Development Operations
Mini Project

COURSE CODE: CS423 INSTRUCTOR: ISRAR AHMAD

CI/CD Pipeline with Git/Github, Jenkins and Docker

This project to build Continuous Integration and Continuous Delivery (CI/CD) pipeline of *containerized applications*, an automated approach to software development. You'll set up CI/CD pipeline using the popular tools and Docker and see the benefits they bring to the DevOps pipeline. This will be a group project and each group should consist of ***4 people*** **only**.

Instructions

Generally, there are four types of environments in the software development process such as Development, Testing, Staging and Production. The **Development environment**, also known as Dev, is where the software develops, and all the updates take place at first. The software in this environment is usually updated with all changes when developers commit to the Dev branch. Once the developer is satisfied then he/she commits changes to the testing branch which deploys all updates on **Testing environment** for QAs to test new and changed code whether via automated or non-automated techniques. Thus, testers can ensure the quality of the code by finding any bugs and reviewing all bug fixes. However, if all tests pass, the test environment can automatically move the code to the next deployment **staging environment**. This environment is nearly the same as the production environment to ensure the software works correctly. The target here is to test the complete software and observe how it will look on the production server. Finally, after all tests, the changes are deployed to the **Production environment** for end users.

You as DevOps engineer are asked to automate this process from development to deployment of software on these environments.

Task 1: Selection and deployment on localhost of any simple open-source application.

1. Clone the following simple open-source application from GitHub built using ReactJs and NodeJS.
   a. **ReactNodeTesting app (https://github.com/eljamaki01/ReactNodeTesting)**
   b. You may also clone any simple open-source application built using any of the following technologies
   i. Frontend: React or Angular ii.
   Backend: NodeJs, Python, Java, PhP.
2. Carefully follow all the instructions inside the Readme file to deploy an application on your machine for a demo.
3. As you and your team will be making further changes to this application so you must initialize this cloned repository code to your new repository and push it to your GitHub repository. Add other team member as a collaborator to your GitHub project.
4. Update your repository settings such that no collaborator should be able to commit directly to the main branch.

5. You must put your repository privately and add me as a collaborator to that repository.

6. Deploy both frontend and backend services on separate **containers** named *frontend* & *backend*. Create a separate container for **the database** if it is being used by your application.

Task 2: Create three EC2 instances on AWS for Dev, Testing and Staging environment.

1. Give an appropriate name such that it represents environment.
2. Each environment Amazon Machine Image (AMI) should be Ubuntu Server 22.04 LTS (HVM)
3. All instances must share one common security group such that if you add/remove any rule, it should affect all instances.
4. Configure your environments such that it includes required dependencies for your React and Nodejs project.
5. Add or remove rules in your security group where required.
6. As you will be working in a group you may create firstAWS instance in one account and second in other account.

Task 3: Automating application from development to deployment. This time you must use ***declarative script*** to create pipeline jobs.

NOTE:

A. You must deploy application services in a docker container in each environment.
B. *Frontend* and *backend* should run as a service in a separate container. Create a separate container for **the database** if it is being used by your application.

2. If the developer pushes changes to the *dev* branch, then Jenkins must fetch from *dev* branch and deploy it on *Dev* or *Development* environment.

3. If you as a developer are satisfied, then commit changes to the *testing* branch.

4. If the developer pushes changes to *the testing* branch, then Jenkins must fetch from *the testing* branch and deploy it on to the *Testing* environment.

5. Use any testing tool for testing and trigger a job in Jenkins to merge changes into *master/main* branch.

6. If the changes are pushed to *master/main* branch, then Jenkins will fetch *master/main* branch and deploy on the *Staging* environment.

What to Hand In

1. All changes should be committed to your repository
2. Submit the zip file of your Jenkins backup.
3. Submit the zip file of your Jenkins scripted pipeline.
4. Submit the word document that includes
   A. Screenshots of deployed application after cloning
   B. Screenshots of files you have created to deploy services in a container
   C. List down any challenges you have faced while deploying
   D. Screenshots of the scripted pipeline
   E. Screenshots of the jobs failed to build (at least two)
   F. Screenshots of the jobs build successfully (at least two)
   G. Screenshots of the pipeline build successfully