Certification Project (DevOps) Mar-2023

Problem Statement

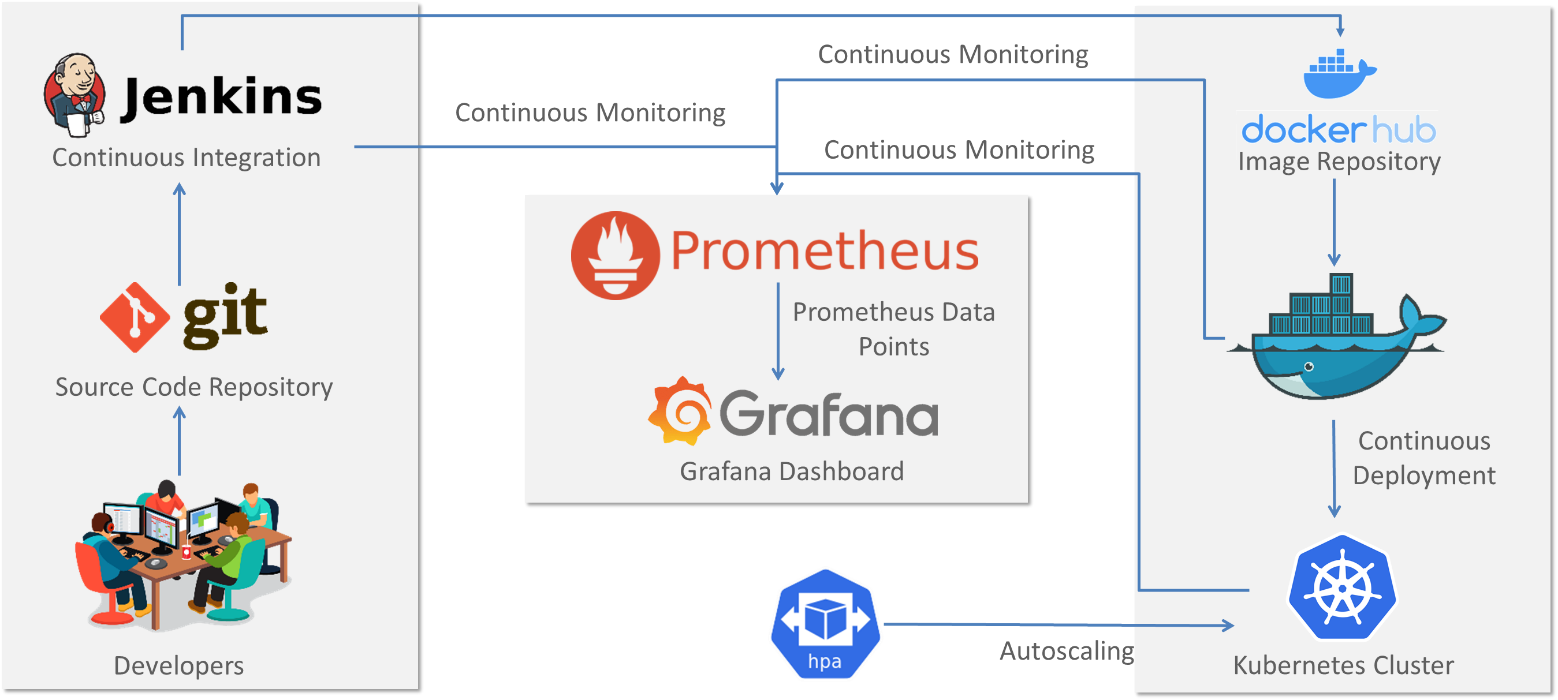
## **Problem Statement**

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 Docker file 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

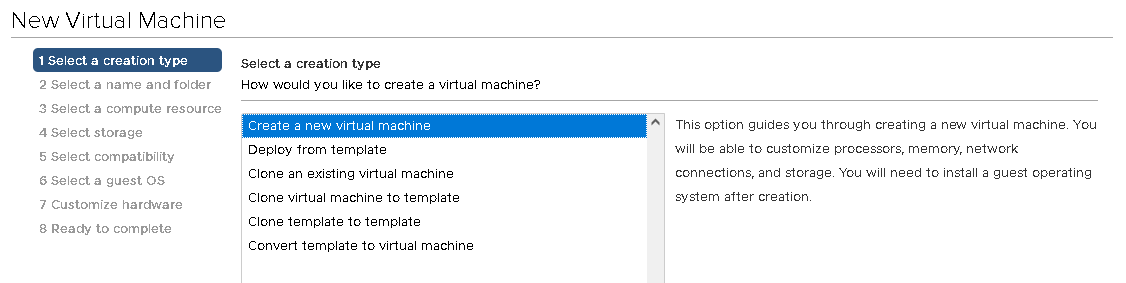
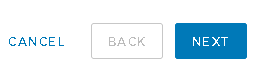
**GitHub:** <https://github.com/bhavukm/cicd-pipeline-train-schedule-autodeploy>

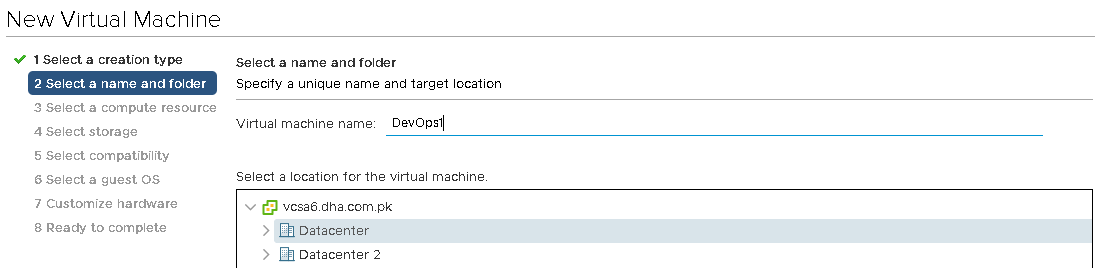
**Solution:**

**Jenkins-Server Installation: (Master and Slave VM Installation on VMWarre)**

Jenkins server can be installed on local system vm or any cloud or on-prem data Centre. In my case I used my VMWare infrastructure for Jenkins Server installation. Followings are the steps creating VM in VMWare Host.

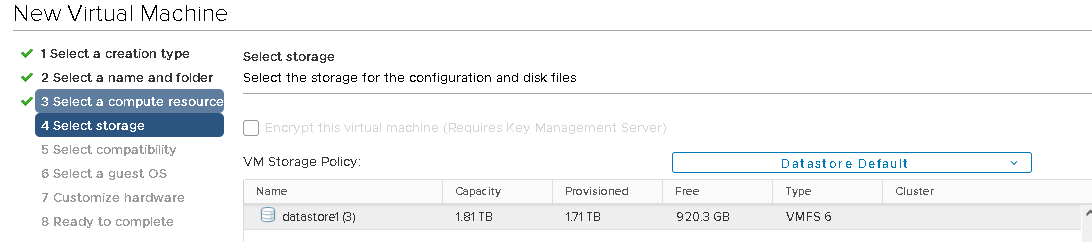
1. Right click host. Select create new machine. Click next

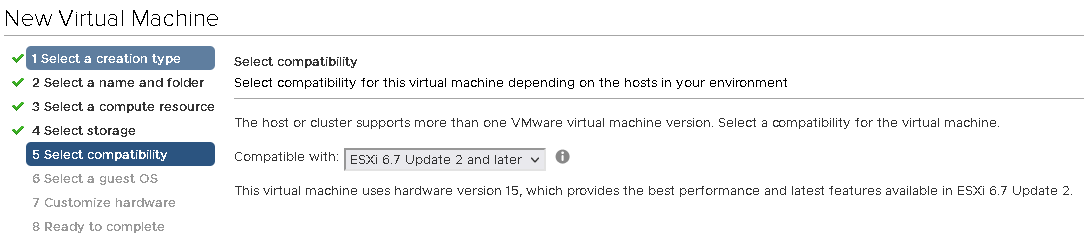


1. Type the name of Jenkins VM. Click next.

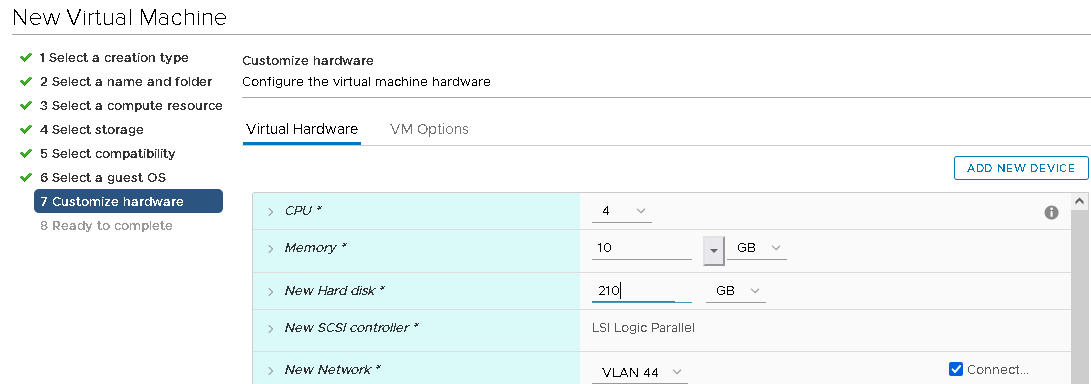
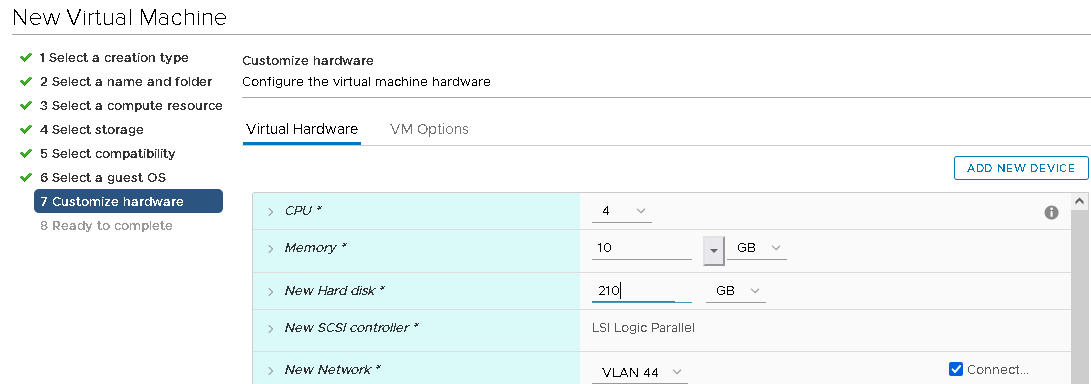


1. Select the data store. Click next

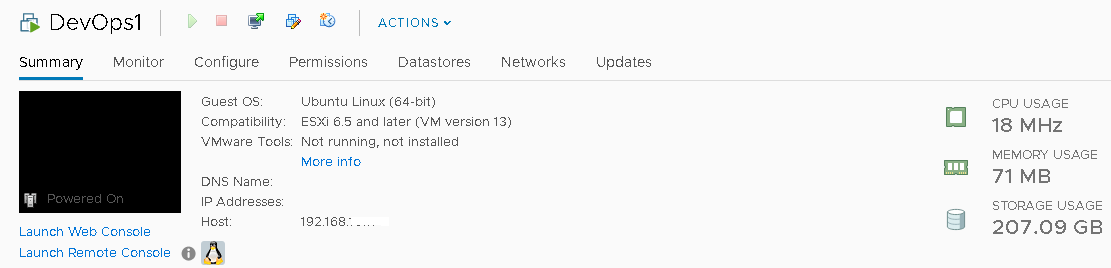


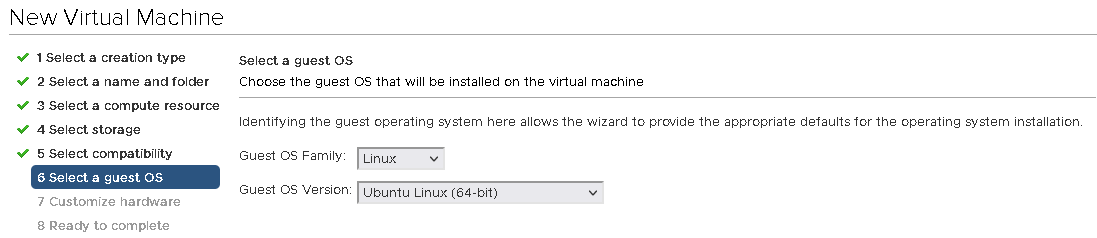
1. choose esxi version as compatibility metrics. Click next

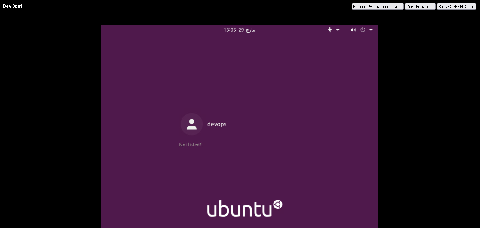
4. Assign compute resources, RAM, Processor, hard disk and ISO file and Network



1. Click next and after Ubuntu installation check installed vm resources.

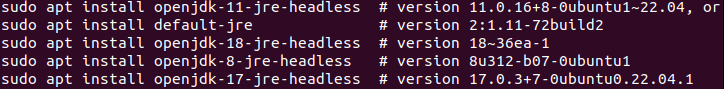


1. Create d the Salve VM in the Same WAY on VMWare host.
2. login to the server
3. Create devop1 user on both Master and Slave VM for password less authentication
4. Assign IPs (192.68.33.100 and 192.168.33.101 respectively to master and slave
5. Login as devop1 user to the client machine
6. ssh devop1@192.168.33.101
7. Create SSH private/public key pair without passphrase
8. ssh-keygen -t rsa -f ~/.ssh/id\_rsa
9. I have the ssh key so executed the following commands
10. ssh-keygen
11. ssh-copy-id user@host
12. ## or if your server uses custom port no:
13. ssh-copy-id "devop1@192.169.33.101 -p edureka2023"
14. ssh devop1@192.168.33.101



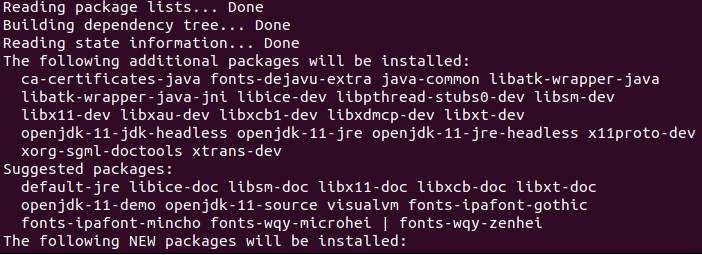
1. After setting up master slave password less authentication
2. logged in to the Ubuntu server, run the following commands for installation of java, Jenkin and docker.

java –version



sudo apt update

sudo apt install openjdk-11-jdk –y



**Add Jenkins Repository**

curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo tee /usr/share/keyrings/jenkins-keyring.asc &gt; /dev/null

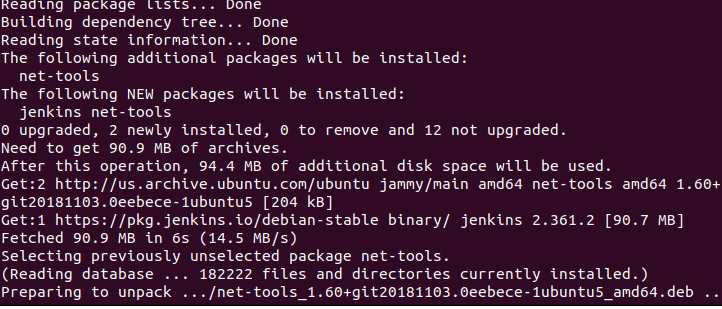
**Add the Jenkins software repository to the source list and provide the authentication key**

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 &gt; /dev/null

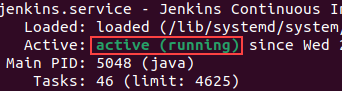
**Install Jenkins**

sudo apt update

sudo apt install jenkins –y



sudo systemctl status Jenkins



sudo systemctl enable --now Jenkins

**Modify Firewall to Allow Jenkins**

sudo ufw allow 8080

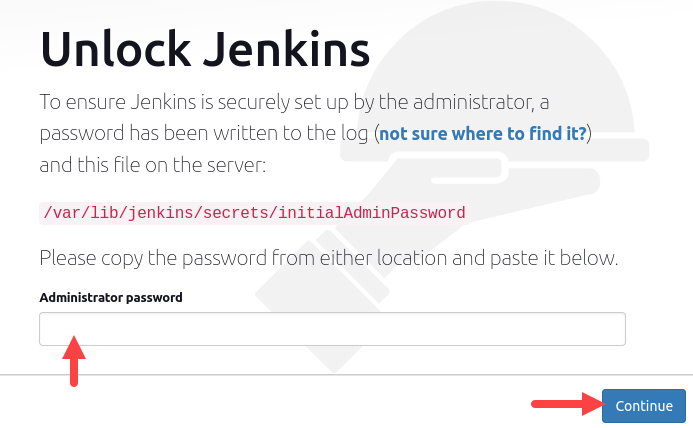


sudo ufw status

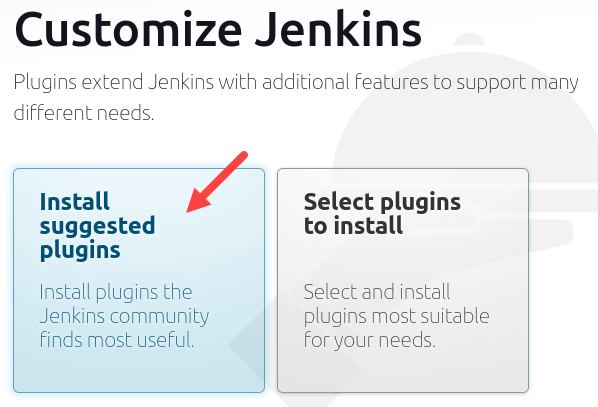
sudo ufw enable

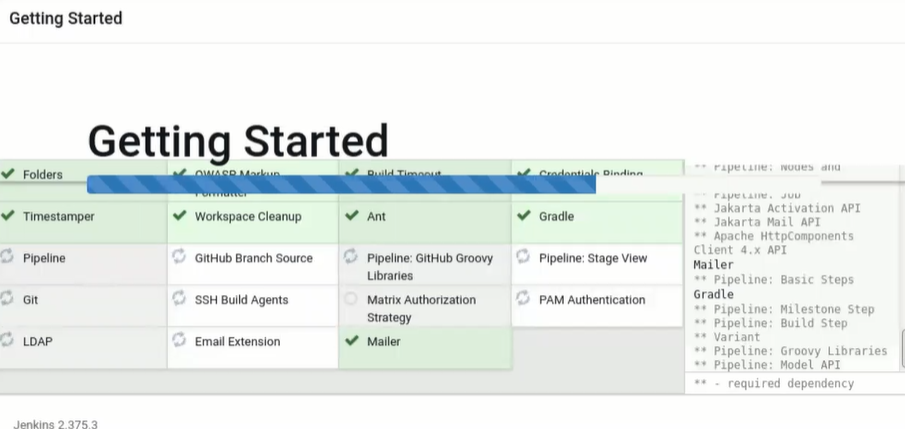
**Set up Jenkins**

<http://192.168.33.100:8080>

<http://localhost:8080>

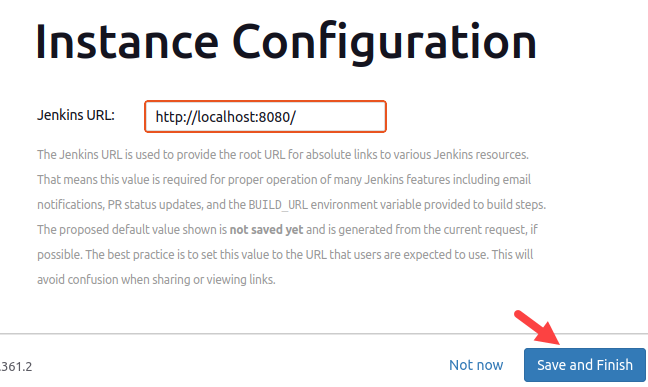
**To obtain the password run below command. It will show the password.**

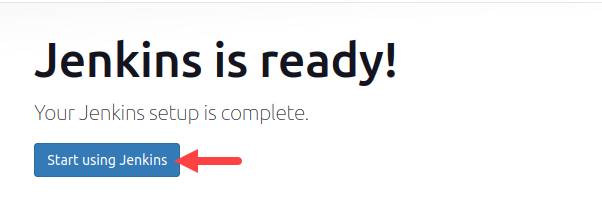
sudo cat /var/lib/jenkins/secrets/initialAdminPassword



* **We can also install plugins as required. Now create account and setup instance configuration. Now instance is ready.**







* Jenkins-server installation and user creation is finished now.
* Docker Installation with given below commands
* sudo apt-get update –y
* 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"

* sudo apt-get update –y
* sudo apt-get install docker-ce docker-ce-cli containerd.io –y
* sudo docker version
* sudo systemctl start docker
* sudo systemctl enable docker
* sudo groupadd docker
* sudo usermod -aG docker jenkins
* sudo systemctl restart jenkins
* sudo systemctl restart docker

1. **Master, Slave , Jenkins and Docker installation is finished now.**
2. **Kubernetes installation and creating cluster master and worker**
3. sudo apt-get update –y
4. sudo apt install docker.io (Install docker. Already installed)
5. docker ––version
6. sudo systemctl enable docker (Start and Enable Docker)
7. sudo systemctl start docker
8. sudo systemctl status docker
9. curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add (Install Kubernetes)
10. sudo apt-add-repository "deb http://apt.kubernetes.io/kubernetes-xenial main"

(Add Software Repositories)

1. sudo apt-get install kubeadm kubelet kubectl (Kubernetes Installation Tools)
2. sudo apt-mark hold kubeadm kubelet kubectl
3. Kubernetes Deployment sudo swapoff –a
4. Assign Unique Hostname for Each Server Node
5. sudo hostnamectl set-hostname master-node
6. sudo hostnamectl set-hostname worker1
7. Initialize Kubernetes on Master Node
8. sudo kubeadm init --pod-network-cidr=192.168.33.100/24
9. create a directory for the cluster:
10. mkdir -p $HOME/.kube
11. sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
12. sudo chown $(id -u):$(id -g) $HOME/.kube/config
13. Deploy Pod Network to Cluster
14. sudo kubectl apply -f <https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml>
15. Verify that everything is running and communicating:
16. kubectl get pods --all-namespaces

## Join the Worker Node to Cluster

1. Edit the hosts. Properties file and add the new worker node
2. master.ip=192.168.33.100
3. master.fqdn=devop
4. master.root\_password=edureka2023
5. master.external\_ip=9.30.15.9
6. worker.1.ip=9.0.10.1
7. worker.1.fqdn=devop1
8. worker.1.root\_password=edureka2023
9. ./install.sh --add-workers

To verify that the worker node was added to the cluster:

1. kubectl get nodes

NAME STATUS ROLES AGE VERSION

devop Ready master 3h44m v1.14.1

devop1 Ready <none> 81s v1.14.1

## Join the Worker Node to Cluster second method

Run the below command only on Master

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

kubeadm join 192.168.33.101:6443 --token 1642s5.ih5q6mdtf0pt9jey --discovery-token-ca-cert-hash sha256:d35bc841bd1ad7fd0223e506c8484bcafe9aa59427535b2709ed4b41201ce81b

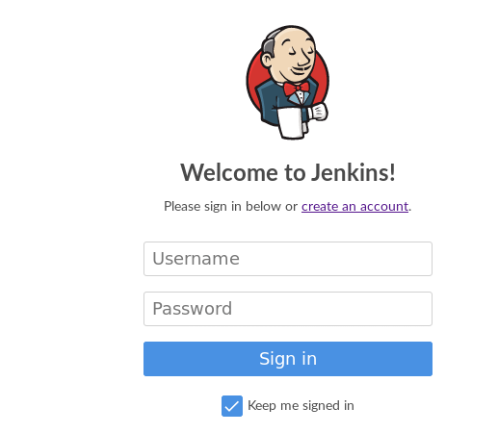
to regenerate token on master

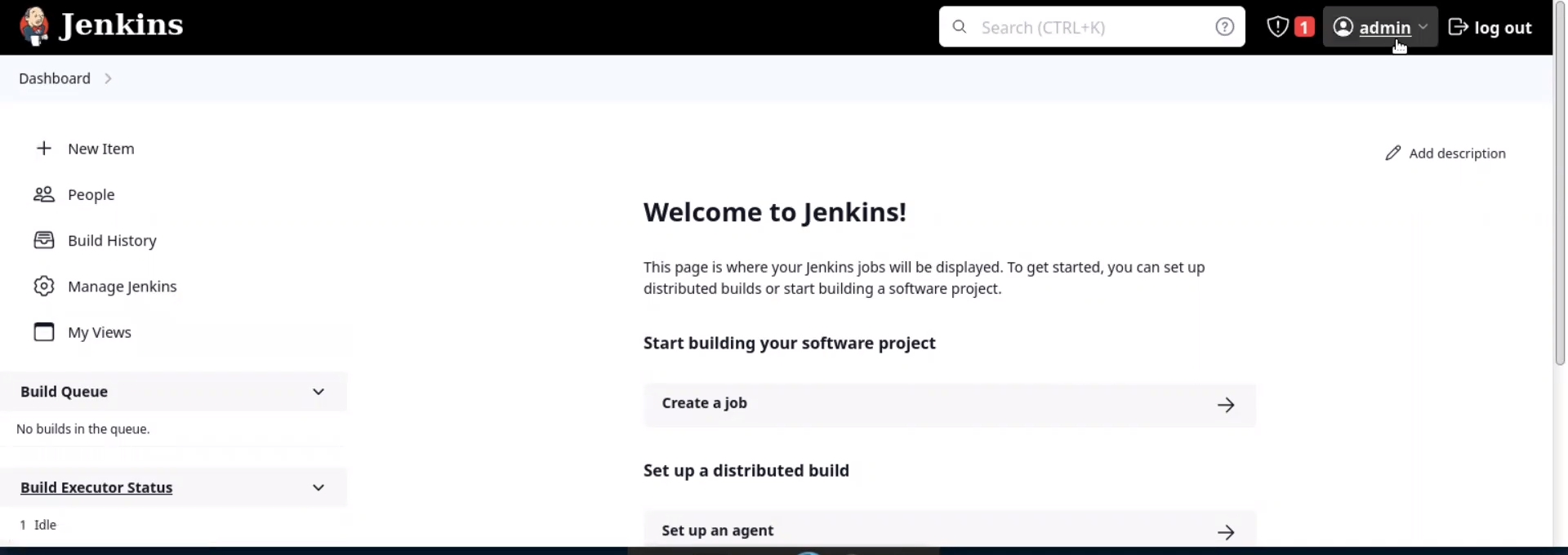
kubeadm token create --print-join-command

kubectl get nodes

install cluster networking plugin and run this command on both nodes master and worker

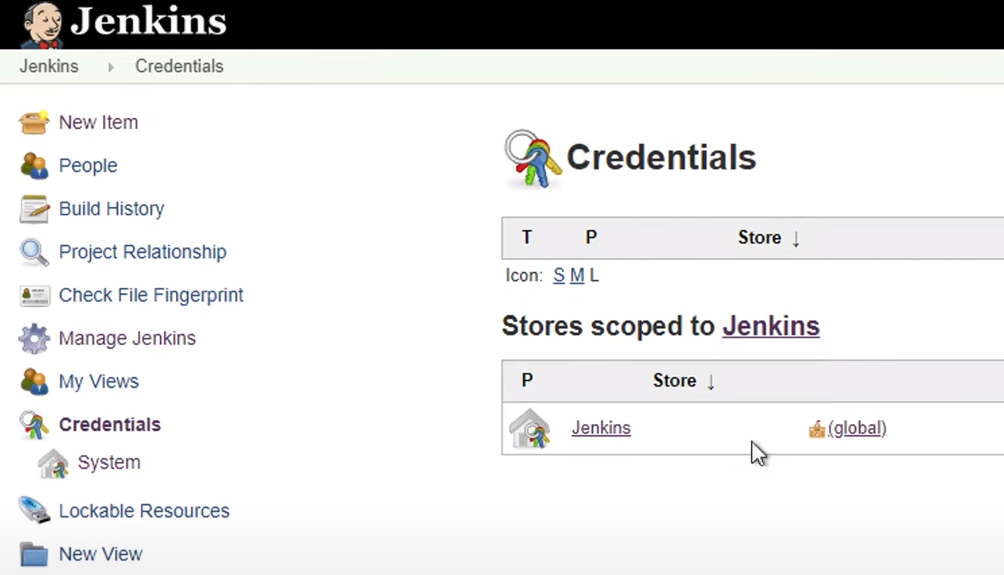
sudo sysctl net.bridge.bridge-nf-call-iptables=1

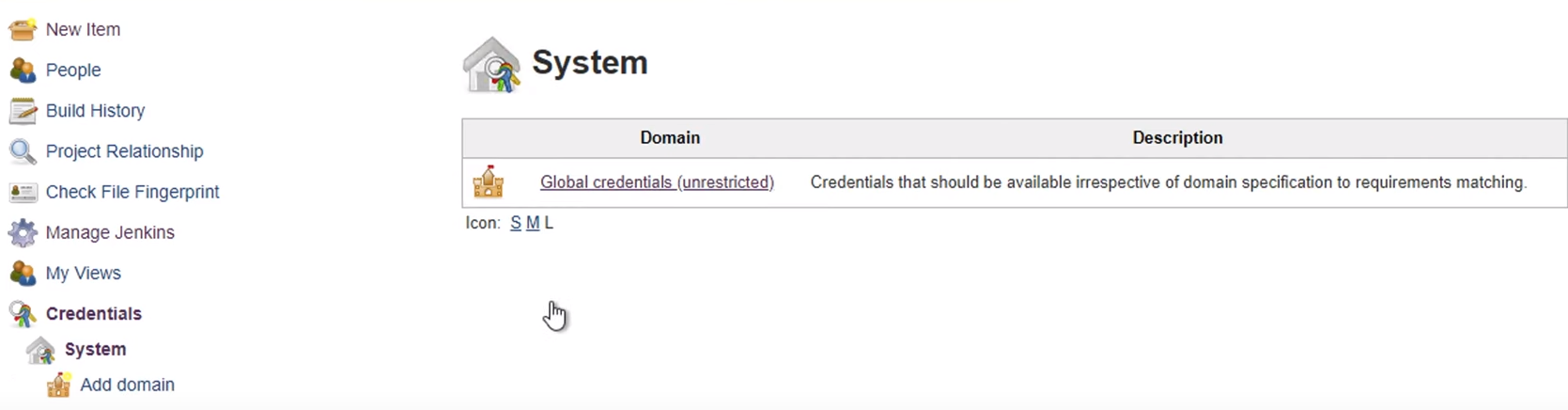
* **So kubernetes cluster nodes are ready**
* **To start with Jenkins devops pipelines**
* login to Jenkins UI: [192.168.33.100:8080](http://jenkins-server-public-ip:8080) as above Jenkins sever is configured earlier
* Install all the suggested plugins + continuous deploy plugin (for Kubernetes deployment)
* 

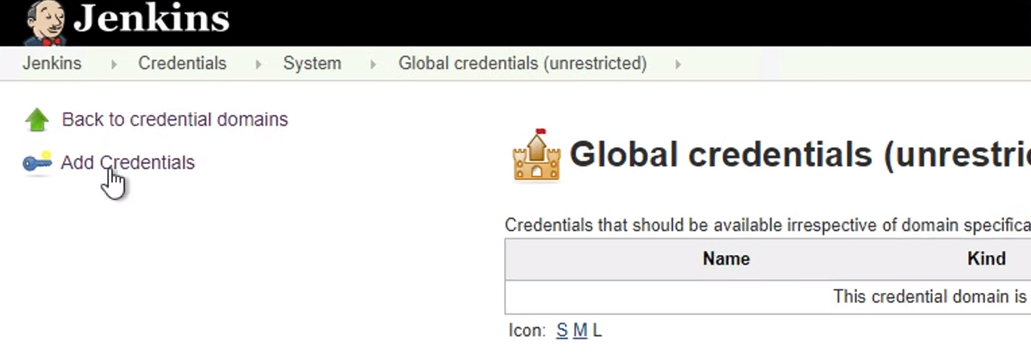


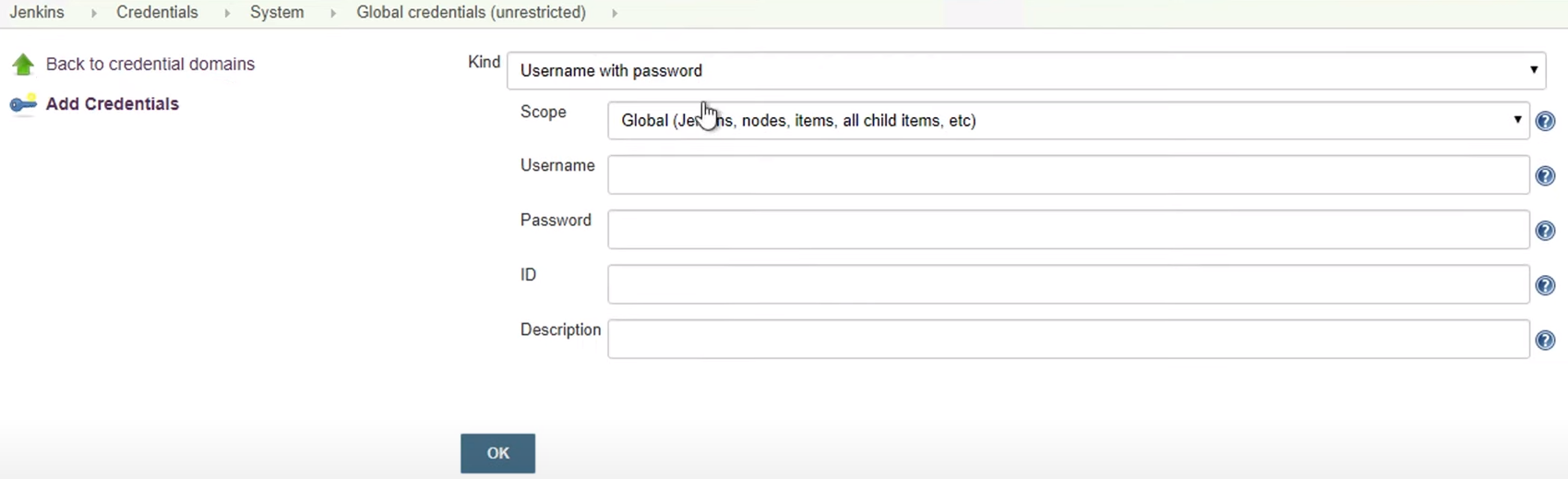
**Add GitHub Credentials in Jenkins**

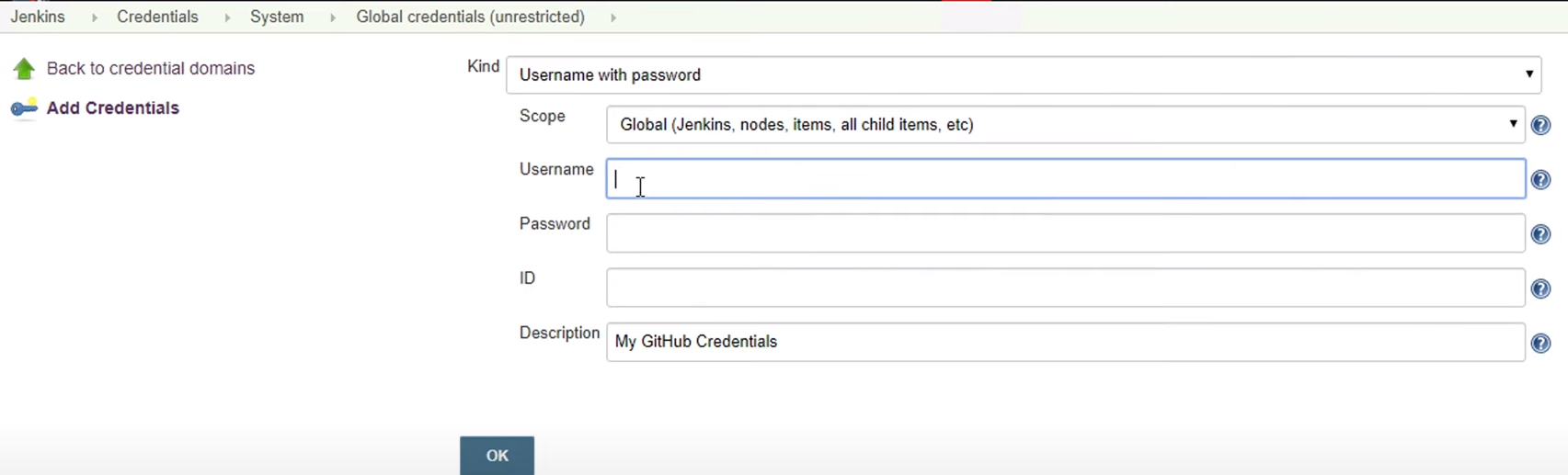
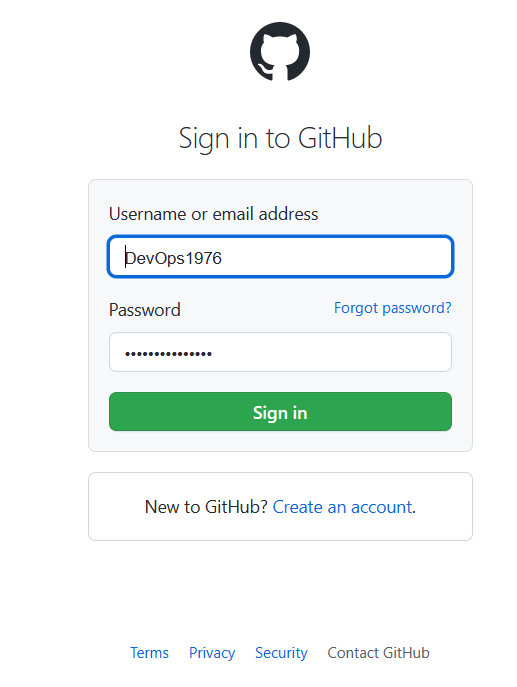
go to “Jenkins” -> Click on “Credentials” on the menu, which will list all the credentials that I have created inside Jenkins so far as shown below.

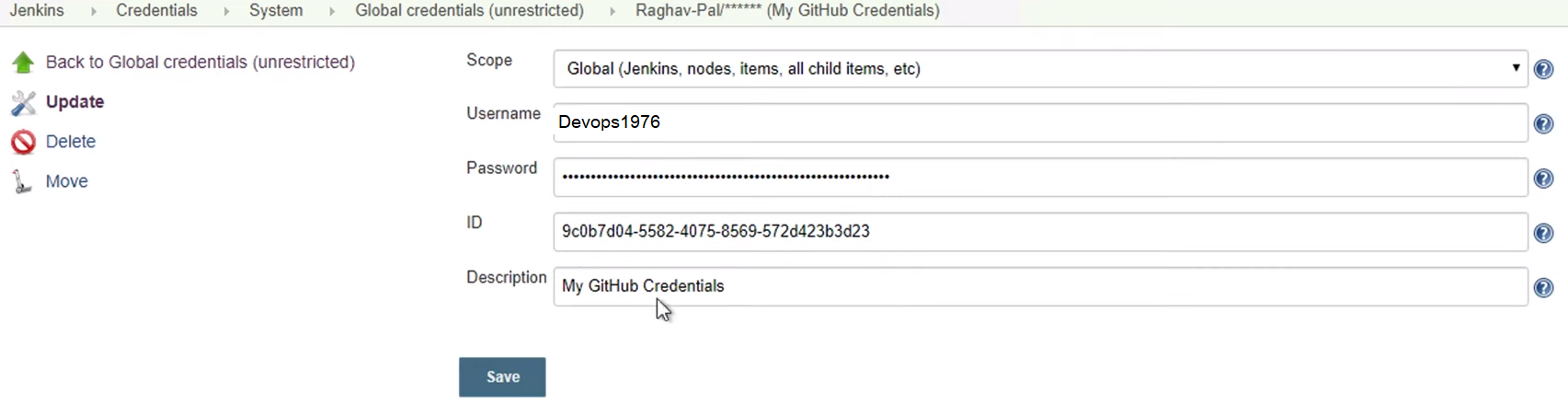


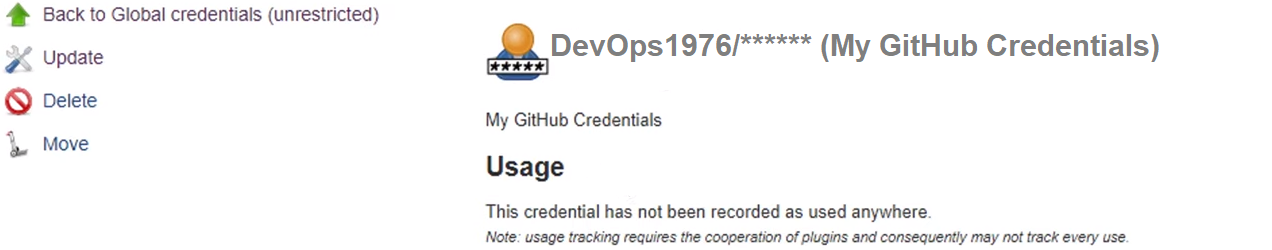


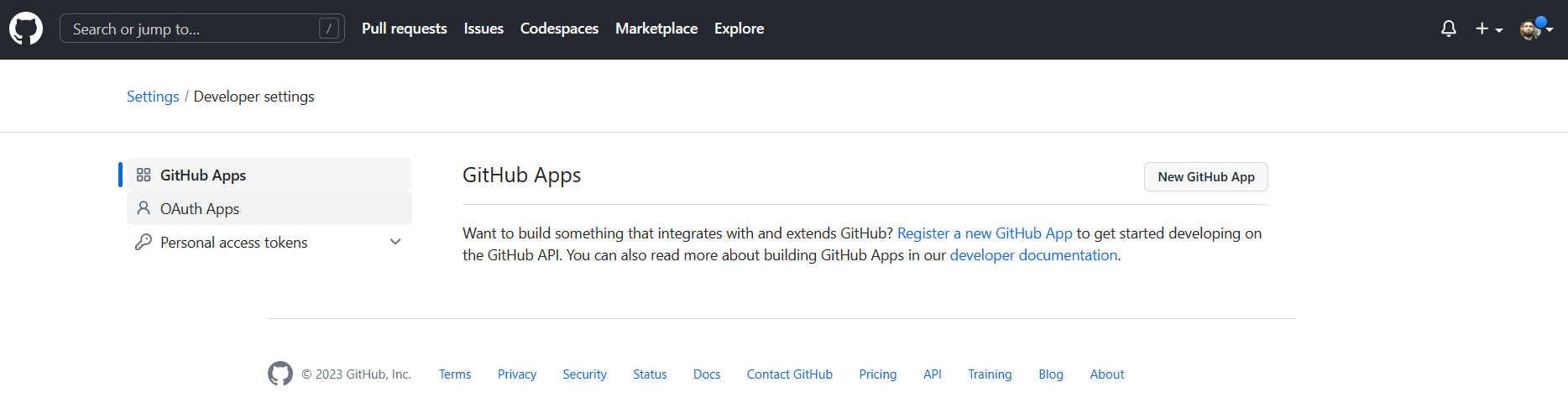


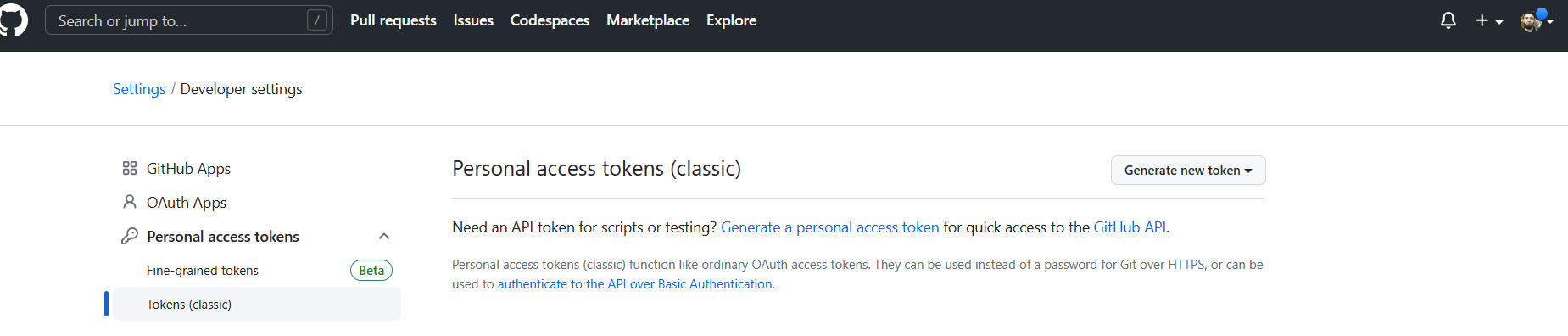


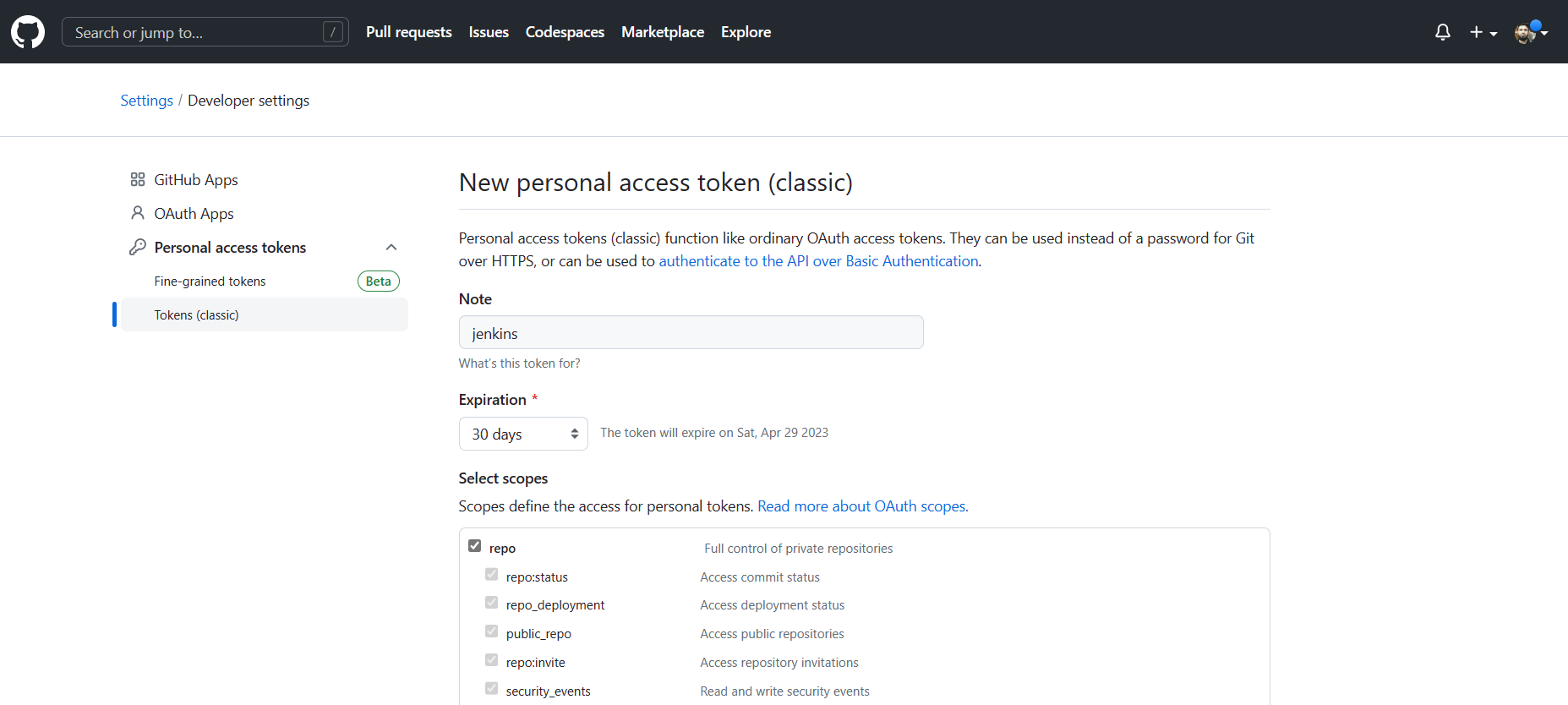
 

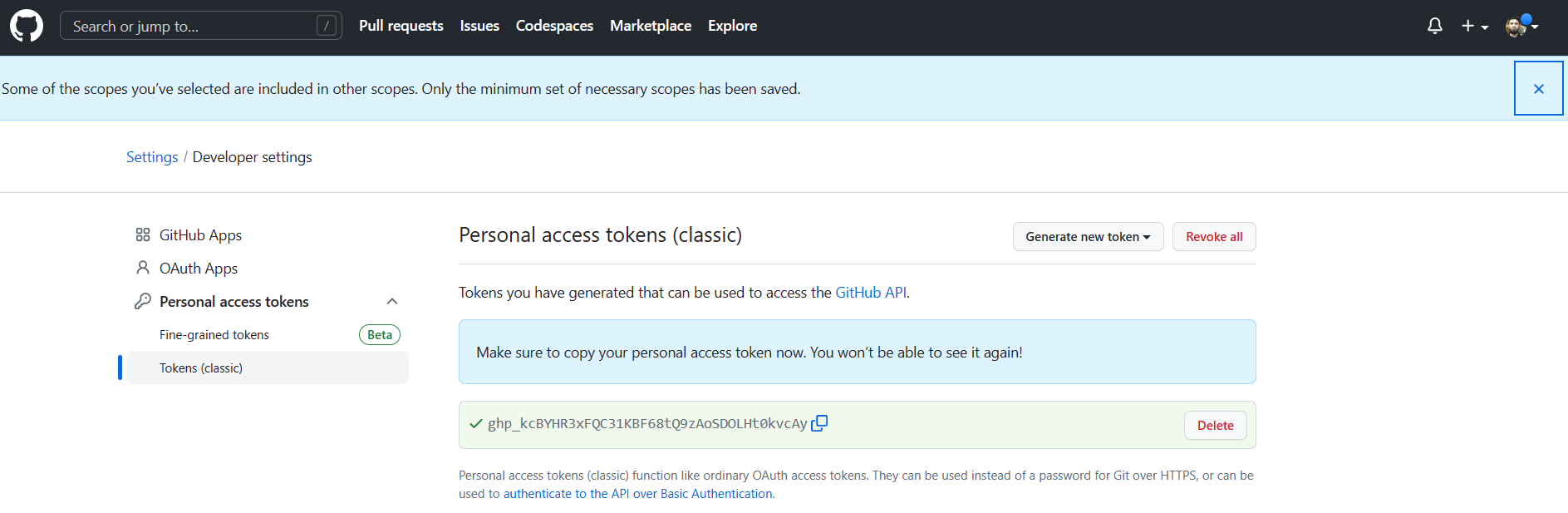






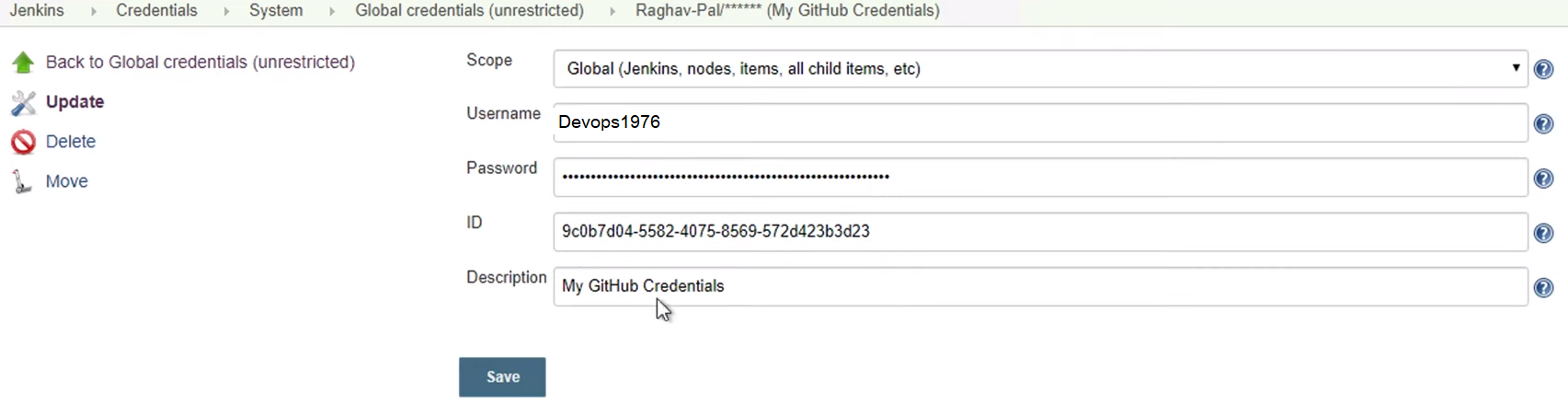






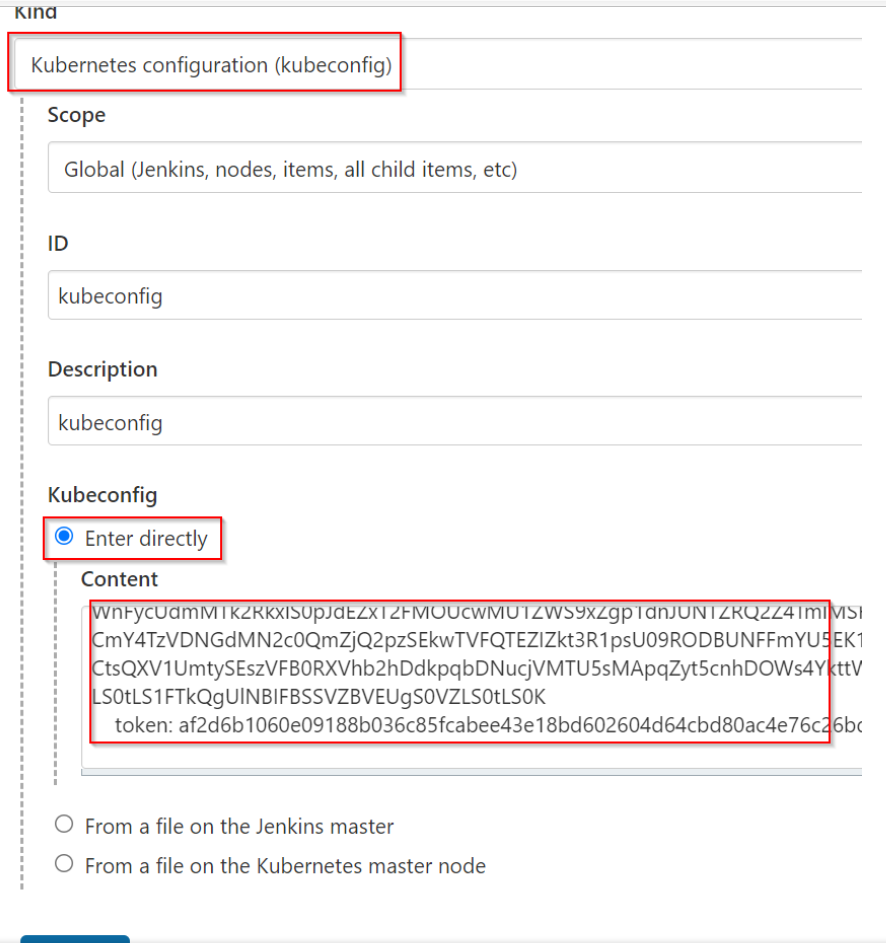
Copy token ghp\_kcBYHR3xFQC31KBF68tQ9zAoSDOLHt0kvcAy

Login to then Jenkins and Add Docker Hub Credentials in Jenkins



**Click Ok**

* **Add the Kubeconfig from the Kubernetes master as a credential in Jenkins**
* **Log in to the Kubernetes master node**
* ssh [devop@<192.168.33.100](mailto:devop@%3c192.168.33.100)>;
* display the contents of our Kubeconfig
* cat ~/.kube/config
* copy the output file
* Login to Kubernetes master server.
* Go to Jenkins Dashboard > Manage Jenkins > Manage Credentials > Jenkins > Global Credentials > Add Credential  
  Here select Type as Kubernetes configuration (kubeconfig)



* 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: 192.168.33.100
* 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
* 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.
* 192.168.33.100:8080
* 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 {
* myprjimg = "devops1976/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(myprjimg)
* 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://192.168.33.100:8081/",
* 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.
* **Kubernetes Horizontal Pod Autoscaler setup**
* #Log in to the master server.
* ssh devop@192.168.33.100
* #Install the Kubernetes Metrics API in the Cluster
* Clone the Git repository.
* git clone https://github.com/kubernetes-incubator/metrics-server.git
* Change to the metrics-server directory.
* cd metrics-server
* Check out the appropriate version of the repository.
* git checkout ed0663b3b4ddbfab5afea166dfd68c677930d22e
* Create the necessary objects.
* kubectl create -f deploy/1.8+/
* Make sure the server is up and running.
* kubectl get pods -n kube-system
* Verify everything is running as expected.
* kubectl get --raw /apis/metrics.k8s.io
* #Configure a Horizontal Pod Autoscaler to Autoscale the Train Schedule App
* Open a new browser tab and navigate to https://github.com/bhavukm/cicd-pipeline-train-schedule-autoscaling
* In the upper-right corner, click Fork and create a fork in your local repository.
* Click the link for the train-schedule-kube.yml file.
* Click the pencil icon to edit the file.
* Under the containers: section, add the following code to the bottom of that section. Make sure the spacing matches such that resources: is is even with the other top-level keys (i.e. image:, ports:, etc.)
* resources:
* requests:
* cpu: 200m
* At the very bottom of the file, add the following code.
* apiVersion: autoscaling/v2beta1
* kind: HorizontalPodAutoscaler
* metadata:
* name: train-schedule
* namespace: default
* spec:
* scaleTargetRef:
* apiVersion: apps/v1
* kind: Deployment
* name: train-schedule-deployment
* minReplicas: 1
* maxReplicas: 4
* metrics:
* - type: Resource
* resource:
* name: cpu
* targetAverageUtilization: 50
* Click Commit changes at the bottom of the window.
* Select the entire contents of the file and copy them to the clipboard.
* Back in the master server, return to the home directory.
* cd ~/
* Create train-schedule-kube.yml.
* vi train-schedule-kube.yml
* Paste the text copied to the clipboard in a previous step.
* Save the file and exit the editor.
* Apply our changes.
* kubectl apply -f train-schedule-kube.yml
* Make sure the pods are starting.
* kubectl get pods -w
* Verify the autoscaler is running.
* kubectl get hpa
* Test the System
* Open a new browser tab and navigate to the public IP address associated with the Kubernetes node. Specify the port on the address by using :8080.
* Generate CPU load on the server by navigating to NODE\_PUBLIC\_IP:8080/generate-cpu-load, replacing NODE\_PUBLIC\_IP with the public IP address associated with the Kubernetes node.
* Back in the terminal for the master server, check the status of the autoscaler.
* kubectl get hpa
* Set up a watch for the autoscaler.
* kubectl get hpa -w
* Open a new terminal window and connect to the master server.
* Run a busybox container.
* kubectl run -i --tty load-generator --image=busybox /bin/sh
* Generate load on the server to verify the autoscaler.
* while true; do wget -q -O- http://192.168.33.100:8080/generate-cpu-load; done