Project 2 - CI/CD Pipeline Using Docker

Team number - 13

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

Basic Idea

Develop a build system that leverages Docker for implementing continuous integration/deployment(CI/CD) pipeline. A git commit must kick off packaging a Docker Image and provisioning it in a VM.

The Problem and current solutions

Code bases for projects are generally maintained with a separate branch for each of the features. On such technical projects many developers work simultaneously with each of them making tens of code pushes to the relevant feature branches on an average day. There are already good tools available to auto-test the feature branches e.g. Travis-CI for GitHub. The real problem creeps in, aptly termed as "integration hell", when the developers merge their code into develop or master branch. This becomes even more cumbersome when integration tests need a running environment.

Current CI/CD solutions follow multi-tiered environments approach - development, test, staging and production. Note that each of these environments are managed independently of each other. Hence, each of these environments may have different
configurations different library versions or even different Operating Systems. This leads to the popular problem known as *"it works on my machine"* syndrome where an application that works on one environment stops working on some other due to the above mentioned problems.

Some other issues are enlisted below:

- Difficulty supporting diverse language stacks and tooling
- Slow provisioning and setup of build and test environments
- Low throughput of jobs and software shipped to stage or production
- Inconsistencies between environments

Solution using Docker

A docker based solution has the following advantages over the traditional ways:

- Eliminate system and language conflicts by isolating in containers
- Run more jobs faster
- Ship more software
- Standardized yet flexible environment

A docker based solution ensures that CI/CD happens quickly and after each commit the code gets tested. 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.

Project scope

The following steps draw a picture of the project scope

1. Whenever there's a code push, a webhook from GitHub notifies Jenkins of the update.
2. Jenkins fetches the GitHub repository, including the Dockerfile describing the image, as well as the application and test code.
3. Jenkins builds a Docker image on the Jenkins slave node.
4. Jenkins instantiates the Docker container on the slave node, and executes the appropriate tests.
5. If the tests pass then the image is pushed up to Docker Trusted registry.

6. A VM gets provisioned with the most recent and updated image running on it.