Whitepaper zu AWS

Continuous Integration und Continuous Delivery in AWS

Continuous Integration und Continuous Delivery in AWS: Whitepaper zu AWS

Table of Contents

Continuous Integration und Continuous Delivery in AWS

Veröffentlichungsdatum: 27. Oktober 2021 (Dokumentversionen)

Überblick

In diesem Paper wird erläutert, von welchen Funktionalitäten Sie profitieren, wenn Sie in Ihrer Softwareentwicklungsumgebung auf Continuous Integration (kontinuierliche Integration, CI) und Continuous Delivery (kontinuierliche Bereitstellung, CD) setzen und dafür die Tools von Amazon Web Services (AWS) einsetzen. Continuous Integration und Continuous Delivery sind bewährte Methoden und ein wichtiger Bestandteil einer DevOps-Initiative.

Die Herausforderung der Softwarebereitstellung

Unternehmen stehen heute vor den Herausforderungen einer sich schnell verändernden Wettbewerbslandschaft, steigender Sicherheitsanforderungen und der Skalierbarkeit der Leistung. Sie müssen die Lücke zwischen Produktionsstabilität und schneller Funktionsentwicklung schließen. Continuous Integration und Continuous Delivery (CI/CD) sind Methoden, mit denen Sie schnell Softwareänderungen vornehmen können, während die Stabilität und Sicherheit des Systems aufrechterhalten bleiben.

Amazon hat schon früh erkannt, dass die geschäftlichen Anforderungen an die Bereitstellung von Funktionen für Amazon.com-Einzelhandelskunden, mit Amazon verbundene Unternehmen und Amazon Web Services (AWS) neue und innovative Möglichkeiten zur Bereitstellung von Software erfordern. In der Größenordnung eines Unternehmens wie Amazon müssen Tausende unabhängiger Softwareteams in der Lage sein, parallel zu arbeiten, um Software schnell, sicher, zuverlässig und ohne Ausfalltoleranz bereitzustellen.

Amazon und andere zukunftsorientierte Unternehmen lernten, wie man Software mit hoher Geschwindigkeit bereitstellt, und leisteten mit DevOps Pionierarbeit. DevOps ist eine Kombination aus kulturellen Philosophien, Verfahren und Tools, durch die Organisationen Anwendungen und Services mit hoher Geschwindigkeit bereitstellen können. Mithilfe der DevOps-Prinzipien können Organisationen Produkte schneller weiterentwickeln und verbessern als Organisationen, die herkömmliche Prozesse für die Softwareentwicklung und Infrastrukturverwaltung verwenden. Dank dieses Geschwindigkeitsvorteils können Unternehmen ihre Kunden besser bedienen und sich effektiver auf dem Markt durchsetzen.

Einige dieser Prinzipien, z. B. die Zwei-Pizza-Regel und die Microservices/Service-orientierte Architektur (SOA), sind nicht Gegenstand dieses Whitepapers. In diesem Whitepaper werden die CI/CD-Funktionen erörtert, die Amazon entwickelt und kontinuierlich verbessert hat. CI/CD ist der Schlüssel zur schnellen und zuverlässigen Bereitstellung von Softwarefunktionen.

AWS bietet diese CI/CD-Funktionen jetzt als eine Reihe von Entwicklerservices an: AWS CodeStar, AWS CodeCommit, AWS CodePipeline, AWS CodeBuild, AWS CodeDeploy und AWS CodeArtifact. Entwickler und IT-Operations-Experten, die DevOps anwenden, können diese Dienste nutzen, um Software schnell und sicher bereitzustellen. Zusammen helfen sie Ihnen, den Quellcode Ihrer Anwendung sicher zu speichern und die Versionskontrolle darauf anzuwenden. Sie können AWS CodeStar verwenden, um mithilfe dieser Services schnell einen End-to-End-Softwareversions-Workflow zu orchestrieren. In bestehenden Umgebungen bietet AWS CodePipeline die Flexibilität,

jeden Service unabhängig mit Ihren vorhandenen Tools zu integrieren. Auf diese hochverfügbaren, einfach zu integrierenden Services kann wie auf jeden anderen AWS-Service über die AWS-Managementkonsole, die AWS-Programmierschnittstellen (APIs) und AWS-Softwareentwicklungs-Toolkits (SDKs) zugegriffen werden.

Was sind Continuous Integration und Continuous Delivery/Deployment?

In diesem Abschnitt werden die Praktiken der Continuous Integration und Continuous Delivery erörtert und es wird der Unterschied zwischen Continuous Delivery und Continuous Deployment erläutert.

Continuous Integration

Continuous Integration (CI) ist eine Entwicklungsmethode, bei der Entwickler ihre Codeänderungen regelmäßig in einem zentralen Repository zusammenführen. Anschließend werden automatisierte Builds und Tests ausgeführt. CI bezieht sich meist auf die Entwicklungs- oder Integrationsphase des Softwareveröffentlichungsprozesses. Sie erfordert sowohl eine Automatisierungskomponente (z. B. einen CI- oder Build-Service) als auch eine kulturelle Komponente (z. B. das Erlernen der häufigen Integration). Die Hauptziele der CI bestehen darin, Bugs schneller zu entdecken und zu beheben, die Softwarequalität zu optimieren und den Zeitraum zu minimieren, in dem neue Softwareaktualisierungen validiert und eingeführt werden.

Die Continuous Integration konzentriert sich auf kleinere Commits und kleinere zu integrierende Codeänderungen. Ein Entwickler führt in regelmäßigen Abständen, mindestens einmal täglich, einen Commit für den Code aus. Der Entwickler zieht Code aus dem Code-Repository, um sicherzustellen, dass der Code auf dem lokalen Host zusammengeführt wird, bevor auf den Buildserver übertragen wird. Zu diesem Zeitpunkt führt der Buildserver die verschiedenen Tests aus und akzeptiert das Code-Commit oder lehnt es ab.

Zu den grundlegenden Herausforderungen bei der Implementierung von CI gehören häufigere Commits in die gemeinsame Codebasis, die Pflege eines einzelnen Quellcode-Repositorys, die Automatisierung von Builds und die Automatisierung von Tests. Zu den weiteren Herausforderungen gehören Tests in produktionsähnlichen Umgebungen durchzuführen, dem Team Einblicke in den Prozess zu bieten und es Entwicklern zu ermöglichen, problemlos jede Version der Anwendung zu erhalten.

Continuous Delivery und Deployment

Continuous Delivery (CD) ist eine Softwareentwicklungsmethode, bei der Codeänderungen automatisch erstellt, getestet und für eine Produktionsversion vorbereitet werden. Sie erweitert die

Continuous Integration, indem nach Abschluss der Entwicklungsphase alle Codeänderungen in einer Testumgebung und/oder in einer Produktionsumgebung bereitgestellt werden. Continuous Delivery kann mit einem Workflow-Prozess vollständig oder, mit manuellen Schritten an kritischen Stellen, teilweise automatisiert werden. Bei einer korrekten Implementierung der Continuous Delivery steht Entwicklern stets ein Build-Artefakt für die Bereitstellung zur Verfügung, das bereits einen standardisierten Testprozess durchlaufen hat.

Beim Continuous Deployment werden Änderungen automatisch in einer Produktionsumgebung bereitgestellt. Diese vollständige Automatisierung des Software-Einführungsprozesses bedeutet, dass keine explizite Genehmigung durch einen Entwickler vorliegen muss. Dies wiederum ermöglicht eine kontinuierliche Kunden-Feedback-Schleife zu Beginn des Produktlebenszyklus.

Continuous Delivery ist nicht dasselbe wie Continuous Deployment

Fälschlicherweise wird oft angenommen, dass bei der Continuous Delivery jede Änderung, für die ein Commit ausgeführt wurde, sofort nach dem Bestehen automatisierter Tests auf die Produktion angewendet wird. Bei der Continuous Delivery geht es jedoch nicht darum, jede Änderung sofort auf die Produktion anzuwenden, sondern sicherzustellen, dass jede Änderung auf die Produktion angewendet werden kann.

Bevor Sie eine Änderung für die Produktion bereitstellen, können Sie einen Entscheidungsprozess implementieren, um sicherzustellen, dass die Produktionsbereitstellung autorisiert und geprüft ist. Diese Entscheidung kann von einer Person getroffen und dann von Tools ausgeführt werden.

Dank Continuous Delivery wird aus der Entscheidung, live zu gehen, eine geschäftliche Entscheidung, keine technische Entscheidung. Die technische Validierung erfolgt, wann immer ein Commit ausgeführt wird.

Die Einführung einer Änderung in die Produktion ist kein störendes Ereignis. Für die Bereitstellung muss das technische Team die Arbeit an den nächsten Änderungen nicht unterbrechen. Es werden weder ein Projektplan noch eine Übergabedokumentation oder ein Wartungsfenster benötigt. Die Bereitstellung wird zu einem wiederholbaren Prozess, der in Testumgebungen mehrfach durchgeführt wurde und sich bewährt hat.

Nutzen von Continuous Delivery

CD bietet Ihrem Softwareentwicklungsteam zahlreiche Vorteile, darunter die Automatisierung des Prozesses, die Verbesserung der Entwicklerproduktivität, die Verbesserung der Codequalität und die schnellere Bereitstellung von Updates für Ihre Kunden.

Den Software-Einführungsprozess automatisieren

CD bietet Ihrem Team eine Methode zum Einchecken von Code, der automatisch entwickelt, getestet und für die Freigabe in die Produktion vorbereitet wird, sodass Ihre Softwarebereitstellung effizient, robust, schnell und sicher ist.

Produktivität der Entwickler steigern

CD-Verfahren verbessern die Produktivität Ihres Teams, indem sie Entwickler von manuellen Aufgaben befreien, komplexe Abhängigkeiten entwirren und die Konzentration auf die Bereitstellung neuer Softwarefunktionen ermöglichen. Anstatt ihren Code in andere Teile des Geschäfts zu integrieren und sich um die Bereitstellung dieses Codes auf einer Plattform zu kümmern, können sich Entwickler auf die Codierungslogik konzentrieren, die die von Ihnen benötigten Funktionen bietet.

Die Codequalität verbessern

CD kann Ihnen helfen, Fehler früh im Bereitstellungsprozess zu entdecken und zu beheben, bevor sie später zu größeren Problemen werden. Ihr Team kann problemlos zusätzliche Arten von Codetests durchführen, da der gesamte Prozess automatisiert wurde. Mit der Routine, mehr Tests durchzuführen, und das häufiger, können Teams schneller iterieren und erhalten sofortiges Feedback zu den Auswirkungen der Änderungen. Auf diese Weise können Teams Code hoher Qualität mit einer hohen Stabilitäts- und Sicherheitsgarantie schreiben. Entwickler erfahren dank sofortigem Feedback, ob der neue Code funktioniert und ob störende Änderungen oder Bugs eingeführt wurden. Fehler, die zu Beginn des Entwicklungsprozesses erkannt werden, lassen sich am einfachsten beheben.

Aktualisierungen schneller bereitstellen

CD hilft Ihrem Team, Kunden schnell und häufig Updates bereitzustellen. Wenn CI/CD implementiert ist, kann das gesamte Team schneller arbeiten, was auch für die Veröffentlichung von

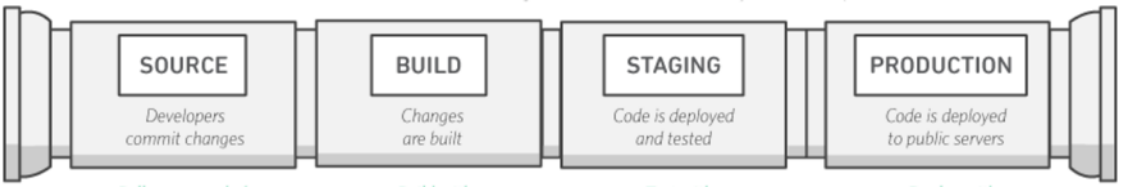
Funktionen und Fehlerkorrekturen gilt. Unternehmen können schneller auf Marktveränderungen, Sicherheitsherausforderungen, Kundenanforderungen und Kostendruck reagieren. Wenn beispielsweise eine neue Sicherheitsfunktion erforderlich ist, kann Ihr Team CI/CD mit automatisierten Tests implementieren, um den Fix schnell und zuverlässig mit hoher Sicherheit in Produktionssysteme einzuführen. Was früher Wochen und Monate gedauert hat, kann jetzt in Tagen oder sogar Stunden erledigt werden.

Continuous Integration und Continuous Delivery implementieren

In diesem Abschnitt wird erläutert, wie Sie mit der Implementierung eines CI/CD-Modells in Ihrer Organisation beginnen können. Thema dieses Whitepapers ist es dagegen nicht, wie eine Organisation mit einem ausgereiften DevOps- und Cloud-Transformationsmodell eine CI/CD-Pipeline entwickelt und verwendet. Um Sie auf Ihrem DevOps-Weg zu unterstützen, verfügt AWS über eine Reihe zertifizierter DevOps-Partner, die Ressourcen und Tools bereitstellen können. Weitere Informationen zur Vorbereitung einer Umstellung auf die AWS Cloud finden Sie im Whitepaper Aufbau eines Cloud-Betriebsmodells.

Der Weg zu Continuous Integration/Continuous Delivery (CI/CD)

Sie können sich CI/CD als Pipeline vorstellen (siehe folgende Abbildung): Neuer Code wird an einem Ende eingereicht, in verschiedenen Phasen (Quelle, Entwicklung, Staging und Produktion) getestet und dann als produktionsbereiter Code veröffentlicht. Wenn Ihre Organisation noch nicht mit CI/CD vertraut ist, kann sie sich dieser Pipeline iterativ nähern. Das bedeutet, dass Sie klein anfangen und in jeder Phase iterieren sollten, damit Sie Ihren Code so verstehen und entwickeln können, dass er das Wachstum Ihrer Organisation unterstützt.



CI/CD-Pipeline

Jede Phase der CI/CD-Pipeline stellt im Bereitstellungsprozess eine logische Einheit dar. Darüber hinaus fungiert jede Stufe als Tor, das einen bestimmten Aspekt des Codes untersucht. Wenn der Code die Pipeline durchläuft, sollte seine Qualität in den späteren Phasen höher sein, da ja immer mehr Aspekte überprüft werden. Werden in einer frühen Phase Probleme erkannt, durchläuft der Code die Pipeline nicht weiter. Die Ergebnisse der Tests werden sofort an das Team gesendet und alle weiteren Builds und Releases werden gestoppt, wenn die Software eine Phase nicht erfolgreich besteht.

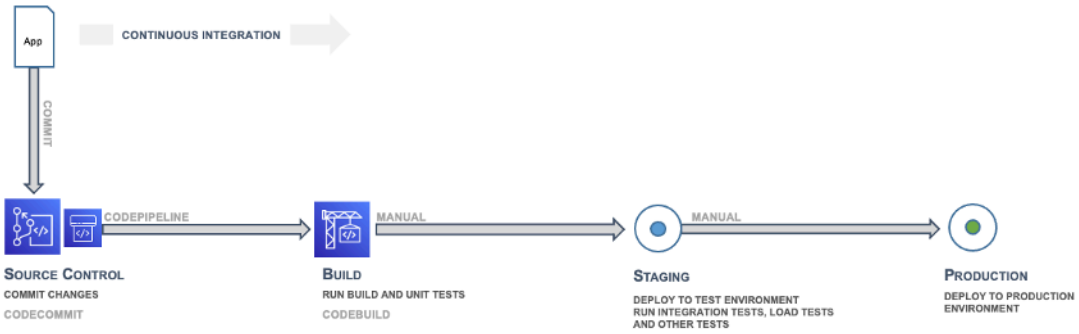
Diese Phasen sind lediglich Vorschläge. Sie können die Phasen natürlich an Ihre Geschäftsanforderungen anpassen. Einige Phasen können für mehrere Arten von Tests, Sicherheit

und Leistung wiederholt werden. Abhängig von der Komplexität Ihres Projekts und der Struktur Ihrer Teams können einige Phasen auf verschiedenen Ebenen mehrmals wiederholt werden. So kann das Endprodukt eines Teams beispielsweise zu einer Abhängigkeit im Projekt des nächsten Teams werden. Dies bedeutet, dass das Endprodukt des ersten Teams anschließend als Artefakt im Projekt des nächsten Teams verwendet wird.

Das Vorhandensein einer CI/CD-Pipeline hat große Auswirkungen auf die Reifung der Fähigkeiten Ihrer Organisation. Die Organisation sollte mit kleinen Schritten beginnen und nicht gleich versuchen, eine vollständig ausgereifte Pipeline mit mehreren Umgebungen, vielen Testphasen und Automatisierung in allen Phasen aufzubauen. Denken Sie daran, dass selbst Organisationen mit hochausgereiften CI/CD-Umgebungen ihre Pipelines kontinuierlich verbessern müssen.

Der Aufbau einer CI/CD-fähigen Organisation ist eine Reise mit vielen Zielen entlang des Weges. Im nächsten Abschnitt wird ein möglicher Weg beschrieben, den Ihre Organisation einschlagen könnte, angefangen bei der Continuous Integration bis hin zu den verschiedenen Stufen der Continuous Delivery.

Continuous Integration



Continuous Integration – Quelle und Entwicklung

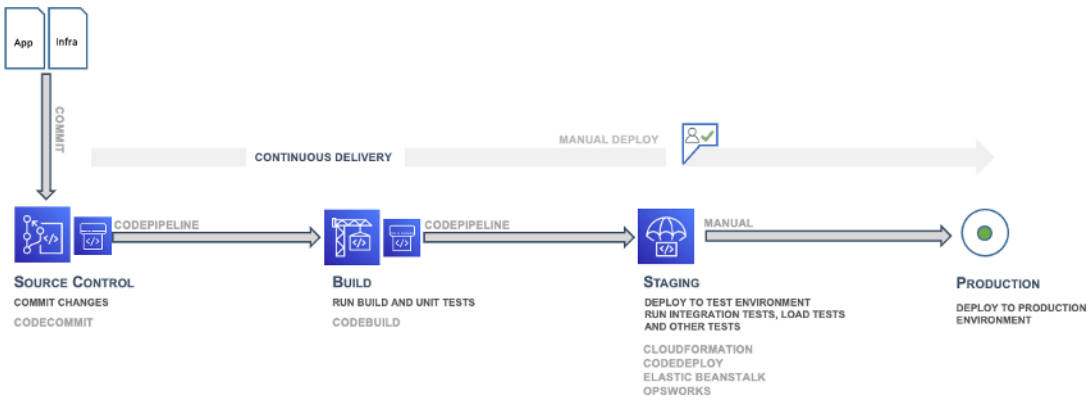
Die erste Phase der CI/CD-Reise besteht darin, hinsichtlich der Continuous Integration zu reifen. Sie sollten sicherstellen, dass alle Entwickler ihren Code regelmäßig in ein zentrales Repository (das z. B. in CodeCommit oder GitHub gehostet wird) übergeben und alle Änderungen in einem Release-Zweig für die Anwendung zusammenführen. Kein Entwickler sollte über isolierten Code verfügen. Wenn ein Funktionszweig für einen bestimmten Zeitraum benötigt wird, sollte er durch möglichst häufige Zusammenführungen aus dem Upstream auf dem neuesten Stand gehalten werden. Häufige Commits und Zusammenführungen mit abgeschlossenen Arbeitsschritten werden dem Team empfohlen, um eine Routine zu entwickeln, und werden durch den Prozess gefördert. Ein Entwickler,

der Code früh und häufig zusammenführt, muss später wahrscheinlich weniger Integrationsprobleme beheben.

Sie sollten Entwickler außerdem dazu anhalten, so früh wie möglich Einheitentests für ihre Anwendungen zu erstellen und diese Tests durchzuführen, bevor sie den Code in das zentrale Repository übertragen. Fehler, die in den Anfangsphasen der Softwareentwicklung erkannt werden, sind mit weniger Zeit- und Kostenaufwand zu beheben.

Wenn der Code in einen Zweig in einem Quellcode-Repository übertragen wird, sendet eine Workflow-Engine, die diesen Zweig überwacht, einen Befehl an ein Entwickler-Tool, um den Code zu erstellen und die Einheitentests in einer kontrollierten Umgebung durchzuführen. Der Entwicklungsprozess sollte entsprechend dimensioniert sein, um alle Aktivitäten zu verarbeiten, einschließlich Übertragungen und Tests, die während der Commit-Phase stattfinden können, um ein schnelles Feedback zu erhalten. In dieser Phase können auch andere Qualitätsprüfungen wie die Abdeckung von Einheitentests, Stilprüfung und statische Analyse durchgeführt werden. Schließlich erstellt das Entwickler-Tool einen oder mehrere binäre Builds und andere Artefakte wie Bilder, Stylesheets und Dokumente für die Anwendung.

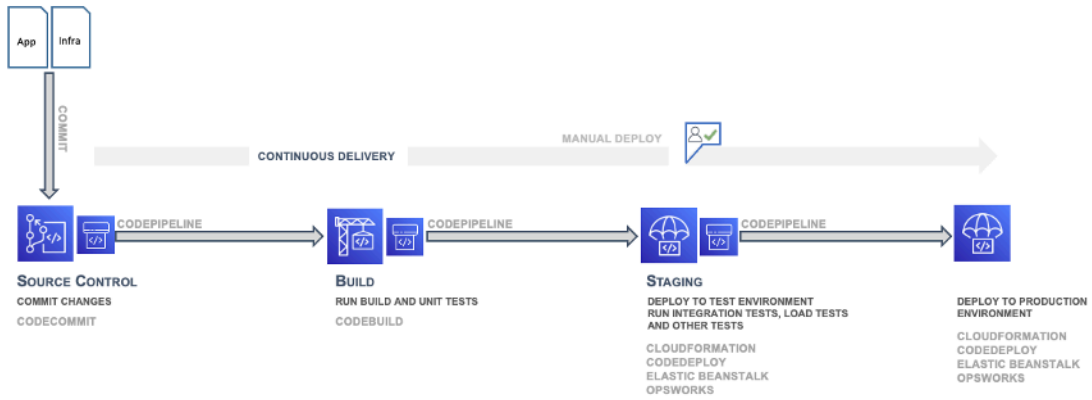
Continuous Delivery: Eine Staging-Umgebung erstellen



Continuous Delivery – Staging

Continuous Delivery (CD) ist die nächste Phase und umfasst die Bereitstellung des Anwendungscodes in einer Staging-Umgebung, bei der es sich um ein Replikat des Produktionsstacks handelt, und das Ausführen weiterer Funktionstests. Die Staging-Umgebung kann eine statische Umgebung sein, die zum Testen vorgefertigt wurde. Sie können aber auch eine dynamische Umgebung mit festgeschriebener Infrastruktur und Konfigurationscode zum Testen und Bereitstellen des Anwendungscodes bereitstellen und konfigurieren.

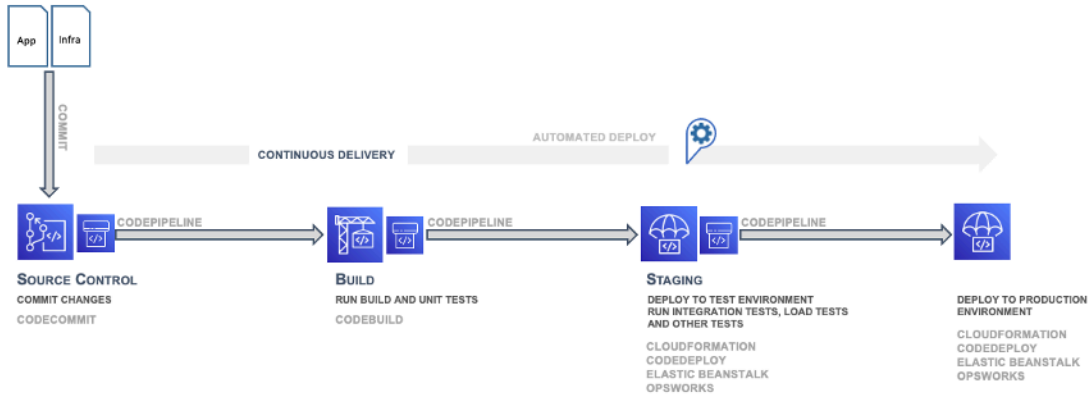
Continuous Delivery: Eine Produktionsumgebung erstellen



Continuous Delivery – Produktion

In der Bereitstellungspipeline folgt nach der Staging-Umgebung die Produktionsumgebung, die ebenfalls mithilfe von Infrastructure as Code (IaC) entwickelt wird.

Continuous Deployment



Continuous Deployment

Die letzte Phase der CI/CD-Bereitstellungspipeline ist das Continuous Deployment, was die vollständige Automatisierung des gesamten Software-Release-Prozesses einschließlich der Bereitstellung in der Produktionsumgebung umfassen kann. In einer vollständig ausgereiften CI/CD-Umgebung ist der Pfad zur Produktionsumgebung vollständig automatisiert, sodass Code mit hoher Sicherheit bereitgestellt werden kann.

Reife und Weiterentwicklung

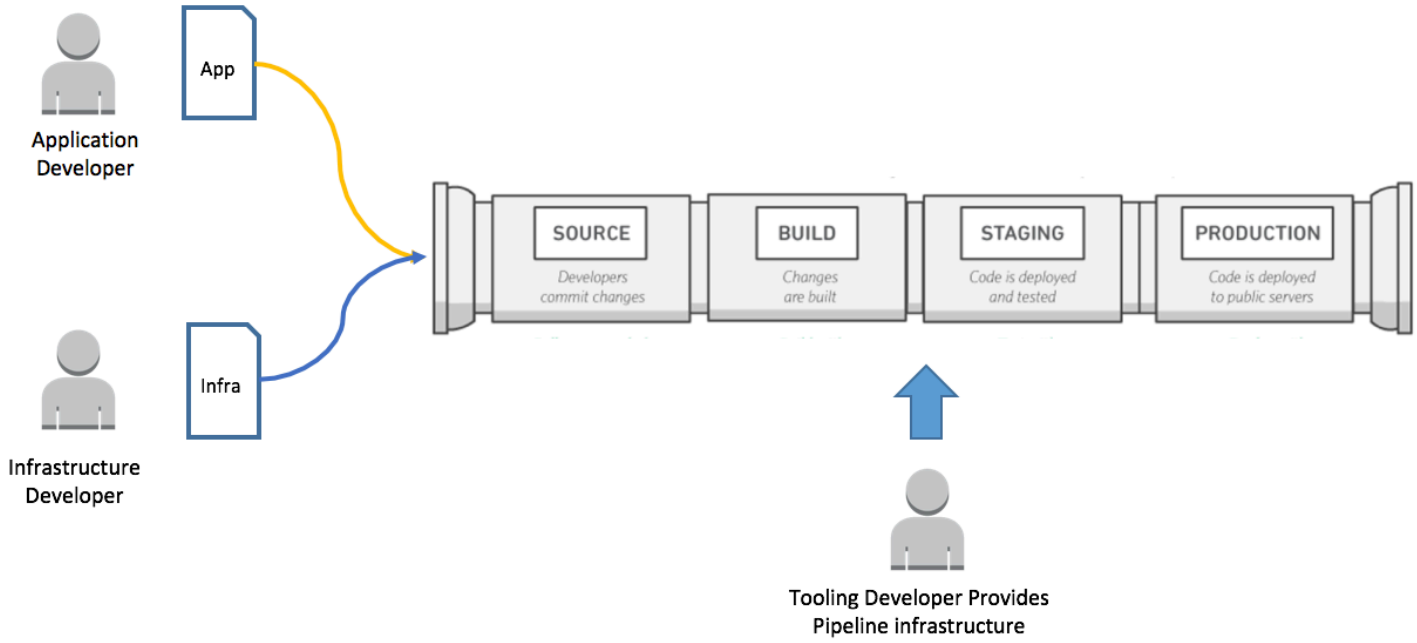
Wenn Ihre Organisation reift, entwickelt sie das CI/CD-Modell weiter, um weitere der folgenden Verbesserungen einzubeziehen:

- Zusätzliche Staging-Umgebungen für spezifische Leistungs-, Compliance-, Sicherheits- und Benutzeroberflächen (UI)-Tests
- Einheitentests der Infrastruktur und des Konfigurationscodes zusammen mit dem Anwendungscode
- Integration mit anderen Systemen und Prozessen wie Codeüberprüfung, Problemverfolgung und Ereignisbenachrichtigung
- Integration mit Datenbankschemamigration (falls zutreffend)
- Zusätzliche Schritte für die Prüfung und Geschäftsgenehmigung

Selbst die ausgereiftesten Organisationen, die komplexe CI/CD-Pipelines mit mehreren Umgebungen verwenden, versuchen ihre Prozesse kontinuierlich zu verbessern. DevOps ist kein Ziel, sondern eine Reise. Es wird kontinuierlich Feedback zur Pipeline eingeholt. Verbesserungen von Geschwindigkeit, Skalierung, Sicherheit und Zuverlässigkeit werden durch die Zusammenarbeit der verschiedenen Entwicklungsteams erzielt.

Teams

AWS empfiehlt, drei Entwicklerteams für die Implementierung einer CI/CD-Umgebung zusammenzustellen: ein Anwendungsteam, ein Infrastrukturteam und ein Toolsteam (siehe folgende Abbildung). Diese Struktur basiert auf einer Reihe von bewährten Methoden, die in schnell wachsenden Startups, großen Unternehmen und bei Amazon selbst entwickelt und angewendet wurden. Für die Teamgröße hat sich die Zwei-Pizza-Regel bewährt: Teams sollten nicht mehr Mitglieder haben, als von zwei großen Pizzen satt werden, also etwa 10–12 Personen. Der Grund dafür ist, dass sinnvolle Gespräche an Grenzen stoßen, wenn die Gruppengröße zunimmt und sich mehr und mehr Nebengespräche entwickeln.



Anwendungs-, Infrastruktur- und Toolsteams

Anwendungsteam

Das Anwendungsteam erstellt die Anwendung. Anwendungsentwickler sind für Backlog, Storys und Einheitentests zuständig und entwickeln Funktionen basierend auf einem bestimmten Anwendungsziel. Das organisatorische Ziel dieses Teams ist es, dass diese Entwickler möglichst wenig Zeit für nicht zum Kern gehörende Anwendungsaufgaben aufwenden müssen.

Das Anwendungsteam sollte nicht nur über funktionale Programmierkenntnisse in der Anwendungssprache, sondern auch über Plattformkenntnisse und ein Verständnis der Systemkonfiguration verfügen. Dadurch können sie sich ausschließlich auf die Entwicklung von Funktionen und das Verstärken der Anwendung konzentrieren.

Infrastrukturteam

Das Infrastrukturteam schreibt den Code, der die für die Ausführung der Anwendung erforderliche Infrastruktur erstellt und konfiguriert. Dieses Team verwendet möglicherweise native AWS-Tools wie AWS CloudFormation oder generische Tools wie Chef, Puppet oder Ansible. Das Infrastrukturteam legt fest, welche Ressourcen benötigt werden, und arbeitet eng mit dem Anwendungsteam zusammen. Bei kleineren Anwendungen kann es ausreichen, wenn das Infrastrukturteam aus nur einer oder zwei Personen besteht.

Das Team sollte über Kenntnisse in Methoden zur Bereitstellung von Infrastrukturen wie AWS CloudFormation oder HashiCorp Terraform verfügen. Das Team sollte auch Kenntnisse in der Konfigurationsautomatisierung mit Tools wie Chef, Ansible, Puppet oder Salt entwickeln.

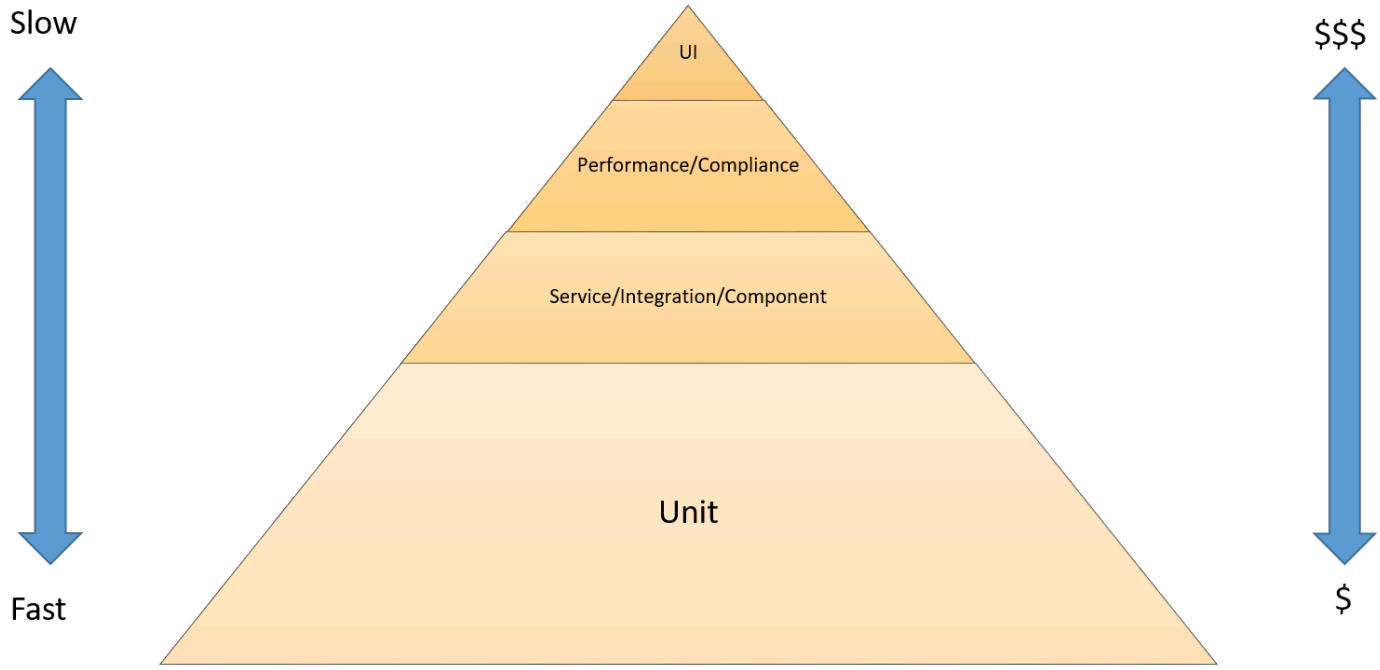
Toolsteam

Das Toolsteam entwickelt und verwaltet die CI/CD-Pipeline. Seine Mitglieder sind für die Infrastruktur und die Tools verantwortlich, aus denen die Pipeline besteht. Sie sind nicht Teil des Zwei-Pizza-Teams. Sie erstellen jedoch ein Tool, das von den Anwendungs- und Infrastrukturteams in der Organisation verwendet wird. Beim Toolsteam ist die kontinuierliche Reifung besonders wichtig, damit es den reifenden Anwendungs- und Infrastrukturteams immer einen Schritt voraus ist.

Das Toolsteam muss in der Lage sein, alle Teile der CI/CD-Pipeline zu entwickeln und zu integrieren. Dazu gehören das Erstellen von Repositorys für die Quellcodeverwaltung, Workflow-Engines, Build-Umgebungen, Testframeworks und Artefakt-Repositorys. Dieses Team kann zum Beispiel Software wie AWS CodeStar, AWS CodePipeline, AWS CodeCommit, AWS CodeDeploy, AWS CodeBuild und AWS CodeArtifact zusammen mit Jenkins, GitHub, Artifactory, TeamCity und ähnliche Tools implementieren. Einige Organisationen nennen dies möglicherweise ein DevOps-Team, AWS rät jedoch davon ab und schlägt stattdessen vor, DevOps als die Summe der Personen, Prozesse und Tools bei der Softwarebereitstellung zu betrachten.

Testphasen bei Continuous Integration und Continuous Delivery

Die drei CI/CD-Teams sollten in den verschiedenen Phasen der CI/CD-Pipeline Tests in den Lebenszyklus der Softwareentwicklung integrieren. Allgemein sollte so früh wie möglich getestet werden. Die folgende Testpyramide ist ein Konzept von Mike Cohn, das er in seinem Buch Succeeding with Agile vorstellt. Sie zeigt die verschiedenen Softwaretests in Bezug auf ihre Kosten und Geschwindigkeit, mit der sie ausgeführt werden.



CI/CD-Testpyramide

Einheitentests bilden die Basis der Pyramide. Sie sind am schnellsten auszuführen und am kostengünstigsten. Daher sollten Einheitentests den Großteil Ihrer Teststrategie ausmachen. Als Faustregel sind etwa 70 Prozent ein guter Wert. Einheitentests sollten den gesamten Code nahezu vollständig abdecken, da in dieser Phase erkannte Fehler schnell und kostengünstig behoben werden können.

Service-, Komponenten- und Integrationstests stehen in der Pyramide über den Einheitentests. Diese Tests erfordern detaillierte Umgebungen und sind daher in Bezug auf die Infrastrukturanforderungen teurer und werden langsamer ausgeführt. Leistungs- und Compliancetests sind die nächste Stufe. Sie erfordern Umgebungen in Produktionsqualität und sind noch teurer. UI- und Benutzerakzeptanztests stehen an der Spitze der Pyramide und erfordern ebenfalls Umgebungen in Produktionsqualität.

Alle diese Tests sind Teil einer vollständigen Strategie, um die hohe Qualität der Software sicherzustellen. Aus Gründen der Entwicklungsgeschwindigkeit liegt der Schwerpunkt jedoch auf der Anzahl der Tests und der Abdeckung in der unteren Hälfte der Pyramide.

In den folgenden Abschnitten werden die CI/CD-Phasen erörtert.

Die Quelle einrichten

Zu Beginn des Projekts ist es wichtig, eine Quelle einzurichten, in der Sie Ihren Rohcode sowie Konfigurations- und Schemaänderungen speichern können. Wählen Sie in der Quellphase ein Quellcode-Repository aus, z. B. eines, das in GitHub oder AWS CodeCommit gehostet wird.

Builds einrichten und ausführen

Die Build-Automatisierung ist für den CI-Prozess unerlässlich. Beim Einrichten der Build-Automatisierung besteht die erste Aufgabe darin, das richtige Entwicklungstool auszuwählen. Es gibt viele Entwicklungstools, zum Beispiel:

- Ant, Maven und Gradle für Java
- Make für C/C++
- Grunt für JavaScript
- Rake für Ruby

Welches Entwicklungstool für Sie am besten geeignet ist, hängt von der Programmiersprache Ihres Projekts und den Kenntnissen Ihres Teams ab. Nachdem Sie das Entwicklungstool ausgewählt haben, müssen alle Abhängigkeiten zusammen mit den Entwicklungsschritten in den Build-Skripten klar definiert werden. Es hat sich auch bewährt, die endgültigen Build-Artefakte mit Versionsnummern zu versehen, da dies die Bereitstellung und die Nachverfolgung von Problemen erleichtert.

Erstellung

In der Entwicklungsphase nehmen die Entwicklungstool alle Änderungen am Quellcode-Repository als Eingabe an, entwickeln die Software und führen die folgenden Arten von Tests aus:

Einheitentest – Testet einen bestimmten Codeabschnitt, um sicherzustellen, dass der Code sich wie erwartet verhält. Einheitentests werden während der Entwicklungsphase von Softwareentwicklern durchgeführt. In dieser Phase können eine statische Codeanalyse, Datenflussanalyse, Codeabdeckung und andere Softwareüberprüfungsprozesse angewendet werden.

Statische Codeanalyse – Dieser Test wird durchgeführt, ohne die Anwendung nach den Build- und Einheitentests tatsächlich auszuführen. Diese Analyse ist hilfreich, um Codierungsfehler und Sicherheitslücken zu finden und die Einhaltung der Codierungsrichtlinien sicherzustellen.

Staging

In der Staging-Phase werden vollständige Umgebungen geschaffen, die die spätere Produktionsumgebung widerspiegeln. Die folgenden Tests werden durchgeführt:

Integrationstest – Überprüft die Schnittstellen zwischen Komponenten anhand des Softwareentwurfs. Integrationstests sind ein iterativer Prozess und erleichtern die Entwicklung von robusten Schnittstellen und Systemintegrität.

Komponententest – Testet die Nachrichtenweitergabe zwischen verschiedenen Komponenten und deren Ergebnisse. Ein Hauptziel dieses Tests könnte die Idempotenz bei Komponententests sein. Tests können extrem große Datenmengen oder ungewöhnliche Situationen und abnormale Eingaben umfassen.

Systemtest – Testet das System vollständig und überprüft, ob die Software die Geschäftsanforderungen erfüllt. Dies kann das Testen der Benutzeroberfläche (UI), der API, der Backend-Logik und des Endzustands umfassen.

Leistungstest – Ermittelt die Reaktionsfähigkeit und Stabilität eines Systems, wenn es unter einer bestimmten Workload arbeitet. Leistungstests werden auch verwendet, um andere Qualitätsmerkmale des Systems wie Skalierbarkeit, Zuverlässigkeit und Ressourcenauslastung zu untersuchen, zu messen, zu validieren oder zu verifizieren. Leistungstests können zum Beispiel Belastungstests, Stresstests und Spike-Tests sein. Leistungstests werden für das Benchmarking anhand vordefinierter Kriterien verwendet.

Compliance-Test – Überprüft, ob die Codeänderung den Anforderungen einer nicht funktionalen Spezifikation und/oder bestimmten Vorschriften entspricht. Damit wird ermittelt, ob Sie die gültigen Standards implementieren und erfüllen.

Benutzerakzeptanztest – Validiert den End-to-End-Geschäftsablauf. Dieser Test wird von einem Endbenutzer in einer Staging-Umgebung durchgeführt und prüft, ob das System die Anforderungen der Anforderungsspezifikation erfüllt. In der Regel wenden Kunden in dieser Phase Alpha- und Beta-Testmethoden an.

Produktion

Schließlich, nachdem die vorherigen Tests bestanden wurden, wird die Staging-Phase in einer Produktionsumgebung wiederholt. In dieser Phase kann ein abschließender Canary-Test durchgeführt werden, indem der neue Code nur auf einer kleinen Teilmenge von Servern oder sogar

nur auf einem Server oder in einer AWS-Region bereitgestellt wird, bevor Code in der gesamten Produktionsumgebung bereitgestellt wird. Einzelheiten zur sicheren Bereitstellung in der Produktion werden im Abschnitt Bereitstellungsmethoden behandelt.

Im nächsten Abschnitt wird die Entwicklung der Pipeline erörtert, um diese Phasen und Tests zu integrieren.

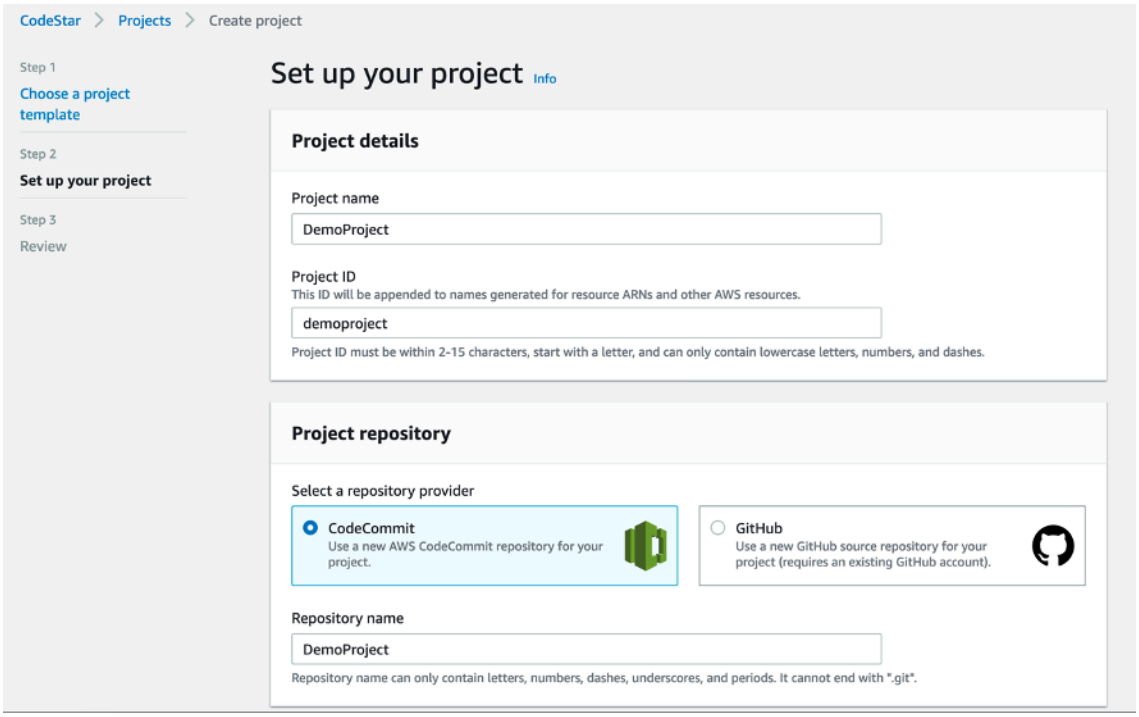
Pipeline entwickeln

In diesem Abschnitt wird die Entwicklung der Pipeline erörtert. Erstellen Sie zunächst eine Pipeline, die nur die für die CI benötigten Komponenten enthält, und wechseln Sie später zu einer Pipeline für die Continuous Delivery mit mehr Komponenten und Phasen. In diesem Abschnitt wird auch erläutert, wie Sie AWS Lambda-Funktionen und manuelle Genehmigungen für große Projekte verwenden sowie für mehrere Teams, Zweige und AWS-Regionen planen können.

Start mit einer minimal funktionsfähigen Pipeline für die Continuous Integration

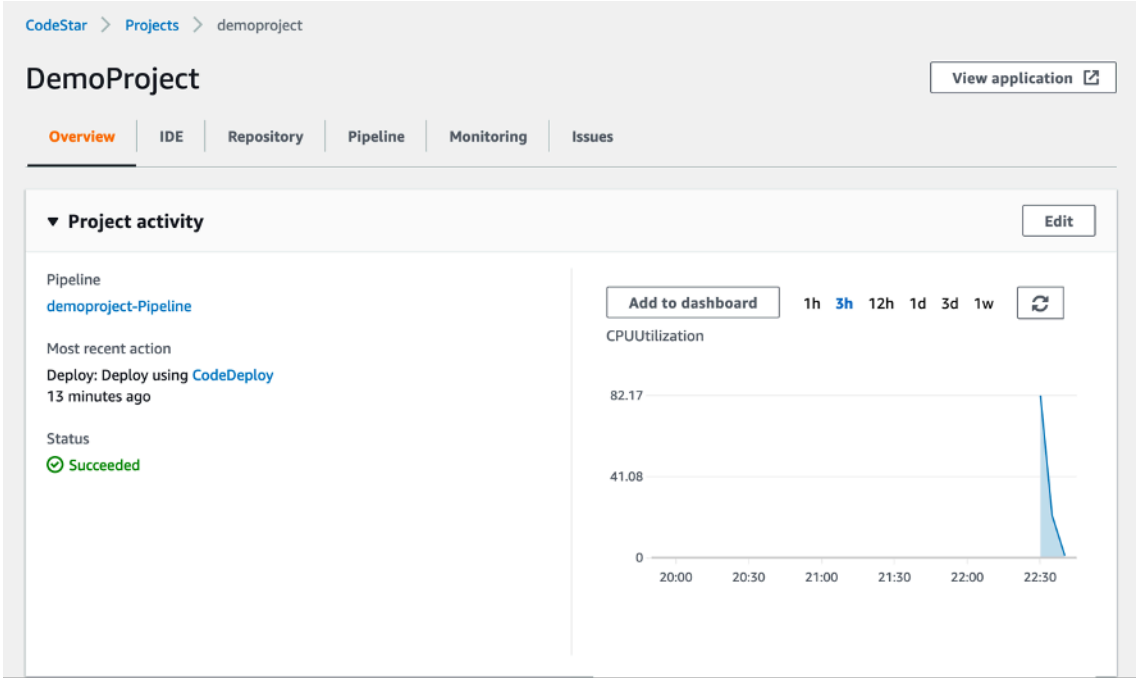
Der Weg Ihrer Organisation zur Continuous Delivery beginnt mit einer Pipeline, die minimal funktionsfähig ist (Minimal Viable Pipeline, (MVP). Wie unter Continuous Integration und Continuous Delivery implementieren erläutert, können Teams mit einem sehr einfachen Prozess beginnen, z. B. mit der Implementierung einer Pipeline, die eine Überprüfung des Codestils oder einen einzelnen Einheitentest ohne Bereitstellung durchführt.

Eine wichtige Komponente dabei ist ein Orchestrierungstool für die Continuous Delivery. Zur Unterstützung beim Aufbau dieser Pipeline hat Amazon AWS CodeStar entwickelt.



AWS CodeStar-Einrichtungsseite

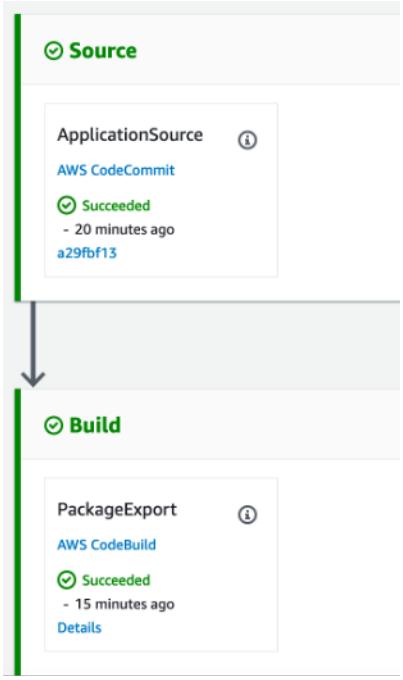
AWS CodeStar verwendet AWS CodePipeline, AWS CodeBuild, AWS CodeCommit und AWS CodeDeploy mit einem integrierten Einrichtungsprozess, Tools, Vorlagen und einem Dashboard. AWS CodeStar enthält die Tools, die Sie für eine schnelle Entwicklung, Erstellung und Bereitstellung von Anwendungen in AWS benötigen. Auf diese Weise können Sie den Code schneller freigeben. Kunden, die bereits mit der AWS-Managementkonsole vertraut sind und ein höheres Maß an Kontrolle wünschen, können Entwicklertools ihrer Wahl manuell konfigurieren und individuelle AWS-Services nach Bedarf bereitstellen.



AWS CodeStar-Dashboard

AWS CodePipeline ist ein CI/CD-Service, der über AWS CodeStar oder über die AWS-Managementkonsole für schnelle und zuverlässige Anwendungs- und Infrastrukturaktualisierungen verwendet werden kann. Basierend auf den von Ihnen definierten Release-Prozessmodellen erstellt und testet AWS CodePipeline Ihren Code bei jeder Codeänderung und stellt ihn bereit. So können Sie Funktionen und Updates schnell und verlässlich liefern. Sie können auf einfache Weise eine End-to-End-Lösung erstellen. Nutzen Sie unsere fertigen Plug-Ins für häufig genutzte Drittanbieterservices wie GitHub oder integrieren Sie auf jeder Stufe Ihrer entsprechenden Prozesse Ihre eigenen, angepassten Plug-Ins. Mit AWS CodePipeline zahlen Sie nur für das, was Sie tatsächlich nutzen. Es fallen keine Vorausleistungen an und Sie gehen keine langfristigen Verpflichtungen ein.

Die Schritte von AWS CodeStar und AWS CodePipeline entsprechen direkt den Quell-, Entwicklungs-, Staging- und Produktionsphasen der CI/CD. Während eine Continuous Delivery wünschenswert ist, können Sie mit einer einfachen zweistufigen Pipeline beginnen, die das Quell-Repository überprüft und eine Build-Aktion ausführt:

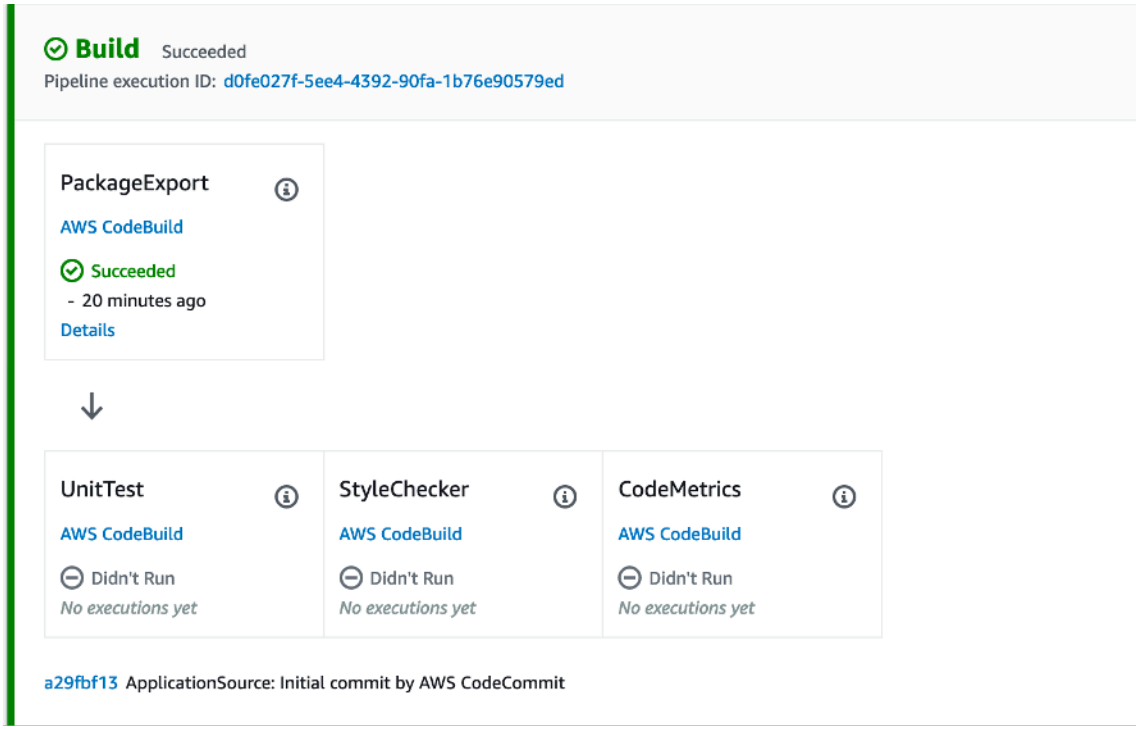


AWS CodePipeline – Quell- und Entwicklungsphasen

Für AWS CodePipeline kann die Quellphase Eingaben von GitHub, AWS CodeCommit und Amazon Simple Storage Service (Amazon S3) akzeptieren. Die Automatisierung des Erstellungsprozesses ist ein wichtiger erster Schritt zur Implementierung der Continuous Delivery und hin zum Continuous Deployment. Wenn bei der Produktion von Build-Artefakten kein manuelles Eingreifen mehr erforderlich ist, wird Ihr Team entlastet, es passieren weniger Fehler, die durch manuelles Verpacken entstehen, und Sie können häufiger verbrauchbare Artefakte verpacken.

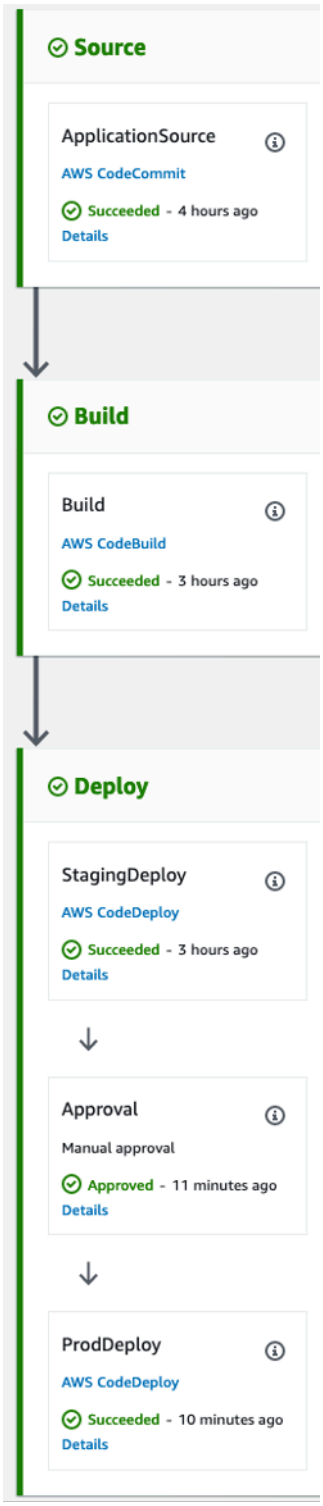
AWS CodePipeline arbeitet nahtlos mit AWS CodeBuild zusammen, einem vollständig verwalteten Build-Service. So ist es einfacher, einen Erstellungsschritt innerhalb Ihrer Pipeline einzurichten, der Ihren Code verpackt und Einheitentests durchführt. Mit AWS CodeBuild müssen Sie keine eigenen Buildserver bereitstellen, verwalten oder skalieren. AWS CodeBuild skaliert kontinuierlich und verarbeitet mehrere Builds gleichzeitig, sodass Ihre Builds nicht in einer Warteschlange warten müssen. AWS CodePipeline lässt sich auch in Buildserver wie Jenkins, Solano CI und TeamCity integrieren.

In der folgenden Entwicklungsphase werden beispielsweise drei Aktionen (Einheitentests, Codestilprüfungen und Erfassung von Codemetriken) parallel ausgeführt. Mit AWS CodeBuild können diese Schritte als neue Projekte ohne weiteren Aufwand beim Erstellen oder Installieren von Buildservern für die Verarbeitung der Last hinzugefügt werden.



AWS CodePipeline – Build-Funktionalität

Die in der Abbildung AWS CodePipeline – Quell- und Entwicklungsphasen gezeigten Quell- und Entwicklungsphasen sowie ergänzende Prozesse und Automatisierung unterstützen Ihr Team beim Übergang zur Continuous Integration. Bei diesem Reifegrad müssen Entwickler regelmäßig auf die Erstellungs- und Testergebnisse achten. Sie müssen wachsen und eine stabile Grundlage für Einheitentests aufrechterhalten. Dies stärkt wiederum das Vertrauen des gesamten Teams in die CI/CD-Pipeline und fördert deren Einführung.



AWS CodePipeline-Phasen

Pipeline für Continuous Delivery

Nachdem die Pipeline für die Continuous Integration implementiert wurde und ergänzende Prozesse eingerichtet wurden, können Ihre Teams mit dem Übergang zur Pipeline für die Continuous Delivery beginnen. Bei diesem Übergang müssen Teams sowohl die Erstellung als auch die Bereitstellung von Anwendungen automatisieren.

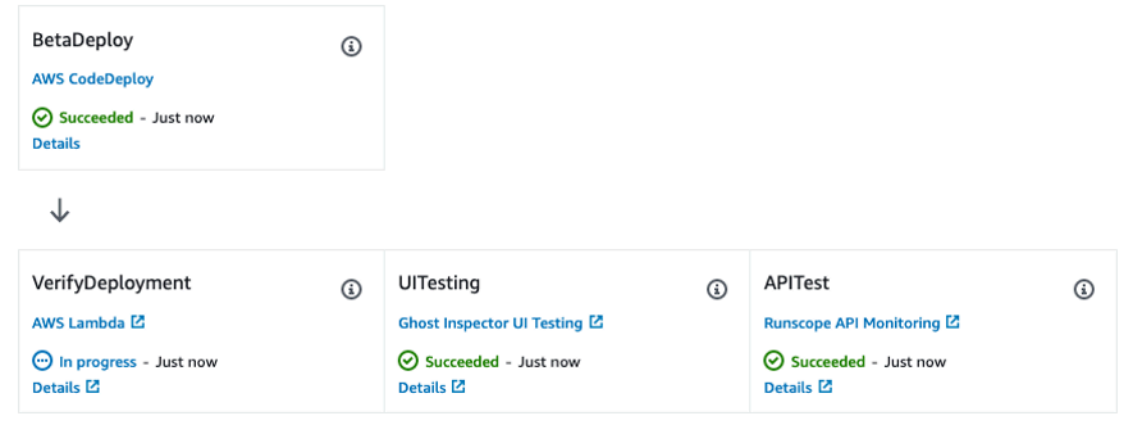
Eine Continuous-Delivery-Pipeline ist durch das Vorhandensein von Staging- und Produktionsschritten gekennzeichnet, bei denen der Produktionsschritt nach einer manuellen Genehmigung durchgeführt wird.

Auf die gleiche Weise, wie die Continuous-Integration-Pipeline entwickelt wurde, können Ihre Teams schrittweise mit dem Aufbau einer Continuous-Delivery-Pipeline beginnen, indem sie ihre Bereitstellungsskripte schreiben.

Abhängig von den Anforderungen einer Anwendung können einige der Bereitstellungsschritte durch vorhandene AWS-Services abstrahiert werden. AWS CodePipeline kann beispielsweise direkt mit den folgenden Services integriert werden: AWS CodeDeploy, einem Service, der Code-Bereitstellungen für Amazon-EC2-Instances und On-Premises-Instances automatisiert, AWS OpsWorks, einem Konfigurationsverwaltungsdienst, der Ihnen hilft, Anwendungen mit Chef zu betreiben, und AWS Elastic Beanstalk, einem Service für die Bereitstellung und Skalierung von Webanwendungen und -services.

AWS bietet eine ausführliche Dokumentation zur Implementierung und Integration von AWS CodeDeploy in Ihre Infrastruktur und Pipeline.

Nachdem Ihr Team die Bereitstellung der Anwendung erfolgreich automatisiert hat, können die Bereitstellungsphasen mit verschiedenen Tests erweitert werden. Sie können beispielsweise weitere sofort einsatzbereite Integrationen mit Diensten wie Ghost Inspector oder Runscope hinzufügen, wie in der folgenden Abbildung dargestellt.



AWS CodePipeline – Codetests in Bereitstellungsphasen

Lambda-Aktionen hinzufügen

AWS CodeStar und AWS CodePipeline unterstützen die Integration mit AWS Lambda. Diese Integration ermöglicht die Implementierung einer Vielzahl von Aufgaben, z. B. das Erstellen benutzerdefinierter Ressourcen in Ihrer Umgebung, die Integration in Systeme von Drittanbietern (wie Slack) und die Durchführung von Überprüfungen Ihrer neu bereitgestellten Umgebung.

Lambda-Funktionen können in CI/CD-Pipelines für die folgenden Aufgaben verwendet werden:

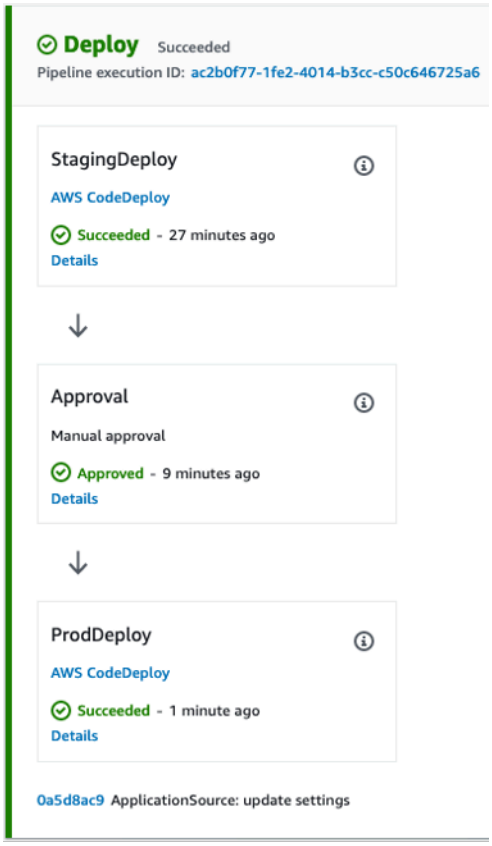
- Ausführen von Änderungen an Ihrer Umgebung durch das Anwenden oder Aktualisieren einer AWS CloudFormation-Vorlage
- Bedarfsabhängiges Erstellen von Ressourcen in einer Phase einer Pipeline mithilfe von AWS CloudFormation und Löschen der Ressourcen in einer anderen Phase
- Bereitstellen von Anwendungsversionen ohne Ausfallzeiten in AWS Elastic Beanstalk mit einer Lambda-Funktion, die Werte des Canonical-Name-Datensatzes (CNAME) austauscht
- Bereitstellung in Amazon Elastic Container Service (ECS)-Docker-Instances
- Sichern von Ressourcen mithilfe eines AMI-Snapshots vor dem Entwickeln oder Bereitstellen
- Integrieren von Drittanbieterprodukten in Ihre Pipeline, wie beispielsweise das Versenden von Nachrichten an einen Internet Relay Chat (IRC)-Client

Manuelle Genehmigungen

Fügen Sie einer Stufe in einer Pipeline eine Genehmigungsaktion hinzu, wenn Sie möchten, dass die Pipeline-Verarbeitung an einem bestimmten Punkt angehalten wird, damit jemand mit den

erforderlichen AWS Identity and Access Management (IAM)-Berechtigungen die Aktion erlauben oder ablehnen kann.

Wenn die Aktion genehmigt wurde, wird die Pipeline-Verarbeitung fortgesetzt. Wenn die Aktion abgelehnt wird – oder wenn niemand die Aktion innerhalb von sieben Tagen nach Erreichen der Aktion genehmigt oder ablehnt und die Pipeline stoppt – entspricht das Ergebnis einer fehlschlagenden Aktion und die Pipeline-Verarbeitung wird nicht fortgesetzt.



AWS CodeDeploy – Manuelle Genehmigungen

Änderungen des Infrastrukturcodes in einer CI/CD-Pipeline bereitstellen

Mit AWS CodePipeline können Sie in jeder Phase Ihrer Pipeline AWS CloudFormation als Bereitstellungsaktion auswählen. Sie können dann die spezifische Aktion auswählen, die AWS CloudFormation ausführen soll, z. B. Stacks erstellen oder löschen oder Änderungssätze erstellen und ausführen. Ein Stack ist ein AWS CloudFormation-Konzept und stellt eine Gruppe verwandter AWS-Ressourcen dar. Es gibt zwar viele Möglichkeiten, Infrastructure as Code bereitzustellen, AWS empfiehlt jedoch AWS CloudFormation als skalierbare, vollständige Lösung, die den umfassendsten Satz von AWS-Ressourcen als Code beschreiben kann. AWS empfiehlt

außerdem, AWS CloudFormation in einem AWS CodePipeline-Projekt zur Nachverfolgung von Infrastrukturänderungen und -tests zu verwenden.

CI/CD für Serverless-Anwendungen

Sie können auch AWS CodeStar, AWS CodePipeline, AWS CodeBuild und AWS CloudFormation verwenden, um CI/CD-Pipelines für Serverless-Anwendungen zu entwickeln. Serverless-Anwendungen integrieren verwaltete Services wie Amazon Cognito, Amazon S3 und Amazon DynamoDB mit ereignisgesteuerten Diensten und AWS Lambda zur Bereitstellung von Anwendungen auf eine Weise, die keine Verwaltung von Servern erfordert. Wenn Sie Serverless-Anwendungen entwickeln, können Sie eine Kombination von AWS CodePipeline, AWS CodeBuild und AWS CloudFormation verwenden, um das Erstellen, Testen und Bereitstellen von Serverless-Anwendungen zu automatisieren, die in Vorlagen ausgedrückt werden, die mit dem AWS Serverless Application Model erstellt wurden. Weitere Informationen finden Sie in der AWS Lambda-Dokumentation zum Automatisieren der Bereitstellung von Lambda-basierten Anwendungen.

Mit AWS Serverless Application Model Pipelines (AWS SAM Pipelines) können Sie auch sichere CI/CD-Pipelines erstellen, die den bewährten Methoden Ihrer Organisation entsprechen. AWS SAM Pipelines ist eine neue Funktion von AWS SAM CLI, mit der Sie innerhalb von Minuten auf die Vorteile von CI/CD zugreifen können, z. B. Beschleunigung der Bereitstellungshäufigkeit, Verkürzung der Vorlaufzeit für Änderungen und Reduzierung von Bereitstellungsfehlern. AWS SAM Pipelines werden mit einer Reihe von Standard-Pipeline-Vorlagen für AWS CodeBuild/CodePipeline geliefert, die den Best Practices der AWS-Bereitstellung folgen. Weitere Informationen und eine Möglichkeit, sich das Tutorial anzusehen, finden Sie im Blog Introducing AWS SAM Pipelines.

Pipelines für mehrere Teams, Zweige und AWS-Regionen

Bei einem großen Projekt ist es nicht ungewöhnlich, dass mehrere Projektteams an verschiedenen Komponenten arbeiten. Wenn mehrere Teams ein einziges Code-Repository verwenden, kann es so ausgelegt werden, dass jedes Team seinen eigenen Zweig hat. Es sollte auch einen Integrations- oder Release-Zweig für die endgültige Zusammenführung des Projekts geben. Wenn eine serviceorientierte oder Microservice-Architektur verwendet wird, könnte jedes Team sein eigenes Code-Repository haben.

Wird im ersten Szenario eine einzelne Pipeline verwendet, ist es möglich, dass ein Team den Fortschritt der anderen Teams aufhält, weil es die Pipeline blockiert. AWS empfiehlt, dass Sie spezifische Pipelines für Teamzweige und eine weitere Release-Pipeline für die endgültige Produktlieferung erstellen.

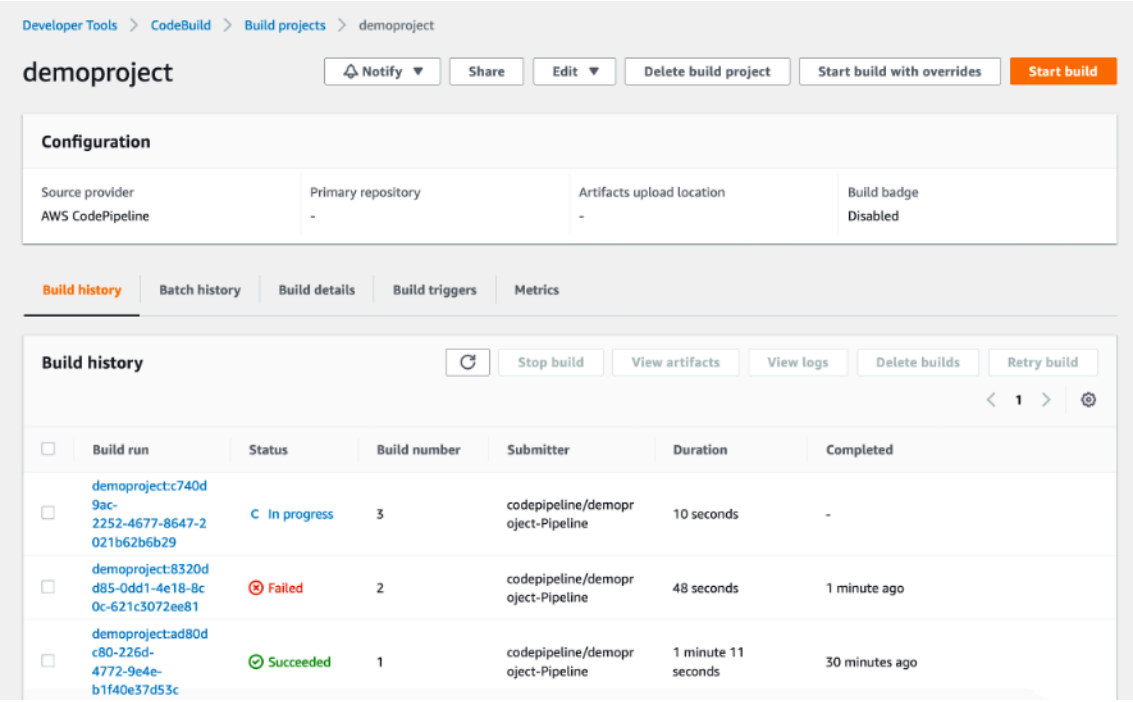
Pipeline-Integration mit AWS CodeBuild

AWS CodeBuild soll es Ihrer Organisation ermöglichen, einen hochverfügbaren Erstellungsprozess mit nahezu unbegrenzter Skalierung aufzubauen. AWS CodeBuild bietet Schnellstartumgebungen für eine Reihe gängiger Sprachen sowie die Möglichkeit, jeden von Ihnen angegebenen Docker-Container auszuführen.

Dank der Vorteile einer engen Integration mit AWS CodeCommit, AWS CodePipeline und AWS CodeDeploy sowie Git- und CodePipeline Lambda-Aktionen ist das CodeBuild-Tool sehr flexibel.

Software kann durch die Einbeziehung einer `buildspec.yml`-Datei erstellt werden, die jeden der Entwicklungsschritte identifiziert, einschließlich Aktionen vor und nach dem Erstellen oder bestimmter Aktionen, die durch das CodeBuild-Tool ausgeführt werden.

Sie können den detaillierten Verlauf jedes Builds mithilfe des CodeBuild-Dashboards anzeigen. Ereignisse werden als Amazon CloudWatch Logs-Protokolldateien gespeichert.



CloudWatch Logs-Protokolldateien in AWS CodeBuild

Pipeline-Integration mit Jenkins

Sie können das Jenkins-Entwicklungstool verwenden, um Delivery-Pipelines zu erstellen. Diese Pipelines verwenden Standardaufträge, die Schritte zur Implementierung der Continuous-Delivery-

Phasen definieren. Dieser Ansatz ist jedoch möglicherweise nicht optimal für größere Projekte, da der aktuelle Status der Pipeline zwischen den Neustarts von Jenkins nicht fortbesteht, die Implementierung der manuellen Genehmigung nicht einfach ist und die Nachverfolgung des Status einer komplexen Pipeline kompliziert sein kann.

Stattdessen empfiehlt AWS, dass Sie die Continuous Delivery mit Jenkins mithilfe des AWS Code Pipeline-Plug-Ins implementieren. Dieses Plug-In ermöglicht die Beschreibung komplexer Workflows mit einer domänenspezifischen, Groovy-ähnlichen Sprache und kann zur Orchestrierung komplexer Pipelines verwendet werden. Die Funktionalität des AWS Code Pipeline-Plug-Ins kann durch die Verwendung von Satelliten-Plug-Ins erweitert werden, z. B. mit dem Pipeline Stage View-Plug-In, das den aktuellen Fortschritt der in einer Pipeline definierten Phasen visualisiert, oder dem Pipeline Multibranch-Plug-In, das Builds aus verschiedenen Zweigen gruppiert.

AWS empfiehlt, dass Sie Ihre Pipeline-Konfiguration in Jenkinsfile speichern und in ein Quellcode-Repository einchecken lassen. Dies ermöglicht das Nachverfolgen von Änderungen am Pipeline-Code und wird bei der Arbeit mit dem Pipeline Multibranch-Plug-In noch wichtiger. AWS empfiehlt außerdem, dass Sie Ihre Pipeline in Phasen aufteilen. Dadurch werden die Pipeline-Schritte logisch gruppiert und das Pipeline Stage View-Plug-In ermöglicht, den aktuellen Zustand der Pipeline zu visualisieren.

Die folgende Abbildung zeigt ein Beispiel für eine Jenkins-Pipeline mit vier definierten Phasen, die vom Pipeline Stage View-Plug-In visualisiert werden.

Definierte Phasen der Jenkins-Pipeline, visualisiert durch das Pipeline Stage View-Plug-In

Bereitstellungsmethoden

Sie können mehrere Bereitstellungsstrategien und -varianten für die Einführung neuer Softwareversionen in einem Continuous-Delivery-Prozess in Betracht ziehen. In diesem Abschnitt werden die gängigsten Bereitstellungsmethoden erläutert: alle auf einmal (direkt bereitstellen), fortlaufend, unveränderlich und blau/grün. AWS gibt an, welche dieser Methoden von AWS CodeDeploy und AWS Elastic Beanstalk unterstützt werden.

In der folgenden Tabelle sind die Merkmale der einzelnen Bereitstellungsmethoden zusammengefasst.

Methode	Auswirkungen einer fehlgeschlagenen Bereitstellung	Bereitstellungszeit	Ohne Ausfallzeit	Keine DNS-Änderung	Rollback-Vorgang	Code bereitgestellt für
Direkt bereitstellen	Ausfallzeit	🕒	X	✓	Erneut bereitstellen	Vorhandene Instances
Fortlaufend	Einzelner Batch außer Betrieb. Alle erfolgreichen Batches führen vor dem Ausfall die neue Anwendung	🕒 🕒 †	✓	✓	Erneut bereitstellen	Vorhandene Instances

Methode	Auswirkungen einer fehlgeschlagenen Bereitstellung	Bereitstellungszeit	Ohne Ausfallzeit	Keine DNS-Änderung	Rollback-Vorgang	Code bereitgestellt für
	sversion aus.					
Fortlaufend mit zusätzlichem Batch (Beanstalk)	Minimal, wenn der erste Batch fehlschlägt, ansonsten ähnlich wie Rolling (Fortlaufend).	🕒 🕒 🕒 †	✓	✓	Erneut bereitstellen	Neue und vorhandene Instances
Immutable (Unveränderlich)	Minimal	🕒 🕒 🕒 🕒	✓	✓	Erneut bereitstellen	Neue Instances
Datenverkehrsaufteilung	Minimal	🕒 🕒 🕒 🕒	✓	✓	Datenverkehr umleiten und neue Instances beenden	Neue Instances
Blau/Grün	Minimal	🕒 🕒 🕒 🕒	✓	X	Zur alten Umgebung zurückwechseln	Neue Instances

Direkt (In-Place-Bereitstellung)

Die direkte Bereitstellung (In-Place-Bereitstellung) ist eine Methode, mit der Sie neuen Anwendungscode auf einer vorhandenen Serverflotte bereitstellen können. Diese Methode ersetzt den gesamten Code in einer Bereitstellungsaktion. Sie erfordert Ausfallzeiten, da alle Server in der Flotte auf einmal aktualisiert werden. Es ist nicht erforderlich, vorhandene DNS-Datensätze zu aktualisieren. Im Falle einer fehlgeschlagenen Bereitstellung können Vorgänge nur wiederhergestellt werden, indem der Code erneut auf allen Servern bereitgestellt wird.

In AWS Elastic Beanstalk heißt diese Bereitstellungsart All at once und ist für einzelne Anwendungen sowie Anwendungen mit Lastenausgleich verfügbar. In AWS CodeDeploy heißt diese Bereitstellungsmethode In-Place-Bereitstellung mit einer Bereitstellungskonfiguration von `AllAtOnce`.

Fortlaufende Bereitstellung

Bei der fortlaufenden Bereitstellung wird die Flotte aufgeteilt, sodass nicht für die gesamte Flotte auf einmal ein Upgrade ausgeführt wird. Während des Bereitstellungsprozesses werden zwei Softwareversionen, neue und alte, in derselben Flotte ausgeführt. Diese Methode ermöglicht ein Update ohne Ausfallzeit. Wenn die Bereitstellung fehlschlägt, ist nur der aktualisierte Teil der Flotte betroffen.

Bei einer Variante der fortlaufenden Bereitstellungsmethode, Canary-Release genannt, wird die neue Softwareversion zunächst auf einem sehr kleinen Prozentsatz der Server bereitgestellt. Auf diese Weise können Sie auf einigen Servern beobachten, wie sich die Software in der Produktion verhält, und gleichzeitig die Auswirkungen von Änderungen minimieren. Wenn bei einer Canary-Bereitstellung eine erhöhte Fehlerrate auftritt, wird ein Rollback der Software ausgeführt. Andernfalls wird der Anteil der Server mit der neuen Version schrittweise erhöht.

AWS Elastic Beanstalk folgt dem fortlaufenden Bereitstellungsmuster mit zwei Bereitstellungsoptionen, fortlaufend und fortlaufend mit zusätzlichem Batch. Mit diesen Optionen kann die Anwendung zunächst hochskaliert werden, bevor Server außer Betrieb genommen werden, wobei die volle Funktionsfähigkeit während der Bereitstellung erhalten bleibt. AWS CodeDeploy setzt dieses Muster als eine Variante einer direkten Bereitstellung mit Mustern wie OneAtATime und HalfAtATime um.

Unveränderliche und Blau-Grün-Bereitstellung

Mit einem unveränderlichen Muster wird Anwendungscode bereitgestellt, indem ein völlig neuer Satz von Servern mit einer neuen Konfiguration oder Version des Anwendungscodes gestartet wird. Dieses Muster nutzt die Cloud-Fähigkeit, dass neue Serverressourcen mit einfachen API-Aufrufen erstellt werden.

Die Blau-/Grün-Bereitstellungsstrategie ist eine Art der unveränderlichen Bereitstellung, die zusätzlich die Erstellung einer anderen Umgebung erfordert. Sobald die neue Umgebung eingerichtet ist und alle Tests bestanden wurden, wird der Datenverkehr auf diese neue Bereitstellung verlagert. Wichtig ist dabei, dass die alte Umgebung, d. h. die „blaue“ Umgebung, im Leerlauf gehalten wird, falls ein Rollback erforderlich ist.

AWS Elastic Beanstalk unterstützt unveränderliche und Blau-Grün-Bereitstellungsmuster. AWS CodeDeploy unterstützt ebenfalls das Blau-Grün-Muster. Weitere Informationen darüber, wie AWS-Services diese unveränderlichen Muster umsetzen, finden Sie im Whitepaper Blau-Grün-Bereitstellungen in AWS.

Änderungen des Datenbankschemas

Moderne Software verfügt häufig über eine Datenbankebene. In der Regel wird eine relationale Datenbank verwendet, in der sowohl Daten als auch die Struktur der Daten gespeichert werden. Oft ist es notwendig, die Datenbank im Continuous-Delivery-Prozess zu modifizieren. Der Umgang mit Änderungen in einer relationalen Datenbank erfordert besondere Überlegungen. Hierbei sind andere Herausforderungen präsent als bei der Bereitstellung von Anwendungsbinärdateien. Wenn Sie eine Anwendungsbinärdatei aktualisieren, beenden Sie normalerweise die Anwendung, aktualisieren sie und starten sie dann erneut. Sie kümmern sich nicht wirklich um den Anwendungsstatus, der außerhalb der Anwendung behandelt wird.

Beim Upgrade von Datenbanken müssen Sie den Status jedoch berücksichtigen, da eine Datenbank viel Status, aber vergleichsweise wenig Logik und Struktur enthält.

Das Datenbankschema vor und nach der Anwendung einer Änderung sollte jeweils als unterschiedliche Version der Datenbank betrachtet werden. Mit Tools wie Liquibase und Flyway können Sie die Versionen verwalten.

Im Allgemeinen verwenden diese Tools Varianten der folgenden Methoden:

- Der Datenbank eine Tabelle hinzufügen, in der eine Datenbankversion gespeichert ist.
- Die Befehle zur Datenbankänderung protokollieren und in versionierten Änderungssätzen zusammenfassen. Im Fall von Liquibase werden diese Änderungen in XML-Dateien gespeichert. Flyway verwendet eine etwas andere Methode, bei der die Änderungssätze als separate SQL-Dateien oder gelegentlich als separate Java-Klassen für komplexere Übergänge behandelt werden.
- Wenn Liquibase eine Datenbank aktualisieren soll, überprüft es die Metadatentabelle und bestimmt, welche Änderungssätze ausgeführt werden, um die Datenbank mit der neuesten Version auf den neuesten Stand zu bringen.

Zusammenfassung der bewährten Methoden

Nachstehend sind einige bewährte Methoden und zu vermeidende Prozesse für CI/CD aufgeführt.

Empfohlen:

- Behandeln Sie Ihre Infrastruktur als Code.
  - Verwenden Sie die Versionskontrolle für Ihren Infrastrukturcode.
  - Nutzen Sie Bugtracking-/Ticketing-Systeme.
  - Lassen Sie Kollegen Änderungen überprüfen, bevor Sie sie anwenden.
  - Richten Sie Infrastruktur-Codemuster/-designs ein.
  - Testen Sie Infrastrukturänderungen wie Codeänderungen.
- Stellen Sie integrierte Entwicklerteams mit nicht mehr als 12 autarken Mitgliedern zusammen.
- Lassen Sie alle Entwickler für ihren Code häufig Commits in den Hauptstamm ausführen, ohne lange Funktionszweige.
- Übernehmen Sie konsequent ein Build-System wie Maven oder Gradle in Ihrer gesamten Organisation und standardisieren Sie Builds.
- Lassen Sie die Entwickler Einheitentests entwickeln, um die Codebasis zu annähernd 100% abzudecken.
- Stellen Sie sicher, dass die Einheitentests in Dauer, Anzahl und Umfang 70% der gesamten Tests ausmachen.
- Stellen Sie sicher, dass die Einheitentests auf dem neuesten Stand sind und nicht vernachlässigt werden. Fehler bei Einheitentests sollten behoben, nicht umgangen werden.
- Behandeln Sie Ihre Continuous-Delivery-Konfiguration als Code.
- Richten Sie rollenbasierte Sicherheitskontrollen ein (d. h. legen Sie fest, wer wann was tun kann).
  - Überwachen/verfolgen Sie nach Möglichkeit jede Ressource.
  - Richten Sie Warnungen zu Diensten, Verfügbarkeit und Reaktionszeiten ein.
  - Halten Sie sich an das Motto „Erfassen, lernen und verbessern“.
  - Teilen Sie den Zugriff mit allen Teammitgliedern.
  - Planen Sie Metriken und Überwachung für den Lebenszyklus.
- Definieren und verfolgen Sie Standardmetriken.
  - Anzahl der Builds

  - Anzahl der Bereitstellungen
  - Durchschnittliche Zeit, bis Änderungen für die Produktion übernommen werden
  - Durchschnittliche Zeit von der ersten Pipeline-Phase bis zu jeder einzelnen Phase
  - Anzahl der Änderungen, die für die Produktion übernommen werden
  - Durchschnittliche Entwicklungszeit
- Verwenden Sie für jeden Zweig und jedes Team eigene Pipelines.

Zu vermeiden:

- Lange Zweige mit großen komplizierten Zusammenführungen
- Manuelle Tests
- Manuelle Genehmigungsprozesse, Gates, Codeüberprüfungen und Sicherheitsüberprüfungen

Fazit

Continuous Integration und Continuous Delivery bieten ein ideales Szenario für die Anwendungsteams Ihrer Organisation. Die Entwickler übertragen ihren Code einfach in ein Repository. Dieser Code wird integriert, getestet, bereitgestellt, erneut getestet, mit der Infrastruktur zusammengeführt, durchläuft Sicherheits- und Qualitätsprüfungen und kann mit extrem hoher Sicherheit bereitgestellt werden.

Wenn CI/CD verwendet wird, wird die Codequalität verbessert und Software-Updates werden schnell und mit hoher Gewissheit geliefert, dass es keine wesentlichen Änderungen geben wird. Die Auswirkungen einer beliebigen Version können mit Daten aus Produktion und Betrieb korreliert werden. CI/CD kann auch für die Planung des nächsten Zyklus verwendet werden – eine wichtige DevOps-Praxis bei der Cloud-Transformation Ihrer Organisation.

Weitere Informationen

Weitere Informationen zu den in diesem Whitepaper behandelten Themen finden Sie in den folgenden AWS-Whitepapers:

- Übersicht über die Bereitstellungsoptionen in AWS
- Blau-Grün-Bereitstellungen in AWS
- CI/CD-Pipelines durch die Integration von Jenkins mit AWS CodeBuild und AWS CodeDeploy einrichten
- Microservices in AWS
- Docker in AWS: Container in der Cloud betreiben

Mitwirkende

Dieses Dokument ist unter der Mitarbeit folgender Personen und Unternehmen entstanden:

- Amrish Thakkar, Principal Solutions Architect, AWS
- David Stacy, Senior Consultant – DevOps, AWS Professional Services
- Asif Khan, Solutions Architect, AWS
- Xiang Shen, Senior Solutions Architect, AWS

Dokumentversionen

Abonnieren Sie den RSS-Feed, um über Aktualisierungen des Whitepapers benachrichtigt zu werden.

Update-Historie-Änderung	Update-Historie-Beschreibung	Update-Historie-Datum
Erstveröffentlichung	Erstveröffentlichung des Whitepapers	27. Oktober 2021
Erstveröffentlichung	Erstveröffentlichung des Whitepapers	1. Juni 2017

Hinweise

Kunden sind eigenverantwortlich für die unabhängige Bewertung der Informationen in diesem Dokument zuständig. Dieses Dokument: (a) dient rein zu Informationszwecken, (b) spiegelt die aktuellen Produktangebote und Verfahren von AWS wider, die sich ohne vorherige Mitteilung ändern können, und (c) impliziert keinerlei Verpflichtungen oder Zusicherungen seitens AWS und dessen Tochtergesellschaften, Lieferanten oder Lizenzgebern. AWS-Produkte oder -Services werden im vorliegenden Zustand und ohne ausdrückliche oder stillschweigende Gewährleistungen, Zusicherungen oder Bedingungen bereitgestellt. Die Verantwortung und Haftung von AWS gegenüber seinen Kunden wird durch AWS-Vereinbarungen geregelt. Dieses Dokument ist weder ganz noch teilweise Teil der Vereinbarungen zwischen AWS und seinen Kunden und ändert diese Vereinbarungen auch nicht.