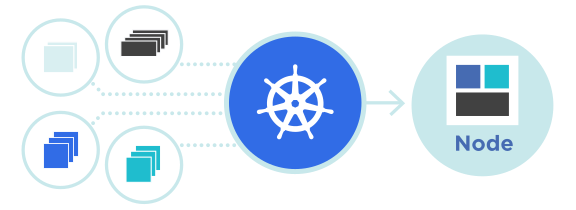
**Overview**



Kubernetes Cluster

Kubernetes, also known as K8s, is an open source system for automating deployment, scaling, and management of containerized applications. A production-quality Kubernetes cluster requires planning and preparation. If your Kubernetes cluster is to run critical workloads, it must be configured to be resilient.

There are two categories of nodes in a Kubernetes cluster, namely:

* **Master Node:** This handles the control API calls for the pods, replications controllers, services, nodes and other components of a Kubernetes cluster.
* **Worker Node:** Provides the run-time environments for the containers. A set of container pods can span multiple nodes. In your Master / Worker nodes.

**Before you begin**

* A compatible Linux host. The Kubernetes project provides generic instructions for Linux distributions based on Debian and Red Hat, and those distributions without a package manager.
* 2 GB or more of RAM per machine (any less will leave little room for your apps).
* 2 CPUs or more.
* Full network connectivity between all machines in the cluster (public or private network is fine).
* Unique hostname, MAC address, and product\_uuid for every node. See here for more details.
* Certain ports are open on your machines. See here for more details.
* Swap configuration.

**Lab Setup**

The basic lab setup described in this guide consists of three servers:

* one Master Node
* two Worker Nodes for running containerized workloads. Here I have used 3 virtual machines. You can add additional nodes to meet your desired environment load requirements. For high availability (HA), three control plane nodes are required along with a control plane API endpoint.

The nodes in this lab setup are as follows:

| Server Role | Host Name | Configuration | IP Address |  
| Master | master | 8GB Ram, 4vcpus | 192.168.0.193 |  
| Worker | worker1 | 4GB Ram, 2vcpus | 192.168.0.143 |  
| Worker | worker2 | 4GB Ram, 2vcpus | 192.168.0.103 |

**Common Installation (Master and Worker nodes)**

This section will describe the common installation for master and worker nodes.

**Install Vim (Optional)**

It’s an editor and will be used to edit config files from terminal. There are many other ways but it’s my personal preferences to edit files from terminal.

sudo apt-get install vim

**Swap configuration**

The default behavior of a kubelet was to fail to start if swap memory was detected on a node. You MUST disable swap if the kubelet is not properly configured to use swap. Write the following command to swap off.

sudo swapoff -a

It will disable swapping temporarily. To make this change persistent across reboots, make sure swap is disabled in config files like /etc/fstab, systemd.swap, depending how it was configured on your system.

**Verify the MAC address and product\_uuid are unique for every node**

You can get the MAC address of the network interfaces using the command

ip link

or

ifconfig -a

The product\_uuid can be checked by using the command

sudo cat /sys/class/dmi/id/product\_uuid

It is very likely that hardware devices will have unique addresses, although some virtual machines may have identical values. Kubernetes uses these values to uniquely identify the nodes in the cluster. If these values are not unique to each node, the installation process may fail. Also need to check required ports. These required ports need to be open in order for Kubernetes components to communicate with each other. You can use tools like netcat to check if a port is open. For example:

nc 127.0.0.1 6443 -v

The pod network plugin you use may also require certain ports to be open.

**Enable kernel modules and configure sysctl**

Enable kernel modules with the following commands

sudo modprobe overlay  
sudo modprobe br\_netfilter

Add some settings to sysctl with the following command

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

Terminal window should looks like the following snippet

master@master:~$ 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

And the ouput will be like

net.bridge.bridge-nf-call-ip6tables = 1  
net.bridge.bridge-nf-call-iptables = 1  
net.ipv4.ip\_forward = 1

Configure persistent loading of modules with the following command

sudo tee /etc/modules-load.d/k8s.conf <<EOF  
overlay  
br\_netfilter  
EOF

Reload sysctl with the following command

sudo sysctl --system

**Install container runtime (Master and Worker Nodes)**

To run containers in Pods, Kubernetes uses a container runtime. By default, Kubernetes uses the Container Runtime Interface (CRI) to interface with your chosen container runtime. If you don’t specify a runtime, kubeadm automatically tries to detect an installed container runtime by scanning through a list of known endpoints. If multiple or no container runtimes are detected kubeadm will throw an error and will request that you specify which one you want to use. For my lab I have used containerd as CRI.

**Containerd**

This section outlines the necessary steps to use containerd as CRI runtime.To install containerd on your system, follow the instructions on getting started with containerd. Return to this step once you’ve created a valid config.toml configuration file. You can find this file under the path /etc/containerd/config.toml.

On Linux the default CRI socket for containerd is /run/containerd/containerd.sock. Follow the installation process.

* Set up Docker’s apt repository.

# Add Docker's official GPG key:  
sudo apt-get update  
sudo apt-get install apt-transport-https ca-certificates curl gpg  
sudo install -m 0755 -d /etc/apt/keyrings  
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc  
sudo chmod a+r /etc/apt/keyrings/docker.asc  
   
# Add the repository to Apt sources:  
echo \  
 "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/ubuntu \  
 $(. /etc/os-release && echo "$VERSION\_CODENAME") stable" | \  
 sudo tee /etc/apt/sources.list.d/docker.list > /dev/null  
sudo apt-get update

* Install containerd

sudo apt-get install containerd.io

* Run the following commands to configure containerd

sudo mkdir -p /etc/containerd  
sudo containerd config default | sudo tee /etc/containerd/config.toml

* Verify that the configuration is correct, particularly the plugins.”io.containerd.grpc.v1.cri” section. Ensure the sandbox\_image is properly set:

[plugins."io.containerd.grpc.v1.cri".containerd]  
 snapshotter = "overlayfs"  
[plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc.options]  
 SystemdCgroup = true

* To edit configuration write the following command and then edit the above lines if required.

sudo vi /etc/containerd/config.toml

* Restart containerd

sudo systemctl restart containerd  
sudo systemctl enable containerd  
systemctl status containerd

* Enable IP Forwarding

Enable IP forwarding temporarily and permanently:

a. Temporary option

echo 1 | sudo tee /proc/sys/net/ipv4/ip\_forward

b. Permanently Add the following line to /etc/sysctl.conf:

sudo sh -c "echo 'net.ipv4.ip\_forward = 1' >> /etc/sysctl.conf"

Apply the changes:

sudo sysctl -p

* Validate Setup After making these changes, validate the container runtime and IP forwarding:

a. Validate Containerd Use crictl to check the containerd status:

sudo crictl info

This should return details about the container runtime if it is correctly configured.

b. Validate IP Forwarding Verify that IP forwarding is enabled:

cat /proc/sys/net/ipv4/ip\_forward

**Install Kubeadm, Kubectl and Kubelet**

These instructions are for Kubernetes v1.30.

* Download the public signing key for the Kubernetes package repositories. If the directory /etc/apt/keyrings does not exist, it should be created before the curl command, read the note below.

sudo mkdir -p -m 755 /etc/apt/keyrings

Then write the following curl command.

curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.30/deb/Release.key | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg

* Add the appropriate Kubernetes apt repository. Note that This will overwrite any existing configuration in /etc/apt/sources.list.d/kubernetes.list

echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.30/deb/ /' | sudo tee /etc/apt/sources.list.d/kubernetes.list

1. Update the apt package index, install kubelet, kubeadm and kubectl, and pin their version:

sudo apt-get update  
sudo apt-get install kubelet kubeadm kubectl  
sudo apt-mark hold kubelet kubeadm kubectl

1. (Optional) Enable the kubelet service before running kubeadm:

sudo systemctl enable --now kubelet

**Creating a cluster with kubeadm**

**Network setup**

kubeadm similarly to other Kubernetes components tries to find a usable IP on the network interfaces associated with a default gateway on a host. To find out what this IP is on a Linux host you can use:

ip route show # Look for a line starting with "default via"

[IMPORTANT] Before go to next steps the following criteria must be satisfied.

* Worker nodes must be ready!
* Need to make sure network connectivity between cluster nodes!
* kubectl, kubelet and kubeadm must be installed and enabled

**Initialize control-plane node (Master Node only)**

To initialize the control-plane node run:

kubeadm init <args>

While --apiserver-advertise-address can be used to set the advertise address for this particular control-plane node's API server, --control-plane-endpoint can be used to set the shared endpoint for all control-plane nodes. --control-plane-endpoint allows both IP addresses and DNS names that can map to IP addresses. Please contact your network administrator to evaluate possible solutions with respect to such mapping. Here is an example mapping:

192.168.0.193 cluster-endpoint

Where 192.168.0.193 is the IP address of this node and cluster-endpoint is a custom DNS name that maps to this IP. This will allow you to pass --control-plane-endpoint=cluster-endpoint to kubeadm init and pass the same DNS name to kubeadm join. Later you can modify cluster-endpoint to point to the address of your load-balancer in an high availability scenario. Turning a single control plane cluster created without --control-plane-endpoint into a highly available cluster is not supported by kubeadm. So write the following command.

sudo kubeadm init --pod-network-cidr=192.168.0.0/16 --cri-socket=/var/run/containerd/containerd.sock --v=5

It’ll produce output like the following snippet.

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  
   
Alternatively, if you are the root user, you can run:  
   
 export KUBECONFIG=/etc/kubernetes/admin.conf  
   
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 root:  
   
kubeadm join 192.168.0.193:6443 --token 1rc7cy.ln076tz43nj0ghjr \  
 --discovery-token-ca-cert-hash sha256:a6c7f7d99f35fec5f274d50fd1120bebd7c518b4eaa8174ebcba87fbd33c7677

Note down the above output carefully and follow the instructions carefully for next steps. First write the following commands

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

Then run the following command:

export KUBECONFIG=/etc/kubernetes/admin.conf

**Permissions of admin.config (Master Node only)**

Write the following command to set permissions of admin.config

sudo chmod -R 755 /etc/kubernetes/admin.conf

[!IMPORTANT] To reset kubeadm config need to write the following command:

sudo kubeadm reset -f

**Install Pod network add-on (Only on Master Node)**

This section contains important information about networking setup and deployment order. Read all of this advice carefully before proceeding. You must deploy a Container Network Interface (CNI) based Pod network add-on so that your Pods can communicate with each other. Cluster DNS (CoreDNS) will not start up before a network is installed. Take care that your Pod network must not overlap with any of the host networks: you are likely to see problems if there is any overlap. (If you find a collision between your network plugin’s preferred Pod network and some of your host networks, you should think of a suitable CIDR block to use instead, then use that during kubeadm init with — pod-network-cidr and as a replacement in your network plugin’s YAML).

**Install calico**

Check the Kubernetes API server’s address and port configuration. Typically, the API server runs on port 6443. Make sure the kube-apiserver is correctly configured to bind to the appropriate address. So install calico on kubernetes cluster by following the [link](https://docs.tigera.io/calico/latest/getting-started/kubernetes/quickstart)

* Install the Tigera Calico operator and custom resource definitions.

kubectl create -f https://raw.githubusercontent.com/projectcalico/calico/v3.28.0/manifests/tigera-operator.yaml

* Install Calico by creating the necessary custom resource.

kubectl create -f https://raw.githubusercontent.com/projectcalico/calico/v3.28.0/manifests/custom-resources.yaml

* Confirm that all of the pods are running with the following command.

watch kubectl get pods -n calico-system

It should produce output as the following snippet.

master@master:~$ watch kubectl get pods -n calico-system  
   
Every 2.0s: kubectl get pods -n calico-system master: Wed May 22 15:55:46 2024  
   
NAME READY STATUS RESTARTS AGE  
calico-kube-controllers-6965d8c554-ds7lp 1/1 Running 0 25s  
calico-node-jlqfs 0/1 Running 0 25s  
calico-node-vgp2q 0/1 Running 0 25s  
calico-typha-7dc95cc6b6-xp6hn 1/1 Running 0 25s  
csi-node-driver-qkd26 2/2 Running 0 25s  
csi-node-driver-twg5q 2/2 Running 0 25s

Wait until all status are running. So final output should be the following snippet.

master@master:~$ watch kubectl get pods -n calico-system  
   
Every 2.0s: kubectl get pods -n calico-system master: Wed May 22 15:57:31 2024  
   
NAME READY STATUS RESTARTS AGE  
calico-kube-controllers-6965d8c554-ds7lp 1/1 Running 0 2m10s  
calico-node-jlqfs 1/1 Running 0 2m10s  
calico-node-vgp2q 1/1 Running 0 2m10s  
calico-typha-7dc95cc6b6-xp6hn 1/1 Running 0 2m10s  
csi-node-driver-qkd26 2/2 Running 0 2m10s  
csi-node-driver-twg5q 2/2 Running 0 2m10

Check the Kubernetes API server’s address and port configuration. Typically, the API server runs on port 6443. Make sure the kube-apiserver is correctly configured to bind to the appropriate address.

**Join with kubernetes cluster (Worker Node only)**

Write the following similar command to join with master node. Note: this command comes from above output from master.

sudo kubeadm join 192.168.0.193:6443 --token 1rc7cy.ln076tz43nj0ghjr \  
 --discovery-token-ca-cert-hash sha256:a6c7f7d99f35fec5f274d50fd1120bebd7c518b4eaa8174ebcba87fbd33c7677

**Check cluster status**

Write the following command to check cluster status

kubectl cluster-info

It should produce the output like the following snippet

master@master:~$ kubectl cluster-info  
Kubernetes control plane is running at https://192.168.0.193:6443  
CoreDNS is running at https://192.168.0.193:6443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy  
   
To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.

Write to following command to check node status with the following kubectl command

kubectl get nodes -o wide

The output should be the following snippet.

master@master:~$ kubectl get nodes -o wide  
NAME STATUS ROLES AGE VERSION INTERNAL-IP EXTERNAL-IP OS-IMAGE KERNEL-VERSION CONTAINER-RUNTIME  
master Ready control-plane 6m55s v1.30.1 192.168.0.193 <none> Ubuntu 24.04 LTS 6.8.0-31-generic containerd://1.6.31  
worker1 Ready <none> 4m31s v1.30.1 192.168.0.143 <none> Ubuntu 24.04 LTS 6.8.0-31-generic containerd://1.6.31  
master@master:~$

**Deploy first hello pod to kubernetes cluster**

Write the following command to deply hello pod at kubernetes cluster with the following command

kubectl apply -f hello-pod.yaml

Please check the yaml file to deploy hello pod.

apiVersion: v1  
kind: Pod  
metadata:  
 name: hello-pod  
spec:  
 containers:  
 - name: hello  
 image: busybox  
 command: ['sh', '-c', 'echo Hello from Hello Pod! && sleep 3600']

It should produce the following output for success case.

master@master:~$ kubectl apply -f hello-pod.yaml   
pod/hello-pod created

Write the following command to check all pods.

kubectl get pods -o wide

You should get the following output for success case.

master@master:~$ kubectl get pods -o wide  
NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES  
hello-pod 1/1 Running 0 72s 192.168.235.132 worker1 <none> <none>  
master@master:~$

**Kubernetes Dashboard (Master Node Only)**

Dashboard is a web-based Kubernetes user interface. You can use Dashboard to deploy containerized applications to a Kubernetes cluster, troubleshoot your containerized application, and manage the cluster resources. You can use Dashboard to get an overview of applications running on your cluster, as well as for creating or modifying individual Kubernetes resources (such as Deployments, Jobs, DaemonSets, etc). For example, you can scale a Deployment, initiate a rolling update, restart a pod or deploy new applications using a deploy wizard.

Dashboard also provides information on the state of Kubernetes resources in your cluster and on any errors that may have occurred.

To install Kubernetes Dashboard first we need to install helm as per official documentation recommendation. Please check official documentation from [here](https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/) and from their [github](https://github.com/kubernetes/dashboard/tree/master" \t "_blank) repository.

**Install helm**

To install helm follow official instructions from [here](https://helm.sh/docs/intro/install/)

For ubuntu 24.04 please write the following commands one by one to install helm.

curl https://baltocdn.com/helm/signing.asc | gpg --dearmor | sudo tee /usr/share/keyrings/helm.gpg > /dev/null  
sudo apt-get install apt-transport-https --yes  
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/helm.gpg] https://baltocdn.com/helm/stable/debian/ all main" | sudo tee /etc/apt/sources.list.d/helm-stable-debian.list  
sudo apt-get update  
sudo apt-get install helm

**Install Kubernetes Dashboard**

Write the following commands one by one to install Kubernetes Dashboard on cluster

helm repo add kubernetes-dashboard https://kubernetes.github.io/dashboard/  
helm upgrade --install kubernetes-dashboard kubernetes-dashboard/kubernetes-dashboard --create-namespace --namespace kubernetes-dashboard

The following output should be produced

master@master:~$ helm upgrade --install kubernetes-dashboard kubernetes-dashboard/kubernetes-dashboard --create-namespace --namespace kubernetes-dashboard  
WARNING: Kubernetes configuration file is group-readable. This is insecure. Location: /etc/kubernetes/admin.conf  
WARNING: Kubernetes configuration file is world-readable. This is insecure. Location: /etc/kubernetes/admin.conf  
Release "kubernetes-dashboard" does not exist. Installing it now.  
NAME: kubernetes-dashboard  
LAST DEPLOYED: Tue May 21 18:38:36 2024  
NAMESPACE: kubernetes-dashboard  
STATUS: deployed  
REVISION: 1  
TEST SUITE: None  
NOTES:  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*\*\* PLEASE BE PATIENT: Kubernetes Dashboard may need a few minutes to get up and become ready \*\*\*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
   
Congratulations! You have just installed Kubernetes Dashboard in your cluster.  
   
To access Dashboard run:  
 kubectl -n kubernetes-dashboard port-forward svc/kubernetes-dashboard-kong-proxy 8443:443  
   
NOTE: In case port-forward command does not work, make sure that kong service name is correct.  
 Check the services in Kubernetes Dashboard namespace using:  
 kubectl -n kubernetes-dashboard get svc  
   
Dashboard will be available at:  
 https://localhost:8443

Now write the following command to check status

kubectl -n kubernetes-dashboard get svc -o wide

**Install screen (optional)**

To run any service in background we can use screen. To install it write the following command.

sudo apt-get install screen

To attach any service to screen write the following command.

screen -S <screen-name>

An example:

screen -S kubernetes-dashboard

Then just write commands whatever need to run services. To detach from screen just Ctrl+A and Ctrl+D to detach from screen.

To re-attach screen again just write the following command:

screen -dr <screen-name>

Example:

screen -dr kubernetes-dashboard

**Port forwarding for kubernetes-dashboard**

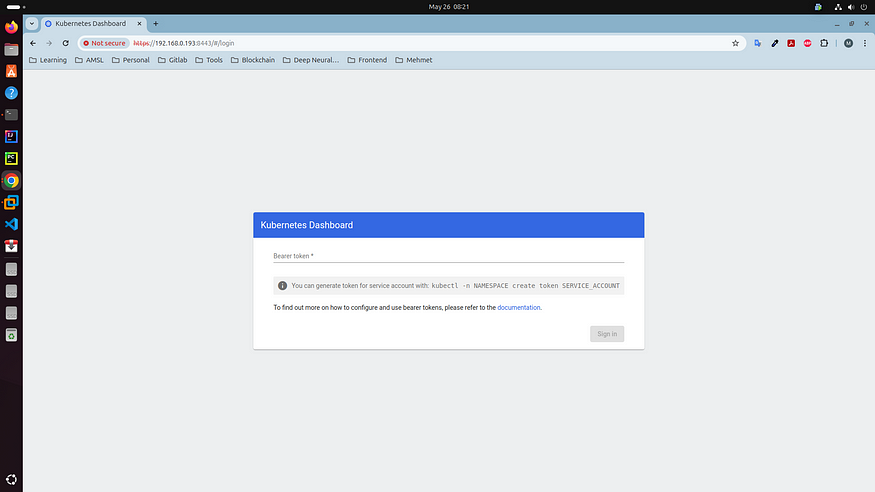
To access Dashboard run:

kubectl -n kubernetes-dashboard port-forward --address 0.0.0.0 svc/kubernetes-dashboard-kong-proxy 8443:443

Now access dashboard from the following url, for my case ip address is 192.168.0.193

https://192.168.0.193:8443

Open url on web browser and after redirection to login screen you should show the screen.



**Creating sample user**

In this guide, we will find out how to create a new user using the Service Account mechanism of Kubernetes, grant this user admin permissions and login to Dashboard using a bearer token tied to this user.

For each of the following snippets for ServiceAccount and ClusterRoleBinding, you should copy them to new manifest files like dashboard-adminuser.yaml and use the following command

kubectl apply -f dashboard-adminuser.yaml

to create them.

**dashboard-adminuser.yaml**

apiVersion: v1  
kind: ServiceAccount  
metadata:  
 name: admin-user  
 namespace: kubernetes-dashboard

**Creating a ClusterRoleBinding**

In most cases after provisioning the cluster using kops, kubeadm or any other popular tool, the ClusterRole cluster-admin already exists in the cluster. We can use it and create only a ClusterRoleBinding for our ServiceAccount. If it does not exist then you need to create this role first and grant required privileges manually. Write the following command.

kubectl apply -f cluster-role.yaml

**cluster-role.yaml**

apiVersion: rbac.authorization.k8s.io/v1  
kind: ClusterRoleBinding  
metadata:  
 name: admin-user  
roleRef:  
 apiGroup: rbac.authorization.k8s.io  
 kind: ClusterRole  
 name: cluster-admin  
subjects:  
- kind: ServiceAccount  
 name: admin-user  
 namespace: kubernetes-dashboard

**Getting a Bearer Token for ServiceAccount**

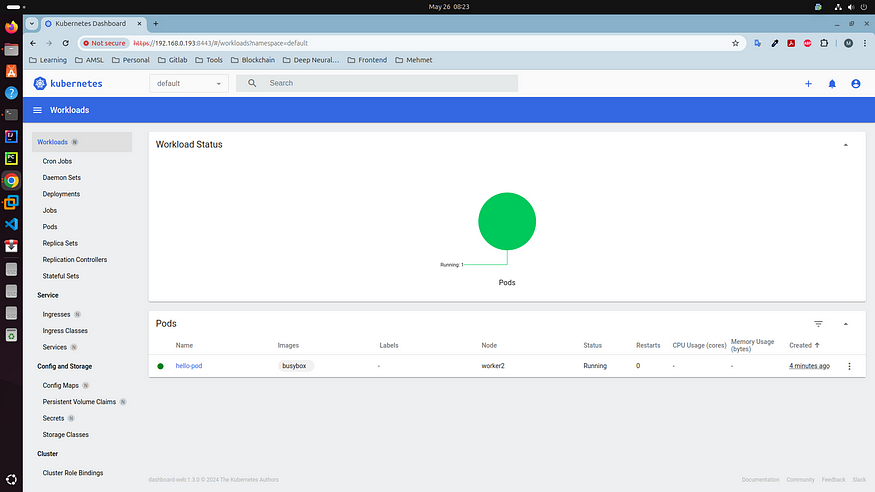
Now we need to find the token we can use to log in. Execute the following command:

kubectl -n kubernetes-dashboard create token admin-user

It should print something like:

master@master:~$ kubectl -n kubernetes-dashboard create token admin-user  
eyJhbGciOiJSUzI1NiIsImtpZCI6IjJYaXFaNVpPMXlaSUNsTGhkLXVtLXRtb2dwWTAtWkNuZlNqMzNkMU1sVVEifQ..sP-Qcrh37iZ4Xc9K04SaGyBrfFfzoXFZJwJgh4YU7zMoKzbE7rafXajD18UtNKrwFZA30Bp157uxDy6OqmVKL4YAk5P1TyVNFSK4LfMtfx9XHUFhxGvItnE7MDPyc1D4u1Ib5-K-x3ZwYnTToX3IgY8vwh5mVs32bYBfgAKg4KqbHt2T0c00Ow4D79O-GVlp4Opc92auRYzLG-Ks7x-kS24ZVePJgUuQ7aowP2e5IPnjK1hIrZE44TeWyuCdnGn15SglybuQRzWVScfwcM3Yo0y09nw3CwvCmShONOOzHg857Tn6amwXWvHDKn4jwyDZnQJg22UXfdPFHIlEuptA  
master@master:~$

Now copy the token and paste it into the Enter token field on the login screen. Click the Sign in button and that's it. You are now logged in as an admin. You should show the screen like the following screenshot.



For more details please visit [here](https://github.com/kubernetes/dashboard/blob/master/docs/user/access-control/creating-sample-user.md)

**References**

* <https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/>
* <https://docs.tigera.io/calico/latest/getting-started/kubernetes/quickstart>
* <https://helm.sh/docs/intro/install/>
* <https://github.com/kubernetes/dashboard>
* <https://github.com/kubernetes/dashboard/blob/master/docs/user/access-control/creating-sample-user.md>
* <https://computingforgeeks.com/install-kubernetes-cluster-ubuntu-jammy/>