

Hello friends, we will be deploying a Petshop Java Based Application. This is an everyday use case scenario used by several organizations. We will be using Jenkins as a CICD tool and deploying our application on a Docker container and Kubernetes cluster. Hope this detailed blog is useful.

We will be deploying our application in two ways, one using Docker Container and the other using K8S cluster.

Project Repo: [github.com/Aj7Ay/jpetstore-6.git](https://github.com/Aj7Ay/jpetstore-6.git)

**STEPS:**

Step 1 -- Create an Ubuntu(22.04) T2 Large Instance

Step 2 -- Install Jenkins, Docker and Trivy

Step 3 -- Install Plugins like JDK, Sonarqube Scanner, Maven, OWASP Dependency Check

Step 4 -- Configure Sonar Server in Manage Jenkins

Step 5 -- Install OWASP Dependency Check Plugins

Step 6 -- Docker plugin and credential Setup

Step 7 -- Adding Ansible Repository in Ubuntu and install Ansible

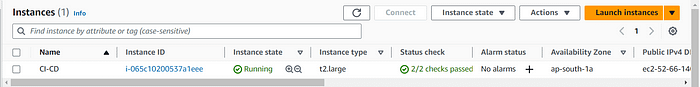
Step 8 -- Kuberenetes Setup

Step 9 -- Master-slave Setup for Ansible and Kubernetes

Now, let’s get started and dig deeper into each of these steps:-

**STEP1:Create an Ubuntu(22.04) T2 Large Instance**

Launch an AWS T2 Large Instance. Use the image as Ubuntu. You can create a new key pair or use an existing one. Enable HTTP and HTTPS settings in the Security Group and open all ports (not best case to open all ports but just for learning purposes it’s okay).



**Step 2 — Install Jenkins, Docker and Trivy**

**2A — To Install Jenkins**

Connect to your console, and enter these commands to Install Jenkins

COPY

vi jenkins.sh

#!/bin/bash  
sudo apt update -y  
#sudo apt upgrade -y  
wget -O - https://packages.adoptium.net/artifactory/api/gpg/key/public | tee /etc/apt/keyrings/adoptium.asc  
echo "deb [signed-by=/etc/apt/keyrings/adoptium.asc] https://packages.adoptium.net/artifactory/deb $(awk -F= '/^VERSION\_CODENAME/{print$2}' /etc/os-release) main" | tee /etc/apt/sources.list.d/adoptium.list  
sudo apt update -y  
sudo apt install temurin-17-jdk -y  
/usr/bin/java --version  
curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudo tee \  
 /usr/share/keyrings/jenkins-keyring.asc > /dev/null  
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \  
 https://pkg.jenkins.io/debian-stable binary/ | sudo tee \  
 /etc/apt/sources.list.d/jenkins.list > /dev/null  
sudo apt-get update -y  
sudo apt-get install jenkins -y  
sudo systemctl start jenkins  
sudo systemctl status jenkins

sudo chmod 777 jenkins.sh  
./jenkins.sh # this will installl jenkins

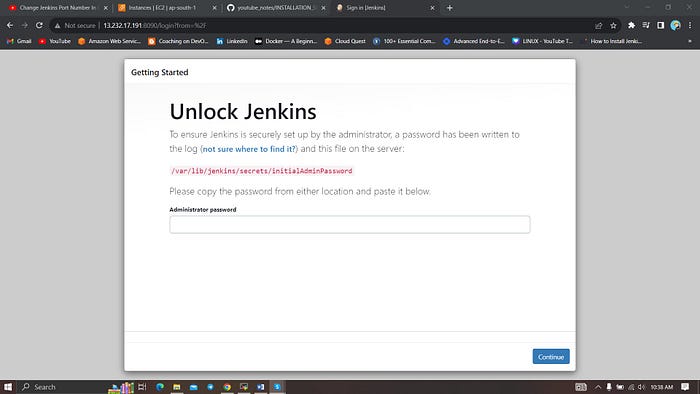
Once Jenkins is installed, you will need to go to your AWS EC2 Security Group and open Inbound Port 8080 and 8090, 9000 for sonar, since Jenkins works on Port 8080.

But for this Application case, we are running Jenkins on another port. so change the port to 8090 using the below commands.

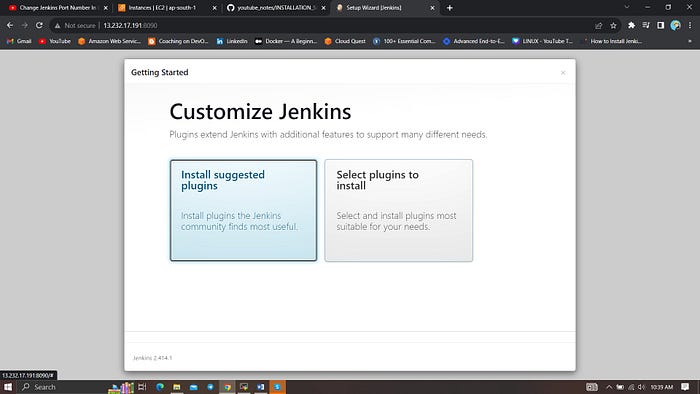
sudo systemctl stop jenkins  
sudo systemctl status jenkins  
cd /etc/default  
sudo vi jenkins #chnage port HTTP\_PORT=8090 and save and exit  
cd /lib/systemd/system  
sudo vi jenkins.service #change Environments="Jenkins\_port=8090" save and exit  
sudo systemctl daemon-reload  
sudo systemctl restart jenkins  
sudo systemctl status jenkins

Now, grab your Public IP Address

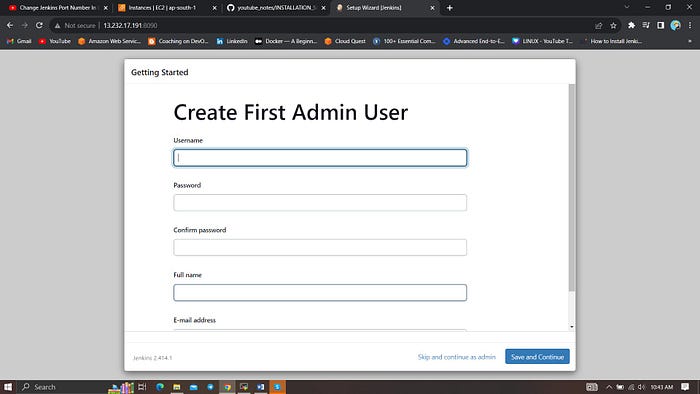
<EC2 Public IP Address:8090>  
sudo cat /var/lib/jenkins/secrets/initialAdminPassword



Unlock Jenkins using an administrative password and install the suggested plugins.

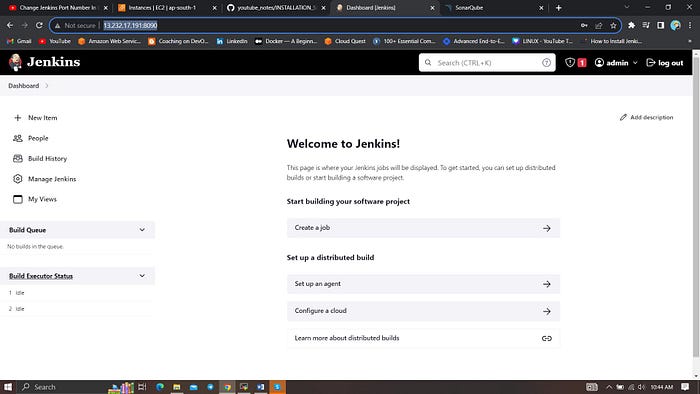


Jenkins will now get installed and install all the libraries.



Create a user click on save and continue.

Jenkins Getting Started Screen.

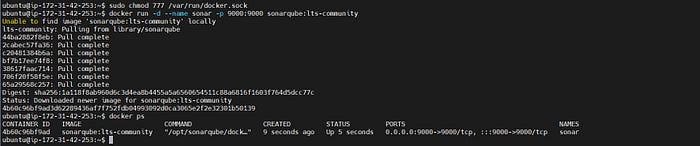


**2B — Install Docker**

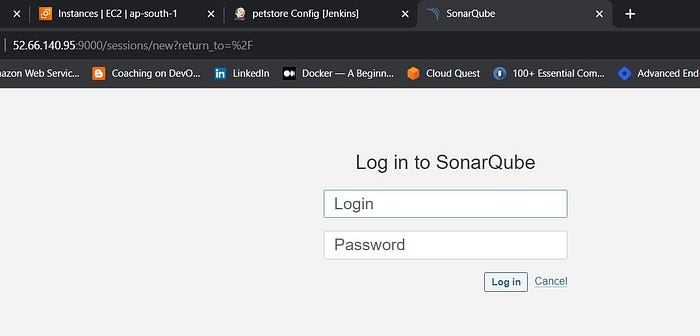
sudo apt-get update  
sudo apt-get install docker.io -y  
sudo usermod -aG docker $USER #my case is ubuntu  
newgrp docker  
sudo chmod 777 /var/run/docker.sock

After the docker installation, we create a sonarqube container (Remember added 9000 ports in the security group).

docker run -d --name sonar -p 9000:9000 sonarqube:lts-community

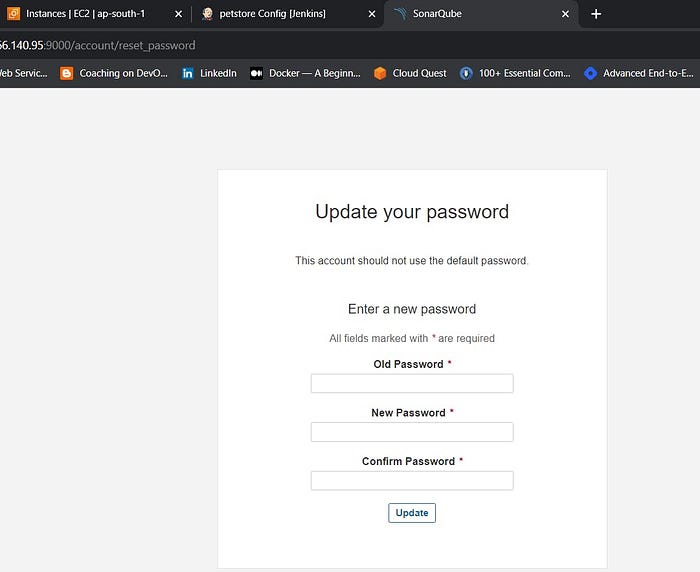


Now our sonarqube is up and running

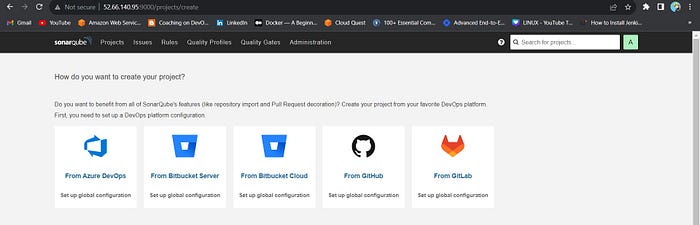


Enter username and password, click on login and change password

username admin  
password admin



Update New password, This is Sonar Dashboard.



**2C — Install Trivy**

vi trivy.sh

sudo apt-get install wget apt-transport-https gnupg lsb-release -y  
  
wget -qO - https://aquasecurity.github.io/trivy-repo/deb/public.key | gpg --dearmor | sudo tee /usr/share/keyrings/trivy.gpg > /dev/null  
  
echo "deb [signed-by=/usr/share/keyrings/trivy.gpg] https://aquasecurity.github.io/trivy-repo/deb $(lsb\_release -sc) main" | sudo tee -a /etc/apt/sources.list.d/trivy.list  
  
sudo apt-get update  
  
sudo apt-get install trivy -y

Next, we will log in to Jenkins and start to configure our Pipeline in Jenkins

**Step 3 — Install Plugins like JDK, Sonarqube Scanner, Maven, OWASP Dependency Check**

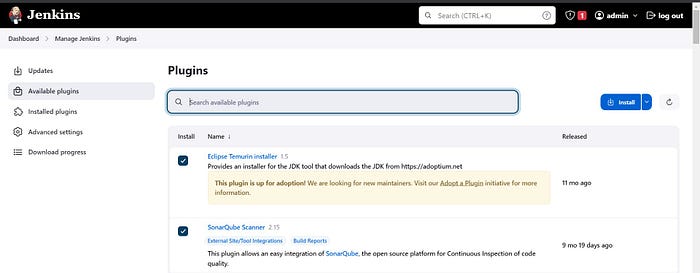
**3A — Install Plugin**

Goto Manage Jenkins →Plugins → Available Plugins →

Install below plugins

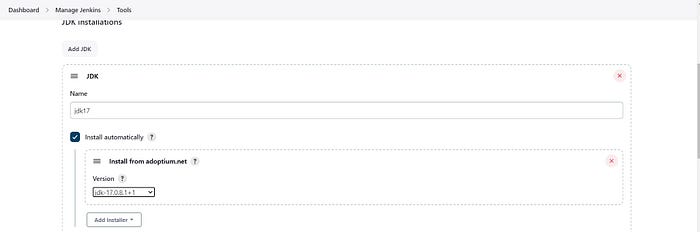
1 → Eclipse Temurin Installer (Install without restart)

2 → SonarQube Scanner (Install without restart)



**3B — Configure Java and Maven in Global Tool Configuration**

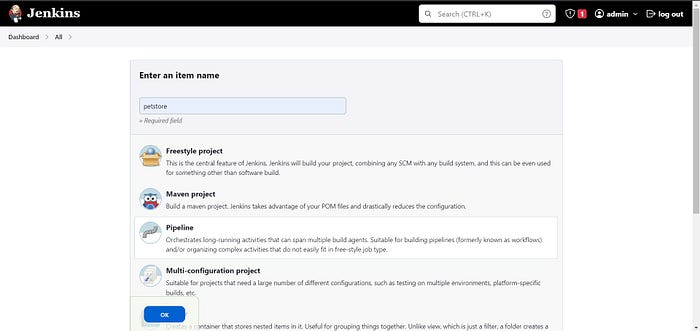
Goto Manage Jenkins → Tools → Install JDK(17) and Maven3(3.6.0) → Click on Apply and Save





**3C — Create a Job**

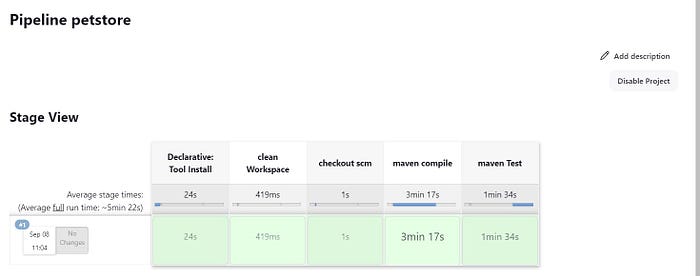
Label it as Petstore, click on Pipeline and OK.



Enter this in Pipeline Script,

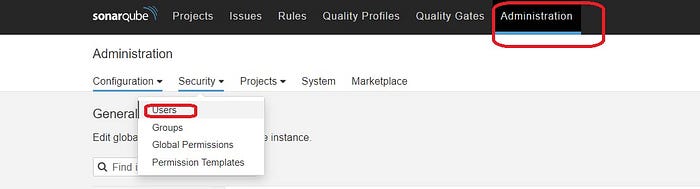
pipeline{  
 agent any  
 tools {  
 jdk 'jdk17'  
 maven 'maven3'  
 }  
 stages{  
 stage ('clean Workspace'){  
 steps{  
 cleanWs()  
 }  
 }  
 stage ('checkout scm') {  
 steps {  
 git 'https://github.com/Aj7Ay/jpetstore-6.git'  
 }  
 }  
 stage ('maven compile') {  
 steps {  
 sh 'mvn clean compile'  
 }  
 }  
 stage ('maven Test') {  
 steps {  
 sh 'mvn test'  
 }  
 }  
 }  
}

The stage view would look like this,

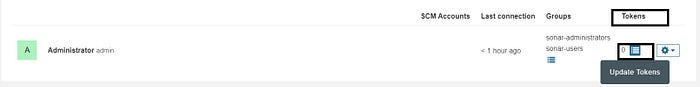


**Step 4 — Configure Sonar Server in Manage Jenkins**

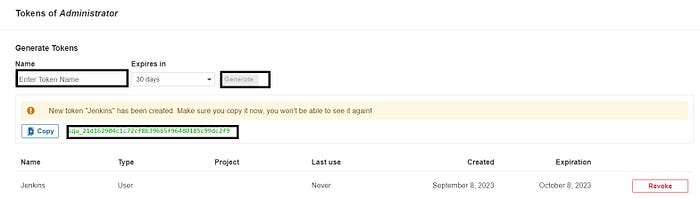
Grab the Public IP Address of your EC2 Instance, Sonarqube works on Port 9000, so <Public IP>:9000. Goto your Sonarqube Server. Click on Administration → Security → Users → Click on Tokens and Update Token → Give it a name → and click on Generate Token



click on update Token

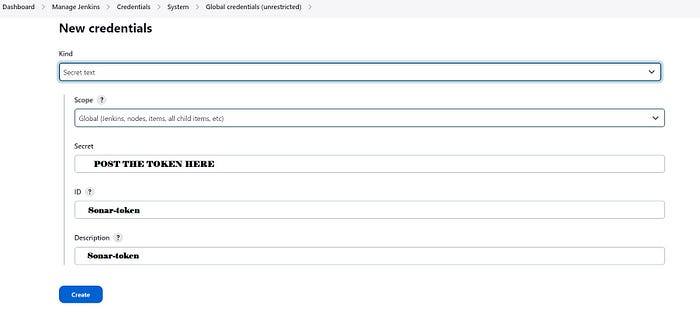


Create a token with a name and generate



copy Token

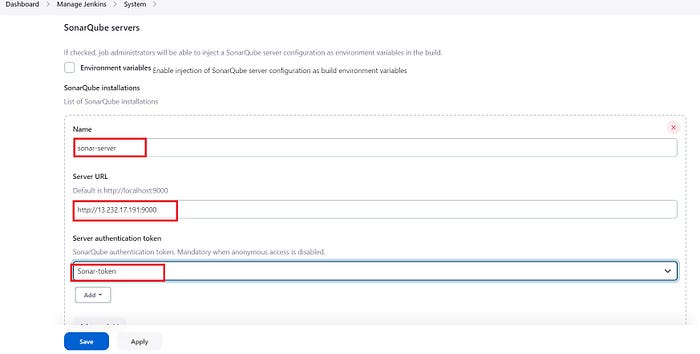
Goto Jenkins Dashboard → Manage Jenkins → Credentials → Add Secret Text. It should look like this



You will this page once you click on create



Now, go to Dashboard → Manage Jenkins → System and Add like the below image.

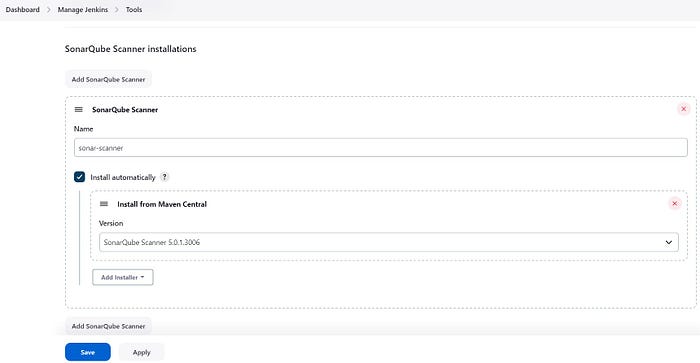


Click on Apply and Save

The Configure System option is used in Jenkins to configure different server

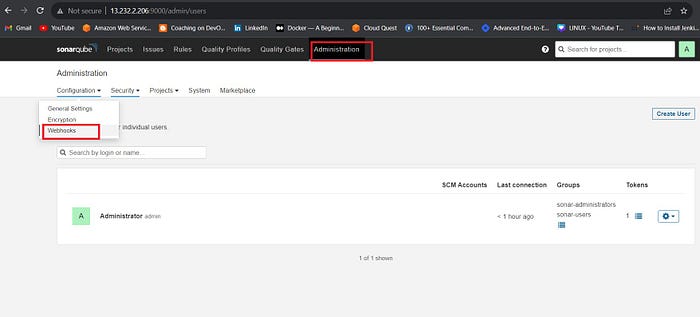
Global Tool Configuration is used to configure different tools that we install using Plugins

We will install a sonar scanner in the tools.

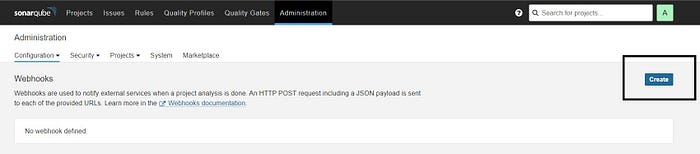


In the Sonarqube Dashboard add a quality gate also

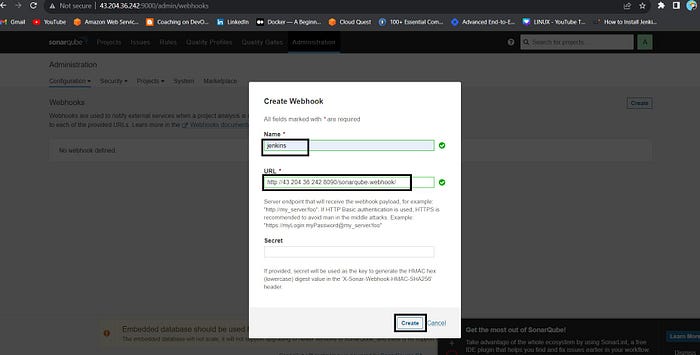
Administration → Configuration →Webhooks



Click on Create

Add details

#in url section of quality gate  
<http://jenkins-public-ip:8090>/sonarqube-webhook/



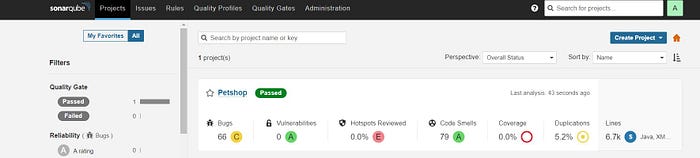
Let’s go to our Pipeline and add Sonarqube Stage in our Pipeline Script.

#under tools section add this environment  
environment {  
 SCANNER\_HOME=tool 'sonar-scanner'  
 }  
# in stages add this  
stage("Sonarqube Analysis "){  
 steps{  
 withSonarQubeEnv('sonar-server') {  
 sh ''' $SCANNER\_HOME/bin/sonar-scanner -Dsonar.projectName=Petshop \  
 -Dsonar.java.binaries=. \  
 -Dsonar.projectKey=Petshop '''  
 }  
 }  
 }  
 stage("quality gate"){  
 steps {  
 script {  
 waitForQualityGate abortPipeline: false, credentialsId: 'Sonar-token'   
 }  
 }  
 }

Click on Build now, you will see the stage view like this



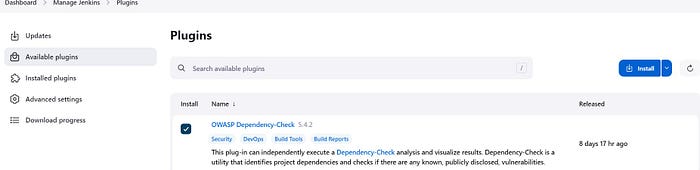
To see the report, you can go to Sonarqube Server and go to Projects.



You can see the report has been generated and the status shows as passed. You can see that there are 6.7k lines. To see a detailed report, you can go to issues.

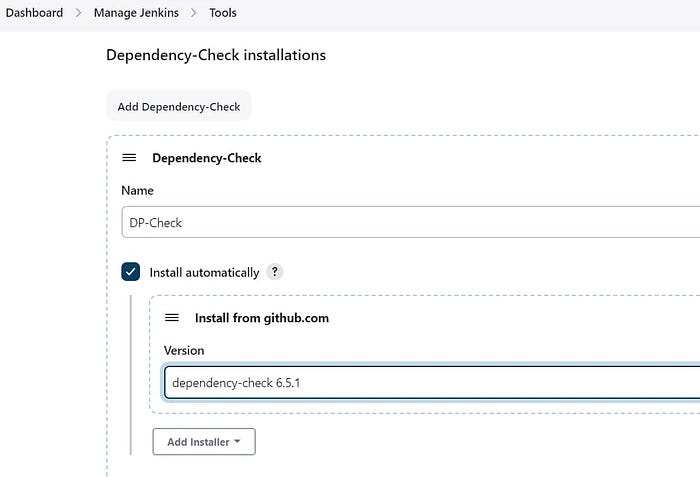
**Step 5 — Install OWASP Dependency Check Plugins**

GotoDashboard → Manage Jenkins → Plugins → OWASP Dependency-Check. Click on it and install it without restart.



First, we configured the Plugin and next, we had to configure the Tool

Goto Dashboard → Manage Jenkins → Tools →



Click on Apply and Save here.

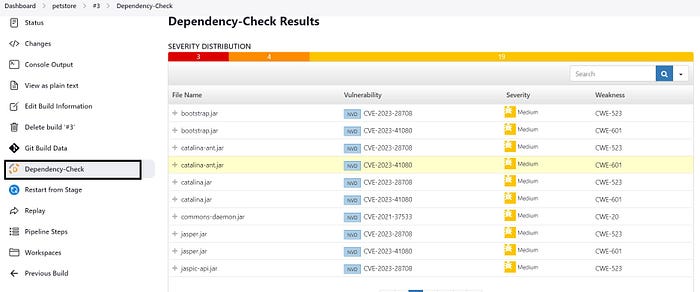
Now go configure → Pipeline and add this stage to your pipeline and build.

stage ('Build war file'){  
 steps{  
 sh 'mvn clean install -DskipTests=true'  
 }  
 }  
 stage("OWASP Dependency Check"){  
 steps{  
 dependencyCheck additionalArguments: '--scan ./ --format XML ', odcInstallation: 'DP-Check'  
 dependencyCheckPublisher pattern: '\*\*/dependency-check-report.xml'  
 }  
 }

The stage view would look like this,



You will see that in status, a graph will also be generated and Vulnerabilities.



**Step 6 — Docker plugin and credential Setup**

We need to install the Docker tool in our system, Goto Dashboard → Manage Plugins → Available plugins → Search for Docker and install these plugins

Docker

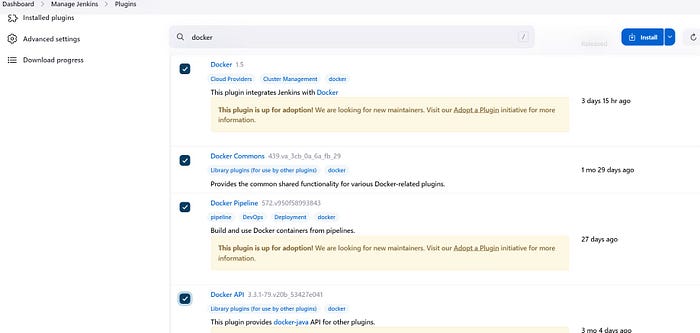
Docker Commons

Docker Pipeline

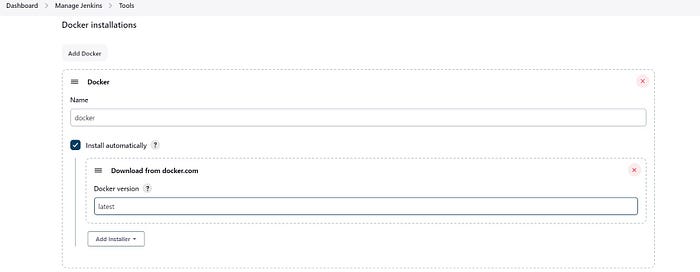
Docker API

docker-build-step

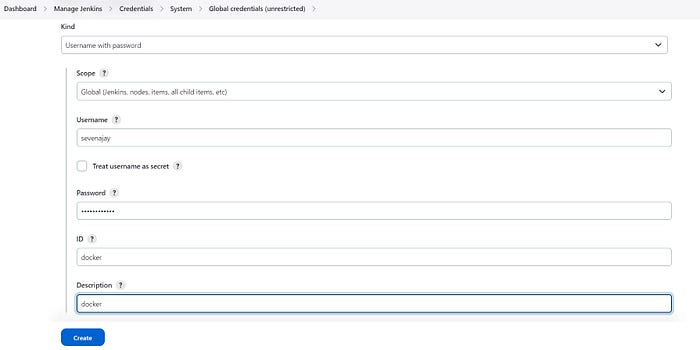
and click on install without restart



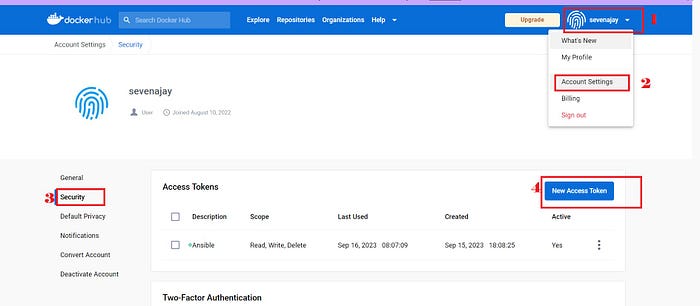
Now, goto Dashboard → Manage Jenkins → Tools →



Add DockerHub Username and Password under Global Credentials



create a personal Access token from the docker hub which is used for ansible-playbook



copy it and save for later.

Let’s install Ansible on the Jenkins server

**STEP 7 -Adding Ansible Repository in Ubuntu**

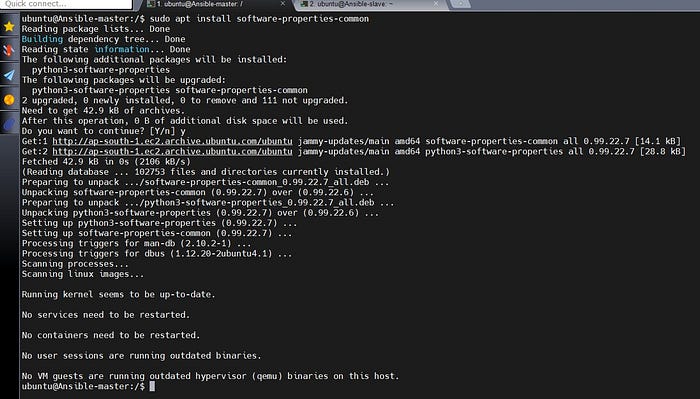
Now we are going to run the below commands on the Ansible server

Step1:Update your system packages:

sudo apt-get update

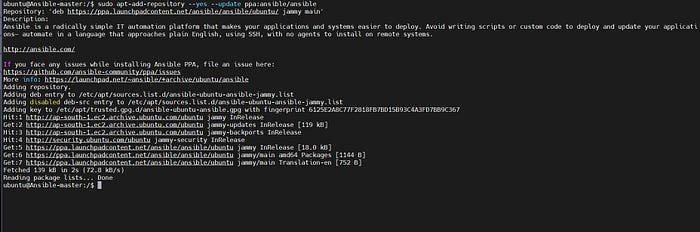
Step 2: First Install Required packages to install Ansible.

sudo apt install software-properties-common



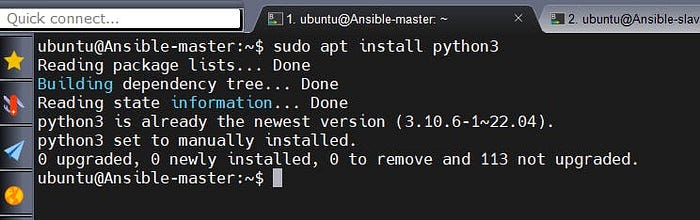
Step3: Add the ansible repository via PPA

sudo add-apt-repository --yes --update ppa:ansible/ansible



Install Python3 on the Ansible master

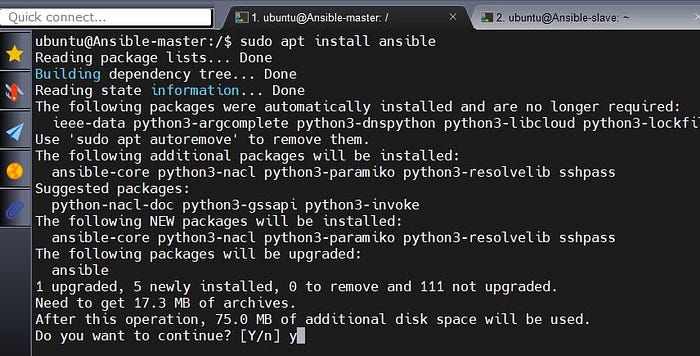
sudo apt install python3



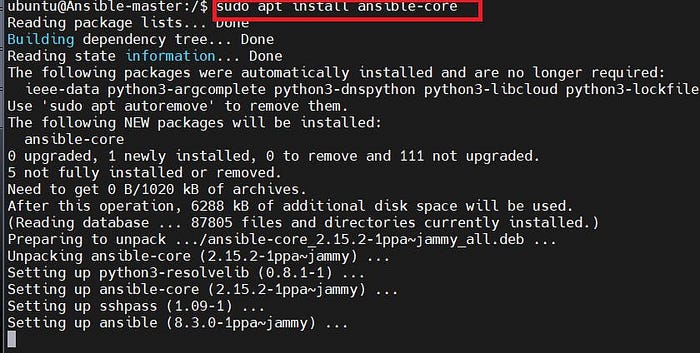
**Install Ansible on Ubuntu 22.04 LTS**

Step1: Install Ansible on Ubuntu 22.04 LTS

sudo apt install ansible -y



sudo apt install ansible-core -y



Step2: To check version :

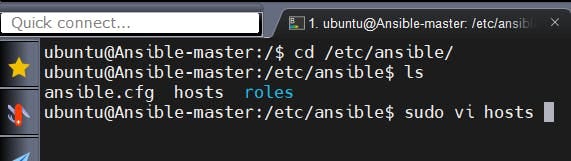
ansible --version

**Create an Inventory file in Ansible**

To add inventory you can create a new directory or add in the default Ansible hosts file

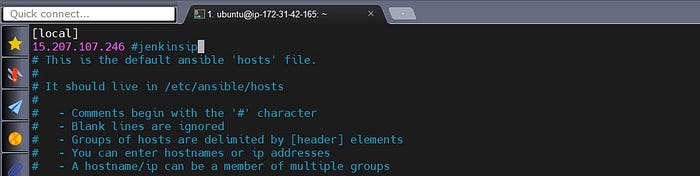
cd /etc/ansible  
sudo vi hosts

Now go to the host file inside the Ansible server and paste the public IP of the Jenkins



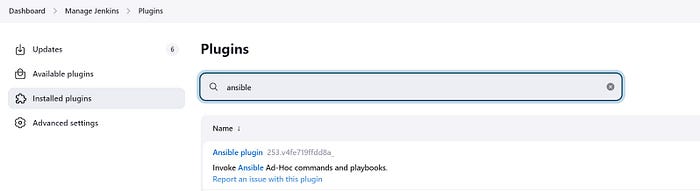
You can create a group and paste ip address below:

[local]#any name you want  
Ip of Jenkins

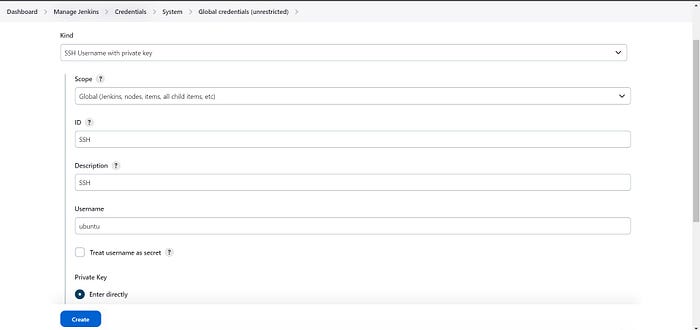


save and exit from the file.

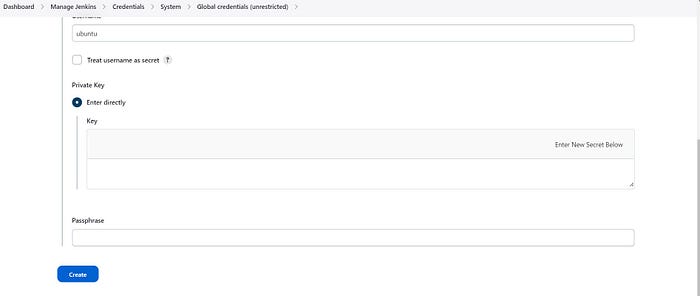
Let’s install The Ansible plugin to integrate with Jenkins.



Now add Credentials to invoke Ansible with Jenkins.



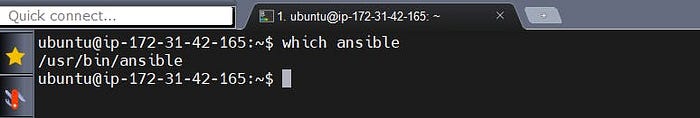
In the private key section, Select Enter directly and add your Pem file for the key.



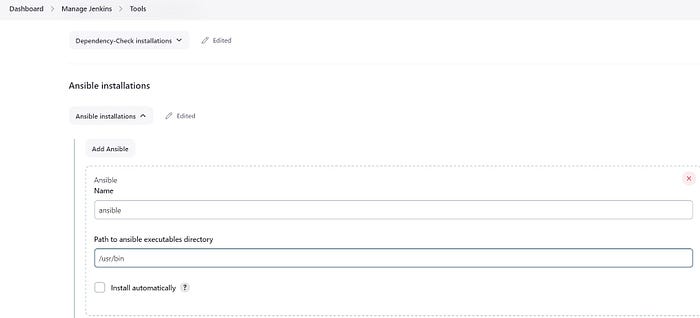
and finally, click on Create.

Give this command in your Jenkins machine to find the path of your ansible which is used in the tool section of Jenkins.

which ansible

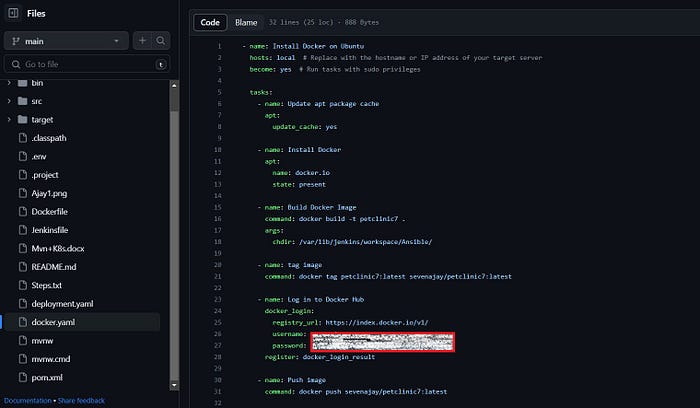


Copy that path and add it to the tools section of Jenkins at ansible installations.



Now write an Ansible playbook to create a docker image, tag it and push it to the docker hub, and finally, we will deploy it on a container using Ansible.

Just sample code.

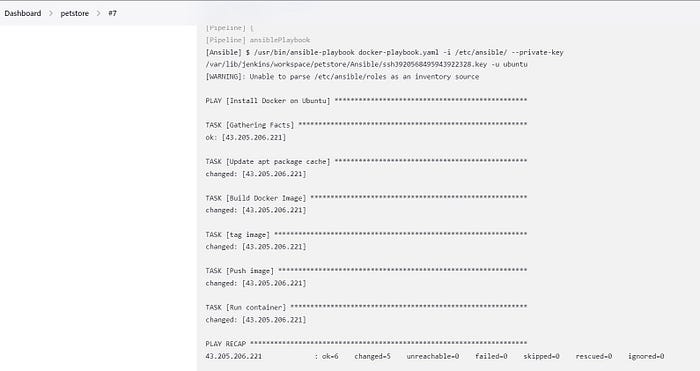


- name: docker build and push  
 hosts: docker # Replace with the hostname or IP address of your target server  
 become: yes # Run tasks with sudo privileges  
  
 tasks:  
 - name: Update apt package cache  
 apt:  
 update\_cache: yes   
  
 - name: Build Docker Image  
 command: docker build -t petstore .  
 args:  
 chdir: /var/lib/jenkins/workspace/petstore  
  
 - name: tag image  
 command: docker tag petstore:latest sevenajay/petstore:latest   
  
 - name: Log in to Docker Hub  
 community.docker.docker\_login:  
 registry\_url: https://index.docker.io/v1/  
 username: sevenajay  
 password: <docker pat>  
  
 - name: Push image  
 command: docker push sevenajay/petstore:latest  
  
 - name: Run container  
 command: docker run -d --name pet1 -p 8081:8080 sevenajay/petstore:latest

Add this stage to the pipeline to build and push it to the docker hub, and run the container.

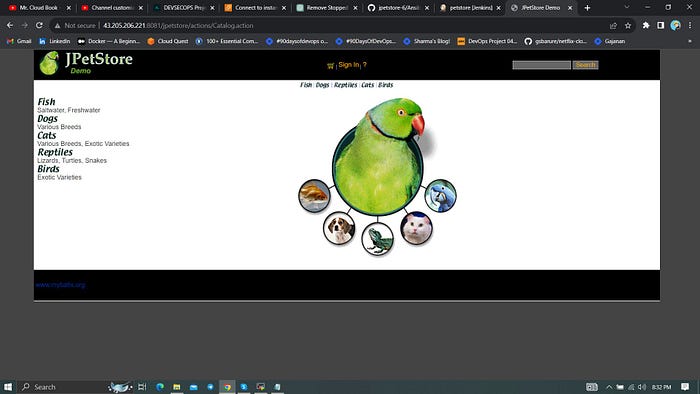
stage('Install Docker') {  
 steps {  
 dir('Ansible'){  
 script {  
 ansiblePlaybook credentialsId: 'ssh', disableHostKeyChecking: true, installation: 'ansible', inventory: '/etc/ansible/', playbook: 'docker-playbook.yaml'  
 }   
 }   
 }  
 }

Output of pipeline



Now copy the IP address of Jenkins and paste it into the browser

<jenkins-ip:8081>/jpetstore



**Step 8 — Kuberenetes Setup**

Connect your machines to Putty or Mobaxtreme

Take-Two Ubuntu 20.04 instances one for k8s master and the other one for worker.

Install Kubectl on Jenkins machine also.

**Kubectl on Jenkins to be installed**

sudo apt update  
sudo apt install curl  
curl -LO https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl  
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl  
kubectl version --client

**Part 1 — — — — — Master Node — — — — — —**

sudo hostnamectl set-hostname K8s-Master

**— — — — — Worker Node — — — — — —**

sudo hostnamectl set-hostname K8s-worker

**Part 2 — — — — — — Both Master & Node — — — — —**

sudo apt-get update   
  
sudo apt-get install -y docker.io  
sudo usermod –aG docker Ubuntu  
newgrp docker  
sudo chmod 777 /var/run/docker.sock  
  
sudo curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -  
  
sudo tee /etc/apt/sources.list.d/kubernetes.list <<EOF  
deb https://apt.kubernetes.io/ kubernetes-xenial main  
EOF  
  
sudo apt-get update  
  
sudo apt-get install -y kubelet kubeadm kubectl  
  
sudo snap install kube-apiserver

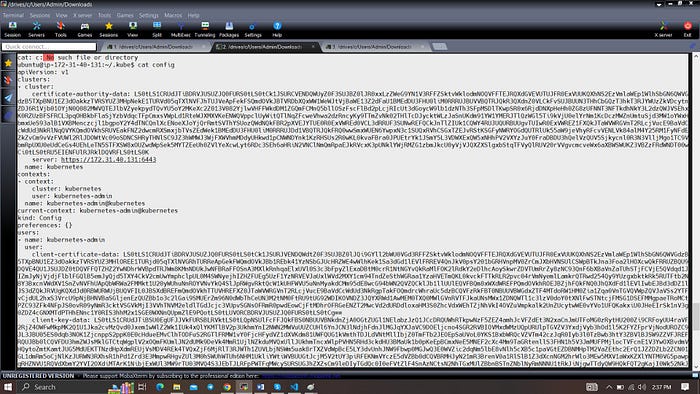
**Part 3 — — — — — — — — Master — — — — — — — -**

sudo kubeadm init --pod-network-cidr=10.244.0.0/16  
# in case your in root exit from it and run below commands  
mkdir -p $HOME/.kube  
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config  
sudo chown $(id -u):$(id -g) $HOME/.kube/config  
kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml

**— — — — — Worker Node — — — — — —**

sudo kubeadm join <master-node-ip>:<master-node-port> --token <token> --discovery-token-ca-cert-hash <hash>

Copy the config file to Jenkins master or the local file manager and save it

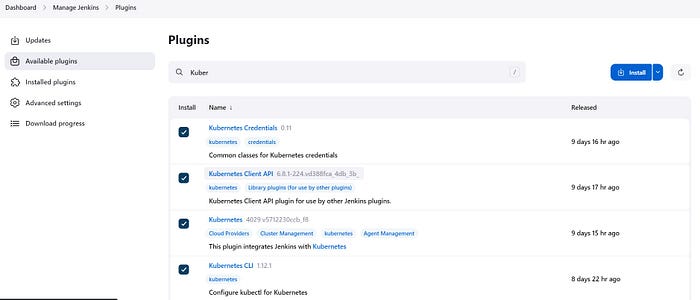


copy it and save it in documents or another folder save it as secret-file.txt

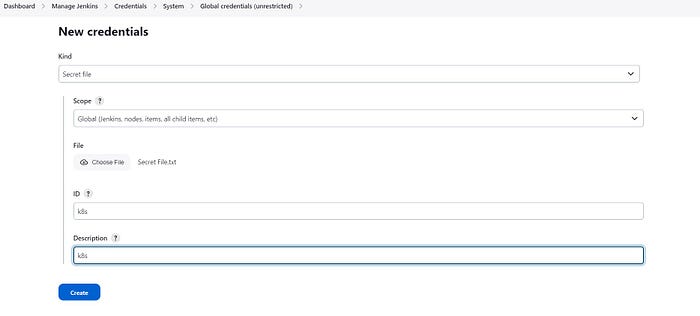
NOTE:

create a new textfile for the config file as secret-file.txt, store the copied above complete config details and add it in the credentials section.

Install Kubernetes Plugin, Once it’s installed successfully



go to manage Jenkins → manage credentials → Click on Jenkins global → add credentials



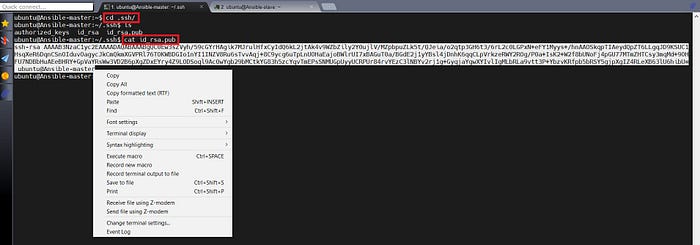
**STEP 9 — Master-slave Setup for Ansible and Kubernetes**

To communicate with the Kubernetes clients we have to generate an SSH key on the ansible master node and exchange it with K8s Master System.

ssh-keygen

Change the directory to .ssh and copy the public key (id\_[rsa.pub](http://rsa.pub/))

cd .ssh  
cat id\_rsa.pub #copy this public key



Once you copy the public key from the Ansible master, move to the Kubernetes machine change the directory to .ssh and paste the copied public key under authorized\_keys.

cd .ssh #on k8s master   
sudo vi authorized\_keys

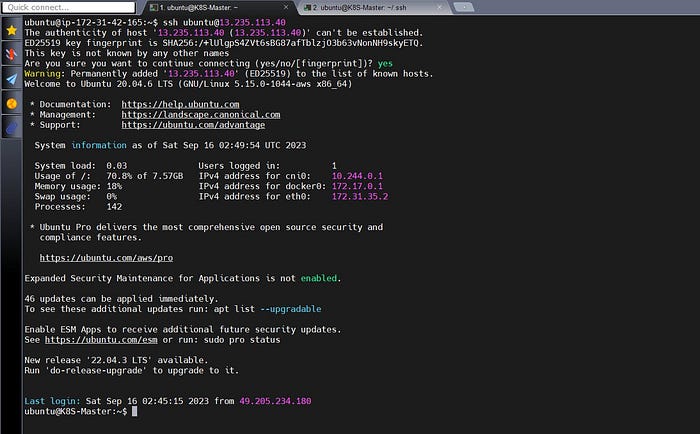


Note: Now, insert or paste the copied public key into the new line. make sure don’t delete any existing keys from the authorized\_keys file then save and exit.

By adding a public key from the master to the k8s machine we have now configured keyless access. To verify you can try to access the k8s master and use the command as mentioned in the below format.

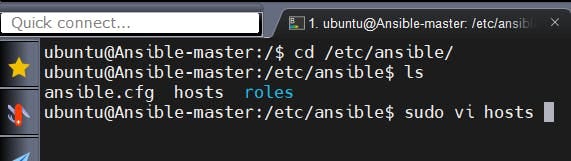
COPY

ssh ubuntu@<public-ip-k8s-master>



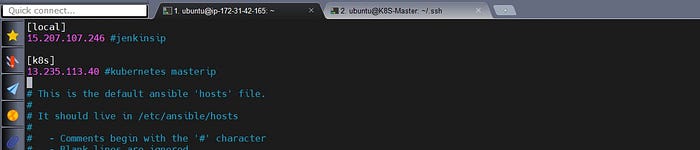
Verifying the above SSH connection from the master to the Kubernetes we have configured our prerequisites.

Now go to the host file inside the Ansible server and paste the public IP of the k8s master.



You can create a group and paste ip address below:

[k8s]#any name you want  
public ip of k8s-master

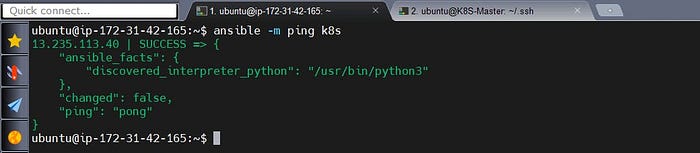


**Test Ansible Master Slave Connection**

Use the below command to check Ansible master-slave connections.

ansible -m ping k8s  
ansible -m ping all#use this one

If all configuration is correct then you would get below output.



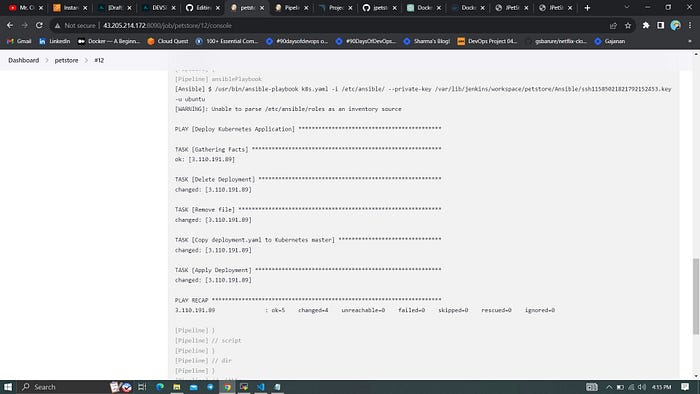
let’s create a simple ansible playbook for Kubernetes deployment.

---  
- name: Deploy Kubernetes Application  
 hosts: k8s # Replace with your target Kubernetes master host or group  
 gather\_facts: yes # Gather facts about the target host  
  
 tasks:  
 - name: Copy deployment.yaml to Kubernetes master  
 copy:  
 src: /var/lib/jenkins/workspace/petstore/deployment.yaml # Assuming Jenkins workspace variable  
 dest: /home/ubuntu/  
 become: yes # Use sudo for copying if required  
 become\_user: root # Use a privileged user for copying if required  
  
 - name: Apply Deployment  
 command: kubectl apply -f /home/ubuntu/deployment.yaml

Now add the below stage to your pipeline.

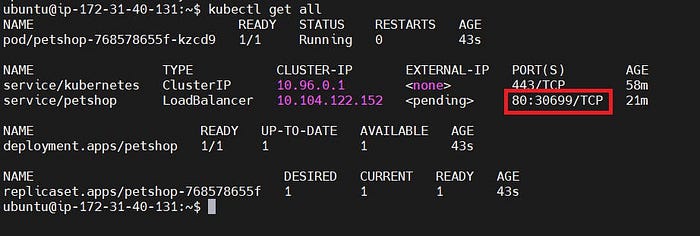
stage('k8s using ansible'){  
 steps{  
 dir('Ansible') {  
 script{  
 ansiblePlaybook credentialsId: 'ssh', disableHostKeyChecking: true, installation: 'ansible', inventory: '/etc/ansible/', playbook: 'kube.yaml'  
 }  
 }   
 }  
 }

output

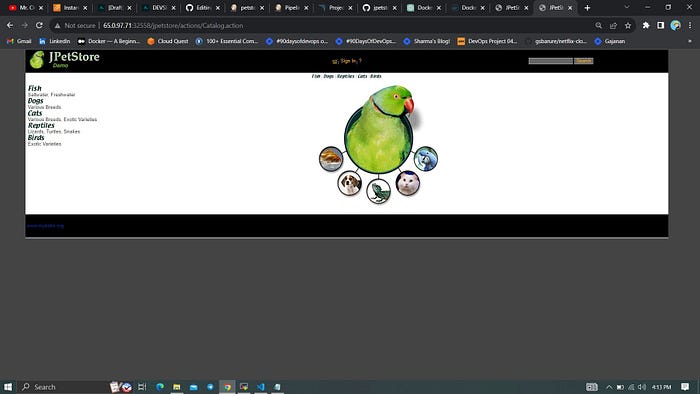


In the Kubernetes cluster give this command

kubectl get all  
kubectl get svc



<slave-ip:serviceport(30699)>/jpetstore



complete Pipeline

pipeline{  
 agent any  
 tools {  
 jdk 'jdk17'  
 maven 'maven3'  
 }  
 environment {  
 SCANNER\_HOME=tool 'sonar-scanner'  
 }  
 stages{  
 stage ('clean Workspace'){  
 steps{  
 cleanWs()  
 }  
 }  
 stage ('checkout scm') {  
 steps {  
 git 'https://github.com/Aj7Ay/jpetstore-6.git'  
 }  
 }  
 stage ('maven compile') {  
 steps {  
 sh 'mvn clean compile'  
 }  
 }  
 stage ('maven Test') {  
 steps {  
 sh 'mvn test'  
 }  
 }  
 stage("Sonarqube Analysis "){  
 steps{  
 withSonarQubeEnv('sonar-server') {  
 sh ''' $SCANNER\_HOME/bin/sonar-scanner -Dsonar.projectName=Petstore \  
 -Dsonar.java.binaries=. \  
 -Dsonar.projectKey=Petstore '''  
 }  
 }  
 }  
 stage("quality gate"){  
 steps {  
 script {  
 waitForQualityGate abortPipeline: false, credentialsId: 'Sonar-token'   
 }  
 }  
 }  
 stage ('Build war file'){  
 steps{  
 sh 'mvn clean install -DskipTests=true'  
 }  
 }  
 stage("OWASP Dependency Check"){  
 steps{  
 dependencyCheck additionalArguments: '--scan ./ --format XML ', odcInstallation: 'DP-Check'  
 dependencyCheckPublisher pattern: '\*\*/dependency-check-report.xml'  
 }  
 }  
 stage('Ansible docker Docker') {  
 steps {  
 dir('Ansible'){  
 script {  
 ansiblePlaybook credentialsId: 'ssh', disableHostKeyChecking: true, installation: 'ansible', inventory: '/etc/ansible/', playbook: 'docker.yaml'  
 }   
 }   
 }  
 }  
 stage('k8s using ansible'){  
 steps{  
 dir('Ansible') {  
 script{  
 ansiblePlaybook credentialsId: 'ssh', disableHostKeyChecking: true, installation: 'ansible', inventory: '/etc/ansible/', playbook: 'kube.yaml'  
 }  
 }   
 }  
 }  
 }  
}