Continuous Delivery Pipeline for Amazon ECS Using Jenkins, GitHub, and Amazon ECR

This getting started guide is intended to help you set up and configure a continuous delivery pipeline for Amazon EC2 Container Service (Amazon ECS) using Jenkins, GitHub, and the Amazon EC2 Container Registry (Amazon ECR). The pipeline builds Docker images from a GitHub repository, pushes those images to an ECR registry, creates an ECS task definition, and then uses that task definition to create a service on the ECS cluster. We use Jenkins to orchestrate the different steps in the workflow.

Prerequisites

To use this guide, you must have the following software components installed:

- Python - a prerequisite for the AWS CLI
- PIP - a prerequisite for the AWS CLI
- AWS CLI
- Homebrew (OS X only)

  Note: Homebrew is the package manager for OS X. You will need homebrew to install jq.

- Chocolatey NuGet (Windows only) - a package manager for Windows

  Note: when installing Chocolatey, you might have to launch command window as Administrator. Once you have Chocolatey installed on your machine, you can use it to install the remaining prerequisites.

- jq - a command line utility for parsing JSON output
- Git command line tools - used to clone and push files to and from GitHub
- Docker for Mac, Docker for Windows or Docker Toolbox

Step 1: Build an ECS Cluster

1. Create an AWS access key and secret key by opening a terminal window, and then typing the following:

   aws iam create-access-key --user-name <user_name>

   `<user_name>` is an IAM user with *AdministratorAccess* IAM Policy.

2. Copy and paste the output from the previous command to a text file

Note: *AdministratorAccess* is a managed policy that allows attached entities to perform all actions against all resources. Although we're using it here for convenience, you should remove the *AdministratorAccess* policy should be removed from your IAM user when it's no longer needed.

3. Create an AWS profile on your local machine. From a command prompt, type:

   ```
   aws configure
   ```

   At the prompts, paste your aws access key ID, aws secret key ID, the preferred region (us-west-2), and **json** as the output format.

4. Create an SSH key in the us-west-2 region. We will use this SSH key to login to the Jenkins server to retrieve the administrator password.

   - Open the EC2 console
   - Under the **Networking & Security**, choose **Key Pairs**
   - Choose **Create Key Pair**
   - Assign a name to the key pair by typing a name in the **Key pair** name field, then click the **Create** button

   A file will be downloaded to your default download directory

5. (OS X only) Change the working directory to your download directory and change permission so only the current logged in user can read it. `<file_name>` is the name of the .pem file you downloaded earlier: `chmod 400 <file_name>`

6. Clone the git repository that contains the CloudFormation templates to create the infrastructure we'll use to build our pipeline.

7. Open a command prompt and clone the Github repository that has the template. `git clone https://github.com/jicowan/hello-world`

8. Change the working directory to the directory that was created when you cloned the repository. At the command prompt, type or paste the following. Where `<key_name>` is the name of an SSH key in the region where you're creating the ECS cluster:

   ```
   aws cloudformation create-stack --template-body file://ecs-cluster.template --stack-name EcsClusterStack --capabilities CAPABILITY_IAM --tags Key=Name,Value=ECS --region us-west-2 --parameters ParameterKey=KeyName,ParameterValue=<key_name> ParameterKey=EcsCluster,ParameterValue=getting-started ParameterKey=AsgMaxSize,ParameterValue=2
   ```

   Note: Do not proceed to the next step until the **Stack Status** shows **CREATE_COMPLETE**. To get the status of the stack at a command prompt, type `aws cloudformation describe-stacks --stack-name EcsClusterStack --query 'Stacks[*].[StackId, StackStatus]'`

Step 2: Create a Jenkins Server

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.

1. Change the current working directory to the root of the cloned repository, and then execute 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`
```

   Note: Do not proceed to the next step until the **Stack Status** shows **CREATE_COMPLETE**. To get the status of the stack type `aws cloudformation describe-stacks --stack-name JenkinsStack --query 'Stacks[*].[StackId, StackStatus]'` at a command prompt.

2. Retrieve the public hostname of the Jenkins server. Open a terminal window and type the following command: `aws ec2 describe-instances --filters "Name=tag-value,Values=JenkinsStack" | jq .Reservations[].Instances[].PublicDnsName`

3. Copy the public hostname

4. SSH into the instance, and then copy the temp password from `/var/lib/jenkins/secrets/initialAdminPassword`.

5. On OS X, use the following command: `ssh -i <full_path_to_key_file> ec2-user@<public_hostname>` For Windows instructions, see Connecting to your Linux Instance from Windows Using PuTTY

6. Run the following command: `sudo cat /var/lib/jenkins/secrets/initialAdminPassword`

7. Copy the output and logout of the instance by typing the following command: `logout`

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 hello-world in the us-west-2 (Oregon) region.

1. Create a ECR registry by running the following command:

   `aws ecr create-repository --repository-name hello-world --region us-west-2`

2. Record the value of the URL of this repository because you will need it later.

3. Verify that you can log in to the repository you created (optional).

Because the Docker CLI doesn't support the standard AWS authentication methods, you need to authenticate the Docker client in another way so 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.

- If you're using OS X, type: `$(aws ecr get-login)`
- If you're running Windows, type: `aws ecr get-login | cmd`

Note: This command will not succeed unless you have the Docker client tools installed on your machine and the Docker Virtual Machine is running. The output should say Login Succeeded.

Step 4: Configure Jenkins First Run

1. Paste the public hostname of the Jenkins server from step 2.3 into a browser.

2. Paste the password you copied from the `/var/lib/jenkins/secrets` directory from Step 2: Create a Jenkins Server (step 2.4) in the password field, and then choose **Next**.

3. Choose **Install suggested plugins**.

4. Create your first admin user by providing the following information:

5. Username: `<username>`
6. Password: `<password>`
7. Confirm password: `<password>`
8. Full name: `<full_name>`
9. Email address: `<email_address>`

10. Choose **Save and finish**.

11. Choose **Start Using Jenkins**.

12. Install Jenkins plugins.

    - In this step, you install the **Amazon ECR plugin** and the **Cloudbees Docker build and publish plugin**. You use the **Amazon ECR plugin** to push Docker images to an Amazon ECR repository. You use the **Cloudbees Docker build and publish plugin** to build Docker images.

13. Log in to Jenkins with your username and password.

14. On the main dashboard, click **Manage Jenkins**.

15. Choose the **Manage plugins** tab.

16. Choose the **Available** tab.

17. Select the **Cloudbees Docker build and publish plugin** and the **Amazon ECR plugin**.

18. Choose **Download now and install after restart**.

19. Choose **Restart Jenkins when installation is complete and no jobs are running**.

Step 5: Create and import SSH keys for Github

In this step, you create an SSH key and import it into GitHub so we can login into Github over SSH.

1. If you're running OS X, open terminal window. If you're running Windows open a Git Bash shell. Run the following command: `ssh-keygen -t rsa -b 4069 -C your_email@company.com`
2. Accept the file location and type a passphrase.
3. Ensure `ssh-agent` is enabled by running the following command: `eval "$(ssh-agent -s)"`
4. Add the SSH key to agent: `ssh-add ~/.ssh/id_rsa`
5. **Note**: If you already have a key called id_rsa, choose another name.
6. Copy the contents of the `id_rsa.pub` file to the clipboard. On OS X you can use the following command: `pbcopy < ~/.ssh/id_rsa.pub`
7. Login to Github. If you don't have a Github account, follow the instructions posted here, https://help.github.com/articles/signing-up-for-a-new-github-account/.
8. In the top right corner of any page, choose your profile picture, then choose settings.
9. In the user settings sidebar, choose **SSH and GPG keys**.
10. Choose **New SSH key** or **Add SSH key**.
11. Type a title for the key.
12. Paste your key in the key field.
13. Click **Add SSH key**.
14. If prompted, confirm your GitHub password.

Step 6: Create a Github Repository (david left here)

In this step you create a repository to store your dockerfile and all its dependencies.

1. Create a repository
2. Login to GitHub.
3. Choose **Start a project** or **new repository**.
4. Type a name for the repository.
5. Choose **Create repository**.

6. Push code to your repository.

7. Open a terminal window (OS X) or a Git Bash shell (Windows).
8. Change the working directory to the root of the `hello-world` repository you cloned earlier
9. Delete the hidden .git directory If you're running OSX, type `rm -fR .git`. Otherwise, type `del /S /F /Q .git`.
10. Reinitialize the repository and push the contents to your new GitHub repository using SSH by running the following command: `git init`

11. Stage your files:

```
git add .
git commit -m "First commit"
```

12.

13. Set your remote origin.
    a. If you are using SSH, run the following command: `git remote add origin 'git@github.com:<your_repo>.git'`
    b. If you are using HTTPS, run the following command: `git remote add origin 'https://github.com/<your_repo>.git'`

14. **Note**: If you created the SSH key for GitHub on your machine, you can use either method. The HTTPS method requires that you to enter your GitHub username and password at the prompts.

15. Push your code to GitHub by running the following command: `git push -u origin master`

16. This project includes a file called taskdef.json. You can view it in the GitHub interface or with a text editor on your local machine. This file is the JSON representation of your ECS task definition.

17. **Note**: You must supply values for the **family** and **name** keys. These are used later in the Jenkins execution scripts. You have to set the value of the **image** key to `%REPOSITORY_URI%:v_%BUILD_NUMBER%`. You will use this mechanism to add the Jenkins build number as a tag to the Docker image.

18. Enable webhooks on your repo so Jenkins is notified when files are pushed

19. Browse to your GitHub repository.

20. Choose **Settings**.

21. Choose **Integrations & Services**.

22. Choose **Add service**.

23. In the search field, type `Jenkins (github plugin)`

24. Enter the public `FQDN/github-webhook/` of your Jenkins server in the Jenkins URL field prepended by your Jenkins username & password.
25. **Note**: If your Jenkins password contains special characters, you will need to encode them using URL escape codes.
26. Make sure there is a trailing / at the end of the URL. For example: `http://username:password@FQDN/github-webhook/`
27. Choose **Update service**.

Step 7: Configure Jenkins

In this step you will create a Jenkins Freestyle project to automate the tasks in your pipeline.

1. Create a freestyle project in Jenkins
2. Login to Jenkins
3. Choose **New Item**, and then type a name for the project.
4. **Note**: Make sure the name does not include spaces.
5. Choose **freestyle project** from the list of project types.
6. Choose **OK**.
7. Under the source code management heading, choose the **git** button.
8. In the repository URL field, type the name of your GitHub repository, `https://github.com/<repo>.git`
9. In **Credentials**, choose the GitHub credentials you create in step 1 of this procedure.
10. Under build triggers, choose **Build when a change is pushed to Github**.
11. Scroll to the build section, and then choose **Add a build step**.
12. Choose execute shell.
13. In the command field, type or paste the following text:

```
#!/bin/bash
DOCKER_LOGIN=`aws ecr get-login --region us-west-2`
${DOCKER_LOGIN}'
```

14. Choose **Add a build step**, and then choose **Docker Build and Publish**.
15. In the **repository name** field enter the name of your ECR repository.
16. In the **tag** field, enter `v_$BUILD_NUMBER`.

17. In the **Docker registry URL**, type the URL of your Docker registry. Use only the fully qualified domain name (FQDN) of the ECR repository you created earlier in Step 3: Create an ECR Registry.

18. Click **Add a build step**.

19. Choose **execute shell**.

20. In the **command** field, type or paste for following text. Be sure to replace the `<ECR_repo>` and `<cluster_name>` with the appropriate values from your environment:

```
#!/bin/bash
#Constants
REGION=us-west-2
REPOSITORY_NAME=<ECR_repo>
CLUSTER=<cluster_name>
FAMILY=`sed -n 's/.*"family": "\(.*\)",/\1/p' taskdef.json`
NAME=`sed -n 's/.*"name": "\(.*\)",/\1/p' taskdef.json`
SERVICE_NAME=${NAME}-service
#Store the repositoryUri as a variable
REPOSITORY_URI=`aws ecr describe-repositories --repository-names ${REPOSITORY_NAME} --region ${REGION} | jq .repositories[].repositoryUri | tr -d '"'`
#Replace the build number and respository URI placeholders with the constants above
sed -e "s;%BUILD_NUMBER%;${BUILD_NUMBER};g" -e "s;%REPOSITORY_URI%;${REPOSITORY_URI};g" taskdef.json > ${NAME}-v_${BUILD_NUMBER}.json
#Register the task definition in the repository
aws ecs register-task-definition --family ${FAMILY} --cli-input-json file://${WORKSPACE}/${NAME}-v_${BUILD_NUMBER}.json --region ${REGION}
SERVICES=`aws ecs describe-services --services ${SERVICE_NAME} --cluster ${CLUSTER} --region ${REGION} | jq .failures[]`
#Get latest revision
REVISION=`aws ecs describe-task-definition --task-definition ${NAME} --region ${REGION} | jq .taskDefinition.revision`

#Create or update service
if [ "$SERVICES" == "" ]; then
  echo "entered existing service"
  DESIRED_COUNT=`aws ecs describe-services --services ${SERVICE_NAME} --cluster ${CLUSTER} --region ${REGION} | jq .services[].desiredCount`
  if [ ${DESIRED_COUNT} = "0" ]; then
    DESIRED_COUNT="1"
  fi
  aws ecs update-service --cluster ${CLUSTER} --region ${REGION} --service ${SERVICE_NAME} --task-definition ${FAMILY}:${REVISION} --desired-count ${DESIRED_COUNT}
else
  echo "entered new service"
  aws ecs create-service --service-name ${SERVICE_NAME} --desired-count 1 --task-definition ${FAMILY} --cluster ${CLUSTER} --region ${REGION}
fi
```

21. **Note**: Before saving this project, make sure that the variable CLUSTER is set to the name you gave your cluster, the REPOSITORY_NAME is set to the name of your ECR registry, and the REGION is set to the region where you created your ECS cluster.

22. Click **save**.

23. Make a change to a file in your repository, e.g. readme.md, and push it to GitHub (from step 6: Create a GitHub Repository, repeat step 2, or use the Github interface). If you've configured things correctly, Jenkins pulls the code from your Git repository into a workspace, build the container image, pushes the container image to ECR, creates a task and service definition, and starts your service.

24. Confirm that your service is running.

25. Log in to the AWS Management Console

26. Under **compute**, choose **EC2 Container Service**.

27. Choose the name of the cluster your created earlier, for example, `getting-started`.

28. On the **services** tab, choose the name of the service your created, for example, `hello-world-service`.
29. On the **task** tab, choose the **RUNNING** task.

30. Under **Containers**, click on the twisty next to the container name.

31. Under the **network bindings**, choose the IP address in the External Link column.

You should see the following image in your browser:

Hello world!

My hostname is 59ff9fcb9fff