Jenkins Pipeline to deploy Angular App to AppEngine GCP

vijay kumar
Nov 20, 2018 · 10 min read

This post is all about creating a Continuous Deployment in Jenkins for Angular App to GCP. This could also be used as a reference to deploy the Nodejs apps to GCP since most of the stages are similar with language specific commands for building, packing and deploying the app

Table of Contents

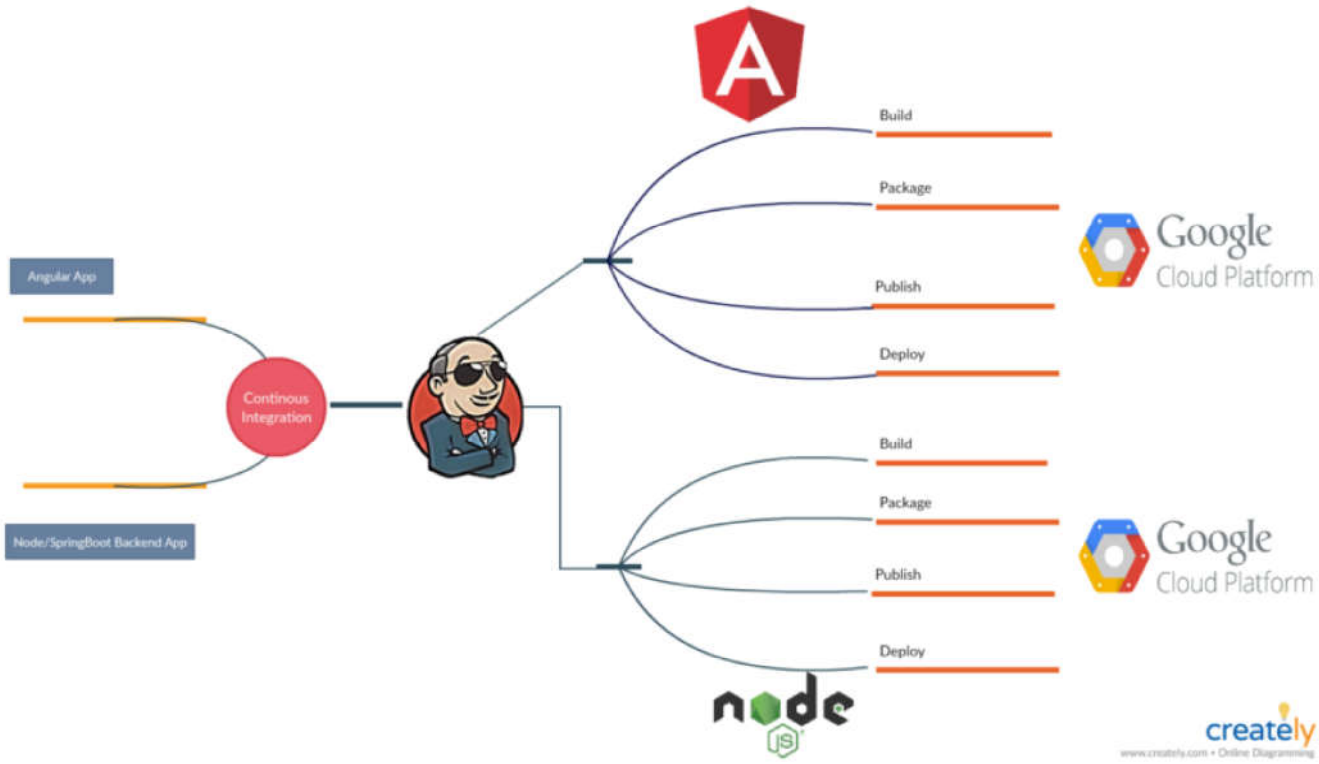
Table of Contents

- A Brief Introduction:
- Why GCP?
- Install Jenkins
- App and Repo Creation
- Pipeline in Jenkins
- * Click on the new item
- * Create a sample pipeline project
- * Describe the pipeline
- * Pipeline script
- CI in Jenkins
- Jenkins Plugins
- Configure Global Tools in Jenkins
- Configure Credentials in Jenkins
- Configure Systems in Jenkins

- Declarative Pipeline
- Understanding Declarative Pipelines
- Initial Declaration
- Stages in Pipeline
- Declaring Deploy stage

A Brief Introduction:

Jenkins is one of the most widely used tools to develop Continuous Integration and Continuous Delivery pipelines. The need for CI/CD cannot be overstated, since the advent of developing agile based applications. That doesn't mean that it should not be used for the more traditional waterfall model though.



Continuous Integration is for continually triggering the phases such as build/report/publish/package after every code commit and package the code ready for deployment. This is mandatory since every time when a piece of code is checked in your repo , say bitbucket, every time a build should be triggered and the app should be bundled.

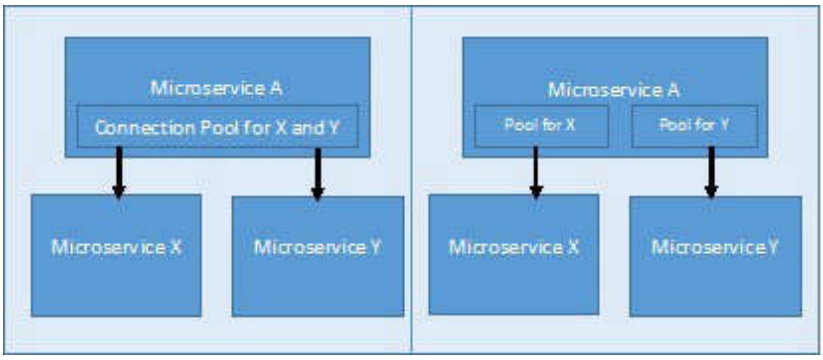
For the QA team to test the applications, they should be available in the sandbox at all times. It is really inefficient to manually deploy the applications to the sandbox/

test environments every time when a patch is made or a bug fix is done.

Enter Jenkins!!!. The world of CICD has been made simple with the introduction of pipeline items, where different phases can be automated to happen after each successful completion of a stage (Build/Package/Publish/Deploy)

Why GCP?

- Micro-services architecture emphasises that each independent modules be hosted as an independent application. GCP supports the micro-services architecture by default where we can have a single project embed multiple applications (which are language agnostic).



- The AppEngine part of GCP supports automatic scaling based on the request throughput and usage
- Kubernetes is supported natively in GCP
- GCP also makes a developers life much easier by allowing the admin to . SSH into the remote machine and deploy manually.

There are many more advantages like debugging, logging which are supported natively as well as support for greater set of analytic tools

Install Jenkins

The article in the link explains about installing and using Jenkins in a macOSX machine. In a windows or Unix/Linux based machines use the respective package managers to fetch the app. Please follow this link to either startup Jenkins as a standalone or behind a NGINX reverse proxy

App and Repo Creation

I am not going to give an introduction on creating the angular/nodesjs/springboot apps and push the code to a remote repo as it is beyond the scope of the article. There are numerous online articles for that . Here are some of the references

Angular App reference:

A Quick Guide to Help you Understand and Create Angular 6 Apps

This post is divided into two parts:

medium.freecodecamp.org

Nodejs App reference

Building a simple REST API with NodeJS and Express.

Have you been working on front-end technologies and been feeling like you're missing out something in the whole...

medium.com

BItbucket references

Learn Git with Bitbucket Cloud | Atlassian Git Tutorial

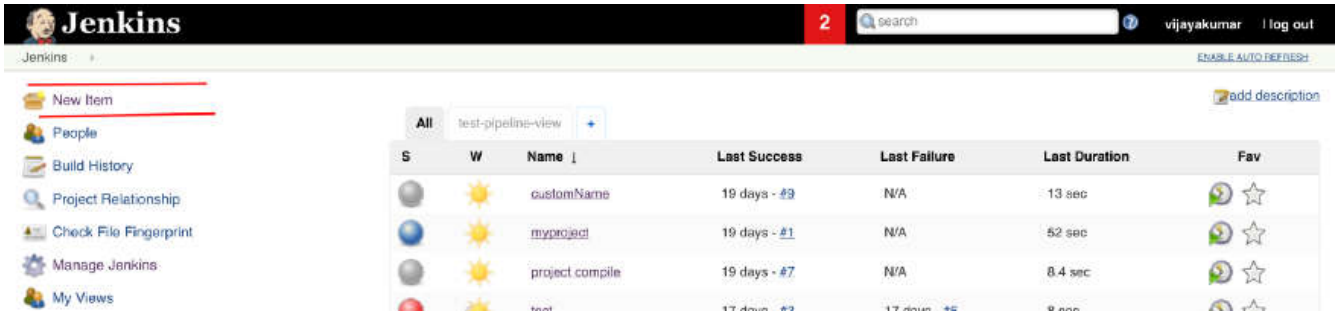
Learn Git with Bitbucket Cloud

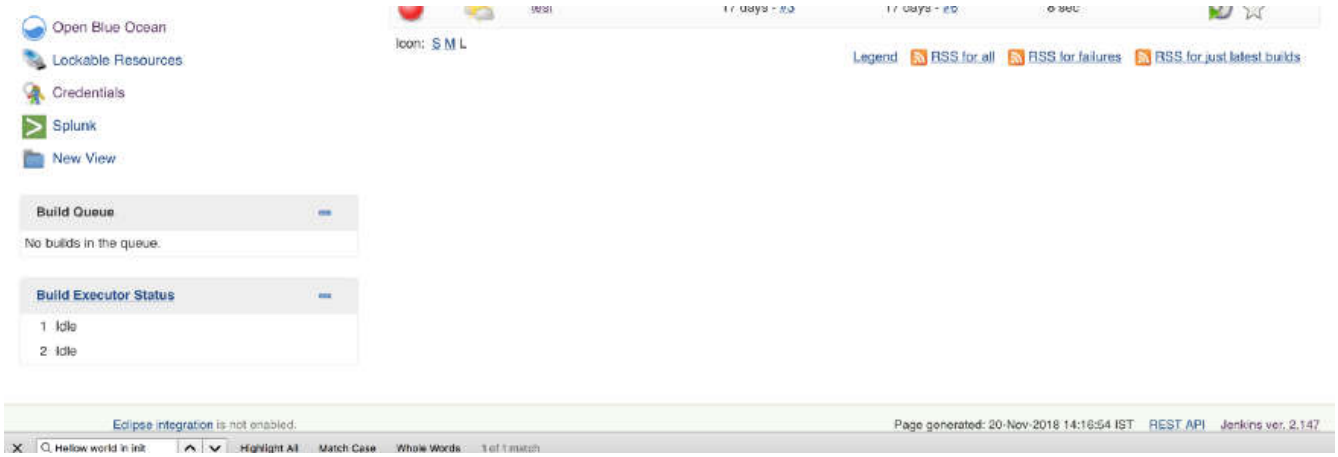
| Atlassian Git Tutorial Learn Git with Bitbucket Cloudwww.atlassian.com

Pipeline in Jenkins

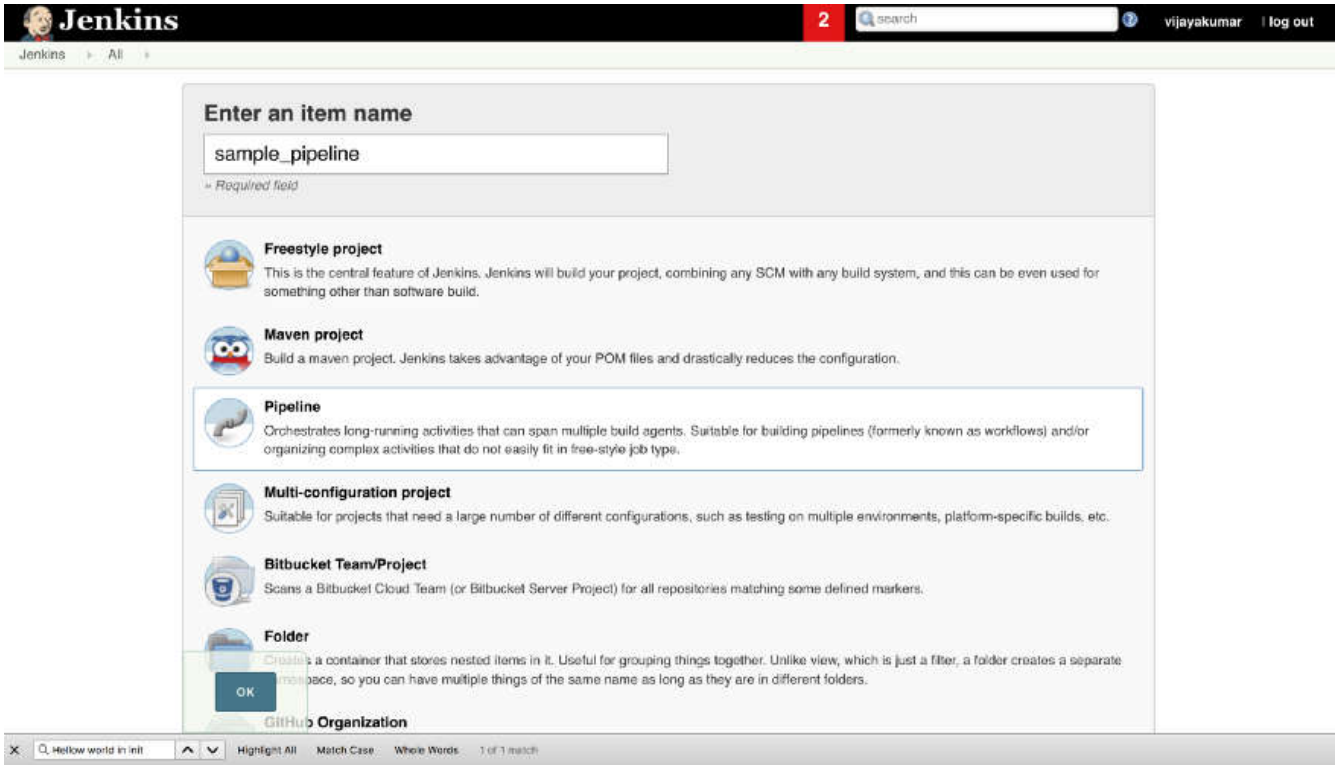
Let us create a sample pipeline project in Jenkins

* Click on the new item

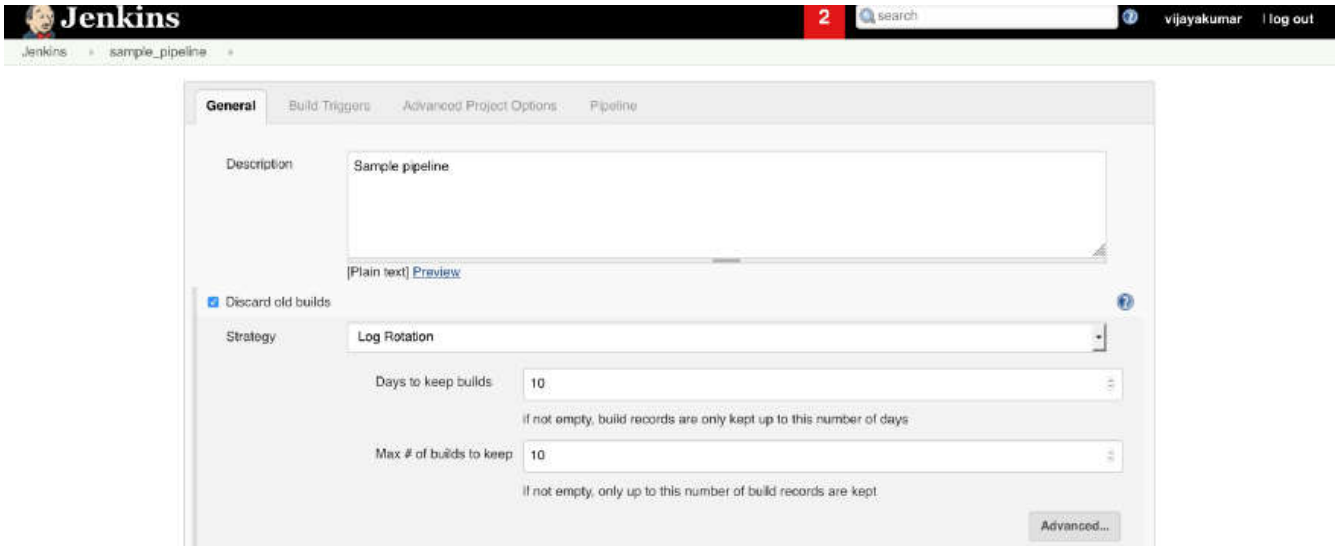




* Create a sample pipeline project

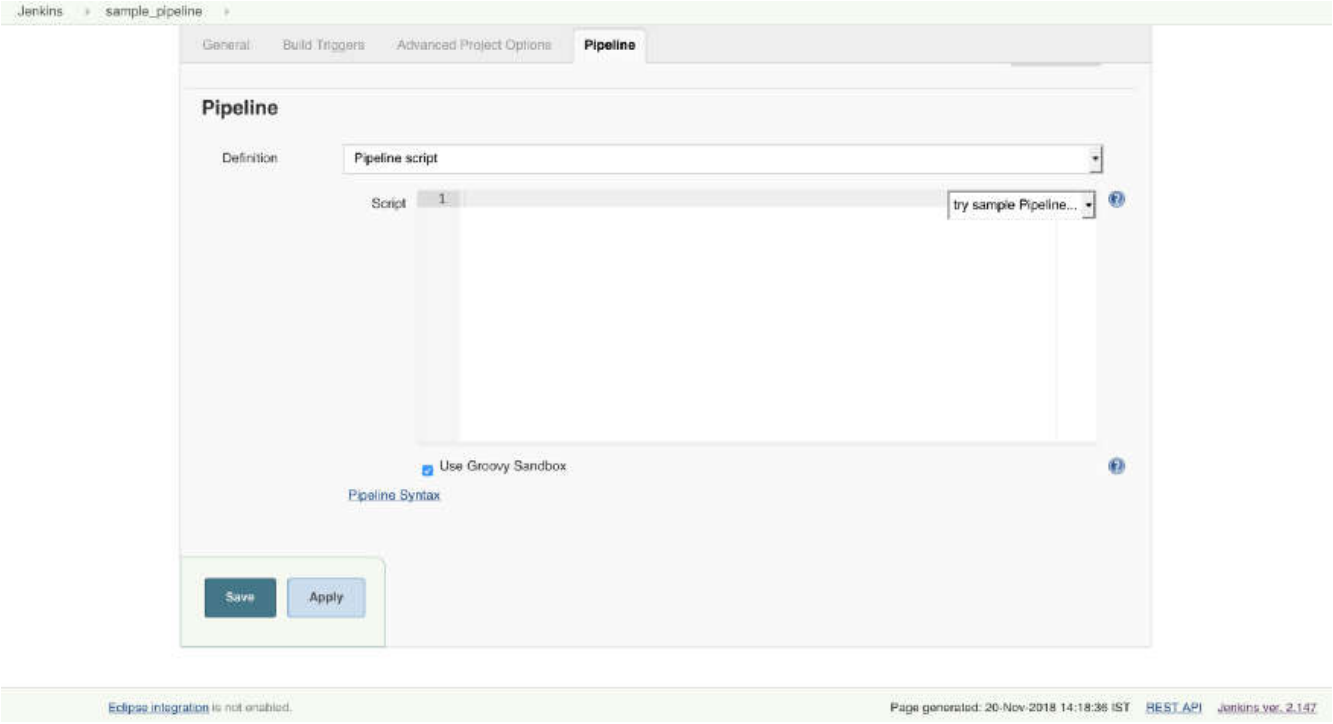


* Describe the pipeline

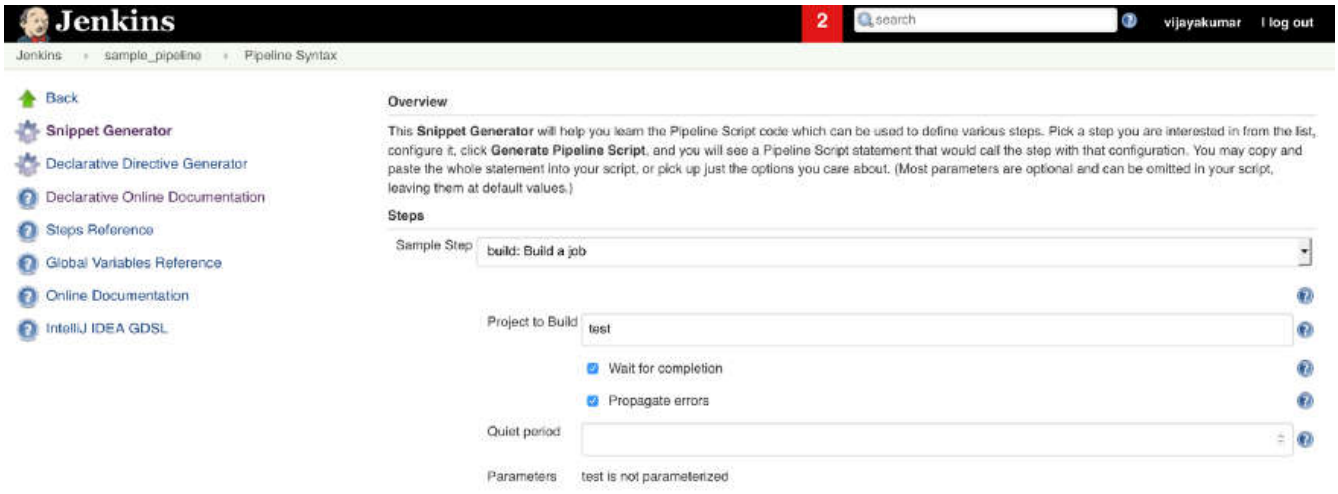


Configuring the ‘Discard old builds’ is optional

* Pipeline script



Now this is where scripting comes into picture. The user can select the ***Definition*** of the pipeline to either be a ***Pipeline script*** or a ***Pipeline file*** that is stored in SCM



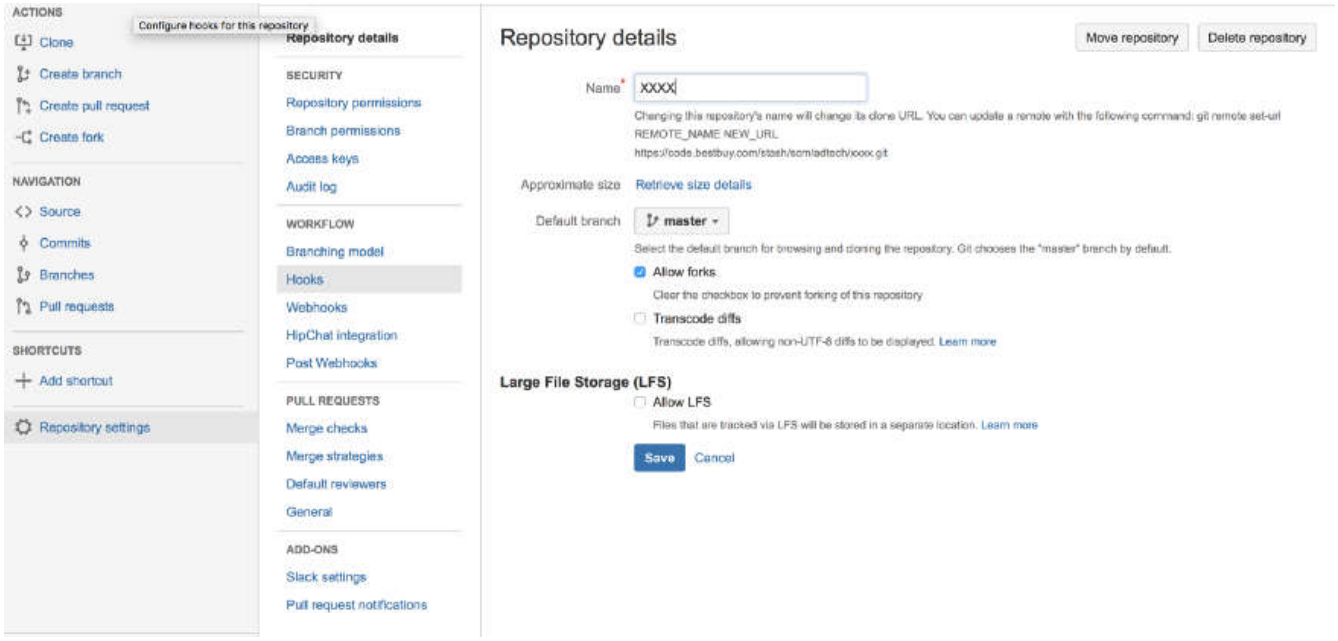
Snippet Generator

Here we are going to look at Pipeline declarations that are and executed directly from SCM. This eliminates the need for maintaining a separate space in Jenkins for the configuration file. The declaration pipeline is also much easier to configure.

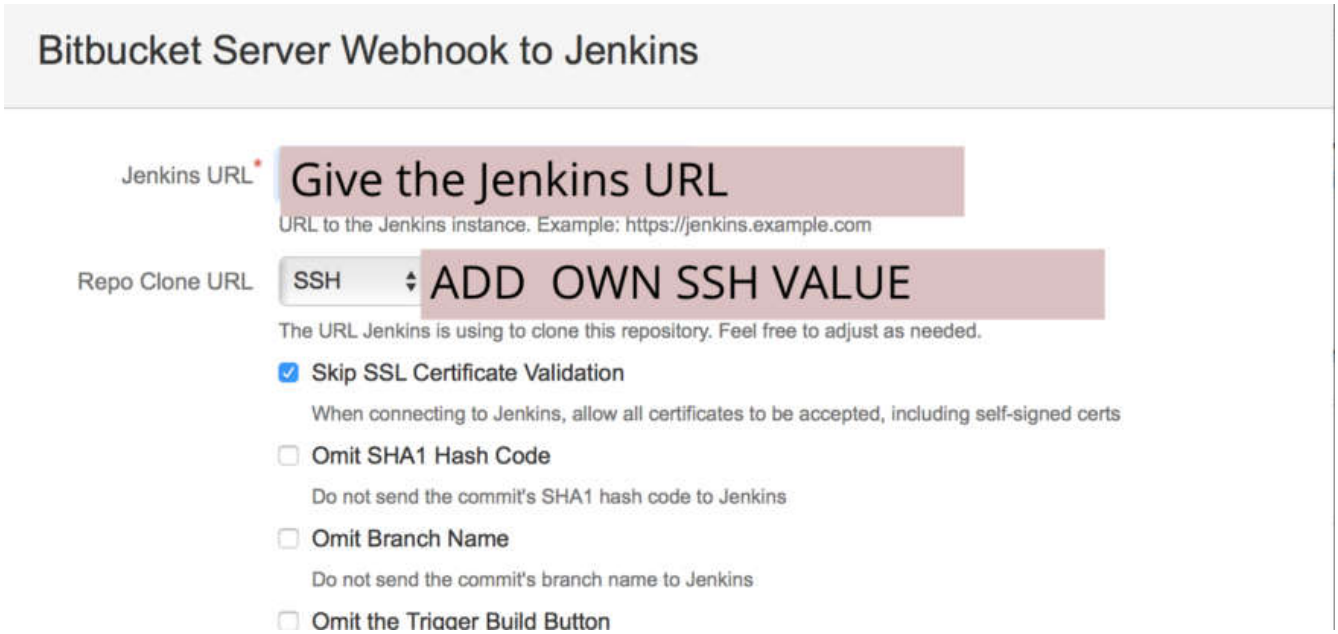
CI in Jenkins

Continuous Integration is made possible in Jenkins by means of enabling Web-hooks.

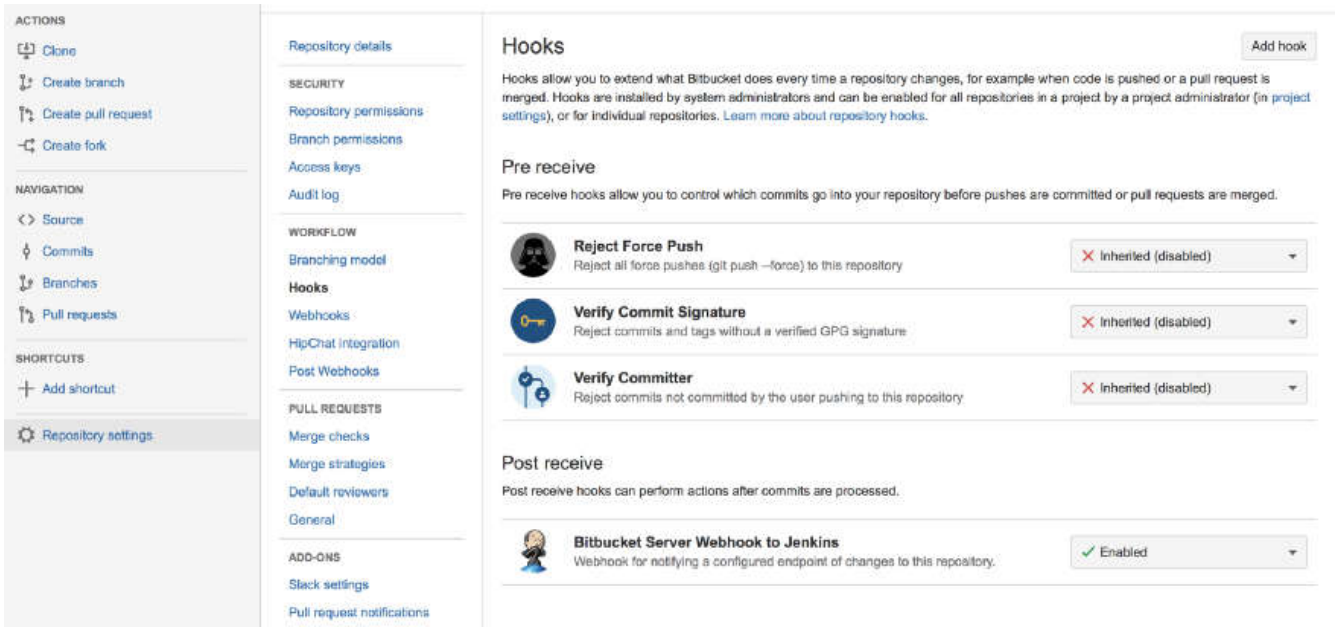
- Allow the webhooks plugin for the bitbucket server to be installed in your bitbucket tool and then go to the repository settings



- Now go to the *Hooks* section and enable the **Bitbucket Server Webhooks to Jenkins**

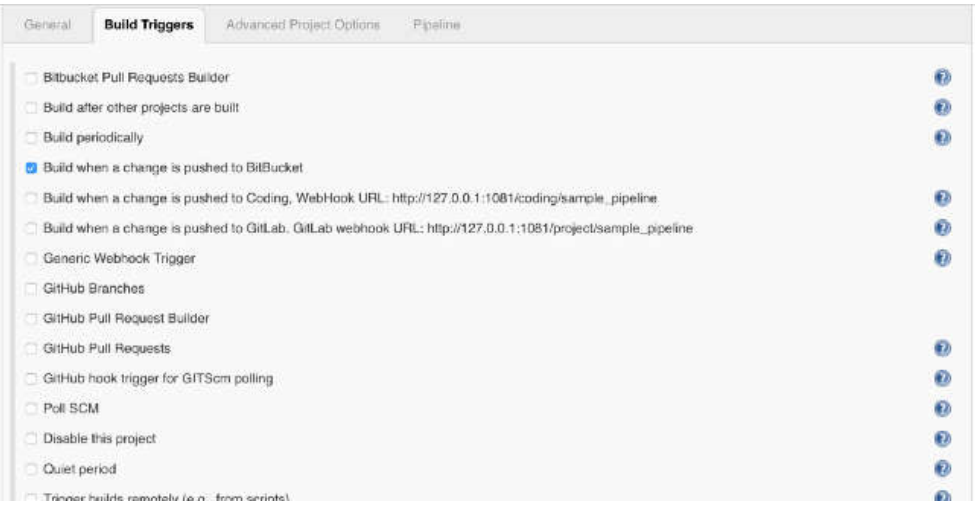


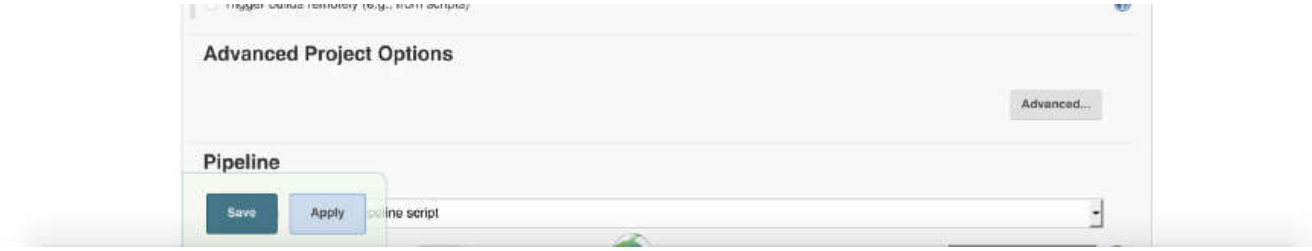
- Once the webhook is enabled , it gets reflected in the repo



> *Webhooks are nothing but functions that run on the web when something is triggered. In our case, we would want the Jenkins pipeline function to be triggered when we push the code to SCM, each time.*

- Now go back to Jenkins and enable the **Build Trigger**



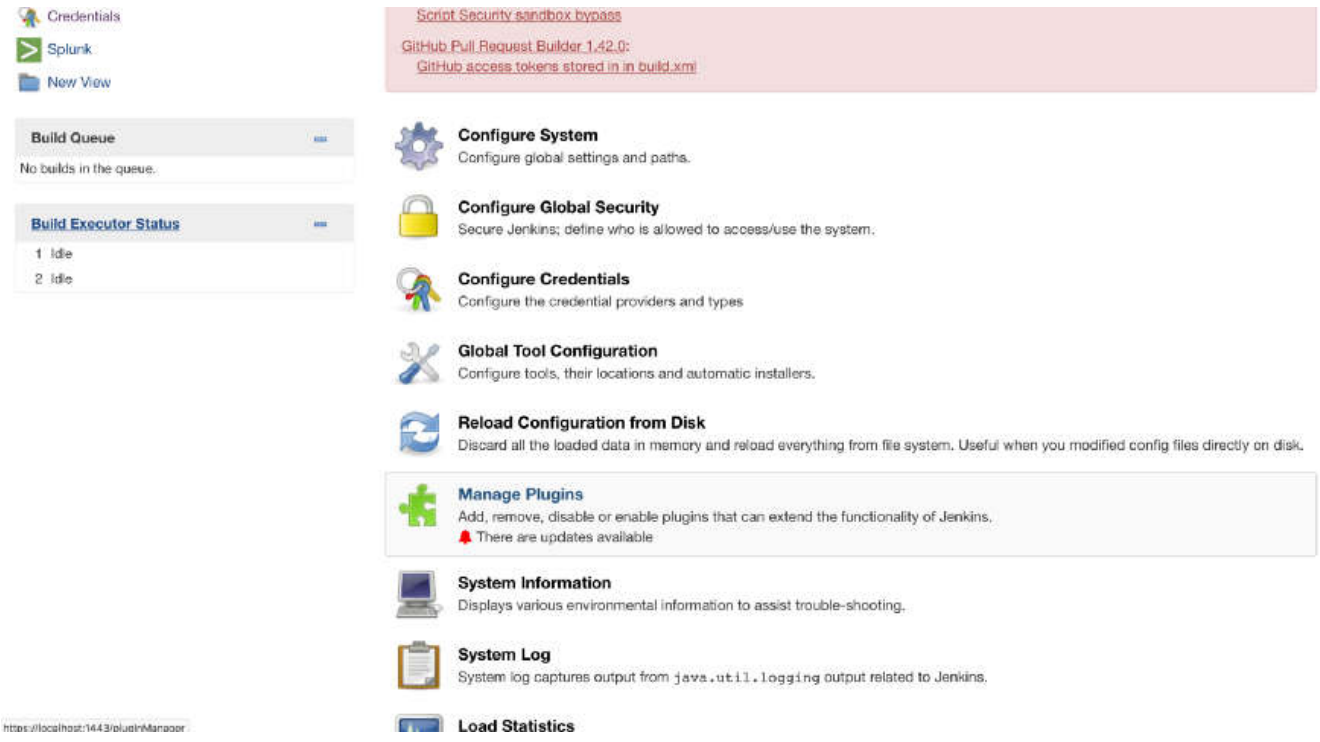


Jenkins Plugins

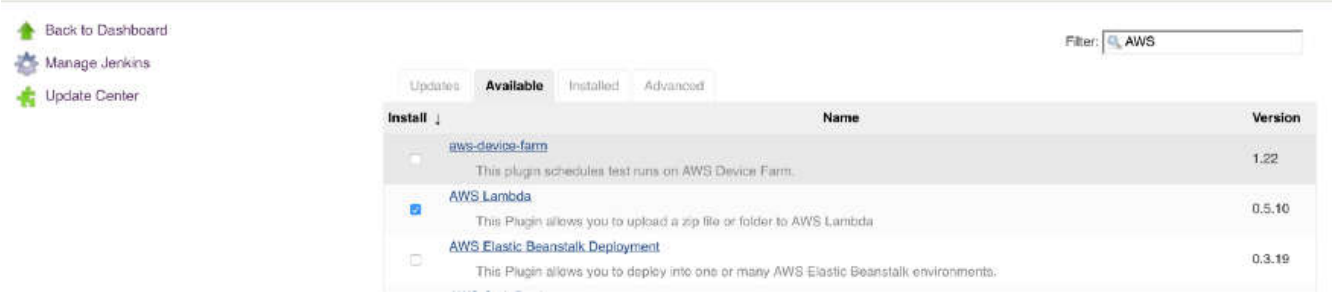
There will be cases when some of the options in Jenkins may not be available by default. So proper plugins need to be installed for every function.

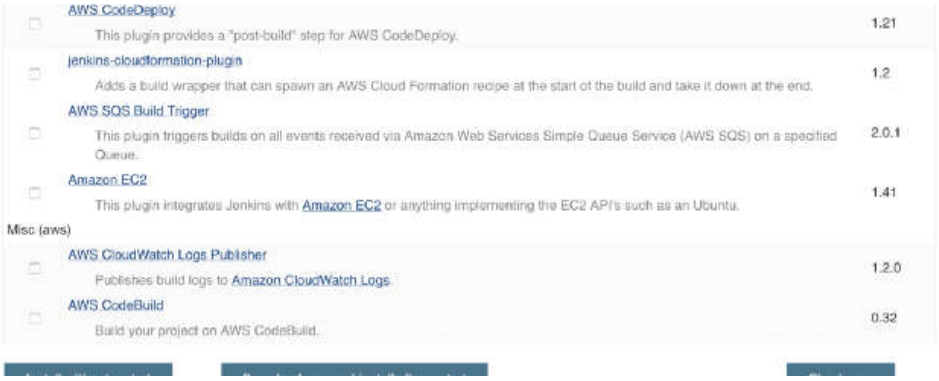
> *For example if build is to be triggered on every push to bitbucket, then Bitbucket plugin should be installed first. Here is a gist of all the plugins that I happen to install. Some of the plugins may not be needed for your configuration.*

- Go to *Manage Jenkins-> Manage plugins*



- Go to *Available -> Search* the plugins needed and install them. Make sure you select *Install without restart* every time you wanted a plugin to be installed





Here is a gist of all plugins that I installed

Configure Global Tools in Jenkins

These are the building blocks of the pipeline allowing the user to execute build/publish and deploy commands. We are going to make use of three major tools — ***Java, Git & Node***. Since Jenkins runs from local, it is important that the local machine has the necessary tools installed in it. The other option is that we can instruct Jenkins to download the tools over the internet.

- Go to *Manage Jenkins -> Global Tool Configuration*



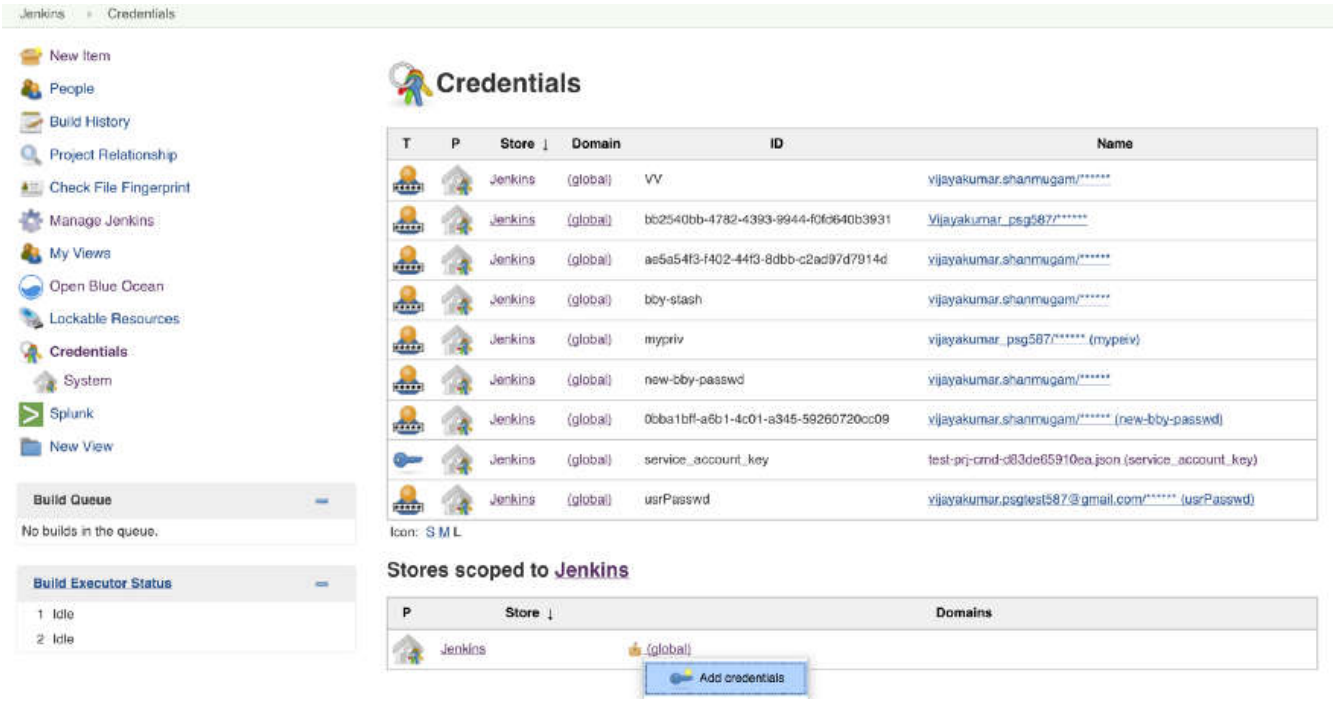
All the three tool configurations are give here

- Configure the executable paths of the tools by locating the bin folder for each

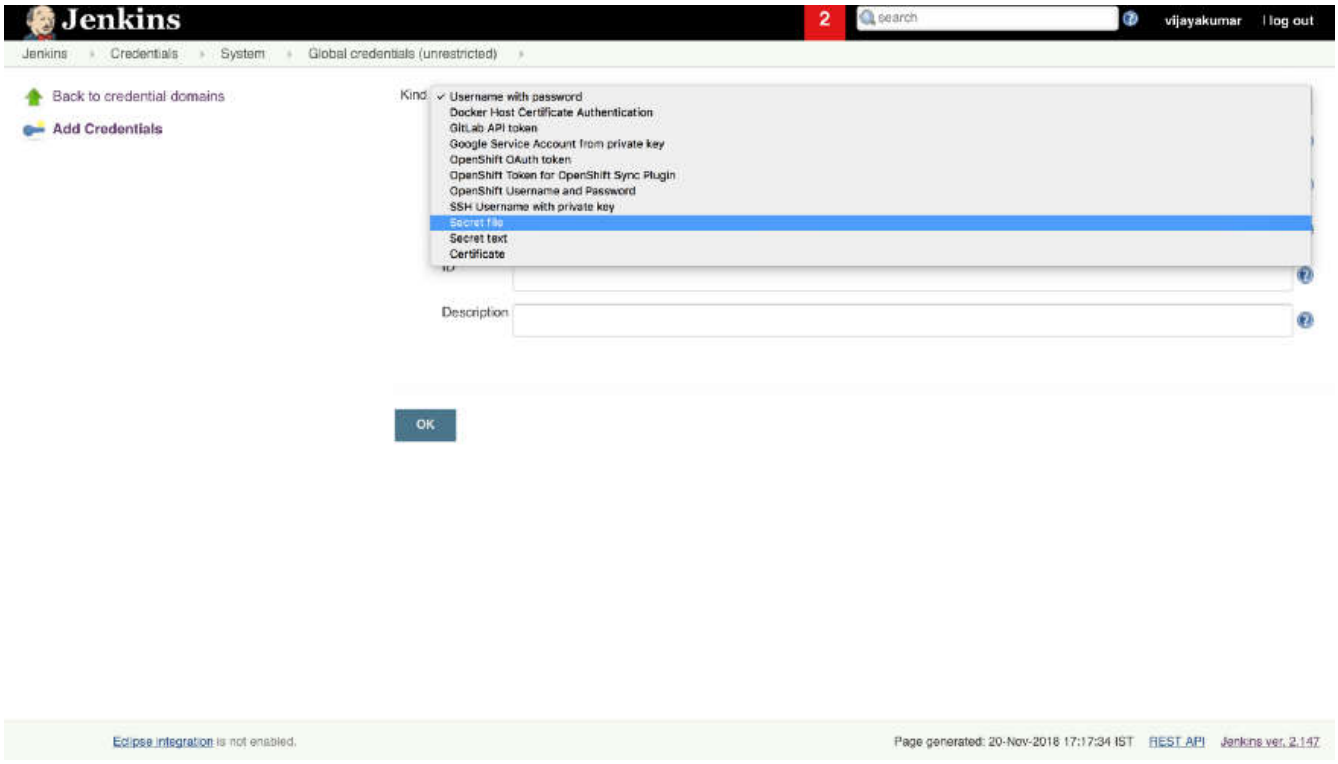
```
which java

which node

which git
```

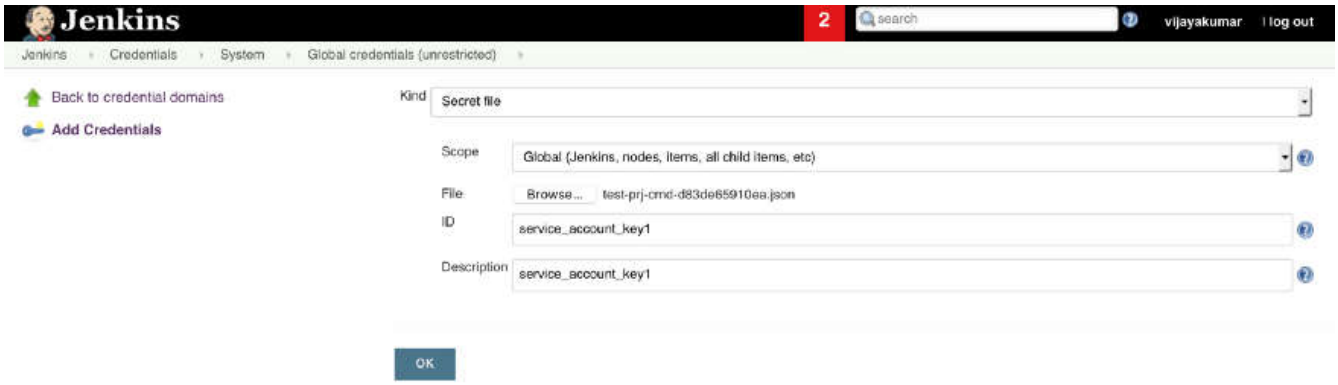
Configure Credentials in Jenkins

- Go to *Jenkins -> Credentials*



- Select domain *global* -> *Add credentials*
- Select *Secret file* option and upload the service account key generated from GCP



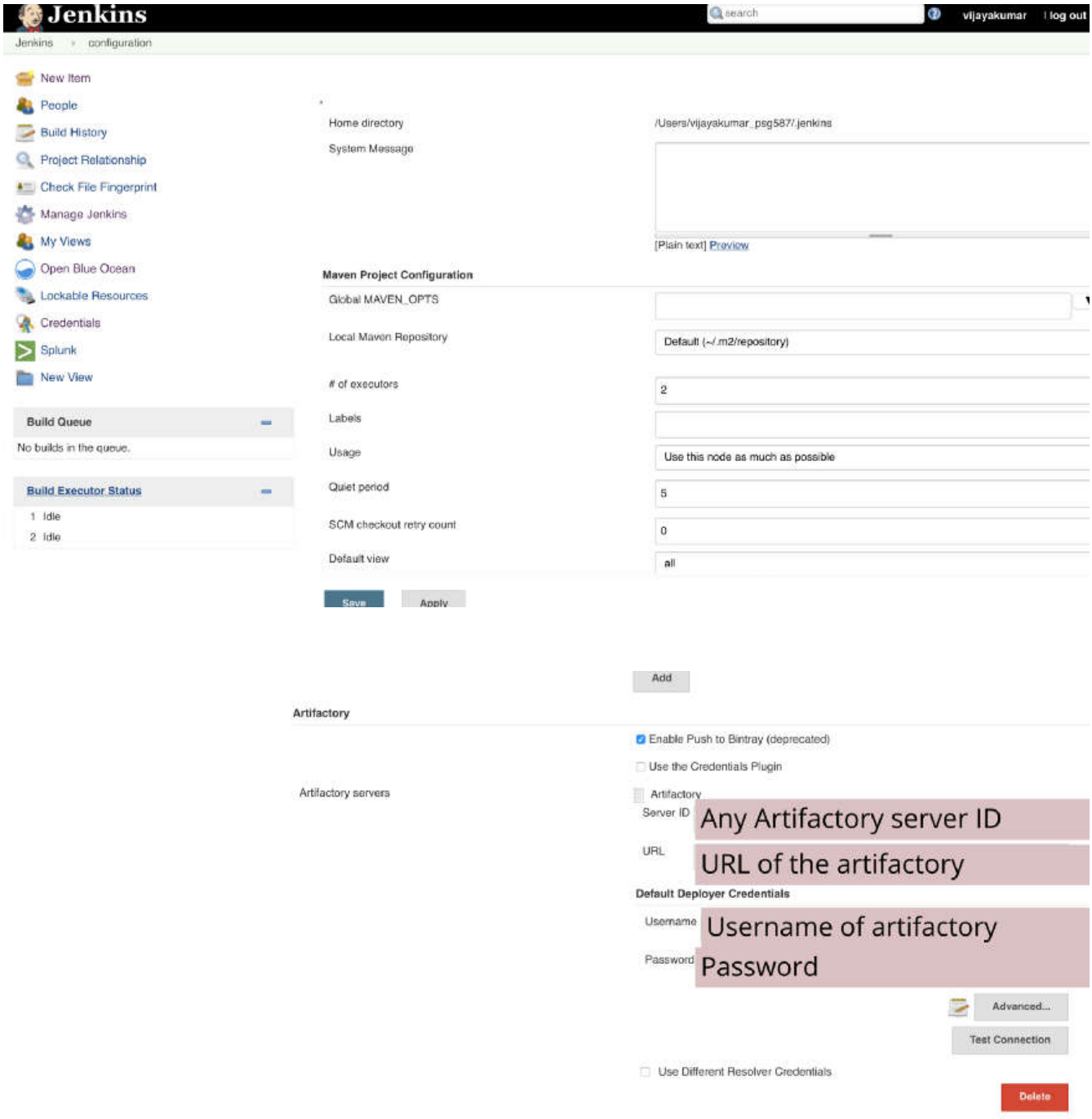


> *Make sure the* Credentials *and the* Plain Credentials *plugin are installed*

Configure Systems in Jenkins

The next step would be to configure the systems such as the slack channels that Jenkins interacts with, Also specify the Artifactory where the bundled apps will be published to, most preferably your Jfrog artifactory

- Go to *Manage Jenkins-> Configure System*



> *Make sure that the* Artifactory *plugin is installed*

Declarative Pipeline

Pipelines are nothing but a stream of different stages that we want the Jenkins to create. These are provided as Jenkins interpretable configurations directives to create the projects . Jenkins needs to know what commands are to be

executed at a certain stage. Since Jenkins is built using java, its compliers are integrated to interpret 'Groovy' — its close ally. Groovy has a lot of embedded features of java along with in-built closures. It is a statically typed, dynamic language which makes coding much more easier. Thus the support for Groovy scripts by default

> *But many found that learning a whole new language for configuration was not worthwhile and Jenkins came up with the solution — Declarative Jenkins Pipeline*
>
> *I personally find declarative pipelines more appealing because of its ability to allow the user to write the configuration as directives*

Understanding Declarative Pipelines

Initial Declaration

Advantage of using declarative pipelines is the ability to store the Jenkins configuration over in SCM

The first statement starts with defining the directive for **pipeline**. All our directives should be inside the **pipeline** block

```
pipeline{

      agent any
}
```

Specifying the **agent** as **any** allows the Jenkins master to select any slave node under its command to execute the *pipeline script*

Now the next step is specifying the **tools** to be used in Jenkins. Tools are nothing but the build tools that are installed in the Jenkins instance. Before using the tools, we should configure them as shown above

```
pipeline {
 agent any

environment {
        GOOGLE_PROJECT_ID = 'test-prj-cmd';

         GOOGLE_SERVICE_ACCOUNT_KEY =
```

```
credentials('service_account_key');
    }

tools {

        git 'localGit'
        jdk 'localJava'
        nodejs 'localNode'

}
```

The environment variables to be used in Jenkins can be declared inside the environment section like above. The usage of **credentials** directive is only possible if the Credentials/Plain Credentials plugin have been installed. The Declaration pipeline requires the instances of that particular class available in the environment. Now to use the service account key that was uploaded in the **Credentials** page, we declare an environment variable like above

The tools to be used can be declared specifying the named aliases of the executable paths (see the Global tool section)

Stages in Pipeline

All pipelines involve multiple stages and declarative pipelines allow us to use stage convention.

```
pipeline {
 agent any

environment {
        GOOGLE_PROJECT_ID = 'test-prj-cmd';
         GOOGLE_SERVICE_ACCOUNT_KEY =
credentials('service_account_key');
    }

tools {

        git 'localGit'
        jdk 'localJava'
        nodejs 'localNode'

}
 stages{
  stage('Init'){
   steps{
     sh '''#!/bin/bash
     echo "JAVA_HOME = ${JAVA_HOME}";
     echo "PATH = ${PATH}";
```

```
        echo "MAVEN_HOME = ${M2_HOME}";

        echo "this is the project id environment"+GOOGLE_PROJECT_ID;
        npm install -g @angular/cli@6.0.8;
        npm install
       '''


       echo "THis is the credentails:${GOOGLE_SERVICE_ACCOUNT_KEY}";
       //readFileStep();
       println "Init success..";

       }

      }

     stage('Build'){
      steps{

       echo "Starting build ...."
       sh '''#!/bin/bash
             ng build --aot --prod
        '''
       println "BUILD NUMBER = $BUILD_NUMBER"
       println "Build Success.."
      }
      post {
             always {

               ----------Post build steps ------

              }

          }

    }
  }
  }
```

There is a lot of info here. I'll try to explain the necessary directives.

Stages are the provisions to provide every phase that Jenkins has to go through. We have the ability to provide multiple stages through ***stage*** declarative.

If a script needs to be executed then those can be provided inside the ***steps*** scope. We are trying to interpret the script as ***bash*** command instead of the normal ***shell*** script (it can also be done in ***bat*** mode for windows) and hence we declare the interpret to be used as

```
#!/bin/bash
```

The ***post*** declarative provides provision for the user to specify any actions to be taken after each successive phase.

println is a groovy command to print statements and ***echo*** is the command to print statements in the console inside a script block

Declaring Deploy stage

The above gist gives a detail description of how to deploy an Angular app to the GCP App Engine. Again the presence of ***app.yaml*** is required and the article assumes it to be present implicitly. We require ***gcloud sdk*** to interact and most of its components to interact with GCP

- Use curl to get the ***gcloud sdk***. Specify the required version to be fetched
- Unzip it using the ***tar*** command to */tmp/*folder
- Now install ***gcloud sdk*** in jenkins by from installation script
- Set the ***Path*** variable for cloud specifying where to find the bin location
- Configure the project to be used using the following command

```
gcloud config set project <project_id>
```

- Authenticate into the ***GCP*** using the following command

```
gcloud auth activate-service-account --key-file <the account json
file, in this case the credentials provided in environment>
```

- Now use the ***gcloud*** command to deploy

```
gcloud app deploy
```

> *Note: App.yaml contains the configuration for deploying the app in GCP*
>
> *There are other parts I have added where in the user can publish the artifactory and also send email notifications of every stage but chose not to explain them since the article's main focus is to deploy to google cloud*

Now come back to Jenkins and either trigger a build manually using — ***Build Now*** or by checking in some changes to SCM. You should see all the stages of the pipeline successfully executed

Each stage with the build result

I hope this gives a good idea to get started with Jenkins. Shoot me up with any questions and I will try to answer them as best as possible. Happy reading!!!