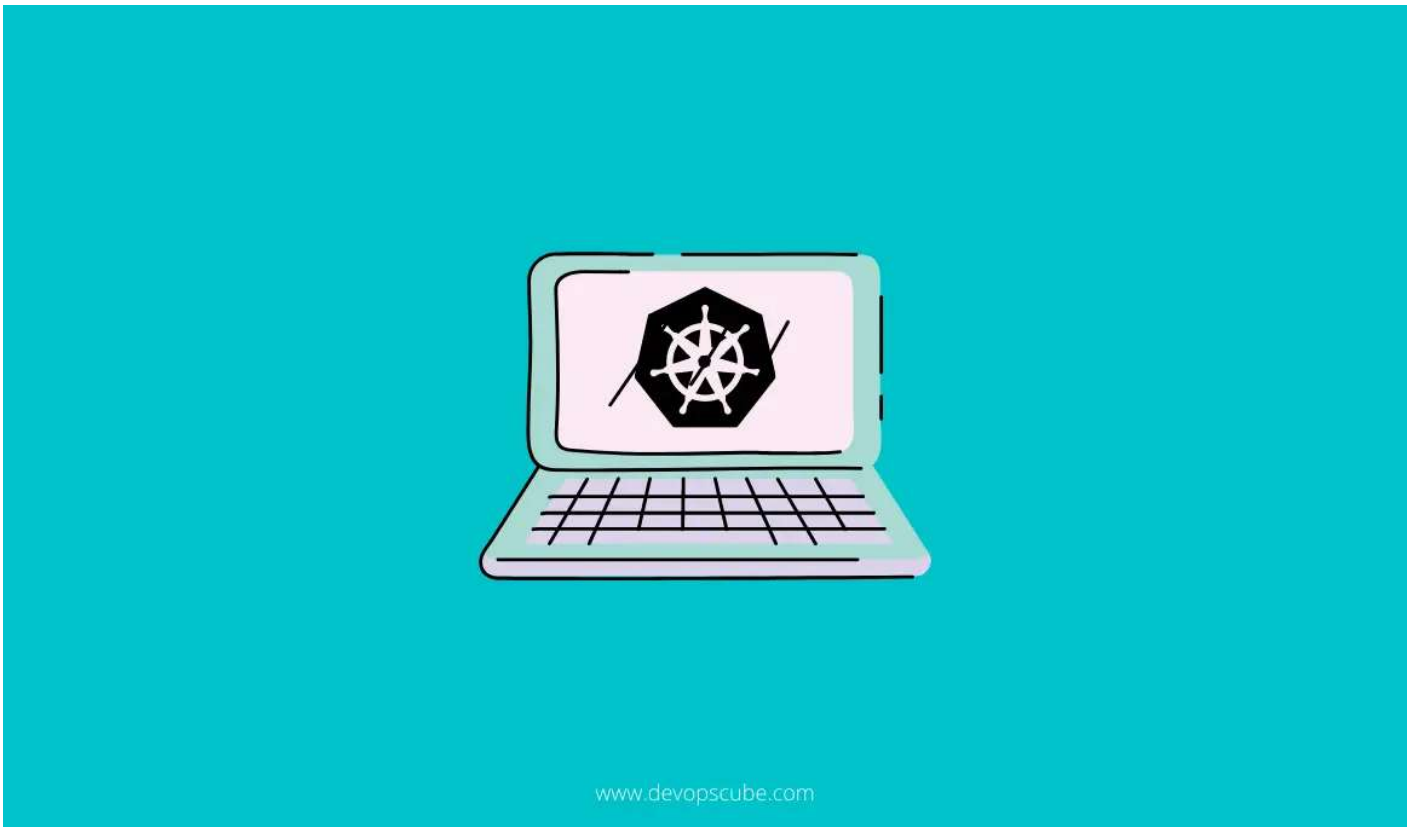
K — KUBERNETES

How to Setup Kubernetes Cluster on Vagrant VMs

by **Bibin Wilson** · May 19, 2021



In this Kubernetes tutorial, I have covered the step-by-step guide to setup Kubernetes cluster on Vagrant. It is a multinode kubernetes setup using kubeadm.

Vagrant is a great utility to set up Virtual machines on your local workstation. I pretty much use vagrant for most of my testing and learning purposes. If you are new to Vagrant, see my beginners vagrant guide

This guide primarily focuses on the Kubernetes automated setup using Vagrantfile and shell scripts. I have created this as part of my Certified Kubernetes Security Specialist exam guide with cluster version 1.20.

Automated Kubernetes Cluster Setup on Vagrant

I have written a basic Vagrantfile and scripts so that anyone can understand and make changes as per their requirements.

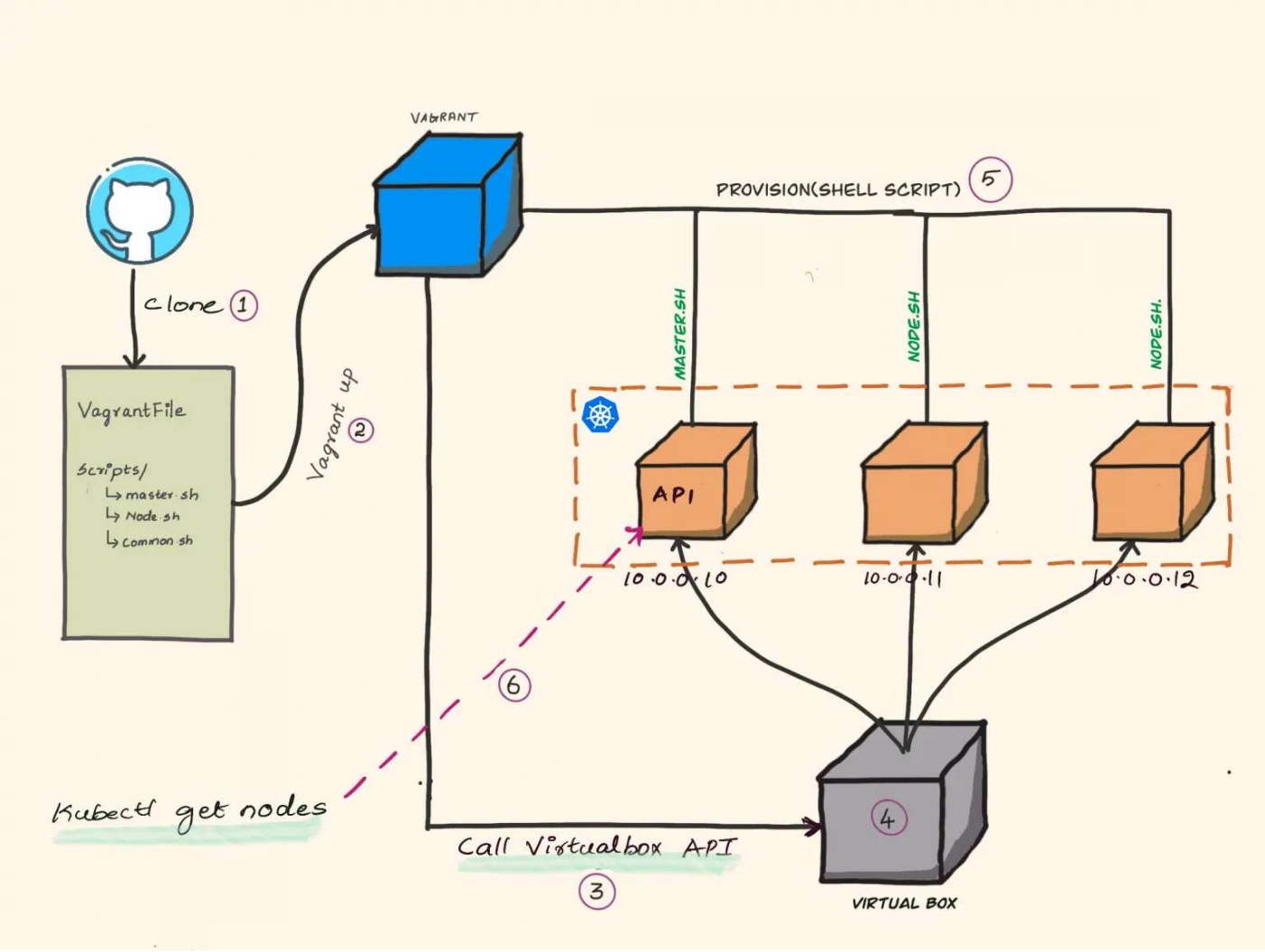
Here is the summary of the setup.

1. Single vagrant up command will create three VMs and configures all essential kubernetes components and configuration using Kubeadm.
2. Calico Network Plugin, Metrics server, and Kubernetes dashboard gets installed as part of the setup.
3. The kubeconfig file gets added to all the nodes in the cluster so that you can execute kubectl commands from any node.
4. The kubeconfig file and the kubernetes dashboard access token get added to the configs folder where you have the Vagrantfile. You can use the

5 You can shut down the VMs when not in use and start them again whenever needed. All the cluster configurations remain intact without any issues. The nodes get connected automatically to the master during the startup.

6 You can delete all the VMs in one command and recreate the setup with a `vagrant up` command any time you need.

Here is the high level overview of the setup.



If you are preparing for any of the Kubernetes certifications, you need a cluster to practice all the exam scenarios.

You can use these Vagrant scripts to set up your local practice environment.

And specifically, for CKA and CKS, you can expect Kubeadm related exam questions like bootstrapping and upgrading the kubernetes cluster using kubeadm.

For Kubeadm cluster bootstrapping, refer to my guide to setup Kubernetes cluster using Kubeadm

> **Important Note**: If you are preparing for CKA/CKAD/CKS certification, make use of the CKA/CKAD/CKS Voucher Codes before the price increases.

Kubernetes-Kubeadm Vagrant Github Repository

The Kubeadm Vagrantfile and scripts are hosted on Github repository.

Clone the repository to follow along with the guide.

Setup Kubernetes Cluster on Vagrant

> **Note:** You need a minimum of 16 Gig RAM workstation to run this setup without any issues.

Follow the steps given below to spin up the Kubernetes cluster and validate all the cluster configurations.

Step 1: To create the cluster, first cd into the cloned directory.

```
cd vagrant-kubeadm-kubernetes
```

Step 2: Execute the vagrant command. It will spin up three nodes. One master and two worker nodes. Kubernetes installation and configuration happens through the shell script present in the scripts folder.

```
vagrant up
```

> **Note:** If you are running it for the first time, Vagrant will first download the ubuntu box mentioned in the Vagrantfile. This is a one-time

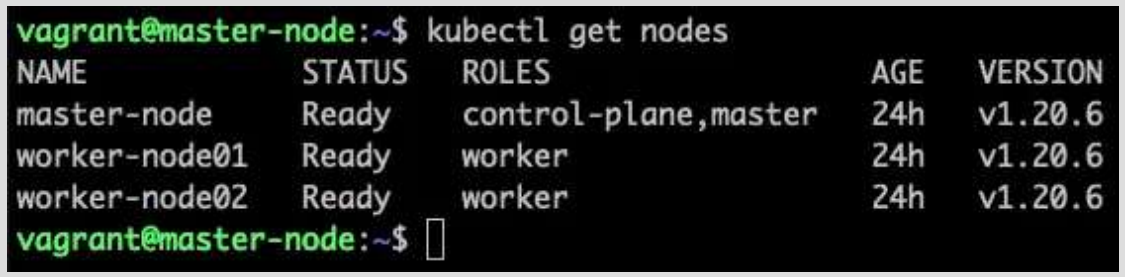
Step 3: Log in to the master node to verify the cluster configurations.

```
vagrant ssh master
```

Step 4: List all the cluster nodes to ensure the worker nodes are connected to the master and in a ready state.

```
kubectl top nodes
```

You should see the output as shown below.



Step 5: List all the pods in `kube-system` namespace and ensure it is in running state.

```
kubectl get po -n kube-system
```

```
vagrant@master-node:~$ kubectl get po -n kube-system
NAME                                      READY   STATUS
calico-kube-controllers-6d7b4db76c-ls5lc  1/1     Running
calico-node-b2b29                         1/1     Running
calico-node-kc5bg                         1/1     Running
calico-node-lb4nl                         1/1     Running
coredns-74ff55c5b-9dptm                   1/1     Running
coredns-74ff55c5b-tmbrm                   1/1     Running
etcd-master-node                          1/1     Running
kube-apiserver-master-node                1/1     Running
kube-controller-manager-master-node       1/1     Running
kube-proxy-l7hhd                          1/1     Running
kube-proxy-psfg5                          1/1     Running
kube-proxy-spth2                          1/1     Running
kube-scheduler-master-node                1/1     Running
metrics-server-65f8747c84-hz62k           1/1     Running
vagrant@master-node:~$
```

Step 6: Deploy a sample Nginx app and see if you can access it over the nodePort.

```
kubectl apply -f https://raw.githubusercontent.com/scriptcamp/kubeadm-scripts/main/manifests/sample-app.yaml
```

You should be able to access Nginx on any of the node's IPs on port `32000` . For example, `http://10.0.0.11:32000`

That's it! You can start deploying and testing other applications.

To shut down the Kubernetes VMs, execute the halt command.

```
vagrant halt
```

Whenever you need the cluster, just execute,

```
vagrant up
```

To destroy the VMs,

```
vagrant destroy
```

> **Note:** If you want applications to persist data on each cluster or pod restart, make sure you use the persistent volume type "local" attached to a nodeSelector.

Access Kubernetes Cluster From Workstation Terminal

Once Vagrant execution is successful, you will see a configs folder with a few files (config, join.sh, and token) inside the cloned repo. These are generated during the run time.

Copy the config file to your `$HOME/.kube` folder if you want to interact with the cluster from your workstation terminal. You should have kubectl installed on your workstation.

For example, I did the following in my mac keeping `vagrant-kubeadm-kubernetes` folder as the current directory.

```
mkdir -p $HOME/.kube
cp configs/config $HOME/.kube
```

Alternatively, you can set a Kubeconfig env variable as shown below. Make sure you

```
export KUBECONFIG=$(PWD)/config
```

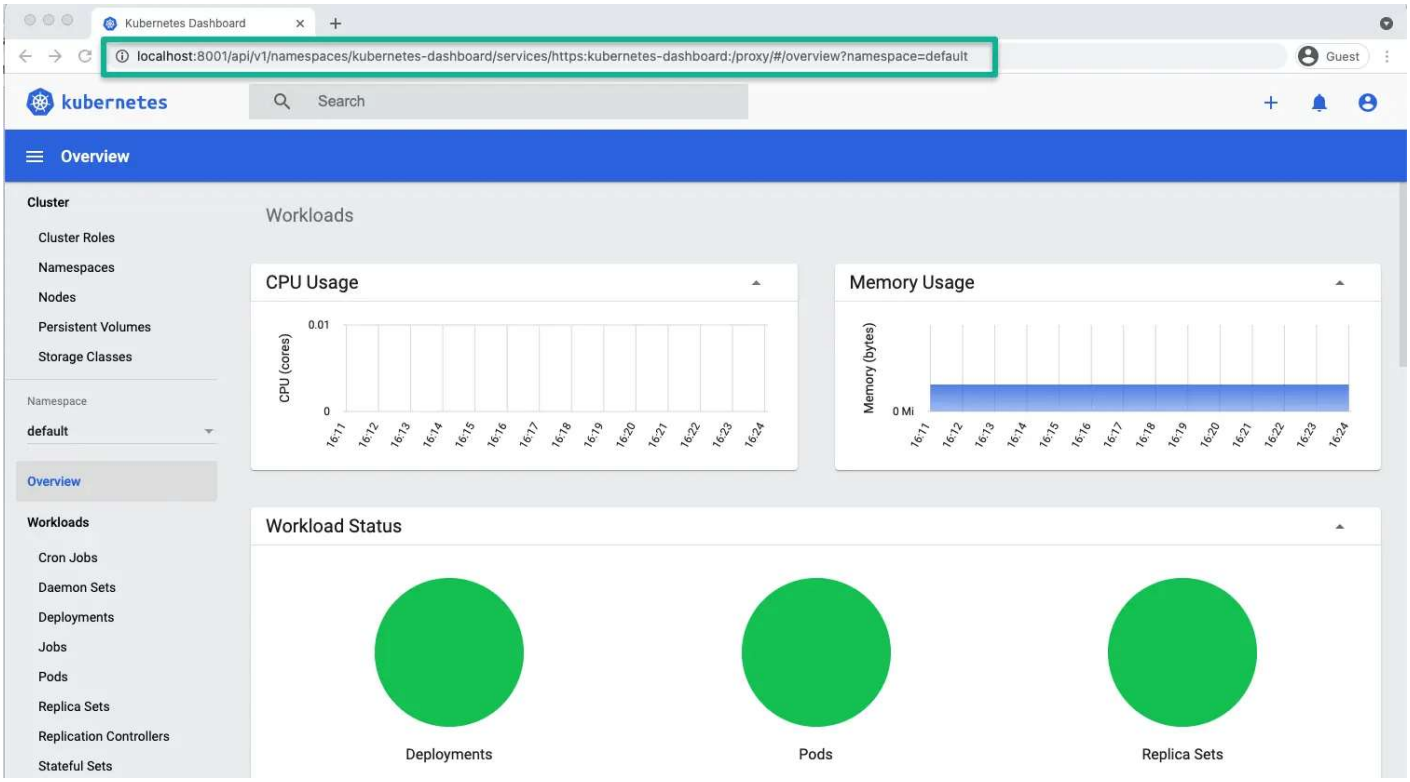
Once you copy the kubeconfig (config) file to your local `$HOME/.kube` directory you can run the kubectl command against the cluster and access the kubernetes dashboard.

Run kubectl proxy to access the Kubernetes dashboard.

```
kubectl proxy
```

The `token` file inside the configs folder contains the sign-in token for the kubernetes dashboard. If you want to use the kubernetes dashboard, use the token and log in from the following URL

```
http://localhost:8001/api/v1/namespaces/kubernetes-
dashboard/services/https:kubernetes-dashboard:/proxy/#/login
```



Kubeadm Vagrantfile & Scripts Explanation

Here is the file tree for the Vagrant repo.

```
├── Vagrantfile
├── configs
│   ├── config
│   ├── join.sh
│   └── token
└── scripts
    ├── common.sh
    ├── master.sh
    └── node.sh
```

The configs folder and files get generated only after the first run

As I explained earlier, the config file contains the config, token, and join.sh file.

In the previous section, I have already explained `config` and `token`. The `join.sh` file has the worker node join command with the token created during kubeadm master node initialization.

Since all the nodes share the folder containing the Vagrantfile, the worker nodes can read the `join.sh` file and join the master automatically during the first run. It is a one-time task.

If you login to any node and access the `/vagrant` folder, you will see Vagrantfile and scripts as it is shared between the VMs.

Let's have a look at the Vagrantfile

```
Vagrant.configure("2") do |config|
    config.vm.provision "shell", inline: <<-SHELL
        apt-get update -y
        echo "10.0.0.10  master-node" >> /etc/hosts
        echo "10.0.0.11  worker-node01" >> /etc/hosts
        echo "10.0.0.12  worker-node02" >> /etc/hosts
    SHELL

    config.vm.define "master" do |master|
      master.vm.box = "bento/ubuntu-18.04"
      master.vm.hostname = "master-node"
      master.vm.network "private_network", ip: "10.0.0.10"
      master.vm.provider "virtualbox" do |vb|
          vb.memory = 4048
          vb.cpus = 2
      end
      master.vm.provision "shell", path: "scripts/common.sh"
```

```
    (1..2).each do |i|

    config.vm.define "node0#{i}" do |node|
      node.vm.box = "bento/ubuntu-18.04"
      node.vm.hostname = "worker-node0#{i}"
      node.vm.network "private_network", ip: "10.0.0.1#{i}"
      node.vm.provider "virtualbox" do |vb|
          vb.memory = 2048
          vb.cpus = 1
      end
      node.vm.provision "shell", path: "scripts/common.sh"
      node.vm.provision "shell", path: "scripts/node.sh"
    end

    end
end
```

As you can see, I have added the following IPs for nodes, and it is added to the host's file entry of all the nodes with its hostname with a common shell block that gets executed on all the VMs.

1. 10.0.0.10 (master)
2. 10.0.0.11 (node01)
3. 10.0.0.11 (node02)

Also, the worker node block is in a loop. So if you want more than two worker nodes or have only one worker node, you need to replace `2` with the desired number in the loop declaration. If you add more nodes, ensure you add the IP to the host's file entry.

For example, for 3 worker nodes, you need to have `1..4` loop.

master.sh, node.sh and common.sh

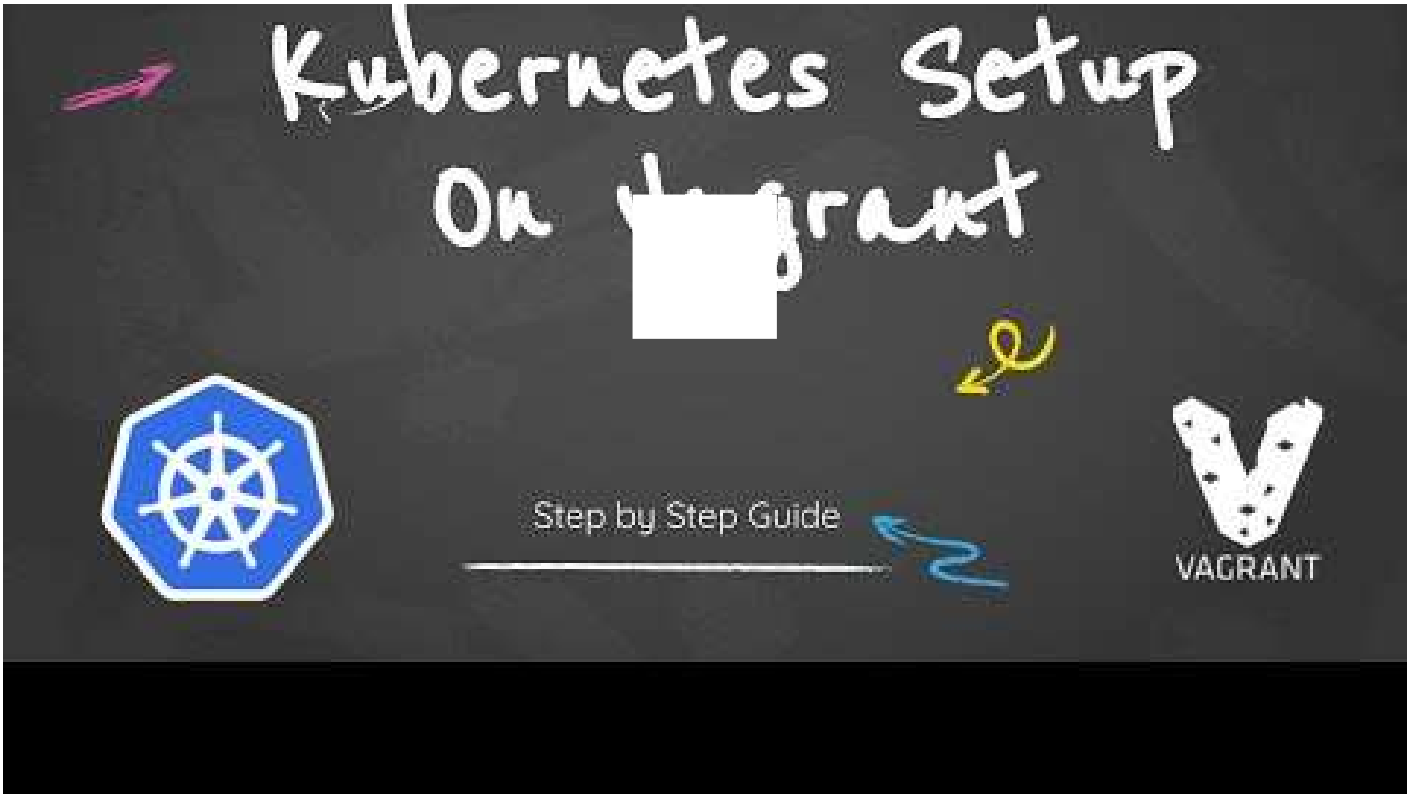
The three shell scripts get called as `provisioners` during the Vagrant run to configure the cluster.

1. **common.sh**: – A self-explanatory list of commands which installs docker, kubeadm, kubectl and kubelet on all the nodes. Also, disables swap.
2. **master.sh**: – contains commands to initialize master, install the calico plugin, metrics server, and kubernetes dashboard. Also, copies the kube-config, join.sh, and token files to the configs directory.
3. **node.sh:-** reads the join.sh command from the configs shared folder and join the master node. Also, copied the kubeconfig file to /home/vagrant/.kube location to execute kubectl commands.

`common.sh` installs kubernetes version `1.20.6-00` to have the same cluster version for CKA/CKAD and CKS preparation. If you would like the latest version, remove the version number from the command.

Video Documenation

I have documented the whole process in a Youtube video. Check out the video if you want to see the live setup.



Conclusion

It is good to have a Local kubernetes cluster setup that you can spin up and tear down whenever you need without spending much time.

To set up the kubernetes cluster on Vagrant, all you have to do is, clone the repo and run the vagrant up command.

Moreover, you are a DevOps engineer and work on the Kubernetes cluster, you can have a production-like setup locally for development and testing.

You can add more tools and utilities like helm, ingress controller, Prometheus, etc to the existing script and customize as per your requirements.