

FULL STACK CONTINUOUS DELIVERY SU AWS

CI/CD

Continuous Delivery



Simone Merlini | 10 Marzo 2017

Con il lancio di [CodeBuild](#) avvenuto lo scorso novembre [sul palco del re:Invent](#), AWS ha aggiunto il tassello mancante alla suite di strumenti gestione del ciclo di sviluppo del software

[Noi c'eravamo](#) e non aspettavamo altro per poterci mettere all'opera e implementare finalmente **un sistema di Continuous Delivery interamente basato su servizi gestiti da Amazon Web Services**.

Prima di CodeBuild, infatti, la suite di AWS copriva unicamente gli aspetti relativi al source control ([CodeCommit](#)), alla gestione dei rilasci ([CodeDeploy](#)) e all'orchestrazione ([CodePipeline](#)), costringendo gli sviluppatori ad utilizzare tool di terze parti (ad esempio [Jenkins](#)) per la gestione delle fasi di build e test. Tool che vanno configurati e gestiti manualmente, con tutte le problematiche che ne conseguono in termini di complessità, costi e affidabilità della soluzione.

Pensiamo infatti a quanto lo stack di Continuous Delivery da un lato sia critico (un inconveniente a questi tool può compromettere ore o giorni di lavoro di un intero team di sviluppo), ma dall'altro rappresenti, specie per team DevOps piccoli e agili, un oggetto estraneo alle attività "core", su cui concentrare il minor sforzo possibile; **una sorta di *black-box* che deve semplicemente funzionare**.

Uno stack di Continuous Delivery deve:

- Essere estremamente affidabile (nessun single-point-of-failure)
- Garantire la durabilità dei dati (in particolare per quanto riguarda i repository)
- Richiedere un effort di gestione e configurazione minimo
- Integrarsi facilmente sia con gli ambienti e i tool di sviluppo che con quelli di produzione
- Poter scalare facilmente di pari passo con la crescita del team e dei progetti
- Costare poco 😊

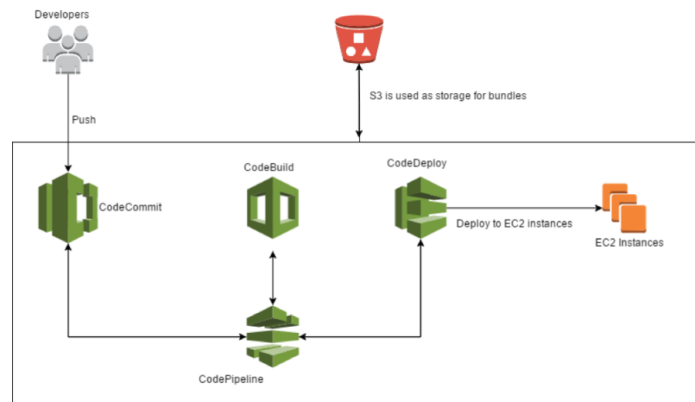
Per gli utenti AWS, l'utilizzo di uno stack di CD interamente basato sui suoi servizi managed è il modo più naturale ed immediato di rispondere a tutti questi requisiti

La soluzione

Rivediamo brevemente quali sono i tool che abbiamo a disposizione:

- **CodeCommit:** un servizio di repository git completamente gestito e scalato automaticamente
- **CodeBuild:** un sistema di build e testing basato sui container
- **CodeDeploy:** un sistema agent-based per effettuare il deploy sulle istanze EC2 in modo automatico.
- **CodePipeline:** un orchestrator completamente integrato nell'ecosistema AWS, in grado di far interagire gli altri tre servizi fra loro, oppure con provider di terze parti

Il nostro team ha realizzato una **soluzione completa e automatizzata per la gestione del software lifecycle su Amazon Web Services**, applicabile sia su ambienti ibridi (in questo caso l'integrazione con il repository e la build avviene in Cloud, mentre il deploy può svolgersi su qualsiasi tipo di istanza, sia in Cloud che on-premise), che su ambienti interamente Cloud-based, con le macchine di staging e produzione anch'esse sul Cloud.



Gli sviluppatori fanno push del codice su CodeCommit, parte un trigger e CodePipeline prende il codice e lo dà in pasto a CodeBuild che esegue la build e i test e crea l'artifact pronto per il deploy. A questo punto CodePipeline passa il pacchetto a CodeDeploy che effettua il rilascio su un pool di istanze EC2. A ogni passaggio S3 viene usato come storage di appoggio degli artifacts.

Utilizzare solo servizi AWS, compatibili e progettati appositamente per cooperare tra di loro, ha il vantaggio di poter disporre di trigger specifici e setup estremamente semplificati. **Tutto scala automaticamente e senza la necessità di interventi tecnici particolari**; il team è quindi sollevato dall'onere di dimensionare a priori l'infrastruttura di Continuous Delivery.

E' possibile, inoltre, gestire la security e i permessi in modo granulare sfruttando IAM role e IAM policy. A differenza di ciò che avviene con l'utilizzo di altri strumenti poi, è possibile rimanere confinati all'interno della sandbox di un singolo account Amazon, senza dover esporre nessun endpoint all'esterno per avviare il processo di build.

Abbiamo scoperto che CodeBuild si occupa delle fasi di build e test sfruttando il provisioning dei container; questi ultimi si avviano solo quando necessario, ottimizzando così le prestazioni ed

eliminando la necessità di istanze dedicate come avviene ad esempio durante l'utilizzo di Jenkins.

Altra caratteristica fondamentale di CodeBuild è l'isolamento, che permette di **segregare ogni build sia a livello applicativo, sia a livello delle release.**

I Costi AWS

Prima di cominciare col tutorial, buttiamo un occhio ai costi AWS generati da questa soluzione:

Il prezzo di ciascuna pipeline di CodePipeline è di 1\$/mese, mentre, per i primi 5 utenti (50GB e 10000 richieste git), il servizio CodeCommit è totalmente gratuito. Dal sesto utente in avanti il costo è di 1\$/mese a utente.

Il costo del servizio CodeBuild è calcolato in base ai minuti di utilizzo effettivo dei container che effettuano le build, modello che permette un notevole risparmio economico rispetto ai classici runner basati su istanze EC2, che sono invece tariffate su base oraria

CodeDeploy è gratuito, senza limiti di traffico o di tempo.

A questi vanno aggiunti i costi di storage e banda generati da S3, che viene utilizzato come storage di appoggio degli artifact tra uno step e l'altro di ogni pipeline.

Ovviamente i costi possono variare molto in base al contesto e all'organizzazione di ogni team, ma ci sentiamo di dire che nella maggioranza dei casi questo stack risulti **una delle soluzioni più economiche in assoluto.**

Entriamo ora nel vivo dell'implementazione .

CodeCommit

Il primo servizio da configurare è CodeCommit.

Creiamo il repository accedendo alla console AWS oppure, nel caso di accesso programmatico, alla CLI.

The screenshot shows the 'Create repository' page in the AWS CodeCommit console. At the top, there is a title 'Create repository' with a help icon. Below the title is a brief instruction: 'Create a secure repository to store and share your code. Begin by typing a repository name and a description for your repository. Repository names are included in the URLs for that repository.' A blue information box contains the text: 'Access to the repository. Users connecting to an AWS CodeCommit repository for the first time must complete setup steps before they can use it. Learn more'. Below this, there are two input fields: 'Repository name*' with the value 'MyFirstCodeCommit' and 'Description' with the value 'My first CodeCommit repo'. At the bottom, there is a '*Required' label, a 'Cancel' button, and a 'Create repository' button.

Creiamo ora un IAM user che ci permetta di procedere con le principali azioni git sul repository appena creato.

Connect to your repository

You are signed in using [federated access](#) or temporary credentials. The only supported connection method for these sign-in types is to use the credential manager included with the AWS CLI, as documented below. To configure a connection using SSH or Git credentials over HTTPS, sign in as an [IAM user](#).

Follow the steps below to connect to your repository from your local computer.

Connection type HTTPS
 SSH

Operating system Linux, MacOS, or Unix
 Windows

Prerequisites

1. Install Git (1.7.9 or later supported). If you don't have Git installed, [install it now](#).
2. Install the [AWS CLI](#).
3. At the terminal, type `aws configure` and [configure the AWS CLI with your IAM user access key and secret key](#).
4. Attach an appropriate [AWS CodeCommit managed policy](#) to the IAM user. [Learn more](#)

Steps to clone your repository

1. At the terminal, paste the following commands:

```
git config --global credential.helper 'aws codecommit credential-helper $!'
git config --global credential.UseHttpPath true
```
2. Clone your repository to your local computer and start working on code:

```
git clone https://git-codecommit.eu-west-1.amazonaws.com/v1/repos/MyFirstCodeCommit
```
3. If using MacOS, [Disable the Keychain Access utility](#) for connections to AWS CodeCommit.

[I want more detailed instructions](#)

Per ottenere i permessi di accesso al repo, occorre autenticarsi come IAM user. Ecco i possibili metodi:

- *Git credential helper*, sfrutta la CLI. Serve quindi una coppia di chiavi per l'accesso;
- *chiave SSH* che è possibile caricare dalla gestione utenti IAM;
- *username e password* generati, sempre, dalla console di gestione IAM (utilizzo su HTTPS).

A ogni push del codice sorgente dell'app, questo viene preso in carico da CodeBuild, che si occupa del provisioning di un ambiente temporaneo e isolato (container) all'interno del quale si svolgeranno le fasi di build e test.

CodeBuild

Prima di procedere alle fasi di test e build, configuriamo CodeBuild specificando il taglio del container, il tipo di immagine desiderata (ad esempio container Linux con preinstallato un framework fra quelli disponibili o un'immagine custom) e specificando i passaggi che desideriamo che CodeBuild svolga per effettuare test e build dell'app. Quest'ultimo passaggio prevede la scelta tra due metodologie: dichiarazione dei comandi inline oppure mediante un file denominato `buildspec.yml` che andrà a richiamare gli hook del ciclo di build.

Noi abbiamo scelto di utilizzare quest'ultimo metodo poiché, in questo modo, il file YAML buildspec può essere versionato insieme al codice dell'applicazione, dandoci la possibilità di cambiare le procedure di test e build appena prima della fase di build stessa.

Configure your project

Specify settings for your build project.

Project name* MyFirstCodeBuild ⓘ

Description ⓘ Add description

Source: What to build

Source provider* AWS CodeCommit

Repository* MyFirstCodeCommit

Environment: How to build

Environment image* Use an image managed by AWS CodeBuild
 Specify a Docker image

Operating system* Choose an operating system

Build specification* Use the buildspec.yml in the source code root directory
 Insert build commands

Artifacts: Where to put the artifacts from this build project

Artifacts type* Amazon S3 ⓘ

Artifacts name* MyArtifact ⓘ

Bucket name* test_bucket

Service role

Specify a service role that enables AWS CodeBuild to call dependent AWS services on your behalf. [Learn more.](#)

Create a service role in your account
 Choose an existing service role from your account

Role name* codebuild-MyFirstCodeBuild-service-role

▶ Show advanced settings

*Required

Cancel Continue

All'interno di ciascun hook, quindi, abbiamo specificato i comandi che CodeBuild dovrà eseguire, momento per momento.

Di seguito un esempio generico della struttura di un file buildspec.yml posto nella root directory del repository:

```
version: 0.1

environment_variables:
plaintext:
JAVA_HOME: "/usr/lib/jvm/java-8-openjdk-amd64"

phases:
install:
commands:
- apt-get update -y
- apt-get install -y maven
pre_build:
commands:
- echo Nothing to do in the pre_build phase...
build:
commands:
- echo Build started on `date`
- mvn install
post_build:
commands:
- echo Build completed on `date`
artifacts:
files:
- target/messageUtil-1.0.jar
discard_paths: yes
```

Dall'ultima fase del buildspec.yml abbiamo fatto in modo di ottenere un bundle che fosse compatibile con CodeDeploy, servizio utilizzato nella prossima fase; il pacchetto ottenuto dall'ultima fase di build contiene sia l'applicazione pronta per essere messa in opera, sia gli script da eseguire per installarla e configurarla sull'istanza di deploy.

CodeDeploy

Il funzionamento di CodeDeploy prevede l'esecuzione dei propri script dall'interno dell'istanza; per questo, su ogni istanza dell'infrastruttura target del processo, andranno installati la CLI di AWS e il CodeDeploy agent.

Creiamo un file contenente la dichiarazione degli hook, questa volta denominato appspec.yml. Dopo aver creato da zero gli script necessari, abbiamo deciso, per ciascun hook, quali e quanti di essi chiamare.

Create application

Create an application and choose a deployment type. Specify the instances to deploy to. Specify the conditions for a successful deployment.

Application name* MyFirstCodeDeploy

Deployment group name* production

Deployment type

Choose the deployment to use to deploy your application. [Learn more](#)

In-place deployment
Updates the instances in the deployment group with the latest application revision. During a deployment, each instance will be briefly taken offline for its update.

Blue/green deployment
Replaces the instances in the deployment group with new instances and deploys the latest application revision to them. After instances in the replacement environment are registered with a load balancer, instances from the original environment are deregistered and can be terminated.

Add instances

Identify the instances you want to include in the deployment group. We will deploy the application revision to the instances that match the instance tag keys and values or Auto Scaling group names you specify.

Requirements for each instance in the deployment:

1. Each Amazon EC2 instance must be launched with the correct IAM instance profile attached. [Learn more](#)
2. Each Amazon EC2 instance must have identifying Amazon EC2 tags ([Learn more](#)) or be in an Auto Scaling group. [Learn more](#)
3. Each on-premises instance must have an associated IAM user, identifying on-premises instance tags, and a configuration file. [Learn more](#)
4. The AWS CodeDeploy agent must be installed and running on each instance. [Learn more](#)

Search by tags

	Tag type	Key	Value	Instances	
1	Amazon EC2	Name	MyInstance	0	
2	Amazon EC2				

Il file YAML appspec, in questo specifico caso, prevede che nel bundle vengano create due cartelle: una (*App*) dedicata all'applicazione e una (*Scripts*) dedicata agli script.

Di seguito un esempio generico della struttura di un file appspec.yml posto nella root directory del bundle:

```
version: 0.0
os: linux
files:
- source: App/
destination: /var/www/html/
- source: nginx.conf
destination: /etc/nginx/
hooks:
BeforeInstall:
- location: Scripts/UnzipResourceBundle.sh
timeout: 300
runas: root
- location: Scripts/InstallDependencies.sh
timeout: 300
runas: ubuntu
AfterInstall:
- location: Scripts/FixPermissions.sh
timeout: 300
runas: root
```

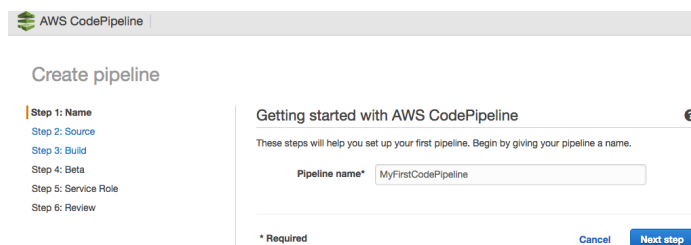
```
ApplicationStart:
- location: Scripts/WebServerStart.sh
timeout: 300
runas: root
ValidateService:
- location: Scripts/ValidateService.sh
timeout: 300
runas: ubuntu
```

CodePipeline

Parallelamente ai tre tool che abbiamo appena descritto, lavora CodePipeline, l'orchestrator dei servizi che si occupa di passare gli output generati da ciascuna fase al servizio successivo che li utilizzerà come input per svolgere il proprio compito. Ogni fase del processo di Continuous Integration sfrutta la piattaforma S3 come storage di supporto.

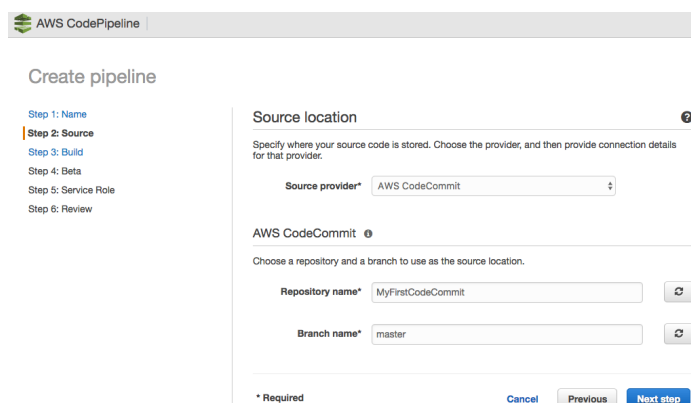
Affinché CodePipeline funzioni, occorre creare una pipeline; per fare ciò, accediamo alla console AWS ed effettuiamo i seguenti passaggi:

- scegliere un nome per la pipeline;



The screenshot shows the AWS CodePipeline console interface. On the left, a sidebar lists the steps: Step 1: Name (selected), Step 2: Source, Step 3: Build, Step 4: Beta, Step 5: Service Role, and Step 6: Review. The main content area is titled 'Getting started with AWS CodePipeline' and contains the text: 'These steps will help you set up your first pipeline. Begin by giving your pipeline a name.' Below this text is a form with a 'Pipeline name*' field containing the text 'MyFirstCodePipeline'. At the bottom of the form are 'Cancel' and 'Next step' buttons. A '* Required' label is visible at the bottom left of the form area.

- scegliere la sorgente, nel nostro caso CodeCommit;



The screenshot shows the AWS CodePipeline console interface. On the left, a sidebar lists the steps: Step 1: Name, Step 2: Source (selected), Step 3: Build, Step 4: Beta, Step 5: Service Role, and Step 6: Review. The main content area is titled 'Source location' and contains the text: 'Specify where your source code is stored. Choose the provider, and then provide connection details for that provider.' Below this text is a form with a 'Source provider*' dropdown menu set to 'AWS CodeCommit'. Underneath, there is a section for 'AWS CodeCommit' with the text: 'Choose a repository and a branch to use as the source location.' This section contains two input fields: 'Repository name*' with the text 'MyFirstCodeCommit' and 'Branch name*' with the text 'master'. At the bottom of the form are 'Cancel', 'Previous', and 'Next step' buttons. A '* Required' label is visible at the bottom left of the form area.

- configurare il passo di build, nel nostro caso CodeBuild;

AWS CodePipeline

Create pipeline

Step 1: Name
Step 2: Source
Step 3: Build
Step 4: Beta
Step 5: Service Role
Step 6: Review

Build

Choose the build provider that you want to use or that you are already using.

Build provider* AWS CodeBuild

AWS CodeBuild

AWS CodeBuild is a fully managed build service that builds and tests code in the cloud. CodeBuild scales continuously. You only pay by the minute. [Learn more](#)

Configure your project

Select an existing build project
 Create a new build project

Project name* MyFirstCodeBuild

[View project details](#)

* Required Cancel Previous Next step

- scegliere del provider di deploy, nel nostro caso CodeDeploy.

AWS CodePipeline

Create pipeline

Step 1: Name
Step 2: Source
Step 3: Build
Step 4: Beta
Step 5: Service Role
Step 6: Review

Beta

Choose how you deploy to instances. Choose the provider, and then provide the configuration details for that provider.

Deployment provider* AWS CodeDeploy

AWS CodeDeploy

Choose one of your existing applications, or create a new one in AWS CodeDeploy.

Application name* MyFirstCodeDeploy

Choose one of your existing deployment groups, or create a new one in AWS CodeDeploy.

Deployment group* production

* Required Cancel Previous Next step

La Pipeline è stata creata con successo.



MyFirstCodePipeline

View progress and manage your pipeline.

Edit

Release change

Source

Source i

AWS CodeCommit

No executions yet



Build

CodeBuild i

AWS CodeBuild

No executions yet



Beta

production i

AWS CodeDeploy

No executions yet

Nel momento in cui CodeCommit viene collegato a CodePipeline si crea automaticamente un trigger che fa corrispondere ad ogni `git push` effettuato l'avvio del processo descritto nella pipeline creata.

Il codice sorgente risultato da questa fase viene consegnato, sotto forma di input, a CodeBuild, il quale genera un bundle che servirà a sua volta da input a CodeDeploy. Sarà quest'ultimo ad installare l'app sull'infrastruttura target.

N.B. è importante verificare la correttezza del codice generato alla fine di ciascuna fase da ciascuno strumento; essendo ciascun risultato il punto di partenza per la fase successiva, se un output risultasse sbagliato, tutto il processo di deploy risulterebbe compromesso.

Noi stiamo utilizzando questa soluzione da ormai qualche settimana e siamo estremamente soddisfatti: abbiamo eliminato tutti gli effort di gestione della nostra precedente infrastruttura di Continuous Delivery, aumentando il numero di build che possiamo gestire in parallelo, e stiamo anche spendendo sensibilmente meno. Questo stack si rivela quindi ottimo in termini di efficienza, affidabilità ed economicità.

E voi, cosa ne pensate?

Se la nostra soluzione vi ha incuriositi e l'idea di implementarla nel vostro flusso di lavoro vi esalta, lasciate un commento o [contattateci](#)! Saremo felici di rispondere ad ogni vostra domanda, di leggere le vostre impressioni e, perché no, di aiutarvi a trarre il massimo vantaggio da questa applicazione.



Simone Merlini

CEO e co-fondatore di beSharp, Cloud Ninja ed early adopter di qualsiasi tipo di soluzione *aaS. Mi divido tra la tastiera del PC e quella a tasti bianchi e neri; sono specializzato nel deploy di cene pantagrueliche e nel test di bottiglie d'annata.

Get in touch

beSharp.it

proud2becloud@besharp.it

Copyright © 2011-2021 by beSharp srl - P.IVA IT02415160189