CI/CD Pipeline with Docker

Project ID - 2, Team ID - 13

Team Members

201301095 - Kushal Singh
201302161 - Aaditya M Nair
201405529 - Atul Rajmane
201201045 - Abhishek Bhat

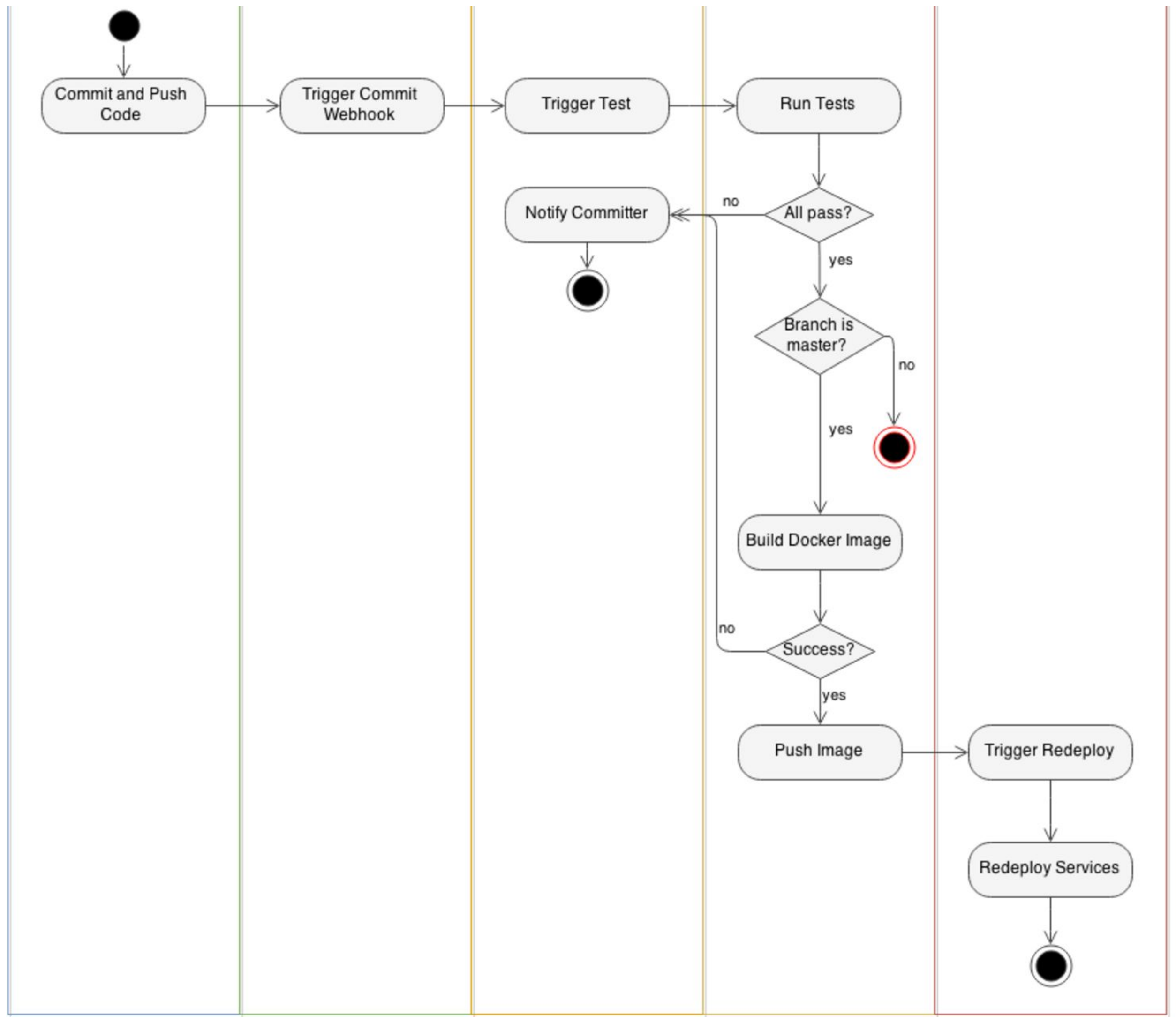
Overview

The basic idea for the project is to try to create a build system which will use Docker for implementing continuous integration (CI) / Continuous Deployment Pipeline. A git based commit should be used for starting of a build for a docker image which would then be run and provisioned in a Virtual Machine. After every commit a series of test cases is then run on the code to ensure the correctness of the code. After all the test-cases pass, the image gets updated on docker-hub registry, and a VM gets provisioned which can then run the software directly (after pulling the image from the docker-hub).

This entire process ensures that the most recent and updated version of the code is available to the person who is using the software and this speeds up the overall process by at least 2-3 folds.

Approach

We plan to use a docker based solution for this problem of ensuring fast continuous delivery and deployment. Current CI/CD solutions follow a multi-tiered environment approach - where each of the environments is managed differently from the others. The use of docker based approach eliminates the system and language conflicts by isolating the things in a containers. The use of container will ensure a smooth integrated environment throughout. It will also make the process more streamlined and faster. For every push a webhook will notify of the changes in the code. For every such notification the DockerFile and the code will be pulled and a docker image will be build. Tests will be run on the slave nodes and if all the test-cases pass, then a VM will be provisioned for the same. This approach utilizes the fact that containers are lightweight and faster compared to other solutions, and a Docker container can be easily moved from one place to another with minimum overhead and the containers can then be run directly without any hassle.



High Level Design

On a very high level, this whole application contains three different moving parts:

1. **Git**
   Git will be controlled by the use of Git-Hooks. Whenever a user commits some code, a code will be called which will contact the CI server to start the tests and to inform the main pipeline process of a commit. The user can specify in the commit message whether or not the tests are to be run on this particular commit.

   The following files need to be present in the git repository:

1. The Tests File
2. The Build Config File
3. The Dockerfile

2. **CI server**
   The CI server will receive the job and execute it and it will report the status of the test to the main pipeline app. The next steps will be based upon this status of the job.

3. **Docker**
   Once the tests are successful, the project will be converted to a docker image and uploaded to Docker Hub.
   If on the other hand, the build is unsuccessful, the user is notified and no Docker image is formed.

Project Plan

1. **Configure GitHub**
   Set up a GitHub repository with the contents listed below and configure it to notify the Jenkins server when a new commit happens. That is via the “Webhooks and Services” section of the “Settings” page.
   - Application source
   - The test code for the application
   - A Dockerfile that describes how to build the application container, and copies over the necessary application and test code.

2. **Configure Jenkins Master node**
   Once a node is set up and the base Jenkins image is installed, and the service is up and running the GitHub Plugin needs to be installed on the Jenkins master. This plugin allows for a Jenkins job to be initiated when a change is pushed to the concerned GitHub repository.

3. **Configure Jenkins Slave node**
   Set up a node that runs a Jenkins slave and create a Jenkins job on the same to test the application.

4. **Configure the target node**
   Set up a node for the target application and provision the application on the same.

Requirements

The following software need to be present on the system running this app

1. *Git*
2. *Docker.*
3. *Libvirt*
4. *MongoDB*
5. *Python with the following libraries:*
   A. Flask
   B. PyMongo
   C. Docker

Expected Outcome

There are two possible outcomes - success and failure.

- Success - This status tells that all the test cases ran successfully. This also triggers the latest image push to Docker Trusted Registry and the deployment of the application on the target node.
- Failure - This status tells that some/all of the test cases failed.

In both the cases a notification email will be sent to all the contributors of the project. Of course, the email contents will vary based on the status.