Reboot the Web Landscape with DevOps and Cloud

Workshop Project

edureka!

DEVOPS WORKSHOP: REBOOT THE WEB LANDSCAPE WITH DEVOPS AND CLOUD

Problem Statement

Aim: To build an automated pipeline using the top DevOps tools to deploy the code into production website, based on the requirement.

Objective: To build an automated pipeline, by first setting up the Jenkins server and then containerize the application. Then, we will setup the Kubernetes cluster and continuously deploy the Docker Image. Moving further we will check the health of the pods.

Tools Covered in the Project:

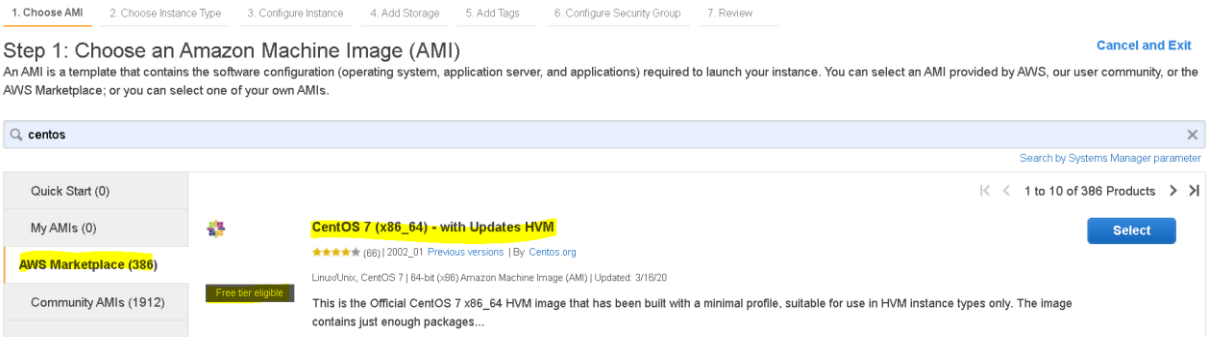
- Git
- Jenkins
- Docker
- Kubernetes

Steps Involved:

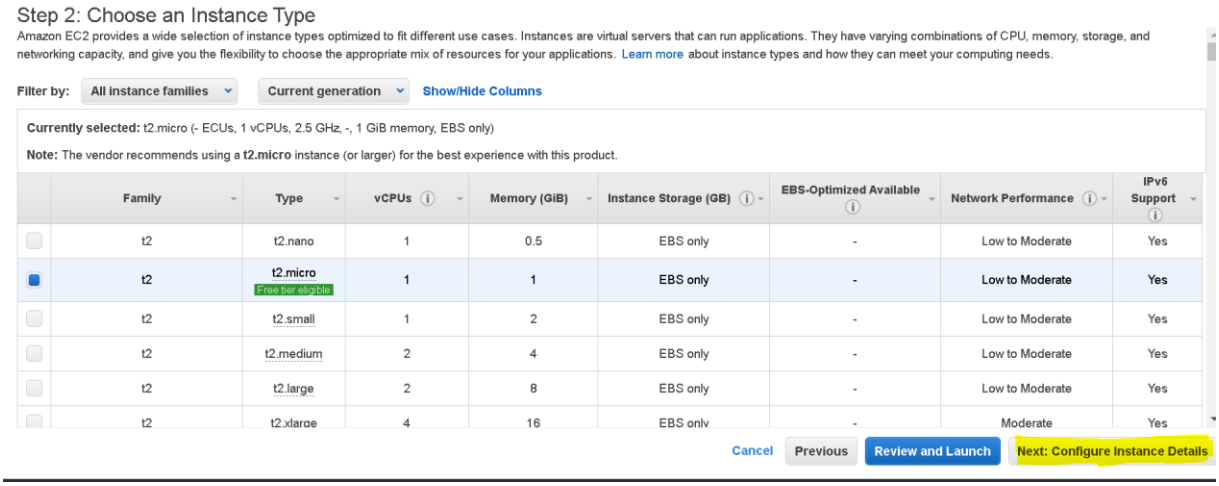
- The team of developers working on new features will merge their code to a GitHub repo.
- As soon as the code reaches GitHub, using a CI (Continuous Integration) pipeline, setup in Jenkins, automated builds will be triggered.
- The automated builds will frequently deploy new features to the production website.
- Every build will prepare a Docker file and push docker images to docker-hub.
- Every docker image will be deployed (Continuous Deployment) to a Kubernetes-cluster.
- To check health of the pods, we may inject Liveness probes in the pod specification.

Jenkins-Server Installation:

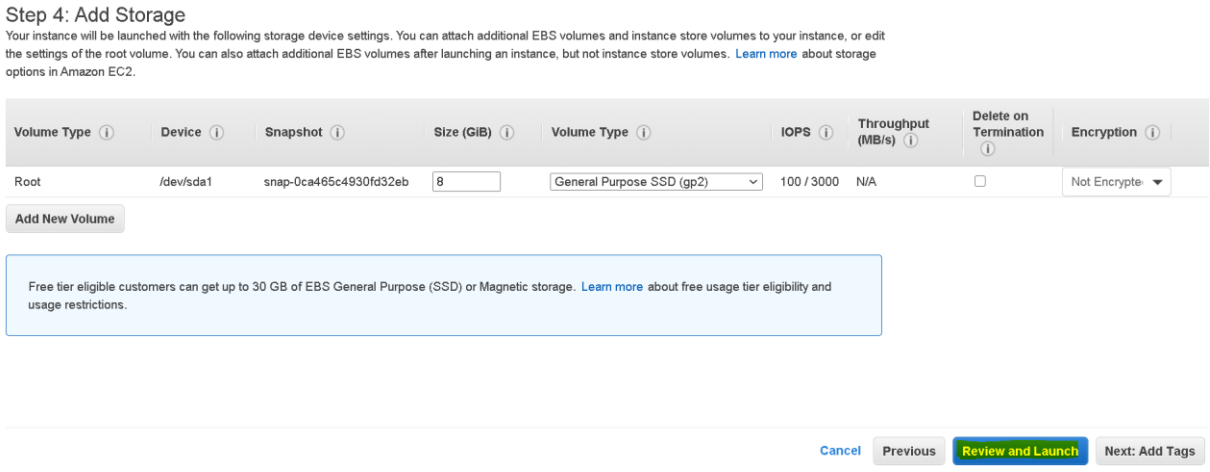
1. On the AWS Management Console, click launch instance, and choose Centos 7 AMI:



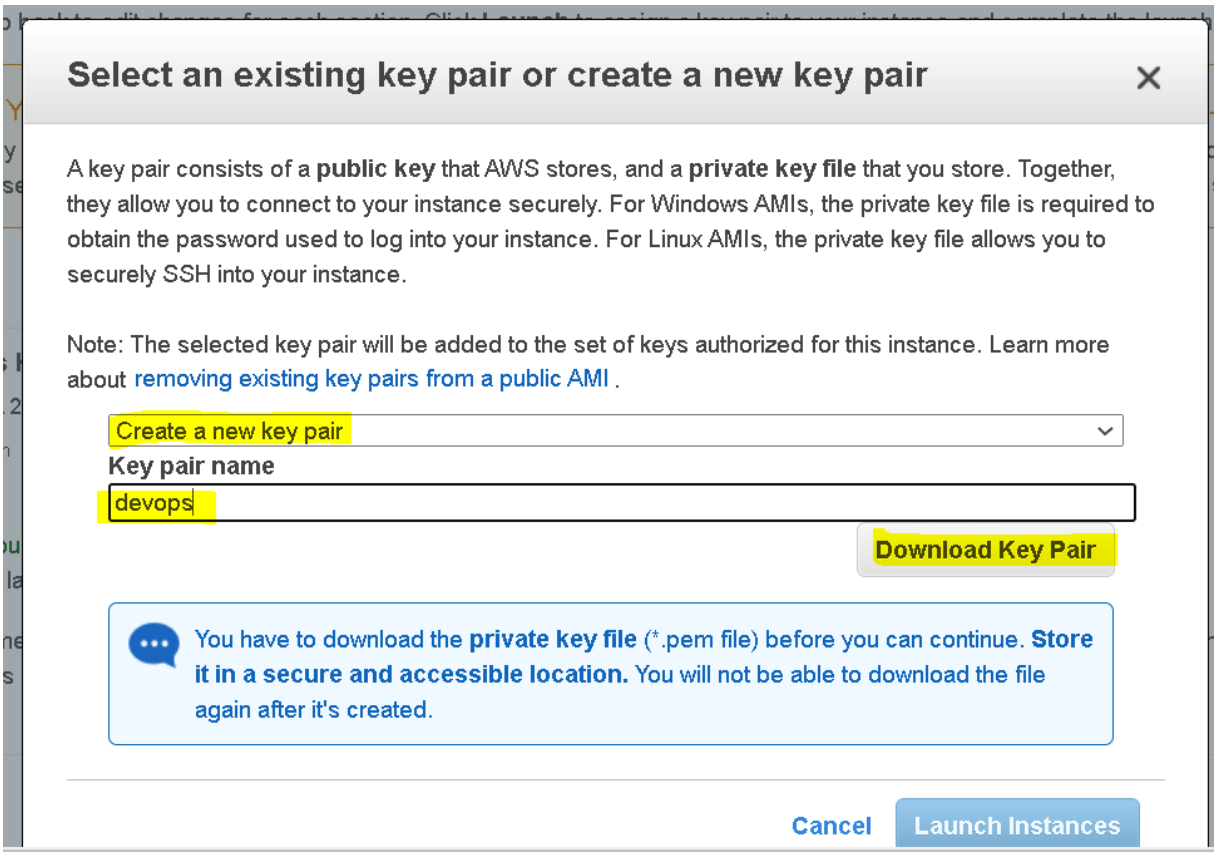
2. Keep clicking "Next: Configure Instance Details":



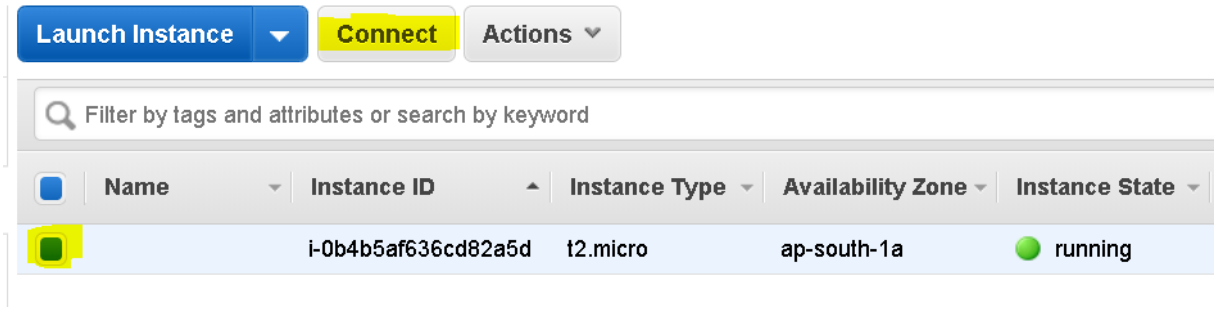
3. Then click "Review and Launch" and then finally click "Launch":



4. Create a New key pair and save the public key in your local system:



5. Then choose the instance and click on connect to SSH into the Jenkins-server:



6. After you have logged in to the server, run the following commands in sequence.

```
sudo yum update –y

sudo yum -y remove java

sudo yum -y install java-1.8.0-openjdk

sudo yum –y install wget

sudo wget -O /etc/yum.repos.d/jenkins.repo http://pkg.jenkins-ci.org/redhat-stable/jenkins.repo

sudo rpm --import https://jenkins-ci.org/redhat/jenkins-ci.org.key
```

```
sudo yum -y install jenkins-2.190.1-1.1.noarch
sudo systemctl enable jenkins
sudo systemctl start Jenkins
sudo yum -y install docker
sudo systemctl start docker
sudo systemctl enable docker
sudo groupadd docker
sudo usermod -aG docker jenkins
sudo systemctl restart jenkins
sudo systemctl restart docker
```

7. This finishes the Jenkins-server installation

Creating a Kubernetes Cluster:-

1. Please follow the same above steps to setup a **Kubernetes Master** and **Worker** nodes. Choose the AMI as per the given details below:

Kubernetes Master AMI: Ubuntu 16.04 LTS; **Instance Type:** t3.micro $0.0112 (83 paise) per hour

Worker AMI: Ubuntu 16.04 LTS; **Instance Type:** t2.micro

2. After you have the Kubernetes Master and Worker nodes up and running, please SSH into both nodes simultaneously and run below commands on both master and worker nodes in sequence:-

```
sudo apt-get update
# (Install Docker CE)
## Set up the repository:
### Install packages to allow apt to use a repository over HTTPS
sudo apt-get update && sudo apt-get install -y \ apt-transport-https
ca-certificates curl software-properties-common gnupg2
# Add Docker's official GPG key:
```

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key --keyring /etc/apt/trusted.gpg.d/docker.gpg add -

```
# Add the Docker apt repository:
sudo add-apt-repository \
  "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) \
  stable"
# Install Docker CE
sudo apt-get update && sudo apt-get install -y \
  containerd.io=1.2.13-2 \
  docker-ce=5:19.03.11~3-0~ubuntu-$(lsb_release -cs) \
  docker-ce-cli=5:19.03.11~3-0~ubuntu-$(lsb_release -cs)
# Set up the Docker daemon
cat <<EOF | sudo tee /etc/docker/daemon.json
{
  "exec-opts": ["native.cgroupdriver=systemd"],
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "100m"
  },
  "storage-driver": "overlay2"
}
EOF
# Create /etc/systemd/system/docker.service.d
sudo mkdir -p /etc/systemd/system/docker.service.d
# Restart Docker
sudo systemctl daemon-reload
sudo systemctl restart docker
# If you want the docker service to start on boot, run the following
command:
sudo systemctl enable docker
```

```
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 -

cat <<EOF | sudo tee /etc/apt/sources.list.d/kubernetes.list

deb https://apt.kubernetes.io/ kubernetes-xenial main

EOF

sudo apt-get update

sudo apt-get install -y kubelet kubeadm kubectl

sudo apt-mark hold kubelet kubeadm kubectl

kubeadm version -o short

systemctl daemon-reload

systemctl restart kubelet
```

#Run the below command only on Master:

```
#Bootstrap a Kubernetes cluster

sudo kubeadm init

# 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


# copy ‘kubeadm join’ command to run it on worker nodes.

#run the below command on worker nodes – Note: The command will differ in your case as the token will be different

sudo kubeadm join 172.31.36.201:6443 --token eko5kq.fgbazut2dchk17n8 --discovery-token-ca-cert-hash sha256:585644c1825a657b962035aded8e886164d94666ce07b06998236231b765674b

# to regenerate token on master :

kubeadm token create --print-join-command

# check cluster nodes on the master
```

```
kubectl get nodes

# install cluster networking plugin

sudo sysctl net.bridge.bridge-nf-call-iptables=1   #run this command on both master and worker nodes

# deploy weave plugin on the master

kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version | base64 | tr -d '\n')"
```

Now, follow the below steps in sequence to setup the DevOps project: -

1. Login to Jenkins UI: http://jenkins-server-public-ip:8080
   Install all the suggested plugins + continuous deploy plugin (for Kubernetes deployment)

We need to show the password for the admin user to log in to our Jenkins web interface:

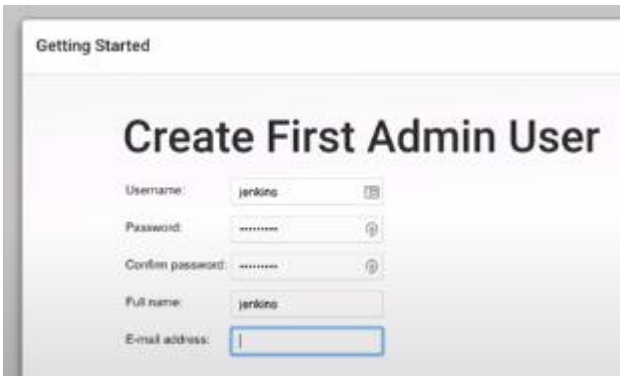

```
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

Copy the string that is output and paste it into the Administrator password field in your browser. Click **Continue**.

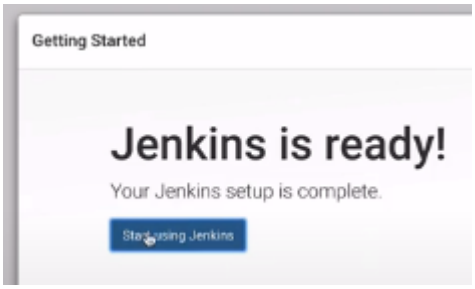
For the Create First Admin User form, provide the following information:

- Username: jenkins
- Password: jenkins
- Confirm password: jenkins
- Full name: jenkins

Email address: jenkins@jenkins.com

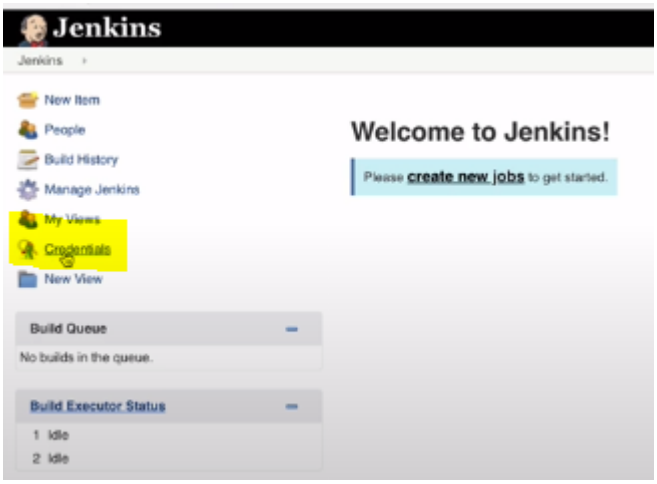


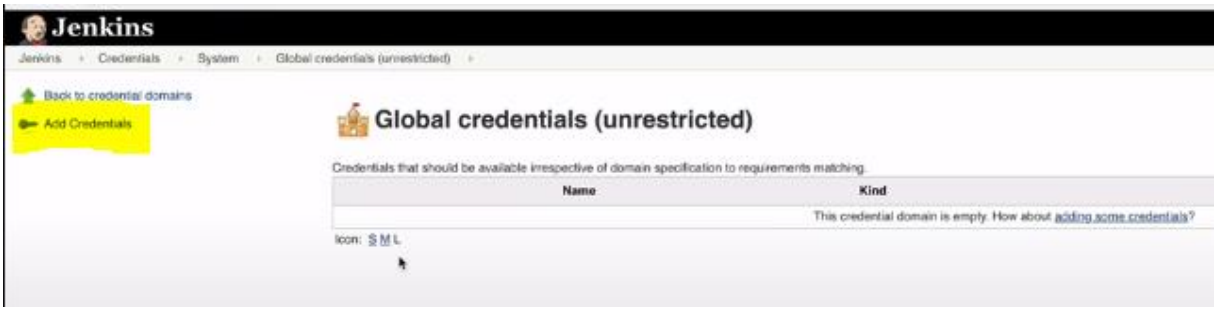
Click **Save and continue**. Next, click **Start using Jenkins**.

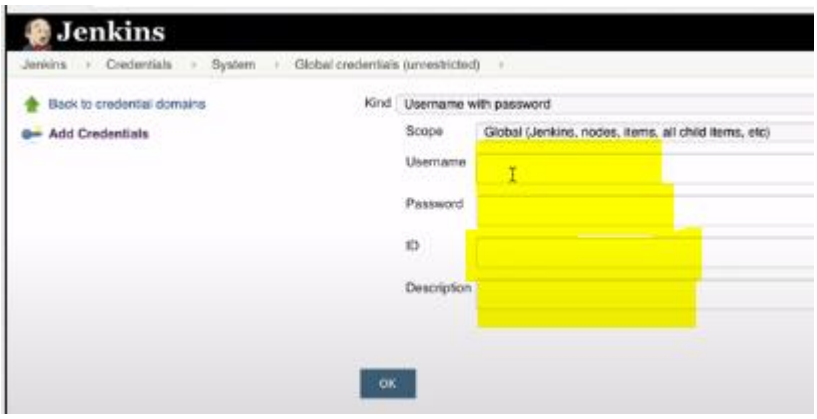


Prepare the Jenkins Environment and Verify Your Configuration with an Initial Deploy

Add GitHub Credentials in Jenkins

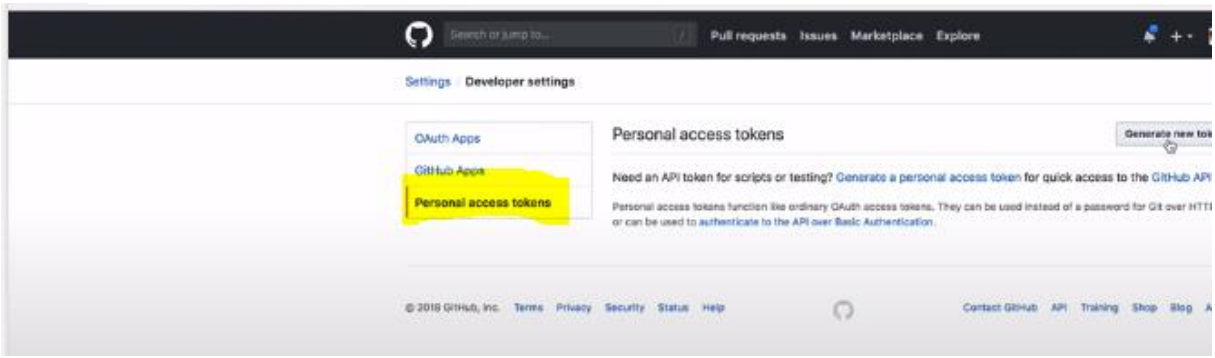


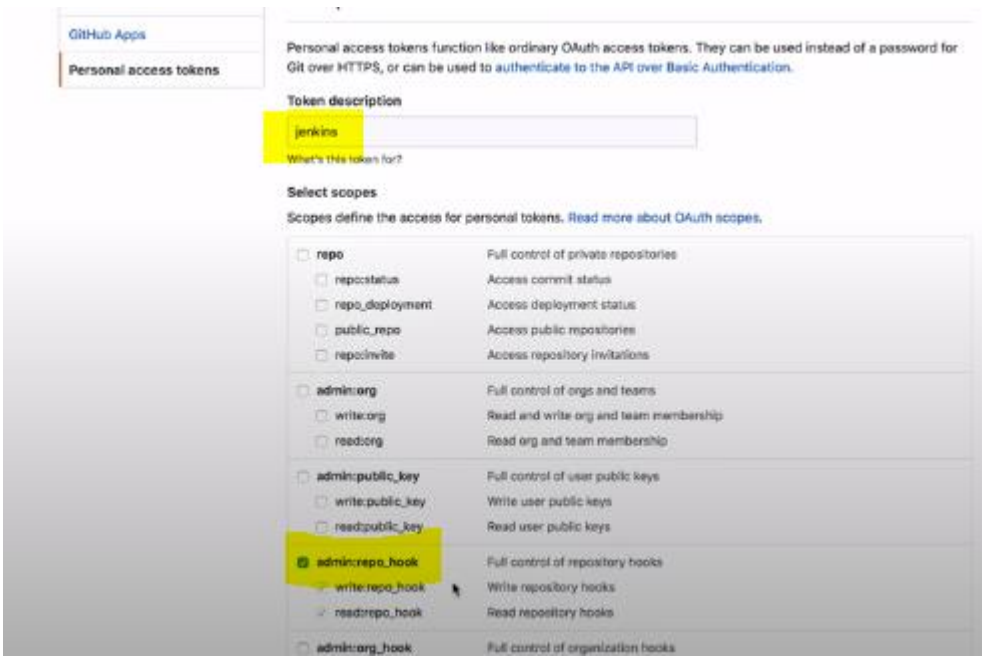


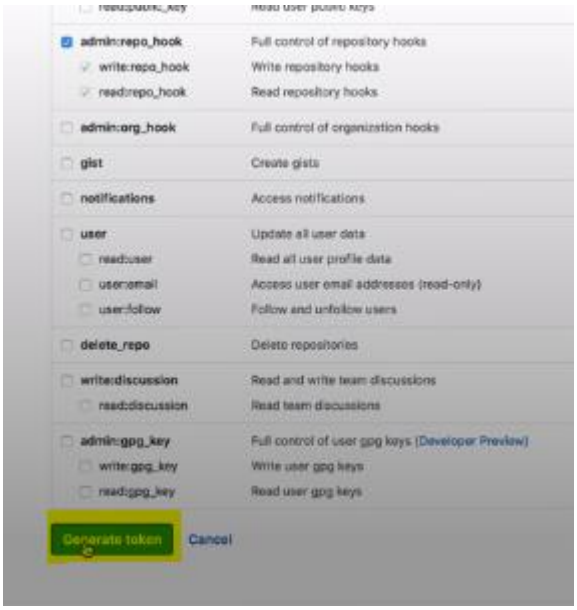


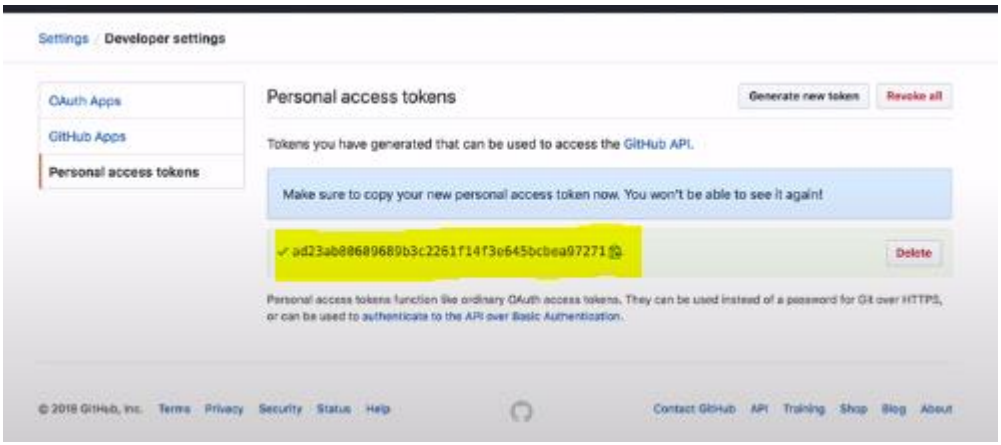
We will use a GitHub API token for the next step. Navigate to the GitHub tab in your browser. Click your profile picture in the top right of the page, click **Settings**, click **Developer settings**, click **Personal access tokens**, and finally click **Generate new token**.

Name this token "jenkins" and be sure to click the checkbox next to **admin:repo_hook**. Click **Generate token** at the bottom of the page. Copy the token to your clipboard.







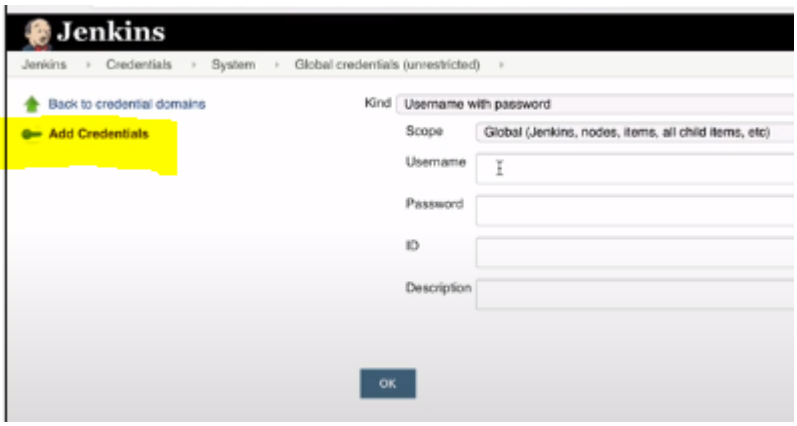


Back in the Jenkins tab in your browser, click **Credentials** in the menu on the left of the page and then click **global**. Click **Add Credentials** in the menu on the left of the page. Provide the following information:

- *Username*: Provide your GitHub username
- *Password*: Paste the API token from your clipboard.
- *ID*: github_key
- *Description*: GitHub Key

Click **OK**.

Click **Add Credentials** in the menu on the left of the page.



Add Docker Hub Credentials in Jenkins

Note: You will need a DockerHub account for this step.

- *Username*: Provide your DockerHub username
- *Password*: Provide your DuckerHub password
- *ID*: docker_hub_login
- *Description*: Docker Hub Login

Click **OK**.

Add the Kubeconfig from the Kubernetes master as a credential in Jenkins

We will need to view the contents of our Kubeconfig for this step. Log in to the Kubernetes master node by navigating to the hands-on lab page, copy the *Kubernetes Master Public IP*, and use the credentials for that instance to log in via SSH:

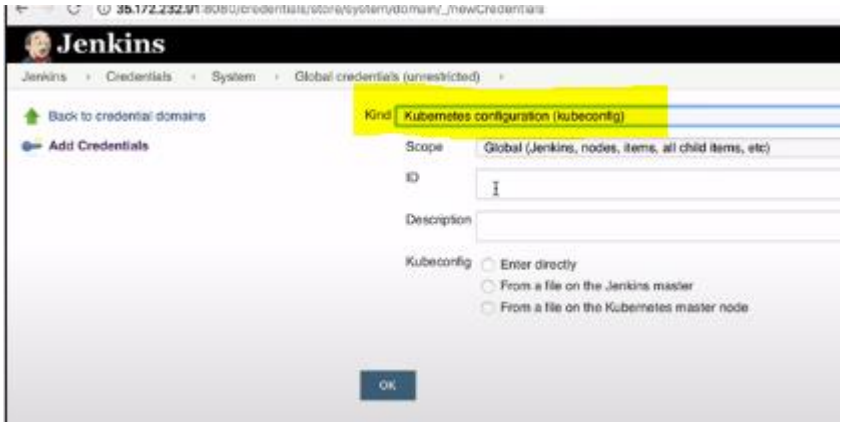
```
ssh ubuntu@<KUBERNETES_MASTER_PUBLIC_IP>;
```

Next, display the contents of our Kubeconfig:

```
cat ~/.kube/config
```

Copy the output of this file to your clipboard. We will need to paste this into Jenkins, so navigate back to the Jenkins tab in your browser.

Click **Add Credentials** in the menu on the left of the page.



Add credentials with the following information:

- *Kind*: Kubernetes configuration (kubeconfig)
- *ID*: kubeconfig
- *Description*: Kubeconfig
- *Kubeconfig*: **Enter directly**
    - *Content*: Paste the contents of ~/.kube/config

Click **OK**.

Configure Environment Variables

On the main page of Jenkins, click **Manage Jenkins**. Click **Configure System**.

In the *Global Properties* section, click the checkbox next to **Environment variables**. Click **Add**.

- **Name**: KUBE_MASTER_IP
- **Value**:

Click **Apply**.

In the *GitHub* section, click **Add GitHub Server** and then click **GitHub Server**.

- **Name**: GitHub
- **Credentials**: Click **Add** and then click **Jenkins**
    - **Kind**: Secret text
    - **Secret**: Paste the GitHub API token from the earlier step
    - **ID**: github_secret
    - **Description**: GitHub Secret

Click **Add**. Click the dropdown next to *Credentials* and select the **GitHub Secret** we just added. Click **Save**.

Fork the GitHub Repository

Open the following link in a new tab in your browser:

https://github.com/bhavukm/cicd-pipeline-train-schedule-autodeploy

Click Fork in the top-right of the page.

Click Jenkinsfile to open the file, then click the Edit icon in the top-right of the window.

- Change the DOCKER_IMAGE_NAME at the top of the Jenkinsfile to use your Docker Hub username instead of bhavukm.
- Click Commit Changes.

Set Up Project

Back in the Jenkins tab in our browser, click **New Item**. Use a *Name* of "train-schedule" and select **Multibranch Pipeline** as the type. Click **OK**.

In the *Branch Sources* section, click **Add source**, and then click **GitHub**.

- **Credentials**: Select the *GitHub Key*
- **Owner**: Enter your GitHub username
- **Repository**: Select **cicd-pipeline-train-schedule-autodeploy**
- In the *Behaviors* section, delete both *Discover pull requests* options by clicking the red **X** in the top right of each of their respective sections.

Click **Save**.

Click **train-schedule** in the top-left of the page and then click on **master**.

The initial build will take some time. Wait a few moments until your build gets to the *DeployToProduction* stage. When it is ready, hover your mouse over the blue box and click **Proceed**.

On the hands-on lab page, copy the **Kubernetes Master Public IP** and navigate to it in a new tab in your browser, using port 8080.

<KUBERNETES_MASTER_PUBLIC_IP>;:8080

The train-schedule app will load.

Add a Smoke Test with Automated Deployment and Remove the Human Approval Step from the Pipeline, Then Deploy

In the GitHub tab in your browser, click on the **Jenkinsfile** to open it. Click the **Edit** icon in the top-right of the page to edit this file.

Remove the human input step from the deployment and add a smoke test before the production deployment. Your **Jenkinsfile** should look like this:

```
pipeline {
    agent any
    environment {
        //be sure to replace "bhavukm" with your own Docker Hub
username
        DOCKER_IMAGE_NAME = "bhavukm/train-schedule"
        CANARY_REPLICAS = 0
    }
    stages {
        stage('Build') {
            steps {
                echo 'Running build automation'
                sh './gradlew build --no-daemon'
                archiveArtifacts artifacts:
'dist/trainSchedule.zip'
            }
        }
        stage('Build Docker Image') {
            when {
                branch 'master'
            }
            steps {
                script {
                    app = docker.build(DOCKER_IMAGE_NAME)
                    app.inside {
                        sh 'echo Hello, World!'
                    }
```

```
                }
            }
    }
        stage('Push Docker Image') {
            when {
                branch 'master'
            }
            steps {
                script {

docker.withRegistry('https://registry.hub.docker.com',
'docker_hub_login') {
                        app.push("${env.BUILD_NUMBER}")
                        app.push("latest")
                    }
                }
            }
        }
        stage('CanaryDeploy') {
            when {
                branch 'master'
            }
            environment {
                CANARY_REPLICAS = 1
            }
            steps {
                kubernetesDeploy(
                    kubeconfigId: 'kubeconfig',
                    configs: 'train-schedule-kube-canary.yml',
                    enableConfigSubstitution: true
```

```
                )
            }
        }
        stage('SmokeTest') {
            when {
                branch 'master'
            }
            steps {
                script {
                    sleep (time: 5)
                    def response = httpRequest (
                        url: "http://$KUBE_MASTER_IP:8081/",
                        timeout: 30
                    )
                    if (response.status != 200) {
                        error("Smoke test against canary
deployment failed.")
                    }
                }
            }
        }
        stage('DeployToProduction') {
            when {
                branch 'master'
            }
            steps {
                milestone(1)
                kubernetesDeploy(
                    kubeconfigId: 'kubeconfig',
                    configs: 'train-schedule-kube.yml',
```

```
                    enableConfigSubstitution: true
                )
            }
        }
    }
    post {
        cleanup {
            kubernetesDeploy (
                kubeconfigId: 'kubeconfig',
                configs: 'train-schedule-kube-canary.yml',
                enableConfigSubstitution: true
            )
        }
    }
}
```

Click **Commit Changes** to save your changes to the Jenkins file. The deployment will start automatically and can be viewed in the Jenkins tab of your browser.

Demonstrate the Pipeline in Action

In the GitHub tab in your browser, navigate to the main page of your fork by clicking on **cicd-pipeline-train-schedule-autodeploy** at the top of the page.

Click on **branches** to display the three branches of this repository. Click on **New pull request** for the *new-code* branch.

Change the following fields on this page:

- **base fork**: Set this to your personal fork of the **cicd-pipeline-train-schedule-autodeploy** repo
- **base**: master

The page will update and show the changes from the *new-code* branch to the *master* branch.

Click **Create pull request**. When the page updates, click **Merge pull request**. Finally, click **Confirm merge**.

Back in the Jenkins tab in your browser, a new build should spin up shortly.

Navigate to the tab in your browser that displays the **train-schedule** application. Refresh this page to see the changes that were made.

edureka!