AWS Well-Architected フレームワーク

ハイパフォーマンスコンピューティングレンズ

ハイパフォーマンスコンピューティングレンズ: AWS Well-Architected フレームワーク

Table of Contents

要約 1
要約 1
はじめに 2
定義 3
一般的な設計の原則 4
シナリオ 7
疎結合のシナリオ 9
密結合のシナリオ 10
リファレンスアーキテクチャ 11
従来のクラスター環境 11
バッチ処理ベースのアーキテクチャ 14
キューベースのアーキテクチャ 15
ハイブリッドなデプロイ 17
サーバーレス 19
Well-Architected フレームワークの 5 本の柱 22
運用上の優秀性の柱 22
設計の原則 22
ベストプラクティス 23
セキュリティの柱 25
設計の原則 25
定義 25
ベストプラクティス 26
信頼性の柱 28
設計の原則 29
定義 29
ベストプラクティス 30
パフォーマンス効率の柱 31
設計の原則 32
定義 33
ベストプラクティス 33
コスト最適化の柱 40
設計の原則 40
定義 41
ベストプラクティス 42

まとめ 44
寄稿者 45
その他の資料 46
改訂履歴 47
注意 48

ハイパフォーマンスコンピューティングレンズ - AWS Well-Architected フレームワーク

公開日: 2019 年 12 月 (改訂履歴)

要約

このドキュメントでは、AWS Well-Architected フレームワークのハイパフォーマンスコンピューティング (HPC) レンズについて説明しています。このドキュメントでは、一般的な HPC のシナリオについて説明し、ワークロードがベストプラクティスに従って設計されていることを確認するための重要な要素を示しています。

はじめに

AWS Well-Architected フレームワークの目的は、AWS でシステムを構築する際の選択肢の#所と短所をお客様が理解できるように支援することです。フレームワークを使用することによって、信頼性が高く、安全かつ効率的で、コスト効率に優れたシステムをクラウド内で設計および運用するためのアーキテクチャ上のベストプラクティスを学びます。フレームワークにより、アーキテクチャをベストプラクティスに照らし合わせて一貫的に測定し、改善すべき点を特定する手段を提供します。Well-Architected システムによってビジネスの成功の可能性が大いに高まると当社は確信しています。

この「レンズ」では、AWS クラウドでハイパフォーマンスコンピューティング (HPC) ワークロードを設計、デプロイ、および構築する方法に焦点を当てています。クラウドでは、HPC ワークロードは極めてよく動作します。HPC ワークロードはその自然な起伏やバースト特性により、従量制のクラウドインフラストラクチャに適しています。クラウドリソースを微調整し、クラウドネイティブアーキテクチャを作成する機能で、HPC ワークロードの所要時間が自然に加速します。

ここでは簡潔に、HPC ワークロードに固有の Well-Architected フレームワークの詳細のみを取り上げています。アーキテクチャを設計する際には AWS Well-Architected フレームワークのホワイトペーパーを読み、ベストプラクティスや質問を検討することをお勧めします。

このホワイトペーパーの対象者は、最高技術責任者 (CTO)、アーキテクト、デベロッパー、オペレーションチームメンバーなどの技術担当者です。このホワイトペーパーを読むことで、クラウド環境における HPC の設計や運用の際に活用する AWS のベストプラクティスと戦略を理解することができます。

定義

AWS Well-Architected フレームワークは、運用上の優秀性、セキュリティ、信頼性、パフォーマンス効率、コスト最適化という 5 本の柱を基本としています。ソリューションを設計する際に、ビジネスの状況に応じて 5 本の柱の間でトレードオフを行います。これらのビジネス上の決定により、エンジニアリングの優先順位を決めることができます。開発環境では信頼性を犠牲にすることでコストを削減する、あるいはミッションクリティカルなソリューションでは信頼性を最適化するためにコストをかけることがあります。セキュリティと運用性は、通常、他の柱とトレードオフされることはありません。

この文書では、疎結合ワークロード (高スループットコンピューティング (HTC) と呼ぶコミュニティもあります) と密結合ワークロードをはっきりと区別しています。サーバーベースおよびサーバーレスの設計も網羅しています。これらの区別の詳細については、シナリオのセクションをご参照ください。

AWS クラウドの一部の語彙は、一般的な HPC 用語とは異なる場合があります。例えば、HPC ユーザーはサーバーを「ノード」と言い、AWS は仮想サーバーを「インスタンス」と言います。 HPC ユーザーが一般的に「ジョブ」と言うものは、AWS では「ワークロード」と言います。

AWS ドキュメントでは、「vCPU」という用語を「スレッド」または「ハイパースレッド」 (あるいは物理コアの半分) と同義で使用しています。AWS で HPC アプリケーションのパフォーマンスまたはコストを定量化する際には、この 2 点にお気をつけください。

クラスタープレイスメントグループは、ネットワーク要件が最も高いアプリケーションでのコンピューティングインスタンスをグループ化する AWS のメソッドです。プレイスメントグループは物理的なハードウェア要素ではありません。ネットワークの低レイテンシー半径内にすべてのノードを保持する、簡単な論理ルールです。

AWS クラウドインフラストラクチャはリージョンとアベイラビリティーゾーンを中心として構築されます。リージョンは世界中の物理的場所であり、複数のアベイラビリティーゾーンが配置されています。アベイラビリティーゾーンは 1 つ以上の独立したデータセンターで構成されます。各データセンターは、冗長性のある電源、ネットワーキング、接続を備えており、別々の設備に収容されています。HPC ワークロードの特性に応じて、クラスターをアベイラビリティーゾーンに広げる (信頼性を高める) か、単一のアベイラビリティーゾーン内にとどめる (低レイテンシーを重視する) ことができます。

一般的な設計の原則

従来のコンピューティング環境では、アーキテクチャの決定は静的に 1 回限りのイベントとして実装されることが多く、コンピューティングシステムの存続期間中にソフトウェアやハードウェアの主要な更新が行われないこともあります。プロジェクトやその状況が変化し続けるにつれて、当初の決定が原因で、変化するビジネス要件にシステムが対応できなくなる可能性があります。

クラウドではそんなことはありません。クラウドインフラストラクチャは、プロジェクトの成長に合わせて拡大できるため、最適化された機能が継続的に可能になります。クラウドでは自動化やオンデマンドのテストが可能なため、インフラストラクチャの設計変更によって#じる影響のリスクを軽減できます。このため、システムを時間とともに進化させ、イノベーションを標準プラクティスとしてプロジェクトで活用できるようになります。

Well-Architected フレームワークは一般的な設計の原則を提供することにより、ハイパフォーマンスコンピューティングを使ってクラウドでの優れた設計を促進します。

- 動的アーキテクチャ: 安定状態モデルを使用する固定化した静的なアーキテクチャとコストの見積もりは避けてください。アーキテクチャは動的である必要があります。HPC に対する要求に応じて、時間とともに成長および縮小するためです。アーキテクチャ設計とコスト分析を、HPC アクティビティの自然なサイクルに明示的に一致させてください。たとえば、集中的なシミュレーション作業の後、作業が設計段階からラボに移行するにつれて需要が減少する場合があります。または、長期的に安定したデータ蓄積フェーズの後に、大規模な分析やデータ削減フェーズが続く場合があります。多くの従来のスーパーコンピューティングセンターとは異なり、AWS クラウドでは、長いキュー、長時間のクォータアプリケーション、カスタマイズやソフトウェアインストールの制限を回避する支援を行います。HPCの取り組みの多くは本質的にバースト性があるため、順応性と従量制を持つクラウドのパラダイムによく適合します。AWS の順応性のある従量制モデルにより、オーバーサブスクライブシステム (キューで待機させられる) か、アイドルシステム (お金を無駄にする) かという苦渋の選択がなくなります。コンピューティングクラスターなどの環境を、特定のニーズに合わせていつでも「適切なサイズ」にすることができます。
- 調達モデルをワークロードにあわせる: AWS では、さまざまな HPC 使用パターンに対応したいろんなコンピューティング調達モデルを提供しています。適切なモデルを選択することで、必要なものだけに料金を支払うことができます。例えば、研究機関は同じ天気予報アプリケーションを異なる方法で実行することがあります。
  - パラメータの一括処理やアンサンブルが多い気象変数の役割を調査している学術研究プロジェクトがあります。これらのシミュレーションで懸念するべきことは、緊急性ではなくコストです。そのため、Amazon EC2 スポットインスタンスが最適です。スポットインスタンスでは

Amazon EC2 の未使用キャパシティーを利用でき、オンデマンド価格と比較して最大 90% 安くなります。

  - 山火事のシーズンには、現地の風に関する予報を随時更新することで、消防士の安全が確保されます。シミュレーションが 1 分でも遅延するごとに、安全に避難できる機会が減ってしまいます。そのため、オンデマンドインスタンスをこれらのシミュレーションに使用して、分析のバーストを可能にし、中断なく結果が得られるようにする必要があります。
  - 毎朝、午後のテレビ放送用に天気の予測が行われます。予定したリザーブドインスタンスを使用すれば、必要なキャパシティーを毎日適切な時間に利用できるようにすることが可能です。この料金設定モデルを使用すると、オンデマンドインスタンスよりも割り引いた料金になります。
- データから開始する: アーキテクチャの設計を始める前に、データの明確な全体像を把握する必要があります。データの出所、サイズ、速度、更新を考慮してください。パフォーマンスとコストの全体的な最適化では、コンピューティングに重点を置き、データを考慮してください。AWS には、データの視覚化を含む強力なデータと関連サービスがあるため、データから最大限の価値を引き出すことができます。
- アーキテクチャ実験を簡素化する自動化: コードによる自動化で、低コストでシステムを作成およびレプリケートすると、手作業でかかるコストを回避できます。コードの変更を追跡し、影響を監査して、必要な場合は以前のバージョンに戻すことができます。インフラストラクチャを簡単に実験できるため、パフォーマンスとコストを考慮してアーキテクチャを最適化できます。AWS は AWS ParallelCluster などのツールを提供しており、コードとしての HPC クラウドインフラストラクチャで開始できます。
- コラボレーションの有効化: HPC は世界中の多くの国にまたがって共同作業することがほとんどです。直接の共同作業よりも、より広い HPC と科学のコミュニティの間で方法と結果を共有することがよくあります。そのため、どのツール、コード、データを誰と共有するかを事前に考慮することが重要となります。配信方法は、この設計プロセスの一部である必要があります。例えば、ワークフローは AWS でさまざまな方法で共有できます。Amazon マシンイメージ (AMI)、Amazon Elastic Block Store (Amazon EBS) スナップショット、Amazon Simple Storage Service (Amazon S3) バケット、AWS CloudFormation テンプレート、AWS ParallelCluster 設定ファイル、AWS Marketplace 製品、およびスクリプトを使用できます。AWS セキュリティとコラボレーション機能を最大限に活用すれば、AWS がお客様と共同作業者にとって優れた環境となり、HPC の問題を解決できます。これにより、コンピューティングソリューションとデータセットの影響が拡大し、選択したグループ内で安全に共有したり、より広範なコミュニティと共有して公開したりできるようになります。
- クラウドネイティブ設計を使用する: 通常、ワークロードを AWS に移行する際にオンプレミス環境をレプリケートする必要はありませんし、最適な方法とは言えません。AWS のサービスが持つ広さと深さのおかげで、新しい設計パターンとクラウドネイティブソリューションを使用

し、HPC ワークロードを新たな方法で実行できます。 例えば、ユーザーまたはグループはそれぞれ個別のクラスターを使用できます。クラスターは負荷に応じて独立してスケーリングできます。ユーザーは、AWS Batch などのマネージドサービス、または AWS Lambda などのサーバーレスコンピューティングに依存し、基盤となるインフラストラクチャを管理できます。従来のクラスタースケジューラを使用しないことを検討し、ワークロードで必要な場合にのみスケジューラを使用してください。クラウドでは HPC クラスターは永続性を必要とせず、一時的なリソースになる場合があります。クラスターのデプロイを自動化すると、1 つのクラスターを終了し、同じパラメータまたは異なるパラメータを使用して新しいクラスターをすばやく起動できます。この方法は、必要に応じて環境を作成します。

- 実際のワークロードをテストする: クラウドでのテストが、本番ワークロードがクラウド上でどのように実行されるかを知る唯一の方法です。大抵の HPC アプリケーションは複雑で、メモリ、CPU、ネットワークパターンを減らして簡単なテストにすることはほとんどの場合できません。また、インフラストラクチャのアプリケーション要件は、モデルが使用するアプリケーションソルバー (数学的な方法またはアルゴリズム)、モデルのサイズと複雑さなどによって異なります。このため、一般的なベンチマークは実際の HPC の本番パフォーマンスの信頼できる予測因子にはなりません。同様に、小さなベンチマークセットまたは「トイプロブレム (問題を非常に単純化した例題)」でアプリケーションをテストしても、ほとんど意味がありません。 AWS では、実際のご使用分のみお支払いいただきます。そのため、お客様独自の代表モデルを使用して、現実的な概念実証を行うことが可能です。クラウドベースのプラットフォームの主な利点は、移行前に現実的かつ本格的なテストを実行できることにあります。
- 結果までにかかる時間とコスト削減のバランスを取る: 最も重要なパラメータである時間とコストを使って、パフォーマンスを分析します。時間に依存しないワークロードには、コストの最適化に取り組む必要があります。スポットインスタンスは、通常、タイムクリティカルではないワークロードの中で最も安価な方法です。たとえば、ある研究者が来年の会議の前に分析しなければならないラボ測定値が多数ある場合、スポットインスタンスでは決められた研究予算内で可能な限り多くの測定値を分析することができます。逆に、緊急対応モデリングなどのタイムクリティカルなワークロードの場合、パフォーマンスと引き換えにコストの最適化が得られるため、インスタンスタイプ、調達モデル、クラスターサイズは、最短かつ最も迅速な実行時間を選択する必要があります。プラットフォームを比較する場合、リソースのプロビジョニング、データのステージング、または従来の環境ではジョブキューに費やされた時間などの非コンピューティング要素を含む、解決までの時間全体を考慮することが重要です。

シナリオ

HPC のケースでの問題は通常、並列処理技術を必要とする複雑なコンピューティングにあります。Well-Architected HPC インフラストラクチャは、計算を行う間、パフォーマンスを維持し、計算を支援できます。HPC ワークロードは、ゲノミクス、計算化学、金融リスクモデリング、コンピューター支援エンジニアリング、天気予報、地震イメージングなどの従来のアプリケーションに加え、機械学習、ディープラーニング、自動運転などの新しいアプリケーションにも及びます。さらに、これらの計算をサポートする従来のグリッドまたは HPC クラスターは、特定のワークロード用に最適化した一部のクラスター属性を持つアーキテクチャに著しく類似しています。AWS では、ネットワーク、ストレージタイプ、コンピューティング (インスタンス) タイプ、さらにデプロイ方法も戦略的に選択し、特定のワークロードのパフォーマンス、コスト、使いやすさを最適化できます。

HPC は、同時に実行している並列処理間における相互作用の程度に基づいて、疎結合ワークロードと密結合ワークロードの 2 つのカテゴリに分類できます。疎結合の HPC はシミュレーション全体の過程で、複数処理または並列処理が互いに強く作用しません。密結合 HPC では並列処理を同時に実行し、シミュレーションの各反復またはステップで相互に定期的に情報を交換します。

疎結合のワークロードでは、計算またはシミュレーション全体を完了するには、数百から数百万の並列処理を必要とすることがよくあります。これらのプロセスは、シミュレーション中に任意の順序と速度で発生します。このため、疎結合シミュレーションに必要なコンピューティングインフラストラクチャの柔軟性を提供できます。

密結合ワークロードには、シミュレーションの各反復で定期的に情報を交換するプロセスがあります。通常、これらの密結合シミュレーションは同種のクラスターで実行されます。コアまたはプロセッサの合計数は、インフラストラクチャで許可される場合、数十から数千、場合によっては数十万の範囲になります。シミュレーション中での処理の相互作用により、コンピューティングノードやネットワークインフラストラクチャなどのインフラストラクチャに余分な負荷がかかります。

さまざまな疎結合および密結合アプリケーションを実行するために使用するインフラストラクチャは、ノード間処理の相互作用の能力によって区別されます。両方のシナリオに当てはまる基本的な側面がある一方、それぞれのシナリオに特有な設計上の考慮事項があります。AWS で HPC インフラストラクチャを選択する際は、両方のシナリオについて次の基本事項を考慮してください。

- ネットワーク: ネットワーク要件は、通信トラフィックが最も少ない疎結合アプリケーションのような要件の低いケースから、帯域幅が広くレイテンシーの低い高性能ネットワークを必要とする密結合および超並列アプリケーションにいたるまでさまざまです。

- ストレージ: HPC 計算では、データを独自の方法で使用、作成、移動します。ストレージインフラストラクチャは、計算の各ステップでこれらの要件をサポートする必要があります。入力データは起動時に保存されることがほとんどで、実行中により多くのデータが作成および保存され、実行が完了すると出力データはリザーバがある場所に移動します。考慮すべき要素には、データサイズ、メディアタイプ、転送速度、共有アクセス、ストレージプロパティ (耐久性や可用性など) が含まれます。ノード間で共有ファイルシステムを使用すると便利です。例えば、Amazon Elastic File System (EFS) などのネットワークファイルシステム (NFS) 共有、または Amazon FSx for Lustre などの Lustre ファイルシステムです。
- コンピューティング: Amazon EC2 インスタンスタイプは HPC ワークロードで利用可能なハードウェア機能を定義します。ハードウェア機能には、プロセッサタイプ、コア周波数、プロセッサ機能 (ベクトル拡張など)、メモリとコアの比率、ネットワークパフォーマンスが含まれます。AWS では、インスタンスを HPC ノードと同じものと見なします。これらの用語は、本ホワイトペーパーでも同じ意味として使用します。
  - AWS は、基盤となる EC2 インスタンスタイプを選択することなくコンピューティングにアクセスする機能を備えたマネージドサービスを提供します。AWS Lambda および AWS Fargate は、基盤となるサーバーをプロビジョニングおよび管理することなくワークロードを実行できるようにするコンピューティングサービスです。
- デプロイ: AWS には、HPC ワークロードをデプロイするための多くのオプションがあります。インスタンスは、AWS マネジメントコンソール から手動で起動できます。自動デプロイでは、いろんなプログラミング言語でエンドツーエンドのソリューションをコーディングできるさまざまなソフトウェア開発キット (SDK) を利用できます。bash シェルスクリプトと AWS コマンドラインインターフェイス (AWS CLI) を組み合わせたものは、人気のある HPC デプロイのオプションです。
  - AWS CloudFormation テンプレートを使用すると、コードとして記述されたアプリケーションに合わせた HPC クラスターを指定できるため、数分で起動できます。AWS ParallelCluster は、CloudFormation を介したクラスターの起動を、従来のクラスターエクスペリエンス用に既にインストールされているソフトウェア (コンパイラやスケジューラなど) と調整するオープンソースソフトウェアです。
  - AWS は、Amazon Elastic Container Service (Amazon ECS)、Amazon Elastic Kubernetes Service (Amazon EKS)、AWS Fargate、AWS Batch などのコンテナベースのワークロードにマネージドデプロイメントサービスを提供します。
  - AWS Marketplace や AWS パートナーネットワーク (APN) より、追加のサードパーティー製ソフトウェアオプションも利用できます。

クラウドコンピューティングにより、インフラストラクチャコンポーネントとアーキテクチャ設計を簡単に実験できます。AWS では、インスタンスタイプ、EBS ボリュームタイプ、デプロイ方法などをテストして、最小のコストで最高のパフォーマンスを見つけることを強くお勧めします。

疎結合のシナリオ

疎結合ワークロードは、小さなジョブの処理を多数伴います。通常は小さなジョブを 1 つのノードで実行し、1 つのプロセス、またはそのノード内の並列化のための共有メモリ並列化 (SMP) を持つ複数のプロセスを使用します。

並列処理またはシミュレーションでの反復を後処理して、シミュレーションから 1 つのソリューションまたは検出を作成します。疎結合アプリケーションは、モンテカルロシミュレーション、画像処理、ゲノミクス解析、電子設計自動化 (EDA) など、多くの分野で使用されています。

疎結合のワークロードで 1 つのノードまたはジョブが失われても、通常は計算全体が遅れることはありません。失われた作業は後で取得するか、完全に省略することができます。計算に関連するノードは、仕様と出力が異なる場合があります。

疎結合ワークロードに適したアーキテクチャでは、次のことを考慮してください。

- ネットワーク: 並列プロセスが相互に作用することは通常ないため、ワークロードの実行可能性またはパフォーマンスは、インスタンス間のネットワークの帯域幅や待機時間の影響を受けません。したがって、クラスター化されたプレースメントグループはパフォーマンスを向上させることなく回復性を弱めるため、このシナリオでは必要とされません。
- ストレージ: 疎結合ワークロードはストレージ要件が異なり、データセットのサイズと希望するデータ転送とデータの読み書きパフォーマンスによって変わります。
- コンピューティング: それぞれのアプリケーションにより異なりますが、全般的にアプリケーションにおけるコンピューティングに対するメモリの比率は、元になっている EC2 インスタンスタイプによります。いくつかのアプリケーションは、EC2 インスタンスの画像処理用演算プロセッサ (GPU) や フィールドプログラマブルゲートアレイ (FPGA) のアクセラレーターを活用するように最適化されています。
- デプロイ: 疎結合シミュレーションは、しばしば多くの (時には数百万の) コンピューティングコアで実行します。コンピューティングコアは、パフォーマンスを犠牲にすることなく、アベイラビリティゾーンをまたいで実行することも可能です。疎結合シミュレーションは、AWS Batch や AWS ParallelCluster などのエンドツーエンドのサービスとソリューションを使用して、または Amazon Simple Queue Service (Amazon SQS)、Auto Scaling、AWS Lambda、AWS Step Functions などの AWS のサービスの組み合わせを介して、デプロイできます。

密結合のシナリオ

密結合アプリケーションは、相互に依存して計算を実行する並列プロセスで構成されています。疎結合コンピュテーションとは異なり、密結合シミュレーションでは、すべてのプロセスが連携して反復するので、お互い通信が必要です。1回の反復は、シミュレーション全体の 1 ステップとして定義されています。密結合計算は、数百から数百万回を超える反復を、数十から数千のプロセスまたはコアに依存しています。通常、1 つのノードで計算が失敗すると、全体としての計算の失敗につながります。完全にエラーとなるリスクを避けるために、コンピュテーションの間にアプリケーションレベルでのチェックポイントを定期的に作成し、確実な状態からシミュレーションを再開できるようにします。

これらのシミュレーションは、プロセス間通信にメッセージパッシングインターフェイス (MPI) を使用しています。OpenMP 経由での Shared Memory Parallelism は、MPI とともに使われます。密結合 HPC ワークロードの例としては、計算流体力学、気象予測、油層シミュレーションなどがあります。

密結合 HPC ワークロードに適合したアーキテクチャでは、次のことを考慮します。

- ネットワーク: 密結合計算に必要なネットワーク要件があります。ノード間の通信が遅いと、結果として全体の計算が遅くなります。ネットワークのパフォーマンスを一定に保つには、最大のインスタンスサイズ、強化されたネットワーク、そしてクラスターのグループ配置が必要です。これらのテクニックは、シミュレーションの実行時間を最小化し、全体的なコストを軽減します。密結合アプリケーションのサイズはさまざまです。多数のプロセスまたはコアにまたがる大きな処理サイズの場合、通常はうまく並列化します。全体的な計算要件が低い小規模なケースでは、ネットワークに対する要求が最も大きくなります。特定の Amazon EC2 インスタンスでは、Elastic Fabric Adapter (EFA) をネットワークインターフェイスとして使用することで、AWS の規模で高レベルのノード間通信が必要なアプリケーションを実行できます。EFA のカスタムビルドしたオペレーティングシステムのバイパスハードウェアインターフェイスにより、インスタンス間通信のパフォーマンスが向上します。これは密結合アプリケーションをスケールするときに重要です。
- ストレージ: 密結合ワークロードのストレージ要件が様々ありますが、データセットのサイズや、データ転送、読み込み、書き込みに必要なパフォーマンスによって決定します。一時的なデータストレージや、スクラッチスペースには特別な配慮が必要です。
- コンピューティング: EC2 インスタンスは、変更可能なコアとメモリレシオによる様々な設定で提供されています。並列アプリケーションの場合、メモリ集約型の並列シミュレーションをより多くのコンピューティングノードに分散して、コアあたりのメモリ要件を減らし、最高のパフォーマンスを発揮するインスタンスタイプをターゲットにします。密結合アプリケーションでは、似たようなコンピュートノードから構成される同種のクラスターが必要です。最大のインスタンスサイズを

ターゲットにすると、ノード間のネットワークレイテンシーが最小限に抑えられ、ノード間の通信では最高のネットワークパフォーマンスを提供できます。

- デプロイ: 様々なデプロイのオプションが利用できます。「従来の」クラスター環境でシミュレーターを起動するような、エンドツーエンドの自動化が実現可能です。クラウドのスケーラビリティによって、数百もの大規模なマルチプロセスを一度に起動することができます。キューで待機する必要はありません。密結合シミュレーションは、AWS Batch や AWS ParallelCluster などのエンドツーエンドのソリューションとともにデプロイできます。また、CloudFormation や EC2 Fleet などの、AWS のサービスに基づいたソリューションを通してデプロイすることもできます。

リファレンスアーキテクチャ

多くのアーキテクチャは疎結合ワークロードと密結合ワークロードのどちらにも適用されますが、シナリオによって若干の修正が必要です。従来のオンプレミスのクラスターでは、クラスターインフラストラクチャに万能のアプローチが強制されました。しかし、クラウドでは幅広い可能性を提供でき、パフォーマンスとコストの最適化が可能です。スケジューラとログインノードによる従来のクラスターエクスペリエンスから、クラウドネイティブなソリューションによる、コスト効率化の強みを備えたクラウドネイティブなアーキテクチャまで、クラウドでは幅広い設定ができます。次に 5 つのリファレンスアーキテクチャを挙げます。

トピック

- 従来のクラスター環境
- バッチ処理ベースのアーキテクチャ
- キューベースのアーキテクチャ
- ハイブリッドなデプロイ
- サーバーレス

従来のクラスター環境

多くのユーザーが、従来の HPC 環境に似た環境でクラウドジャーニーを開始します。このような環境は多くの場合、ジョブを起動するためにログインノードを必要とします。

従来のクラスタープロビジョニングへの一般的なアプローチは、ユーザーの特定のタスクのカスタマイズと組み合わせたコンピューティングクラスターの AWS CloudFormation テンプレートに基づいています。AWS ParallelCluster は、AWS CloudFormation に基づくエンドツーエンドのクラスタープロビジョニング機能の例です。アーキテクチャの複雑な説明はテンプレート内に隠匿されてはいま

すが、一般的な設定オプションにより、ユーザーはインスタンスタイプ、スケジューラ、またはアプリケーションのインストールやデータの同期などのブートストラップアクションを選択できます。テンプレートは、従来の HPC クラスターの「ルックアンドフィール」を備えた HPC 環境を提供するように構築および実行できますが、さらにスケーラビリティのメリットも追加されています。ログインノードは、スケジューラ、共有ファイルシステム、および実行環境を管理します。その一方でジョブがキューに送信されると、自動スケーリング機能により、追加のインスタンスが自動でスピンアップされます。インスタンスがアイドルになると、それらは自動的に終了します。

クラスターは永続的な設定でデプロイするか、一時リソースとして扱うことができます。永続的なクラスターは、ログインインスタンスと、固定サイズのコンピューティングフリート、または送信されたジョブの数に応じてコンピューティングフリートを増減する Auto Scaling グループに関連付けられたコンピューティングフリートで、デプロイされます。永続的クラスターでは、常に何らかのインフラストラクチャが実行されています。もしくは、クラスターは、各ワークロードが独自のクラスターで実行される一時的なものとして扱うことができます。一時的なクラスターは、自動化により有効化されます。例えば、bash スクリプトを AWS CLI と組み合わせたり、Python スクリプトと AWS SDK を組み合わせて、エンドツーエンドのケース自動化を実現します。それぞれのケースで、リソースのプロビジョニングと起動が行われ、ノードにデータが配置され、複数のノードでジョブが実行され、ケースの出力は自動的に取得されるか、Amazon S3 に自動的に送信されます。ジョブが完了すると、インフラストラクチャは終了します。これらのクラスターはインフラストラクチャをコードとして扱い、コストを最適化します。また、インフラストラクチャの変更を完全にバージョン管理できます。

従来のクラスターのアーキテクチャを、疎結合と密結合ワークロードに使用できます。最適なパフォーマンスを得るために、密結合ワークロードでは、同種のインスタンスタイプを持つクラスター化された配置グループでコンピューティングフリートを使用する必要があります。

リファレンスアーキテクチャ

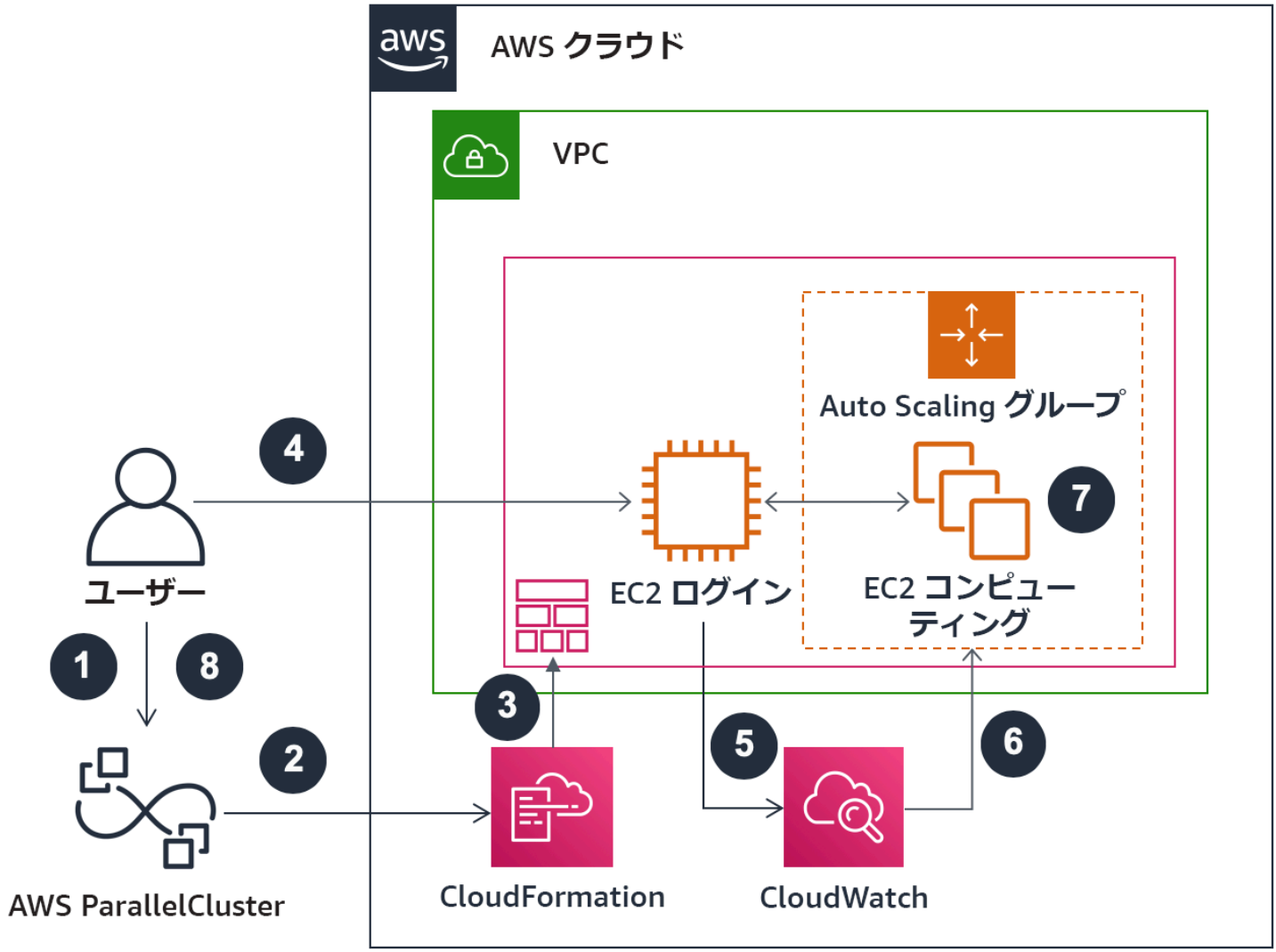

図 1: AWS ParallelCluster でデプロイされた従来のクラスター

ワークフローのステップ:

1. ユーザーは、AWS ParallelCluster CLI および設定ファイルの仕様を使用して、クラスターの作成を開始します。
2. AWS CloudFormation は、クラスターのテンプレートファイルの記述に従い、クラスターアーキテクチャを構築します。クラスターテンプレートファイルには、ユーザーによるカスタムの設定 (例えば、設定ファイルの編集やウェブインターフェイスの使用など) が提供されています。
3. AWS CloudFormation が、カスタマイズされた HPC ソフトウェアやアプリケーションで作成された EBS スナップショットから、インフラストラクチャをデプロイします。HPC ソフトウェアやアプリケーションには、NFS エクスポートを通してクラスターインスタンスがアクセスできます。

4. ユーザーがログインインスタンスにログインし、スケジューラ (SGE や Slurm など) にジョブを送信します。
5. ログインインスタンスは、ジョブのキューのサイズに基づき、CloudWatch にメトリクスを発行します。
6. CloudWatch は、ジョブのキューのサイズがしきい値を超えている場合には、コンピューティングインスタンスの数を増やすために Auto Scaling イベントをトリガーします。
7. コンピューティングフリートにより、スケジュールされたジョブが処理されます。
8. [オプション] ユーザーがすべてのリソースのクラスター削除と終了を開始します。

バッチ処理ベースのアーキテクチャ

AWS Batch はフルマネージドサービスであり、リソースをプロビジョニングしたりスケジューラを管理したりすることなく、クラウドで大規模なコンピューティングワークロードを実行できます。AWS Batchを使用することで、デベロッパー、科学者、およびエンジニアが AWS で数十万のバッチコンピューティングジョブを簡単かつ効率的に実行できるようになります。AWS Batch は、送信されたバッチジョブのボリュームと指定されたリソース要件に基づいて、最適な量とタイプのコンピューティングリソース (CPU やメモリに最適化されたインスタンスなど) を動的にプロビジョニングします。AWS Batch では、Amazon EC2 やスポットインスタンスなどの幅広い AWS コンピューティングサービスや機能、およびバッチコンピューティングワークロードを計画、スケジュール、実行します。ジョブの実行に必要なバッチコンピューティングのソフトウェアやサーバークラスターをインストールしたり、管理したりする必要がないため、結果の分析や新しい洞察を得ることに集中できます。

AWS Batch でアプリケーションをコンテナにパッケージし、ジョブの依存性を指定し、AWS マネジメントコンソール や CLI、SDK を使ってバッチジョブを送信します。実行パラメータとジョブの依存関係を指定し、一般的なバッチコンピューティングワークフローエンジンと言語 (例えば、Pegasus WMS、Luigi、AWS Step Functions など) と統合できます。AWS Batch は、デフォルトのジョブキューと計算環境の定義を提供し、すぐに開始できるようにします。

AWS Batch ベースのアーキテクチャは、疎結合と密結合、両方のワークロードに使用できます。密結合ワークロードでは、AWS Batch でマルチノードの並列ジョブを使用する必要があります。

リファレンスアーキテクチャ

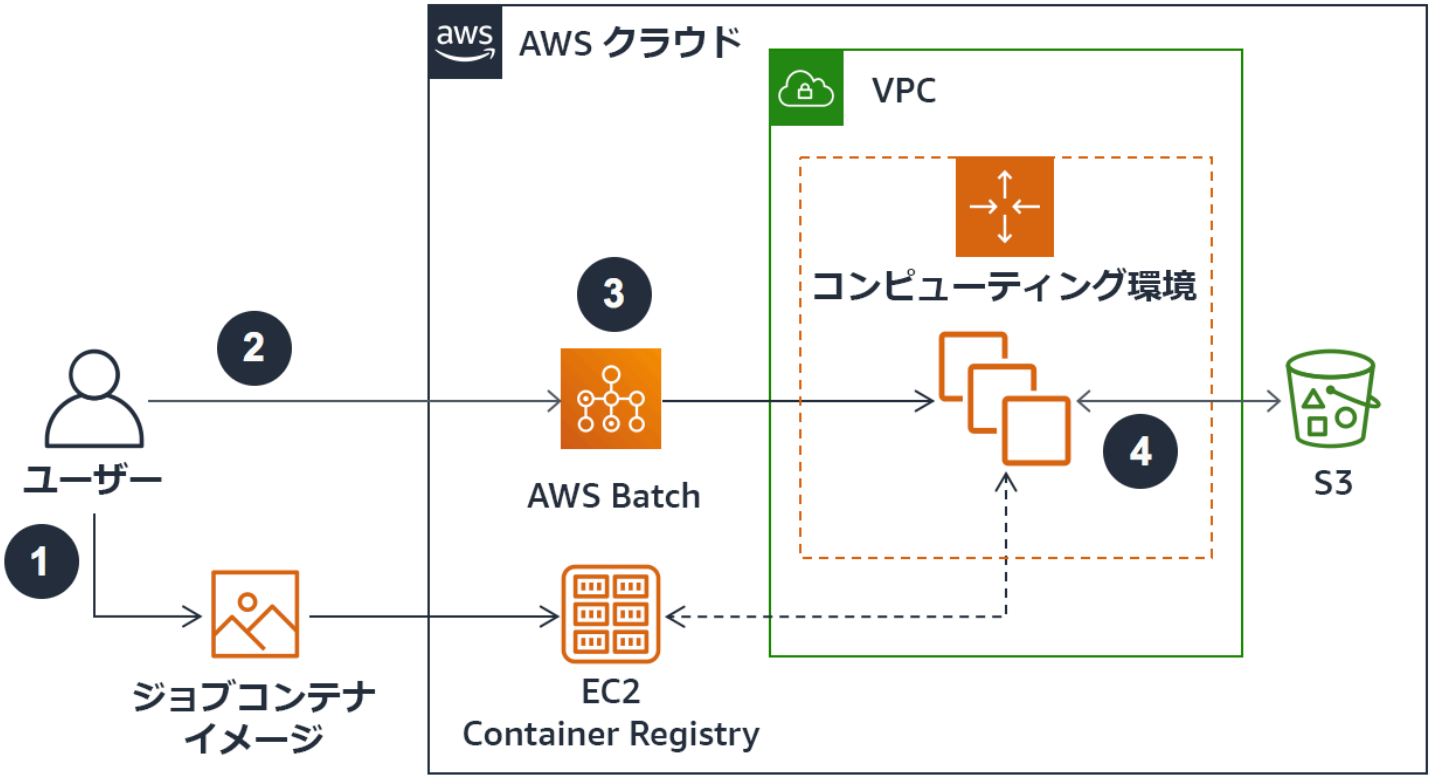

図 2: AWS Batch アーキテクチャの例

ワークフローのステップ:

1. ユーザーがジョブのコンテナを作成し、コンテナを Amazon EC2 Container Registry (Amazon ECR) または他のコンテナレジストリ (例えば DockerHub) にアップロードします。また、AWS Batch にジョブの定義を作成します。
2. ユーザーが AWS Batch のジョブキューにジョブを送信します。
3. AWS Batch がコンテナレジストリからイメージを取得し、キューにあるジョブを処理します。
4. 各ジョブからの入力データと出力データは、S3 バケットに保存されます。

キューベースのアーキテクチャ

Amazon SQS はフルマネージド型のメッセージキューのサービスです。これにより、コンピューティングのステップおよび後処理のステップからの、処理前ステップの疎結合化が簡単になります。独自の機能を実行する個別のコンポーネントからアプリケーションを構築することで、スケーラビリティと信頼性が向上します。コンポーネントの疎結合化は、最新のアプリケーションを設計するためのベストプラクティスです。Amazon SQS は多くの場合、クラウドネイティブの疎結合ソリューションの中心にあります。

Amazon SQS はしばしば、AWS CLI や AWS SDK のスクリプトによるソリューションと共に編成され、ユーザーはデスクトップから AWS のコンポーネントと直接やりとりすることなしにアプリケーションをデプロイできます。AWS Batch のようなサービスマネージドなデプロイに対して、SQS や EC2 によるキューベースのアーキテクチャでは、セルフマネージドのコンピューティングインフラストラクチャを必要とします。

キューベースのアーキテクチャは疎結合ワークロードには最良ですが、密結合ワークロードに適用すると、たちまち複雑になってしまうことがあります。

リファレンスアーキテクチャ

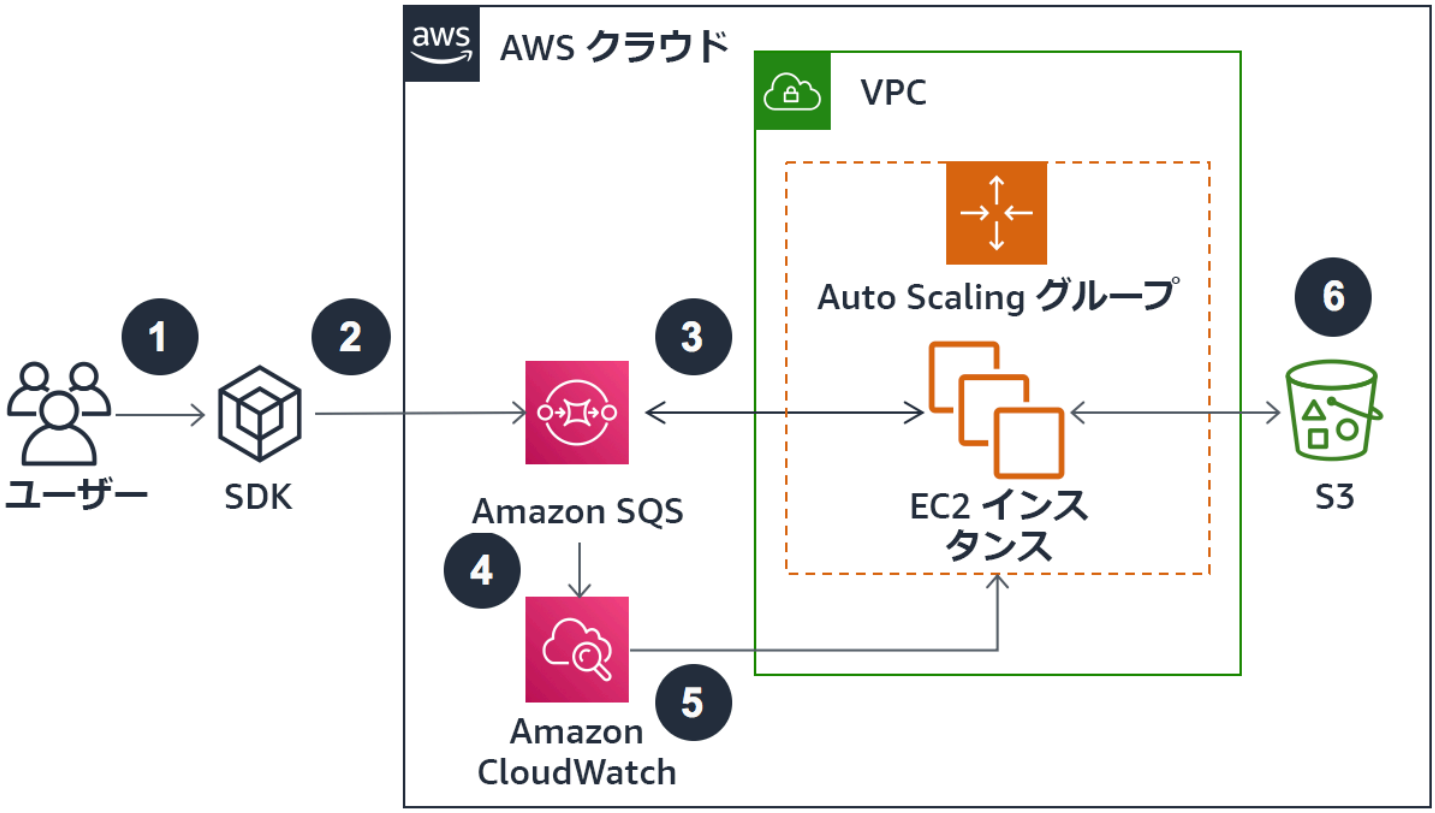

図 3: 疎結合のワークロード用にデプロイされた Amazon SQS

ワークフローのステップ:

1. 複数のユーザーが、AWS CLI や SDK でジョブを送信します。
2. ジョブが Amazon SQS のキューにメッセージとして追加されます。
3. EC2 インスタンスがキューをポーリングして、ジョブの処理を始めます。
4. Amazon SQS は、キューにあるメッセージ (ジョブ) の数に基づき、メトリクスを発行します。

5. キューが指定した長さよりも長くなくなったときに Auto Scaling に通知するための Amazon CloudWatch アラームが設定されます。Auto Scaling が EC2 インスタンスの数を増やします。
6. EC2 インスタンスがソースデータを引き出し、結果データを S3 バケットに保管します。

ハイブリッドなデプロイ

ハイブリッドデプロイは、主にオンプレミスのインフラストラクチャに投資をし、AWS を使用したい組織によって検討されます。このアプローチにより、組織はオンプレミスのリソースを増強し、すぐに AWS へ完全に移行するのではなく、AWS への代替パスを作っておくことができます。

ハイブリッドなシナリオは、ワークロードの分離といった最小限な調整から、スケジューラによるジョブ配置などの緊密に統合されたアプローチまでさまざまです。例えば、組織はワークロードを分離し、特定の種類のワークロードを AWS のインフラストラクチャで実行することがあります。もしくは、オンプレミスの処理とインフラストラクチャーに多額の投資をしている組織では、ジョブをスケジューリングするソフトウェアや、ジョブを送信するポータルを AWS リソースと一緒に管理することで、よりシームレスなエンドユーザー体験を望むかもしれません。商用またはオープンソースのいくつかのジョブスケジューラーは、必要に応じて AWS のリソースを動的にプロビジョニングしたり、プロビジョニングを解除する機能を提供しています。ベースとなるリソースマネジメントはネイティブな AWS インテグレーション (例えば、AWS CLI や API) に依存し、スケジューラーによっては環境を高度にカスタマイズできます。ジョブスケジューラは AWS リソースの管理に役立ちますが、スケジューラはデプロイを成功させるための 1 つの側面にすぎません。

ハイブリッドシナリオを正常に運用するための重要な要素は、データの局所性とデータ移動です。いくつかの HPC ワークロードでは、重要なデータセットは不要か、あるいは生成されません。したがって、データ管理はそれほど重要ではありません。しかしながら、大きなインプットデータを必要としたり、重要な出力データを生成するようなジョブはボトルネックとなりえます。データ管理のためのテクニックは、組織により異なります。例えば、ある組織ではエンドユーザーがジョブ送信スクリプトを使ってデータ転送を管理していますが、他の組織ではデータセットがある場所で単にいくつかのジョブを実行しているだけです。別の組織では両方の場所でデータを複製することを選択しているかもしれませんし、さらに他の組織では複数のオプションを組み合わせて使うことを選択しているかもしれません。

データ管理のアプローチに応じて、AWS はハイブリッドデプロイを支援するいくつかのサービスを提供します。例えば、AWS Direct Connect はオンプレミス環境と AWS との間に、専用のネットワーク接続を確立します。また、AWS DataSync はオンプレミスのストレージから、Amazon S3 や Amazon Elastic File System にデータを自動的に移動します。AWS Marketplace や AWS パートナーネットワーク (APN) より、追加のサードパーティー製ソフトウェアオプションも利用できます。

ハイブリッドデプロイのアーキテクチャは、疎結合と密結合ワークロードに使用できます。しかしながら、最適なパフォーマンスのためには、密結合ワークロードがオンプレミスまたは AWS のどちらかに 1 つは存在していなければなりません。

リファレンスアーキテクチャ

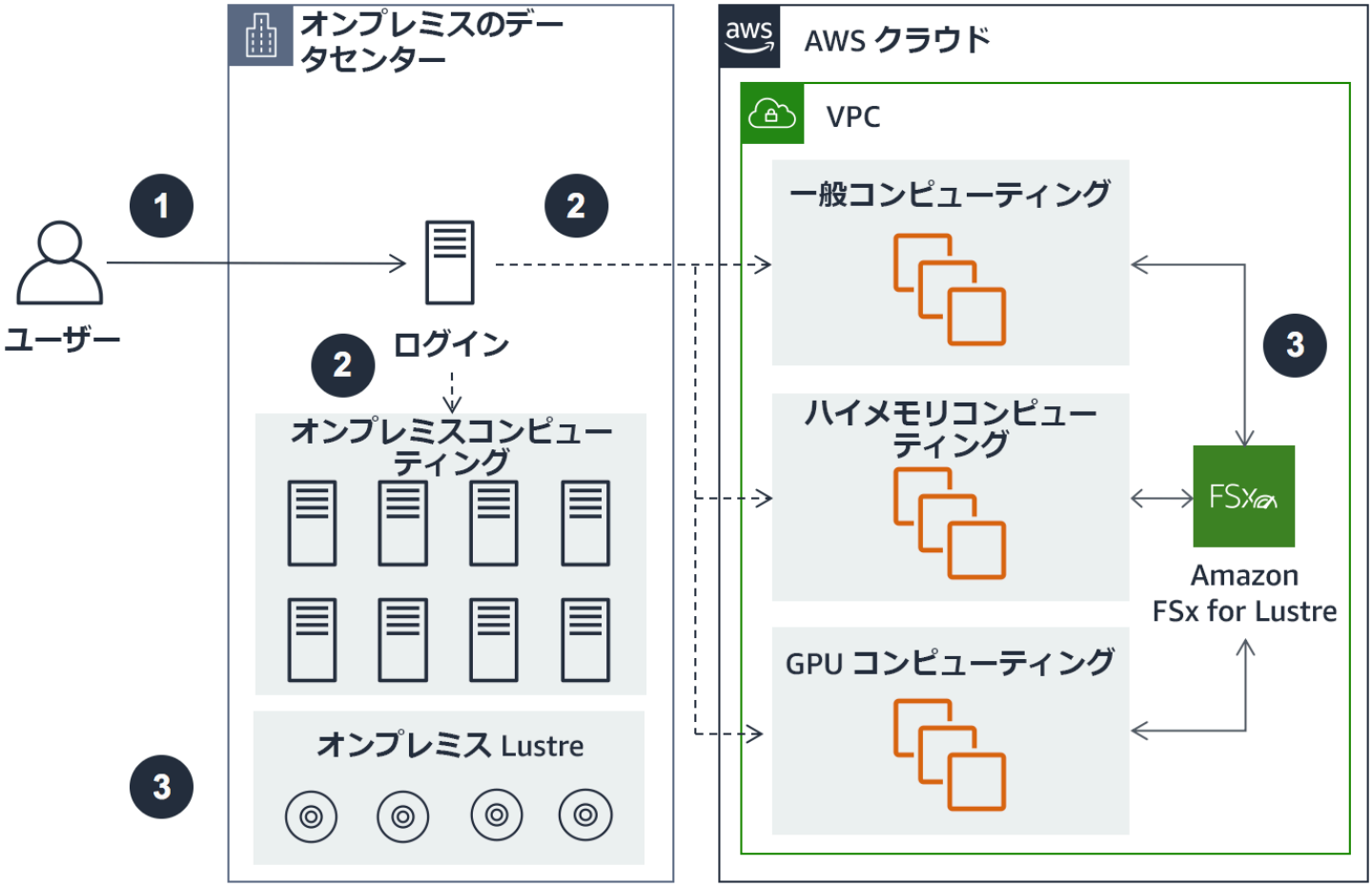

図 3: ハイブリッドでスケジュールベースのデプロイの例

ワークフローのステップ:

1. ユーザーが、オンプレミスのログインノードにあるスケジューラ (例えば Slurm) にジョブを送信します。
2. スケジューラは設定に従って、オンプレミスのコンピューティングか AWS インフラストラクチャにあるジョブを実行します。
3. ジョブは、実行される場所に応じた共有ストレージにアクセスします。

サーバーレス

疎結合クラウドジャーニーが完全にサーバーレスな環境になることがしばしばあります。これによりアプリケーションに集中でき、サーバーのプロビジョニングをマネージドサービスに任せることができます。サーバーをプロビジョニングや管理をすることなく、AWS Lambda がコードを実行できます。使用したコンピューティング時間に対してのみお支払いいただきます。コードが実行されていない時間には料金が発生しません。コードをアップロードすると、Lambda がコードの実行とスケールに必要なすべての処理を行います。また、Lambda は、他の AWS のサービスからのイベントを自動的にトリガーする機能も備えています。

スケーラビリティは、Lambda のサーバーレスアプローチによる 2 つ目の強みです。例えば、複数のメモリがあるコンピューティングのコアなど、それぞれのワーカーは最適なサイズであっても、アーキテクチャによって数千もの Lambda ワーカーが同時に発生し、膨大なコンピューティングのスループットキャパシティに達し、HPC ラベルをもたらすことがあります。例えば、同じアルゴリズムによって呼び出された膨大な数のファイルが分析される場合や、膨大な数のゲノムが並行して分析される場合、または膨大な数のゲノムの遺伝子的位置をモデリングしなければならない場合などです。スケールを行う最大値と、スケールする速度が問題になります。サーバーベースのアーキテクチャでは、リクエストに応じてキャパシティーを増やすために数分程度の時間が必要ですが (EC2 インスタンスなどの仮想マシンを使用している場合でも)、サーバーレス Lambda 関数は数秒でスケールします。AWS Lambda は、コンピューティング集約型の結果に対する予期しないリクエストに即座に応答する HPC インフラストラクチャを有効にし、事前のリソースの無駄なプロビジョニングを必要とすることなく、さまざまな数のリクエストに対応することができます。

コンピューティングに加えて、HPC ワークフローを支援する他のサーバーレスアーキテクチャがあります。AWS Step Functions は、さまざまな AWS のサービスをつなぎ合わせることで、パイプラインの複数のステップを調整できます。例えば、自動化されたゲノミクスパイプラインを、調整のために AWS Step Functions で、保存のために Amazon S3 で、小さなタスクのために AWS Lambda で、およびデータ処理のために AWS Batch で、それぞれ作成できます。

サーバーレスアーキテクチャは疎結合ワークロードに最適であり、他の HPC アーキテクチャと結合する場合にはワークフローのコーディネート役としても最適です。

リファレンスアーキテクチャ

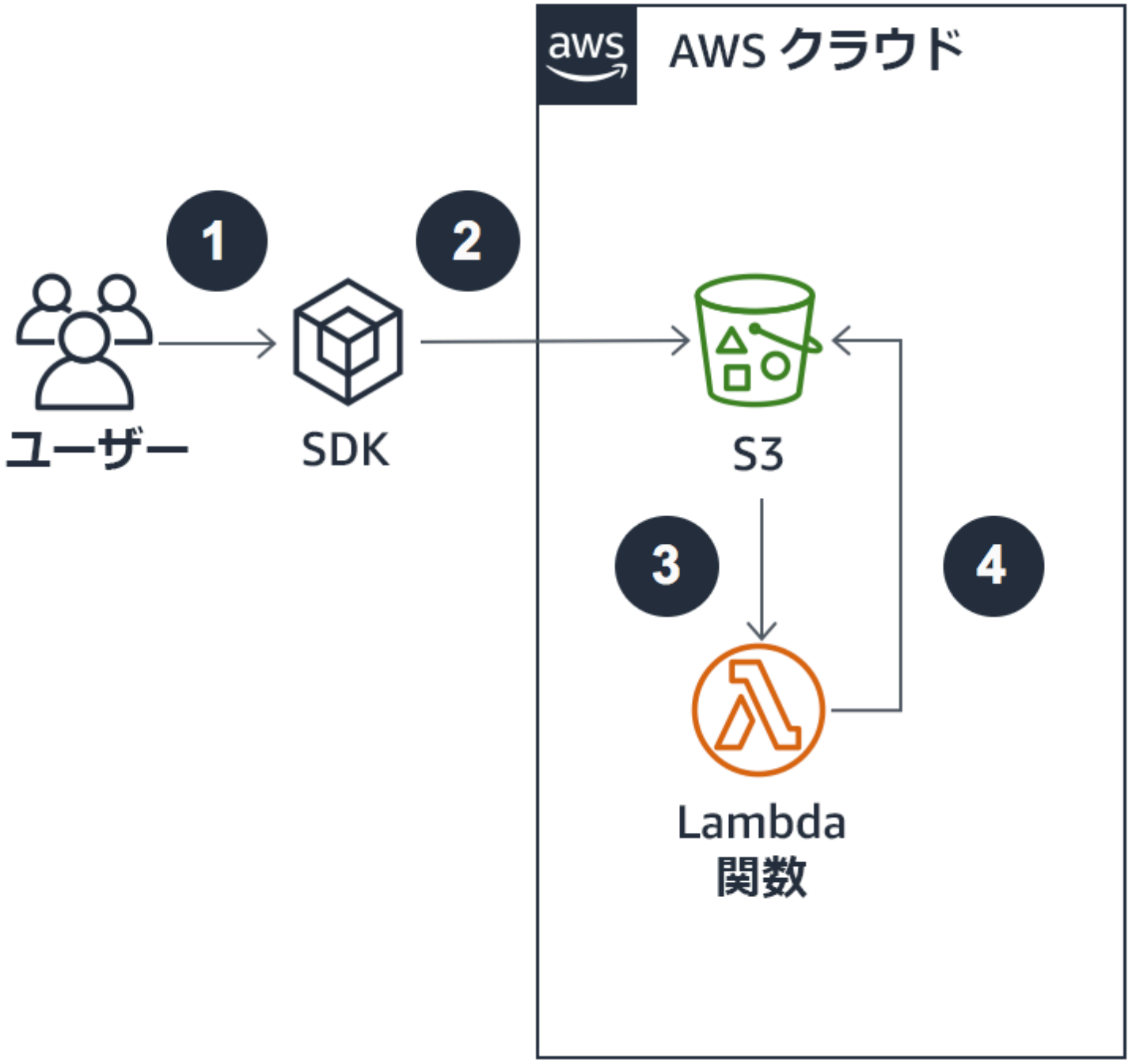

図 4: Lambda をデプロイした、疎結合ワークロードの例

ワークフローのステップ:

1. ユーザーが、AWS CLI または SDK 経由でファイルを S3 バケットにアップロードします。
2. 入力ファイルは、入力を表すプレフィックス (例えば、input/) をつけて保存されます。
3. S3 のイベントが自動的に Lambda の機能をトリガーして、入力データを処理します。

4. 出力ファイルは、出力を表すプレフィックス (例えば、output/) をつけて S3 バケットに戻され、保存されます。

Well-Architected フレームワークの 5 本の柱

このセクションでは、Well-Architected フレームワークの 5 本の柱という面から HPC を説明します。それぞれの柱では、設計の原則、定義、ベストプラクティス、評価質問、検討事項、キーとなる AWS のサービス、そして役に立つリンクについて説明します。

トピック

- 運用上の優秀性の柱
- セキュリティの柱
- 信頼性の柱
- パフォーマンス効率の柱
- コスト最適化の柱

運用上の優秀性の柱

運用の優秀性の柱には、ビジネス価値を提供し、継続的にサポートプロセスと手順を改善するために、システムを運用および監視する能力が含まれます。

トピック

- 設計の原則
- ベストプラクティス

設計の原則

クラウドでは、いくつかの原則が運用上の優秀性をもたらします。特に HPC ワークロードでは、次のものが強調されます。設計の原則については、AWS Well-Architected フレームワークホワイトペーパーもご覧ください。

- クラスターのオペレーションを自動化する: クラウドでは、ワークロード全体をコードとして定義し、コードを利用して更新できます。これにより、繰り返しのプロセスや手順を自動化できます。一貫したインフラストラクチャーを再作成でき、運用手順を実装できるというメリットがあります。これにはジョブ送信のプロセスと、ジョブの開始や完了、失敗などのイベントへの応答の自動化が含まれます。HPC では、ユーザーがジョブごとにいくつもの手順、たとえば 対象ファイルのアップロードやスケジューラーへのジョブ送信、結果ファイルの移動の繰り返しを当然のこととし

て期待することがよくあります。こうした繰り返し手順をスクリプトやイベント駆動のコードにより自動化して、ユーザビリティを最大化し、コストと失敗を最小化します。

- 適用可能な場合はラウドネイティブなアーキテクチャを使用する: HPC アーキテクチャは、一般的には 2 つのうち 1 つの形式をとります。1 つ目の形式は、ログインインスタンスとコンピューティングノード、そしてジョブスケジューラからなる従来のクラスター設定です。2 つめの形式は、自動化されたデプロイとマネージドサービスによるクラウドネイティブなアーキテクチャです。それぞれの (一時的な) クラスターのために単一のワークロードを実行するか、サーバーレスの能力を使用します。クラウドネイティブなアーキテクチャは、公開された先進技術により運用を最適化できます。しかしながら、HPC ユーザーが望む環境に合っているものが、最良の技術的アプローチです。

ベストプラクティス

クラウド内での運用の卓越性について 3 つの領域のベストプラクティスがあります。

トピック

- 準備
- 運用する
- 進歩する

準備、運用、進化の分野の詳細については、AWS Well-Architected フレームワークホワイトペーパーをご参照ください。このホワイトペーパーでは進化について説明していません。

準備

AWS Well-Architected フレームワークホワイトペーパーで、対応するセクションをご確認ください。

ワークロードのデプロイを準備する際には、専用のソフトウェアパッケージ (商用またはオープンソース) を使用してシステム情報を可視化し、こちらの情報を活用してお客様のワークロードのアーキテクチャパターンを定義することをご検討ください。AWS ParallelCluster または AWS CloudFormation などの自動化ツールを使用して、変数で設定可能な方法でこれらのアーキテクチャを定義します。

クラウドには複数のスケジューリングオプションがあります。1 つは AWS Batch を使用することです。これは、シングルノードタスクおよびマルチノードタスクの両方をサポートする完全マネージド

型バッチ処理サービスです。もう 1 つは、スケジューラを使用しないオプションです。たとえば、一時的なクラスターを作成して、単一のジョブを直接実行します。

HPCOPS 1: クラスター全体でアーキテクチャを標準化する方法

HPCOPS 2: 従来のスケジューラ、AWS Batch、または一時的なクラスターを使用したスケジューラなしでのジョブをスケジュールする方法

運用する

運用は、定期的な標準化と管理の必要があります。自動化、頻繁にある小さい変更、定期的な品質保証テスト、および変更を追跡、監査、ロールバック、およびレビューするための定義済みメカニズムを重視してください。変更は小さいもので頻繁にはなく、スケジュールされたダウンタイムを必要としませんし、手動で実行する必要もありません。ワークロードの主要な運用指標に基づいた幅広いログとメトリクスを収集して確認し、継続した運用を確保する必要があります。

AWS では、HPC オペレーションを処理するための追加ツールを使用できます。これらのツールは、モニタリング支援からデプロイの自動化まで、さまざまです。例えば、Auto Scaling で失敗したインスタンスを再起動したり、CloudWatch を使用してクラスターの負荷メトリクスをモニタリングしたり、ジョブ終了時の通知を設定したり、マネージドサービス (AWS Batch など) を使用して失敗したジョブの再試行ルールを実装したりできます。クラウドネイティブツールで、アプリケーションのデプロイと変更管理を大幅に改善できます。

手動または自動のリリース管理プロセスは、小さな増分変更と追跡されたバージョンに基づいている必要があります。操作に影響を与えることなく、問題を引き起こすリリースを元に戻すことができる必要があります。AWS CodePipeline や AWS CodeDeploy などの継続的統合および継続的デプロイツールを使用して、変更デプロイを自動化します。AWS CodeCommit などのバージョン管理ツールでソースコードの変更を追跡し、AWS CloudFormation テンプレートなどの自動化ツールでインフラストラクチャ設定を追跡します。

HPCOPS 3: 変更の影響を最小限に抑えながら、ワークロードを進化させる方法

HPCOPS 4: ワークロードをモニタリングして、期待どおりに動作していることを確認する方法

HPC にクラウドを使用する場合、新しい運用について考慮してください。オンプレミスクラスターのサイズは固定されていますが、クラウドクラスターは必要に合わせて拡張できます。HPC のクラウドネイティブアーキテクチャも、オンプレミスアーキテクチャとは異なる動作を行います。たとえば、ジョブの到着時、ジョブ送信とオンデマンドインスタンスリソースのプロビジョニングには、異なるメカニズムを使用します。クラウドの弾力性とクラウドネイティブアーキテクチャの動的な性質に対応できる運用手順を採用する必要があります。

進歩する

進化するプラクティス領域に関して、HPC に固有のベストプラクティスはありません。詳細については、AWS Well-Architected フレームワークホワイトペーパーの対応するセクションをご確認ください。

セキュリティの柱

セキュリティの柱とは、リスクの評価とその軽減戦略を通してビジネス価値を提供しながら、情報、システム、資産を保護する能力のことを指します。

トピック

- 設計の原則
- 定義
- ベストプラクティス

設計の原則

クラウドには、システムセキュリティの強化に役立つ原則が数多くあります。AWS Well-Architected フレームワークホワイトペーパーにある設計の原則が推奨されており、HPC ワークロードによって異なることはありません。

定義

クラウド内でのセキュリティには、5 つのベストプラクティス領域があります。

- Identity and Access Management (IAM)

- 発見的統制
- インフラストラクチャ保護
- データ保護
- インシデント対応

システムを設計する前に、セキュリティ対策を確立する必要があります。アクセス許可の制御、セキュリティインシデントの特定、システムやサービスの保護、さらにデータ保護を介してデータの機密性と完全性の維持ができる必要があります。明確に定義され、実践に裏打ちされたプロセスを利用して、セキュリティインシデントに対応してください。これらのツールやテクニックは、データ損失の予防や規制遵守という目的を達成するためにも重要です。

AWS の責任共有モデルにより、このクラウドを導入した組織はセキュリティとコンプライアンスの目標を達成することができます。AWS がこのクラウドサービスの基盤となるインフラストラクチャを物理的に保護しているため、AWS のお客様はサービスを使用して目標を達成することだけに集中できます。AWS クラウドは、セキュリティデータへのアクセスや、セキュリティイベントに対応する自動化した手段を提供します。

セキュリティに関するベストプラクティス領域はどれも重要で、AWS Well-Architected フレームワークホワイトペーパーで十分に説明されています。発見的統制、インフラストラクチャ保護、インシデント対応の領域について、AWS Well-Architected フレームワークホワイトペーパーで説明されています。このホワイトペーパーには記載がないため、HPC ワークロードの変更は必要ありません。

ベストプラクティス

トピック

- Identity and Access Management (IAM)
- 発見的統制
- インフラストラクチャ保護
- データ保護
- インシデント対応

Identity and Access Management (IAM)

Identity and Access Management は、情報セキュリティプログラムの重要な一部分です。これで、許可および認証されたユーザーのみがリソースにアクセスできるようになります。たとえば、プリン

シパル (アカウント内のアクションを実行するユーザー、グループ、サービス、およびロール) の定義、これらのプリンシパルを参照するポリシーの構築、強力な認証情報の実装が可能となります。これらの権限管理機能は、認証と承認の中枢となる概念です。

HPC ワークロードを自律的かつ一時的に実行し、機密データの露出を制限します。自律型デプロイは、インスタンスへの人によるアクセスが最小限である必要があります。そうすることで、リソースの公開が最小限に抑えられます。HPC データを限られた時間内で生成することで、潜在的に不正なデータアクセスの可能性を最小限に抑えます。

HPCSEC 1: マネージドサービス、自律型方式、および一時的なクラスターを使用して、ワークロードインフラストラクチャへの人によるアクセスを最小限に抑える方法

HPC アーキテクチャでは、さまざまなマネージドコンピューティングサービス (AWS Batch、AWS Lambda など) および非マネージドコンピューティングサービス (Amazon EC2 など) を使用できます。アーキテクチャが EC2 インスタンスへの接続などのコンピューティング環境への直接アクセスを必要とする場合、普通、ユーザーは Secure Shell (SSH) を介して接続し、SSH キーで認証します。これは、従来のクラスターシナリオでは一般的なアクセスモデルです。この場合、SSH キーを含むすべての認証情報は適切に保護され、定期的にローテーションされる必要があります。

一方、AWS Systems Manager には、ブラウザベースのインタラクティブなシェルや CLI 機能を提供するフルマネージドサービス (Session Manager) があります。これらで、インバウンドポートを開いたり、要塞ホストを維持したり、SSH キーの管理なしに、安全で監査可能なインスタンス管理が可能です。Session Manager は、ProxyCommand をサポートする SSH クライアントを介してアクセスできます。

HPCSEC 2: 認証情報を保護および管理する方法

発見的統制

発見的統制のベストプラクティス領域に関して、HPC に固有のベストプラクティスはありません。AWS Well-Architected フレームワークホワイトペーパーで、対応するセクションをご確認ください。

インフラストラクチャ保護

インフラストラクチャのベストプラクティス領域に関して、HPC に固有のベストプラクティスはありません。AWS Well-Architected フレームワークホワイトペーパーで、対応するセクションをご確認ください。

データ保護

システムを設計する前に、基本的なセキュリティ対策を確立する必要があります。例えば、データ分類によって、機密レベルに基づいた組織データの分類を行います。さらに暗号化することで、不正なアクセスを判読できないようにしデータを保護します。これらのツールやテクニックは、データ損失の予防あるいは規制遵守という目的を達成するためにも重要です。

HPCSEC 3: アーキテクチャが結果のライフサイクルを通じて、ストレージの可用性と耐久性のデータ要件に対処する方法

機密性や規制上の義務のレベルに対応した分類だけでなく、HPC データは、データが次にいつどのように使用されるかに応じた分類も可能です。最終結果が保持されることがほとんどで、中間結果は必要に応じ再作成できるため保持する必要はありません。注意深くデータ評価と分類を行うことで、重要なデータを Amazon S3 や Amazon EFS などのより回復力のあるストレージソリューションにプログラムで移行することが可能となります。

データの有効期限とプログラムによるデータ処理を理解することで、Well-Architected インフラストラクチャの露出を最小限に抑え、最大限の保護を実現できます。

インシデント対応

インシデント対応のベストプラクティス領域に関して、HPC に固有のベストプラクティスはありません。AWS Well-Architected フレームワークホワイトペーパーで、対応するセクションをご確認ください。

信頼性の柱

信頼性の柱には、インフラストラクチャまたはサービスの障害からの復旧、必要に応じた動的なコンピューティングリソースの獲得、設定ミスや一時的なネットワークの問題などによる障害の軽減などのシステムの能力が含まれます。

トピック

- 設計の原則
- 定義
- ベストプラクティス

設計の原則

クラウドでは、信頼性の向上に役立つ原則が数多くあります。特に HPC ワークロードでは、次のものが強調されます。詳細については、AWS Well-Architected フレームワークホワイトペーパーの設計の原則をご参照ください。

- 水平方向にスケールして、総合的なシステムの可用性を高める: システム全体に対する単一の障害の影響を軽減する可能性のある水平スケーリングをオプションとして検討することは重要です。例えば、1 つの大きな共有 HPC クラスターが複数のジョブを実行するのではなく、Amazon インフラストラクチャ全体に複数のクラスターを作成して、潜在的な障害のリスクをさらに分離することを検討してください。インフラストラクチャはコードとして扱うことができるため、単一のクラスター内でリソースを水平方向にスケーリングできるだけでなく、個々のケースを実行するクラスターの数を水平方向にスケーリングすることも可能です。
- キャパシティーの推測を止める: HPC クラスターセットをプロビジョニングして、現在のニーズを満たし、需要の増減に応じて手動または自動でスケーリングできます。例えば、使用していないときにアイドル状態のコンピューティングノードを終了し、実行します。あるいは、使用していないときにはアイドル状態のコンピューティングノードを終了し、キューで待機するのではなくクラスターを同時に実行することで、複数のコンピューティングを処理します。
- 自動化の変更を管理する: インフラストラクチャの変更を自動化すると、クラスターインフラストラクチャをバージョン管理下に置き、以前に作成したクラスターの正確な複製を作成できます。自動化の変更を管理する必要があります。

定義

クラウド内で信頼性のあるベストプラクティスには 3 つの領域があります :

- 基盤
- 変更管理
- 障害の管理

変更管理の領域については、AWS Well-Architected フレームワークホワイトペーパーに説明があります。

ベストプラクティス

トピック

- 基盤
- 変更管理
- 障害の管理

基盤

HPCREL 1: アカウントに対する AWS のサービスの制限を管理する方法

AWS では、お客様が意図せずにリソースを過剰にプロビジョニングすることのないよう、サービスの制限を設定しています (チームが要求できる各リソースの数の上限)。

HPC アプリケーションでは、多数のコンピューティングインスタンスを同時に必要とすることがよくあります。そのため、HPC ワークロードにとって、水平スケーリングの機能と利点はとても重要です。ただし、水平方向にスケーリングするには、AWS のサービス制限を拡大してから、大きなクラスター 1 つまたは多数の小さなクラスターに一度に大きなワークロードをデプロイする必要があります。

多くの場合で、大規模なデプロイ要件を処理するには、サービスの制限をデフォルト値から増やす必要があります。引き上げをリクエストするには、AWS サポート までお問い合わせください。

変更管理

変化管理のベストプラクティス領域に関して、HPC に固有のベストプラクティスはありません。AWS Well-Architected フレームワークホワイトペーパーで、対応するセクションをご確認ください。

障害の管理

複雑なシステムでは障害が時折発生することが予想されるため、これらの障害の認識、対応、再発生の防御が重要です。障害のシナリオには、クラスターの起動障害、または特定のワークロードでの障害などがあります。

HPCREL 2: チェックポイントを使用して、アプリケーションを障害から回復させる方法

耐障害性の改善にはいくつかの方法があります。長時間実行する場合、コードに定期的なチェックポイントを組み込むと、障害が発生してもパーシャル状態から続行できます。チェックポイント設定はアプリケーションレベルでの障害管理の一般的な機能で、多くの HPC アプリケーションにすでに組み込まれています。アプリケーションが定期的に中間結果を書き出すのが、最も一般的です。中間結果は潜在的なアプリケーションエラーの分析情報を提供し、必要に応じてケースを再起動することで、作業が部分的に失われるようにします。

極めて費用対効果が高いが潜在的に割り込み可能なインスタンスを使用している場合、チェックポイント機能はスポットインスタンスで役立ちます。さらに、一部のアプリケーションでは、デフォルトのスポット割り込み動作を変更する (例えば、インスタンスを終了するのではなく、停止または休止状態にする) ことでメリットを享受できる場合があります。障害管理のためにチェックポイントを使用する場合、ストレージオプションの耐久性を考慮することが重要です。

HPCREL 3: アーキテクチャの耐障害性の計画方法

複数のアベイラビリティーゾーンにデプロイする場合、耐障害性を改善できます。密結合 HPC アプリケーションの低レイテンシー要件では、個々のケースが単一のクラスターのプレースメントグループとアベイラビリティーゾーン内に存在している必要があります。一方、疎結合アプリケーションにはこのような低レイテンシー要件はないため、複数のアベイラビリティーゾーンにデプロイする機能により障害管理を改善できます。

この設計に決定する際には、信頼性とコストの柱間のトレードオフを考慮してください。コンピューティングおよびストレージインフラストラクチャ (ヘッドノードや接続済みストレージなど) の複製には追加コストがかかります。データをアベイラビリティーゾーンまたは別の AWS リージョンに移動する際には、データ転送料金が発生する場合があります。緊急ではないユースケースの場合、災害対策 (DR) イベントの一部として別のアベイラビリティーゾーンに移動すること以外、望ましい方法がない場合があります。

パフォーマンス効率の柱

パフォーマンス効率の柱では、コンピューティングリソースを効率的に使ってシステム要件を満たし、需要が変化し技術が進化するのに合わせて効率性を維持することに重点を置きます。

トピック

- 設計の原則
- 定義
- ベストプラクティス

設計の原則

クラウドで HPC を設計する際、パフォーマンス効率の達成に役立つ原則が数多くあります。

- アプリケーション用クラスターの設計: 従来のクラスターは静的であるため、クラスター用にアプリケーションを設計する必要があります。AWS には、アプリケーションのクラスターを設計する機能があります。アプリケーションごとに個別のクラスターを使用する万能モデルは不要になりました。AWS でいろいろなアプリケーションを実行する場合、さまざまなアーキテクチャを使って各アプリケーションの要求を満たすことができます。このため、コストを最小限に抑えながら最高のパフォーマンスを実現できます。
- 分かりやすいユースケースでパフォーマンスをテストする: 特定のアーキテクチャで HPC アプリケーションのパフォーマンスを評価するには、アプリケーション自体の分かりやすいデモを実行することが最良の方法です。予想外の小規模または大規模なデモンストレーションのケース (予想されるコンピューティング、メモリ、データ転送、またはネットワークトラフィックがないケース) では、AWS でアプリケーションパフォーマンスの分かりやすいテストを実施できません。システム固有のベンチマークは、基盤となるコンピューティングインフラストラクチャパフォーマンスを理解しますが、アプリケーションが全体としてどのように実行されるかは反映されていません。AWS の従量課金制モデルでは、迅速かつ費用対効果の高い概念実証を実現できます。
- クラウドネイティブアーキテクチャの使用 (該当する場合): クラウドでは、マネージド、サーバーレス、クラウドネイティブのアーキテクチャのおかげでサーバーを実行および保守する必要なく、従来のコンピューティングアクティビティを実行できます。HPC のクラウドネイティブコンポーネントは、コンピューティング、ストレージ、ジョブオーケストレーション、およびデータとメタデータのオーケストレーションを対象としています。さまざまな AWS のサービスを使って、ワークロードプロセスの各ステップを分離し、より高性能な機能に最適化することができます。
- 実験頻度の増加: 仮想的で自動化できるリソースを使うことで、異なるタイプのインスタンス、ストレージ、設定を使用した比較テストを簡単に実施できるようになります。

定義

クラウドのパフォーマンス効率を向上させるには、次の 4 つのベストプラクティス領域があります。

- 選択
- レビュー
- モニタリング
- トレードオフ

レビュー、モニタリング、トレードオフの領域は、AWS Well-Architected フレームワークホワイトペーパーに説明があります。

ベストプラクティス

トピック

- 選択
- コンピューティング
- ストレージ
- ネットワーク
- レビュー
- モニタリング
- トレードオフ

選択

特定のシステムに最適なソリューションは、作業負荷の種類によって異なります。Well-architected システムには複数のソリューションがあり、さまざまな機能を使ってパフォーマンスを改善します。HPC アーキテクチャでは、キュー、バッチ、クラスターコンピューティング、コンテナ、サーバーレス、イベント駆動など、複数の異なるアーキテクチャ要素に依存できます。

コンピューティング

HPCPERF 1: コンピューティングソリューションの選択方法

特定の HPC アーキテクチャに最適なコンピューティングソリューションは、ワークロードのデプロイ方法、自動化の程度、使用パターン、設定によって異なります。プロセスの各ステップで、異なるコンピューティングソリューションを選択できます。アーキテクチャで使用するコンピューティングソリューションを正しく選択しないと、パフォーマンス効率が低下するおそれがあります。

インスタンスは仮想化サーバーで、さまざまなファミリーとサイズで幅広い機能を提供します。一部のインスタンスファミリーは、コンピューティング、メモリ、または GPU を集中的に使用するワークロードなどの特定のワークロードを対象としています。その他のインスタンスは汎用です。

ターゲットワークロードと汎用インスタンスファミリーはどちらも、HPC アプリケーションに役立ちます。特に HPC を対象とするインスタンスには、GPU や FPGA などのコンピューティングに最適化されたファミリーや高速インスタンスタイプが含まれます。

一部のインスタンスファミリーには、ファミリー内でバリアントを提供する追加機能があります。例えば、あるインスタンスファミリーには、ローカルストレージ、より優れたネットワーク機能、または異なるプロセッサを備えたバリアントがあります。これらのバリアントはインスタンスタイプマトリクスで表示でき、一部の HPC ワークロードではパフォーマンスが向上することがあります。

各インスタンスファミリー内において、複数のインスタンスサイズでリソースの垂直スケーリングが可能です。一部のアプリケーションではより大きなインスタンスタイプ (24xlarge など) を必要としますが、他のアプリケーションではアプリケーションがサポートする数またはプロセスに応じて、より小さなタイプ (large など) で実行できます。密結合ワークロードを使用する場合、最大のインスタンスタイプで最適なパフォーマンスが得られます。

T シリーズのインスタンスファミリーは CPU 使用率が中程度のアプリケーション向けに設計されているため、CPU パフォーマンスのベースラインレベルを超えたバーストからメリットが得られます。ほとんどの HPC アプリケーションはコンピューティング集約型であるため、T シリーズインスタンスファミリーではパフォーマンスが低下します。

アプリケーションは要件 (必要なコア、プロセッサ速度、メモリ要件、ストレージのニーズ、ネットワーク仕様など) によって異なります。インスタンスファミリーとインスタンスタイプを選択する際は、まずアプリケーションの特定のニーズを考慮しましょう。インスタンスタイプは、特定のアプリケーションコンポーネントのターゲットインスタンスを必要とするアプリケーションに合わせて組み合わせることができます。

コンテナは、特にアプリケーションが既にコンテナ化されている場合、多くの HPC ワークロードにとって魅力的なオペレーティングシステム仮想化の方法です。AWS Batch、Amazon Elastic Container Service (ECS)、Amazon Elastic Container Service for Kubernetes (EKS) などの AWS のサービスは、コンテナ化されたアプリケーションのデプロイに役立ちます。

関数は実行環境を抽象化します。AWS Lambda を使用することで、インスタンスをデプロイ、実行、または保守せずにコードを実行できます。多くの AWS のサービスは、サービス内のアクティビティに基づいてイベントを発行します。大抵は、Lambda 関数はサービスイベントからトリガーできます。例えば、Lambda 関数はオブジェクトが Amazon S3 にアップロードされた後に実行できます。HPC ユーザーの多くが Lambda を使用して、ワークフローの一部としてコードを自動的に実行しています。

選択したコンピューティングインスタンスを起動するとき、いくつか選択肢があります。

- オペレーティングシステム: 現在のオペレーティングシステムは、最高のパフォーマンスを実現し、最新のライブラリへのアクセスを確保するために非常に重要です。
- 仮想化タイプ: AWS Nitro System で実行される新世代 EC2 インスタンス。Nitro System は、ホストハードウェアのすべてのコンピューティングリソースとメモリリソースをインスタンスに提供し、全体的なパフォーマンスを向上させます。専用の Nitro Card によって、高速ネットワーク、高速 EBS、および I/O アクセラレーションが可能になります。インスタンスは、管理ソフトウェアのリソースを抑制しません。

Nitro Hypervisor は、軽量のハイパーバイザーで、メモリと CPU の割り当てを管理し、ベアメタルとほぼ同等のパフォーマンスを提供します。また、Nitro System は、Nitro Hypervisor なしでベアメタルインスタンスを実行できるようにします。ベアメタルインスタンスを起動すると、基盤となるサーバーを立ち上げします。これには、すべてのハードウェアおよびファームウェアコンポーネントの確認が含まれます。これは、仮想化インスタンスと比較した場合、ベアメタルインスタンスがワークロードを開始できるようになるまでに時間がかかるからです。リソースが要求に基づいて起動および終了する動的 HPC 環境で動作する場合、追加の初期化時間を考慮する必要があります。

HPCPERF 2: アプリケーションのコンピューティング環境を最適化する方法

基盤となるハードウェア機能: AMI の選択に加えて、基盤となる Intel プロセッサのハードウェア機能を活用することで、環境をさらに最適化できます。基盤となるハードウェアを最適化する際に考慮すべき 4 つの主要な方法があります。

1. 高度なプロセッサ機能
2. Intel ハイパースレッディングテクノロジー
3. プロセッサアフィニティ
4. プロセッサステートコントロール

HPC アプリケーションは、これらの高度なプロセッサ機能 (例えば、Advanced Vector Extensions) のメリットを得ることができ、Intel アーキテクチャ用にソフトウェアをコンパイルすることで、計算速度を向上させることができます。アーキテクチャ固有の命令のコンパイラオプションは、コンパイラによって異なります (コンパイラの使用ガイドをご確認ください)。

AWS はデフォルトで、一般的に「ハイパースレッディング」と呼ばれる、 Intel ハイパースレッディングテクノロジーを有効にします。ハイパースレッディングは、ハイパースレッドごとに 1 つのプロセス (コアごとに 2 つのプロセス) を許可することにより、一部のアプリケーションのパフォーマンスを向上させます。ほとんどの HPC アプリケーションは、ハイパースレッディングを無効にすることでメリットを得るため、HPC アプリケーションにとって好ましい環境になる傾向があります。ハイパースレッディングは、Amazon EC2 で簡単に無効にできます。ハイパースレッディングを有効にしてアプリケーションをテストする場合を除き、HPC アプリケーションの実行時にはハイパースレッディングを無効にし、プロセスを起動して個別にコアに固定することをお勧めします。CPU またはプロセッサアフィニティにより、プロセスの固定が容易になります。

プロセッサアフィニティはさまざまな方法でコントロールできます。例えば、オペレーティングシステムレベルで設定したり (Windows と Linux の両方で可能) 、スレッディングライブラリ内のコンパイラフラグとして設定したり、実行中に MPI フラグとして指定したりできます。プロセッサアフィニティをコントロールするために選択するべき方法は、ワークロードとアプリケーションによって異なります。

AWS を使用すると、特定のインスタンスタイプで、プロセッサの状態コントロールを調整できます。パフォーマンスを最適化するために、C ステート (アイドル状態) および P ステート (動作状態) の設定を変更することを検討できます。デフォルトの C ステートと P ステートの設定は、ほとんどのワークロードに最適な、最高のパフォーマンスを提供します。ただし、シングルコアまたはデュアルコアの高い周波数を犠牲にしてレイテンシーを短縮したり、スパイク状のターボブースト周波数とは対照的に低い周波数で一貫したパフォーマンスを実現したりする方がアプリケーションにメリットがある場合は、選択したインスタンスで使用可能な C ステートまたはP ステートの設定をお試しください。

コンピューティング環境を最適化するために利用可能な、多くのコンピューティングオプションがあります。クラウドデプロイでは、オペレーティングシステムからインスタンスタイプ、ベアメタルデプロイまで、あらゆるレベルでの実験が可能です。静的クラスターはデプロイ前に調整されるため、クラウドベースのクラスターの実験に費やす時間は、目的のパフォーマンスを達成するために極めて重要です。

ストレージ

HPCPERF 3: ストレージソリューションの選択方法

特定の HPC アーキテクチャに最適なストレージソリューションは、そのアーキテクチャを対象とする個々のアプリケーションに大いに依存します。ワークロードのデプロイ方法、自動化の程度、および目的とするデータライフサイクルパターンも要因になります。AWS は幅広いストレージオプションを提供しています。コンピューティングと同じで、アプリケーションの特定のストレージニーズを対象とした場合に、最高のパフォーマンスが得られます。AWS では、「万能」なアプローチのためにストレージをオーバープロビジョニングすることは必要ありません。また、大規模で高速な共有ファイルシステムは必ずしも必要ではありません。コンピューティングで最適な選択をすることは、HPC パフォーマンスを最適化するために重要です。そして、多くの HPC アプリケーションは、最高速のストレージソリューションのメリットを享受することがありません。

HPC デプロイには、たびたびクラスターコンピューティングノードからアクセスされる共有または高性能のファイルシステムが必要になります。AWS Managed Services、AWS Marketplace 出品、APN パートナーソリューション、および EC2 インスタンスにデプロイされたオープンソース設定からこれらのストレージソリューションを実装するために使用できるアーキテクチャパターンがいくつかあります。特に、Amazon FSx for Lustre は、高性能の並列ファイルシステムを必要とする HPC アーキテクチャに対して、高い費用対効果や効率的なソリューションを提供するマネージドサービスです。また、共有ファイルシステムは、Amazon EBS ボリュームまたはインスタンスストアボリュームを持つ Amazon Elastic File System (EFS) または EC2 インスタンスから作成することもできます。多くの場合、シンプルな NFS マウントが共有ディレクトリの作成に使用されます。

ストレージソリューションを選択する際、ローカルストレージと共有ストレージのいずれか、またはその両方に EBS-backed インスタンスを選択できます。大抵の場合、EBS ボリュームは HPC ストレージソリューションの基盤です。磁気ハードディスクドライブ (HDD)、汎用ソリッドステートドライブ(SSD)、および高 IOPS ソリューション用のプロビジョンド IOPS SSD など、さまざまなタイプの EBS ボリュームが利用可能です。スループット、IOPS パフォーマンス、コストの面で異なります。

Amazon EBS に最適化されたインスタンスを選択することにより、パフォーマンスをさらに向上させることができます。EBS 最適化インスタンスは、最適化された設定スタックを使用し、Amazon EBS I/O 専用の追加キャパシティーを提供します。この最適化により、Amazon EBS I/O とインスタンスとのその他のネットワークトラフィック間の競合を最小限に抑えることで、EBS ボリュームで最高のパフォーマンスを発揮します。EBS に最適化されたインスタンスを選択して、パフォーマン

スの一貫性を高め、低遅延ネットワークに依存していたり、EBS ボリュームへの I/O データの集中的なニーズがあったりする HPC アプリケーションに使用します。

EBS 最適化インスタンスを起動するには、デフォルトで EBS 最適化が有効になるインスタンスタイプを選択するか、起動時に EBS 最適化を有効にするインスタンスタイプを選択します。

Nonvolatile Memory Express (NVMe) SSD ボリューム (特定のインスタンスファミリーでのみ使用可能) を含むインスタンスストアボリュームは、一時的なブロックレベルストレージに使用できます。EBS 最適化とインスタンスストアボリュームのサポートについては、インスタンスタイプマトリックスをご参照ください。

ストレージソリューションを選択するときは、ユーザーのアクセスパターンに合うようにして、目的のパフォーマンスを実現してください。さまざまなストレージタイプと設定を簡単に試すことができます。HPC ワークロードでは、最も高価なオプションが最高のパフォーマンスのソリューションであるとは限りません。

ネットワーク

HPCPERF 4: ネットワークソリューションの選択方法

HPC ワークロードに最適なネットワークソリューションは、レイテンシー、帯域幅、およびスループットの要件によって異なります。密結合 HPC アプリケーションでは、大抵の場合、コンピューティングノード間のネットワーク接続に可能な限り低いレイテンシーが必要です。中規模の密結合ワークロードの場合、ネットワークを横断しないでアプリケーションが完全にインスタンスに収まるよう、多数のコアを持つ大きなインスタンスタイプを選択することができます。

または、一部のアプリケーションはネットワークに結びついており、高いネットワークパフォーマンスが必要です。これらのアプリケーションの場合は、より高いネットワークパフォーマンスのインスタンスを選択できます。ファミリー内で最大のインスタンスタイプを使用すると、最高のネットワークパフォーマンスが得られます。詳細については、インスタンスタイプマトリックスをご参照ください。[7]

大規模な密結合アプリケーションには、インスタンス間のレイテンシーが低い複数のインスタンスが必要です。AWS では、アベイラビリティーゾーン内のインスタンスの論理グループであるクラスタープレイスメントグループにコンピューティングノードを起動することで、これを実現しています。クラスタープレイスメントグループは、インスタンス間の完全な 2 分割帯域幅を含む、ノンブロッキングおよびノンオーバーサブスクライブ接続を提供します。複数のインスタンスにまたがる

レイテンシーに敏感な密結合アプリケーションには、クラスタープレイスメントグループを使用します。

クラスタープレイスメントグループに加えて、密結合アプリケーションは、Amazon EC2 インスタンスに接続できるネットワークデバイスである Elastic Fabric Adapter (EFA) のメリットを得ます。EFA は、クラウドベースの HPC システムで従来使用されていた TCP トランスポートよりも、一貫したより低いレイテンシーと高いスループットを提供します。HPC アプリケーションがネットワークインターフェイスハードウェアと直接通信できるようにする Libfabric API を介して、OS バイパスアクセスモデルを有効にします。EFA は、インスタンス間通信のパフォーマンスを向上させ、既存の AWS ネットワークインフラストラクチャで動作するように最適化されており、密結合アプリケーションのスケーリングに非常に重要です。[13]

アプリケーションが EFA の OS バイパス機能を利用できない場合や、インスタンスタイプが EFA をサポートしていない場合は、拡張ネットワーキングをサポートするインスタンスタイプを選択することで、最適なネットワークパフォーマンスを取得できます。拡張ネットワーキングは、ハードウェアでエミュレートされたデバイスではなくパススルーを使用することで、EC2 インスタンスのネットワークパフォーマンスを上げ、CPU 使用率を抑えます。この方法により、EC2 インスタンスは、従来のデバイス仮想化と比較して、より高い帯域幅、より高いパケット/秒処理、およびインスタンス間でより低いレイテンシーを実現できます。

拡張ネットワーキングは、現世代のすべてのインスタンスタイプで利用できます。サポートされているドライバーを備えた AMI が必要です。現在のほとんどの AMI は、サポートされているドライバーを備えていますが、カスタム AMI には更新されたドライバーが必要な場合があります。拡張ネットワーキングやインスタンスサポートの有効化の詳細については、拡張ネットワーキングのドキュメントをご参照ください。

疎結合ワークロードは通常、非常に低レイテンシーのネットワークの影響を受けず、クラスタープレイスメントグループを使用する必要や、同じアベイラビリティーゾーンまたはリージョンにインスタンスを保持する必要はありません。

レビュー

確認するベストプラクティス領域に関して、HPC に固有のベストプラクティスはありません。AWS Well-Architected フレームワークホワイトペーパーで、対応するセクションをご確認ください。

モニタリング

モニタリングのベストプラクティス領域に関して、HPC に固有のベストプラクティスはありません。AWS Well-Architected フレームワークホワイトペーパーで、対応するセクションをご確認ください。

トレードオフ

モニタリングのベストプラクティス領域に関して、HPC に固有のベストプラクティスはありません。AWS Well-Architected フレームワークホワイトペーパーで、対応するセクションをご確認ください。

コスト最適化の柱

コスト最適化の柱には、HPC システムを全ライフサイクルにわたり向上および改善する継続的プロセスが含まれます。最初の PoC (実証支援) の初期設計から、製品ワークロードの継続運用まで、本ドキュメントのプラクティスを採用すれば、コスト意識の高いシステムを構築・運用して、ビジネス上の成果とコストの最小化の両方を達成できるようになります。これにより、最大のビジネス投資収益率を得ることができます。

トピック

- 設計の原則
- 定義
- ベストプラクティス

設計の原則

クラウド内の HPC では、多数の原則に従い、最適化を実現することができます。

- 消費モデルの採用: 消費するコンピューティングリソースに対してのみ費用をかけます。HPC ワークロードは増減するため、状況に応じてリソースキャパシティーを調整することでコスト削減が可能です。例えば、低レベルのランレート HPC キャパシティーを事前にプロビジョニングおよびリザーブして、より高い割引を受けることができます。その一方、バースト要件はスポットまたはオンデマンドの料金設定でプロビジョニングされ、必要に応じてオンラインになります。
- 特定のジョブのインフラストラクチャコストを最適化する: 多くの HPC ワークロードは、データ転送、前処理、計算、後処理、データ転送、ストレージといった各ステップを含むデータ処理パイプラインの一部です。クラウドでは、大規模で高価なサーバーではなく、コンピューティングプラットフォームが、各ステップで最適化されます。たとえば、パイプラインの 1 つのステップで大量のメモリが必要な場合は、メモリを大量に消費するアプリケーション用に、より高価な大容量メモリサーバーを購入するだけで済みます。他のステップでは、小規模で安価なコンピューティングプラットフォームで適切に実行します。ワークロードの各ステップでインフラストラクチャを最適化することで、コストは削減されます。

- 最も効率的な方法でのワークロードのバースト: クラウド内の水平スケーリングにより、HPC ワークロードの節約が実現できます。水平方向のスケーリングを行う場合、多くのジョブまたはワークロード全体の反復処理が同時に実行され、合計経過時間が短縮されます。アプリケーションによっては、間接的なコスト削減しながら水平スケーリングにより実質ベースのコストにすることができます。
- スポット料金を利用する: Amazon EC2 スポットインスタンスは、オンデマンドインスタンスと比較して大幅な割引で AWS の予備のコンピューティングキャパシティーを提供します。ただし、EC2 がキャパシティーを回収する必要がある場合は、スポットインスタンスを中断できます。 多くの場合、スポットインスタンスは、柔軟性や耐障害性のあるワークロードにとって、最もコスト効率が良いリソースです。HPCワークロードは断続的に発生するため、スポットインスタンスに適しています。スポットインスタンスが中断するリスクは、スポットアドバイザーを使用することで最小限に抑えることができます。また、デフォルトの中断動作を変更したり、スポットフリートを使用してスポットインスタンスを管理したりすることにより、中断の影響を軽減できます。時折ワークロードを再起動する必要性がありますが、スポットインスタンスのコスト削減によって簡単に相殺できるでしょう。
- コストと時間のトレードオフを評価する: 密結合された大規模な並列ワークロードは、広範囲に設定可能なコア数で実行できます。これらのアプリケーションの場合、ケースの実行効率は通常、コア数が増えると低下します。類似する種類やサイズのケースが多数実行して、コスト対所要時間の曲線を作成できます。スケーリングは計算要件とネットワーク要件の比率に大きく依存するため、曲線はケースタイプとアプリケーションの両方に固有のものになります。大きなワークロードの場合、小さなワークロードよりもさらにスケーリングできます。コストと所要時間のトレードオフを理解して、時間依存のワークロードをより多くのコアでより短時間に実行できます。一方、より少ないコアで最大の効率を発揮して実行することで、コストの節約を実現できます。時間とコストのバランスを取りたい場合、ワークロードはその中間に位置することになります。

定義

4 つの分野のベストプラクティスなど、クラウド内でのコストの最適化:

- 費用対効果の高いリソース
- 需要と供給の一致
- 費用認識
- 時間の経過に伴う最適化

需要と供給の一致、費用認識、および長期的な最適化の領域に関しては、AWS Well-Architected フレームワークホワイトペーパーで説明しています。

ベストプラクティス

トピック

- 費用対効果の高いリソース
- 需要と供給の一致
- 費用認識
- 時間の経過に伴う最適化

費用対効果の高いリソース

HPCCOST 1: コスト最適化を目的とした、ワークロードで利用可能なコンピューティングオプションおよびストレージオプションの評価方法

HPCCOST 2: ジョブの完了時間とコストのトレードオフの評価方法

システムにとって適切なインスタンス、リソース、および機能をを使用することが、コスト管理の鍵になります。インスタンスを選択することよって、HPC ワークロードを実行する全体的なコストが増減する場合があります。たとえば、密結合された HPC ワークロードは、複数の小規模サーバーのクラスターで実行するのに 5 時間かかる場合があります。一方で、少数の大規模サーバーのクラスターは 1 時間あたり 2 倍の費用がかかりますが、結果は 1 時間で計算され、全体として費用が節約されます。ストレージの選択もまた、コストに影響する可能性があります。ジョブの所要時間とコスト最適化の潜在的なトレードオフを考慮した上で、異なるインスタンスサイズとストレージオプションを使用してワークロードをテストし、コストを最適化します。

ニーズに最も適した方法で EC2 インスタンスやその他のサービスを利用できるように、AWS には柔軟でコスト効率が良いさまざまな料金オプションがあります。オンデマンドインスタンスは、最小契約料金の支払いの必要もなく、コンピューティング性能に対して時間単位で利用料金が発生します。リザーブドインスタンスを使用すると、キャパシティーを予約できるようになり、オンデマンド料金と比べて節約できます。スポットインスタンスを使用すると、未使用の Amazon EC2 キャパシティーを活用し、オンデマンド料金に比べてさらに節約できます。

Well-Architected システムでは、最もコスト効率の良いリソースを使用します。また、前処理および後処理にマネージドサービスを使用することにより、コストを削減することもできます。例えば、完了した実行データを保存して後処理するサーバーを管理するのではなく、データを Amazon S3 に保存してから、Amazon EMR または AWS Batch で後処理することができます。

多くの AWS のサービスでは、コストをさらに低減する機能を提供します。例えば、Auto Scaling は EC2 と統合され、ワークロードの要求に基づいてインスタンスを自動的に起動および終了します。FSx for Lustre は S3 とネイティブに統合され、S3 バケットのコンテンツ全体を Lustre ファイルシステムとして提供します。これにより、コスト効率の良い S3 ストレージで長期データを維持しながら、即時のワークロード用に最小限の Lustre ファイルシステムをプロビジョニングすることで、ストレージコストを最適化できます。S3 はさまざまなストレージクラスを提供するため、データに対して最もコスト効率の良いクラスを使用できます。Glacier または Glacier Deep ストレージクラスを使用すると、低コストでデータをアーカイブできます。

さまざまなインスタンスタイプ、ストレージ要件、およびアーキテクチャを試してみると、望ましいパフォーマンスを維持しながら、コストを最小限に抑えられます。

需要と供給の一致

需要と供給の一致のベストプラクティス領域に関して、HPC に固有のベストプラクティスはありません。AWS Well-Architected フレームワークホワイトペーパーで、対応するセクションをご確認ください。

費用認識

費用認識のベストプラクティス領域に関して、HPC に固有のベストプラクティスはありません。AWS Well-Architected フレームワークホワイトペーパーで、対応するセクションをご確認ください。

時間の経過に伴う最適化

長期的な最適化のベストプラクティス領域に関して、HPC に固有のベストプラクティスはありません。AWS Well-Architected フレームワークホワイトペーパーで、対応するセクションをご確認ください。

まとめ

このレンズは、クラウドの高性能コンピューティングワークロード向けに、効率が良く、費用対効果が#く、安全で信頼のおけるシステムを設計して運#するためのアーキテクチャに関するベストプラクティスを提供します。プロトタイプの HPC アーキテクチャと包括的な HPC 設計原則について説明しました。HPC のレンズを通して、Well-Architected の 5 つの柱を再度確認し、既存の、または提案された HPC アーキテクチャのレビューに役立つ一連の質問を提供しました。フレームワークをアーキテクチャに適用すると、安定した効率的なシステムを構築するのに役立ちます。HPC アプリケーションの実行とフィールドの境界を押し広げるのに集中できます。

寄稿者

本ドキュメントは、次の人物および組織が寄稿しました。

- Aaron Bucher、HPC スペシャリストソリューションアーキテクト、アマゾン ウェブ サービス
- Omar Shorbaji、グローバルソリューションアーキテクト、アマゾン ウェブ サービス
- Linda Hedges、HPC アプリケーションエンジニア、アマゾン ウェブ サービス
- Nina Vogl、HPC スペシャリストソリューションアーキテクト、アマゾン ウェブ サービス
- Sean Smith、HPC ソフトウェア開発エンジニア、アマゾン ウェブ サービス
- Kevin Jorissen、ソリューションアーキテクト – 気候および天候、アマゾン ウェブ サービス
- Philip Fitzsimons、シニアマネージャー (Well-Architected)、アマゾン ウェブ サービス

その他の資料

詳細については、以下をご参照ください。

- AWS Well-Architected フレームワーク
- ハイパフォーマンスコンピューティング (https://aws.amazon.com/hpc)
- https://d1.awsstatic.com/whitepapers/Intro_to_HPC_on_AWS.pdf
- AWS での電子設計の自動化 (EDA) ワークフローの最適化
- 実世界における AWS のスケーラビリティ

改訂履歴

このホワイトペーパーの更新に関する通知を受け取るには、RSS フィードをサブスクライブしてください。

変更	説明	日付
ホワイトペーパーの更新	マイナーな更新	December 19, 2019
ホワイトペーパーの更新	マイナーな更新	November 1, 2018
初版発行	ハイパフォーマンスコンピューティングレンズが初めて公開されました。	November 1, 2017

注意

お客様は、この文書に記載されている情報を独自に評価する責任を負うものとします。本書は、(a) 情報提供のみを目的としており、(b) AWS の現行製品と慣行ついて説明していますが、予告なしに変更されることがあり、(c) AWS およびその関連会社、サプライヤーまたはライセンサーからの契約上の義務や保証をもたらすものではありません。AWS の製品やサービスは、明示または暗示を問わず、一切の保証、表明、条件なしに「現状のまま」提供されます。お客様に対する AWS の責任は、AWS 契約により規定されます。本書は、AWS とお客様の間で行われるいかなる契約の一部でもなく、そのような契約の内容を変更するものでもありません。