



INDIAN INSTITUTE OF
INFORMATION
TECHNOLOGY

DevOps and its Applications

CS457

Assignment-1

Jenkins

Under the Guidance of – Dr Uma S

Submitted By:

1. Aqtar parveez (18BCS010)
2. G K Bharath Bhushan (18BCS026)
3. K V S Chaitanya (18BCS045)
4. Neha T (18BCS060)
5. Rahul S (18BCS075)
6. Varun Awati (18BCS108)
7. Sushanth B Patil (18BCS102)
8. Trishul K S (18BCS104)

Setting up CI/CD Jenkins pipeline for Kubernetes

Tools and Technologies used :

- Github
- Docker and Docker hub
- Jenkins
- Kubernetes Cluster

Prerequisites:

- NodeJS v8+
- 2 AWS Ec2 Ubuntu instances of size t2.medium and 15 GB of volumes attached.
- Install Docker and kubernetes on AWS Instances.

Step - 1 : Setting Up kubernetes Cluster

We have used the kubeadm tool to set up the kubernetes cluster. Setting up a kubernetes cluster containing 2 nodes master and worker.

1. Install docker and add docker daemon after that enable and start the docker on both the nodes
2. Install Kubernetes (Kubeadm ,Kubelet and Kubectl) on both the nodes
3. Initialize the master node using kubeadm. Output of this command would be a key , through which worker nodes can join the Kubernetes cluster. Copy the token and save it somewhere.

```
ubuntu@ip-172-31-7-68: ~  
ubuntu@ip-172-31-7-68:~$ sudo swapoff -a  
swapoff: -a: swapoff failed: No such file or directory  
ubuntu@ip-172-31-7-68:~$ sudo hostnamectl set-hostname master-node  
ubuntu@ip-172-31-7-68:~$ sudo kubeadm init --pod-network-cidr=10.244.0.0/16  
[init] Using Kubernetes version: v1.22.3  
[preflight] Running pre-flight checks  
[preflight] Pulling images required for setting up a Kubernetes cluster  
[preflight] This might take a minute or two, depending on the speed of your internet connection  
[preflight] You can also perform this action in beforehand using 'kubeadm config images pull'  
[certs] Using certificateDir folder "/etc/kubernetes/pki"  
[certs] Generating "ca" certificate and key  
[certs] Generating "apiserver" certificate and key  
[certs] apiserver serving cert is signed for DNS names [kubernetes.kubernetes.default.kubernetes.default.svc.kubernetes.default.cluster.local master-node] and IP  
s [10.96.0.1 172.31.7.68]  
[certs] Generating "apiserver-kubelet-client" certificate and key  
[certs] Generating "front-proxy-ca" certificate and key  
[certs] Generating "front-proxy-client" certificate and key  
[certs] Generating "etcd/ca" certificate and key  
[certs] Generating "etcd/server" certificate and key  
[certs] etcd/server serving cert is signed for DNS names [localhost master-node] and IP  
s [172.31.7.68 127.0.0.1 ::1]  
[certs] Generating "etcd/peer" certificate and key  
[certs] etcd/peer serving cert is signed for DNS names [localhost master-node] and IP  
s [172.31.7.68 127.0.0.1 ::1]  
[certs] Generating "etcd/healthcheck-client" certificate and key  
[certs] Generating "apiserver-etcd-client" certificate and key  
[certs] Generating "sa" key and public key  
[kubeconfig] Using kubeconfig folder "/etc/kubernetes"  
[kubeconfig] Writing "admin.conf" kubeconfig file  
[kubeconfig] Writing "kubelet.conf" kubeconfig file  
[kubeconfig] Writing "controller-manager.conf" kubeconfig file  
[kubeconfig] Writing "scheduler.conf" kubeconfig file  
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kube  
let/kubeadm-flags.env"  
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yam  
l"  
[kubelet-start] Starting the kubelet  
[control-plane] Using manifest folder "/etc/kubernetes/manifests"  
[control-plane] Creating static Pod manifest for "kube-apiserver"  
[control-plane] Creating static Pod manifest for "kube-controller-manager"  
[control-plane] Creating static Pod manifest for "kube-scheduler"  
[etcd] Creating static Pod manifest for local etcd in "/etc/kubernetes/manifests"  
[wait-control-plane] Waiting for the kubelet to boot up the control plane as stati
```

```
ubuntu@ip-172-31-34-62: ~  
Your Kubernetes control-plane has initialized successfully!  
  
To start using your cluster, you need to run the following as a regular user:  
  
    mkdir -p $HOME/.kube  
    sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config  
    sudo chown $(id -u):$(id -g) $HOME/.kube/config  
  
You should now deploy a pod network to the cluster.  
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:  
    https://kubernetes.io/docs/concepts/cluster-administration/addons/  
  
Then you can join any number of worker nodes by running the following on each as r  
oot:  
  
kubeadm join 172.31.34.62:6443 --token bumzlh.ama2w89b0ep22rlz \\  
    --discovery-token-ca-cert-hash sha256:c78095c3b2344887752838c6d757f68ae88a40c4  
9ed7ff64c2f3475ac0fa60f8
```

Using the token copied earlier run this command on worker node :

```
kubeadm join 172.31.34.62:6443 --token bumzlh.ama2w89b0ep22rlz \ --  
discovery-token-ca-cert-hash  
sha256:c78095c3b2344887752838c6d757f68ae88a40c49ed7ff64c2f3475ac0fa  
60f8
```

Get the following output:

```
ubuntu@ip-172-31-40-68: ~
Setting up ubuntu-fan (0.12.13) ...
Created symlink /etc/systemd/system/multi-user.target.wants/ubuntu-fan.service → /lib/systemd/system/ubuntu-fan.service.
Processing triggers for systemd (245.4-4ubuntu3.13) ...
Processing triggers for man-db (2.9.1-1) ...
Processing triggers for dbus (1.12.16-2ubuntu2.1) ...
Processing triggers for libc-bin (2.31-0ubuntu9.2) ...
ubuntu@ip-172-31-40-68:~$ sudo apt-mark hold docker-ce kubelet kubeadm kubectl
docker-ce set on hold.
kubelet set on hold.
kubeadm set on hold.
kubectl set on hold.
ubuntu@ip-172-31-40-68:~$ echo "net.bridge.bridge-nf-call-iptables=1" | sudo tee -a /etc/sysctl.conf
net.bridge.bridge-nf-call-iptables=1
ubuntu@ip-172-31-40-68:~$ sudo sysctl -p
net.bridge.bridge-nf-call-iptables = 1
ubuntu@ip-172-31-40-68:~$ sudo kubeadm join 172.31.34.62:6443 --token bumzih.ama2w89b0ep22xiz \
--discovery-token-certificate sha256:c78095c3b2344987752938c6d757f68ae88a40c49ed7ff64c2f3475a00fa60f8
[preflight] Running pre-flight checks
[WARNING IsDockerSystemdCheck]: detected "cgroupsfs" as the Docker cgroup driver. The recommended driver is "systemd". Please follow the guide at https://kubernetes.io/docs/setup/cri/
[WARNING SystemVerification]: this Docker version is not on the list of validated versions: 20.10.7. Latest validated version: 18.09
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -oyaml'
[kubelet-start] Downloading configuration for the kubelet from the "kubelet-config-v1.15" ConfigMap in the kube-system namespace
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Activating the kubelet service
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap...

This node has joined the cluster:
 * Certificate signing request was sent to apisserver and a response was received.
 * The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the control-plane to see this node join the cluster.

ubuntu@ip-172-31-40-68:~$
```

When we run **kubectl get nodes** on master node Now there would be 2 nodes one master and one newly joined worker node newly joined node's role name would be <none> to label it as worker Using the token copied earlier

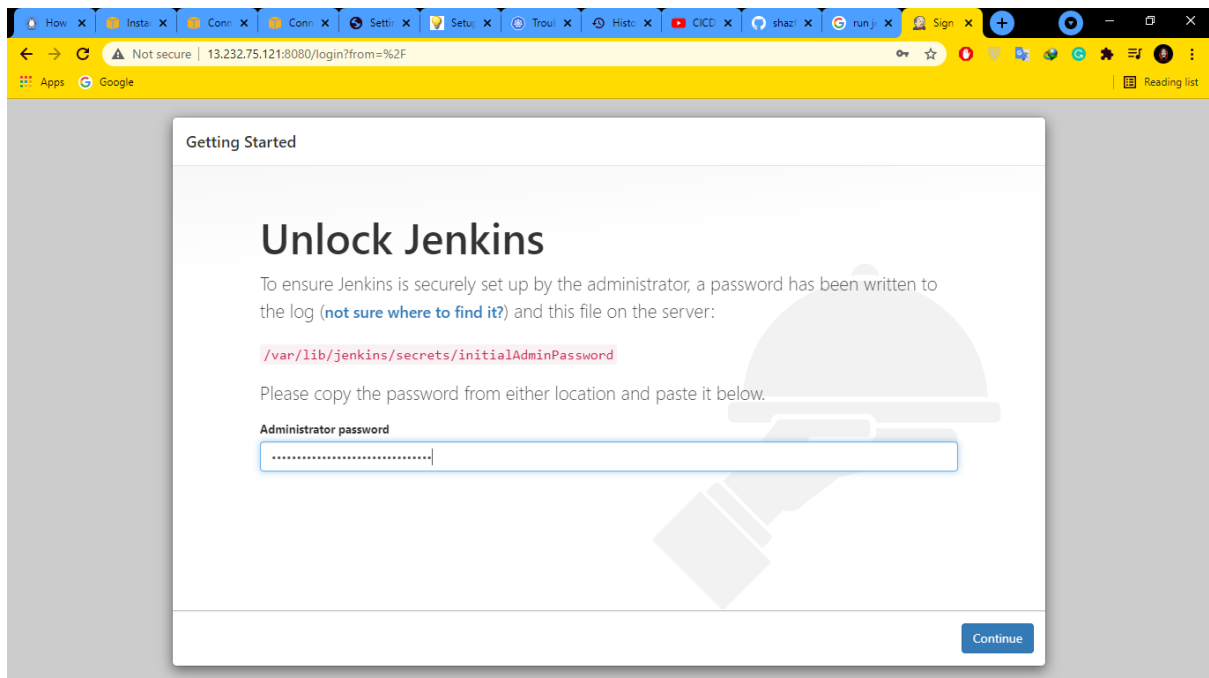
```
ubuntu@ip-172-31-34-62: ~
ubuntu@ip-172-31-34-62:~$ mkdir -p $HOME/.kube
ubuntu@ip-172-31-34-62:~$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
ubuntu@ip-172-31-34-62:~$ sudo chown $(id -u):$(id -g) $HOME/.kube/config
ubuntu@ip-172-31-34-62:~$ sudo kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml
unable to recognize "https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml": Get http://localhost:8080/api?timeout=32s: dial tcp 127.0.0.1:8080: connect: connection refused
unable to recognize "https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml": Get http://localhost:8080/api?timeout=32s: dial tcp 127.0.0.1:8080: connect: connection refused
unable to recognize "https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml": Get http://localhost:8080/api?timeout=32s: dial tcp 127.0.0.1:8080: connect: connection refused
unable to recognize "https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml": Get http://localhost:8080/api?timeout=32s: dial tcp 127.0.0.1:8080: connect: connection refused
unable to recognize "https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml": Get http://localhost:8080/api?timeout=32s: dial tcp 127.0.0.1:8080: connect: connection refused
unable to recognize "https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml": Get http://localhost:8080/api?timeout=32s: dial tcp 127.0.0.1:8080: connect: connection refused
ubuntu@ip-172-31-34-62:~$ kubectl get nodes
NAME          STATUS    ROLES    AGE   VERSION
master        NotReady  master   4m2s  v1.15.7
node1         NotReady  <none>    54s   v1.15.7
ubuntu@ip-172-31-34-62:~$
```

Step - 2 : Setting up jenkins On aws ec2 ubuntu

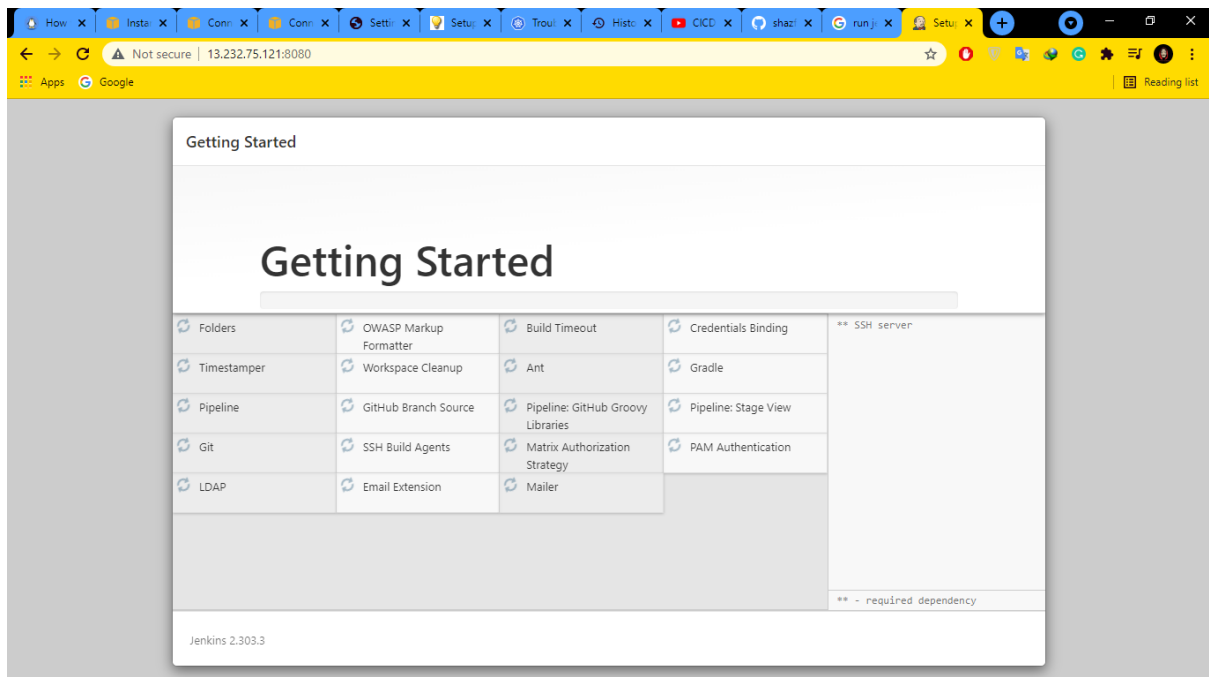
1. Install Java and Jenkins. Run the jenkins file, the first time it asks for an admin password. Run **cat /var/lib/jenkins/secrets/initialAdminPassword** and copy the password and paste it.

ubuntu@ip-172-31-34-62: ~

```
ubuntu@ip-172-31-34-62:~$ sudo cat /var/lib/jenkins/secrets/initialAdminPassword
5e641ceeba5d481a8e2b9c5d85b90a53
ubuntu@ip-172-31-34-62:~$
```

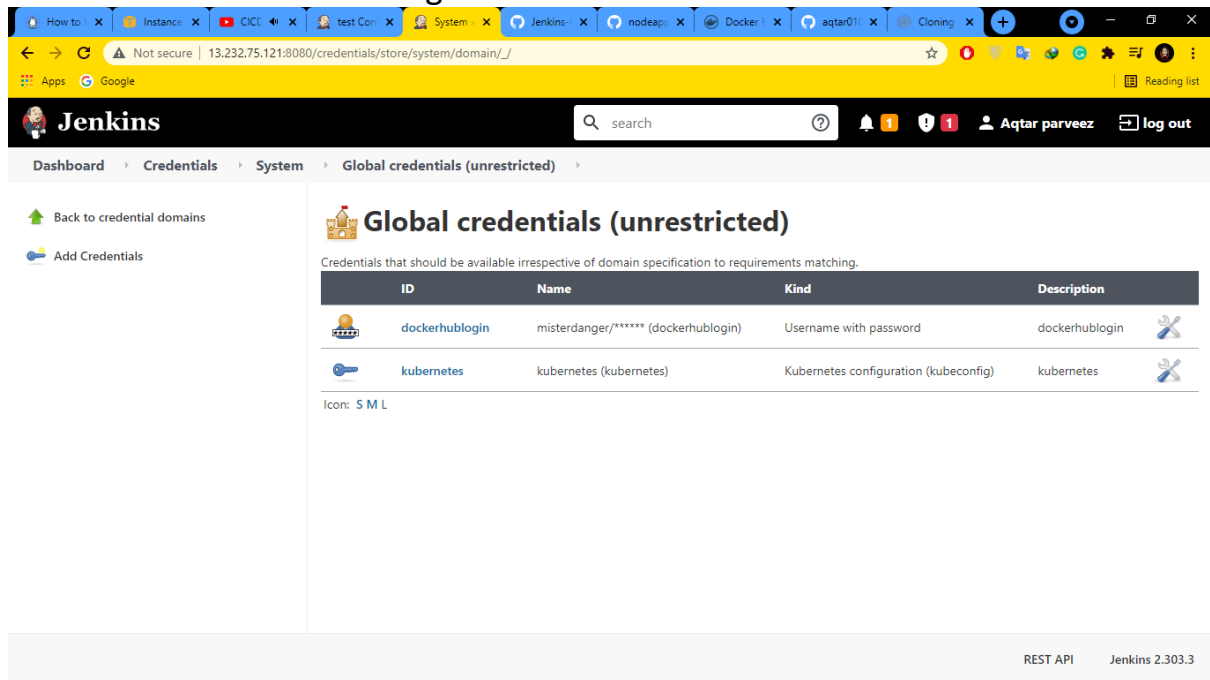


Enter the credentials and select the default plugin pack and wait for it to install. After installing start using Jenkins.



Go to Manage Jenkins -> Manage Plugins -> Available and install plugins for Nodejs, Docker, Kubernetes, Github, Kubernetes CLI.

Configure Docker Hub and Kubernetes Cluster credentials. Go to Manage Jenkins -> Manage credentials and add a credential for docker hub with your username and password. Create a credential id (which will be used later) and description. Similarly add Kubernetes cluster by adding the config file i.e. home->.kube->config and save it.



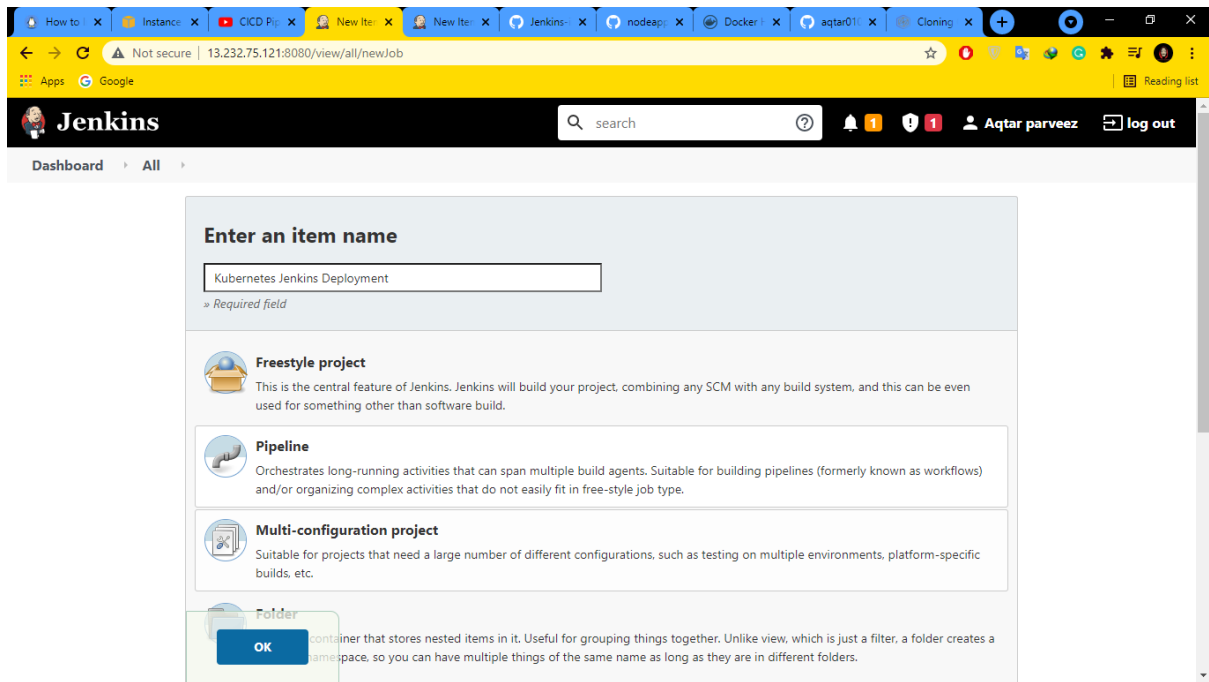
Install docker on Jenkins server and add current ubuntu user and Jenkins to docker group

Step - 3: Setting up code

We made a simple nodejs application, code is on github: [Github Link](#)

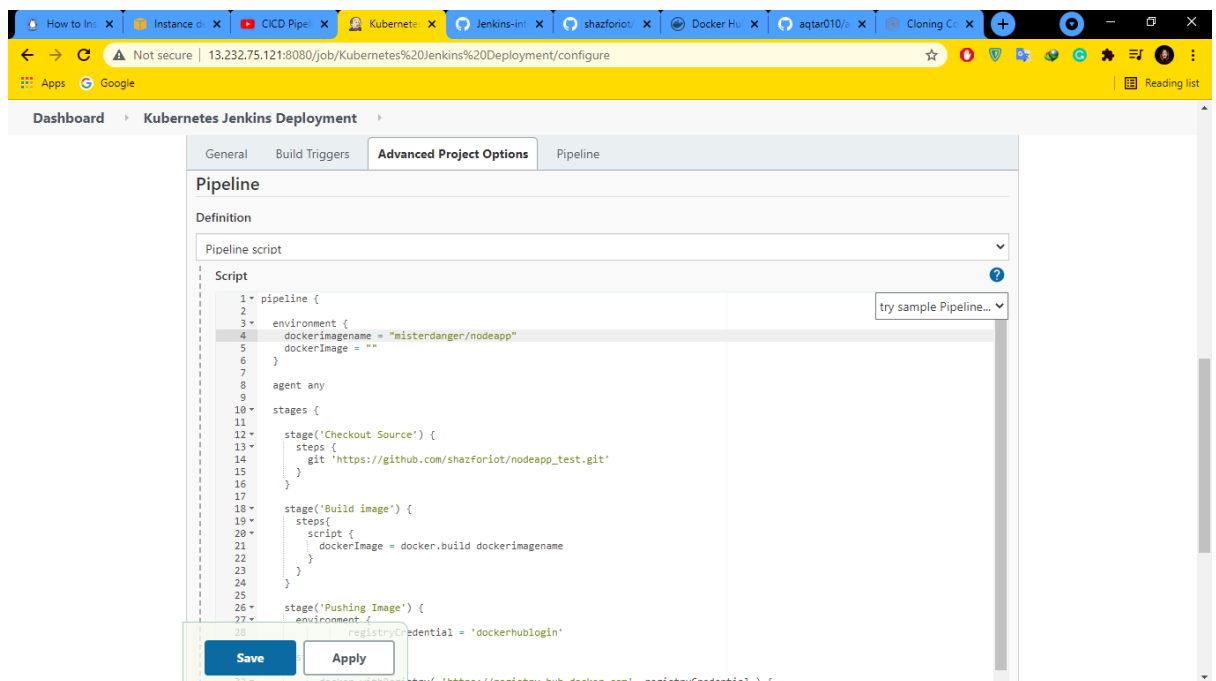
Step - 4: Building the pipeline

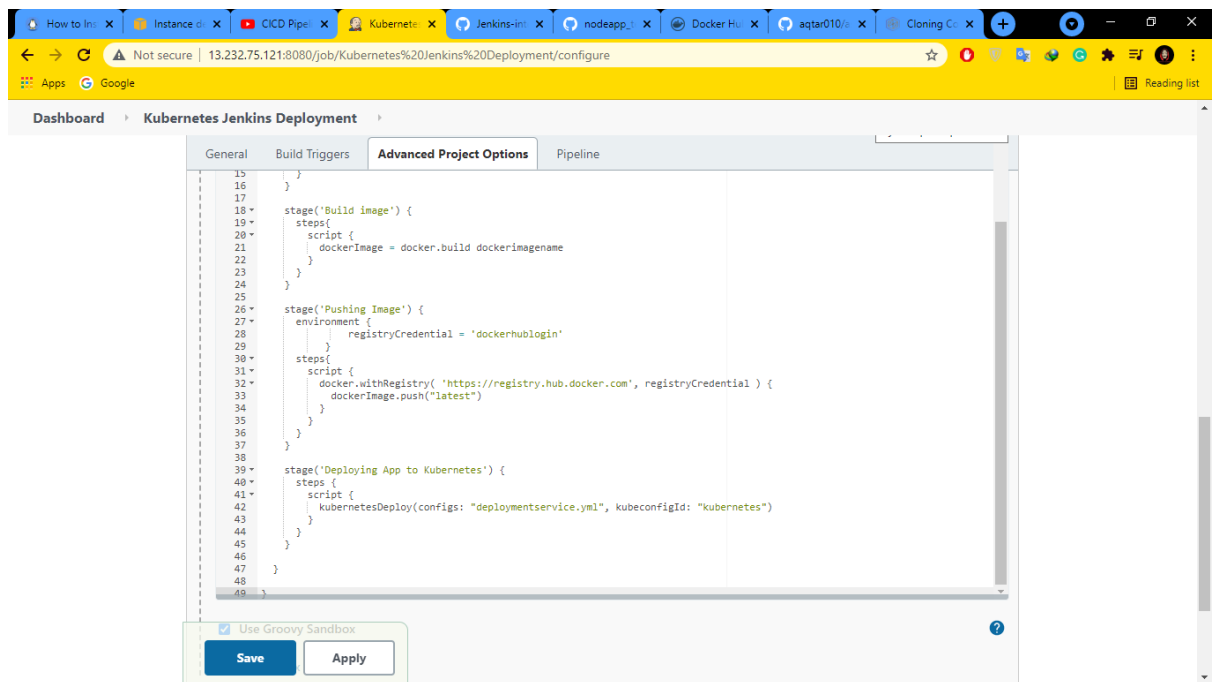
1. Create a jenkins pipeline.



Add pipeline code to create stages which include:

- A. Create Environment & Git clone
- B. Docker Build
- C. Docker login
- D. Push image to docker hub
- F. Copy deploymentservice.yml file to kubernetes master





The Pipeline Code:

```
pipeline {

    environment {

        dockerimagename = "misterdanger/nodeapp"
        dockerImage = ""
    }

    agent any

    stages {
        stage('Checkout Source') {
            steps {
                git 'https://github.com/aqtar010/ass1.git'
            }
        }

        stage('Build image') {
            steps{
                script {
                    dockerImage = docker.build dockerimagename
                }
            }
        }

        stage('Pushing Image') {
```

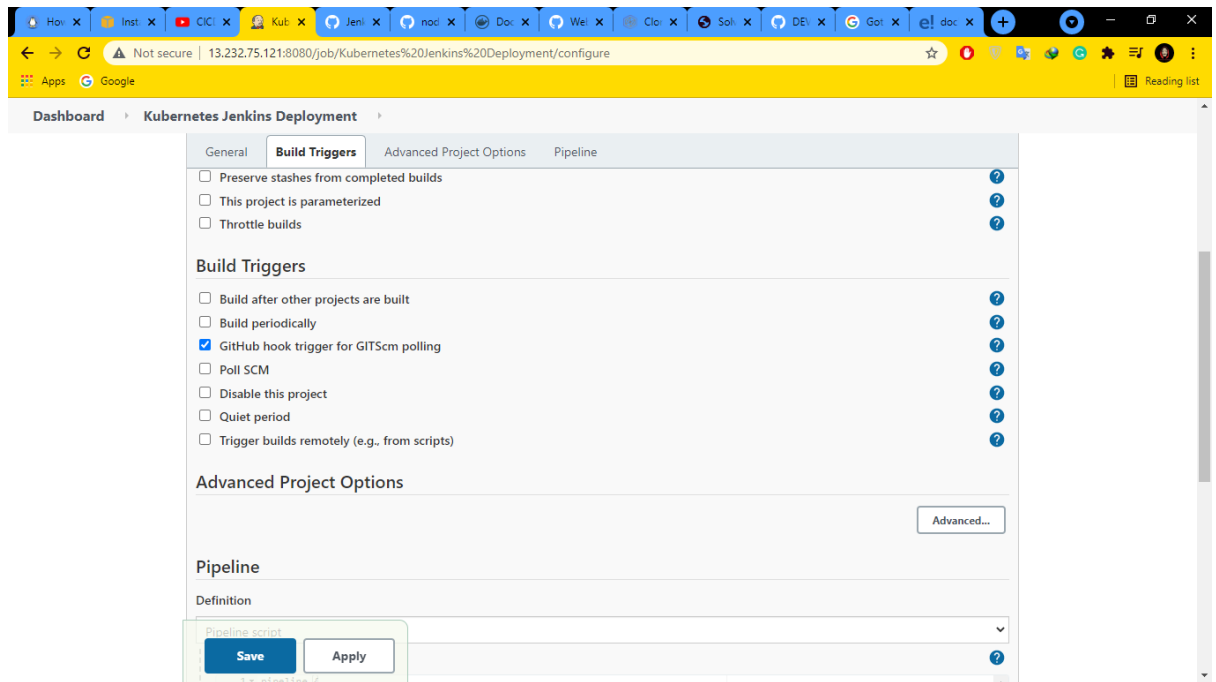


```

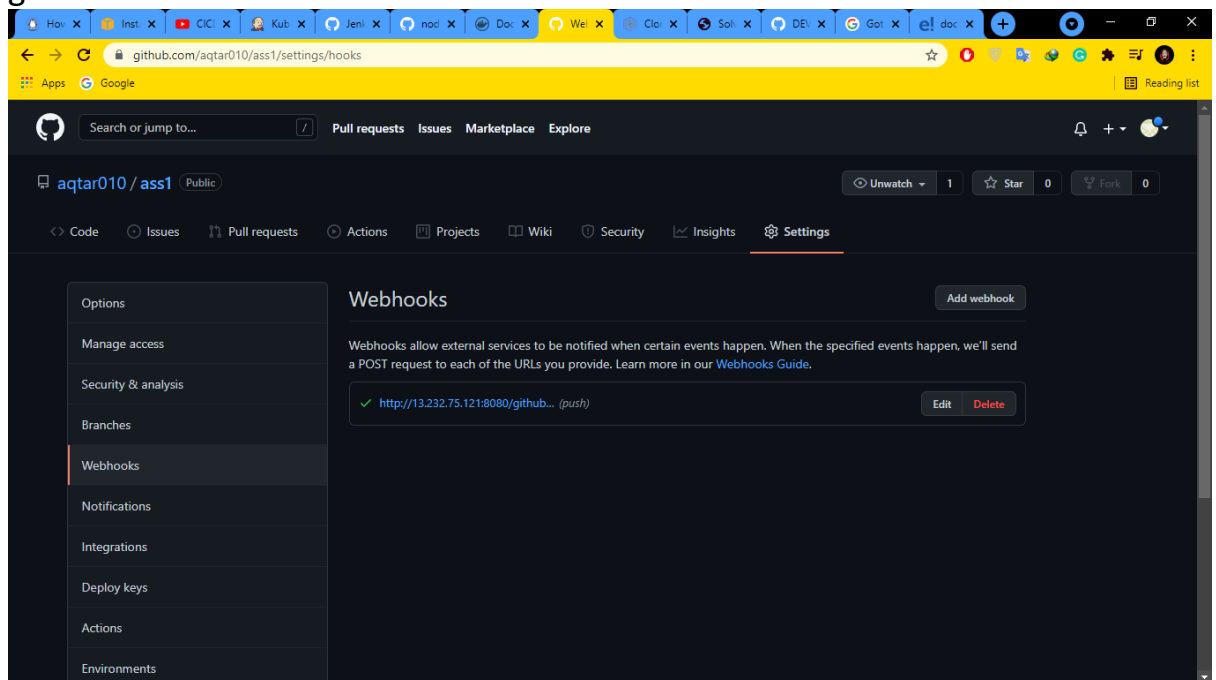
environment {
    registryCredential = 'dockerhublogin'
}
steps{
    script {
        docker.withRegistry( 'https://registry.hub.docker.com',
registryCredential ) {
            dockerImage.push("latest")
        }
    }
}
stage('Deploying App to Kubernetes') {
    steps {
        script {
            kubernetesDeploy(configs: "deployment-service.yml", kubeconfigId:
"kubernetes")
        }
    }
}
}
}

```

Select the github hook trigger option and save.



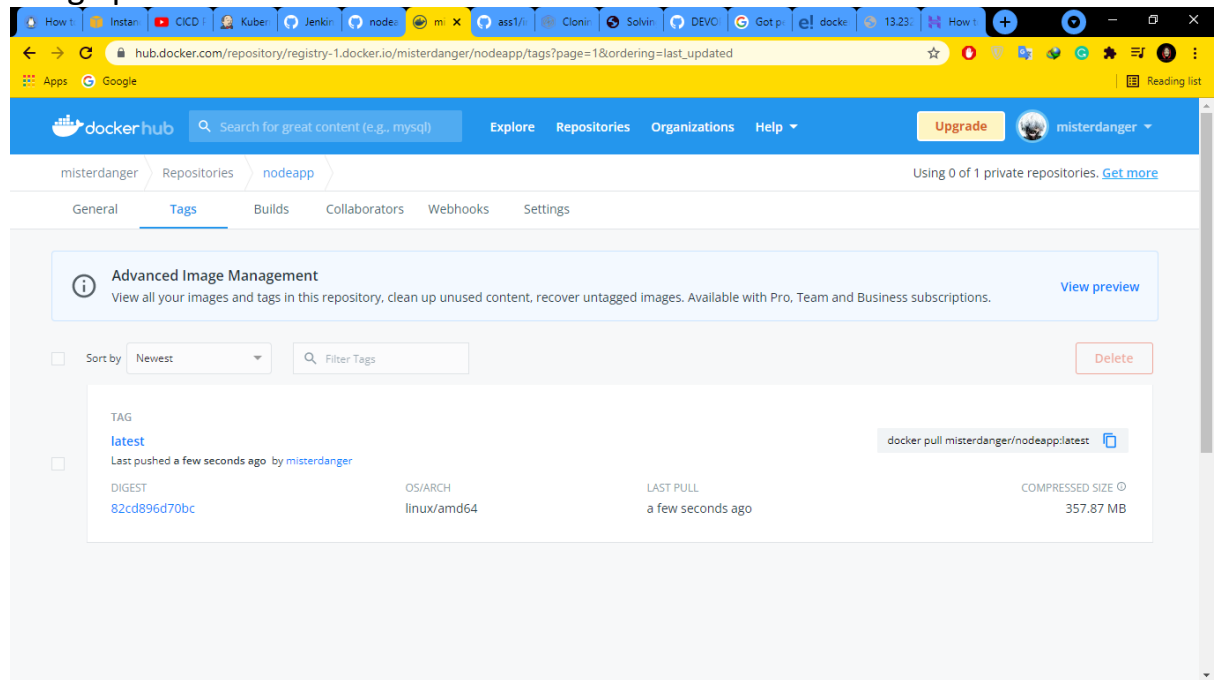
Go to github->your repo and add a webhook and enter the jenkins ip and port no. and add the webhook . It will ping your server and show a green tick on confirmation.



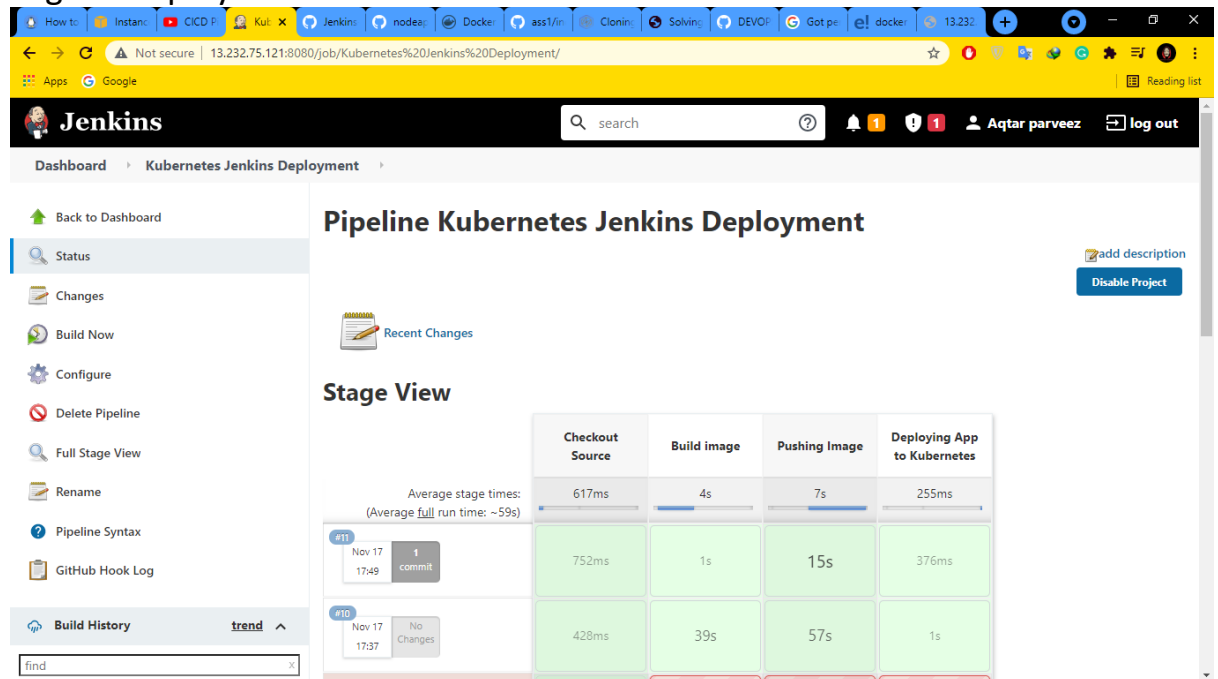
Build The Job on Jenkins

Output:

Image pushed on to docker Hub:



Logs of Deployment on Jenkins:



Checking the deployments on kubernetes server

1. To get list of deployments on server run **kubectl get deployments**
2. To get list of services on kubernetes server run **kubectl get svc**

```
[root@k8smaster ~/.kube]# kubectl get deployments
No resources found in default namespace.
[root@k8smaster ~/.kube]# kubectl get pods
No resources found in default namespace.
[root@k8smaster ~/.kube]# kubectl get svc
NAME                TYPE                CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
kubernetes           ClusterIP           10.96.0.1        <none>            443/TCP          129m
[root@k8smaster ~/.kube]# kubectl get deployment
NAME                READY    UP-TO-DATE    AVAILABLE    AGE
nodeapp-deployment  1/1      1              1            44s
[root@k8smaster ~/.kube]# kubectl get pods
NAME                READY    STATUS      RESTARTS    AGE
nodeapp-deployment-75f9979565-q7nr8  1/1      Running      0            53s
[root@k8smaster ~/.kube]# kubectl get svc
NAME                TYPE                CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
kubernetes           ClusterIP           10.96.0.1        <none>            443/TCP          132m
nodeapp-service      LoadBalancer        10.102.240.163   <pending>         5000:31110/TCP    62s
[root@k8smaster ~/.kube]# kubectl get rs
NAME                DESIRED    CURRENT    READY    AGE
nodeapp-deployment-75f9979565  1          1          1        2m6s
[root@k8smaster ~/.kube]#
```

node-js service is running on Nodeport, so Deployment is Successful

From the terminal output, node-js-service is running on port number 31110, so

open tcp custom 31110 on kubernetes service instance

