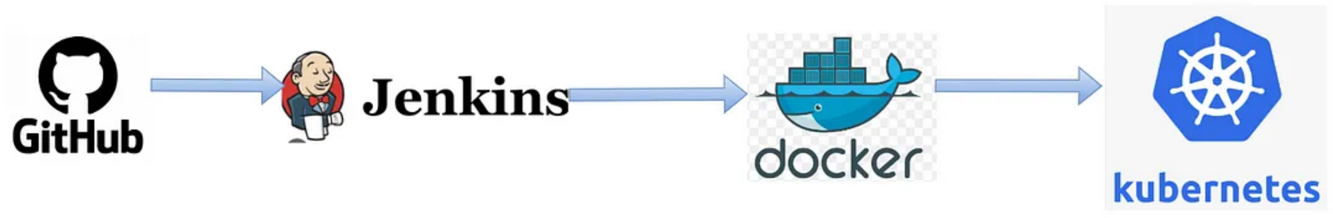
DevOps Project (CI/CD Jenkins pipeline for kubernetes)

Subhasmitadas · Follow
11 min read · Jul 31, 2023

project Url: https://medium.com/p/fa2ee6c4e6c6/edit



In this article we are going to focus on Setting up CI/CD Jenkins pipeline for kubernetes. We will be using GitHub, Docker, DockerHub, Jenkins and Kubernetes Cluster to achieve this.

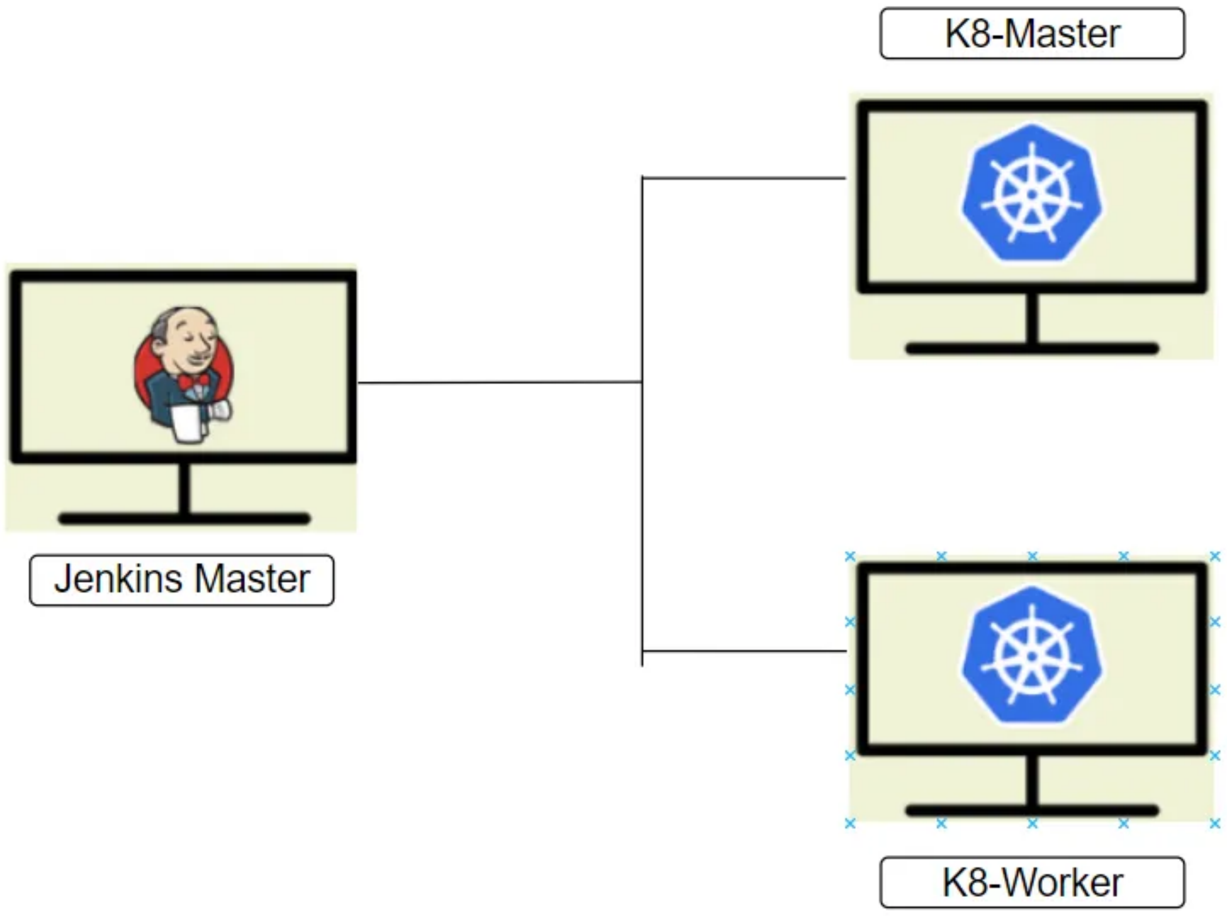
For the complete lab session we are going to setup every component from scratch.

1. Understanding the complete setup

Alright so before we start setting up our CI/CD pipeline I am assuming you have setup your kuernetes cluster already.

As you can see in the sketch we will have three servers(virtual machine) -

1. Jenkins Server
2. Kubernetes Master
3. Kubernetes Worker

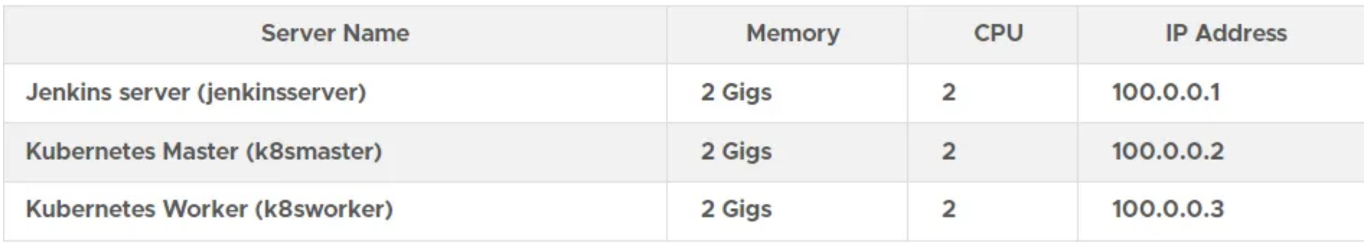


On left you can see **Jenkins server** where we have only installed **Jenkins** and you should keep in mind that we are not running Jenkins inside kubernetes cluster. (*It is not recommended to run Jenkins inside kubernetes cluster.*

On the right hand side you will see two servers one is **k8smaster** for kubernetes master node and **k8sworker** for kubernetes worker.

So in short our **Jenkins server** will connect to **k8smaster** and **k8sworker** for doing the continuous deployment.

Since we have provisioned our Servers using Vagrant so here the configuration -



Server Name	Memory	CPU	IP Address
Jenkins server (jenkinsserver)	2 Gigs	2	100.0.0.1
Kubernetes Master (k8smaster)	2 Gigs	2	100.0.0.2
Kubernetes Worker (k8sworker)	2 Gigs	2	100.0.0.3

For the reference here is the vagrantfile -

```
Vagrant.configure(“2”) do |config|
config.vm.define “jenkinsserver” do |jenkinsserver|
jenkinsserver.vm.box_download_insecure = true
jenkinsserver.vm.box = “hashicorp/bionic64”
jenkinsserver.vm.network “forwarded_port”, guest: 8080, host: 8080
jenkinsserver.vm.network “forwarded_port”, guest: 8081, host: 8081
jenkinsserver.vm.network “forwarded_port”, guest: 9090, host: 9090
jenkinsserver.vm.network “private_network”, ip: “100.0.0.1”
jenkinsserver.vm.hostname = “jenkinsserver”
jenkinsserver.vm.provider “virtualbox” do |v|
v.name = “jenkinsserver”
v.memory = 2048
v.cpus = 2
end
end

config.vm.define “k8smaster” do |k8smaster|
k8smaster.vm.box_download_insecure = true
k8smaster.vm.box = “hashicorp/bionic64”
k8smaster.vm.network “private_network”, ip: “100.0.0.2”
k8smaster.vm.hostname = “k8smaster”
k8smaster.vm.provider “virtualbox” do |v|
v.name = “k8smaster”
v.memory = 2048
v.cpus = 2
```

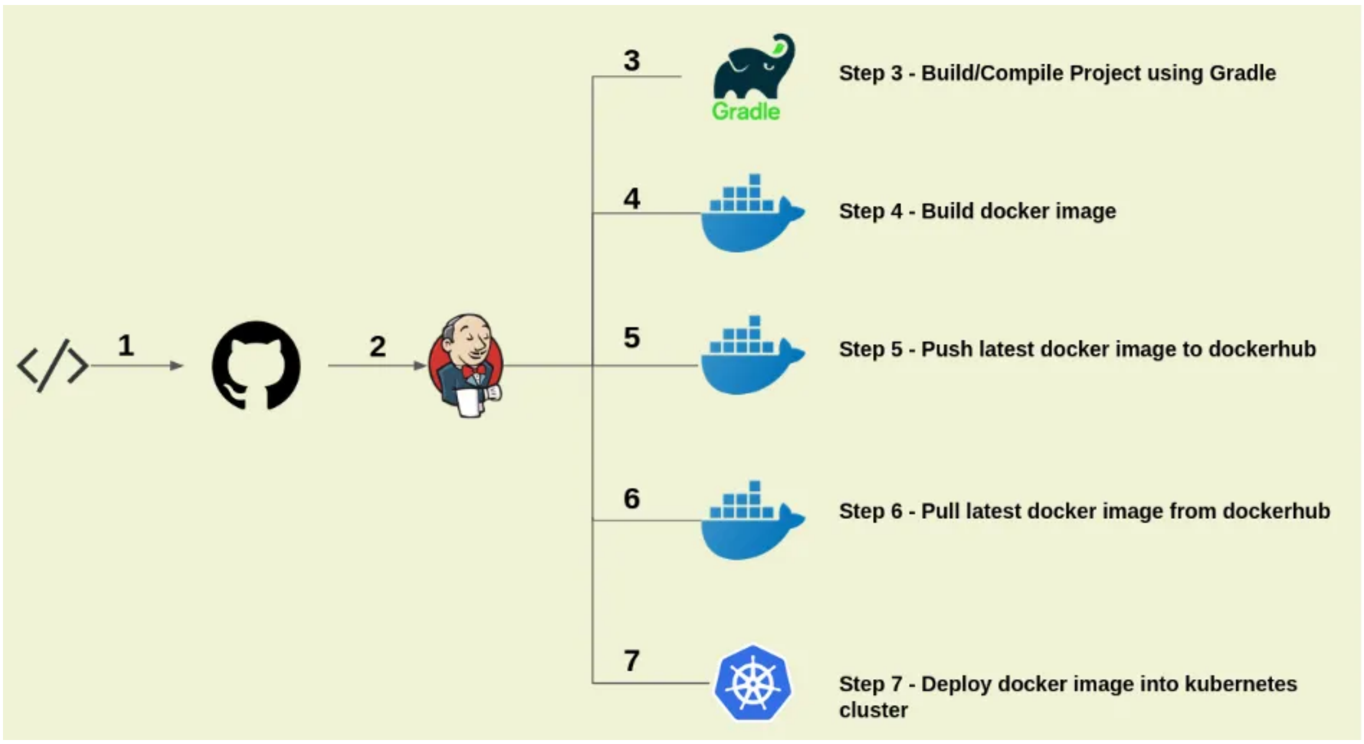
```
end
end

config.vm.define “k8sworker” do |k8sworker|
k8sworker.vm.box_download_insecure = true
k8sworker.vm.box = “hashicorp/bionic64”
k8sworker.vm.network “private_network”, ip: “100.0.0.3”
k8sworker.vm.hostname = “k8sworker”
k8sworker.vm.provider “virtualbox” do |v|
v.name = “k8sworker”
v.memory = 2048
v.cpus = 2
end
end
end
```

2. Where does Github and Docker Hub fits in the CI/CD

We will use Git Hub as version control to push our application code. In this lab session we will be using **Spring Boot Application.**

Secondly we will use DockerHub for uploading/pushing the Docker image. Here is the overview of our GitHub and DockerHub flow -


1
2
3
Gradle
Step 3 - Build/Compile Project using Gradle
4
Step 4 - Build docker image
5
Step 5 - Push latest docker image to dockerhub
6
Step 6 - Pull latest docker image from dockerhub
7
Step 7 - Deploy docker image into kubernetes cluster

Step 1 — Checkin/Push your code to GitHub

Step 2 — Pull your code from GitHub into your Jenkins server

Step 3 — Use Gradle/Maven build tool for building the artifacts

Step 4 — Create Docker image

Step 5 — Push your latest Docker image to DockerHub

Step 6 — Pull the latest image from DockerHub into jenkins.

Step 7 — Then use **k8s-spring-boot-deployment.yml** to deploy your application inside your kubernetes cluster.

3. Install Jenkins on your jenkinsserver

Alright lets start with Jenkins installation on our **jenkinsserver**

3.1 Start vagrant box

If your vagrant box is not running then start up your vagrant box -

```bash
vagrant up
```

BASH

3.2 Log into vagrant

After starting your vagrant box you can login into it -

```bash
vagrant ssh jenkinsserver
```

BASH

3.3 Update ubuntu repositories

Update the repositories of your ubuntu -

```bash
sudo apt update
```

BASH

3.4 Install Java

You also need Java as pre-requisite for installing jenkins, so lets first install java -

```
sudo apt install openjdk-8-jdk
```

BASH

Verify your java installation by running the following command

```
java -version
```

BASH

It should return with java version

```
openjdk version "1.8.0_265"
OpenJDK Runtime Environment (build 1.8.0_265-8u265-b01-0ubuntu2~18.04-b01)
OpenJDK 64-Bit Server VM (build 25.265-b01, mixed mode)
```

BASH

3.5 Install Jenkins

Now we can install the jenkins on our Ubuntu server, use the following command to install it -

```
wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo apt-key
```

BASH

```bash
sudo sh -c 'echo deb https://pkg.jenkins.io/debian-stable binary/ > \
    /etc/apt/sources.list.d/jenkins.list'
```

BASH

```bash
sudo apt-get update
```

BASH

```bash
sudo apt-get install jenkins
```

BASH

(*Note : For more installation options for MacOS, Windows, CentOS or RedHat please refer to official documentation for jenkins installation*)

3.6 Verify Jenkins installation

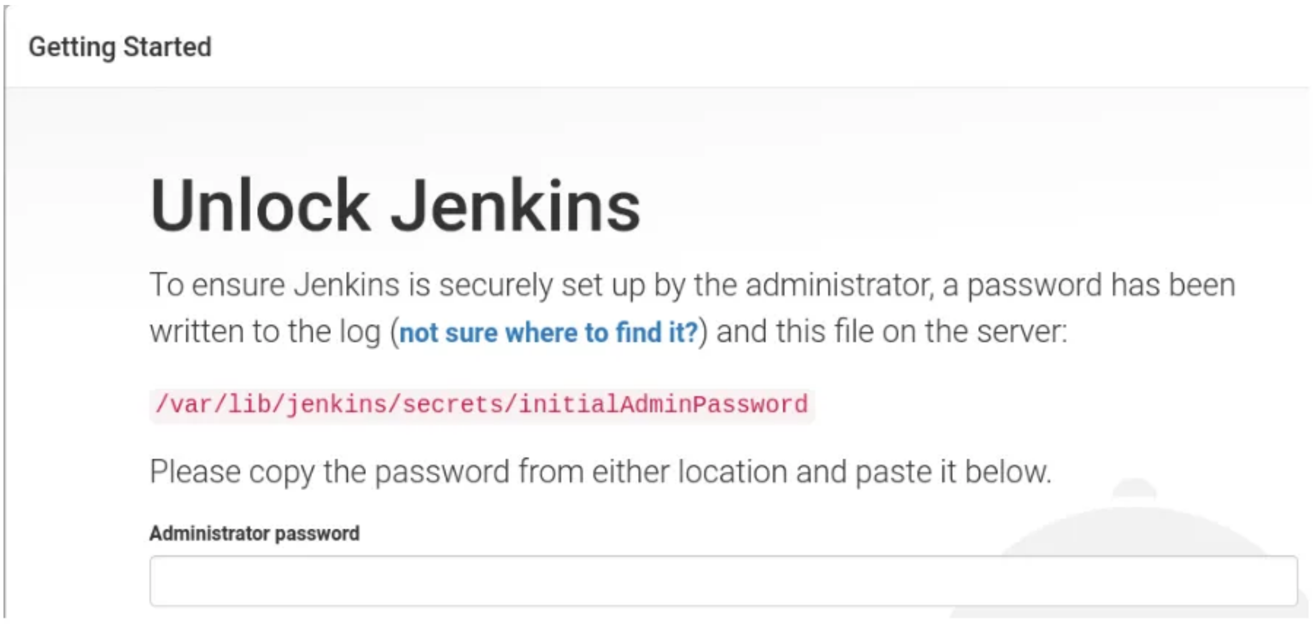
After installing Jenkins you can verify the jenkins installation by accessing the initial login page of the jenkins.

Since we have installed Jenkins on virtual machine with IP 100.0.0.1, so you can access using following URL

```
http://100.0.0.1:8080/
```

BASH

And if you have installed the jenkins correctly then you should be able to see the following initial login page of Jenkins


Getting Started

Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log (not sure where to find it?) and this file on the server:

`/var/lib/jenkins/secrets/initialAdminPassword`

Please copy the password from either location and paste it below.

Administrator password

4. Fetch Default jenkins password

As you can see you need to provide Default jenkins(initialAdminPassword) administrator password.

So the question is — How to find Default Jenkins password(initialAdminPassword)?

There are two ways find Default Jenkins password-

1. Using the Jenkins Logs file located at — `/var/log/jenkins/jenkins.log`
2. Using the Jenkins secret — `/var/lib/jenkins/secrets/initialAdminPassword`

For Option 1 — Run the following command -

```
cat /var/log/jenkins/jenkins.log
```

BASH

And then look for following -

```
*************************************************************
*************************************************************
*************************************************************

Jenkins initial setup is required. An admin user has been created and a passwo
Please use the following password to proceed to installation:

e0d325fb18a64797bd81dcb77e865237

This may also be found at: /var/lib/jenkins/secrets/initialAdminPassword
```

For Option 2 — Run the following command -

```
cat /var/lib/jenkins/secrets/initialAdminPassword
```

BASH

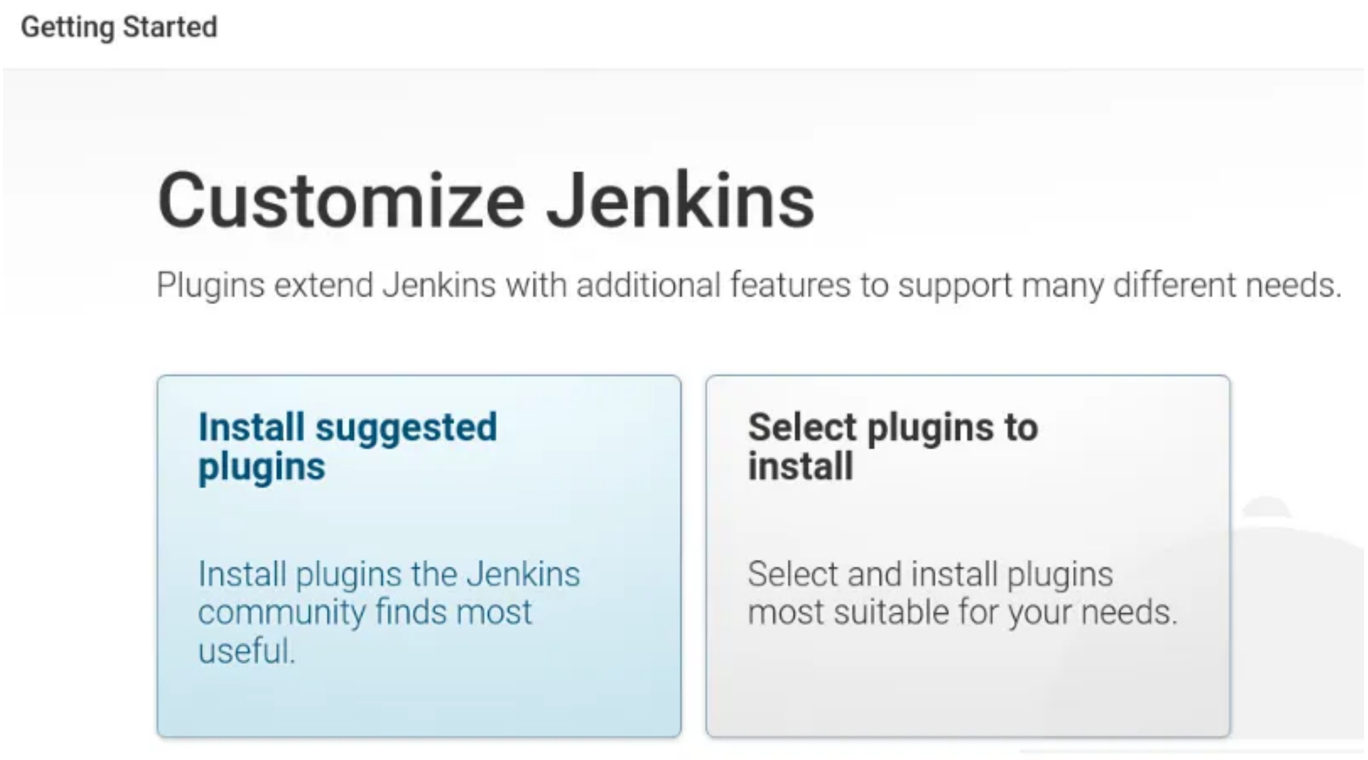
It should return you back the password

```
e0d325fb18a64797bd81dcb77e865237
```

BASH

Customize Jenkins and install suggested plugin

After you have entered your **initialAdminPassword** you should be able to see the following Customize jenkins page -



Getting Started

Customize Jenkins

Plugins extend Jenkins with additional features to support many different needs.

Install suggested plugins

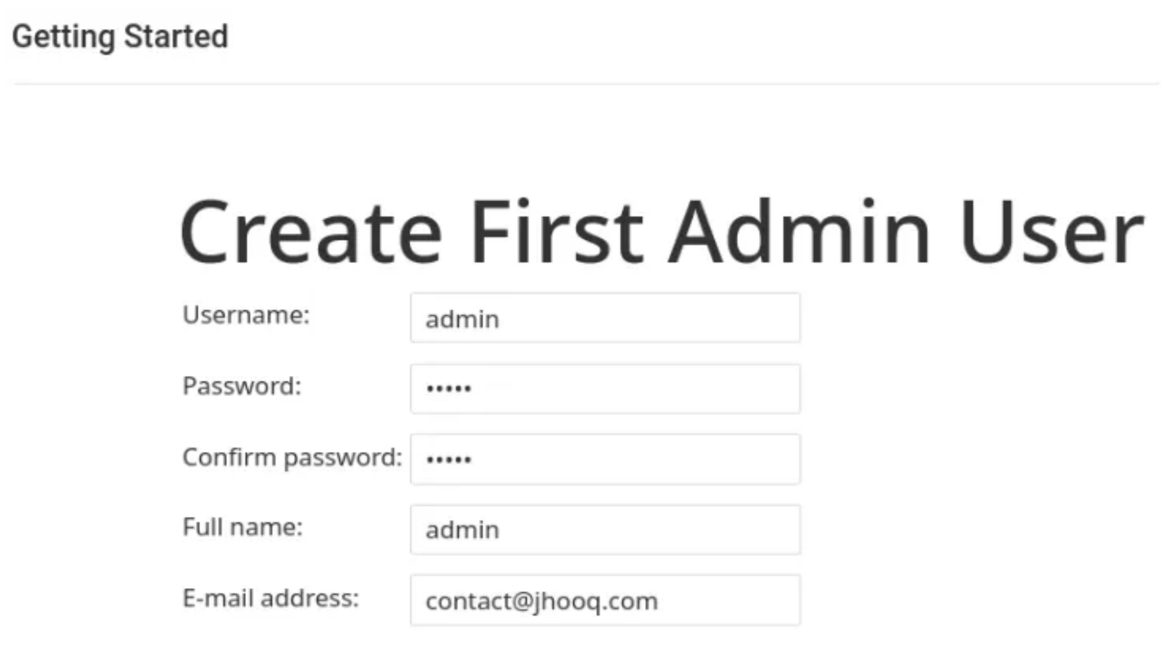
Install plugins the Jenkins community finds most useful.

Select plugins to install

Select and install plugins most suitable for your needs.

After successful login setup username and password

After you have installed all the suggested default plugins, it will prompt you for setting up username and password -

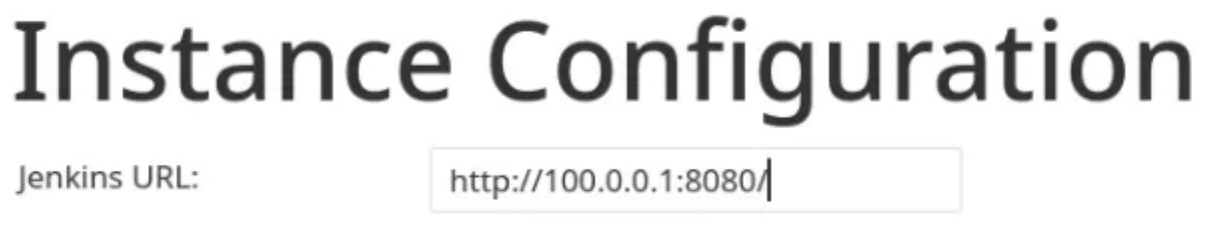


Create First Admin User

Username:	admin
Password:	•••••
Confirm password:	•••••
Full name:	admin
E-mail address:	contact@jhooq.com

Set desired username and password (*In this lab session I am going to keep Username — admin and password — admin*)

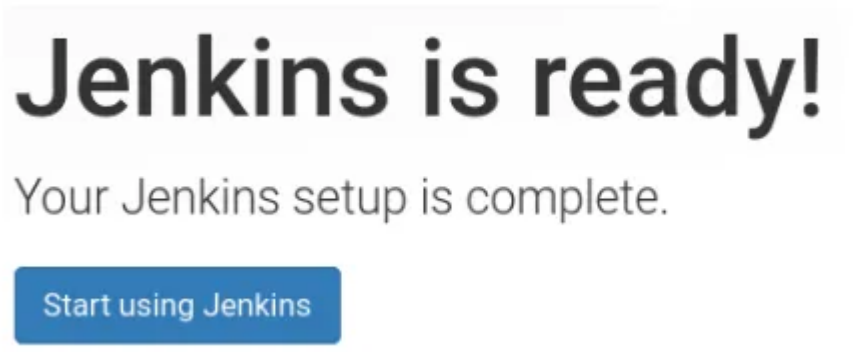
After that it will ask for Jenkins URL and I would recommend you to keep the default one -


Instance Configuration

Jenkins URL: http://100.0.0.1:8080/

Jenkins Create First Admin User

Now you jenkins is ready to use -


Jenkins is ready!

Your Jenkins setup is complete.

Jenkins is Ready

5. Jenkins — Install "SSH Pipeline Steps" plugin and "Gradle"

5.1 SSH Pipeline Steps

Moving ahead we need to install one more plugin **SSH Pipeline Steps** which we are going to use for SSH into **k8smaster** and **k8sworker** server.

For installing plugin please goto — `Manage Jenkins -> Manage Plugin -> Available` then in the search box type **SSH Pipeline Steps.**

Select the plugin and install it without restart.

5.2 Setup Gradle

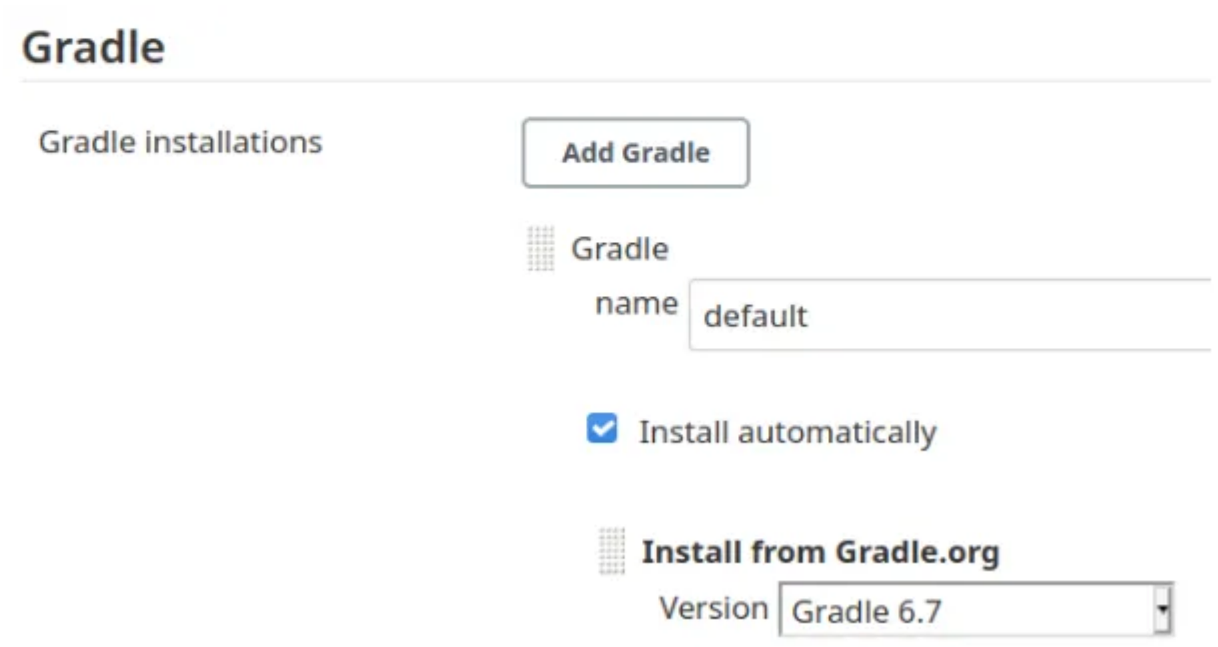
For this lab session we are going to use Spring Boot Application, so for that we need Gradle as our build tool.

To setup Gradle Goto — `Manage Jenkins -> Global Tool Configuration -> Gradle`

Click on `Add Grdle` and then enter name `default`.

After that click on the checkbox — `Install Automatically` and from the drop down `Install from Gradle.org` select latest version.

Refer to the following screenshot


Gradle

Gradle installations

Gradle

name: default

☑ Install automatically

Install from Gradle.org

Version: Gradle 6.7

6. Install Docker on jenkinsserver

Alright now we need to install Docker on **jenkinsserver** since we need to push our docker image to DockerHub.

(Note — We are not installing any plugins for jenkins, till the step no 5 we have completed the jenkins and jenkins plugin setup)

6.1 Logback into jenkinsserver

```
vagrant ssh jenkinsserver
```

BASH

6.2 Install Docker

Use the following command to install the Docker -

```
sudo apt install docker.io
```

BASH

After installing docker verify the installation with following docker command -

```
Client:
 Version:           19.03.6
 API version:       1.40
 Go version:        go1.12.17
 Git commit:        369ce74a3c
 Built:             Wed Oct 14 19:00:27 2020
 OS/Arch:           linux/amd64
 Experimental:      false
```

BASH

6.3 Add Current User to Docker Group

The next step you need to do is to add the current user to Docker Group to avoid the error — *Got permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock* (*For more detail refer to this post*)

Use the following command for adding current user to Docker Group

```bash
sudo usermod -aG docker $USER
```

BASH

6.4 Add Jenkins User to Docker Group

Similar to previous step we also need to add Jenkins User to Docker Group, so that Jenkins can use Docker for building and pusing the docker images.

Run the following command to add Jenkins user to Docker Group

```bash
sudo usermod -aG docker jenkins
```

BASH

7. Take a look at Spring Boot Application

`Note - In this step we are using Spring Boot Application but if you have different application like NodeJs, Angular etc than you can use that application. Rest of the steps are going to be the same.`

Now its time for you to look at our application which we are going to deploy inside kubernetes cluster using Jenkins Pipeline.

To keep the lab session simple we are going to write **Hello World** rest webservice using Spring Boot.

Here is Java class of rest end point -

```
@RestController
public class JhooqDockerDemoController {

    @GetMapping("/hello")
    public String hello() {
        return "Docker Demo - Hello Jhooq";
    }
}
```

JAVA

You can clone the code repo from — Source Code

The source code also include the Dockerfile for building the dockerimage -

```
FROM openjdk:11
ARG JAR_FILE=build/libs/*.jar
COPY ${JAR_FILE} app.jar
ENTRYPOINT ["java","-jar","/app.jar"]
```

DOCKERFILE

8. Write the Pipeline script

To make this lab session more simple I have divided CI/CD Jenkins Pipeline script into 11 steps

8.1 Create Jenkins Pipeline

The first step for you would be to create a pipeline.

1. **Goto :** `Jenkins -> New Items`
2. **Enter an item name :** `Jhooq-SpringBoot`
3. Select Pipeline
4. Click Ok

8.2 Clone the Git Repo

The first principle of the CI/CD pipeline is to clone/checkout the source code, using the same principle we are going to clone the GIT Repo inside Jenkins

```
stage("Git Clone"){

        git credentialsId: 'GIT_HUB_CREDENTIALS', url: 'https://github.com/subh
    }
```

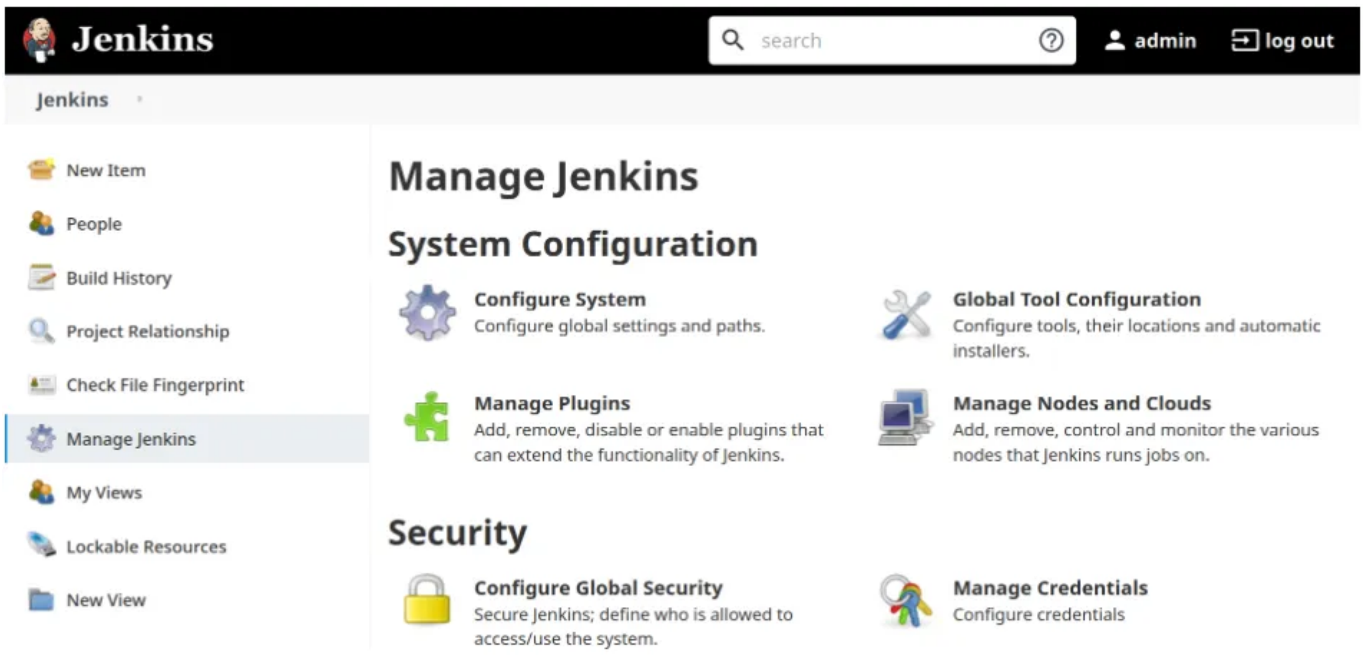
BASH

8.3 Jenkins store git credentials

As you know we cannot store plain text password inside jenkins scripts, so we need to store it somewhere securely.

Jenkins **Manage Credential** provides very elegant way to store GitHub Username and Password.

Goto : `Jenkins -> Manage Jenkins -> Manage Credentials`



Manage Jenkins

System Configuration

Configure System
Configure global settings and paths.

Global Tool Configuration
Configure tools, their locations and automatic installers.

Manage Plugins
Add, remove, disable or enable plugins that can extend the functionality of Jenkins.

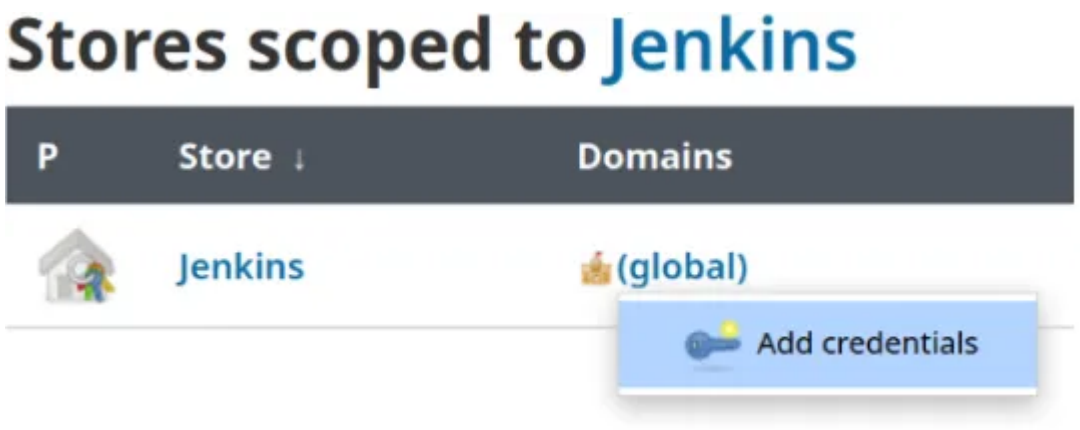
Manage Nodes and Clouds
Add, remove, control and monitor the various nodes that Jenkins runs jobs on.

Security

Configure Global Security
Secure Jenkins; define who is allowed to access/use the system.

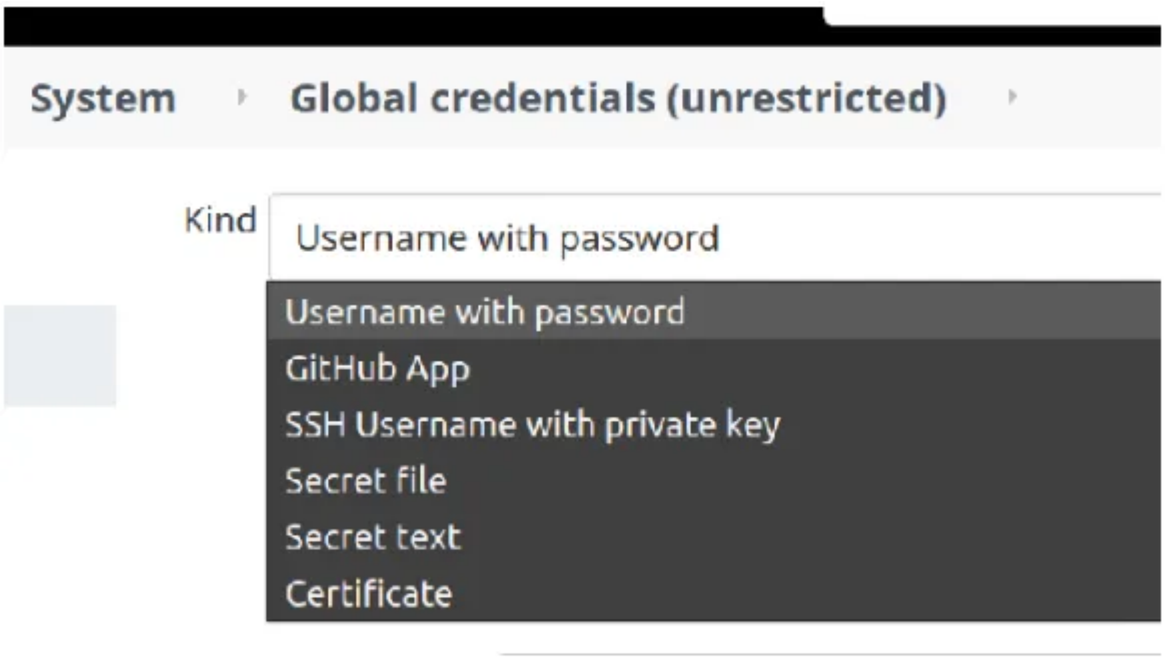
Manage Credentials
Configure credentials

After that Goto : `Stores scoped to Jenkins -> global`



enkins store git credentials

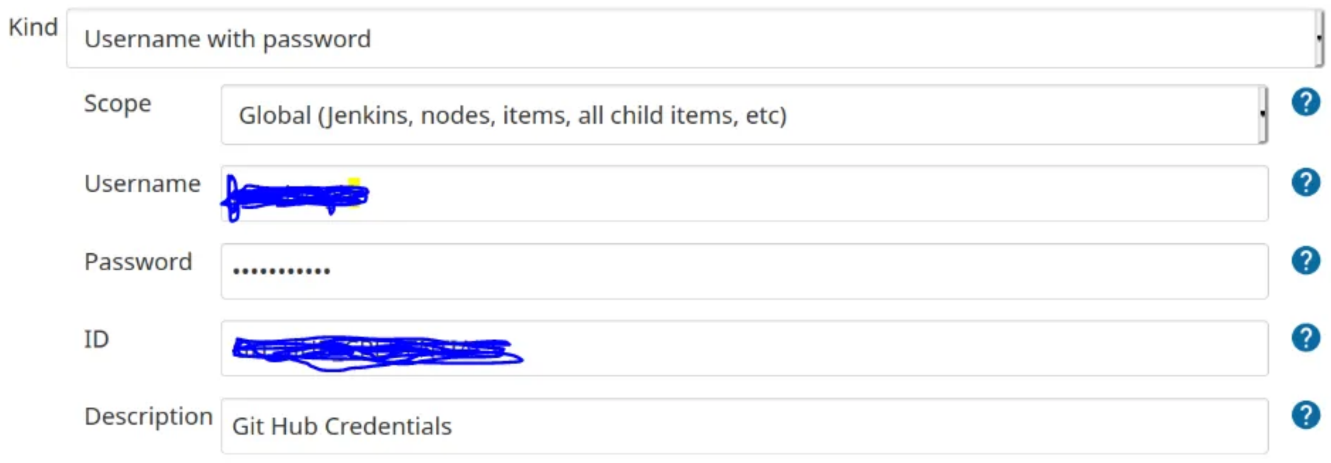
Then select Username with Password


System › Global credentials (unrestricted) ›

Kind: Username with password

- Username with password
- GitHub App
- SSH Username with private key
- Secret file
- Secret text
- Certificate

And then input you GitHub Username and Password. But always remember the ID



Kind	Username with password
Scope	Global (Jenkins, nodes, items, all child items, etc)
Username	[redacted]
Password	•••••••••••
ID	[redacted]
Description	Git Hub Credentials

Note — Keep the ID somewhere store so that you remember — `GIT_HUB_CREDENTIALS`

8.4 Build the Spring Boot Application

Next step would be to build the Spring Boot Application Using Gradle

```
stage('Gradle Build') {

        sh './gradlew build'

    }
```

BASH

8.5 Build Docker image and tag it

After successful Gradle Build we are going to build Docker Image and after that I am going to tag it with the name `jhooq-docker-demo`

```
stage("Docker build"){
         sh 'docker version'
         sh 'docker build -t jhooq-docker-demo .'
         sh 'docker image list'
         sh 'docker tag jhooq-docker-demo rahulwagh17/jhooq-docker-demo:jhooq-do
    }
```

BASH

8.6 Jenkins store DockerHub credentials

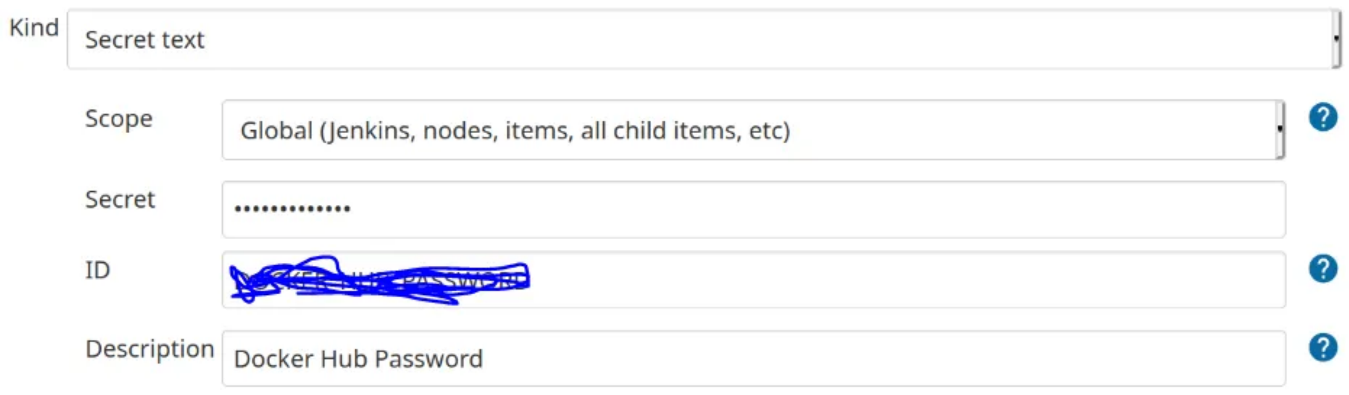
For storing DockerHub Credentials you need to GOTO: `Jenkins -> Manage Jenkins -> Manage Credentials -> Stored scoped to jenkins -> global -> Add Credentials`

From the `Kind` dropdown please select `Secret text`

1. Secret — Type in the DockerHub Password

2. ID — DOCKER_HUB_PASSWORD

3. Description — Docker Hub password



Kind: Secret text

Scope: Global (Jenkins, nodes, items, all child items, etc)

Secret: ••••••••••••••

ID: [redacted]

Description: Docker Hub Password

8.7 Docker Login via CLI

Since I am working inside Jenkins so every step I perform I need to write pipeline script. Now after building and tagging the Docker Image we need to push it to the DockerHub. But before you push to DockerHub you need to authenticate yourself via CLI(command line interface) using `docker login`

So here is the pipeline step for **Docker Login**

```
stage("Docker Login"){
        withCredentials([string(credentialsId: 'DOCKER_HUB_PASSWORD', variable:
            sh 'docker login -u subhasmita17 -p $PASSWORD'
        }
    }
```

BASH

`$DOCKER_HUB_PASSWORD` - Since I cann't disclose my DockerHub password, so I stored my DockerHub Password into Jenkins `Manage Jenkins` and assigned the ID `$DOCKER_HUB_PASSWORD`

8.8 Push Docker Image into DockerHub

After successful Docker login now we need to push the image to DockerHub

```
stage("Push Image to Docker Hub"){
        sh 'docker push  subhasmita17/jhooq-docker-demo:jhooq-docker-demo'
    }
```

BASH

8.9 SSH Into k8smaster server

If you remember we have installed **SSH Pipeline Steps** in step no — 5, now we are going to use that plugin to SSH into k8smaster server

```
stage("SSH Into k8s Server") {
        def remote = [:]
        remote.name = 'K8S master'
        remote.host = '100.0.0.2'
        remote.user = 'vagrant'
        remote.password = 'vagrant'
        remote.allowAnyHosts = true
}
```

BASH

8.10 Copy k8s-spring-boot-deployment.yml to k8smaster server

After successful login copy **k8s-spring-boot-deployment.yml** into k8smaster server

```
stage('Put k8s-spring-boot-deployment.yml onto k8smaster') {
            sshPut remote: remote, from: 'k8s-spring-boot-deployment.yml', into
        }
```

BASH

8.11 Create kubernetes deployment and service

Apply the k8s-spring-boot-deployment.yml which will eventually -

1. Create deployment with name — **jhooq-springboot**
2. Expose service on NodePort

```
stage('Deploy spring boot') {
          sshCommand remote: remote, command: "kubectl apply -f k8s-spring-boot
        }
```

BASH

So here is the final complete pipeline script for my CI/CD Jenkins kubernetes pipeline

```
node {

   stage("Git Clone"){

       git credentialsId: 'GIT_CREDENTIALS', url: 'https://github.com/subhasm
   }

    stage('Gradle Build') {

      sh './gradlew build'

    }

    stage("Docker build"){
        sh 'docker version'
        sh 'docker build -t jhooq-docker-demo .'
        sh 'docker image list'
        sh 'docker tag jhooq-docker-demo subhasmita17/jhooq-docker-demo:jhooq-
    }

    withCredentials([string(credentialsId: 'DOCKER_HUB_PASSWORD', variable: 'P
        sh 'docker login -u subhasmita17 -p $PASSWORD'
    }

    stage("Push Image to Docker Hub"){
        sh 'docker push  subhasmita17/jhooq-docker-demo:jhooq-docker-demo'
    }

    stage("SSH Into k8s Server") {
        def remote = [:]
        remote.name = 'K8S master'
        remote.host = '100.0.0.2'
        remote.user = 'vagrant'
        remote.password = 'vagrant'
        remote.allowAnyHosts = true

        stage('Put k8s-spring-boot-deployment.yml onto k8smaster') {
            sshPut remote: remote, from: 'k8s-spring-boot-deployment.yml', int
        }

        stage('Deploy spring boot') {
          sshCommand remote: remote, command: "kubectl apply -f k8s-spring-boo
        }
    }

}

```

BASH

Conclusion

1. First thing which I did is — setup the kubernetes cluster
2. Install Jenkins on another server
3. Install plugin 'SSH Pipeline Step' for jenkins
4. Install Docker along with Jenkins
5. Setup user group for CurrentUser and Jenkins

6. created Jenkins Pipeline script for continuous Deployment

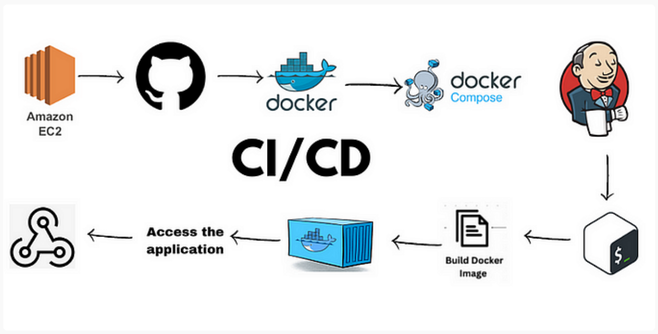
Written by Subhasmitadas

Follow

353 Followers

Never Give-up be strong always

More from Subhasmitadas



Subhasmitadas

Devops Project — 3

Complete Jenkins CI/CD Project

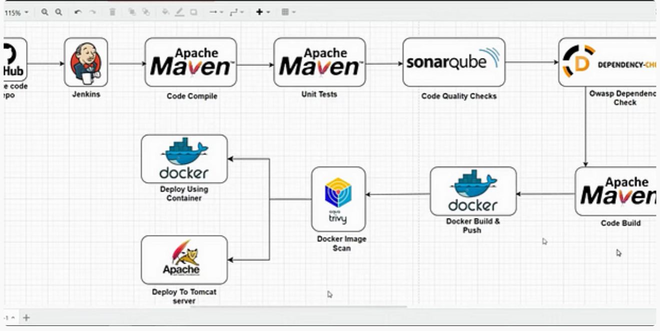
Jul 11, 2023 · 83

Subhasmitadas

Create free tier AWS account, create IAM user and set Cloud...

project url: https://medium.com/p/66e0c1946a73/edit

Jul 13, 2023 · 5 · 1

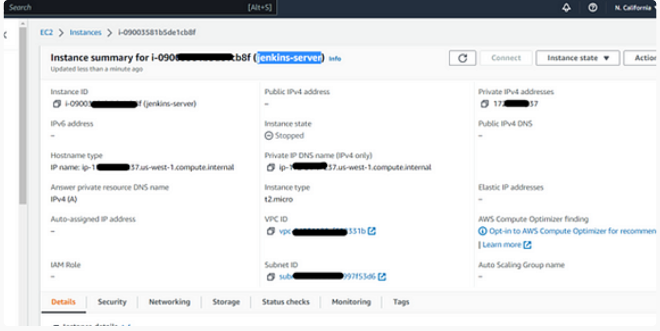


Subhasmitadas

3RD PROJECT

Spring Boot application in Complete CI/CD Pipeline

Jul 7, 2023 5



EC2 > Instances > i-09003581b5de1cb8f

Instance summary for i-090[redacted]b8f (jenkins-server) Info
Updated less than a minute ago

Connect | Instance state | Action

Instance ID
i-09003[redacted]f (jenkins-server)

Public IPv4 address
–

Private IPv4 addresses
17[redacted]37

IPv6 address
–

Instance state
Stopped

Public IPv4 DNS
–

Hostname type
IP name: ip-1[redacted]37.us-west-1.compute.internal

Private IP DNS name (IPv4 only)
ip-1[redacted]37.us-west-1.compute.internal

Answer private resource DNS name
IPv4 (A)

Instance type
t2.micro

Elastic IP addresses
–

Auto-assigned IP address
–

VPC ID
vpc[redacted]331b

AWS Compute Optimizer finding
Opt-in to AWS Compute Optimizer for recommen
| Learn more

IAM Role
–

Subnet ID
sub[redacted]97f53d6

Auto Scaling Group name
–

Details | Security | Networking | Storage | Status checks | Monitoring | Tags

Subhasmitadas

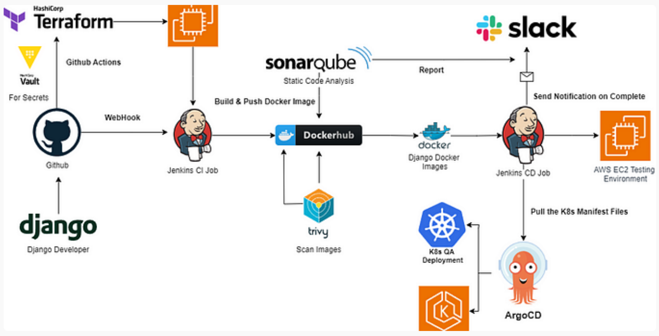
DevOps (Implementation)Project-6

Deploying web app using Jenkins CI/CD declarative pipeline.

Jul 21, 2023 2

See all from Subhasmitadas

Recommended from Medium



Joel Wembo in Django Unleashed

Technical Guide: End-to-End CI/CD DevOps with Jenkins, Docker,...

Building an end-to-end CI/CD pipeline for Django applications using Jenkins, Docker,...

Apr 12 918 21

Minimal Devops

App of Apps Pattern in ArgoCD

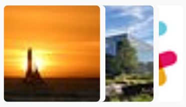
Simplifying Application Hierarchies

Nov 7 11

Lists

Staff picks

766 stories · 1444 saves



Stories to Help You Level-Up at Work

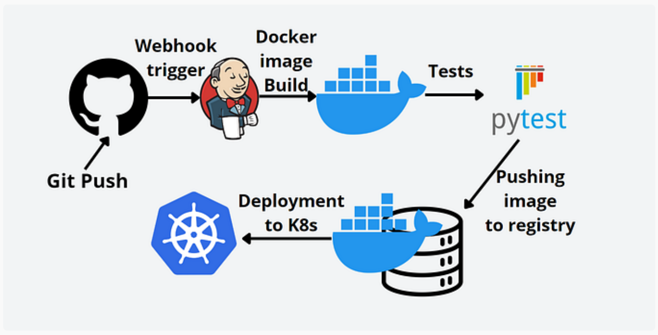
19 stories · 865 saves

Self-Improvement 101

20 stories · 3025 saves

Productivity 101

20 stories · 2565 saves

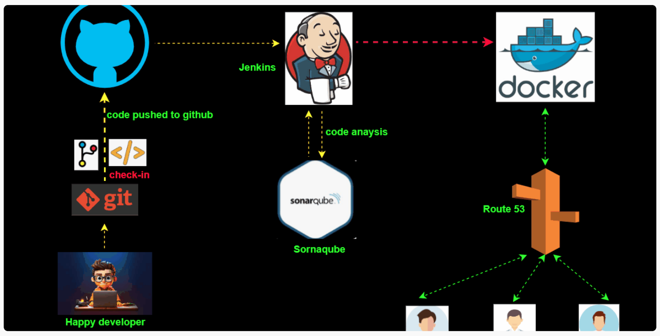


Nikita Volzhin

CI/CD: Deployment to Kubernetes Cluster

Building a Kubernetes cluster is like assembling IKEA furniture, but with more...

Jul 21 · 81

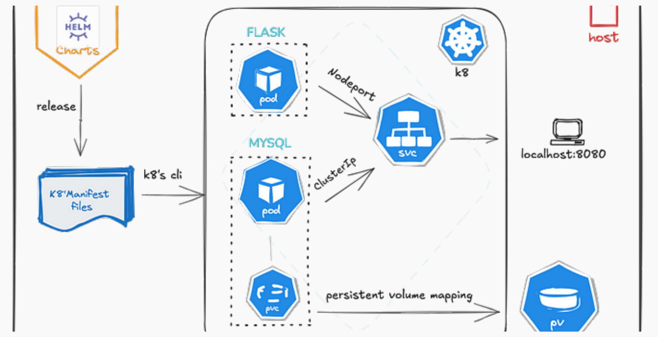


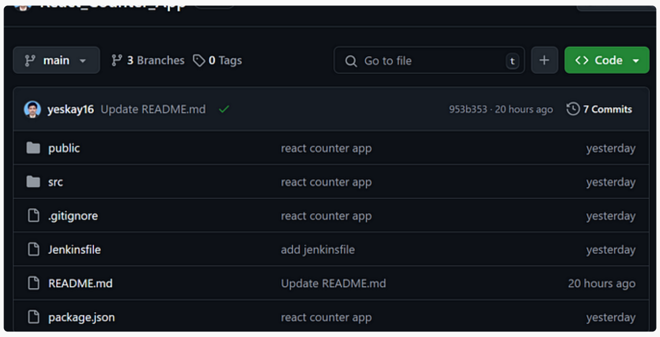
Victor wasonga onyango

Building a Robust CI/CD Pipeline with Jenkins, SonarQube, Docker,...

Introduction

Jul 27 · 74





main · 3 Branches · 0 Tags · Go to file · t · + · Code

yeskay16 Update README.md ✓ 953b353 · 20 hours ago · 7 Commits

public	react counter app	yesterday
src	react counter app	yesterday
.gitignore	react counter app	yesterday
Jenkinsfile	add jenkinsfile	yesterday
README.md	Update README.md	20 hours ago
package.json	react counter app	yesterday

th@n@n in DevOps.dev

Automating Kubernetes Deployments with Helm Charts

Helm chart is the package manager for Kubernetes, which involve directly in...

Nov 6 · 45

Karthik Seenuvasan

Trigger Jenkins — Multibranch Pipleline using Jenkinsfile.

https://youtu.be/vQmS0cEjTyw

Sep 25 · 106

See more recommendations