edureka!

Automate Software Lifecycle in E-Comm with DevOps

Project Documentation: -

***Note**: The below steps explain how to setup the Project on an Ubuntu Linux machine.*

- After installing Git:
  - Setup the Project Repo on your local system/remote server
- After installing Jenkins, Docker, & Docker-Compose
  - Execute the containers via BASH
- Execute the containers via:
  - Jenkins Pipeline script
  - Jenkinsfile (stored in SCM)

Step 1: Install Git

After installing Git, you should pull all the Project code & libraries. The PHP code, Database structure, Docker Compose file and Jenkinsfile can be found at Github repo: https://github.com/VardhanNS/phpmysql-app

To Clone the repo at your local system, run the below command in a new Directory:

```
$ git clone https://github.com/VardhanNS/phpmysql-app
```

Once the command is executed, there will be a local copy of the files for your reference. You can either initialize a new Git repo and copy the contents there or simply use the above shared public repo.

Step 2A: Install Docker, Docker-Compose, Jenkins

Step 2B: Execute Containers via Bash

The idea here is to execute Docker containers locally once just ensure the working before we execute it via the Jenkins CI server. Execution of the containers is done by executing the Docker-compose file.

You can find the Compose file by the name of 'docker-compose.yml'. Goto the directory where this file is present and run the below command from the terminal:

```
$ docker-compose build
```

The above command will build the Image along with other components into a Container so as to make the container executable, but it wouldn't have executed it yet. You should get a confirmation on the terminal about the various steps in the build process that has been successful.

The next step is to run the Docker container which we have just built. Run this command:

```
$ docker-compose up
```

Note: You can add a flag '-d' to the compose up command if you want to run the container in detached mode. This is typically done when executing via Jenkins server. But the flag is not a mandate when we run this command via BASH.

After a few seconds, 2 Container should be up and running. One would be a PHP container and the other would be MySQL container. To verify that both containers are running, run this command:

```
$ docker ps
```

The PHP service is running on port number 8008 in your localhost as defined in the compose file. Open your browser, and hit the URL: localhost:8008 to see the **E-Commerce Application** software running at your system.

Note: Before you build/execute the container the next time, destroy the already running Containers. This is because whenever we execute the container, it will be started with the same name (As mentioned in our compose file. You can change these settings if you wish to).

An important aspect of this Project is you pushing your custom docker image to the docker hub. Create a Dockerhub account and a new Docker Repo if you haven't one already. Run the below commands to establish a connection between your local system and the docker hub.

```
docker login --username=<yourhubusername> --
email=<youremail@company.com>
```

On executing this command, you will be prompted for the password too. The next step is to add a tag the Image you built and then push it to docker hub. The image ID of the image you build is the Image ID displayed when you ran the docker-compose build command.

```
$ docker-compose build
```

Now to rename the Image and add a tag, you have to run the command:

```
docker tag bb38976d03cf yourHubUserName/yourRepoName:imageTag
```

Replace the variables mentioned in the above step according to your needs. The next step is to push the image to the docker hub with command:

```
docker push yourHubUserName/yourRepoName
```

Step 3A: Execute Containers via Jenkins Pipeline Script

After installing Jenkins, open the Jenkins server by going to localhost:8080. This is the port number where Jenkins runs by default.

Click on 'New Item' on the Jenkins Dashboard → Give a name to your project → Choose 'Pipeline Project' and then click on 'Ok'.

Now under the 'PIPELINE' tab, set definition as 'Pipeline Script' and start writing the Groovy script into the code box. You can find the same code in the GitHub repo under the filename 'Jenkinsfile'. Copy and paste the entire code. The code here consists of the steps to perform the container operations we did manually via the BASH in our previous step.

You need to go through the file and make changes according to your needs. (maybe change the path, etc) You can also use the 'Pipeline Syntax' builder to generate the Groovy code if you are working on Groovy for the first time. You can generate individual lines based on your need.

There are various steps which Jenkins will perform via this script namely:

1. Pulling code from GitHub repo
2. Executing the docker containers in the shell of the host
3. Pushing the docker image to the DockerHub

Click on Save & Apply the changes to the Project. Now go to the Project Dashboard and click on 'Build now' option on the left menu. This will execute the steps in the script one after the other and finally your container will be active and running.

Step 3B: Execute Containers via Jenkinsfile

The same process we did in Step 3A can be done in a further automated fashion. Using a Jenkinsfile, we can store the Pipeline Script which we want to run and provide the Jenknsfile path to Jenkins server instead of writing the script manually to build jobs.

Perform the same steps as mentioned in 3A, with the exception in the Pipeline definition. Instead of choosing 'Pipeline Script', choose 'Pipeline script from SCM'. Then you will have to specify the URL of the remote GitHub repo where the code is present followed by its credentials if its a Private repo. Click on save and Build the project.

By now, the E-Commerce App would be running on port no 8008 of your localhost. You can verify the same.

Congratulations DevOps Engineer!

The E-Commerce App is now, up and running!