COME CREARE CI/CD PIPELINE FLESSIBILI SU AWS CON FARGATE E SQS

Amazon Route 53 | AWS CodeDeploy | AWS Lambda | MongoDB

beSharp | 17 Maggio 2019

L'utilizzo di **Pipeline per il deploy automatico del codice** è ormai una feature quasi imprescindibile di ogni progetto di sviluppo in Cloud, in quanto il concetto stesso di architettura scalabile richiede che le macchine virtuali (o i container), che vengono avviati sul Cloud per gestire i picchi di traffico, utilizzino la versione più aggiornata del codice. Inoltre, la creazione di una pipeline automatica **libera i DevOps dalla gestione manuale** di AMIs e Docker images, oltre a eliminare la possibilità di "errori umani" in fase di deploy.

AWS mette a disposizione dei DevOps uno strumento molto potente per la creazione di Pipeline automatiche: **AWS CodePipeline.** Questo servizio totalmente *managed* funziona come un **orchestrator per una pipeline di CI/CD** con funzionalità analoghe a quelle offerte da altri servizi come Jenkins che però vanno installati su un'istanza EC2 e, pertanto, oltre a non essere in alta affidabilità, richiedono un effort significativo di configurazione e manutenzione.

Il flusso più comune di una AWS CodePipeline è composto da tre step:

- **Source:** AWS CodePipeline avvia un container completamente gestito e configurato da AWS che si occupa di eseguire il pull del commit desiderato dal repository git del progetto e salvarlo sul bucket S3 di appoggio della pipeline come un bundle compresso. Questa operazione viene triggerata automaticamente tramite AWS CloudWatch events ogni volta che uno sviluppatore esegue un push se il repository git è ospitato su AWS CodeCommit oppure su GitHub, mentre per altri servizi git è necessario configurare un webhook.
- **Build:** AWS CodePipeline avvia un container Docker configurabile dall'utente tramite il servizio AWS CodeBuild che all'avvio scarica il codice precedentemente salvato su S3 per eseguire gli step di build, test ed eventuali altri step di preparazione al deploy.
- **Deploy:** questo step può utilizzare diversi altri servizi AWS per eseguire il deploy del codice compilato e testato nello step precedente; per esempio può aggiornare la versione di una Webapp tramite AWS CodeDeploy oppure utilizzare AWS CloudFormation per deployare una nuova versione di una AWS Lambda Function o aggiornare le task definition dei container su ECS

Sebbene le features di AWS CodePipeline siano sufficienti per gli use case più comuni, alcune necessità particolari richiedono di sviluppare uno o più step personalizzati, così da avere maggiore flessibilità. In questo articolo vedremo come è possibile **creare una pipeline automatica in grado di eseguire la build di tutti i branch di un repo git hostato su AWS CodeCommit.**

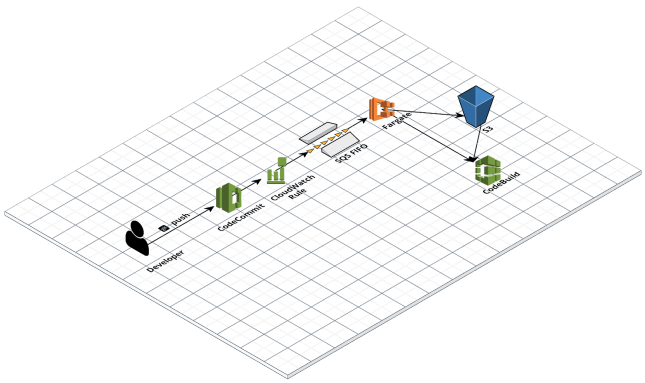
Molti progetti, in particolare quelli di dimensioni ragguardevoli, utilizzano git flow o un flusso analogo per organizzare il repository.

Questo fa sì che vi siano due o più branch (e.g. production, staging, development) contenenti il codice effettivamente deployato sui relativi ambienti e un gran numero di branch feature, contenenti le singole feature in fase di sviluppo assegnate ai rispettivi developer e team, che una volta completate vengono integrate in development e testate.

Tuttavia molto spesso non risulta possibile eseguire l'intera suite di test automatici direttamente dalle workstation degli sviluppatori, sia per motivi di tempi, che per la necessità di testare la sempre crescente integrazione del codice coi vari servizi SaaS di AWS. Per ovviare a questi problemi e ridurre gli errori di integrazione risulterebbe molto comodo poter lanciare direttamente la build e la suite di test ad ogni commit sulle singole features branch tramite AWS CodeBuild, invece che solamente al momento del merge della feature in dev attraverso la CodePipeline appositamente creata per questo ambiente.

Purtroppo al momento AWS CodePipeline non supporta il source da sorgenti multiple; è infatti necessario specificare sia il repo che il branch. Per risolvere il problema, in beSharp abbiamo sviluppato una soluzione creativa sfruttando la potenza di **CloudWatch Events, SQS e Fargate.**

Servizi utilizzati per la soluzione:



Schema dell'infrastruttura della soluzione proposta

CloudWatch Rules: il servizio di AWS che consente di creare regole per eseguire operazioni o in risposta ad eventi riguardanti l'account AWS, come ad esempio l'accensione di una EC2 o, nel nostro caso, un push su un repo CodeCommit, oppure ad intervalli di tempo fissati.

SQS FIFO: il servizio di code completamente gestito ed in alta affidabilità offerto da AWS**.** Nel nostro caso abbiamo usato la versione First In First Out (FIFO) in modo da essere certi di conservare l'ordine dei messaggi.

Fargate: Il terzo componente della soluzione è un container Docker deployato tramite Fargate (ECS), il nuovo servizio di AWS che consente di avviare container *as a service*, senza doversi occupare della gestione dell'infrastruttura sottostante.

In modo analogo al funzionamento standard di AWS CodePipeline abbiamo usato CloudWatch Rules per preparare una regola che viene triggerata al momento del push da parte di uno sviluppatore su uno qualsiasi dei branch. La regola ha due azioni configurate:

la prima accoda un messaggio in una coda SQS, mentre la seconda avvia il container Fargate. Il messaggio inserito nella coda è il *json* che descrive l'intero evento che ha avviato l'esecuzione della CloudWatch Rule e che contiene il nome del repo, il nome del branch e l'id del commit appena inviato dallo sviluppatore.

L'event pattern della regola sarà simile a questo:

```
{
  "source": [
    "aws.codecommit"
  ],
  "detail-type": [
    "CodeCommit Repository State Change"
  ],
  "resources": [
    "arn:aws:codecommit:eu-west-1:<ACCOUNT_ID>:<REPOSITORY>",
    ...
  ],
  "detail": {
    "event": [
      "referenceCreated",
      "referenceUpdated"
    ]
  }
}
```

La coda SQS FIFO contiene perciò i messaggi corrispondenti agli eventi di push del codice sul repository e viene consumata dai container Fargate. Per evitare che messaggi corrotti possano venire ri-processati all'infinito, abbiamo aggiunto una **dead letter queue** dove i messaggi vengono trasferiti dopo due tentativi di lettura falliti.

Una volta avviato dalla CloudWatch Rule, **il container Fargate legge i messaggi dalla coda,** esegue il pull del commit dal repository CodeCommit, salva il bundle compresso del codice su s3 ed infine lancia AWS CodeBuild coi parametri corretti.

Il container Docker è stato creato usando il **Dockerfile:**

```
FROM ubuntu:16.04
RUN apt-get update
RUN apt-get install wget -y
RUN apt-get install numactl -y
RUN apt-get install jq -y
```

```
RUN apt-get install zip -y
RUN apt-get install git -y
RUN apt-get install software-properties-common -y
RUN add-apt-repository ppa:jonathonf/python-3.6 -y
RUN apt-get update
RUN apt-get install python3.6 -y
RUN wget https://bootstrap.pypa.io/get-pip.py
RUN python3.6 get-pip.py
RUN pip3.6 install awscli --upgrade
RUN pip3.6 install boto3
RUN mkdir /pipeline_source
WORKDIR /pipeline_source
ADD ./codecommit_source.sh /pipeline_source/codecommit_source.sh
RUN chmod +x /pipeline_source/codecommit_source.sh
CMD /pipeline_source/codecommit_source.sh
```

Come si può vedere sono richiesti solo pacchetti standard di bash, oltre alla AWS CLI. Lo script *codecommit_source.sh* viene avviato al momento dell'accensione del container, ed esegue la logica appena descritta.

Un **codecommit_source.sh** di esempio è mostrato qui sotto:

```
#!/bin/bash
set -Eeuxo pipefail

MESSAGE=$(aws sqs receive-message --queue-url https://sqs.eu-west-1.amazonaws.com/<ac
count-id>/custom-codecommit-events.fifo --wait-time-seconds 20)
RECEIPT_HANDLE=$(echo $MESSAGE | jq -r '.Messages | .[] | .ReceiptHandle')

aws sqs delete-message --queue-url https://sqs.eu-west-1.amazonaws.com/<account-id>/c
ustom-codecommit-events.fifo --receipt-handle $RECEIPT_HANDLE

if [ -n "$MESSAGE" ]
then
EVENT=$(echo $MESSAGE | jq -r '.Messages | .[] | .Body | fromjson')
REPOSITORY_NAME=$(echo $EVENT | jq -r '.detail | .repositoryName')
COMMIT_ID=$(echo $EVENT | jq -r '.detail | .commitId')
BRANCH_NAME=$(echo $EVENT | jq -r '.detail | .referenceName')
REPO_URL=https://git-codecommit.eu-west-1.amazonaws.com/v1/repos/$REPOSITORY_NAME
git config --global credential.helper '!aws codecommit credential-helper $@'
git config --global credential.UseHttpPath true
git clone --depth 10 --branch $BRANCH_NAME $REPO_URL

cd $REPOSITORY_NAME
git checkout $COMMIT_ID
rm -rf .git
zip -r ../$COMMIT_ID.zip .
cd ..

rm -rf $REPOSITORY_NAME

if [ -s $COMMIT_ID.zip ]
then
CODEBUILD_PROJECT=$REPOSITORY_NAME

if [ $BRANCH_NAME != "test" ] && [ $BRANCH_NAME != "develop" ] && [ $BRANCH_NAME !=
"staging" ]
```

```
then

aws s3 cp $COMMIT_ID.zip s3://$CODE_BUCKET/$REPOSITORY_NAME/$BRANCH_NAME/$COMMIT_ID.zi
p
echo s3://$CODE_BUCKET/$REPOSITORY_NAME/$BRANCH_NAME/$COMMIT_ID.zip
aws codebuild start-build --project-name $CODEBUILD_PROJECT --environment-variables-ov
erride name=COMMIT_ID,value=$COMMIT_ID,type=PLAINTEXT --source-type-override S3 --sou
rce-location-override $CODE_BUCKET/$REPOSITORY_NAME/$BRANCH_NAME/$COMMIT_ID.zip --art
ifacts-override type=NO_ARTIFACTS

fi
fi
else
echo "no message in queque"
fi
```

Infine, chi gestisce il source code dovrà aver cura di **creare/modificare il buildspec** in modo da salvare gli output della build su S3 con un nome facilmente leggibile.

La soluzione qui riportata può essere facilmente modificata per funzionare anche **in caso di account multipli.** Ad esempio possono essere presenti due account: il primo account ("master") contenente l'ambiente di produzione e i repos mentre il secondo gli ambienti di staging/development e le pipeline. Per far ciò è necessario aggiungere un *ruolo* all'account "master" che possa essere assunto da staging in modo da fare il pull dei repository. Infine sarà anche necessario configurare event bus su entrambi gli account in modo da condividere i messaggi di prod relativi ai repo con l'account di sviluppo.

Per concludere AWS CodePipeline è uno strumento molto potente, ma per alcuni casi di uso non è sufficiente e va perciò affiancato a soluzioni custom come quella proposta che sono facilmente configurabili usando l'ampia suite di servizi messi a disposizione da AWS.

Vuoi raccontarci di una tua soluzione di CD/CI innovativa o avere ulteriori infomazioni su quella proposta in questo articolo? Non esitare a commentare e/o a **contattarci!**

beSharp

Dal 2011 beSharp guida le aziende italiane sul Cloud. Dalla piccola impresa alla grande multinazionale, dal manifatturiero al terziario avanzato, aiutiamo le realtà più all'avanguardia a realizzare progetti innovativi in campo IT.

Get in touch

beSharp.it
proud2becloud@besharp.it

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