

FULL STACK CONTINUOUS DELIVERY ON AWS

CI/CD

Continuous Delivery



Simone Merlini | 10 March 2017

AWS DevOps

With the launch of [CodeBuild](#) last November [on the stage of re:Invent](#), AWS added the missing puzzle piece to the suite of software development lifecycle management tools.

[We were there](#), and this was all we had been waiting for to be able to get to work and implement **a Continuous Delivery system entirely based on services operated by Amazon Web Services.**

Before CodeBuild, AWS's suite only covered aspects relating to source control ([CodeCommit](#)), release management ([CodeDeploy](#)) and orchestration ([CodePipeline](#)), forcing developers to use third-party tools (for example [Jenkins](#)) to manage the build and test phases. Tools that have to be configured and operated manually, with all the issues that this entails in terms of the complexity, costs, and reliability of the solution.

We are thinking about how the Continuous Delivery stack is, on the one hand, critical (an incident with one of these tools can compromise hours or days of a whole development team's work), but on the other hand, it represents—especially for small, agile DevOps teams—an object which is foreign to the “core” activities which should have the least effort focused on it possible; **a sort of black box that just needs to work.**

A Continuous Delivery stack must:

- Be extremely reliable (no single-point-of-failure)
- Ensure the durability of the data (in particular with regard to repositories)
- Require minimal effort to manage and configure
- Easily integrate with both development and production environments and tools
- Be easy to scale in order to grow with both the team and their projects
- Be cheap

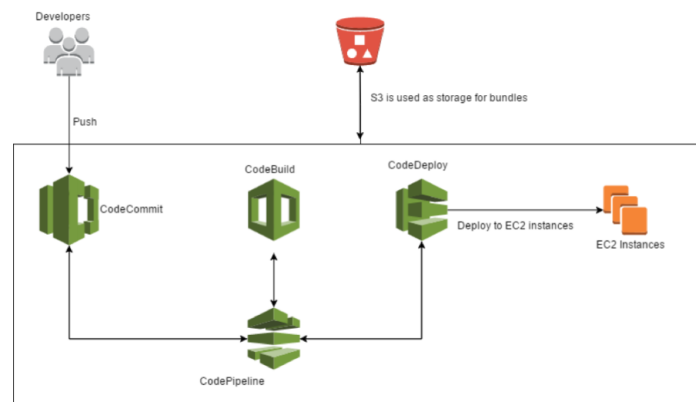
For AWS users, the use of a CD stack which is entirely based on its managed services is the most natural and immediate option that meets all these requirements.

The Solution

Let's briefly review which tools we have available to us:

- **CodeCommit:** a fully-managed and automatically-scaled git repository
- **CodeBuild:** a build and testing system based on containers
- **CodeDeploy:** an agent-based system to deploy EC2 instances automatically
- **CodePipeline:** an orchestrator that is fully integrated into the AWS ecosystem, which is able to make the other three services interact with one another or with third-party providers.

Our team has developed a **comprehensive and automated solution for managing the software lifecycle on Amazon Web Services**, applicable both to hybrid environments (in this case, integration with the repository and build is done in the Cloud, whereas the deploy can take place on any type of instance, both in the Cloud or on-premise), and to fully Cloud-based environments, with the staging and production machines which are also in the Cloud.



The developers push the code on CodeCommit, a trigger is set off, and CodePipeline takes the code and feeds it into CodeBuild, which carries out the build and the tests and creates the artifact, ready for the deploy. At this point, CodePipeline passes the packet to CodeDeploy which releases it on a pool of EC2 instances. At every step, S3 is used as supporting storage for the artifacts.

Using only AWS services, which are compatible and specifically designed to cooperate with each other, has the advantage of putting specific triggers and extremely simplified setups at our disposal. **Everything scales automatically and without the need for any particular technical operations**; the team is, therefore relieved of the burden of sizing the Continuous Delivery infrastructure in advance.

It is also possible to manage security and permissions at a granular level by using IAM role and IAM policy. Unlike what happens when using other tools, it is possible to remain confined within the sandbox of a single Amazon account, without having to externally present an endpoint to start the build process.

We have found that CodeBuild takes care of the build and test phases by using provisioning of the containers; these start up only when needed, thus optimising performance and eliminating the need for dedicated instances, as happens when using Jenkins, for example.

Another key feature of CodeBuild is isolation, which allows for **each build to be segregated at both the application and release levels.**

AWS Costs

Before we begin with the tutorial, let's have a look at the AWS costs incurred by this solution:

The price of each CodePipeline pipeline is \$1/month, whilst for the first 5 users (50GB and 10000 git requests), the CodeCommit service is totally free. For the sixth user onwards, the cost is \$1/month per user.

The cost of the CodeBuild service is calculated according to the minutes of the actual use of the containers carried out by the builds, a model that allows for significant cost savings compared to traditional runners based on EC2 instances, which are charged on an hourly basis.

CodeDeploy is free, with no limits on traffic or time.

The additional costs to be considered are the storage and bandwidth costs generated by S3, which is used as supporting storage for the artifacts between one step and the next of each pipeline.

Obviously, costs can vary widely depending on the context and organisation of every team, but we would like to say that in most cases, this stack works out to be **one of the cheapest solutions around.**

Now let us enter into the heart of the implementation.

CodeCommit

The first service to be configured is CodeCommit.

We create the repository by logging into the AWS console or, in the case of programming access, to the CLI.

The screenshot shows the 'Create repository' form in the AWS 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'. The form has two input fields: 'Repository name*' with the value 'MyFirstCodeCommit' and 'Description' with the value 'My first CodeCommit repo'. At the bottom left, there is a note '*Required'. At the bottom right, there are two buttons: 'Cancel' and 'Create repository'.

Create repository ?

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.

i **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](#)

Repository name*

Description

*Required Cancel Create repository

Now we create an IAM user that will allow us to proceed with the main git actions on the repository that has just been created.

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](#)

To get the necessary permissions to access the repo, you must authenticate as an IAM user. Here are the possible methods:

- *Git credential helper*, which uses the CLI. In which case, a key pair is needed for access;
- an *SSH key* that can be generated by IAM user management;
- a *username and password*, also generated by the IAM management console (using HTTPS).

With each push of the app's source code, this is taken over by CodeBuild, which deals with the provisioning of a temporary, isolated environment (container) inside which the build and test phases will take place.

CodeBuild

Before proceeding to the test and build phases, we must configure CodeBuild, specifying the container size, the type of image desired (for example, a Linux container with a preinstalled framework from the ones available or a custom image) and specifying the steps we want CodeBuild to take to carry out the test and build of the app. This last step means a choice between two methodologies: declaring the inline commands or using a file called `buildspec.yml` that will invoke the hooks of the build cycle.

We chose to use the latter method because, in this way, the YAML `buildspec` file can be versioned along with the application code, giving us the opportunity to change the test and build procedures just before the build phase itself.

So, within each hook, we have specified the commands that CodeBuild will perform, moment by moment.

Below is a generic example of the structure of a buildspec.yml file located in the root directory of the 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
```

Since the last phase of the buildspec.yml, we made sure to get a bundle that was compatible with CodeDeploy, the service used in the next phase; the package obtained from the last build phase contains both the application, ready for implementation, and the scripts to run to install it and configure it on the deploy instance.

CodeDeploy

For CodeDeploy to function, its scripts must be executed from within the instance; for this to happen, the AWS CLI and the CodeDeploy agent must be installed in each instance of the process' target infrastructure.

We create a file containing the declaration of the hooks, this time called `appspec.yml`. After having created the necessary scripts from scratch, we decided which and how many of them to call for each hook.

The screenshot shows the 'Create application' page in the AWS CodeDeploy console. It includes a form for 'Application name*' (MyFirstCodeDeploy) and 'Deployment group name*' (production). Below this is the 'Deployment type' section with two radio buttons: 'In-place deployment' (selected) and 'Blue/green deployment'. The 'Add instances' section contains a blue box with requirements for each instance in the deployment. At the bottom, there is a 'Search by tags' table with columns for Tag type, Key, Value, and Instances.

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

The YAML `appspec` file, in this specific case, requires two folders to be created in the bundle: one (*App*) dedicated to the application and another (*Scripts*) dedicated to the scripts.

Below is a generic example of the structure of an `appspec.yml` file located in the root directory of the 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
```

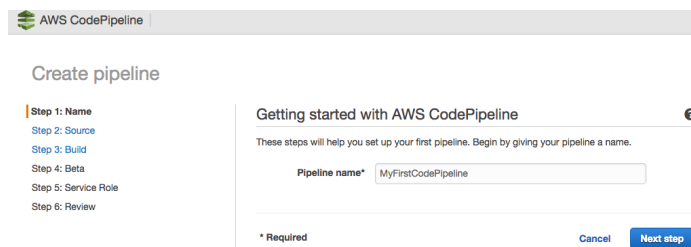
```
ApplicationStart:  
- location: Scripts/WebServerStart.sh  
timeout: 300  
runas: root  
ValidateService:  
- location: Scripts/ValidateService.sh  
timeout: 300  
runas: ubuntu
```

CodePipeline

Working in parallel with the three tools that we have just described is CodePipeline, the service orchestrator that deals with passing the outputs generated by each phase to the next service, which will use them as the inputs needed to perform its task. Each phase of the Continuous Integration process uses the S3 platform as supporting storage.

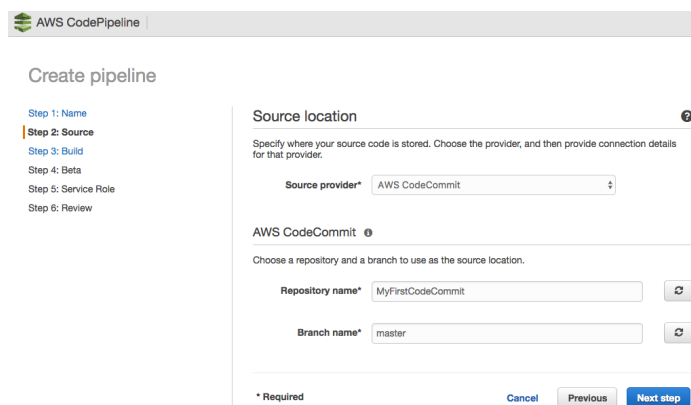
For CodePipeline to function, you need to create a pipeline; to do this, we access the AWS console and perform the following steps:

- choose a name for the pipeline;



The screenshot shows the AWS CodePipeline console interface. On the left, a vertical navigation pane lists the steps of the 'Create pipeline' wizard: Step 1: Name (highlighted), 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 a text input field for 'Pipeline name*' with the value 'MyFirstCodePipeline'. Below the input field are 'Cancel' and 'Next step' buttons. A '* Required' label is visible at the bottom left of the form.

- choose the source, in our case CodeCommit;



The screenshot shows the AWS CodePipeline console interface at the 'Source location' step. The left navigation pane shows Step 2: Source (highlighted). The main content area is titled 'Source location' and contains a dropdown menu for 'Source provider*' set to 'AWS CodeCommit'. Below this, there is a section for 'AWS CodeCommit' with a sub-header 'Choose a repository and a branch to use as the source location.' This section includes two text input fields: 'Repository name*' with the value 'MyFirstCodeCommit' and 'Branch name*' with the value 'master'. Both input fields have a refresh icon to their right. 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.

- configure the build step, in our case 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

- choose the deploy provider, in our case 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

The Pipeline has been successfully created.



MyFirstCodePipeline

View progress and manage your pipeline.

Edit

Release change

Source

Source



AWS CodeCommit

No executions yet



Build

CodeBuild



AWS CodeBuild

No executions yet



Beta

production



AWS CodeDeploy

No executions yet

When CodeCommit is linked to CodePipeline, it automatically creates a trigger that means that every *git push* causes the process described in the pipeline created to initiate.

The source code that results from this phase is delivered in the form of an input to CodeBuild, which generates a bundle that will, in turn, be the input for CodeDeploy. CodeDeploy will then

install the app on the target infrastructure.

N.B. it is important to verify that the code generated at the end of each phase by each tool is correct; as each result is the starting point for the next phase, if an output were to be wrong, the whole deploy process would be compromised.

We have been using this solution for the past few weeks now, and we are delighted indeed with it: we deleted all the management efforts of our previous Continuous Delivery infrastructure, increasing the number of builds that we can manage in parallel, and we are also spending significantly less. This stack has turned out to be excellent in terms of efficiency, reliability, and cost efficiency.

What do you think of it?

If our solution has intrigued you and you like the idea of implementing it in your workflow, leave us a comment or [contact us](#)! We'd be happy to answer all your questions, read your thoughts, and help you get the most out of this application.



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