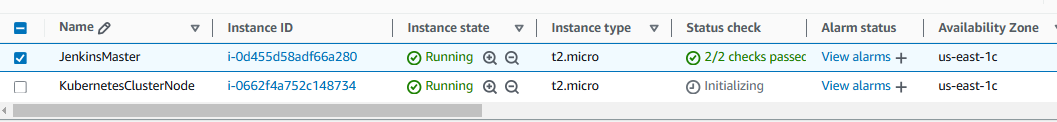
|  |
| --- |
| **Problem Statement**  A retail company Abstergo Corp. has recently setup an online shopping portal(website) to sell their products. Due to fierce competition, the company wants a solution that can reduce the time and effort it needs to enhance the functionality of their website on a regular basis. They are looking for an automated way to deploy the new code (for new features) to production website whenever they want.  ***Business Requirements***  ***•*** The team of developers working on new features will merge their code to a GitHub repo.  • As soon as the code reaches GitHub, using a CI (Continuous Integration) pipeline, setup in Jenkins, automated builds will be triggered.  • The automated builds will frequently deploy new features to the production website.  • Every build will prepare a Dockerfile and push docker images to docker-hub.  • Every docker image will be deployed (Continuous Deployment) to a kubernetes-cluster.  Fork the given repository to your own account and use it as the application for your pipeline project <https://github.com/bhavukm/cicd-pipeline-jenkins-complete> |

***Solution Implementation***

# Step 0: Create 2 Ubuntu VMs in aws: 1 Jenkins Master 2 Kubernetes Cluster Node

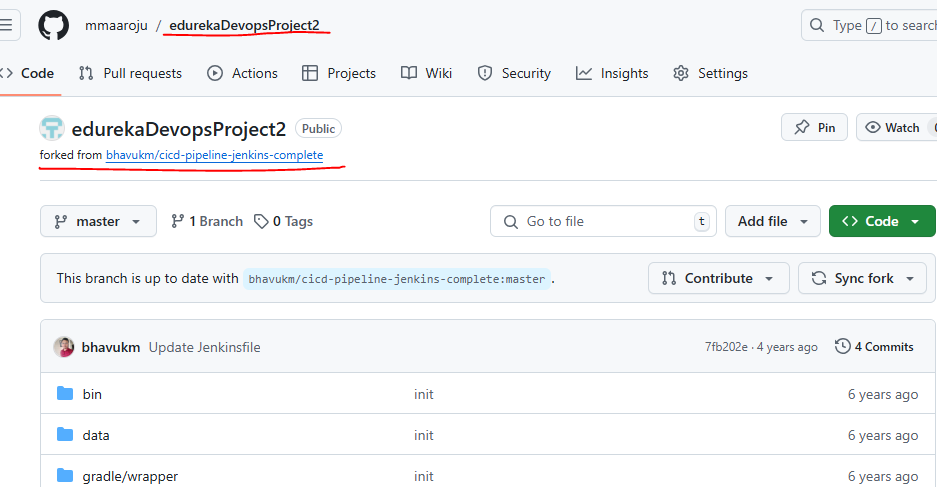
**VM1 (Jenkins Master): This VM will host Jenkins and manage the pipeline.**

**VM 2( Kubernetes Cluster Node): Install Ubuntu on this VM as well, configuring it to act as both the master and worker node for the Kubernetes cluster.**



# Step 1: Fork the GitHub Repository

The repository [bhavukm/cicd-pipeline-jenkins-complete: A complete jenkins pipeline example](https://github.com/bhavukm/cicd-pipeline-jenkins-complete) is forked to <https://github.com/mmaaroju/edurekaDevopsProject2> as shown below:



# Step 2: Setting up the Jenkins Master VM

Install Pre-requisites Java Git ,Jenkins, Ansible in the JenkinsMaster

|  |
| --- |
| 1. ***Install the pre-requisites :***  # Install Jdk:  sudo -i  sudo apt update -y  sudo apt install openjdk-17-jre -y # previous version  java -version      # Install Git  sudo apt install git -y # To install git if required  2. ***Setting Environment Variables***  export JAVA\_HOME=/usr/lib/jvm/java-1.17.0-openjdk-amd64  3. ***Install the Jenkins tool :***  sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \  https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key  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  sudo apt-get install jenkins -y  ***4. Install Docker***  sudo apt install -y docker.io  sudo systemctl start docker  sudo systemctl enable docker  sudo usermod -aG docker Jenkins  sudo su - jenkins  docker ps    5. ***Perform Post\_Installation Activities :***  jenkins --version    systemctl status jenkins  systemctl start jenkins  systemctl enable jenkins  /var/lib/jenkins # Default Installation Dir of Jenkin on Linux Box    Enable Port 8080 in Inbound rules:    Open web browser :  http://<Public\_IP\_Address>:8080/    Copy the secret from the file initialAdminPassword in “ path “/var/lib/Jenkins/secrets”    User name/password: jenkinsadmin  ***6.Install required Jenkins plugins: GitHub Integration, Docker Pipeline, Kubernetes Continuous Deploy, and Pipeline plugins.*** |

# Step 3: Set Up Docker Hub Repository and Token Generation

|  |
| --- |
| * Loginto Docker Hub and Create a new repository in to store Docker images (e.g., mmaaroju/abstergo-portal). * Generate an access token in Docker Hub under Account Settings > Personal access tokens > New Access Token >     Access token description: edureka\_devops  Run : docker login -u mmaaroju  personal access token: dckr\_pat\_eSCcpQhBCBHa\_QBeW4b0kw24vLo  Later the above Docker Hub credentials are configured in Jenkins as part of Step6: |

Step 4: Set Up Kubernetes on VM 2

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ***1. Update System Packages***   |  | | --- | | sudo apt update  sudo apt upgrade -y  sudo swapoff -a  sudo sed -i '/ swap / s/^\(.\*\)$/#\1/g' /etc/fstab  #Rather than above step Find any line that starts with swap and add a # at the beginning to comment it out.  #sudo vim /etc/fstab |   ***2.Install Docker***   |  | | --- | | sudo apt-get update  sudo apt install docker.io -y    sudo systemctl enable docker  sudo systemctl start docker |   ***3. Load the br\_netfilter module required for networking.***   |  | | --- | | sudo modprobe overlay  sudo modprobe br\_netfilter  cat <<EOF | sudo tee /etc/modules-load.d/containerd.conf  overlay  br\_netfilter  EOF    **#Configure sysctl to enable packet filtering on bridges**: Create the configuration file /etc/sysctl.d/k8s.conf to enable the bridge network filtering:  #**To allow iptables to see bridged traffic, as required by Kubernetes, we need to set the values of certain fields to 1.**  sudo tee /etc/sysctl.d/k8s.conf<<EOF  net.bridge.bridge-nf-call-ip6tables = 1  net.bridge.bridge-nf-call-iptables = 1  net.ipv4.ip\_forward = 1  EOF    ~~sudo sh -c "echo 'net.bridge.bridge-nf-call-iptables = 1' > /etc/sysctl.d/k8s.conf"~~  ~~sudo sh -c "echo 'net.bridge.bridge-nf-call-ip6tables = 1' >> /etc/sysctl.d/k8s.conf"~~  **#Ensure the br\_netfilter module is loaded at boot**: Append the following line to /etc/modules-load.d/k8s.conf:  sudo sh -c "echo 'br\_netfilter' > /etc/modules-load.d/k8s.conf"    **#Check that the br\_netfilter module is loaded**:  lsmod | grep br\_netfilter |   ***4. Install curl.***   |  | | --- | | **sudo apt install curl -y**  **#Get the apt-key and then add the repository from which we will install containerd.**  **curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -**  **sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable"** |   5***. Install containerd,***   |  | | --- | | **#Update and then install the containerd package.**  sudo apt update -y  sudo apt install -y containerd.io  #**Execute below command to know the path**  which containerd    sudo systemctl status containerd  **#Setup the default configuration file** for **containerd**:  sudo mkdir -p /etc/containerd  sudo containerd config default | sudo tee /etc/containerd/config.toml  **#Next up, we need to modify the containerd configuration file and ensure that the cgroupDriver is set to systemd. To do so, #edit the following file:**  sudo vim /etc/containerd/config.toml    sudo systemctl restart containerd  sudo systemctl status containerd |     ***6.Install Kubernetes (kubeadm, kubelet, kubectl)***   |  | | --- | | **#Add Kubernetes apt repository**:  ~~sudo apt-get update && sudo apt-get install -y apt-transport-https curl~~  ~~curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -~~  sudo apt-get update  #**apt-transport-https may be a dummy package; if so, you can skip that package**  sudo apt-get install -y apt-transport-https ca-certificates curl gpg  **# If the directory `/etc/apt/keyrings` does not exist, it should be created before the curl command, read the note below.**  #sudo mkdir -p -m 755 /etc/apt/keyrings  #mkdir /etc/apt/keyrings  sudo mkdir -p -m 755 /etc/apt/keyrings  curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.28/deb/Release.key | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg    # **This overwrites any existing configuration in /etc/apt/sources.list.d/kubernetes.list**  echo "deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.28/deb/ /" | sudo tee /etc/apt/sources.list.d/kubernetes.list  sudo apt-get update  **#Install kubelet, kubeadm, and kubectl**:  sudo apt-get install -y kubelet kubeadm kubectl  **#Finally, enable the kubelet service on both systems so we can start it.**  sudo systemctl enable kubelet  **# When a package is on hold, it will not be upgraded automatically, even if newer versions are available.**  sudo apt-mark hold kubelet kubeadm kubectl |   ***7.Initialize Kubernetes Cluster***   |  | | --- | | #**Initialize the cluster**:  sudo kubeadm init --apiserver-advertise-address=172.31.89.180 --pod-network-cidr=192.168.0.0/16 --ignore-preflight-errors=NumCPU --ignore-preflight-errors=Mem    ***Save this:***  kubeadm join 172.31.89.180:6443 --token gwxj8a.bdjn5bnwxow2xi3g \  --discovery-token-ca-cert-hash sha256:5b65322e2fe52187852f40af70b83faafe70e8248d8e95e6f6b16349775f81fd |   ***8.Configure kubectl for Default User***   |  | | --- | | **#To start using your cluster, you need to run the following as a regular user:**  **#Set up kubeconfig for the Default User**  mkdir -p $HOME/.kube  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config  sudo chown $(id -u):$(id -g) $HOME/.kube/config |   ***9.Join Worker Node***   |  | | --- | | **Retrieve the existing token from MASTER Node :**  **The Token is already created in “*Initialize Kubernetes Cluster”* step when the control plane is created above. If not use below command to create.**  #sudo kubeadm token create --print-join-command  sudo kubeadm token list    **Retrieve the CA Certificate Hash from Master Node**  **openssl x509 -pubkey -in /etc/kubernetes/pki/ca.crt | openssl rsa -pubin -outform der 2>/dev/null | sha256sum | awk '{print $1}'**  **#Then you can join any number of worker nodes by running the following on each as root:**  kubeadm join 172.31.89.180:6443 --token gwxj8a.bdjn5bnwxow2xi3g  --discovery-token-ca-cert-hash sha256:5b65322e2fe52187852f40af70b83faafe70e8248d8e95e6f6b16349775f81fd  **#You will see a kubeadm join at the end of the output. Copy and save it in some file. We will have to run this command on the worker node to allow it to join the cluster. If you forget to save it, or misplace it, you can also regenerate it using this command:**  **#sudo kubeadm token create --print-join-command**  **#Verify kubelet is running:**  **sudo systemctl status kubelet** |   ***10.Install Pod Network (Flannel)***   |  | | --- | | **#You should now deploy a pod network to the cluster.** **This is required to interconnect the different Kubernetes components.**  kubectl apply -f <https://docs.projectcalico.org/manifests/calico.yaml>  **#Use the get nodes command to verify that our node is ready.**  kubectl get nodes    **#Also check whether all the default pods are running:**  kubectl get pods --all-namespaces | |

Step 5: Configure Kubernetes Deployment and Service Files

|  |  |  |  |
| --- | --- | --- | --- |
| ***1. Create Kubernetes Deployment (deployment.yaml):***   |  | | --- | | apiVersion: apps/v1  kind: Deployment  metadata:  name: abstergo-deployment  labels:  app: abstergo  spec:  replicas: 2  selector:  matchLabels:  app: abstergo  template:  metadata:  labels:  app: abstergo  spec:  containers:  - name: abstergo-container  image: mmaaroju/abstergo-portal:latest  ports:  - containerPort: 8081  livenessProbe:  httpGet:  path: /  port: 8081  initialDelaySeconds: 15  timeoutSeconds: 1  periodSeconds: 10  resources:  requests:  cpu: 200m |     ***2. Create Kubernetes Service (service.yaml):***   |  | | --- | | apiVersion: v1  kind: Service  metadata:  name: abstergo-service  spec:  type: NodePort  selector:  app: abstergo  ports:  - protocol: TCP  port: 8081  targetPort: 8081 |     ***3. Apply the Configuration:***  Run the following commands on your Kubernetes VM   |  | | --- | | kubectl apply -f deployment.yaml  kubectl apply -f service.yaml | |

Step 6: Set Up Docker Hub and Kubernetes Credentials in Jenkins

|  |  |
| --- | --- |
| ***1.Docker Hub Credentials:***  Go to Manage Jenkins > Manage Credentials.  Add new credentials with the ID **dockerhub-creds** (matching the ID in Jenkinsfile).  Use your Docker Hub username and password.    ***2.Kubernetes Config Credentials:***  To deploy Docker images to a Kubernetes cluster, Jenkins needs access to your Kubernetes cluster configuration file (typically called kubeconfig). This file allows Jenkins to authenticate with Kubernetes and execute deployment commands.   1. In Step4 Subsection 8 ~/.kube/config is set up.        1. **Establish connection between KubernetestClusterNode and Jenkins**  |  | | --- | | Check SSH Keys on the Source (Kubernetes VM):  ls ~/.ssh/  If no files are listed, generate an SSH key pair:  **ssh-keygen -t ecdsa -b 521**    Extract public key from ecdsa.pub and copy:  ~/.ssh$ cat id\_ecdsa.pub    ecdsa-sha2-nistp521 AAAAE2VjZHNhLXNoYTItbmlzdHA1MjEAAAAIbmlzdHA1MjEAAACFBAG6l/jbz+uhxp/UY1rREBInBIRFPBNhuzbj7fqVY1yTtSuGdFP7vRr0z4cdAtjFwUpISI6an5hJMJAa2NKKnZsU9QErvYa0QxciB0VsPDVxTyBdJERmSyAXdFrXuacgdUs9xmDc2i7vMhMzcF6AS92kTzNyUhFARQgEuA0IGHo4L4xy0A== root@ip-172-31-89-180  Now loginto Jenkins Server on the **Jenkins server,** add this key to the ~/.ssh/**authorized\_keys** file for the target user:  echo "ecdsa-sha2-nistp521 AAAAE2VjZHNhLXNoYTItbmlzdHA1MjEAAAAIbmlzdHA1MjEAAACFBAG6l/jbz+uhxp/UY1rREBInBIRFPBNhuzbj7fqVY1yTtSuGdFP7vRr0z4cdAtjFwUpISI6an5hJMJAa2NKKnZsU9QErvYa0QxciB0VsPDVxTyBdJERmSyAXdFrXuacgdUs9xmDc2i7vMhMzcF6AS92kTzNyUhFARQgEuA0IGHo4L4xy0A== root@ip-172-31-89-180" >> ~/.ssh/**authorized\_keys**  Test the SSH connection to ensure that the Kubernetes VM can access the Jenkins server. Execute below command in Kubernetes VM and the IP 172.31.85.246 is Private ID of JenkinsServer to connect remotely:  **ssh** [**root@172.31.85.246**](mailto:root@172.31.85.246)    **Once SSH access is working, run the scp command again to securely transfer files between the Kubernetes VM and Jenkins server:**  **scp ~/.kube/config root@172.31.85.246:~/.kube/config**    Now login to JenkinsMaster to see that this config file securely tranfered in .kube folder: |      1. In Jenkins, go to **Manage Jenkins > Manage Credentials,** Under the same scope as above, click **Add Credentials**. Set **Kind** to **Secret Text**. |

Step 7: Create the Jenkinsfile in Your Repository

|  |
| --- |
| ***1. Create a*** Jenkinsfile ***in the root of your GitHub repository, which will define the pipeline steps.***  pipeline {  agent any  environment {  DOCKER\_HUB\_REPO = ‘mmaaroju/abstergo-portal'  DOCKER\_CREDENTIALS\_ID = 'dockerhub-creds'  }  stages {  stage('Checkout Code') {  steps {  git branch: 'main', url: ' https://github.com/mmaaroju/edurekaDevopsProject2.git'  }  }  stage('Build Docker Image') {  steps {  script {  dockerImage = docker.build("${DOCKER\_HUB\_REPO}:${env.BUILD\_ID}")  }  }  }  stage('Push Docker Image to Docker Hub') {  steps {  script {  docker.withRegistry('https://registry.hub.docker.com', DOCKER\_CREDENTIALS\_ID) {  dockerImage.push("${env.BUILD\_ID}")  dockerImage.push("latest")  }  }  }  }  stage('Deploy to Kubernetes') {  steps {  withKubeConfig([credentialsId: 'k8s-credentials', contextName: 'your-k8s-cluster']) {  script {  sh 'kubectl set image deployment/abstergo-deployment abstergo-container=${DOCKER\_HUB\_REPO}:${env.BUILD\_ID}'  }  }  }  }  }  post {  always {  cleanWs()  }  }  }  2. Push into the repository |

Step 8: Create a New CI/CD Pipeline Job in Jenkins

|  |
| --- |
| ***1. In Jenkins, click on New Item and Name the item (e.g., "*** ***AbstergoCI\_CDPipeline"). Select Pipeline as the job type and click OK.***  ***2. In the configuration page:***  Set the Pipeline definition to **Pipeline script from SCM**.  Select Git as the SCM and enter the URL of your forked GitHub repository (e.g., <https://github.com/mmaaroju/edurekaDevopsProject2.git> ).  Set the branch to main (or the branch where your main code resides) and Save the job configuration. |

# STEP 9: Configure Webhook in GitHub

|  |
| --- |
| ***1.Add Webhook in GitHub***  Go to **Settings** > **Webhooks** in your repository.  Click on Add Webhook and set the Payload URL as http://<VM1\_IP>:8080/github-webhook/.  Select application/json for content type and choose Just the push event.    ***2.Configure Jenkins Job to Listen to Webhook***  In Jenkins, create a new pipeline job.   * Enable **GitHub hook trigger for GITScm polling** under **Build Triggers**. This will allow Jenkins to listen for events from the webhook. |

# STEP 10: Test the CI/CD Pipeline

|  |
| --- |
| 1. Make a change in your repository (e.g., update README or modify code).  2.Commit and push the change to GitHub.  3.Go to Jenkins and observe the pipeline executing:  4.Each stage (Checkout Code, Docker Build, Docker Push, Kubernetes Deployment, Post Build Actions) are complete successfully.  Check Kubernetes to ensure the deployment and service are created.    5. Build is published to dockerhub as shown below screenshot    6. To know the deployed service , pods, replicaset in Kubernetes using kubectl get svc , kubectl get pods and kubectl get replicaset |