HOW TO CREATE FLEXIBLE CI/CD PIPELINE ON AWS WITH FARGATE AND SQS

Amazon Route 53 | AWS CodeDeploy | MongoDB

beSharp | 17 May 2019

The use of **Pipeline for automatic code deployment** is now an almost essential feature of every development project in the Cloud, as the concept of scalable architecture requires that virtual machines (or containers), which are started on the Cloud to manage traffic spikes use the most up-to-date version of the code. Furthermore, the creation of an automated pipeline **frees the DevOps from the manual management** of AMIs and Docker images, as well as eliminating the possibility of "human errors" in the deployment phase.

AWS provides DevOps with a very powerful tool for creating automatic Pipelines: **AWS CodePipeline.** This fully managed service works as an **orchestrator for a CI/CD** pipeline with similar functionalities to those offered by other services such as Jenkins but which must be installed on an EC2 instance and, therefore, in addition to not being highly reliable, require a significant configuration and maintenance effort.

The most common flow of an AWS CodePipeline consists of three steps:

- **Source:** AWS CodePipeline starts a fully managed and configured container from AWS that takes care of pulling the desired commit from the project git repository and saves it to the supporting pipeline S3 bucket as a compressed bundle. This operation is triggered automatically through AWS CloudWatch events whenever a developer performs a push if the git repository is hosted on AWS CodeCommit or GitHub, while for other git services it is necessary to configure a webhook.
- **Build:** AWS CodePipeline starts a Docker container that can be configured by the user through the AWS CodeBuild service, which on startup downloads the code previously saved on S3 to execute the build steps, tests and any other steps to prepare to deploy.
- **Deploy:** this step can use several other AWS services to deploy the compiled and tested code in the previous step; for example, you can update the version of a Web app using AWS CodeDeploy, or use AWS CloudFormation to deploy a new version of an AWS Lambda Function or update the container task definition on ECS.

Although the features of AWS CodePipeline are sufficient for the most common use cases, some special needs require you to develop one or more customized steps, to have more flexibility. In this article, we will see how it is possible to **create an automated pipeline able to build all the branches of a repo git hosted on AWS CodeCommit.**

Many projects, particularly large ones, use git flow or a similar flow to organize the repository.

This means that there are two or more branches (e.g., production, staging, development) containing the code actually deployed on the relevant environments and a large number of branch features, containing the individual features being developed assigned to the respective developer and team, which once completed are integrated into development and publications.

However, very often it is not possible to run the entire suite of automated tests directly from the developers' workstations, both for reasons of time and for the need to test the ever-increasing integration of the code with the various AWS SaaS services. To overcome these problems and reduce integration errors, it would be very convenient to be able to directly launch the build and the test suite at each commit on the individual features branch via AWS CodeBuild, instead of just merge the feature in dev through CodePipeline created explicitly for this environment.

Unfortunately, at the moment, AWS CodePipeline does not support source from multiple inputs; it is in fact necessary to specify both the repo and the branch. To solve the problem, in beSharp, we have developed a creative solution using the power of **CloudWatch Events, SQS, and Fargate.**

Services used for the solution:

CloudWatch Rules: the AWS service that allows you to create rules to perform operations or in response to events concerning the AWS account, such as turning on an EC2 or, in our case, a push on a CodeCommit repo, or to fixed time intervals.

SQS FIFO: the fully managed and highly reliable code service offered by AWS. In our case, we used the First In First Out (FIFO) version to be sure of preserving the order of the messages.

Fargate: The third component of the solution is a Docker container deployed through Fargate (ECS), the new AWS service that allows you to start containers *as a service*, without having to deal with management of the underlying infrastructure.

In a similar way to the standard operation of AWS CodePipeline, we used CloudWatch Rules to prepare a rule that is triggered at the time of push by a developer on any of the branches. The rule has two configured actions:

The first queues a message in an SQS queue, while the second starts the Fargate container. The message entered in the queue is the JSON that describes the whole event that started the CloudWatch Rule and contains the name of the repo, the name of the branch and the ID of the commit just sent by the developer.

The event pattern of the rule will look like this:

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

The SQS FIFO queue, therefore, contains the messages corresponding to the push code events on the repository and is consumed by the Fargate containers. To prevent corrupt messages from being re-processed indefinitely, we added a **dead letter queue** where messages are transferred after two failed read attempts.

Once started from the CloudWatch Rule, **the Fargate container reads messages from the queue,** pulls the commit from the CodeCommit repository, saves the compressed code bundle on s3 and finally launches AWS CodeBuild with the correct parameters.

The Docker container was created using the **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
```

As shown, only standard bash packages are required, in addition to the AWS CLI.
The *codecommit_source.sh*script starts when the container is turned on and executes the logic

described above.

An example of **codecommit_source.sh** is shown below:

```
#!/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>/cu
stom-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
```

Finally, those who manage the source code will have to take care of **creating/modifying the buildspec** to save the build's outputs on S3 with an easily readable name.

The solution shown here can be easily modified to work even **in the case of multiple accounts.** For example, two accounts may be present: the first account ("master") containing the production environment and the repos, while the second accounts for the staging/development environments and the pipelines. To do this, it is necessary to add a *role*to the "master" account that can be taken over by staging to pull the repositories. Finally, it will also be necessary to configure event buses on both accounts in order to share rep messages with the development account.

In conclusion, AWS CodePipeline is a very powerful tool, but for some cases of use, it is not enough and should, therefore, be combined with custom solutions like the one proposed which are easily configurable using the wide suite of services made available by AWS.

Would you like to tell us about your innovative CD/CI solution or have more information on that proposed in this article? Don't hesitate to comment and/or **contact us!**

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