

`kubelet`, containerd () e Pod logs (`/var/log/pods`) per utilizzare dischi di archiviazione di istanze `/var/lib/containerd` temporanee per livelli di prestazioni e throughput più elevati. NVMe I/O

Lo storage temporaneo del nodo può essere condiviso tra i Pod che richiedono storage temporaneo e le immagini dei contenitori che vengono scaricate sul nodo. Se si utilizza Karpenter con Bottlerocket o AL2023 EKS Optimized, AMIs questo può essere configurato impostando su RAID0o, se si utilizza Managed Node Groups, EC2NodeClass instanceStorePolicy impostando l'ingresso come parte dei dati utente. localStoragePolicy NodeConfig

Contribuisci a questa guida

Chiunque può contribuire alla guida sulle migliori pratiche. La Guida alle migliori pratiche EKS è scritta nel AsciiDoc formato su GitHub.

Riepilogo per i contributori esistenti

- Apri `bpg-docs.code-workspace`con VS Code per installare automaticamente l' AsciiDoc estensione.
  - Scopri di più sull'AsciiDoc estensione su Visual Studio Marketplace.
- I file sorgente per il sito Web AWS Docs sono archiviati in `latest/bpg`
- La sintassi è molto simile a markdown.
  - Esamina il riferimento alla sintassi nei documenti. AsciiDoctor
- La piattaforma docs esegue solo la distribuzione. `latest/bpg/images` Ciascuna delle sezioni della guida contiene un collegamento simbolico a questa directory. Ad esempio, `latest/bpg/networking/images` indica. `latest/bpg/images`

Configura un ambiente di modifica locale

Se prevedi di modificare frequentemente la guida, configura un ambiente di modifica locale.

Fork e clona il repository

È necessario conoscere e utilizzare gli editor `git` di `github` testo. Per informazioni su come iniziare a usare `git` `egithub`, vedi Guida introduttiva al tuo GitHub account nei GitHub documenti.

1. Visualizza la guida alle migliori pratiche EKS su. GitHub
2. Crea un fork del repository del progetto. Scopri come eseguire il fork di un repository nei documenti. GitHub
3. Clona il fork del repository del progetto. Scopri come clonare il tuo repository biforcato.

Apri l'area di lavoro VS Code

AWS consiglia di utilizzare Visual Studio Code di Microsoft per modificare la guida. Per ulteriori informazioni su VS Code, consulta Scarica Visual Studio Code e Inizia a usare Visual Studio Code nella documentazione di Visual Studio Code.

1. Apri VS Code.
2. Apri il `bpg-docs.code-workspace` file dal repository clonato.
3. Se è la prima volta che apri questo spazio di lavoro, accetta la richiesta di installare l'estensione. AsciiDoc Questa estensione controlla la sintassi dei AsciiDoc file e genera un'anteprima dal vivo.
4. Naviga fino alla `latest/bpg` directory. Questa directory contiene i file sorgente che vengono distribuiti nel sito di documentazione di AWS. I file sorgente sono organizzati per sezioni della guida, ad esempio «sicurezza» o «rete».

Modificare un file

1. Aprire un file nell'editor.
   - Visualizza la AsciiDoc sintassi per scoprire come creare titoli, link ed elenchi.
   - Puoi usare la sintassi Markdown per formattare il testo, creare elenchi e titoli. Non è possibile utilizzare la sintassi Markdown per creare collegamenti.
2. Apri un'anteprima dal vivo della pagina.
   - Per prima cosa, premi `ctrl-k` o `cmd-k` (a seconda della tastiera). In secondo luogo, premiv. Si apre un'anteprima in visualizzazione divisa.

AWS suggerisce di utilizzare i rami delle funzionalità per organizzare le modifiche. Scopri come creare rami con git.

Invia una Pull Request

Puoi creare una pull request dal GitHub sito Web o dalla GitHub cli.

Scopri come creare una pull request da un fork utilizzando il GitHub sito Web.

Scopri come creare una pull request utilizzando la GitHub cli.

Usa l'editor basato sul web github.dev

L'editor `github.dev` basato sul web è basato su VS Code. Questo è un ottimo modo per modificare più file e visualizzare in anteprima i contenuti senza alcuna configurazione.

Supporta l' AsciiDoc estensione. È possibile eseguire operazioni git utilizzando la GUI. L'editor basato sul web non dispone di una shell o di un terminale per l'esecuzione dei comandi.

È necessario disporre di un GitHub account. Se necessario, ti verrà richiesto di effettuare il login.

Avvia l'editor GitHub basato sul web.

Modifica una singola pagina

È possibile aggiornare rapidamente singole pagine utilizzando GitHub. Ogni pagina contiene un link "# Modifica questa pagina su GitHub" in basso.

1. Vai alla pagina di questa guida che desideri modificare
2. Fai clic sul link «Modifica questa pagina su GitHub» in basso
3. Fai clic sull'icona di modifica a forma di matita in alto a destra nel visualizzatore di GitHub file o premi e
4. Modifica il file
5. Invia le modifiche utilizzando il pulsante «Conferma modifiche...». Questo pulsante crea una GitHub pull request. I manutentori della guida esamineranno questa pull request. Un revisore approverà la pull request o richiederà delle modifiche.

Visualizza e imposta l'ID per una pagina

Questa pagina spiega come visualizzare e impostare l'ID della pagina.

L'ID della pagina è una stringa univoca che identifica ogni pagina del sito di documentazione. Puoi visualizzare l'ID della pagina nella barra degli indirizzi del browser quando ti trovi su una pagina specifica. L'ID della pagina viene utilizzato per l'URL, il nome del file e per creare link di riferimento incrociato.

Ad esempio, se stai visualizzando questa pagina, l'URL nella barra degli indirizzi del browser sarà simile a:

```
https://docs.aws.amazon.com/view-set-page-id.html
```

L'ultima parte dell'URL (`view-set-page-id`) è l'ID della pagina.

Imposta l'ID della pagina

Quando si crea una nuova pagina, è necessario impostare l'ID della pagina nel file sorgente. L'ID della pagina deve essere una stringa concisa e sillabata che descrive il contenuto della pagina.

1. Apri il file sorgente della nuova pagina in un editor di testo.
2. Nella parte superiore del file, aggiungi la riga seguente. Dovrebbe essere al di sopra del primo titolo.

   ```
   [#my-new-page]
   ```

   Sostituiscilo `my-new-page` con l'ID della tua nuova pagina.
3. Salvare il file.

> **Note**
>
> La pagina IDs deve essere unica in tutto il sito di documentazione. Se provi a utilizzare un ID di pagina esistente, riceverai un errore di compilazione.

Creazione di una nuova pagina

Scopri come creare una nuova pagina e aggiornare il sommario della guida.

Crea metadati di pagina

1. Determina il titolo della pagina e il titolo breve della pagina. Il titolo breve della pagina è facoltativo, ma consigliato se il titolo della pagina è composto da più di poche parole.
2. Determina l'ID della pagina. Questo deve essere unico all'interno della EKS Best Practices Guide. La convenzione prevede di utilizzare tutte le parole minuscole e separate con. -
3. Crea un nuovo file asciidoc, in una cartella se necessario, e aggiungi il seguente testo al file:

Example

[.» argomento "] [#<page-id>] = <page-title>:info_titleabbrev: < > page-short-title

Ad esempio,

Example

[.» topic "] [#scalability] = Migliori pratiche di scalabilità EKS:info_titleabbrev: Scalabilità

Aggiungi al sommario

1. Apri il file per la pagina principale nel sommario. Per le nuove sezioni della guida di primo livello, il file principale è`book.adoc`.
2. Nella parte inferiore del file principale, aggiorna e inserisci la seguente direttiva:

   Example

   include: <new-filename>[leveloffset=+1]

   Ad esempio,

   Example

   include: :dataplane.adoc [leveloffset=+1]

Inserisci un'immagine

1. Trova il prefisso dell'immagine per la pagina che stai modificando. Controlla la `:imagesdir:` proprietà nell'intestazione del file. Ad esempio, `` `:imagesdir: images/reliability/ ``
2. Posiziona l'immagine in questo percorso, ad esempio `latest/bpg/images/reliability`
3. Determina il testo alternativo appropriato per la tua immagine. Scrivi una breve descrizione di alto livello dell'immagine. Ad esempio, «diagramma del VPC con tre zone di disponibilità» è un testo alternativo appropriato.
4. Aggiorna il seguente esempio con il testo alternativo e il nome del file di immagine. Inserite nella posizione desiderata.

Example

immagine: <image-filename>[< image-alt-text >]

Ad esempio,

Example

immagine: eks-data-plane-connectivity .jpeg [Diagramma di rete]

Controlla lo stile con Vale

1. Installa la CLI Vale.
2. Esegui `vale sync`
3. Installa l'estensione Vale da Visual Studio Marketplace.
4. Riavvia VS Code e apri un AsciiDoc file
5. VS Code sottolinea il testo problematico. Scopri come lavorare con errori e avvisi nei documenti di VS Code.

Crea un'anteprima locale

1. Installa `asciidoctor` lo strumento utilizzando `brew` Linux o macOS
   - Scopri come installare asciidoctor cli nei documenti. AsciiDoctor
   - Scopri come installare il gestore di pacchetti brew.
2. Apri un terminale e vai a `latest/bpg/`
3. Esegui `asciidoctor book.adoc`
   - Esamina eventuali avvisi ed errori di sintassi
4. Apri il file di `book.html` output.
   - Su macOS, puoi eseguire `open book.html` per aprire l'anteprima nel tuo browser predefinito.

AsciiDoc Cheat Sheet

Formattazione di base

```
*bold text*
_italic text_
`monospace text`
```

Headers

```
= Document Title (Header 1)
== Header 2
=== Header 3
==== Header 4
===== Header 5
====== Header 6
```

Elenchi

Elenchi non ordinati:

```
- Item 1
- Item 2
-- Subitem 2.1
-- Subitem 2.2
- Item 3
```

Elenchi ordinati:

```
. First item
. Second item
.. Subitem 2.1
.. Subitem 2.2
. Third item
```

Link

```
External link:  https://example.com[Link text]
Internal link: <<page-id>>
```

```
Internal link: xref:page-id[Link text]
```

Immagini

```
image::image-file.jpg[Alt text]
```

Blocchi di codice

```
 [source,python]
 ----
 def hello_world():
     print("Hello, World!")
 ----
```

Tabelle

Scopri come creare una tabella di base.

```
[cols="1,1"]
|===
|Cell in column 1, row 1
|Cell in column 2, row 1

|Cell in column 1, row 2
|Cell in column 2, row 2

|Cell in column 1, row 3
|Cell in column 2, row 3
|===
```

Ammonizioni

```
NOTE: This is a note admonition.

WARNING: This is a warning admonition.

TIP: This is a tip admonition.

IMPORTANT: This is an important admonition.
```

```
CAUTION: This is a caution admonition.
```

Anteprima:

> **Note**
>
> Questa è una nota ammonizione.

Include

```
include::filename.adoc[]
```

Le traduzioni sono generate tramite traduzione automatica. In caso di conflitto tra il contenuto di una traduzione e la versione originale in Inglese, quest'ultima prevarrà.