Specification Document

For

Continuous integration / continuous deployment using jenkins

Version 0.0.1

Prepared by Soumyajit Basu

18th July 2020

Table of contents

Introduction

This is a brief documentation for the continuous integration and continuous development process. There are two explanations in the documentation. The first is the traditional way to chain multiple jobs to form a pipeline and the second is pipeline as code which would help define the steps of the process as part of the code.

What is continuous integration?

Continuous integration (CI) is **the practice of automating the integration of code changes from multiple contributors into a single software project**. It's a primary DevOps best practice, allowing developers to frequently merge code changes into a central repository where builds and tests then run.

What is continuous deployment?

Continuous Deployment (CD) is **a software release process that uses automated testing to validate if changes to a codebase** are correct and stable for immediate autonomous deployment to a production environment.

Problem statement

- Availability of the application and its versions in the GitHub
    - Track their versions every time a code is committed to the repository
- Create a Docker Jenkins Pipeline that will create a Docker image from the Dockerfile and host it on Docker Hub.
- It should also pull the Docker image and run it as a Docker container.
- Build the Docker Jenkins Pipeline to demonstrate the continuous integration and continuous delivery workflow.

Tools and technologies

The tools and technologies that are used to solve the problem statement are **Docker**, **Git**, **Jenkins.** The primary tool being Jenkins as this would act as an intermediary to automate the building, packaging and deploying the software.

Docker

Docker is a set of platform as a service (PaaS) products that use OS-level virtualization to deliver software in packages called containers.

Github

Github is a version control software that plays an integral part in the devops process to maintain release versions.

Jenkins

Jenkins is an open source automation server which enables developers around the world to reliably build, test, and deploy their software.

Traditional approach for Jenkins

The traditional approach to creating a continuous integration / continuous deployment process in Jenkins is to chain multiple jobs required to automate the process of building, packaging, validating and deploying a software. To this we need to add a post-build action which would signify to not run the next job if the previous job is failed. These are configured in the form of upstream and downstream projects. Project referring to each job to be put in the queue.

To create jobs first select the option **free style project**. Now under source code management provide the repository url. Specify the respective branch name. Under build configuration use the option execute shell. In the shell provide the respective shell scripts to run. Click on apply and save. Likewise multiple projects can be created where each project would be a single task that can be chained together as part of the pipeline. When a job is chained in the project configuration there is an upstream project and downstream project. **Upstream** project refers to the task which needs to be triggered first and executed. If the job gets executed successfully then the immediate process gets executed. **Downstream** project refers to the intermediate job that would be triggered if the current job is executed successfully. To set a **downstream** job you need to configure the build project and enter the job you need to chain under **post-build actions**.

Downfall of the traditional approach

According to my analysis there are two downfalls of this approach. This approach lacks a dynamic approach to the ci-cd genre. First of all a lot of re-engineering is required to create shell scripts for every build process to trigger. Secondly, if we do not create shell scripts and make it an integral part of the jenkins job, then corruption of the jenkins application may lead to data loss. Also, it would add to a lot of efforts creating shell scripts to trigger builds within jenkins.

Pipeline as code

Pipeline as code gives the added flexibility to configure required jobs as part of the pipeline configurable as code. For that we need to create a Jenkinsfile as part of the application. Write the respective configuration as part of the Jenkinsfile. When executing, Jenkins can automatically configure each job dynamically, executing the job based on the configuration in the jenkinsfile. To run a pipeline, create a pipeline project in jenkins. In the pipeline project configure the pipeline SCM as git. Enter the repo url and configure the branch. Enter the jenkinsfile script path. Click on apply and save.

Note:

1. In case if we are using docker, to deploy the image to dockerhub we will need to configure the credentials of docker hub. To configure the credentials of docker hub go to *manage jenkins > manage credentials > jenkins > global credentials > add credentials*.

2. Also if we are running docker as an agent, then we need to add docker as a node which can be done using *manage jenkins > manage nodes and clouds > master > add label docker*

Resources

- Project used - github project
- Jenkins file - Jenkinsfile configuration
- Docker file - Dockerfile configuration
- Basic pipeline configuration - Basic pipeline setup in Jenkins
- Pipeline as code - Pipeline setup in jenkins

Project requirements

- Availability of the application and its versions in the GitHub
    - Track their versions every time a code is committed to the repository
- Create a Docker Jenkins Pipeline that will create a Docker image from the Dockerfile and host it on Docker Hub
- It should also pull the Docker image and run it as a Docker container
- Build the Docker Jenkins Pipeline to demonstrate the continuous integration and continuous delivery workflow

Company goal is to deliver the product frequently to the production with high-end quality.

The project should use the following:

- Docker: To build the application from a Dockerfile and push it to Docker Hub
- Docker Hub: To store the Docker image
- GitHub: To store the application code and track its revisions
- Git: To connect and push files from the local system to GitHub
- Linux (Ubuntu): As a base operating system to start and execute the project
- Jenkins: To automate the deployment process during continuous integration