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 🙂

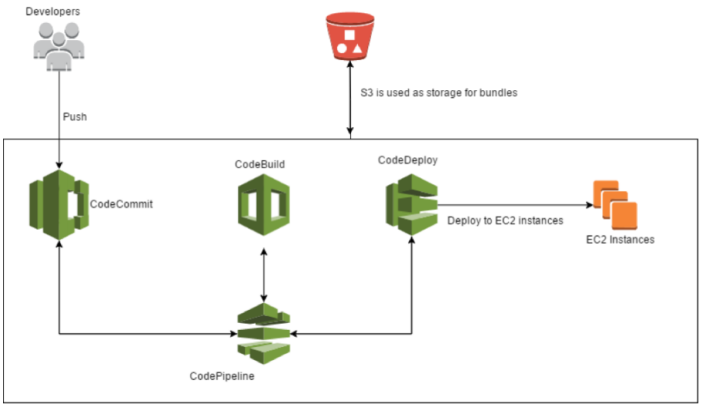
Per gli utenti AWS, l'utilizzo di uno stack di CD interamente basato sui suoi servizi managed è il modo più naturale ed immediato di rispondere a tutti questi requisiti

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 pol 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.

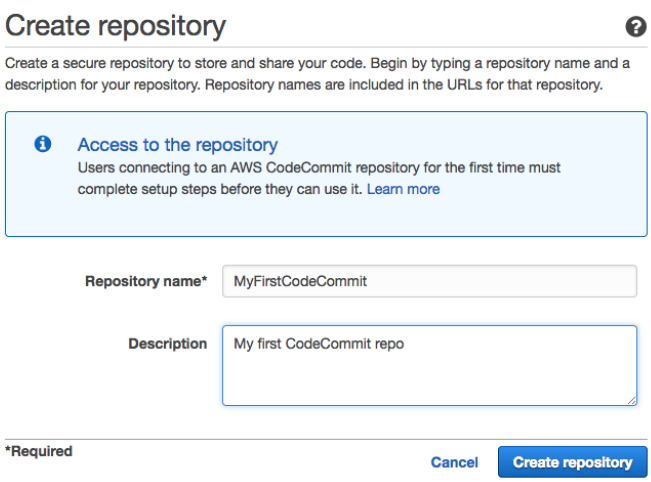
Ovviamente i costi possono variare molto in base al contesto e all'organizzazione di ogni team, ma ci sentiamo di dire che nella maggioranza dei casi questo stack risulti **una delle soluzioni più economiche in assoluto.**

Entriamo ora nel vivo dell'implementazione .

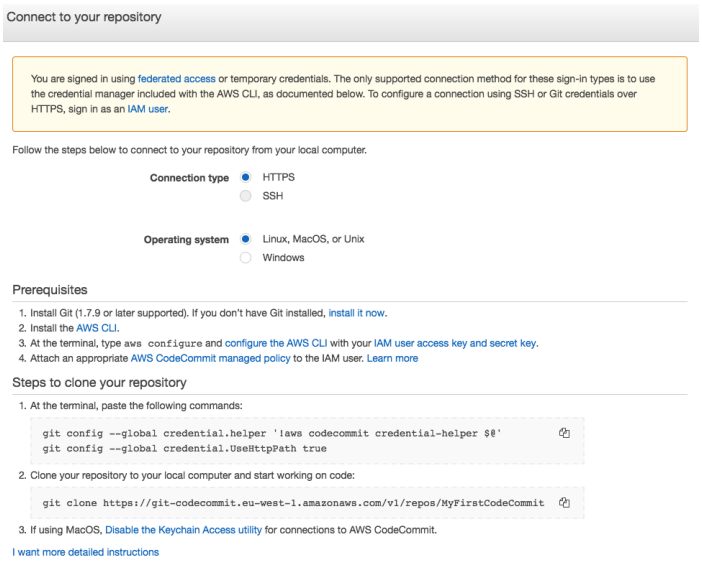
CodeCommit

Il primo servizio da configurare è CodeCommit.

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



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



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

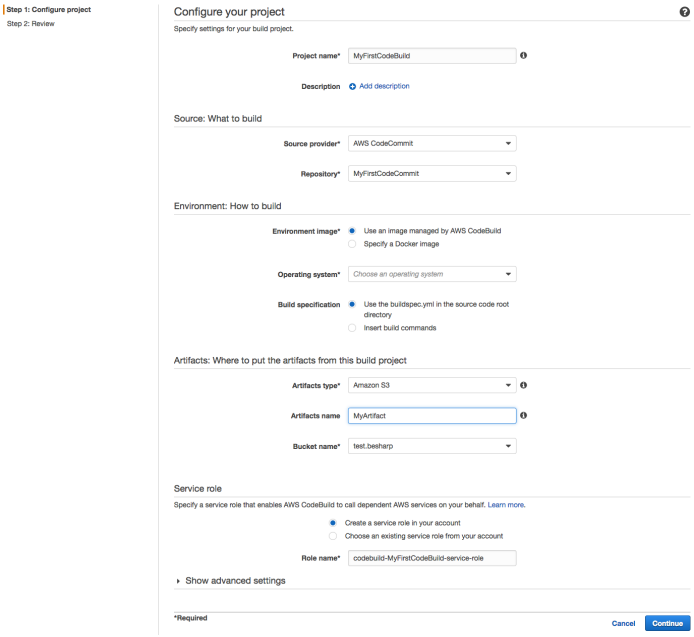
- *Git credential helper,* sfrutta la CLI. Serve quindi una coppia di chiavi per l'accesso;
- *chiave SSH* che è possibile caricare dalla gestione utenti IAM;
- *username e password* generati, sempre, dalla console di gestione IAM (utilizzo su HTTPS).

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.



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:

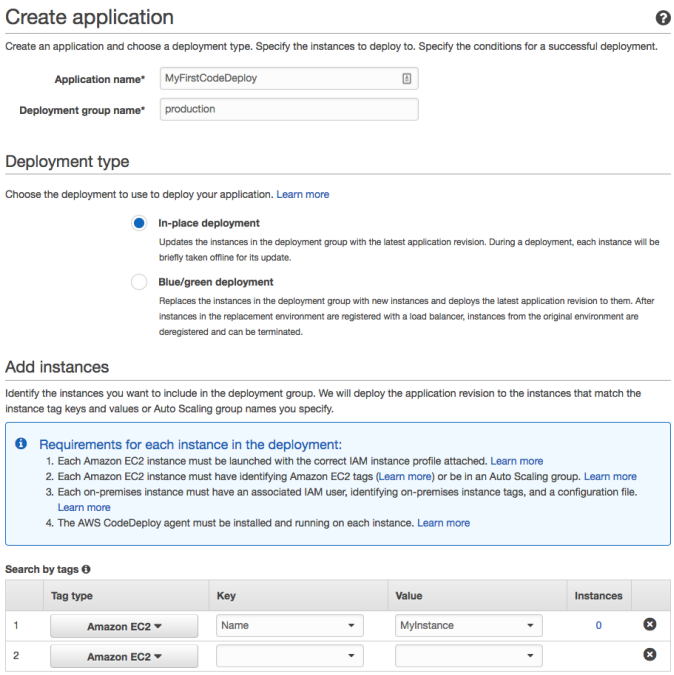
```
version: 0.1

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.



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
```

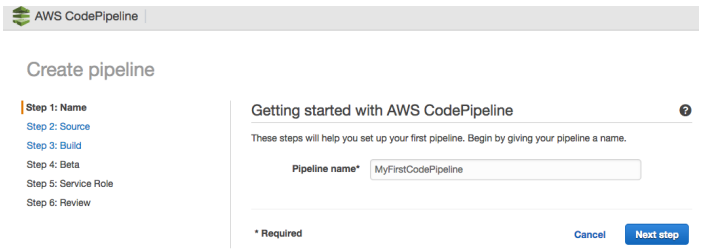
```
ApplicationStart:
- location: Scripts/WebServerStart.sh
timeout: 300
runas: root
ValidateService:
- location: Scripts/ValidateService.sh
timeout: 300
runas: ubuntu
```

CodePipeline

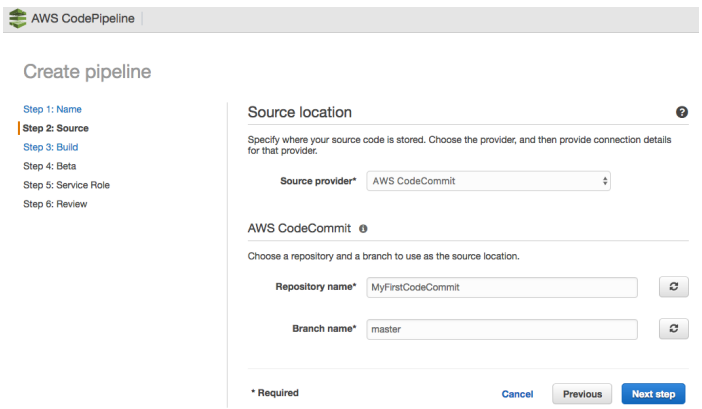
Parallelamente ai tre tool che abbiamo appena descritto, lavora CodePipeline, l'ochestrator dei servizi che si occupa di passare gli output generati da ciascuna fase al servizio successivo che li utilizzerà come input per svolgere il proprio compito. Ogni fase del processo di Continuous Integration sfrutta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

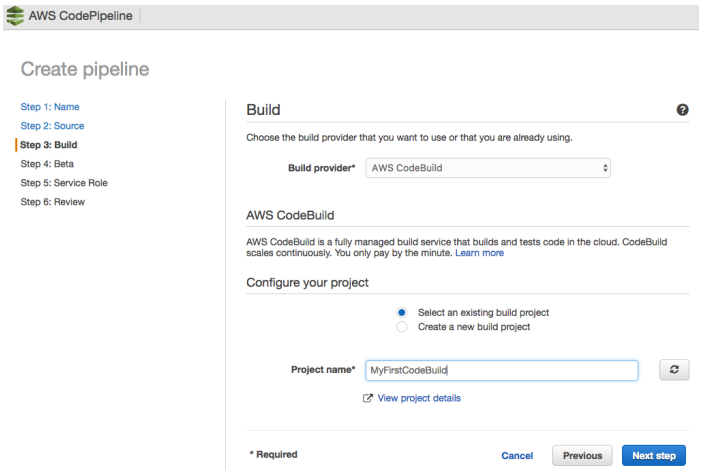
- scegliere un nome per la pipeline;



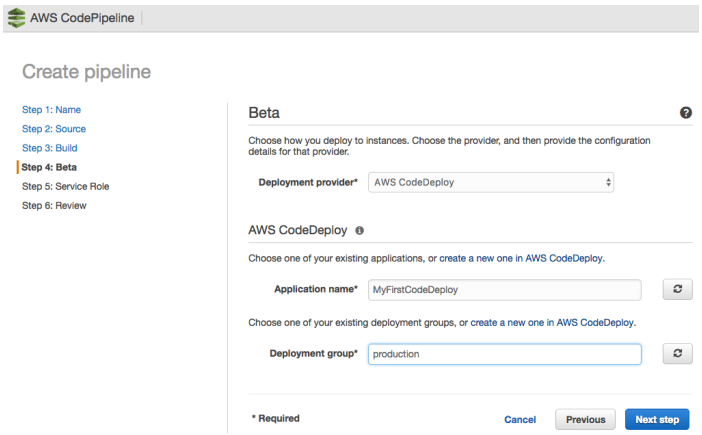
- scegliere la sorgente, nel nostro caso CodeCommit;



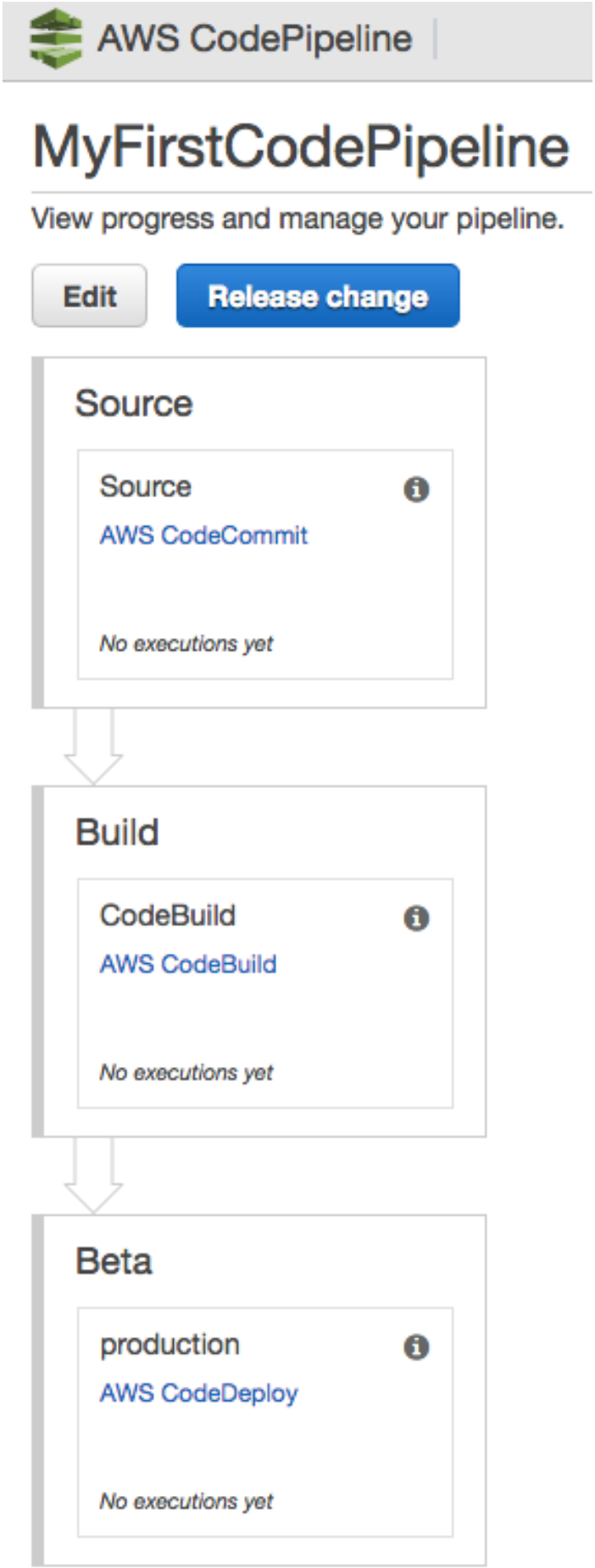
- configurare il passo di build, nel nostro caso CodeBuild;



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



La Pipeline è stata creata con successo.



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