Whitepaper AWS

Praticare l'integrazione e la consegna continue in AWS

Praticare l'integrazione e la consegna continue in AWS: Whitepaper AWS

Table of Contents

Praticare l'integrazione e la consegna continue in AWS

Data di pubblicazione: 27 ottobre 2021 (Revisioni del documento)

Riassunto

Questo whitepaper spiega le caratteristiche e i vantaggi dell'utilizzo di integrazione e consegna continue (CI/CD) con gli strumenti di Amazon Web Services (AWS) in un ambiente di sviluppo software. L'integrazione e la consegna continue sono best practice e una parte fondamentale delle iniziative DevOps.

La sfida della distribuzione del software

Le aziende oggi devono affrontare le sfide poste da scenari competitivi in rapida evoluzione, requisiti di sicurezza in costante mutamento e scalabilità delle prestazioni. Le aziende devono colmare il divario tra la stabilità delle operazioni e lo sviluppo rapido delle funzioni. L'integrazione continua e la consegna continua (CI/CD) sono due pratiche che consentono di apportare rapide modifiche al software mantenendo al contempo la stabilità e la sicurezza del sistema.

Amazon ha realizzato fin da subito che l'esigenza aziendale di rilasciare funzioni per i clienti al dettaglio di Amazon.com, per le filiali di Amazon e per gli Amazon Web Services (AWS) avrebbe richiesto metodi nuovi e innovativi di distribuzione del software. Nel caso delle aziende delle dimensioni di Amazon, migliaia di team software indipendenti devono essere in grado di lavorare in parallelo per distribuire software in modo rapido, sicuro, affidabile e senza alcuna tolleranza per le interruzioni.

Scoprendo come distribuire software ad alta velocità, Amazon e altre organizzazioni lungimiranti hanno aperto la strada all'approccio DevOps. Le prassi DevOps sono un connubio tra cultura, pratica e strumenti per migliorare le capacità di un'azienda di distribuire applicazioni e servizi a velocità elevata. Utilizzando i principi DevOps, le organizzazioni possono evolvere e migliorare i prodotti a un ritmo più veloce rispetto alle organizzazioni che utilizzano processi di sviluppo software e gestione dell'infrastruttura tradizionali. Questa velocità consente alle aziende di offrire servizi migliori ai clienti e offre una maggiore competitività sul mercato.

Alcuni di questi principi, come il two-pizza teams e l'architettura SOA (Microservice/Service-Oriented Architecture), non rientrano nell'ambito di questo whitepaper, che illustra le funzionalità CI/CD che Amazon ha creato e che non ha mai smesso di migliorare. CI/CD è fondamentale per distribuire funzioni software in modo rapido e affidabile.

AWS ora offre queste funzionalità CI/CD sotto forma di un insieme di servizi per sviluppatori: AWS CodeStar, AWS CodeCommit, AWS CodePipeline, AWS CodeBuild, AWS CodeDeploy e AWS CodeArtifact. I professionisti nell'ambito dello sviluppo e delle operazioni IT che praticano l'approccio DevOps possono utilizzare questi servizi per distribuire software in modo rapido e sicuro. Insieme, ti aiutano ad archiviare e ad applicare in modo sicuro il controllo delle versioni al codice sorgente della tua applicazione. Puoi utilizzare AWS CodeStar per orchestrare rapidamente un flusso di lavoro di rilascio software end-to-end utilizzando questi servizi. Nel caso di un ambiente esistente, AWS CodePipeline ha la flessibilità di integrare ogni servizio in modo indipendente con gli strumenti esistenti. Si tratta di servizi altamente disponibili e facilmente integrati a cui è possibile accedere

tramite la AWS Management Console, le interfacce del programma dell'applicazione (API) di AWS e i kit di strumenti di sviluppo software (SDK) AWS come per qualsiasi altro servizio AWS.

Che cos'è l'integrazione continua e la consegna/implementazione continua?

Questa sezione illustra le pratiche di integrazione e consegna continua e spiega la differenza tra la consegna continua e l'implementazione continua.

Integrazione continua

Integrazione continua (CI): si tratta di una pratica di sviluppo software in cui gli sviluppatori uniscono regolarmente le modifiche al codice in un repository centrale, su cui vengono applicati test automatici e build. La CI molto spesso si riferisce alla fase di compilazione o integrazione del processo di rilascio di software e richiede sia una componente di automazione (ad esempio un servizio di CI o di compilazione) che una componente culturale (ad esempio abituarsi a integrare in modo frequente). Gli obiettivi principali della CI sono individuare e risolvere i bug con maggiore tempestività, migliorare la qualità del software e ridurre il tempo richiesto per convalidare e pubblicare nuovi aggiornamenti.

L'integrazione continua si concentra su commit più piccoli e modifiche al codice di minore entità da integrare. Uno sviluppatore esegue il commit del codice a intervalli regolari, almeno una volta al giorno. Lo sviluppatore estrae il codice dal repository del codice per assicurarsi che tale codice venga unito sull'host locale prima di eseguire il push sul server di sviluppo. In questa fase, il server di sviluppo esegue i vari test e accetta o rifiuta il commit del codice.

Le sfide di base dell'implementazione della CI includono commit più frequenti nella base di codice comune, il mantenimento di un unico repository di codice sorgente, l'automazione delle build e l'automazione dei test. Altre sfide includono l'esecuzione di in ambienti simili a quello di produzione, per fornire al team visibilità sul processo e consentire agli sviluppatori di ottenere facilmente qualsiasi versione dell'applicazione.

Consegna e implementazione continue

La consegna continua (CD) è un metodo di sviluppo software in cui le modifiche al codice vengono compilate, testate e preparate per il rilascio nell'ambiente di produzione in modo automatico. Espande il processo di integrazione continua implementando tutte le modifiche al codice nell'ambiente di test e di produzione (o in entrambi) dopo la fase di compilazione. La consegna continua può essere completamente automatizzata con un processo del flusso di lavoro o parzialmente automatizzata con passaggi manuali nei punti critici. Se la consegna continua viene implementata correttamente, gli

sviluppatori hanno sempre a disposizione un artefatto della build pronto per l'implementazione che ha già passato un processo di test standardizzato.

Con l'implementazione continua, le revisioni vengono implementate automaticamente nell'ambiente di produzione senza alcuna approvazione esplicita da parte di uno sviluppatore, automatizzando di fatto l'intero processo di rilascio del software. Questo, a sua volta, consente un circuito continuo di feedback dei clienti nelle prime fasi del ciclo di vita del prodotto.

La consegna continua è diversa dall'implementazione continua

Un pensiero errato sulla consegna continua è la convinzione che tale processo implichi che ogni modifica sottoposta a commit venga applicata all'ambiente di produzione immediatamente dopo aver superato i test automatici. Tuttavia, il punto della consegna continua non è applicare immediatamente ogni modifica all'ambiente di produzione, ma garantire che ogni modifica sia pronta per essere inserita nell'ambiente di produzione.

Prima di implementare una modifica nell'ambiente di produzione, è possibile implementare un processo decisionale per garantire che l'implementazione nell'ambiente di produzione sia autorizzata e sottoposta a verifica. Questa decisione può essere presa da una persona e quindi eseguita dagli strumenti.

Utilizzando la consegna continua, la decisione di procedere al lancio diventa una decisione aziendale e non tecnica. La convalida tecnica avviene su ogni commit.

L'introduzione di una modifica nell'ambiente di produzione non è un evento di disturbo. L'implementazione non richiede che il team tecnico smetta di lavorare sul successivo set di modifiche e non necessita di un piano del progetto, di una documentazione di consegna o di un intervallo di tempo dedicato alla manutenzione. L'implementazione diventa un processo ripetibile che è stato eseguito e collaudato più volte negli ambienti di test.

Vantaggi della consegna continua

La CD offre numerosi vantaggi al team di sviluppo software, tra cui l'automazione dei processi, il miglioramento della produttività degli sviluppatori, il miglioramento della qualità del codice e la consegna più rapida degli aggiornamenti ai clienti.

Processo di rilascio del software automatizzato

La CD offre al team un metodo per eseguire check-in del codice che viene automaticamente compilato, testato e preparato per il rilascio nell'ambiente di produzione per rendere la distribuzione del software efficiente, resiliente, rapida e sicura.

Aumenta la produttività degli sviluppatori

Le pratiche CD aiutano la produttività del team liberando gli sviluppatori dalle attività manuali, districando dipendenze complesse e concentrando l'attenzione sulla fornitura di nuove funzioni nel software. Invece di integrare il codice con altre parti dei cicli di business e di spesa su come implementare questo codice su una piattaforma, gli sviluppatori possono concentrarsi sulla logica di codifica per rilasciare tutte le funzionalità necessarie.

Migliorare la qualità del codice

La CD può aiutarti a scoprire e risolvere i bug nelle prime fasi del processo di consegna prima che si trasformino in problemi più grandi nelle fasi successive. Il tuo team può eseguire con facilità altri tipi di test del codice perché l'intero processo è stato automatizzato. Grazie all'esecuzione di un numero maggiore di test più frequentemente, i team possono iterare più velocemente grazie ai feedback immediati sull'impatto delle modifiche. In questo modo, potranno gestire codice di qualità con un'elevata garanzia di stabilità e sicurezza. Gli sviluppatori sapranno attraverso un feedback immediato se il nuovo codice funziona e se sono state introdotte modifiche o bug. Gli errori rilevati nelle prime fasi del processo di sviluppo sono i più facili da correggere.

Fornisci aggiornamenti più rapidamente

Con la CD il tuo team può rilasciare aggiornamenti per i clienti in modo rapido e con frequenza. Quando la CI/CD viene implementata, la velocità dell'intero team, incluso il rilascio di funzioni e

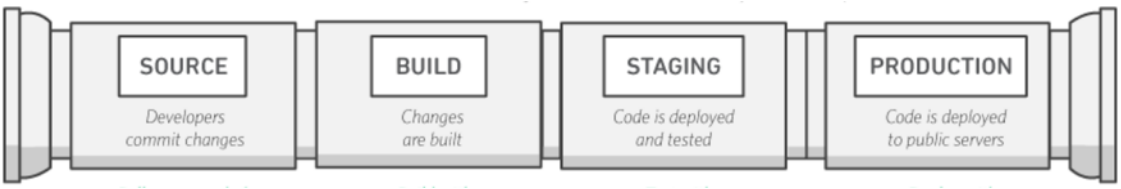
correzioni di bug, aumenta. Le aziende possono rispondere più rapidamente ai cambiamenti del mercato, alle sfide di sicurezza, alle esigenze dei clienti e alle pressioni sui costi. Ad esempio, se è necessaria una nuova funzione di sicurezza, il team può implementare la CI/CD con test automatici per introdurre la correzione in modo rapido e affidabile nei sistemi di produzione con elevata affidabilità. Ciò che prima richiedeva settimane e mesi, ora può essere completato in giorni o addirittura ore.

Implementazione dell'integrazione continua e della consegna continua

In questa sezione vengono illustrati i modi in cui è possibile iniziare a implementare un modello CI/CD nell'organizzazione. In questo whitepaper non viene trattato il modo in cui le organizzazioni con un modello di trasformazione cloud e DevOps maturo creino e utilizzino una pipeline CI/CD. Per accompagnarti nel tuo percorso DevOps, AWS ha una serie di partner DevOps certificati che saranno in grado di fornirti risorse e strumenti. Per ulteriori informazioni sulla preparazione per il passaggio ad AWS Cloud, consulta Realizzazione di un modello operativo su cloud.

Un percorso verso l'integrazione e la consegna continue

L'approccio CI/CD può essere rappresentato come una pipeline (osserva la figura seguente), in cui il nuovo codice viene inviato su un'estremità, testato durante una serie di fasi (origine, compilazione, gestione temporanea e produzione) e quindi pubblicato come codice pronto per l'ambiente di produzione. Se la tua organizzazione non conosce CI/CD, può affrontare questa pipeline in modo iterativo. Ciò significa che è consigliabile iniziare con poco e iterare in ogni fase in modo da poter comprendere e sviluppare il codice per supportare la crescita dell'organizzazione.



Pipeline CI/CD

Ogni fase della pipeline CI/CD è strutturata come un'unità logica nel processo di consegna. Inoltre, ogni fase funge da gate che controlla un determinato aspetto del codice. Man mano che il codice avanza attraverso la pipeline, si presume che la sua qualità sia più alta nelle fasi successive perché ne vengono verificati sempre più aspetti. I problemi rilevati in una fase iniziale impediscono al codice di progredire nella pipeline. I risultati dei test vengono immediatamente inviati al team e tutte le successive build e rilasci vengono interrotti se il software non supera la fase.

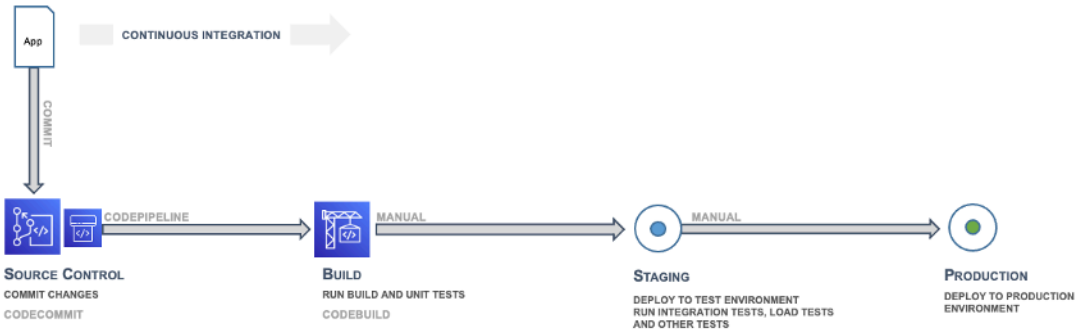
Queste fasi sono dei suggerimenti. Puoi adattarle alle esigenze aziendali. Alcune fasi possono essere ripetute per diversi tipi di test, sicurezza e prestazioni. A seconda della complessità del progetto e della struttura dei team, alcune fasi possono essere ripetute più volte a diversi livelli. Ad esempio,

il prodotto finale di un team può diventare una dipendenza nel progetto del team successivo. Ciò significa che il prodotto finale del primo team viene successivamente approntato come artefatto nel progetto del team successivo.

La presenza di una pipeline CI/CD avrà un grande impatto sulla maturazione delle capacità della tua organizzazione. Per iniziare, l'organizzazione dovrebbe iniziare con fasi di piccole dimensioni invece di cercare di costruire una pipeline completamente matura, con più ambienti e molte fasi di test e automazione in tutte le fasi. Tieni presente che anche le organizzazioni che dispongono di ambienti CI/CD estremamente maturi non smettono comunque di migliorare le proprie pipeline.

Costruire un'organizzazione abilitata per CI/CD è un viaggio, con molte destinazioni lungo il percorso. La sezione successiva illustra un possibile percorso che la tua organizzazione potrebbe intraprendere, a partire dall'integrazione continua attraverso i livelli di consegna continua.

Integrazione continua



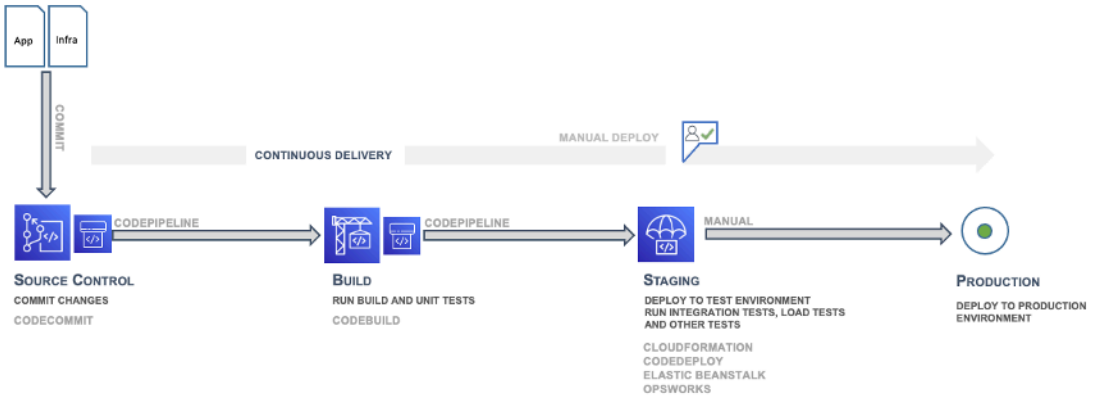
Integrazione continua: origine e compilazione

La prima fase del percorso CI/CD prevede lo sviluppo della maturità nell'integrazione continua. Assicurati che tutti gli sviluppatori eseguano regolarmente il commit del codice in un repository centrale (ad esempio uno ospitato in CodeCommit o GitHub) e che uniscano tutte le modifiche in un branch di rilascio dell'applicazione. Nessuno sviluppatore deve tenere il codice in isolamento. Se per un certo periodo di tempo è necessario un branch di funzioni, questo dovrebbe essere mantenuto aggiornato tramite il merge da upstream il più frequentemente possibile. I commit e i merge frequenti con le unità di lavoro complete sono due pratiche che il team dovrebbe svolgere al fine di sviluppare la disciplina e sono inoltre incoraggiate dal processo. Uno sviluppatore che unisce il codice nelle prime fasi del processo e con frequenza avrà probabilmente meno problemi di integrazione nelle fasi successive del processo.

Ti consigliamo inoltre di incoraggiare gli sviluppatori a creare il prima possibile unit test per le applicazioni e ad eseguirli prima di inviare il codice al repository centrale. Gli errori rilevati nelle prime fasi del processo di sviluppo del software sono i più economici e facili da correggere.

Quando il codice viene inviato a un branch in un repository del codice sorgente, un motore del flusso di lavoro che monitora quel branch invierà a uno strumento di compilazione un comando per la compilazione del codice e l'esecuzione degli unit test in un ambiente controllato. Il processo di compilazione deve essere delle dimensioni appropriate per la gestione di tutte le attività, incluse quelle di push e di test che potrebbero verificarsi durante la fase di commit, per l'invio di feedback rapidi. In questa fase possono avere luogo anche altri controlli di qualità, come la copertura degli unit test, il controllo dello stile e l'analisi statica. Infine, lo strumento di compilazione crea una o più build binarie e altri artefatti, come immagini, fogli di stile e documenti per l'applicazione.

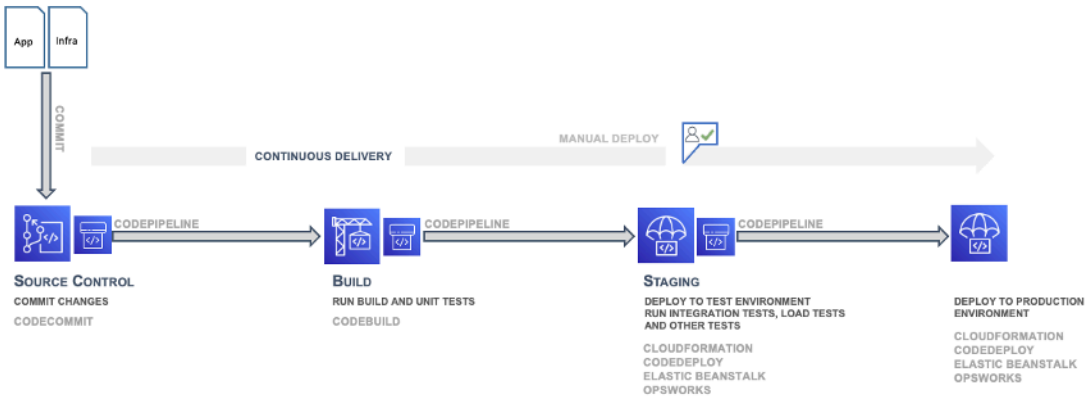
Consegna continua: creazione di un ambiente di gestione temporanea



Consegna continua: gestione temporanea

La consegna continua (CD) è la fase successiva e prevede l'implementazione del codice dell'applicazione in un ambiente di gestione temporanea, ovvero una replica dello stack di produzione, e l'esecuzione di più test funzionali. L'ambiente di gestione temporanea potrebbe essere un ambiente statico preconfigurato per i test oppure un ambiente dinamico di cui hai eseguito il provisioning e che hai configurato con un'infrastruttura con commit e il codice di configurazione per il test e l'implementazione del codice dell'applicazione.

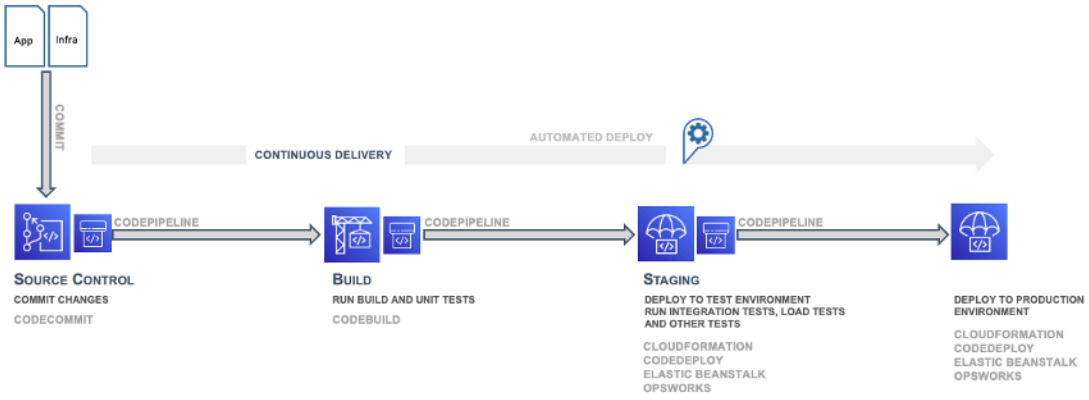
Consegna continua: creazione di un ambiente di produzione



Consegna continua - ambiente di produzione

Nella sequenza della pipeline di implementazione/consegna, dopo l'ambiente di gestione temporanea c'è l'ambiente di produzione, anch'esso creato tramite Infrastructure as Code (IaC).

Distribuzione continua



Distribuzione continua

La fase finale della pipeline di implementazione di CI/CD è l'implementazione continua, che può includere la completa automazione dell'intero processo di rilascio del software, inclusa l'implementazione nell'ambiente di produzione. In un ambiente CI/CD completamente maturo, il percorso verso l'ambiente di produzione è completamente automatizzato, il che consente di implementare il codice con elevata affidabilità.

Maturità e oltre

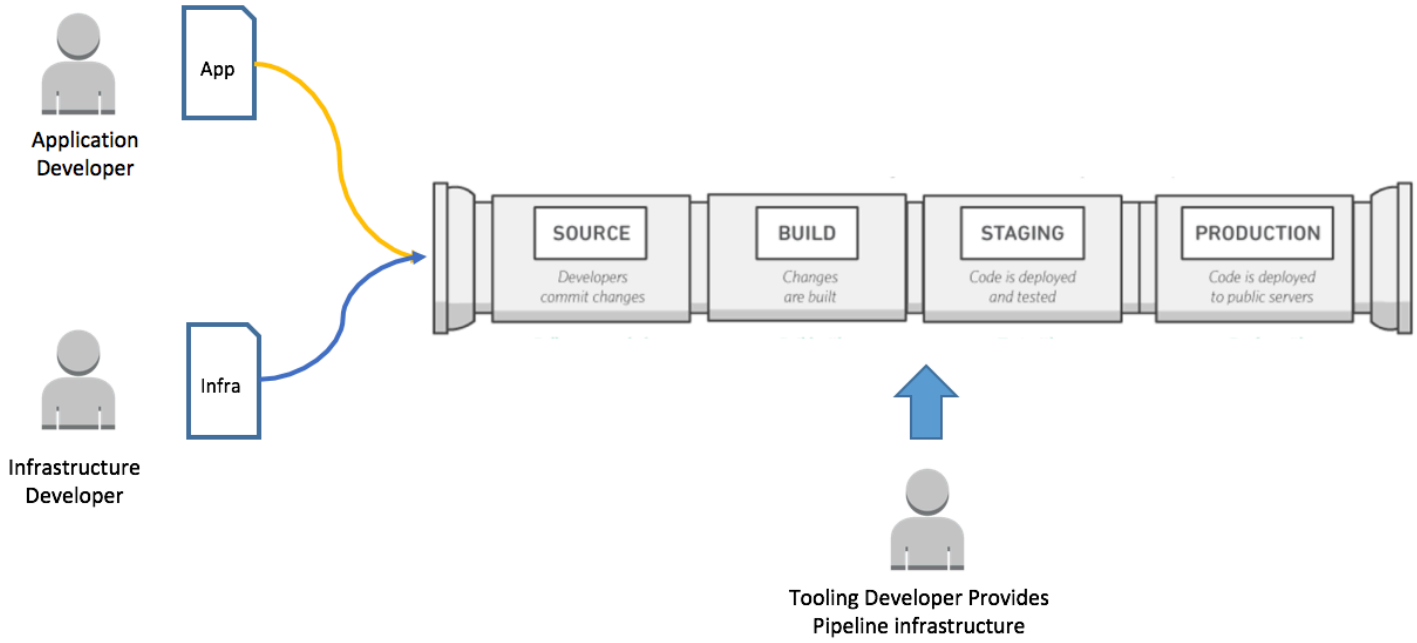
Man mano che matura, l'organizzazione continuerà a sviluppare il modello CI/CD per includere altri dei seguenti miglioramenti:

- Più ambienti di gestione temporanea per test specifici di prestazioni, conformità, sicurezza e interfaccia utente (UI)
- Unit test dell'infrastruttura e del codice di configurazione insieme al codice dell'applicazione
- Integrazione con altri sistemi e processi come la revisione del codice, il monitoraggio dei problemi e la notifica degli eventi
- Integrazione con la migrazione dello schema di database (se applicabile)
- Passaggi aggiuntivi per la verifica e l'approvazione aziendale

Anche le organizzazioni più mature che dispongono di pipeline CI/CD multi-ambiente complesse continuano a cercare miglioramenti. DevOps è un viaggio, non una destinazione. I feedback sulla pipeline vengono raccolti continuamente e i miglioramenti in termini di velocità, dimensionamento, sicurezza e affidabilità vengono raggiunti sotto forma di una collaborazione tra le diverse parti dei team di sviluppo.

Team

AWS consiglia di organizzare tre team di sviluppatori per l'implementazione di un ambiente CI/CD: un team delle applicazioni, un team dell'infrastruttura e un team degli strumenti (fai riferimento alla figura seguente). Questa organizzazione rappresenta un insieme di best practice che sono state sviluppate e applicate nelle startup in rapida evoluzione, nelle grandi organizzazioni aziendali e nella stessa Amazon. Il numero dei membri di un team non dovrebbe superare il numero di persone che possono essere sfamate da due pizze, ovvero circa 10-12 persone. Ciò segue la regola di comunicazione secondo cui le conversazioni significative raggiungono dei limiti proporzionali all'aumento delle dimensioni del gruppo e delle linee di comunicazione.



Team dell'applicazione, dell'infrastruttura e degli strumenti

Team dell'applicazione

Il team dell'applicazione si occupa della creazione dell'applicazione. Gli sviluppatori dell'applicazione sono proprietari del backlog, delle storie e degli unit test e sviluppano funzioni basate su un target di applicazione specifico. L'obiettivo organizzativo di questo team è ridurre al minimo il tempo che gli sviluppatori dedicano alle attività non principali relative all'applicazione.

Oltre ad avere competenze di programmazione funzionali nel linguaggio applicativo, il team dell'applicazione dovrebbe avere competenze relative alla piattaforma e conoscere la configurazione del sistema. In questo modo, il team potrà concentrarsi esclusivamente sullo sviluppo delle funzioni e sul rafforzamento dell'applicazione.

Team dell’infrastruttura

Il team dell'infrastruttura scrive il codice che compilerà e configurerà l'infrastruttura necessaria per eseguire l'applicazione. Questo team può utilizzare strumenti AWS nativi, ad esempio AWS CloudFormation, o strumenti generici come Chef, Puppet o Ansible. Il team dell'infrastruttura deve specificare quali risorse sono necessarie e lavora a stretto contatto con il team dell'applicazione. Il team dell'infrastruttura può essere composto da solo una o due persone, nel caso di applicazioni di piccole dimensioni.

Il team deve avere competenze relative ai metodi di provisioning dell'infrastruttura, come AWS CloudFormation o HashiCorp Terraform. Il team dovrebbe inoltre sviluppare competenze di automazione della configurazione con strumenti come Chef, Ansible, Puppet o Salt.

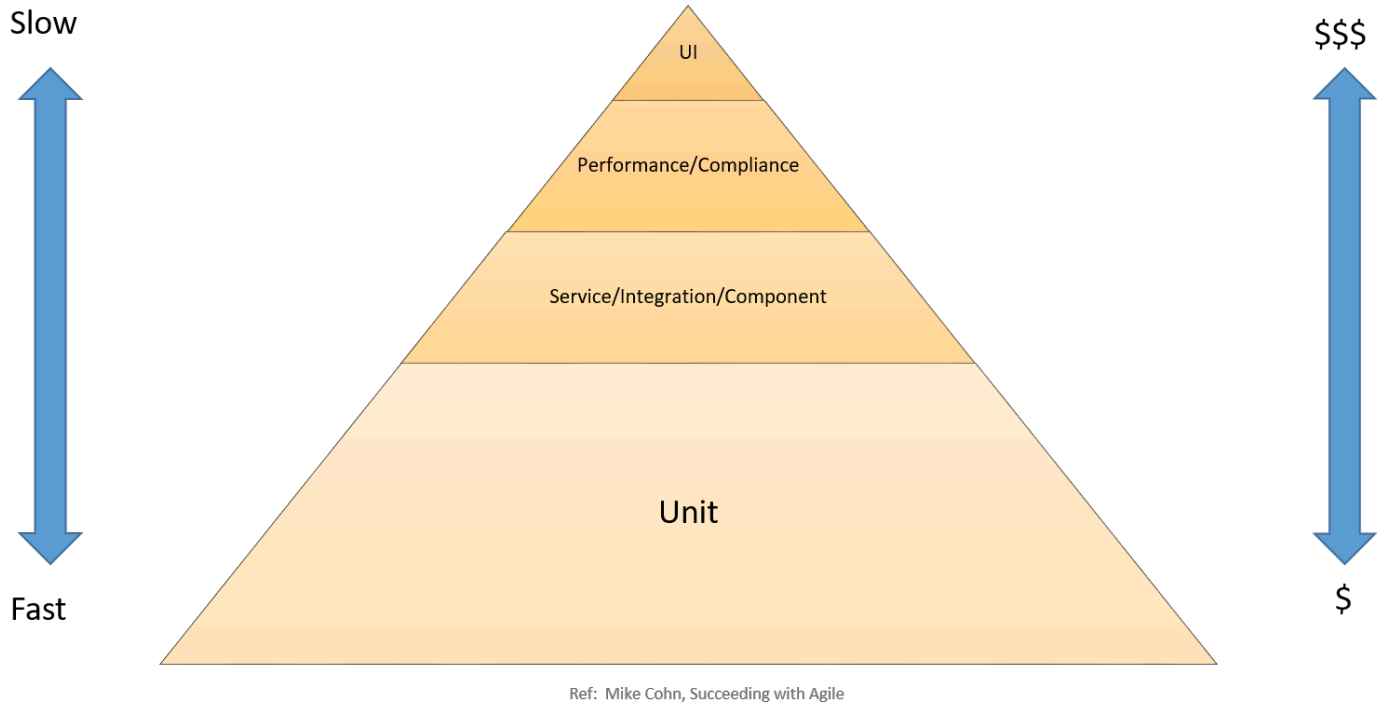
Team degli strumenti

Il team degli strumenti crea e gestisce la pipeline CI/CD. È responsabile dell'infrastruttura e degli strumenti che compongono la pipeline. I membri di questo team non rientrano nel concetto del two-pizza team; tuttavia, creano uno strumento che viene utilizzato dai team dell'applicazione e dell'infrastruttura dell'organizzazione. L'organizzazione deve far crescere continuamente il proprio team degli strumenti, in modo che sia sempre un passo avanti rispetto ai team dell'applicazione e dell'infrastruttura in fase di crescita.

Il team degli strumenti deve essere esperto nella compilazione e nell'integrazione di tutte le parti della pipeline CI/CD. Ciò include la creazione di repository di controllo del codice sorgente, motori del flusso di lavoro, ambienti di compilazione, framework di test e repository di artefatti. Questo team può scegliere di implementare software come AWS CodeStar, AWS CodePipeline, AWS CodeCommit, AWS CodeDeploy, AWS CodeBuild e AWS CodeArtifact, insieme a Jenkins, GitHub, Artifactory, TeamCity e altri strumenti simili. Alcune organizzazioni potrebbero chiamarlo team DevOps, ma AWS sconsiglia di utilizzare questa nomenclatura e incoraggia invece a pensare a DevOps come all'insieme delle persone, dei processi e degli strumenti che compongono la consegna del software.

Fasi di test nei processi di integrazione e consegna continue

I tre team CI/CD devono incorporare i test nel ciclo di vita dello sviluppo del software in diverse fasi della pipeline CI/CD. Nel complesso, i test devono iniziare il prima possibile. La seguente piramide di test è un concetto illustrato da Mike Cohn nel libro Succeeding with Agile. La piramide mostra i vari test del software in relazione al loro costo e alla velocità con cui vengono eseguiti.



Piramide di test CI/CD

Gli unit test si trovano alla base della piramide. Sono i più veloci da eseguire e i meno costosi. Pertanto, gli unit test dovrebbero costituire la parte più importante della strategia di test. Per buona regola generale, dovrebbero corrispondere a circa il 70 percento della piramide di test. Gli unit test devono coprire quasi completamente il codice, poiché i bug rilevati in questa fase possono essere risolti in modo rapido ed economicamente vantaggioso.

I test del servizio, dei componenti e dell'integrazione si trovano ai livelli superiori della piramide rispetto agli unit test. Questi test richiedono ambienti dettagliati e, pertanto, sono più costosi in termini di requisiti di infrastruttura e prevedono un'esecuzione più lenta. I test delle prestazioni e della conformità si trovano al livello successivo. Richiedono ambienti di qualità di produzione e sono ancora più costosi. I test dell'interfaccia utente e di accettazione utente sono in cima alla piramide e richiedono ambienti di qualità di produzione.

Tutti questi test fanno parte di una strategia di test completa per garantire un software di alta qualità. Tuttavia, per la velocità del processo di sviluppo, l'accento è posto sul numero di test e sulla copertura nella metà inferiore della piramide.

Nelle sezioni seguenti sono descritte le fasi di CI/CD.

Configurazione dell'origine

All'inizio del progetto, è essenziale configurare una fonte in cui archiviare il codice non elaborato e le modifiche alla configurazione e allo schema. Nella fase di origine, scegli un repository del codice sorgente, ad esempio uno ospitato in GitHub o AWS CodeCommit.

Configurazione ed esecuzione delle build

L'automazione delle build è un passaggio essenziale del processo CI. Quando si configura l'automazione delle build, occorre innanzitutto scegliere lo strumento di compilazione giusto. Sono disponibili molti strumenti di compilazione, come ad esempio:

- Ant, Maven e Gradle per Java
- Make per C/C++
- Grunt per JavaScript
- Rake per Ruby

Lo strumento di compilazione più adatto alle tue esigenze dipende dal linguaggio di programmazione del progetto e dalle competenze del team. Dopo aver scelto lo strumento di compilazione, tutte le dipendenze devono essere chiaramente definite negli script di compilazione, insieme ai passaggi di compilazione. Questa è inoltre una best practice per il controllo delle versioni degli artefatti della build finali, che semplifica l'implementazione e il monitoraggio dei problemi.

Compilazione

Nella fase di compilazione, gli strumenti di compilazione trattano come input qualsiasi modifica al repository del codice sorgente, compilano il software ed eseguono i seguenti tipi di test:

Unit test: consentono di testare una sezione specifica di codice per assicurarsi che il codice si comporti come previsto. Gli unit test vengono eseguiti dagli sviluppatori del software durante la fase di sviluppo. In questa fase, è possibile applicare processi di analisi del codice statico, di analisi del flusso di dati, di copertura del codice e altri processi di verifica del software.

Analisi del codice statico: questo test viene svolto senza eseguire effettivamente l'applicazione dopo gli unit test e i test di compilazione. Questa analisi può aiutare a rilevare errori di codifica e falle di sicurezza e può inoltre garantire la conformità alle linee guida di codifica.

Gestione temporanea

Nella fase di gestione temporanea, vengono creati ambienti completi che rispecchiano il futuro ambiente di produzione. Vengono eseguiti i seguenti test:

Test di integrazione: consentono di verificare le interfacce tra i componenti rispetto alla progettazione del software. Il test di integrazione è un processo iterativo e facilita la creazione di interfacce solide e dell'integrità del sistema.

Test dei componenti: consentono di testare la trasmissione dei messaggi tra i vari componenti e i relativi risultati. Un obiettivo chiave di questi test potrebbe essere l'idempotenza nel test dei componenti. I test possono includere volumi di dati estremamente elevati o situazioni perimetrali e input anomali.

Test di sistema: consentono di testare il sistema in modo completo e di verificare se il software soddisfa i requisiti aziendali. Questi test potrebbero includere il test dell'interfaccia utente (UI), dell'API, della logica di back-end e dello stato finale.

Test delle prestazioni: consentono di determinare la reattività e la stabilità di un sistema durante il funzionamento con un particolare carico di lavoro. I test delle prestazioni vengono inoltre utilizzati per indagare, misurare, convalidare o verificare altri attributi di qualità del sistema, come scalabilità, affidabilità e utilizzo delle risorse. I tipi di test delle prestazioni possono includere test di carico, test di stress e spike test. I test delle prestazioni vengono utilizzati per il benchmark rispetto ai criteri predefiniti.

Test di conformità: consentono di verificare se la modifica del codice è conforme ai requisiti di una specifica e/o di una normativa non funzionale. Determinano se l'implementazione sta avvenendo nel rispetto degli standard definiti.

Test di accettazione dell'utente: consentono di convalidare il flusso aziendale completo. Questi test vengono eseguiti da un utente finale in un ambiente di gestione temporanea e consentono di verificare se il sistema soddisfa i requisiti della specifica dei requisiti. In genere, i clienti utilizzano metodologie di test alfa e beta in questa fase.

Produzione

Infine, dopo aver superato i test precedenti, la fase di gestione temporanea viene ripetuta in un ambiente di produzione. In questa fase, un test canary finale può essere completato implementando il nuovo codice solo su un piccolo sottoinsieme di server o anche su un solo server, o in una Regione

AWS, prima di implementare il codice nell'intero ambiente di produzione. Le specifiche su come eseguire l'implementazione in modo sicuro nell'ambiente di produzione sono illustrate nella sezione Metodi di implementazione.

Nella sezione successiva viene illustrata la procedura di creazione della pipeline per incorporare le fasi e i test descritti sopra.

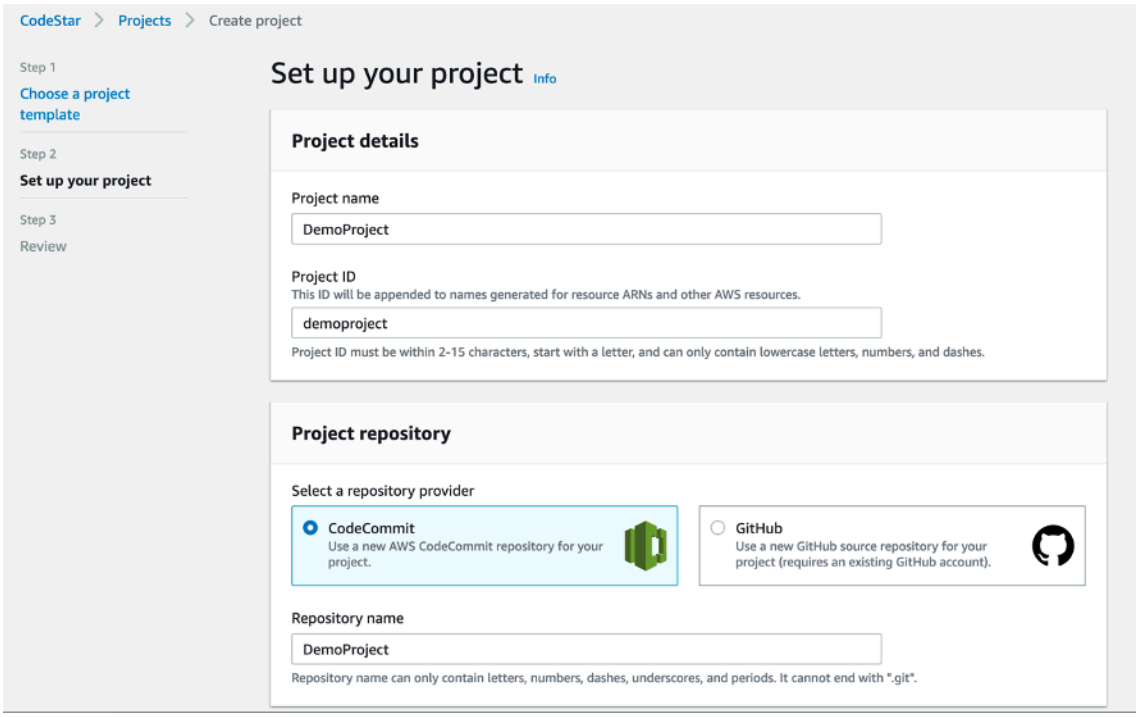
Creazione della pipeline

In questa sezione viene illustrata la procedura di creazione della pipeline. Inizia definendo una pipeline con solo i componenti necessari per il processo di CI e quindi passa successivamente a una pipeline di consegna continua con più componenti e fasi. Questa sezione illustra anche come prendere in considerazione l'utilizzo delle funzioni e delle approvazioni manuali di AWS Lambda per i progetti di grandi dimensioni e come pianificare per più team, branch e Regioni AWS.

Iniziare da una pipeline minima funzionante per l'integrazione continua

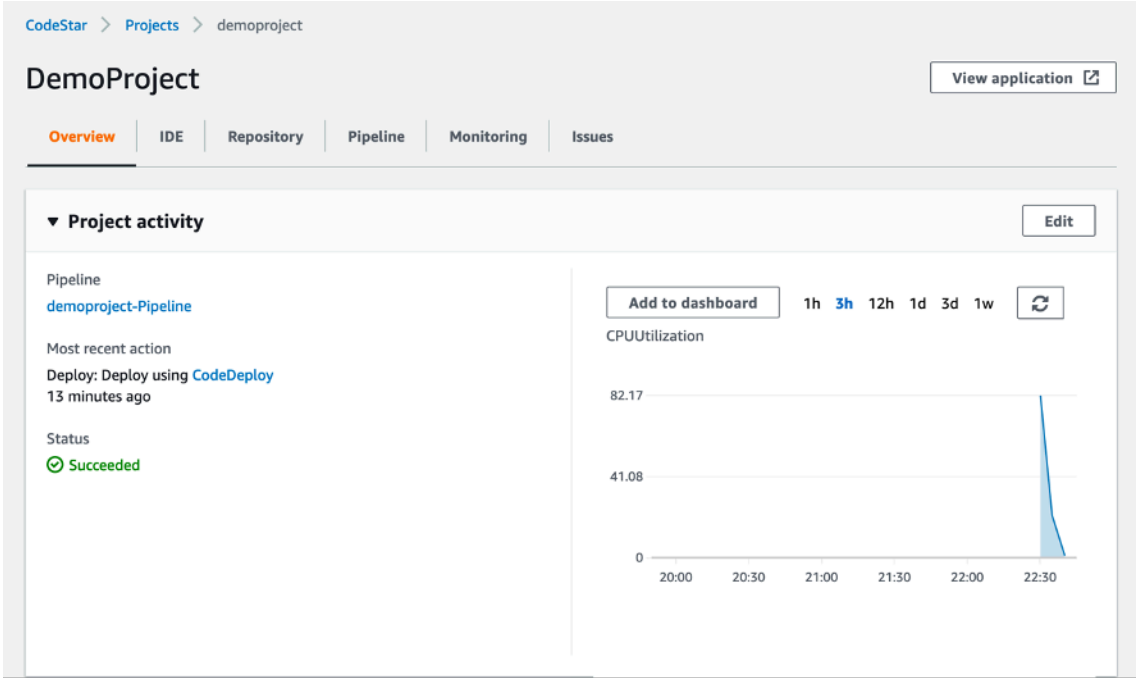
Il percorso dell'organizzazione verso la consegna continua inizia con una pipeline minima funzionante (MVP, Minimun Viable Pipeline). Come discusso nella sezione Implementazione dell'integrazione e della consegna continue, i team possono iniziare con un processo molto semplice, come l'implementazione di una pipeline che esegue un controllo dello stile del codice o un singolo unit test senza implementazione.

Un componente chiave è lo strumento di orchestrazione della consegna continua. Per aiutarti a creare questa pipeline, Amazon ha sviluppato AWS CodeStar.



Pagina di configurazione di AWS CodeStar

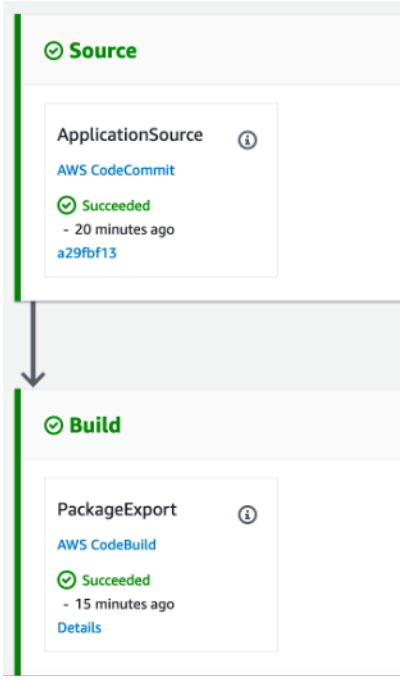
AWS CodeStar utilizza AWS CodePipeline, AWS CodeBuild, AWS CodeCommit e AWS CodeDeploy con un processo di configurazione, strumenti, modelli e pannello di controllo integrati. AWS CodeStar ti fornisce tutto ciò che ti serve per iniziare rapidamente a sviluppare, creare e distribuire applicazioni in AWS. In questo modo, puoi iniziare a rilasciare il codice più velocemente. I clienti che hanno già familiarità con AWS Management Console e cercano un livello di controllo più elevato possono configurare manualmente i loro strumenti di sviluppo preferiti ed effettuare il provisioning dei singoli servizi AWS in base alle loro esigenze.



Pannello di controllo di AWS CodeStar

AWS CodePipeline è un servizio di CI/CD che può essere utilizzato tramite AWS CodeStar o tramite la AWS Management Console per aggiornamenti rapidi e affidabili dell'applicazione e dell'infrastruttura. AWS CodePipeline compila, testa e implementa il codice ogni volta che questo viene modificato, in base ai modelli del processo di rilascio definiti. Questo ti permette di distribuire in modo rapido e affidabile funzionalità e aggiornamenti. Puoi creare una soluzione end-to-end con la massima semplicità impiegando i plug-in preinstallati di servizi di terze parti quali GitHub oppure integrando i tuoi plug-in personalizzati in qualsiasi fase del processo di rilascio. Con AWS CodePipeline, paghi solo per i servizi che utilizzi. Non sono previste tariffe iniziali né impegni a lungo termine.

Le fasi di AWS CodeStar e AWS CodePipeline mappano direttamente alle fasi di origine, compilazione, gestione temporanea e produzione della pipeline CI/CD. Sebbene sia auspicabile la consegna continua, è possibile iniziare con una semplice pipeline in due fasi che controlla il repository del codice sorgente ed esegue un'operazione di compilazione:

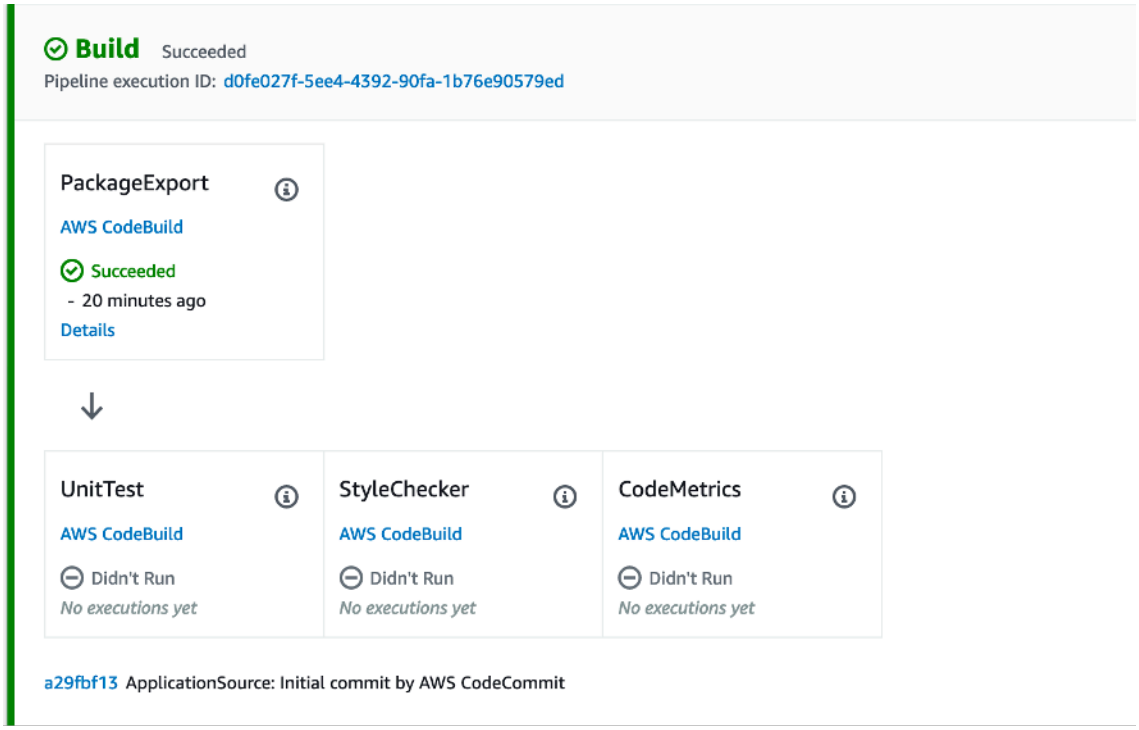


AWS CodePipeline - fasi di origine e di compilazione

Per AWS CodePipeline, la fase di origine può accettare gli input da GitHub, AWS CodeCommit e Amazon Simple Storage Service (Amazon S3). L'automazione del processo di compilazione è un primo passaggio fondamentale per l'implementazione della consegna continua e il passaggio all'implementazione continua. L'eliminazione dell'intervento umano nella produzione di artefatti di compilazione toglie un peso al tuo team, riduce al minimo gli errori introdotti dalla creazione manuale dei pacchetti e consente di iniziare a preparare gli artefatti di consumo con maggiore frequenza.

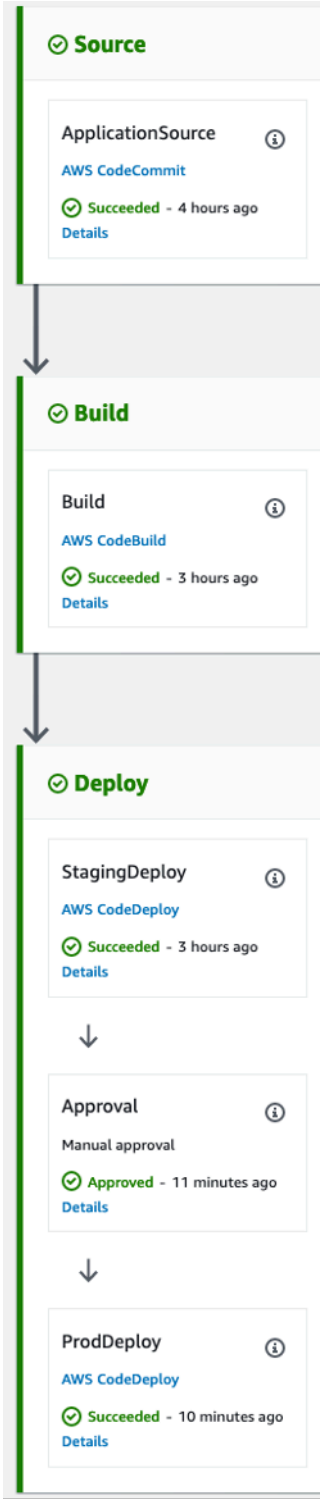
AWS CodePipeline funziona perfettamente con AWS CodeBuild, un servizio di compilazione completamente gestito, per semplificare la configurazione di una fase di compilazione all'interno della pipeline che crea pacchetti di codice ed esegue unit test. Con AWS CodeBuild, non è necessario effettuare il provisioning, gestire o dimensionare i propri server di sviluppo. AWS CodeBuild ridimensiona continuamente ed elabora più build in contemporanea in modo che le build non vengano lasciate in attesa nella coda. AWS CodePipeline si integra inoltre con server di sviluppo come Jenkins, Solano CI e TeamCity.

Ad esempio, nella successiva fase di compilazione, vengono eseguite in parallelo tre operazioni (unit test, controlli dello stile del codice e raccolta dei parametri del codice). Con AWS CodeBuild, questi passaggi possono essere aggiunti come nuovi progetti senza ulteriori sforzi durante la compilazione o l'installazione dei server di sviluppo per la gestione del carico.



AWS CodePipeline - funzionalità di compilazione

Le fasi di origine e di compilazione mostrate nella figura AWS CodePipeline - fasi di origine e di compilazione, insieme ai processi e all'automazione sottostanti, supportano la transizione del team verso l'integrazione continua. A questo livello di maturità, gli sviluppatori devono prestare regolarmente attenzione ai risultati della compilazione e dei test. Devono anche creare e mantenere una base di unit test integra. Ciò, a sua volta, rafforza la fiducia dell'intero team nella pipeline CI/CD e ne promuove l'adozione.



Fasi di AWS CodePipeline

Pipeline di consegna continua

Dopo che la pipeline di integrazione continua è stata implementata e sono stati stabiliti i processi sottostanti, i team possono iniziare la transizione verso la pipeline di consegna continua. Questa transizione prevede l'automatizzazione da parte dei team sia della creazione che dell'implementazione delle applicazioni.

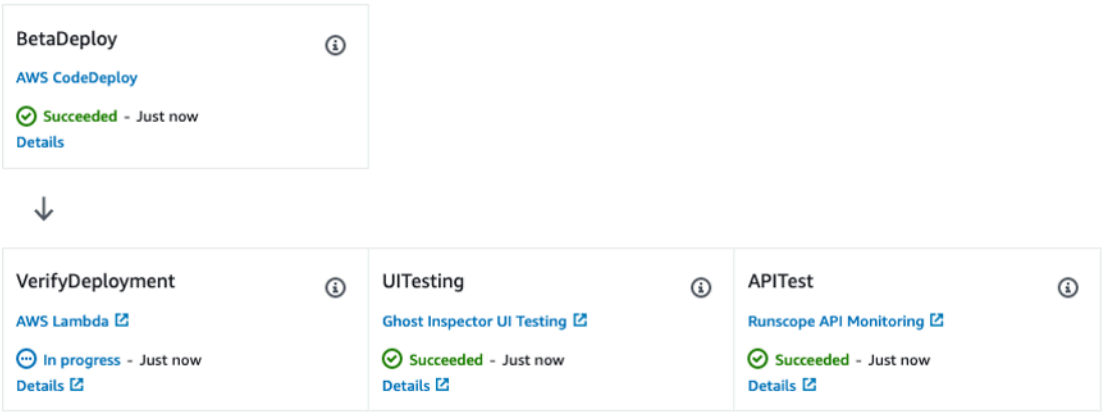
Una pipeline di consegna continua è caratterizzata dalla presenza delle fasi di gestione temporanea e di produzione, dove la fase di produzione viene eseguita dopo un'approvazione manuale.

Così come hanno creato la pipeline di integrazione continua, i team possono iniziare a creare gradualmente una pipeline di consegna continua scrivendo i propri script di implementazione.

A seconda delle esigenze dell'applicazione, è possibile estrarre alcuni passaggi di implementazione dai servizi AWS esistenti. Ad esempio, AWS CodePipeline si integra direttamente con AWS CodeDeploy, un servizio che automatizza le implementazioni di codice nelle istanze Amazon EC2 e nelle istanze in esecuzione on-premise, con AWS OpsWorks, un servizio di gestione della configurazione che consente di utilizzare le applicazioni tramite Chef, e con AWS Elastic Beanstalk, un servizio per l'implementazione e il dimensionamento di applicazioni e servizi Web.

AWS dispone di una documentazione dettagliata su come implementare e integrare AWS CodeDeploy con l'infrastruttura e la pipeline.

Dopo che il team ha automatizzato correttamente l'implementazione dell'applicazione, le fasi di implementazione possono essere ampliate con vari test. Ad esempio, puoi aggiungere altre integrazioni pronte all'uso con servizi come Ghost Inspector, Runscope e altri, come mostrato nella figura seguente.



AWS CodePipeline - test del codice nelle fasi di implementazione

Aggiunta di operazioni Lambda

AWS CodeStar e AWS CodePipeline supportano l'integrazione con AWS Lambda. Questa integrazione consente di implementare un'ampia serie di attività, come la creazione di risorse personalizzate nell'ambiente, l'integrazione con sistemi di terze parti (come Slack) e l'esecuzione di controlli sull'ambiente appena implementato.

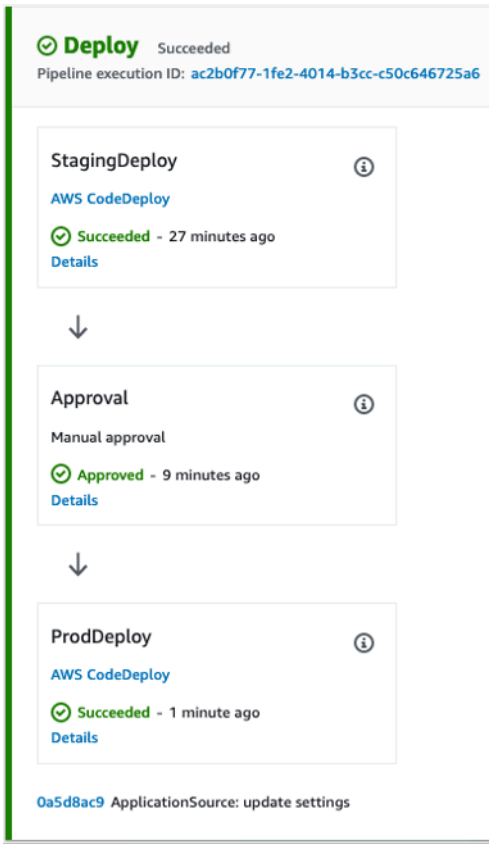
Le funzioni Lambda possono essere utilizzate nelle pipeline CI/CD per eseguire le seguenti attività:

- Implementare modifiche all'ambiente con l'applicazione o l'aggiornamento di un modello di AWS CloudFormation.
- Creare risorse on demand in una fase della pipeline utilizzando AWS CloudFormation ed eliminarle in un'altra fase.
- Implementare le versioni dell'applicazione in AWS Elastic Beanstalk senza tempi di inattività con una funzione Lambda che scambia i valori del record di nome canonico (CNAME).
- Eseguire l'implementazione su istanze Docker di Amazon Elastic Container Service (ECS).
- Eseguire il backup delle risorse prima della compilazione o dell'implementazione mediante la creazione di uno snapshot AMI.
- Aggiungere l'integrazione con i prodotti di terze parti alla pipeline, ad esempio l'invio di messaggi a un client Internet Relay Chat (IRC).

Approvazioni manuali

Aggiungi un'operazione di approvazione a una fase in una pipeline nel punto in cui desideri che l'esecuzione della pipeline venga interrotta per consentire a un utente che dispone delle autorizzazioni AWS Identity and Access Management (IAM) richieste di approvare o rifiutare l'operazione.

Se l'operazione viene approvata, l'elaborazione della pipeline riprende. Se l'operazione viene rifiutata, o se nessuno approva o rifiuta l'operazione entro sette giorni da quando la pipeline raggiunge l'operazione e l'interruzione, il risultato è identico a un'operazione non riuscita e l'elaborazione della pipeline non continua.



AWS CodeDeploy - approvazioni manuali

Implementazione delle modifiche al codice dell'infrastruttura in una pipeline CI/CD

AWS CodePipeline consente di selezionare AWS CloudFormation come operazione di implementazione in qualsiasi fase della pipeline. Puoi quindi scegliere l'operazione specifica che desideri venga eseguita da AWS CloudFormation, ad esempio la creazione o l'eliminazione di stack e la creazione o l'esecuzione dei set di modifiche. Uno stack è un concetto di AWS CloudFormation e rappresenta un gruppo di risorse AWS correlate. Sebbene ci siano molti modi per effettuare il provisioning di Infrastructure as Code, AWS CloudFormation è uno strumento completo consigliato da AWS come soluzione scalabile e completa in grado di descrivere il set più completo di risorse AWS come codice. AWS consiglia di utilizzare AWS CloudFormation in un progetto di AWS CodePipeline per tenere traccia delle modifiche e dei test dell'infrastruttura.

CI/CD per applicazioni serverless

È inoltre possibile utilizzare AWS CodeStar, AWS CodePipeline, AWS CodeBuild e AWS CloudFormation per creare pipeline CI/CD per applicazioni serverless. Le applicazioni serverless

integrano servizi gestiti come Amazon Cognito, Amazon S3 e Amazon DynamoDB con i servizi basati su eventi e AWS Lambda per implementare le applicazioni in un modo che non richiede la gestione dei server. Se sei uno sviluppatore di applicazioni serverless, puoi utilizzare la combinazione di AWS CodePipeline, AWS CodeBuild e AWS CloudFormation per automatizzare le fasi di compilazione, test e implementazione delle applicazioni serverless espresse in modelli creati con AWS Serverless Application Model. Per ulteriori informazioni, fai riferimento alla documentazione su AWS Lambda relativa all'automazione dell'implementazione delle applicazioni basate su Lambda.

Puoi inoltre creare pipeline CI/CD sicure che seguono le best practice della tua organizzazione con le pipeline AWS Serverless Application Model (Pipeline AWS SAM). Le pipeline AWS SAM sono una nuova funzione della CLI di AWS SAM che consente di usufruire in pochi minuti dei vantaggi offerti da CI/CD, come l'accelerazione della frequenza di implementazione, la riduzione dei tempi di applicazione delle modifiche e la riduzione degli errori di implementazione. Le pipeline AWS SAM includono una serie di modelli di pipeline predefiniti per AWS CodeBuild/CodePipeline basati sulle best practice di implementazione di AWS. Per ulteriori informazioni e per guardare il tutorial, consulta il blog Introducing AWS SAM Pipelines.

Pipeline per più team, branch e regioni AWS

Per un progetto di grandi dimensioni, non è raro che più team di progetto lavorino su componenti diversi. Se più team utilizzano un unico repository di codice, questo può essere mappato in modo che ogni team abbia il proprio branch. Ci dovrebbe essere anche un branch di integrazione o di rilascio per l'unione finale del progetto. Se viene utilizzata un'architettura orientata ai servizi o ai microservizi, ogni team potrebbe avere il proprio repository di codice.

Nel primo scenario, se viene utilizzata una singola pipeline, è possibile che un team possa influenzare l'avanzamento degli altri team bloccando la pipeline. AWS consiglia di creare pipeline specifiche per i branch dei team e un'altra pipeline di rilascio per la consegna finale del prodotto.

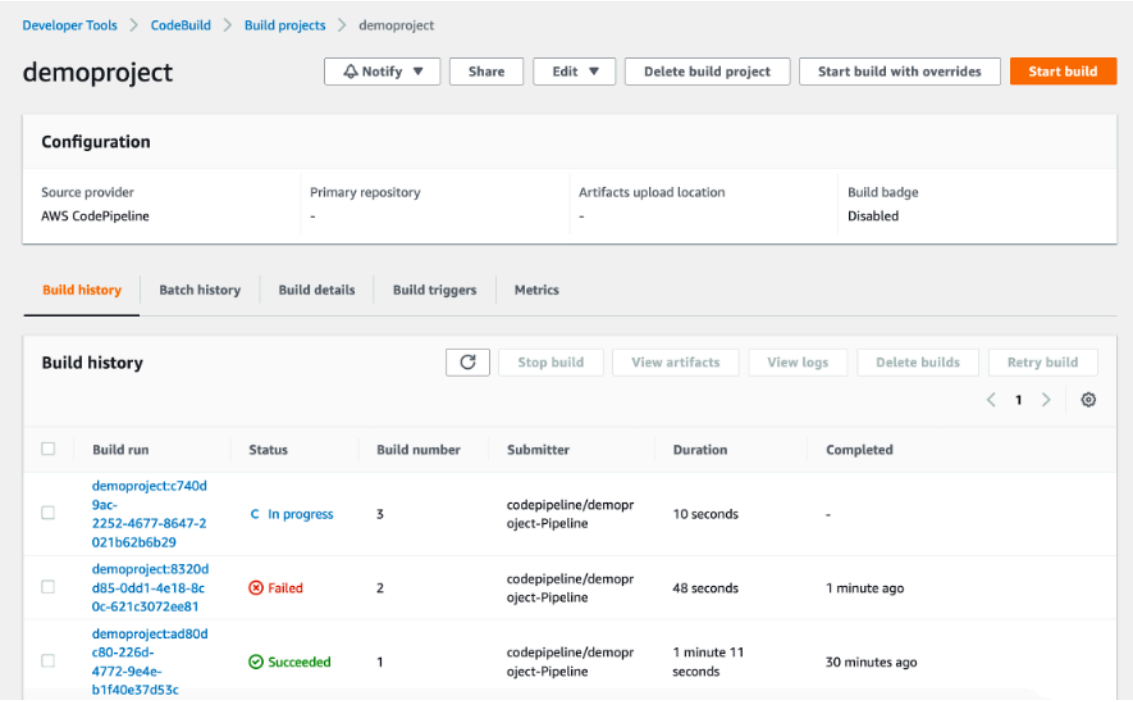
Integrazione della pipeline con AWS CodeBuild

AWS CodeBuild è progettato per consentire alla tua organizzazione di creare un processo di compilazione ad alta disponibilità con scalabilità quasi illimitata. AWS CodeBuild fornisce ambienti di avvio rapido per parecchi dei linguaggi di programmazione più diffusi, oltre alla possibilità di eseguire un qualsiasi container Docker specificato.

Con i vantaggi della stretta integrazione con AWS CodeCommit, AWS CodePipeline e AWS CodeDeploy, così come grazie alle operazioni Lambda di Git e CodePipeline, lo strumento CodeBuild è altamente flessibile.

Il software può essere compilato mediante l'inclusione di un file `buildspec.yml` che identifica ciascuna delle fasi di compilazione, incluse le operazioni precedenti e successive alla compilazione o le operazioni specifiche, tramite lo strumento CodeBuild.

Puoi visualizzare la cronologia dettagliata di ogni build sul pannello di controllo di CodeBuild. Gli eventi vengono archiviati come file di log di Amazon CloudWatch Logs.



File di log di CloudWatch Logs in AWS CodeBuild

Integrazione della pipeline con Jenkins

Puoi utilizzare lo strumento di compilazione di Jenkins per creare pipeline di consegna. Queste pipeline utilizzano processi standard che definiscono i passaggi per l'implementazione delle fasi di consegna continua. Tuttavia, questo approccio potrebbe non essere ottimale per progetti di maggiori dimensioni perché lo stato attuale della pipeline non persiste tra i diversi riavvii di Jenkins, l'implementazione dell'approvazione manuale non è semplice e il monitoraggio dello stato di una pipeline complessa può essere complicato.

AWS consiglia invece di implementare la consegna continua con Jenkins utilizzando il plug-in AWS Code Pipeline. Questo plug-in consente di descrivere flussi di lavoro complessi utilizzando un linguaggio specifico del dominio simile a Groovy e può essere utilizzato per orchestrare pipeline complesse. La funzionalità del plug-in AWS Code Pipeline può essere migliorata tramite plug-in satelliti come il plug-in Pipeline Stage View, che consente di visualizzare l'avanzamento delle fasi

definite in una pipeline, o il plug-in Pipeline Multibranch, che raggruppa le build provenienti da branch diversi.

AWS consiglia di archiviare la configurazione della pipeline in Jenkinsfile e di inserirla inoltre in un repository del codice sorgente. In questo modo, potrai tenere traccia delle modifiche al codice della pipeline, soprattutto quando utilizzi il plug-in Pipeline Multibranch. AWS consiglia inoltre di dividere la pipeline in fasi per raggruppare logicamente le fasi della pipeline e fare inoltre in modo che il plug-in Pipeline Stage View visualizzi lo stato corrente della pipeline.

La figura seguente mostra un esempio di pipeline Jenkins, con quattro fasi definite visualizzate dal plug-in Pipeline Stage View.

Fasi definite della pipeline Jenkins visualizzate dal plug-in Pipeline Stage View

Metodi di distribuzione

È possibile prendere in considerazione diverse strategie e varianti per l'implementazione di nuove versioni del software in un processo di consegna continua. Questa sezione illustra i metodi di implementazione più comuni: tutto contemporaneamente (implementazione in loco), in sequenza, immutabile e blu/verde. AWS indica quali di questi metodi sono supportati da AWS CodeDeploy e AWS Elastic Beanstalk.

Nella tabella seguente sono riepilogate le caratteristiche di ciascun metodo di implementazione.

Metodo	Impatto di una distribuzione non riuscita	Orario di distribuzione	Nessun tempo di inattività	Nessuna modifica DNS	Processo di rollback	Codice distribuito in
Implementazione in loco	Tempo di inattività	🕒	X	✓	Ridistribuisci	Istanze esistenti
In sequenza	Singolo batch fuori servizio. Qualsiasi batch completato con successo prima del fallimento dell'esecuzione di una nuova versione dell'applicazione.	🕒 🕒 †	✓	✓	Ridistribuisci	Istanze esistenti

Metodo	Impatto di una distribuzione non riuscita	Orario di distribuzione	Nessun tempo di inattività	Nessuna modifica DNS	Processo di rollback	Codice distribuito in
In sequenza con batch aggiuntivi (beanstalk)	Minimo se il primo batch ha esito negativo; altrimenti simile a quello dell'implementazione in sequenza.	🕒 🕒 🕒 †	✓	✓	Ridistribuisci	Istanze nuove ed esistenti
Non modificabile	Minima	🕒 🕒 🕒 🕒	✓	✓	Ridistribuisci	Nuove istanze
Suddivisione del traffico	Minima	🕒 🕒 🕒 🕒	✓	✓	Reindirizza il traffico e termina le nuove istanze	Nuove istanze
Blu/verde	Minima	🕒 🕒 🕒 🕒	✓	X	Ritorna all'ambiente precedente	Nuove istanze

Tutto contemporaneamente (implementazione in loco)

L'implementazione Tutto contemporaneamente (in loco) è un metodo che puoi utilizzare per implementare il nuovo codice dell'applicazione in una flotta di server esistente. Questo metodo sostituisce tutto il codice in un'unica operazione di implementazione. Prevede tempi di inattività perché tutti i server della flotta vengono aggiornati contemporaneamente. Non è necessario aggiornare i registri DNS esistenti. Se l'implementazione non riesce, l'unico modo per ripristinare le operazioni è ripetere l'implementazione del codice su tutti i server.

In AWS Elastic Beanstalk, questo metodo di implementazione è denominato Tutto contemporaneamente ed è disponibile per applicazioni singole e con bilanciamento del carico. In AWS CodeDeploy, questo metodo di implementazione è denominato Implementazione in loco con una configurazione di implementazione di `AllAtOnce`.

Implementazione in sequenza

Con l'implementazione in sequenza, la flotta è suddivisa in porzioni in modo da non doverla aggiornare in contemporanea. Durante il processo di implementazione, due versioni software, la nuova e la vecchia, sono in esecuzione sulla stessa flotta. Questo metodo consente di applicare l'aggiornamento senza tempi di inattività. Se l'implementazione non riesce, sarà interessata solo la porzione aggiornata della flotta.

Una variante del metodo di implementazione in sequenza, chiamata rilascio canary, comporta l'implementazione della nuova versione del software inizialmente su una porzione molto ridotta di server. In questo modo, è possibile osservare come si comporta il software nell'ambiente di produzione su pochi server, riducendo al minimo l'impatto delle modifiche che causano interruzioni. In caso di un tasso elevato di errori risultanti da una distribuzione canary, il software viene ripristinato allo stato precedente. In caso contrario, la percentuale di server con la nuova versione viene gradualmente aumentata.

AWS Elastic Beanstalk ha seguito il modello di implementazione in sequenza con due opzioni di implementazione: in sequenza e in sequenza con batch aggiuntivi. Queste opzioni consentono all'applicazione di dimensionare verso l'alto prima di mettere fuori servizio i server, preservando così la piena capacità durante l'implementazione. AWS CodeDeploy applica questo pattern come una variazione dell'implementazione in loco con pattern come OneAtATime e HalfAtATime.

Implementazione immutabile e blu/verde

Il pattern immutabile specifica l'implementazione del codice dell'applicazione tramite l'avvio di un set completamente nuovo di server con una nuova configurazione o versione del codice dell'applicazione. Questo pattern sfrutta la funzionalità cloud in base a cui le nuove risorse del server vengono create con semplici chiamate API.

La strategia di implementazione blu/verde è un tipo di implementazione immutabile che richiede anche la creazione di un altro ambiente. Una volta che il nuovo ambiente è attivo e ha superato tutti i test, il traffico viene spostato su questa nuova implementazione. Fondamentalmente, il vecchio ambiente, ovvero l'ambiente "blu", viene mantenuto inattivo nel caso in cui sia necessario un ripristino dello stato precedente.

AWS Elastic Beanstalk supporta i pattern di implementazione immutabile e blu/verdi. Anche AWS CodeDeploy supporta il pattern blu/verde. Per ulteriori informazioni su come i servizi AWS applicano questi pattern immutabili, consulta il whitepaper Implementazioni blu/verdi su AWS.

Modifiche dello schema di database

È normale che il software moderno abbia un livello di database. In genere, viene utilizzato un database relazionale, che memorizza sia i dati che la struttura dei dati. Spesso è necessario modificare il database nel processo di consegna continua. La gestione delle modifiche in un database relazionale richiede un'attenzione speciale e presenta altre sfide rispetto a quelle che si verificano durante l'implementazione dei file binari delle applicazioni. Di solito, quando si aggiorna un file binario di un'applicazione, si arresta l'applicazione, la si aggiorna e quindi la si riavvia. Non devi preoccuparti dello stato dell'applicazione, che viene gestito al di fuori di quest'ultima.

Quando si aggiornano i database, è necessario tenere in considerazione lo stato, perché un database è composto per la maggior parte dallo stato e solo in minima parte dalla logica e dalla struttura.

Gli schemi del database prima e dopo l'applicazione di una modifica devono essere considerati come versioni diverse del database. È possibile utilizzare strumenti come Liquibase e Flyway per gestire le versioni.

In generale, questi strumenti impiegano alcune varianti dei seguenti metodi:

- Aggiungi una tabella al database in cui è archiviata una versione del database.
- Tieni traccia dei comandi di modifica del database e raggruppali in set di modifiche con versione. Nel caso di Liquibase, queste modifiche vengono archiviate in file XML. Flyway utilizza un metodo leggermente diverso in cui i set di modifiche vengono gestiti come file SQL separati oppure occasionalmente come classi Java separate per transizioni più complesse.
- Quando viene richiesto a Liquibase di aggiornare un database, questo strumento esamina la tabella dei metadati e determina quali set di modifiche eseguire per aggiornare il database con la versione più recente.

Riepilogo delle best practice

Di seguito sono riportate alcune best practice sulle cose da fare e da non fare per CI/CD.

Cose da fare:

- Trattare l'infrastruttura come codice
  - Usare il controllo delle versioni per il codice dell'infrastruttura.
  - Utilizzare sistemi di tracciatura/creazione dei ticket sui bug.
  - Chiedere ai colleghi di rivedere le modifiche prima di applicarle.
  - Stabilire i pattern/progetti del codice dell'infrastruttura.
  - Testare le modifiche all'infrastruttura come le modifiche al codice.
- Creare team integrati di sviluppatori composti da non più di 12 membri autonomi.
- Chiedere a tutti gli sviluppatori di eseguire frequentemente il commit del codice nel trunk principale, senza feature branch di lunga durata.
- Adottare in modo coerente un sistema di compilazione come Maven o Gradle in tutta l'organizzazione e standardizzare le build.
- Chiedere agli sviluppatori di creare unit test per una copertura del 100% della base di codice.
- Assicurarsi che gli unit test rappresentino il 70% dei test complessivi in termini di durata, numero e ambito.
- Assicurarsi che gli unit test siano aggiornati e non trascurati. Gli errori che emergono dagli unit test devono essere corretti e non aggirati.
- Considerare la configurazione della consegna continua come codice.
- Stabilire controlli di sicurezza basati sui ruoli (ovvero chi può fare cosa e quando).
  - Monitorare/tracciare ogni risorsa possibile.
  - Creare avvisi su servizi, disponibilità e tempi di risposta.
  - Acquisire, imparare e migliorare.
  - Condividere l'accesso con tutti i membri del team.
  - Pianificare i parametri e il monitoraggio nel ciclo di vita.
- Tenere traccia dei parametri standard.
  - Numero di build.
  - Numero di implementazioni.

  - Tempo medio necessario affinché le modifiche raggiungano l'ambiente di produzione.
  - Tempo medio dalla prima fase della pipeline a ciascuna fase.
  - Numero di modifiche che raggiungono l'ambiente di produzione.
  - Tempo medio di compilazione.
- Utilizzare pipeline differenti per ciascun branch e team.

Cose da non fare:

- Avere branch di lunga durata con merge grandi e complicati.
- Fare test manuali.
- Implementare processi di approvazione, gate, revisioni del codice e revisioni della sicurezza manuali.

Conclusione

L'integrazione e la consegna continue forniscono uno scenario ideale per i team applicativi dell'organizzazione. Gli sviluppatori inviano semplicemente il codice a un repository. Questo codice verrà integrato, testato, implementato, testato di nuovo, unito all'infrastruttura, sottoposto a revisioni di sicurezza e qualità e sarà pronto per l'implementazione con estrema sicurezza.

Quando si utilizza la CI/CD, la qualità del codice viene migliorata e gli aggiornamenti software vengono rilasciati rapidamente e con la certezza che non ci saranno cambiamenti radicali. L'impatto di qualsiasi rilascio può essere correlato ai dati provenienti dall'ambiente di produzione e delle operazioni. Può essere utilizzato anche per pianificare il ciclo successivo, una pratica DevOps fondamentale nel precorso dell'organizzazione verso il cloud.

Approfondimenti

Per ulteriori informazioni sugli argomenti trattati in questo whitepaper, consulta i seguenti whitepaper AWS:

- Panoramica delle opzioni di implementazione in AWS
- Implementazioni blu/verdi su AWS
- Configurazione della pipeline CI/CD tramite l'integrazione di Jenkins con AWS CodeBuild e AWS CodeDeploy
- Microservizi in AWS
- Docker su AWS: esecuzione dei container nel cloud

Collaboratori

Hanno contribuito alla stesura di questo documento:

- 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

Revisioni del documento

Per ricevere una notifica sugli aggiornamenti di questo whitepaper, iscriviti al feed RSS.

update-history-change	update-history-description	update-history-date
Pubblicazione iniziale	Prima pubblicazione del whitepaper	27 ottobre 2021
Pubblicazione iniziale	Prima pubblicazione del whitepaper	1 giugno 2017

Avvisi

I clienti sono responsabili della propria valutazione autonoma delle informazioni contenute in questo documento. Questo documento: (a) è solo a scopo informativo, (b) mostra le offerte e le pratiche attuali dei prodotti AWS soggette a modifiche senza preavviso, e (c) non crea alcun impegno o garanzia da parte di AWS e dei suoi affiliati, fornitori o licenziatari. I prodotti o servizi AWS sono forniti "così come sono" senza garanzie, dichiarazioni o condizioni di alcun tipo, sia esplicite che implicite. Le responsabilità e gli obblighi di AWS verso i propri clienti sono disciplinati dagli accordi AWS e il presente documento non fa parte né modifica alcun accordo tra AWS e i suoi clienti.