Creating a CI/CD DevOps Pipeline Workshop

Continuous integration and continuous delivery are highly fundamental topics in the software industry, especiallaws cloudformation create-stack --template-body file://ecs-jenkins-demo.template --stack-name JenkinsStack --capabilities CAPABILITY\_IAM --tags Key=Name,Value=Jenkins --region us-west-2 --parameters ParameterKey=EcsStackName,ParameterValue=EcsClusterStacky with cloud and container technologies. Container technologies such as [Docker](https://www.docker.com/), [Kubernetes](https://kubernetes.io/),  and automation servers like [Jenkins](http://jenkins.io/) make it easier to manage our project workflows.

Our goal is to ensure our pipeline works well after each code being pushed. The processes we want to auto-manage:

* Code checkout
* Run tests
* Compile the code
* Create Docker image
* Push the image to Docker Hub
* Pull and run the image

# Step 1 Install the tools.

Install the AWS Command Line Interface (CLI) on your system by using the Pip install utility (Note that you must also have Python installed).

An Access Key and a Secret Access Key will be provided for you.

Edit the ~/.aws/credentials. Add the Access Key and Secret Access Key to the appropriate lines in the file.

Alternatively, use the aws configure command and add the keys via the command line.

In the AWS configuration file in ~/.aws, make sure that the AWS region is set to us-east-1

You will also need to install the jq binary (You can find this in your package management system) as well as the docker binaries.

# Step 2. Clone the project repo

You will be given a git repo that contains a number of template files. Make sure that all further commands are run through this repo:

Now create an Amazon AWS cluster using the aws cloud formation command:

aws cloudformation create-stack --template-body file://ecs-cluster.template --stack-name EcsClusterStack1 --capabilities CAPABILITY\_IAM --tags Key=Name,Value=ECS --region us-west-2 --parameters ParameterKey=KeyName,ParameterValue=<your\_key\_name> ParameterKey=EcsCluster,ParameterValue=getting-started ParameterKey=AsgMaxSize,ParameterValue=2

Make sure that the Cluster has been completed by running the following:

aws cloudformation describe-stacks --stack-name EcsClusterStack1 --query 'Stacks[\*].[StackId, StackStatus]'

# Step 3. Deploy a Jenkins cluster.

Jenkins is a popular server for implementing continuous integration and continuous delivery pipelines. In this example, you'll use Jenkins to build a Docker image from a Dockerfile, push that image to the Amazon ECR registry that you created earlier, and create a task definition for your container. Finally, you'll deploy and update a service running on your ECS cluster.

First, create the Jenkins cluster by implementing the following command:

aws cloudformation create-stack --template-body file://ecs-jenkins-demo.template --stack-name JenkinsStack --capabilities CAPABILITY\_IAM --tags Key=Name,Value=Jenkins --region us-west-2 --parameters ParameterKey=EcsStackName,ParameterValue=EcsClusterStack

Make sure that it is complete by running the following:

aws ec2 describe-instances --filters "Name=tag-value","Values=JenkinsStack" --region us-west-2 | jq .Reservations[].Instances[].PublicDnsName

Once creation is complete: retrieve the jenkins public host name by running:

aws ec2 describe-instances --filters "Name=tag-value","Values=JenkinsStack" --region us-west-2 | jq .Reservations[].Instances[].PublicDnsName

Now, ssh into the instance and copy the temp initial Jenkins password from the directory:

/varlib/jenkins/secrets/initialAdminPassword

To do this, run the following:

sudo cat /var/lib/jenkins/secrets/initialAdminPassword and copy the output to a local file using copy and paste.

# Step 3. Create an ECR Registry

Amazon ECR is a private Docker container registry that you'll use to store your container images. For this example, we'll create a repository named devops-course  in the us-east-2 (Ohio) region.

Run the following command:

aws ecr create-repository --repository-name hello-world --region us-east-2

Record the output value, especially the URL as you’ll need it later.

Because the Docker CLI doesn't support the standard AWS authentication methods, you need to authenticate the Docker client in another way so Amazon ECR knows who is trying to push an image. Using the AWS CLI, you generate an authorization token that you pass into the Docker login command.

type: **$(aws ecr get-login --no-include-email)**

# 