Install Kubernetes Cluster on Ubuntu 20.04 with kubeadm

By Josphat Mutai - November 30, 2021

Kubernetes is a tool for orchestrating and managing containerized applications at scale on on-premise server or across hybrid cloud environments. Kubeadm is a tool provided with Kubernetes to help users install a production ready Kubernetes cluster with best practices enforcement. This tutorial will demonstrate how one can install a Kubernetes Cluster on Ubuntu 20.04 with kubeadm.

For Debian installation: Deploy Kubernetes Cluster on Debian 10 with Kubespray

For Rocky Linux 8: Install Kubernetes Cluster on Rocky Linux 8 with Kubeadm & CRI-O

There are two server types used in deployment of Kubernetes clusters:

- **Master**: A Kubernetes Master is where control API calls for the pods, replications controllers, services, nodes and other components of a Kubernetes cluster are executed

X

The minimum requirements for the viable setup are:

- Memory: **2 GiB** or more of RAM per machine
- CPUs: At least **2 CPUs** on the control plane machine.
- Internet connectivity for pulling containers required (Private registry can also be used)
- Full network connectivity between machines in the cluster – This is private or public

Install Kubernetes Cluster on Ubuntu 20.04

My Lab setup contain three servers. One control plane machine and two nodes to be used for running containerized workloads. You can add more nodes to suit your desired use case and load, for example using **three** control plane nodes for HA.

Server Type	Server Hostname	Specs
Master	k8s-master01.computingforgeeks.com	4GB Ram, 2vcpus
Worker	k8s-worker01.computingforgeeks.com	4GB Ram, 2vcpus
Worker	k8s-worker02.computingforgeeks.com	4GB Ram, 2vcpus

Step 1: Install Kubernetes Servers

Provision the servers to be used in the deployment of Kubernetes on Ubuntu 20.04. The setup process will vary depending on the virtualization or cloud environment you're using.

Once the servers are ready, update them.

```
sudo apt update
sudo apt -y upgrade && sudo systemctl reboot
```

Step 2: Install kubelet, kubeadm and kubectl

Once the servers are rebooted, add Kubernetes repository for Ubuntu 20.04 to all the servers.

```
sudo apt update
```

```
echo "deb https://apt.kubernetes.io/ kubernetes-xenial main" | sudo
tee /etc/apt/sources.list.d/kubernetes.list
```

Then install required packages.

```
sudo apt update
sudo apt -y install vim git curl wget kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl
```

Confirm installation by checking the version of kubectl.

```
$ kubectl version --client && kubeadm version
Client Version: version.Info{Major:"1", Minor:"22",
GitVersion:"v1.22.2",
GitCommit:"8b5a19147530eaac9476b0ab82980b4088bbc1b2",
GitTreeState:"clean", BuildDate:"2021-09-15T21:38:50Z",
GoVersion:"go1.16.8", Compiler:"gc", Platform:"linux/amd64"}
kubeadm version: &version.Info{Major:"1", Minor:"22",
GitVersion:"v1.22.2",
GitCommit:"8b5a19147530eaac9476b0ab82980b4088bbc1b2",
GitTreeState:"clean", BuildDate:"2021-09-15T21:37:34Z",
GoVersion:"go1.16.8", Compiler:"gc", Platform:"linux/amd64"}
```

Step 3: Disable Swap

Turn off swap.

```
sudo sed -i '/ swap / s/^\(.*\)$/#\1/g' /etc/fstab
sudo swapoff -a
```

Enable kernel modules and configure sysctl.

```
sudo modprobe overlay
sudo modprobe br_netfilter

# Add some settings to sysctl
sudo tee /etc/sysctl.d/kubernetes.conf<<EOF
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
net.ipv4.ip_forward = 1
EOF

# Reload sysctl
sudo sysctl --system
```

Step 4: Install Container runtime

To run containers in Pods, Kubernetes uses a container runtime. Supported container runtimes are:

- Docker
- CRI-O
- Containerd

NOTE: You have to choose one runtime at a time.

Installing Docker runtime:

```
# Add repo and Install packages
sudo apt update
sudo apt install -y curl gnupg2 software-properties-common apt-transport-https ca-certificates
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 update
sudo apt install -y containerd.io docker-ce docker-ce-cli

# Create required directories
sudo mkdir -p /etc/systemd/system/docker.service.d

# Create daemon json config file
sudo tee /etc/docker/daemon.json <<EOF
{
  "exec-opts": ["native.cgroupdriver=systemd"]
```

```
    "max-size": "100m"
  },
  "storage-driver": "overlay2"
}
EOF

# Start and enable Services
sudo systemctl daemon-reload
sudo systemctl restart docker
sudo systemctl enable docker
```

Installing CRI-O:

```
# Ensure you load modules
sudo modprobe overlay
sudo modprobe br_netfilter

# Set up required sysctl params
sudo tee /etc/sysctl.d/kubernetes.conf<<EOF
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
net.ipv4.ip_forward = 1
EOF

# Reload sysctl
sudo sysctl --system

# Add Cri-o repo
sudo su -
OS="xUbuntu_20.04"
VERSION=1.22
echo "deb https://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable/$OS/ /" > /etc/apt/sources.list.d/devel:kubic:libcontainers:stable.list
echo "deb http://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable:/cri-o:/$VERSION/$OS/ /" > /etc/apt/sources.list.d/devel:kubic:libcontainers:stable:cri-o:$VERSION.list
curl -L https://download.opensuse.org/repositories/devel:kubic:libcontainers:stable:cri-o:$VERSION/$OS/Release.key | apt-key add -
curl -L https://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable/$OS/Release.key | apt-key add -

# Update CRI-O CIDR subnet
# Install CRI-O
sudo apt update
sudo apt install cri-o cri-o-runc

# Start and enable Service
sudo systemctl daemon-reload
sudo systemctl restart crio
sudo systemctl enable crio
sudo systemctl status crio
```

Installing Containerd:

```
# Configure persistent loading of modules
sudo tee /etc/modules-load.d/containerd.conf <<EOF
overlay
br_netfilter
EOF

# Load at runtime
sudo modprobe overlay
sudo modprobe br_netfilter

# Ensure sysctl params are set
sudo tee /etc/sysctl.d/kubernetes.conf<<EOF
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
net.ipv4.ip_forward = 1
EOF

# Reload configs
sudo sysctl --system

# Install required packages
sudo apt install -y curl gnupg2 software-properties-common apt-transport-https ca-certificates

# Add Docker repo
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"

# Install containerd
# Configure containerd and start service
sudo su -
mkdir -p /etc/containerd
containerd config default>/etc/containerd/config.toml

# restart containerd
sudo systemctl restart containerd
sudo systemctl enable containerd
systemctl status  containerd
```

To use the systemd cgroup driver, set **plugins.cri.systemd_cgroup = true** in `/etc/containerd/config.toml`. When using kubeadm, manually configure the cgroup driver for kubelet

Step 5: Initialize master node

Login to the server to be used as master and make sure that the *br_netfilter* module is loaded:

```
$ lsmod | grep br_netfilter
br_netfilter           22256  0
bridge                151336  2 br_netfilter,ebtable_broute
```

Enable kubelet service.

```
sudo systemctl enable kubelet
```

We now want to initialize the machine that will run the control plane components which includes *etcd* (the cluster database) and the API Server.

```
[config/images] Pulled k8s.gcr.io/kube-apiserver:v1.22.2
[config/images] Pulled k8s.gcr.io/kube-controller-manager:v1.22.2
[config/images] Pulled k8s.gcr.io/kube-scheduler:v1.22.2
[config/images] Pulled k8s.gcr.io/kube-proxy:v1.22.2
[config/images] Pulled k8s.gcr.io/pause:3.5
[config/images] Pulled k8s.gcr.io/etcd:3.5.0-0
[config/images] Pulled k8s.gcr.io/coredns/coredns:v1.8.4
```

If you have multiple CRI sockets, please use `--cri-socket` to select one:

```
# CRI-O
sudo kubeadm config images pull --cri-socket /var/run/crio/crio.sock

# Containerd
sudo kubeadm config images pull --cri-socket /run/containerd/containerd.sock

# Docker
sudo kubeadm config images pull --cri-socket /var/run/docker.sock
```

These are the basic `kubeadm init` options that are used to bootstrap cluster.

```
--control-plane-endpoint :  set the shared endpoint for all control-plane nodes. Can be DNS/IP
--pod-network-cidr : Used to set a Pod network add-on CIDR
--cri-socket : Use if have more than one container runtime to set runtime socket path
--apiserver-advertise-address : Set advertise address for this particular control-plane node's API server
```

Bootstrap without shared endpoint

To bootstrap a cluster without using DNS endpoint, run:

```
sudo kubeadm init \
  --pod-network-cidr=192.168.0.0/16
```

Bootstrap with shared endpoint (DNS name for control plane API)

Set cluster endpoint DNS name or add record to /etc/hosts file.

```
$ sudo vim /etc/hosts
172.29.20.5 k8s-cluster.computingforgeeks.com
```

Create cluster:

```
  --upload-certs \
  --control-plane-endpoint=k8s-cluster.computingforgeeks.com
```

Note: If *192.168.0.0/16* is already in use within your network you must select a different pod network CIDR, replacing 192.168.0.0/16 in the above command.

Container runtime sockets:

Runtime	Path to Unix domain socket
Docker	`/var/run/docker.sock`
containerd	`/run/containerd/containerd.sock`
CRI-O	`/var/run/crio/crio.sock`

You can optionally pass Socket file for runtime and advertise address depending on your setup.

```
# CRI-O
sudo kubeadm init \
  --pod-network-cidr=192.168.0.0/16 \
  --cri-socket /var/run/crio/crio.sock \
  --upload-certs \
  --control-plane-endpoint=k8s-cluster.computingforgeeks.com

# Containerd
sudo kubeadm init \
  --pod-network-cidr=192.168.0.0/16 \
  --cri-socket /run/containerd/containerd.sock \
  --upload-certs \
  --control-plane-endpoint=k8s-cluster.computingforgeeks.com
```

X

```
  --pod-network-cidr=192.168.0.0/16 \
  --cri-socket /var/run/docker.sock \
  --upload-certs \
  --control-plane-endpoint=k8s-cluster.computingforgeeks.com
```

Here is the output of my initialization command.

```
....
[init] Using Kubernetes version: v1.22.2
[preflight] Running pre-flight checks
        [WARNING Firewalld]: firewalld is active, please ensure ports
[6443 10250] are open or your cluster may not function correctly
[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'
[kubelet-start] Writing kubelet environment file with flags to file
"/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Writing kubelet configuration to file "/var/lib
/kubelet/config.yaml"
[kubelet-start] Starting the kubelet
[certs] Using certificateDir folder "/etc/kubernetes/pki"
[certs] Using existing ca certificate authority
[certs] Using existing apiserver certificate and key on disk
[certs] Using existing apiserver-kubelet-client certificate and key
on disk
[certs] Using existing front-proxy-ca certificate authority
[certs] Using existing front-proxy-client certificate and key on disk
```

```
[certs] Using existing etcd/peer certificate and key on disk
[certs] Using existing etcd/healthcheck-client certificate and key on disk
[certs] Using existing apiserver-etcd-client certificate and key on disk
[certs] Using the existing "sa" key
[kubeconfig] Using kubeconfig folder "/etc/kubernetes"
[kubeconfig] Using existing kubeconfig file: "/etc/kubernetes/admin.conf"
[kubeconfig] Using existing kubeconfig file: "/etc/kubernetes/kubelet.conf"
[kubeconfig] Using existing kubeconfig file: "/etc/kubernetes/controller-manager.conf"
[kubeconfig] Using existing kubeconfig file: "/etc/kubernetes/scheduler.conf"
[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"
W0611 22:34:23.276374    4726 manifests.go:225] the default kube-apiserver authorization-mode is "Node,RBAC"; using "Node,RBAC"
[control-plane] Creating static Pod manifest for "kube-scheduler"
W0611 22:34:23.278380    4726 manifests.go:225] the default kube-apiserver authorization-mode is "Node,RBAC"; using "Node,RBAC"
[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 static Pods from directory "/etc/kubernetes/manifests". This can take up to 4m0s
[apiclient] All control plane components are healthy after 8.008181 seconds
[upload-config] Storing the configuration used in ConfigMap "kubeadm-config" in the "kube-system" Namespace
[kubelet] Creating a ConfigMap "kubelet-config-1.21" in namespace kube-system with the configuration for the kubelets in the cluster
[upload-certs] Skipping phase. Please see --upload-certs
[mark-control-plane] Marking the node k8s-master01.computingforgeeks.com as control-plane by adding the label "node-role.kubernetes.io/master=''"
[mark-control-plane] Marking the node k8s-master01.computingforgeeks.com as control-plane by adding the taints [node-role.kubernetes.io/master:NoSchedule]
[bootstrap-token] Using token: zoy8cq.6v349sx9ass8dzyj
```

```
tokens to get nodes
[bootstrap-token] configured RBAC rules to allow Node Bootstrap
tokens to post CSRs in order for nodes to get long term certificate
credentials
[bootstrap-token] configured RBAC rules to allow the csrapprover
controller automatically approve CSRs from a Node Bootstrap Token
[bootstrap-token] configured RBAC rules to allow certificate rotation
for all node client certificates in the cluster
[bootstrap-token] Creating the "cluster-info" ConfigMap in the "kube-
public" namespace
[kubelet-finalize] Updating "/etc/kubernetes/kubelet.conf" to point
to a rotatable kubelet client certificate and key
[addons] Applied essential addon: CoreDNS
[addons] Applied essential addon: kube-proxy

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/

You can now join any number of control-plane nodes by copying
certificate authorities
and service account keys on each node and then running the following
as root:

  kubeadm join k8s-cluster.computingforgeeks.com:6443 --token
sr4l2l.2kvot0pfalh5o4ik \
    --discovery-token-ca-cert-hash
sha256:c692fb047e15883b575bd6710779dc2c5af8073f7cab460abd181fd3ddb29a
18 \
    --control-plane

Then you can join any number of worker nodes by running the following
on each as root:
    --discovery-token-ca-cert-hash
sha256:c692fb047e15883b575bd6710779dc2c5af8073f7cab460abd181fd3ddb29a
18
```

Configure kubectl using commands in the output:

```
mkdir -p $HOME/.kube
sudo cp -f /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

Check cluster status:

```
$ kubectl cluster-info
Kubernetes master is running at https://k8s-
cluster.computingforgeeks.com:6443
KubeDNS is running at https://k8s-cluster.computingforgeeks.com:6443
/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-
info dump'.
```

Additional Master nodes can be added using the command in installation output:

```
kubeadm join k8s-cluster.computingforgeeks.com:6443 --token
sr4l2l.2kvot0pfalh5o4ik \
    --discovery-token-ca-cert-hash
sha256:c692fb047e15883b575bd6710779dc2c5af8073f7cab460abd181fd3ddb29a
18 \
    --control-plane
```

```
kubectl create -f https://docs.projectcalico.org/manifests/tigera-operator.yaml
kubectl create -f https://docs.projectcalico.org/manifests/custom-resources.yaml
```

You should see the following output.

```
customresourcedefinition.apiextensions.k8s.io/bgpconfigurations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/bgppeers.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/blockaffinities.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/clusterinformations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/felixconfigurations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/globalnetworkpolicies.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/globalnetworksets.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/hostendpoints.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamblocks.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamconfigs.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamhandles.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ippools.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/kubecontrollersconfigurations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/networkpolicies.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/apiservers.operator.tig
era.io created
customresourcedefinition.apiextensions.k8s.io/imagesets.operator.tige
ra.io created
customresourcedefinition.apiextensions.k8s.io/installations.operator.
tigera.io created
customresourcedefinition.apiextensions.k8s.io/tigerastatuses.operator
.tigera.io created
namespace/tigera-operator created
Warning: policy/v1beta1 PodSecurityPolicy is deprecated in v1.21+,
unavailable in v1.25+
podsecuritypolicy.policy/tigera-operator created
serviceaccount/tigera-operator created
clusterrole.rbac.authorization.k8s.io/tigera-operator created
clusterrolebinding.rbac.authorization.k8s.io/tigera-operator created
deployment.apps/tigera-operator created
.....
installation.operator.tigera.io/default created
apiserver.operator.tigera.io/default created
```

Confirm that all of the pods are running:

```
$ watch kubectl get pods --all-namespaces
NAMESPACE     NAME
READY   STATUS    RESTARTS   AGE
kube-system   calico-kube-controllers-76d4774d89-nfqrr
1/1     Running   0          2m52s
kube-system   calico-node-kpprr
1/1     Running   0          2m52s
kube-system   coredns-66bff467f8-9bxgm
1/1     Running   0          7m43s
kube-system   coredns-66bff467f8-jgwln
1/1     Running   0          7m43s
kube-system   etcd-k8s-master01.computingforgeeks.com
1/1     Running   0          7m58s
```

```
kube-system   kube-controller-manager-k8s-
master01.computingforgeeks.com   1/1     Running   0          7m58s
kube-system   kube-proxy-bt7ff
1/1     Running   0          7m43s
kube-system   kube-scheduler-k8s-master01.computingforgeeks.com
1/1     Running   0          7m58s
```

Confirm master node is ready:

```
# CRI-O
$ kubectl get nodes -o wide
NAME     STATUS   ROLES                  AGE   VERSION   INTERNAL-IP
EXTERNAL-IP   OS-IMAGE             KERNEL-VERSION     CONTAINER-
RUNTIME
ubuntu   Ready    control-plane,master   38s   v1.22.2
143.198.114.46   <none>        Ubuntu 20.04.3 LTS   5.4.0-88-generic
cri-o://1.22.0

# Containerd
$ kubectl get nodes -o wide
NAME     STATUS   ROLES                  AGE   VERSION   INTERNAL-IP
EXTERNAL-IP   OS-IMAGE             KERNEL-VERSION     CONTAINER-
RUNTIME
ubuntu   Ready    control-plane,master   15m   v1.22.2
143.198.114.46   <none>        Ubuntu 20.04.3 LTS   5.4.0-88-generic
containerd://1.4.11

# Docker
$ kubectl get nodes -o wide
NAME           STATUS   ROLES    AGE   VERSION   INTERNAL-IP
EXTERNAL-IP   OS-IMAGE           KERNEL-VERSION     CONTAINER-RUNTIME
k8s-master01   Ready    master   64m   v1.22.2   135.181.28.113
<none>        Ubuntu 20.04 LTS   5.4.0-37-generic   docker://20.10.8
```

Step 7: Add worker nodes

With the control plane ready you can add worker nodes to the cluster for running scheduled workloads.

If endpoint address is not in DNS, add record to */etc/hosts*.

```
$ sudo vim /etc/hosts
172.29.20.5 k8s-cluster.computingforgeeks.com
```

The join command that was given is used to add a worker node to the cluster.

```
kubeadm join k8s-cluster.computingforgeeks.com:6443 \
  --token sr4l2l.2kvot0pfalh5o4ik \
  --discovery-token-ca-cert-hash
sha256:c692fb047e15883b575bd6710779dc2c5af8073f7cab460abd181fd3ddb29a
18
```

Output:

```
[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-1.21" 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] Starting the kubelet
[kubelet-start] Waiting for the kubelet to perform the TLS
Bootstrap...

This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response
```

Run below command on the control-plane to see if the node joined the cluster.

```
$ kubectl get nodes
NAME                               STATUS   ROLES    AGE   VERSION
k8s-master01.computingforgeeks.com   Ready    master   10m   v1.22.2
k8s-worker01.computingforgeeks.com   Ready    <none>   50s   v1.22.2
k8s-worker02.computingforgeeks.com   Ready    <none>   12s   v1.22.2

$ kubectl get nodes -o wide
```

If the join token is expired, refer to our guide on how to join worker nodes.

- Join new Kubernetes Worker Node to an existing Cluster

Step 8: Deploy application on cluster

If you only have a single node cluster, check our guide on how to run container pods on master nodes:

- Scheduling Pods on Kubernetes Control plane (Master) Nodes

We need to validate that our cluster is working by deploying an application.

```
kubectl apply -f https://k8s.io/examples/pods/commands.yaml
```

Check to see if pod started

```
$ kubectl get pods
NAME           READY   STATUS      RESTARTS   AGE
command-demo   0/1     Completed   0          16s
```

Step 9: Install Kubernetes Dashboard (Optional)

Kubernetes dashboard can be used to deploy containerized applications to a Kubernetes cluster, troubleshoot your containerized application, and manage the cluster resources.

Refer to our guide for installation: How To Install Kubernetes Dashboard with NodePort

Step 10: Install Metrics Server (For checking Pods and Nodes resource usage)

Metrics Server is a cluster-wide aggregator of resource usage data. It collects metrics from the *Summary API*, exposed by **Kubelet** on each node. Use our guide below to deploy it:

- How To Deploy Metrics Server to Kubernetes Cluster

Step 11: Deploy Prometheus / Grafana Monitoring

Prometheus is a full fledged solution that enables you to access advanced metrics capabilities in a Kubernetes cluster. Grafana is used for analytics and interactive visualization of metrics that’s collected and stored in Prometheus database. We have a complete guide on how to setup complete monitoring stack on Kubernetes Cluster:

- Setup Prometheus and Grafana on Kubernetes using prometheus-operator

Step 12: Persistent Storage Configuration ideas (Optional)

If you’re also looking for a Persistent storage solution for your Kubernetes, checkout:

- How To Deploy Rook Ceph Storage on Kubernetes Cluster
- Ceph Persistent Storage for Kubernetes with Cephfs
- Persistent Storage for Kubernetes with Ceph RBD
- How To Configure Kubernetes Dynamic Volume Provisioning With Heketi & GlusterFS

If Nginx is your preferred Ingress controller for Kubernetes workloads, you can use our guide for the installation process:

- Deploy Nginx Ingress Controller on Kubernetes using Helm Chart

More guides:

- Using Horizontal Pod Autoscaler on Kubernetes Cluster
- Install Kubernetes Metrics Server

Similar Kubernetes deployment guides:

- Install Kubernetes Cluster on CentOS 7 with kubeadm
- Install Production Kubernetes Cluster with Rancher RKE
- How To Deploy Lightweight Kubernetes Cluster in 5 minutes with K3s
- Deploy Production Ready Kubernetes Cluster with Ansible & Kubespray

Josphat Mutai

https://computingforgeeks.com/

Founder of Computingforgeeks. Expertise in Virtualization, Cloud, Linux/UNIX Administration, Automation,Storage Systems, Containers, Server Clustering e.t.c.

in