Livre blanc AWS

Mise en pratique de l'intégration continue/ livraison continue sur AWS

Mise en pratique de l'intégration continue/livraison continue sur AWS: Livre blanc AWS

Table of Contents

Mise en pratique de l'intégration continue/livraison continue sur AWS

Date de publication : 27 octobre 2021 (Révisions du document)

Résumé

Ce document présente les fonctions et les avantages de l'intégration continue/livraison continue (CI/CD) et d'Amazon Web Services (AWS) dans un environnement de développement de logiciels. L'intégration et la livraison continues font partie des bonnes pratiques ; elles constituent un élément essentiel d'une initiative DevOps.

Le défi de la livraison de logiciels

À l'heure actuelle, les entreprises sont confrontées aux difficultés liées à l'évolution rapide de la concurrence, des exigences de sécurité et de l'évolutivité des performances. Elles doivent combler le fossé entre la stabilité des opérations et le développement rapide des fonctions. Les pratiques d'intégration et de livraison continues (CI/CD) permettent des changements logiciels rapides tout en maintenant la stabilité et la sécurité du système.

Amazon a très tôt réalisé que les besoins des entreprises liés à la fourniture de fonctions aux clients revendeurs d'Amazon.com, aux filiales d'Amazon et à Amazon Web Services (AWS) impliquaient d'avoir accès à des méthodes nouvelles et innovantes de livraison de logiciels. À l'échelle d'une entreprise comme Amazon, des milliers d'équipes logicielles indépendantes doivent être en mesure de travailler en parallèle pour fournir des logiciels de manière rapide, sécuritaire et fiable, avec une tolérance zéro face aux pannes.

En apprenant comment fournir rapidement des logiciels, Amazon et d'autres organisations avant-gardistes ont été les pionnières du DevOps. Le DevOps est une combinaison de philosophies culturelles, de pratiques et d'outils qui améliorent la capacité d'une organisation à livrer des applications et des services à un rythme plus soutenu. En appliquant les principes du DevOps, les organisations peuvent faire évoluer et améliorer leurs produits plus rapidement que celles qui ont recours à des processus traditionnels de développement de logiciels et de gestion de l'infrastructure. Elles sont ainsi mieux à même de servir leurs clients et de gagner en compétitivité.

Certains de ces principes, tels que les équipes de 10 à 12 personnes et l'architecture de microservices/orientée services sortent du cadre de ce livre blanc. Ce livre blanc traite de la capacité d'intégration continue/livraison continue (CI/CD) qu'Amazon a développée et améliorée en permanence. L'intégration continue/livraison continue est essentielle pour fournir des fonctions logicielles de manière rapide et fiable.

AWS propose désormais ces fonctionnalités CI/CD sous la forme d'un ensemble de services de développement : AWS CodeStar, AWS CodeCommit, AWS CodePipeline, AWS CodeBuild, AWS CodeDeploy et AWS CodeArtifact. Les développeurs et les professionnels des opérations informatiques qui pratiquent le DevOps peuvent utiliser ces services pour livrer des logiciels de manière rapide, fiable et sécuritaire. Utilisés conjointement, ces services vous aident à stocker le code source de votre application et à réaliser un contrôle de version en toute sécurité. Vous pouvez utiliser AWS CodeStar pour orchestrer rapidement un flux de publication de logiciels de bout en bout à l'aide de ces services. Pour un environnement existant, AWS CodePipeline permet d'intégrer

chaque service indépendamment aux outils que vous utilisez déjà. Il s'agit de services hautement disponibles et facilement intégrés, accessibles via AWS Management Console, les interfaces de programmation d'applications (API) AWS et les kits de développement de logiciels (SDK) AWS, comme tout autre service AWS.

Qu'est-ce que l'intégration continue et la livraison/le déploiement continus ?

Cette section traite des pratiques d'intégration continue et de livraison continue et explique la différence entre la livraison continue et le déploiement continu.

Intégration continue

L'intégration continue (CI) est une pratique de développement logiciel selon laquelle les développeurs fusionnent régulièrement leurs modifications de code dans un référentiel central, après quoi des générations et des tests automatisés sont exécutés. La CI renvoie habituellement à la phase de génération ou d'intégration du processus de lancement de logiciels et implique une composante d'automatisation (par exemple, une CI ou un service de génération) et une composante culturelle (par exemple, apprendre à intégrer fréquemment). Les principaux objectifs de l'intégration continue sont les suivants : trouver et corriger plus rapidement les bogues, améliorer la qualité des logiciels et réduire le temps nécessaire pour valider et publier de nouvelles mises à jour de logiciels.

L'intégration continue se concentre sur des validations et des modifications de plus petites portions du code à intégrer. Un développeur valide le code à intervalles réguliers, au moins une fois par jour. Il extrait le code du référentiel de code pour s'assurer que le code sur l'hôte local est fusionné avant de le transmettre au serveur de développement. À ce stade, le serveur de développement exécute les différents tests et accepte ou rejette le code validé.

Les principales difficultés de la mise en œuvre de l'intégration continue incluent des validations plus fréquentes dans la base de code commune, la gestion d'un référentiel de code source unique, ainsi que l'automatisation des générations et des tests. Parmi les autres difficultés, citons les tests dans des environnements similaires à ceux de la production, la visibilité du processus pour l'équipe et la possibilité pour les développeurs d'obtenir facilement n'importe quelle version de l'application.

Livraison et déploiement continus

La livraison continue (CD) est une méthode de développement de logiciels selon laquelle les changements de code sont automatiquement générés, testés et préparés pour une publication dans un environnement de production. Elle est le prolongement de l'intégration continue à travers le déploiement de toutes les modifications de code dans un environnement de test et/ou de production, une fois la phase de génération terminée. La livraison continue peut être entièrement automatisée

au moyen d'un processus de flux ou partiellement automatisée avec des étapes manuelles aux points critiques. Une livraison continue correctement mise en œuvre permet aux développeurs de toujours disposer d'un artefact de génération prêt au déploiement ayant suivi un processus de test standardisé.

Grâce au déploiement continu, les révisions sont déployées automatiquement dans un environnement de production, sans nécessiter d'approbation explicite de la part d'un développeur. Le processus de lancement de logiciels est alors entièrement automatisé. Ainsi, il est possible de créer une boucle de rétroaction continue des clients dès le début du cycle de vie du produit.

La livraison continue n'est pas un déploiement continu

Une idée fausse concernant la livraison continue consiste à considérer que chaque modification validée est appliquée à la production immédiatement après avoir réussi les tests automatisés. En réalité, la livraison continue n'a pas pour objectif d'appliquer immédiatement chaque modification à la production, mais bien de s'assurer que chaque modification peut être mise en production.

Avant de déployer une modification dans un environnement de production, vous pouvez mettre en œuvre un processus décisionnel afin de vous assurer que le déploiement en production est autorisé et audité. Cette décision peut être prise par une personne, puis exécutée par les outils.

En utilisant la livraison continue, la décision de mise en service devient une décision commerciale et non technique. La validation technique a lieu à chaque validation.

Le déploiement d'une modification en production n'est pas un événement perturbateur. Il n'oblige pas l'équipe technique à arrêter de travailler sur la prochaine série de modifications. En outre, il n'est pas nécessaire de préparer un plan de projet, une documentation de transfert ou un créneau de maintenance. Le déploiement devient un processus reproductible qui a été réalisé et éprouvé à de nombreuses reprises dans des environnements de test.

Avantages de la livraison continue

La livraison continue (CD) présente de nombreux avantages pour une équipe de développement de logiciels, notamment l'automatisation du processus, l'amélioration de la productivité des développeurs, l'amélioration de la qualité du code et la livraison plus rapide des mises à jour à vos clients.

Automatisation du processus de lancement de logiciels

Grâce à la livraison continue, votre équipe a accès à une méthode d'archivage du code qui est automatiquement généré, testé et préparé pour la mise en production, ce qui permet une livraison de logiciels efficace, durable, rapide et sécurisée.

Augmentation de la productivité des développeurs

Les pratiques de livraison continue (CD) contribuent à la productivité de l'équipe en libérant les développeurs des tâches manuelles, en démêlant les dépendances complexes et en remettant l'accent sur la livraison de nouvelles fonctions logicielles. Au lieu d'intégrer leur code à d'autres secteurs de l'entreprise et de passer beaucoup de temps à réfléchir à la façon de déployer ce code sur une plateforme, les développeurs peuvent se concentrer sur une logique de codage qui fournit les fonctions dont vous avez besoin.

Amélioration de la qualité du code

La livraison continue (CD) peut aider à détecter et à résoudre les bogues dès le début du processus de livraison, avant que ces bogues ne se transforment en problèmes plus importants. Votre équipe peut facilement effectuer d'autres types de tests de code, puisque l'ensemble du processus est automatisé. En effectuant davantage de tests plus fréquemment, les équipes peuvent itérer leur code plus rapidement et profiter d'un retour d'information immédiat sur l'impact des changements. Elles peuvent ainsi mettre en œuvre un code de qualité avec une garantie élevée de stabilité et de sécurité. Les développeurs sauront immédiatement si le nouveau code fonctionne et si des changements importants ou des bogues ont été introduits. Les erreurs détectées au début du processus de développement sont les plus faciles à corriger.

Livraison rapide des mises à jour

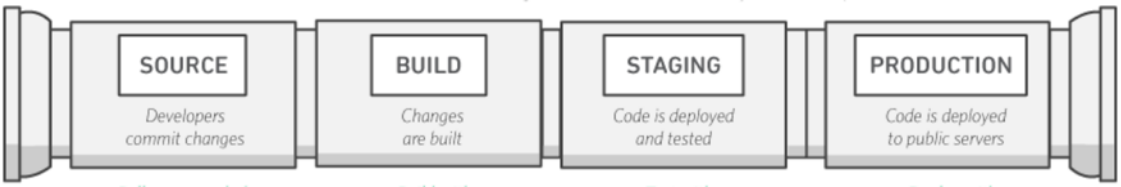
Grâce à la livraison continue, votre équipe peut livrer des mises à jour aux clients rapidement et fréquemment. Si le processus d'intégration continue/livraison continue (CI/CD) est mis en œuvre, il en résulte une vitesse d'exécution accrue de l'ensemble de l'équipe, y compris la publication des fonctions et des corrections de bogues. Les entreprises peuvent réagir plus rapidement aux évolutions du marché, aux défis de sécurité, aux besoins des clients et aux pressions liées aux coûts. Par exemple, si une nouvelle fonction de sécurité est requise, votre équipe peut mettre en œuvre l'intégration continue/livraison continue avec des tests automatisés afin d'introduire le correctif de manière rapide et fiable dans les systèmes de production, en toute confiance. Ce qui prenait auparavant des semaines ou des mois s'accomplit maintenant en quelques jours, voire en quelques heures.

Implémentation de l'intégration et de la livraison continues

Cette section explique comment commencer à mettre en œuvre un modèle d'intégration continue/livraison continue (CI/CD) dans votre organisation. Ce livre blanc n'explique pas comment une organisation dotée d'une initiative DevOps avancée et d'un modèle de transformation du cloud crée et utilise un pipeline CI/CD. Afin de vous aider dans votre transition DevOps, AWS dispose d'un certain nombre de partenaires DevOps certifiés capables d'offrir des ressources et des outils. Pour en savoir plus sur la préparation d'une transition vers le cloud AWS, consultez la section Conception d'un modèle d'exploitation cloud.

Transition vers l'intégration continue/la livraison continue

L'intégration continue/livraison continue (CI/CD) peut être représentée comme un pipeline (voir la figure ci-dessous), où le nouveau code est envoyé d'un côté, testé au cours d'une série de phases (source, génération, intermédiaire et production), puis publié en tant que code prêt pour la production. Si votre organisation est novice en matière d'intégration continue/livraison continue, elle peut aborder ce pipeline de manière itérative. Cela signifie que vous devez commencer modestement, puis itérer le processus à chaque phase afin de pouvoir comprendre et développer votre code de manière à contribuer à la croissance de l'organisation.



Pipeline d'intégration continue/livraison continue (CI/CD)

Chaque phase du pipeline CI/CD est structurée comme une unité logique dans le processus de livraison. En outre, chaque phase agit comme une barrière qui vérifie un certain aspect du code. Au fur et à mesure que le code progresse dans le pipeline, on suppose que le code est de meilleure qualité aux phases ultérieures, car d'autres aspects de ce code continuent d'être vérifiés. Les problèmes décelés à un stade précoce empêchent la progression du code dans le pipeline. Les résultats des tests sont immédiatement envoyés à l'équipe, et toutes les versions et tous les lancements sont interrompus si le logiciel échoue à une phase.

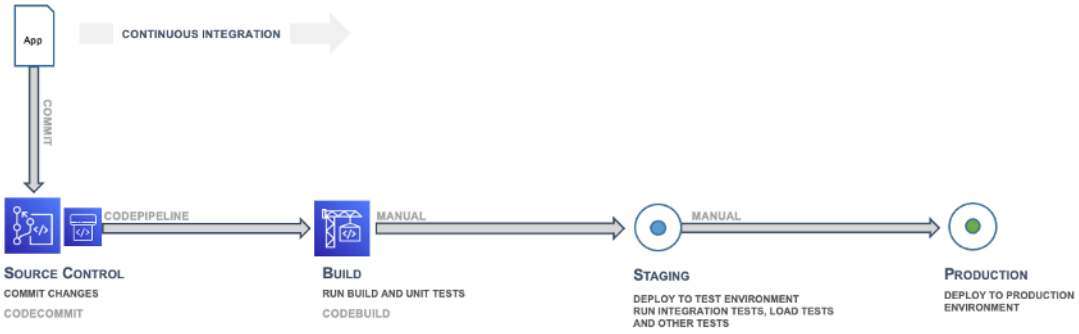
Ces phases sont des suggestions. Vous pouvez les adapter en fonction des besoins de votre entreprise. Certaines phases peuvent être répétées pour différents types de tests, de sécurité et de performances. Selon la complexité de votre projet et la structure de vos équipes, certaines phases

peuvent être répétées plusieurs fois à différents niveaux. Par exemple, le produit final d'une équipe peut devenir une dépendance dans le projet de l'équipe suivante. Cela signifie que le produit final de la première équipe est ensuite testé en tant qu'artefact dans le projet de l'équipe suivante.

La présence d'un pipeline CI/CD aura un impact important sur la maturation des capacités de votre organisation. L'organisation doit commencer progressivement et ne pas essayer de créer un pipeline entièrement mature, avec plusieurs environnements, de nombreuses phases de test et une automatisation à toutes les phases dès le début. N'oubliez pas que même les organisations qui disposent d'environnements CI/CD hautement matures doivent continuer d'améliorer leurs pipelines en permanence.

Le développement d'une organisation compatible avec l'intégration continue/livraison continue est un cheminement, avec de nombreuses destinations en cours de route. La prochaine section traite d'un cheminement possible pouvant être suivi par votre organisation, en commençant par l'intégration continue jusqu'aux niveaux de livraison continue.

Intégration continue



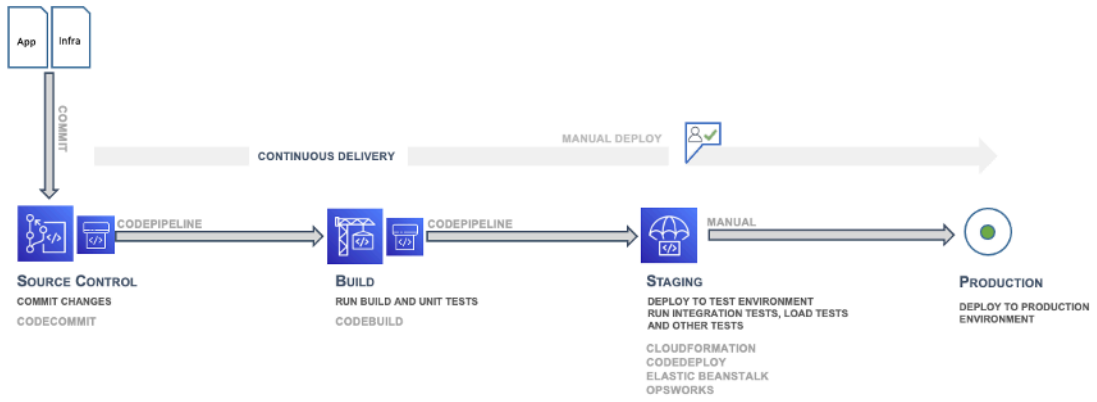
Intégration continue : source et génération

La première phase du parcours d'intégration continue/livraison continue d'une organisation consiste pour celle-ci à devenir mature en matière d'intégration continue. Vous devez vous assurer que tous les développeurs valident régulièrement leur code dans un référentiel central (tel qu'un référentiel hébergé dans CodeCommit ou GitHub) et fusionnent toutes les modifications apportées dans une branche de publication de l'application. Aucun développeur ne doit exploiter son code isolément. Si une branche de fonctions est nécessaire pendant un certain temps, elle doit être mise à jour en fusionnant le code en amont aussi souvent que possible. Il est recommandé de procéder fréquemment à des validations et à des fusions avec des unités de travail complètes, afin que l'équipe en prenne l'habitude. Par ailleurs, cela est encouragé par le processus. Un développeur qui fusionne le code tôt et souvent aura probablement moins de problèmes d'intégration à l'avenir.

Vous devez également encourager les développeurs à créer des tests unitaires pour leurs applications le plus tôt possible et à les exécuter avant de transmettre le code au référentiel central. Les erreurs détectées au début du processus de développement logiciel sont les moins chères et les plus faciles à corriger.

Lorsque le code est transmis à une branche d'un référentiel de code source, un moteur de flux surveillant cette branche envoie une commande à un outil de génération afin de générer le code et d'exécuter les tests unitaires dans un environnement contrôlé. Le processus de génération doit être dimensionné de manière appropriée de façon à gérer toutes les activités, y compris les transmissions et les tests qui peuvent avoir lieu pendant la phase de validation, pour un retour rapide. D'autres contrôles de qualité, tels que la couverture des tests unitaires, le contrôle de style et l'analyse statique, peuvent également avoir lieu à ce stade. Enfin, l'outil de génération crée une ou plusieurs versions binaires et d'autres artefacts, tels que des images, des feuilles de style et des documents pour l'application.

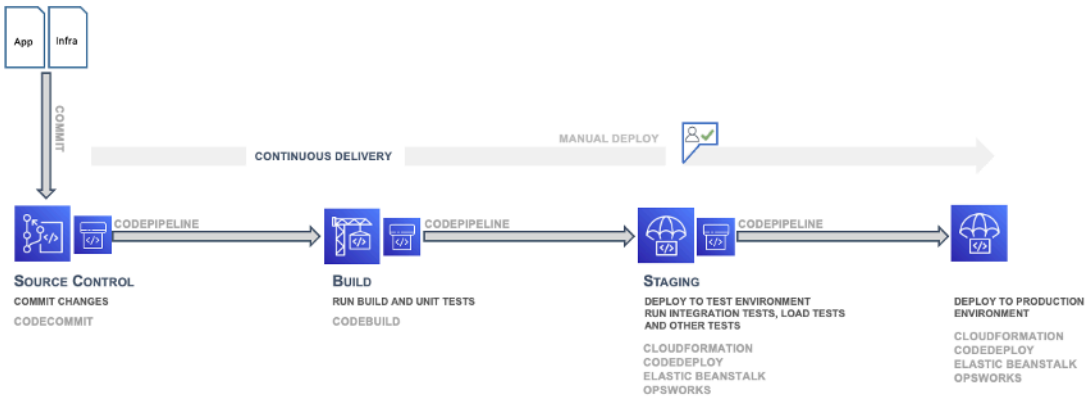
Livraison continue : création d'un environnement intermédiaire



Livraison continue : phase intermédiaire

La prochaine étape est la livraison continue (CD). Elle implique le déploiement du code d'application dans un environnement intermédiaire, qui est une réplique de la pile de production, et l'exécution de tests fonctionnels supplémentaires. L'environnement intermédiaire peut être un environnement statique préconçu pour les tests. Vous pouvez aussi allouer et configurer un environnement dynamique avec une infrastructure dédiée et un code de configuration pour tester et déployer le code d'application.

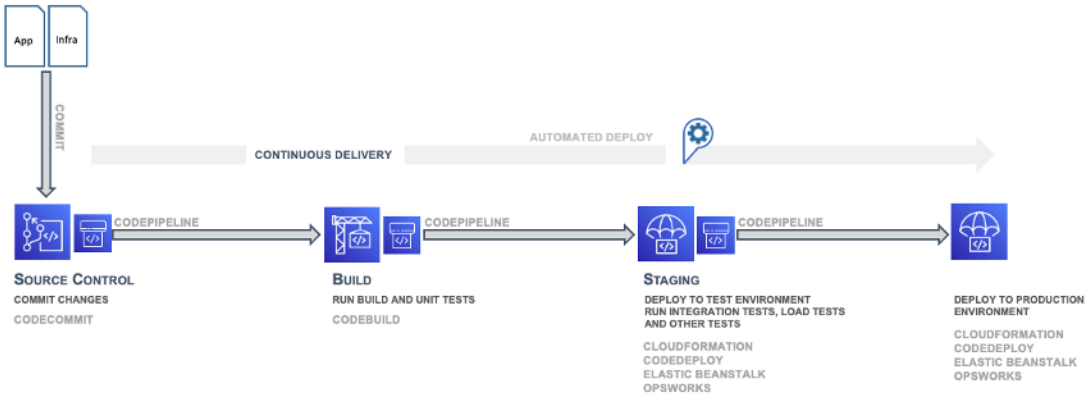
Livraison continue : création d'un environnement de production



Livraison continue : production

Dans la séquence du pipeline de déploiement/livraison, après l'environnement intermédiaire vient l'environnement de production, qui est également créé à l'aide de l'infrastructure IaC (Infrastructure as Code).

Déploiement continu



Déploiement continu

Le déploiement continu constitue la dernière phase du pipeline CI/CD. Il peut inclure l'automatisation complète de l'ensemble du processus de lancement de logiciels, y compris le déploiement dans l'environnement de production. Dans un environnement CI/CD entièrement mature, la progression vers l'environnement de production est entièrement automatisée, ce qui permet de déployer le code en toute confiance.

Maturité et au-delà

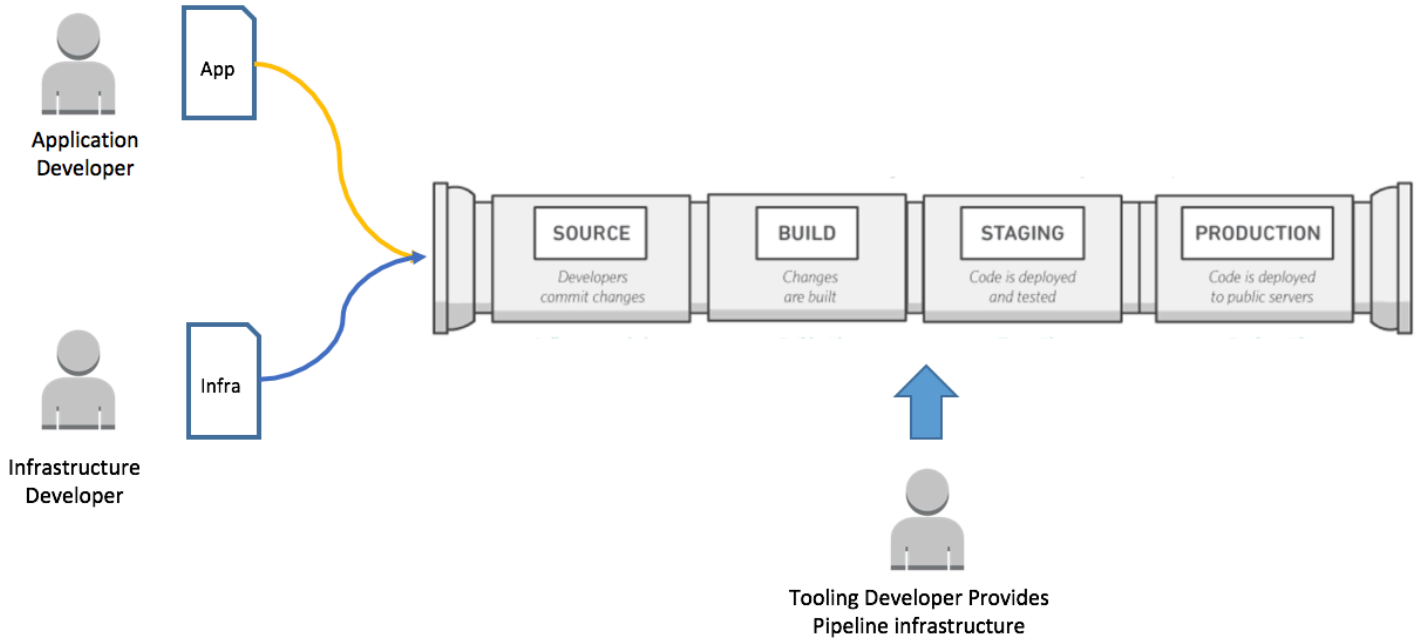
Au fur et à mesure que votre organisation croît en maturité, elle continuera de développer le modèle CI/CD afin d'inclure davantage des améliorations suivantes :

- Davantage d'environnements intermédiaires pour des tests spécifiques de performances, de conformité, de sécurité et d'interface utilisateur
- Tests unitaires de l'infrastructure et du code de configuration avec le code d'application
- Intégration à d'autres systèmes et processus tels que la révision du code, le suivi des problèmes et la notification des événements
- Intégration avec la migration du schéma de base de données (le cas échéant)
- Étapes supplémentaires pour l'audit et l'approbation des activités

Même les organisations les plus matures qui ont des pipelines CI/CD multienvironnements complexes cherchent sans cesse à les améliorer. Le DevOps est un cheminement, pas une destination. Les commentaires au sujet du pipeline sont continuellement recueillis ; une collaboration entre les différents groupes des équipes de développement permettent des améliorations en termes de vitesse, d'échelle, de sécurité et de fiabilité.

Équipes

AWS recommande d'organiser trois équipes de développement pour la mise en œuvre d'un environnement CI/CD : une équipe chargée de l'application, une équipe chargée de l'infrastructure et une équipe chargée des outils (voir la figure ci-dessous). Cette organisation représente un ensemble de bonnes pratiques qui ont été mises au point et appliquées dans les start-up en évolution rapide, les grandes entreprises et dans Amazon même. Les équipes ne doivent pas comprendre plus de 10 à 12 participants, conformément à la règle de communication selon laquelle les conversations significatives atteignent leurs limites lorsque la taille des groupes augmente et que les lignes de communication se multiplient.



Équipes chargées de l'application, de l'infrastructure et des outils

Équipe chargée de l'application

Cette équipe crée l'application. Les développeurs d'applications sont propriétaires du backlog, des scénarios et des tests unitaires. Ils développent en outre des fonctions selon une cible d'application spécifique. Cette équipe a pour objectif organisationnel de réduire au minimum le temps que ces développeurs consacrent à des tâches applicatives non essentielles.

Outre ses compétences en programmation fonctionnelle dans le langage d'application, l'équipe chargée de l'application doit posséder des compétences en matière de plateforme et une compréhension de la configuration du système. Cela lui permettra de se concentrer uniquement sur le développement de fonctions et le renforcement de l'application.

Équipe chargée de l'infrastructure

L'équipe chargée de l'infrastructure écrit le code qui crée et configure l'infrastructure nécessaire à l'exécution de l'application. Cette équipe peut utiliser des outils AWS natifs, tels que AWS CloudFormation, ou des outils génériques, tels que Chef, Puppet ou Ansible. L'équipe chargée de l'infrastructure doit spécifier quelles sont les ressources nécessaires et travaille en étroite collaboration avec l'équipe chargée de l'application. Pour une application de petite taille, l'équipe chargée de l'infrastructure peut être composée d'une ou deux personnes seulement.

Elle doit avoir des compétences dans les méthodes de mise en service de l'infrastructure, telles que AWS CloudFormation ou HashiCorp Terraform. L'équipe doit également acquérir des compétences en automatisation de la configuration à l'aide d'outils tels que Chef, Ansible, Puppet ou Salt.

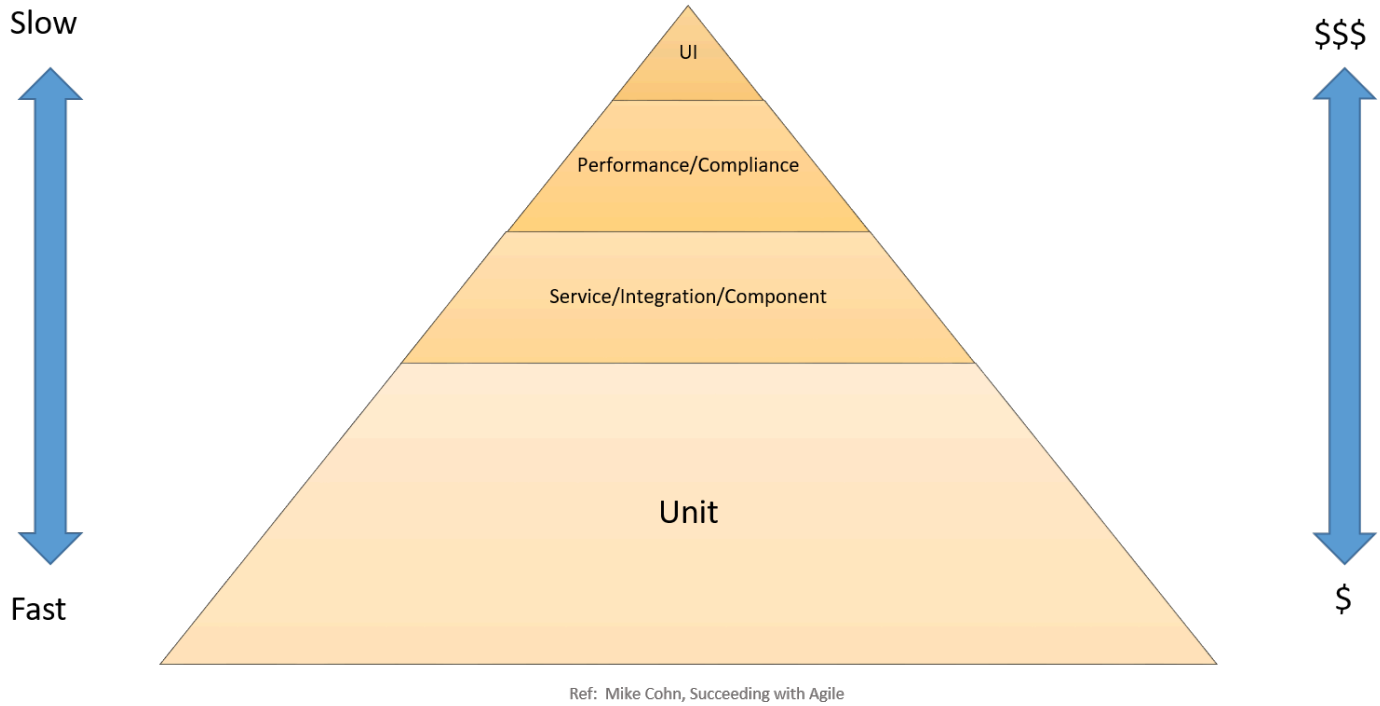
Équipe chargée des outils

L'équipe chargée des outils génère et gère le pipeline CI/CD. Elle est responsable de l'infrastructure et des outils qui composent le pipeline. Elle ne fait pas partie des équipes chargées de l'application et de l'infrastructure de 10 à 12 participants ; elle crée toutefois un outil qui est utilisé par ces deux équipes. L'organisation doit continuellement faire évoluer son équipe chargée des outils, afin que celle-ci garde une longueur d'avance sur les équipes d'application et d'infrastructure en phase de maturation.

L'équipe chargée des outils doit être compétente en matière de génération et d'intégration de toutes les parties du pipeline CI/CD. Cela inclut la génération de référentiels de contrôle de code source, de moteurs de flux, d'environnements de génération, de structures de tests et de référentiels d'artefacts. Cette équipe peut choisir d'implémenter des logiciels tels qu'AWS CodeStar, AWS CodePipeline, AWS CodeCommit, AWS CodeDeploy, AWS CodeBuild et AWS CodeArtifact, ainsi que Jenkins, GitHub, Artifactory, TeamCity et d'autres outils similaires. Dans certaines organisations, ces équipes sont désignées comme des équipes DevOps. Toutefois, AWS le déconseille et encourage plutôt à considérer le DevOps comme la somme des personnes, des processus et des outils impliqués dans la livraison de logiciels.

Phases de test de l'intégration continue et de la livraison continue

Les trois équipes d'intégration continue/livraison continue doivent intégrer les tests dans le cycle de vie du développement logiciel aux différentes phases du pipeline CI/CD. Dans l'ensemble, les tests doivent commencer le plus tôt possible. La pyramide de tests ci-dessous est un concept fourni par Mike Cohn dans son ouvrage Succeeding with Agile. Elle illustre les différents tests logiciels en fonction de leur coût et de leur rapidité d'exécution.



Pyramide de tests CI/CD

Les tests unitaires se trouvent au bas de la pyramide. Ils sont les plus rapides à exécuter et les moins chers. Par conséquent, les tests unitaires doivent constituer l'essentiel de votre stratégie de tests. En règle générale, ils doivent représenter environ 70 % des tests. Les tests unitaires doivent avoir une couverture de code quasi complète car les bogues détectés à cette phase peuvent être corrigés rapidement et à moindre coût.

Dans la pyramide, les tests de service, de composants et d'intégration se situent au-dessus des tests unitaires. Ils nécessitent des environnements détaillés et sont donc plus coûteux en termes d'infrastructure et plus lents à exécuter. Les tests de performances et de conformité constituent le niveau supérieur. Ils nécessitent des environnements de qualité de production et sont encore plus coûteux. Les tests de l'interface utilisateur et d'acceptation par les utilisateurs se situent au sommet de la pyramide. Ils nécessitent eux aussi des environnements de qualité de production.

Tous ces tests s'inscrivent dans le cadre d'une stratégie complète visant à produire un logiciel de haute qualité. Cependant, dans une optique de développement rapide, l'accent est mis sur le nombre de tests et la couverture dans la moitié inférieure de la pyramide.

Les sections ci-dessous traitent des phases d'intégration continue/livraison continue.

Configuration de la source

Au début du projet, il est essentiel de configurer une source dans laquelle vous pouvez stocker votre code brut, ainsi que les modifications de configuration et de schéma. À la phase source, choisissez un référentiel de code source, tel qu'un référentiel hébergé dans GitHub ou AWS CodeCommit.

Configuration et exécution de générations

L'automatisation des générations est essentielle au processus d'intégration continue. Lors de la configuration de l'automatisation des générations, la première tâche consiste à choisir l'outil de génération adapté. Il existe de nombreux outils de génération, tels que :

- Ant, Maven et Gradle pour Java
- Make pour C/C++
- Grunt pour JavaScript
- Rake pour Ruby

L'outil de génération qui vous convient le mieux dépend du langage de programmation de votre projet et des compétences de votre équipe. Après avoir choisi l'outil de génération, toutes les dépendances et les phases de génération doivent être clairement définies dans les scripts de génération. Il est également recommandé de créer une version des artefacts de la génération finale, ce qui facilite le déploiement et le suivi des problèmes.

Phase de génération

Lors de la phase de génération, les outils de génération considèrent comme une entrée toute modification apportée au référentiel du code source, génèrent le logiciel, puis exécutent les types de tests suivants :

Tests unitaires : testent une section de code spécifique pour s'assurer que le code agit comme il est censé le faire. Les tests unitaires sont effectués par les développeurs de logiciels pendant la phase de développement. À ce stade, une analyse de code statique, une analyse du flux de données, une couverture de code et d'autres processus de vérification logicielle peuvent être appliqués.

Analyse de code statique : ce test se réalise sans exécuter réellement l'application après la génération et les tests unitaires. Cette analyse peut aider à détecter les erreurs de codage et les failles de sécurité ; elle peut aussi garantir la conformité aux directives de codage.

Phase intermédiaire

Au cours de la phase intermédiaire, des environnements complets sont créés pour refléter l'environnement de production final. Les tests suivants sont réalisés :

Tests d'intégration : contrôlent les interfaces entre les composants par rapport à la conception du logiciel. Il s'agit d'un processus itératif qui facilite la génération d'interfaces robustes et l'intégrité du système.

Tests des composants : testent le passage des messages entre les différents composants et leurs résultats. Ceci permet entre autres de contrôler l'idempotence des tests de composants. Les tests peuvent inclure des volumes de données extrêmement importants ou des situations périphériques et des entrées anormales.

Tests du système : testent le système de bout en bout et vérifie si le logiciel répond aux exigences de l'entreprise. Cela peut inclure le test de l'interface utilisateur (IU), de l'API, de la logique du backend et de l'état final.

Tests de performances : déterminent la réactivité et la stabilité d'un système lorsqu'il fonctionne sous une charge de travail particulière. Les tests de performances servent également à étudier, à mesurer, à valider ou à vérifier d'autres attributs de qualité du système, tels que la capacité de mise à l'échelle, la fiabilité et l'utilisation des ressources. Il peut s'agir de tests de charge, de tests de résistance et de tests de pics de charge. Ils permettent de procéder à une analyse comparative en fonction de critères prédéfinis.

Tests de conformité : vérifient si le changement de code est conforme aux exigences d'une spécification et/ou d'une réglementation non fonctionnelle. Ils déterminent si vous mettez en œuvre et respectez les normes définies.

Tests d'acceptation par les utilisateurs : valident le flux d'entreprise de bout en bout. Ce test est exécuté par un utilisateur final dans un environnement intermédiaire et confirme si le système répond aux exigences spécifiées. En règle générale, les clients utilisent à ce stade des méthodologies de test alpha et bêta.

Phase de production

Enfin, une fois les tests précédents réussis, la phase intermédiaire est répétée dans un environnement de production. Au cours de cette phase, un test canary final peut être réalisé en déployant le nouveau code uniquement sur un petit sous-ensemble de serveurs, voire un seul

serveur, ou dans une Région AWS avant de le déployer dans l'ensemble de l'environnement de production. Les détails sur le déploiement en production en toute sécurité sont abordés dans la section Méthodes de déploiement.

La section suivante traite de la création du pipeline pour intégrer ces phases et ces tests.

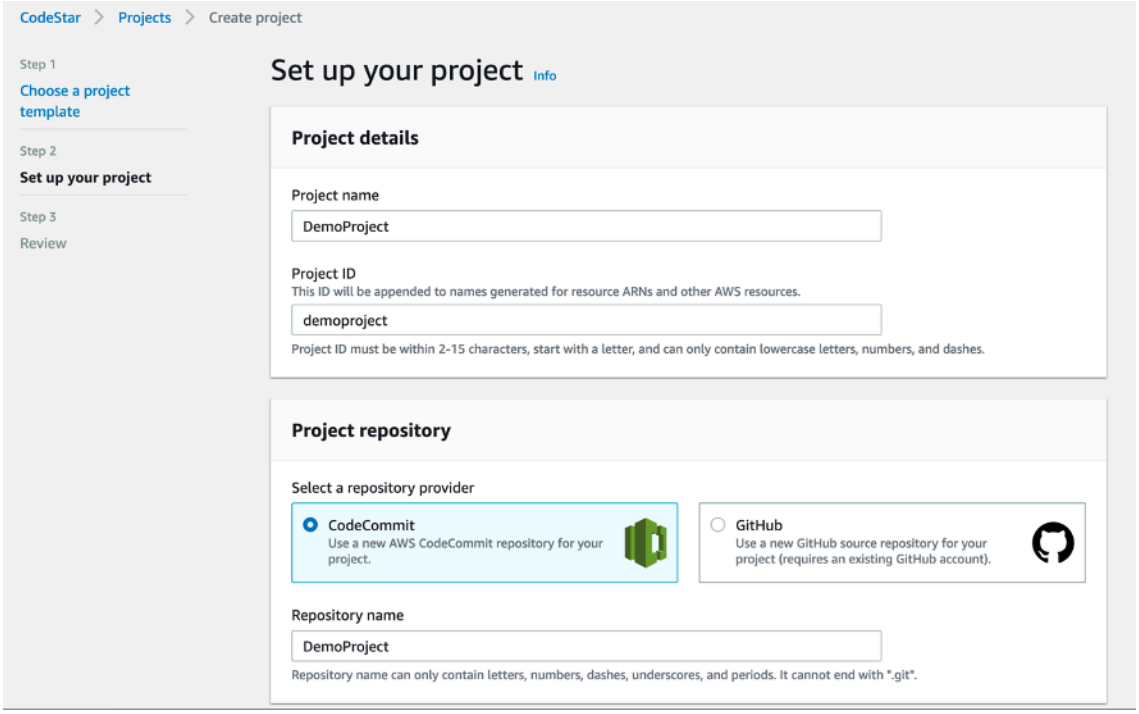
Création du pipeline

Cette section traite de la création du pipeline. Commencez par établir un pipeline comprenant uniquement les composants nécessaires à l'intégration continue, puis passez ensuite à un pipeline de livraison continue avec plus de composants et de phases. Cette section explique également comment il est possible d'utiliser les fonctions AWS Lambda et les approbations manuelles pour les grands projets, planifier plusieurs équipes, branches et Régions AWS.

Création initiale d'un pipeline minimum viable pour l'intégration continue

Le cheminement de votre organisation vers la livraison continue commence par un pipeline minimum viable (MVP). Comme indiqué dans la section Implémentation de l'intégration et de la livraison continues, les équipes peuvent commencer par un processus très simple, comme la mise en œuvre d'un pipeline qui réalise un contrôle du style de code ou un test unitaire unique sans déploiement.

Un outil d'orchestration de la livraison continue en est l'un des principaux composants. Pour vous aider à créer ce pipeline, Amazon a développé AWS CodeStar.



Page de configuration d'AWS CodeStar

AWS CodeStar utilise AWS CodePipeline, AWS CodeBuild, AWS CodeCommit et AWS CodeDeploy avec un processus de configuration intégré, des outils, des modèles et un tableau de bord. Il fournit les outils nécessaires pour développer, générer et déployer rapidement des applications sur AWS. Vous pouvez ainsi commencer à publier du code plus rapidement. Les clients qui utilisent déjà AWS Management Console et recherchent un niveau de contrôle plus élevé peuvent configurer manuellement les outils de développement de leur choix et allouer des services AWS individuels selon leurs besoins.

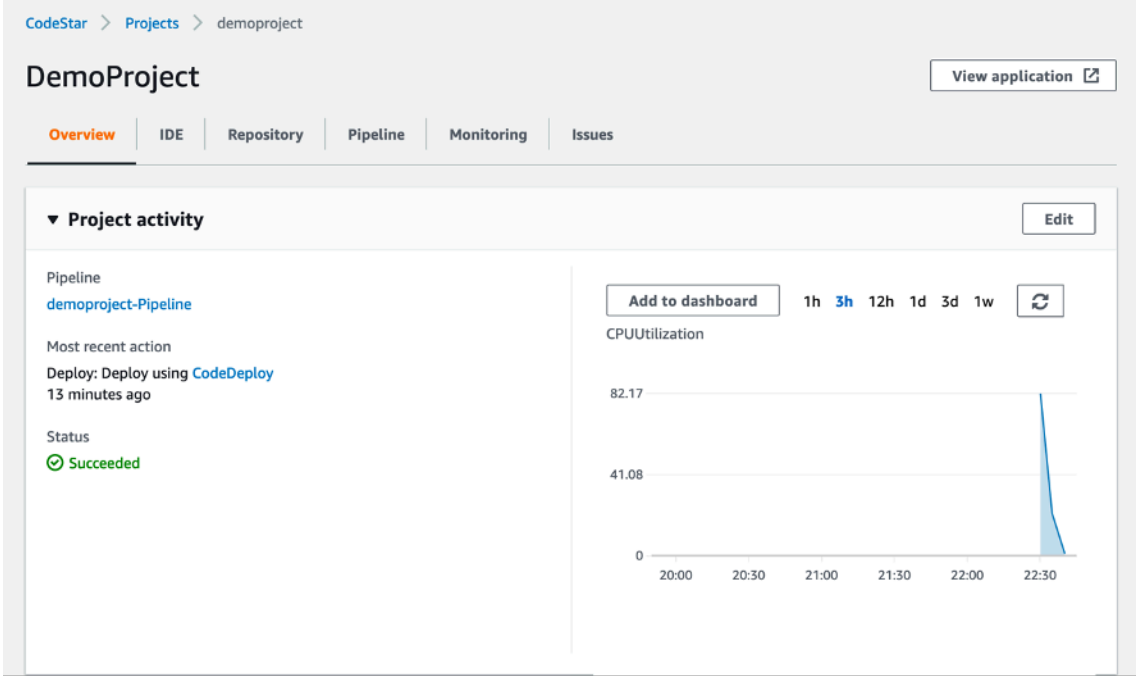
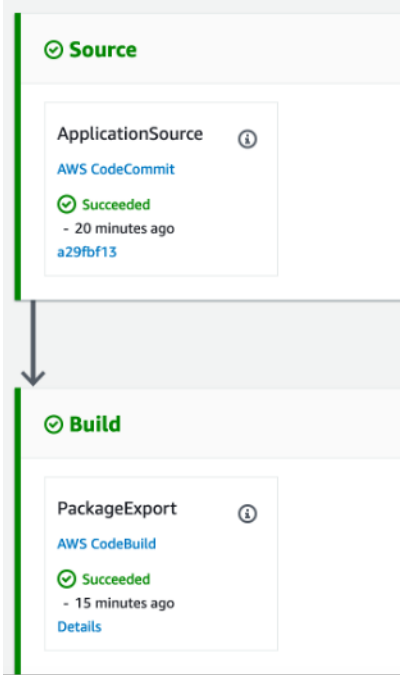


Tableau de bord AWS CodeStar

AWS CodePipeline est un service CI/CD qui peut être utilisé via AWS CodeStar ou via AWS Management Console pour des mises à jour rapides et fiables des applications et de l'infrastructure. AWS CodePipeline génère, teste et déploie le code chaque fois que celui-ci est modifié, en fonction des modèles de processus de lancement que vous définissez. Il est ainsi possible de proposer rapidement des fonctions et des mises à jour en toute sécurité. Vous pouvez facilement créer une solution de bout en bout grâce à nos plugins préintégrés pour des services tiers fréquemment utilisés, comme GitHub, ou en intégrant vos propres plugins personnalisés, à tous les stades de votre processus de lancement. Avec AWS CodePipeline, vous payez seulement ce que vous utilisez. Cela n'implique aucun paiement initial ni engagement à long terme.

Les phases d'AWS CodeStar et d'AWS CodePipeline sont directement corrélées aux phases source, de génération, intermédiaire et de production de l'intégration continue/livraison continue. Bien que la

livraison continue soit souhaitable, vous pouvez commencer par un simple pipeline en deux phases qui contrôle le référentiel source et exécute une action de génération :



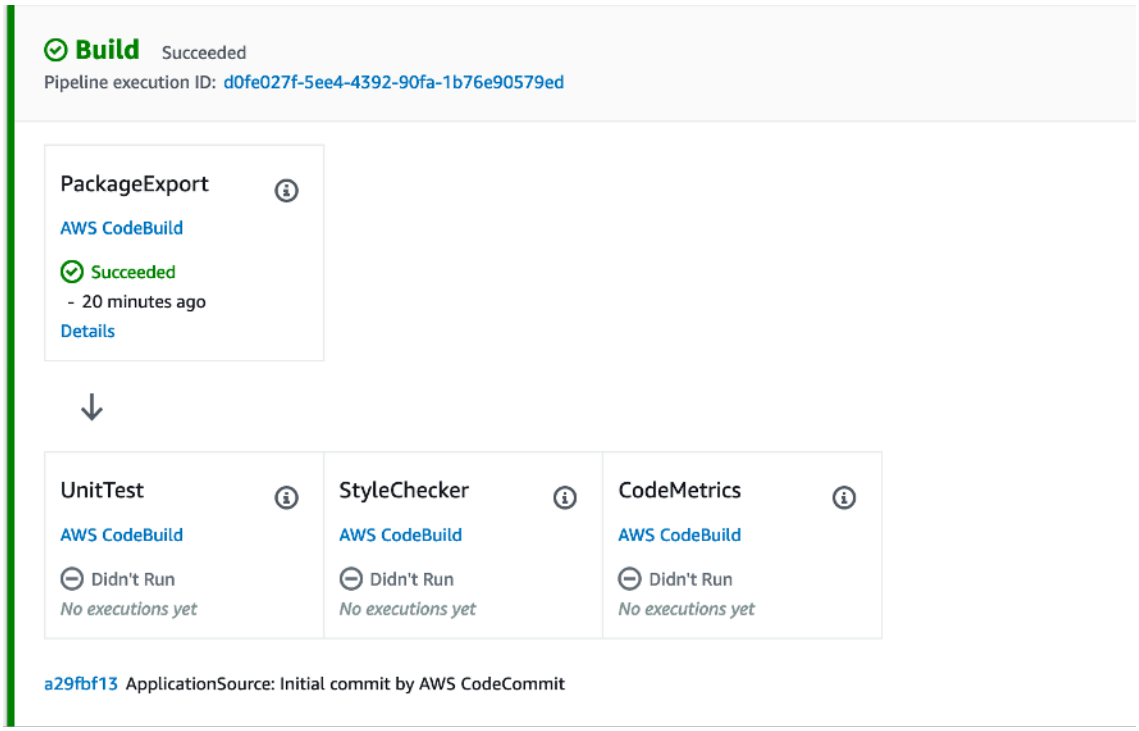
AWS CodePipeline – phases source et de génération

Dans le cas d'AWS CodePipeline, la phase source peut accepter des entrées provenant de GitHub, d'AWS CodeCommit et d'Amazon Simple Storage Service (Amazon S3). L'automatisation du processus de génération est une première phase essentielle pour mettre en œuvre la livraison continue et passer à un déploiement continu. En éliminant les interventions humaines dans la production d'artefacts de génération, vous réduisez la charge de travail de votre équipe, minimisez les erreurs introduites par la génération manuelle de packages et pouvez commencer à créer des packages des artefacts consommables plus fréquemment.

AWS CodePipeline fonctionne de manière transparente avec AWS CodeBuild, un service de génération entièrement géré, afin de faciliter la mise en place au sein de votre pipeline d'une phase de génération qui crée des packages de votre code et exécute des tests unitaires. Grâce à AWS CodeBuild, vous n'avez pas besoin d'allouer, de gérer ou de mettre à l'échelle vos propres serveurs de développement. AWS CodeBuild effectue une mise à l'échelle continue et traite simultanément plusieurs générations afin que celles-ci ne restent pas en file d'attente. AWS CodePipeline s'intègre également à des serveurs de développement tels que Jenkins, Solano CI et TeamCity.

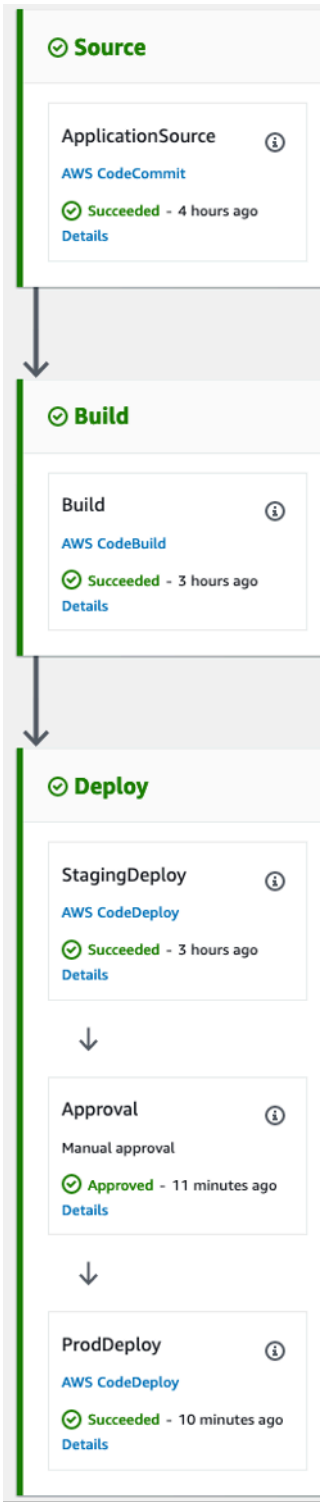
Par exemple, dans la phase de génération ci-dessous, trois actions (tests unitaires, contrôles de style de code et collecte des métriques de code) s'exécutent en parallèle. À l'aide d'AWS CodeBuild, ces

étapes peuvent être ajoutées comme nouveaux projets sans aucun effort supplémentaire, afin de créer ou d'installer des serveurs de développement pour gérer la charge.



AWS CodePipeline – fonctionnalité de génération

Les phases source et de génération illustrées dans la figure AWS CodePipeline – phases source et de génération, ainsi que les processus et l'automatisation connexes, aident à la transition de votre équipe vers une intégration continue. À ce niveau de maturité, les développeurs doivent régulièrement prêter attention aux résultats de génération et de test. Ils doivent également mettre en place et tenir à jour une base de tests unitaires saine. Cela contribue à renforcer la confiance de l'ensemble de l'équipe envers le pipeline CI/CD et favorise l'adoption de ce dernier.



Phases AWS CodePipeline

Pipeline de livraison continue

Une fois que le pipeline d'intégration continue a été mis en œuvre et que les processus connexes ont été établis, vos équipes peuvent commencer la transition vers le pipeline de livraison continue. Cette transition nécessite que les équipes automatisent la génération et le déploiement des applications.

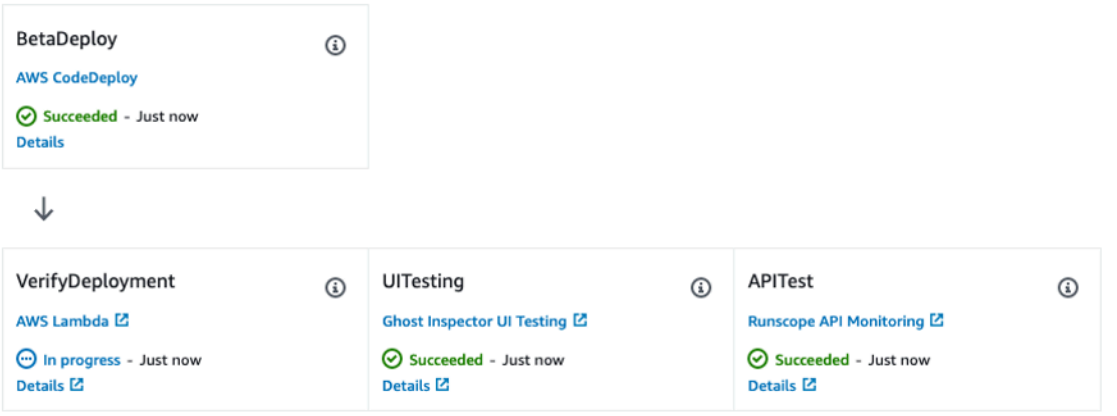
Un pipeline de livraison continue se caractérise par la présence de phases intermédiaires et de production, la phase de production étant réalisée après une approbation manuelle.

De la même manière que pour la mise en place du pipeline d'intégration continue, vos équipes peuvent progressivement commencer à créer un pipeline de livraison continue en écrivant leurs scripts de déploiement.

Selon les besoins d'une application, certaines phases de déploiement peuvent être abstraites par les services AWS en place. Par exemple, AWS CodePipeline s'intègre directement à AWS CodeDeploy, un service qui automatise les déploiements de code sur les instances Amazon EC2 et les instances exécutées sur site, à AWS OpsWorks, un service de gestion de la configuration qui aide à exploiter des applications à l'aide de Chef, et à AWS Elastic Beanstalk, un service de déploiement et de mise à l'échelle des applications et services web.

AWS s'accompagne d'une documentation détaillée sur la façon de mettre en œuvre et d'intégrer AWS CodeDeploy à l'infrastructure et au pipeline.

Une fois que l'équipe a réussi à automatiser le déploiement de l'application, les phases de déploiement peuvent être étendues à l'aide de différents tests. Par exemple, vous pouvez ajouter d'autres intégrations prêtes à l'emploi à des services tels que Ghost Inspector ou Runscope, entre autres, comme illustré dans la figure ci-dessous.



AWS CodePipeline – tests du code aux phases de déploiement

Ajout d'actions Lambda

AWS CodeStar et AWS CodePipeline prennent en charge l'intégration à AWS Lambda. Cette intégration permet de mettre en œuvre un large éventail de tâches, telles que la création de ressources personnalisées dans votre environnement, l'intégration à des systèmes tiers (tels que Slack) et l'exécution de contrôles de l'environnement récemment déployé.

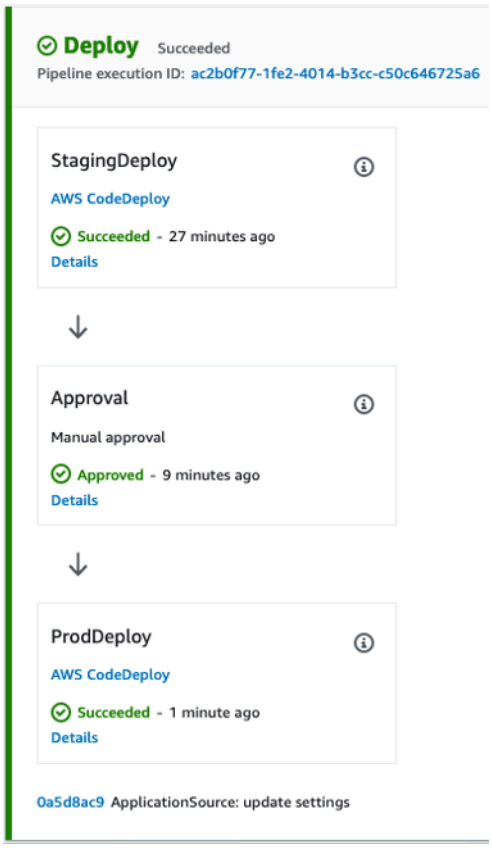
Les fonctions Lambda peuvent être utilisées dans les pipelines CI/CD pour effectuer les tâches suivantes :

- Déploiement des changements dans votre environnement en appliquant ou en mettant à jour un modèle AWS CloudFormation.
- Création de ressources à la demande dans une phase d'un pipeline à l'aide d'AWS CloudFormation et suppression de ces ressources à une autre phase.
- Déploiement de versions d'application sans aucune interruption dans AWS Elastic Beanstalk avec une fonction Lambda qui échange des valeurs CNAME (Canonical Name Record).
- Déploiement d'instances Amazon Elastic Container Service (ECS) Docker.
- Sauvegarde de ressources avant une génération ou un déploiement en créant un instantané de l'AMI.
- Ajout de l'intégration à des produits tiers à votre pipeline, comme l'envoi de messages à un client IRC (Internet Relay Chat).

Approbations manuelles

Ajoutez une action d'approbation à une phase du pipeline au point où l'exécution du pipeline doit s'interrompre, de sorte qu'une personne dotée des autorisations Gestion des identités et des accès AWS requises puisse approuver ou rejeter l'action.

Si l'action est approuvée, l'exécution du pipeline reprend. Si l'action est rejetée, ou si personne n'approuve ou ne rejette l'action sous sept jours après que le pipeline a atteint l'action et a été interrompu, le résultat sera le même que si l'action était en échec, et l'exécution du pipeline cesse.



AWS CodeDeploy – Approbations manuelles

Déploiement des modifications du code d'infrastructure dans un pipeline CI/CD

AWS CodePipeline permet de sélectionner AWS CloudFormation comme une action de déploiement à n'importe quelle phase du pipeline. Vous pouvez ensuite choisir l'action spécifique qu'AWS CloudFormation doit exécuter, telle que la création ou la suppression de piles et la création ou l'exécution de jeux de modifications. Une pile est un concept AWS CloudFormation ; elle représente un groupe de ressources AWS associées. L'infrastructure IaC (Infrastructure as Code) peut être allouée de différentes façons. Toutefois, AWS CloudFormation est un outil exhaustif recommandé par AWS en tant que solution complète et évolutive capable de décrire l'ensemble de ressources AWS le plus complet sous forme de code. AWS recommande d'utiliser AWS CloudFormation dans un projet AWS CodePipeline pour suivre les modifications et les tests d'infrastructure.

Intégration continue/livraison continue pour les applications sans serveur

Vous pouvez également utiliser AWS CodeStar, AWS CodePipeline, AWS CodeBuild et AWS CloudFormation afin de créer des pipelines CI/CD pour les applications sans serveur. Les

applications sans serveur intègrent des services gérés tels qu'Amazon Cognito, Amazon S3 et Amazon DynamoDB à un service guidé par les événements ; AWS Lambda déploie les applications d'une manière qui ne nécessite pas de gérer les serveurs. Si vous êtes un développeur d'applications sans serveur, vous pouvez utiliser conjointement AWS CodePipeline, AWS CodeBuild et AWS CloudFormation afin d'automatiser la génération, le test et le déploiement d'applications sans serveur qui sont exprimées dans des modèles créés d'après le modèle d'application sans serveur d'AWS. Pour en savoir plus, consultez la documentation d'AWS Lambda relative à l'automatisation du déploiement d'applications basées sur Lambda.

Vous pouvez également créer des pipelines CI/CD sécurisés conformes aux bonnes pratiques de votre organisation grâce à AWS Serverless Application Model Pipelines (AWS SAM Pipelines). AWS SAM Pipelines est une nouvelle fonction d'AWS SAM CLI qui permet de profiter en quelques minutes des avantages de l'intégration continue/livraison continue, notamment l'accélération de la fréquence des déploiements, ainsi que la diminution des délais d'implémentation des modifications et la réduction des erreurs de déploiement. AWS SAM Pipelines s'accompagne d'un ensemble de modèles de pipelines par défaut pour AWS CodeBuild/CodePipeline conformes aux bonnes pratiques de déploiement d'AWS. Pour en savoir plus et pour visionner le didacticiel, consultez le blog sur la présentation d'AWS SAM Pipelines.

Pipelines pour plusieurs équipes, branches et régions AWS

Dans le cadre d'un projet de grande envergure, il n'est pas rare que plusieurs équipes de projet travaillent sur différents composants. Si plusieurs équipes utilisent un seul référentiel de code, celui-ci peut être mappé de telle sorte que chaque équipe possède sa propre branche. Ce projet doit également comprendre une branche d'intégration ou de publication pour la fusion finale du projet. Si une architecture de microservices ou orientée services est utilisée, chaque équipe peut disposer de son propre référentiel de code.

Dans le premier scénario, si un seul pipeline est utilisé, il est possible que la progression de plusieurs équipes soit gênée lorsqu'une autre équipe bloque le pipeline. AWS recommande de créer des pipelines spécifiques aux branches de l'équipe et un autre pipeline de publication pour la livraison finale du produit.

Intégration du pipeline à AWS CodeBuild

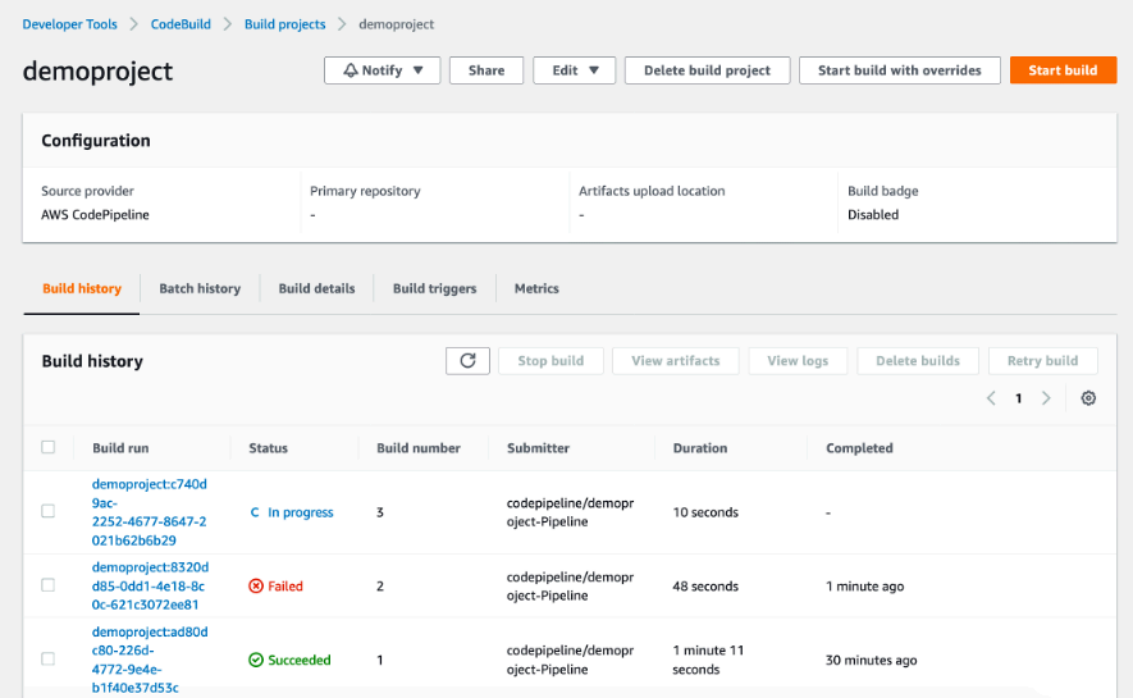
AWS CodeBuild est conçu de façon à ce que votre organisation puisse mettre en place un processus de génération hautement disponible avec une capacité de mise à l'échelle quasi illimitée. AWS

CodeBuild fournit des environnements de démarrage rapide pour un certain nombre de langues courantes, ainsi que la possibilité d'exécuter n'importe quel conteneur Docker que vous spécifiez.

L'outil CodeBuild est très flexible, en raison des avantages d'une intégration étroite à AWS CodeCommit, à AWS CodePipeline et à AWS CodeDeploy, ainsi que des actions Lambda Git et CodePipeline.

Le logiciel peut être créé par l'inclusion d'un fichier `buildspec.yml` qui identifie chacune des phases de génération, y compris les actions avant et après la génération, ou les actions spécifiées au moyen de l'outil CodeBuild.

Vous pouvez consulter l'historique détaillé de chaque génération à l'aide du tableau de bord CodeBuild. Les événements sont stockés sous forme de fichiers journaux Amazon CloudWatch Logs.



Fichiers journaux CloudWatch Logs dans AWS CodeBuild

Intégration du pipeline à Jenkins

Vous pouvez utiliser l'outil de génération Jenkins pour créer des pipelines de livraison. Ces pipelines ont recours à des tâches standard qui définissent les étapes de mise en œuvre des phases de livraison continue. Cependant, cette approche peut ne pas être optimale pour les projets plus importants, car l'état actuel du pipeline ne persiste pas entre les redémarrages de Jenkins, la mise en

œuvre de l'approbation manuelle n'est pas simple et le suivi de l'état d'un pipeline complexe peut être compliqué.

Au lieu de cela, AWS recommande de mettre en œuvre la livraison continue avec Jenkins en utilisant le plugin AWS CodePipeline. Ce plugin permet de décrire des flux complexes à l'aide d'un langage spécifique au domaine de type Groovy et peut être utilisé pour orchestrer des pipelines complexes. Les fonctionnalités du plugin AWS CodePipeline peuvent être améliorées en utilisant des plugins satellites tels que le plugin Pipeline Stage View, qui visualise la progression actuelle des phases définies dans un pipeline, ou le plugin Pipeline Multibranch, qui regroupe les générations provenant de différentes branches.

AWS recommande de stocker la configuration du pipeline dans Jenkinsfile et de l'archiver dans un référentiel de code source. Vous pouvez ainsi suivre les modifications apportées au code du pipeline. Cela est d'autant plus important lorsque vous utilisez le plugin Pipeline Multibranch. AWS recommande également de diviser le pipeline en plusieurs phases. Cela permet de regrouper logiquement les phases du pipeline. En outre, le plugin Pipeline Stage View peut ainsi visualiser l'état actuel du pipeline.

La figure ci-dessous illustre un exemple de pipeline Jenkins, avec quatre phases définies visualisées par le plugin Pipeline Stage View.

Phases définies du pipeline Jenkins visualisées par le plugin Pipeline Stage View

Méthodes de déploiement

Dans le cadre d'un processus de livraison continue, vous pouvez envisager plusieurs stratégies et variantes de déploiement des nouvelles versions de logiciels. Cette section présente les méthodes de déploiement les plus courantes : simultanée (déploiement sur place), propagation, inaltérable et bleu/vert. Parmi ces méthodes, AWS indique lesquelles sont prises en charge par AWS CodeDeploy et AWS Elastic Beanstalk.

Le tableau ci-dessous récapitule les caractéristiques de chaque méthode de déploiement.

Méthode	Impact d'un échec de déploiement	Temps de déploiement	Aucune interruption	Pas de modification DNS	Processus de restauration	Code déployé sur
Déploiement sur place	Temps d'arrêt	🕒	X	✓	Redéploiement	Instances existantes
Propagation	Lot unique hors service. Tous les lots ayant abouti avant l'échec d'exécution de la nouvelle version de l'application.	🕒 🕒 †	✓	✓	Redéploiement	Instances existantes
Propagation avec	Minime si le premier	🕒 🕒	✓	✓	Redéploiement	Instances nouvelles

Méthode	Impact d'un échec de déploiement	Temps de déploiement	Aucune interruption	Pas de modification DNS	Processus de restauration	Code déployé sur
un lot supplémentaire (beanstalk)	lot échoue, sinon similaire à la propagation.	🕒 †				et existantes
Inaltérable	Minimal	🕒 🕒 🕒 🕒	✓	✓	Redéploiement	Nouvelles instances
Répartition du trafic	Minimal	🕒 🕒 🕒 🕒	✓	✓	Réacheminer le trafic et résilier les nouvelles instances	Nouvelles instances
Bleu/vert	Minimal	🕒 🕒 🕒 🕒	✓	X	Revenir à l'ancien environnement	Nouvelles instances

Simultanée (déploiement sur place)

Le déploiement simultané (déploiement sur place) est une méthode que vous pouvez utiliser pour déployer un nouveau code d'application sur une flotte de serveurs existante. Cette méthode remplace tout le code en une seule action de déploiement. Cela nécessite un temps d'arrêt car tous les serveurs de la flotte sont mis à jour en même temps. Il n'est pas nécessaire de mettre à jour les

enregistrements DNS existants. En cas d'échec du déploiement, la seule façon de restaurer les opérations consiste à redéployer le code sur tous les serveurs.

Dans AWS Elastic Beanstalk, on parle de déploiement simultané, qui est disponible pour les applications uniques et à charge équilibrée. Dans AWS CodeDeploy, on parle de déploiement sur place, avec une configuration de déploiement `AllAtOnce`.

Déploiement par propagation

Dans le cas d'un déploiement par propagation, la flotte est divisée en portions afin qu'elle ne soit pas mise à niveau en une seule fois. Au cours du processus de déploiement, deux versions logicielles, la nouvelle et l'ancienne, sont exécutées sur la même flotte. Cette méthode permet une mise à jour sans interruption. En cas d'échec du déploiement, seule la portion mise à jour de la flotte sera affectée.

Une variante de la méthode de déploiement par propagation, appelée version canary, implique tout d'abord le déploiement de la nouvelle version logicielle sur un très faible pourcentage de serveurs. Ainsi, vous pouvez observer le comportement du logiciel en production sur quelques serveurs, tout en minimisant l'impact de modifications importantes. Si, lors d'un déploiement canary, le taux d'erreurs est élevé, la mise à jour du logiciel est annulée. Sinon, le pourcentage de serveurs avec la nouvelle version est progressivement augmenté.

AWS Elastic Beanstalk a appliqué le modèle de déploiement par propagation avec deux options de déploiement, la propagation et la propagation avec un lot supplémentaire. Ces options permettent la mise à l'échelle de l'application avant de mettre les serveurs hors service, ce qui préserve ainsi toutes les capacités pendant le déploiement. AWS CodeDeploy applique ce modèle en tant que variation d'un déploiement sur place avec des modèles tels que OneAtATime et HalfAtATime.

Déploiement inaltérable et bleu/vert

Le modèle inaltérable consiste à déployer le code d'application en démarrant un tout nouvel ensemble de serveurs avec une nouvelle configuration ou une nouvelle version du code d'application. Il tire parti de la capacité du cloud qui permet de créer des ressources de serveur à l'aide de simples appels d'API.

La stratégie de déploiement bleu/vert est un type de déploiement inaltérable qui nécessite également la création d'un autre environnement. Une fois que le nouvel environnement est opérationnel et que tous les tests sont réussis, le trafic est transféré vers ce nouveau déploiement. Il est crucial que

l'ancien environnement, c'est-à-dire l'environnement « bleu », reste inactif au cas où une restauration serait nécessaire.

AWS Elastic Beanstalk prend en charge les modèles de déploiement inaltérable et bleu/vert. AWS CodeDeploy prend également en charge le modèle bleu/vert. Pour en savoir plus sur la façon dont les services AWS réalisent ces modèles inaltérables, consultez le livre blanc Déploiements bleu/vert sur AWS.

Modifications du schéma de la base de données

Les logiciels modernes disposent fréquemment d'une couche de base de données. En règle générale, on utilise une base de données relationnelle, qui stocke les données et la structure des données. Il est souvent nécessaire de modifier la base de données dans le cadre du processus de livraison continue. La gestion des modifications dans une base de données relationnelle nécessite une attention particulière. Elle présente en outre d'autres difficultés que celles rencontrées lors du déploiement de fichiers binaires d'application. En général, lorsque vous mettez à niveau un fichier binaire d'application, vous arrêtez l'application, la mettez à niveau, puis vous la redémarrez. Vous ne vous souciez pas vraiment de l'état de l'application, qui est géré en dehors de celle-ci.

Lors de la mise à niveau des bases de données, vous devez tenir compte de l'état de l'application, car une base de données contient de nombreuses informations d'état, mais relativement peu d'informations sur la logique et la structure.

Le schéma de la base de données avant et après l'application d'une modification doit être considéré comme différentes versions de la base de données. Pour gérer les versions, vous pouvez utiliser des outils tels que Liquibase et Flyway.

En général, ces outils appliquent des variantes des méthodes ci-dessous :

- Ajout d'une table à la base de données dans laquelle une version de base de données est stockée.
- Suivi des commandes de modification de la base de données et regroupement de ces commandes dans des jeux de modifications avec contrôle de version. Dans le cas de Liquibase, ces modifications sont stockées dans des fichiers XML. Flyway applique une méthode légèrement différente, selon laquelle les jeux de modifications sont traités sous forme de fichiers SQL distincts ou occasionnellement comme des classes Java distinctes pour des transitions plus complexes.
- Lorsque Liquibase est invité à mettre à niveau une base de données, il examine la table des métadonnées et détermine les jeux de modifications à exécuter afin de mettre la base de données à jour vers la dernière version.

Résumé des bonnes pratiques

Voici ce qu'il convient de faire ou de ne pas faire en matière d'intégration continue/livraison continue.

À faire :

- Traitez votre infrastructure comme du code.
  - Utilisez le contrôle de version pour votre code d'infrastructure.
  - Utilisez des systèmes de suivi des bogues et de création de tickets.
  - Demandez à vos collègues d'examiner les modifications avant de les appliquer.
  - Établissez des modèles/conceptions de code d'infrastructure.
  - Testez les changements d'infrastructure tels que les changements de code.
- Répartissez les développeurs dans des équipes intégrées ne comptant pas plus de 12 participants autonomes.
- Demandez à tous les développeurs de valider fréquemment le code par rapport au tronc principal, sans branches fonctionnelles de longue durée.
- Adoptez systématiquement un système de génération tel que Maven ou Gradle dans votre organisation et standardisez les générations.
- Demandez aux développeurs de mettre en place des tests unitaires visant à couvrir l'intégralité de la base de code.
- Assurez-vous que les tests unitaires représentent 70 % de l'ensemble des tests en termes de durée, de nombre et de portée.
- Assurez-vous que les tests unitaires sont à jour et ne sont pas négligés. Les échecs des tests unitaires doivent être corrigés et non contournés.
- Considérez votre configuration de livraison continue comme un code.
- Mettez en place des contrôles de sécurité basés sur les rôles (c'est-à-dire qui peut faire quoi et quand).
  - Surveillez toutes les ressources possibles et effectuez-en le suivi.
  - Créez des alertes sur les services, la disponibilité et les temps de réponse.
  - Capturez, apprenez et améliorez.
  - Partagez l'accès avec tous les membres de l'équipe.
  - Planifiez les métriques et la surveillance dans le cycle de vie.
- Conservez les métriques standard et effectuez-en le suivi.

  - Nombre de générations.
  - Nombre de déploiements.
  - Délai moyen pour que les modifications atteignent la production.
  - Délai moyen entre la première phase et chaque phase du pipeline.
  - Nombre de modifications qui arrivent en production.
  - Délai de génération moyen.
- Utilisez des pipelines distincts pour chaque branche et chaque équipe.

À ne pas faire :

- Avoir des branches de longue durée associées à des grandes fusions complexes.
- Procéder à des tests manuels.
- Avoir des processus d'approbation, des portails, des révisions de code et des contrôles de sécurité manuels.

Conclusion

L'intégration et la livraison continues constituent un scénario idéal pour les équipes d'application de votre organisation. Il suffit aux développeurs d'envoyer le code vers un référentiel. Ce code sera intégré, testé, déployé, testé à nouveau, fusionné avec l'infrastructure, fera l'objet de contrôles de sécurité et de qualité, et sera enfin prêt à être déployé en toute confiance.

Grâce à l'utilisation de l'intégration continue/livraison continue, le code est de meilleure qualité et les mises à jour logicielles sont distribuées rapidement, avec une grande certitude qu'il n'y aura pas de changements majeurs. L'impact d'une version publiée peut être corrélé aux données de production et d'exploitation. Il peut également servir à planifier le prochain cycle – une pratique DevOps essentielle pour la transformation du cloud de votre organisation.

Autres lectures

Pour en savoir plus sur les sujets abordés dans ce livre blanc, consultez les livres blancs AWS suivants :

- Présentation des options de déploiement sur AWS
- Déploiements bleu/vert sur AWS
- Configuration d'un pipeline d'intégration/livraison continues en intégrant Jenkins à AWS CodeBuild et à AWS CodeDeploy
- Microservices sur AWS
- Docker sur AWS : exécution de conteneurs dans le cloud

Participants

Les personnes et organisations suivantes ont participé à la préparation du présent document :

- Amrish Thakkar, architecte de solutions principal, AWS
- David Stacy, consultant principal, DevOps, AWS Professional Services
- Asif Khan, architecte de solutions, AWS
- Xiang Shen, architecte de solutions principal, AWS

Révisions du document

Pour être informé des mises à jour de ce livre blanc, abonnez-vous au flux RSS.

update-history-change	update-history-description	update-history-date
Publication initiale	Première publication du livre blanc	27 octobre 2021
Publication initiale	Première publication du livre blanc	1 juin 2017

Mentions légales

Les clients sont responsables de leur propre évaluation indépendante des informations contenues dans ce document. Le présent document : (a) est fourni à titre informatif uniquement, (b) représente les offres et pratiques actuelles de produits AWS, qui sont susceptibles d'être modifiées sans préavis, et (c) ne crée aucun engagement ou assurance de la part d'AWS et de ses affiliés, fournisseurs ou concédants de licences. Les produits ou services AWS sont fournis « en l'état » sans garantie, représentation ou condition, de quelque nature que ce soit, explicite ou implicite. Les responsabilités et obligations d'AWS envers ses clients sont déterminées par les contrats AWS, et le présent document ne fait pas partie d'un contrat entre AWS et ses clients, ni le modifie.