Overview

Containers are revolutionizing the way we package and deploy applications. Traditionally, we would install applications on bare metal hardware or virtual machines in our computing environment. We would have to carefully place applications on infrastructure to minimize conflicting requirements such as different versions of shared libraries. Containers let us partition applications and related dependencies on a shared computing platform.

In this week's assignment, you will build a Docker container using a Dockerfile and a CI/CD infrastructure pipeline.

Requirements

You need to have a personal AWS account and GitHub account for this assignment.

The assignment

Let's start shipping those containers!

Build Image Pipeline

It is a common practice to automate the building of Docker images using a CI/CD pipeline. Launch a Jenkins stack in the us-east-1 region using the following template: jenkins-cf.json

Create a new GitHub Classroom repository by clicking on this link:
https://classroom.github.com/a/4TtZ_bGr

Create a new Jenkins pipeline job called `docker-pipeline` and configure it to use a Jenkinsfile located in your Classroom repository.

- Note, you do not need to setup a pipeline build trigger or GitHub webhook for this assignment.

One of the first things the Jenkinsfile should do is tell Jenkins to clone the repository using the `git` command. Let's use the `Pipeline Syntax` link at the bottom of the pipeline configuration page to generate the proper syntax for this command.

- Click on the `Pipeline Syntax` link at the bottom of the pipeline configuration page and select the `git: Git` sample step.
- Type in your git repository URL (ending with `.git`).
- Click the `Generate Pipeline Script` button.
- Copy-and-paste the generated code into your Jenkinsfile.

Your Jenkinsfile will have three stages: `Setup`, `Build` and `Test`. The `Setup` stage will clone a copy of the assignment GitHub repository (using the git code snippet above). It will also copy a webpage file from an S3 bucket into the project workspace. The `Build` stage will use a Dockerfile to build a new Docker image. Finally, the `Test` stage will create a new Docker container using the Docker image and test it to see if it is working properly.

Let's look at the requirements for each of these stages.

The Setup stage

Create an HTML file called `classweb.html` with the following contents:

```
<html>
<body>
  <h1>My class webserver</h1>
</body>
</html>
```

Store this file in an S3 bucket in the us-east-1 region. The `Setup` stage of the pipeline will copy the `classweb.html` file from the S3 bucket into a file called `index.html` in the workspace.

The Build Stage

The `Build` stage will use a Dockerfile to create a new container image called `classweb` with a tag of `1.0`. Here are the configuration requirements for the Docker image:

- The new image must be based off the `nginx` image.
- Set the maintainer of the image to your name and email address.
- The image should expose port 80.
- Copy the `index.html` which was downloaded from S3 to the /usr/share/nginx/html directory in the container.

The Test Stage

The `Test` stage will validate that the Docker image was created properly. The pipeline should create a new container called `classweb1` using the image built in the previous stage. The container should have the following configuration settings:

- The container should run in a detached (daemon) mode.
- The container should map port 80 within the container to port 80 on the host server.
- The container should have an environment variable mapping the key `NGINX_PORT` to the value `80`.

After launching the container, the pipeline should test the container using the curl command. The container is running on the Jenkins slave node, so you should make an HTTP request to the private IP address of that node.

```
curl -s <private IP address>
```

Finally, once the test completes the pipeline should stop and delete the container.

Console Output

Once you have the pipeline working properly, copy the console output from the last successful build into a file called `console.txt` and check it into the GitHub repository.

Check your work

Here is what the contents of your git repository should look like before final submission:

```
├ Dockerfile
├ Jenkinsfile
└ console.txt
```

Terminate application environment

The last step in the assignment is to terminate your CloudFormation stack on AWS.

Submitting your assignment

I will review your published work on GitHub after the homework due date.