1. **What is Kubernetes?**
   * Define Kubernetes and explain its purpose in the world of container orchestration.
   * Highlight the problems Kubernetes solves (e.g., scaling, fault tolerance, declarative management).

Ans: - Kubernetes is an open-source container orchestration platform. It is used to automate the management, scaling, deployment and orchestration of containers.

Purpose of Kubernetes in container orchestration:

* Easy scaling
* Manages resources efficiently
* Self-healing (automatically recovers from failures by restarting containers, load balancing, …)
* Automates deployment

Problems Kubernetes solves:

* Scaling manually based on varying traffic
* Recovering services, pods when crashed manually is inefficient
* Managing configurations manually (k8s declarative approach solves this problem)
* Service discovery and load balancing help in finding and routing traffic to right containers, even when there are dynamic IPs involved

List and briefly explain features like **scalability, self-healing, service discovery, storage orchestration, automated rollouts/rollbacks**, etc.

Ans: -

**Features of Kubernetes:**

Scalability:

* Kubernetes consist of Horizontal Pod Autoscaler (HPA) which adjusts the number of pods based on resource usage
* Cluster auto scaling adds or removes nodes based on the workload

Self-healing:

* Restarts containers
* Reschedules pods if crashed or if a node fails
* Ensures cluster state matches the defined desired state

Service discovery:

* Services provide a static IP address and DNS names for pods

Storage Orchestration:

* Integrates with cloud providers and on-premises storage systems (e.g., AWS EBS, GCP Persistent Disks, NFS).
* Supports multiple storage types: persistent volumes, ephemeral storage, block storage, and shared file systems.

Automated Rollout/Rollbacks:

* Kubernetes automates application deployment updates and can revert back to previous version in case of any issues.
* Rolling upgrades gradually replaces pods ensuring zero downtime
* Rollbacks revert to a previous version of an application if update fails

**Key Components of Kubernetes**

1. **Master Node (Control Plane):**
   * **API Server:** Gateway/ entry point to etcd.
   * **Controller Manager:** Ensures the cluster state matches the desired state.
   * **Scheduler:** Assigns workloads to appropriate nodes.
   * **etcd:** Distributed key-value store for cluster state.
2. **Worker Nodes:**
   * **Kubelet:** Ensures containers are running in a pod.
   * **Container Runtime:** Runs containerized applications (e.g., Docker, containerd).
   * **Kube-Proxy:** Handles network routing and load balancing.

**Discuss where configuration files for these components are located and why they’re essential for debugging.**

Configuration Files and Logs:

1. **Kubelet:**

* /var/lib/kubelet/config.yaml (on each node)
* /var/log/ kubelet.log
  + **Purpose:** Manages the Kubelet, which ensures pods are running on the node.
* Issues with node-level operations (e.g., pod creation, resource management) often stem from incorrect Kubelet configurations.

1. **Kube-Proxy:**

* /var/lib/kube-proxy/config.conf (on each node)
* /var/log/ kube-proxy.log
  + **Purpose:** Manages networking rules for services and pod communication.
* Misconfigurations can lead to network connectivity issues or failed service communication.

1. **API Server:**

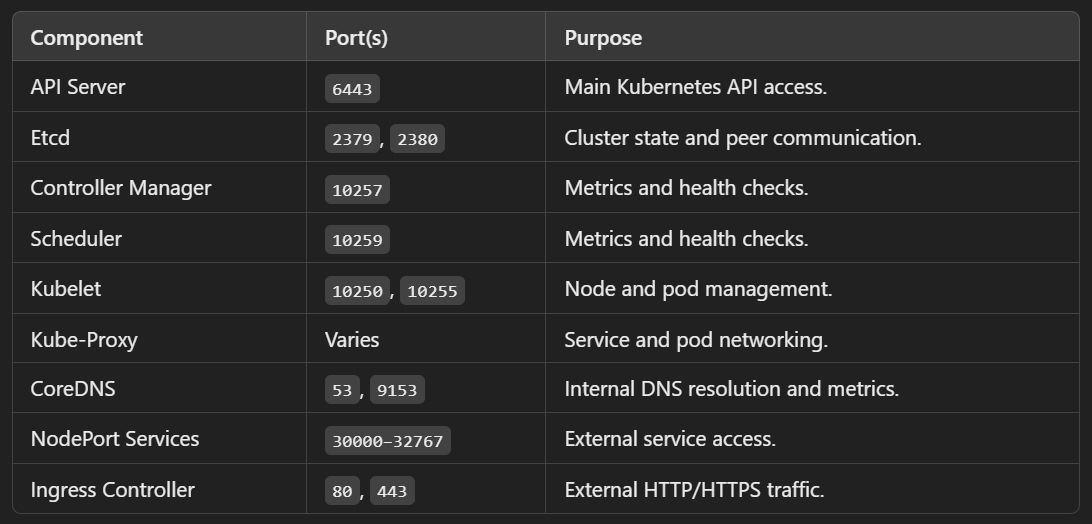
* /etc/kubernetes/manifests/kube-apiserver.yaml (on the control plane node)
* /var/log/ kube-apiserver.log
  + **Purpose:** The API Server handles all Kubernetes API requests.
* Errors here may result in cluster-wide issues, such as an inability to deploy workloads.

1. **Controller Manager:**

* /etc/kubernetes/manifests/kube-controller-manager.yaml
* /var/log/ kube-controller-manager.log
  + **Purpose:** Ensures the desired state of the cluster (e.g., replicas, jobs).
* Used for debugging issues like failing rollouts or unmaintained replica counts.

1. **Scheduler:**

* /etc/kubernetes/manifests/kube-scheduler.yaml
* /var/log/kube-scheduler.log
  + **Purpose:** Schedules pods to nodes based on resource availability and constraints.
* Scheduler misconfigurations or issues in the configuration file.



Set up Kubernetes cluster using kubeadm:

Reference:

<https://github.com/Sagar2366/LearnWithSagar/blob/main/CKA/6_kubeadm_k8s_cluster_setup.md>

Step-1:(bastion)

Create bastion node(ec2 instance):

Select keypair (create if there is not)

T2.micro

ubuntu

Network settings – edit – select created vpc – create SG (“bastion”): inbound security rules -sourcetype: my ip

Auto assign public IP : enable

no of instances(1)

Launch instance

Step-2:(control plane)

Create controlplane node (ec2 instance):

Select keypair (create if there is not)

T2.medium

ubuntu

Network settings – edit – select created vpc – create SG (“control plane”) - inbound security rules -sourcetype: custom : select “bastion”

Auto assign public IP : enable

no of instances(1)

Launch instance

Step-3:(worker nodes)

Worker nodes(ec2 instances):

Ubuntu

T2.micro

Select keypair (create if there is not)

Select created vpc

Select private subnet

Auto assign public IP : enable

Create SG(“workernode”) : inbound security rules -sourcetype: custom : select “bastion”

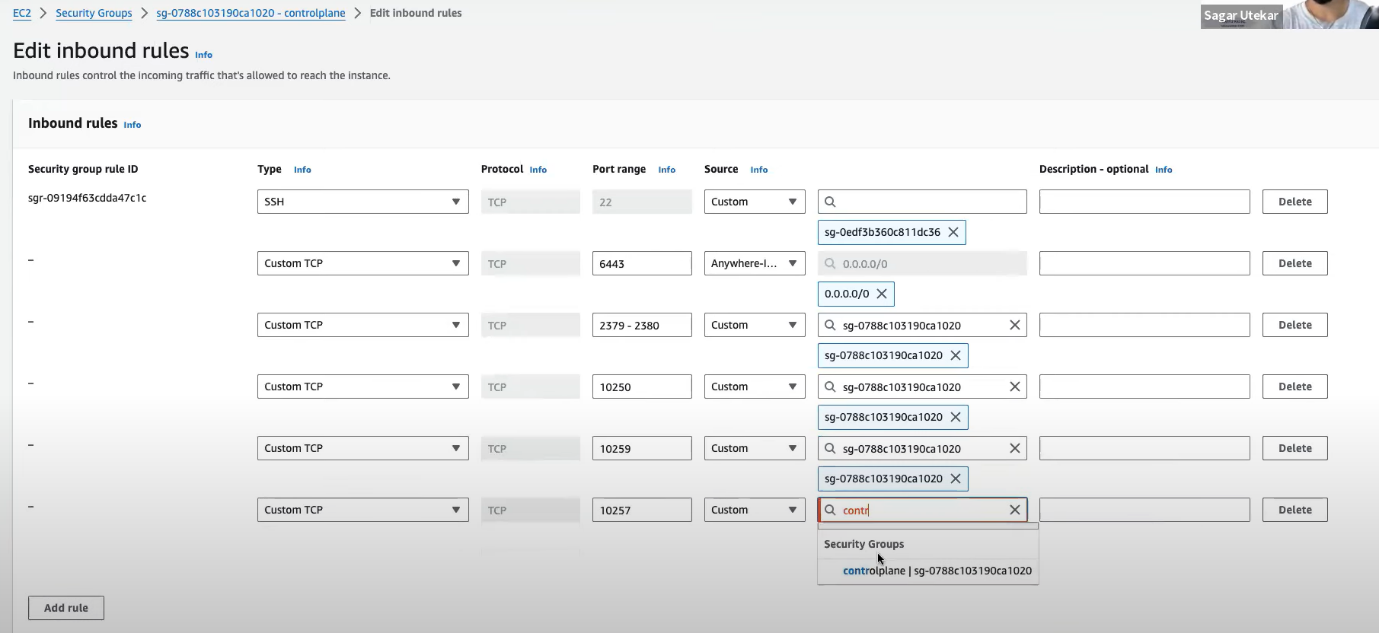
No of instances(2)

Launch instance

Step-4:

**Open ports on control plane**

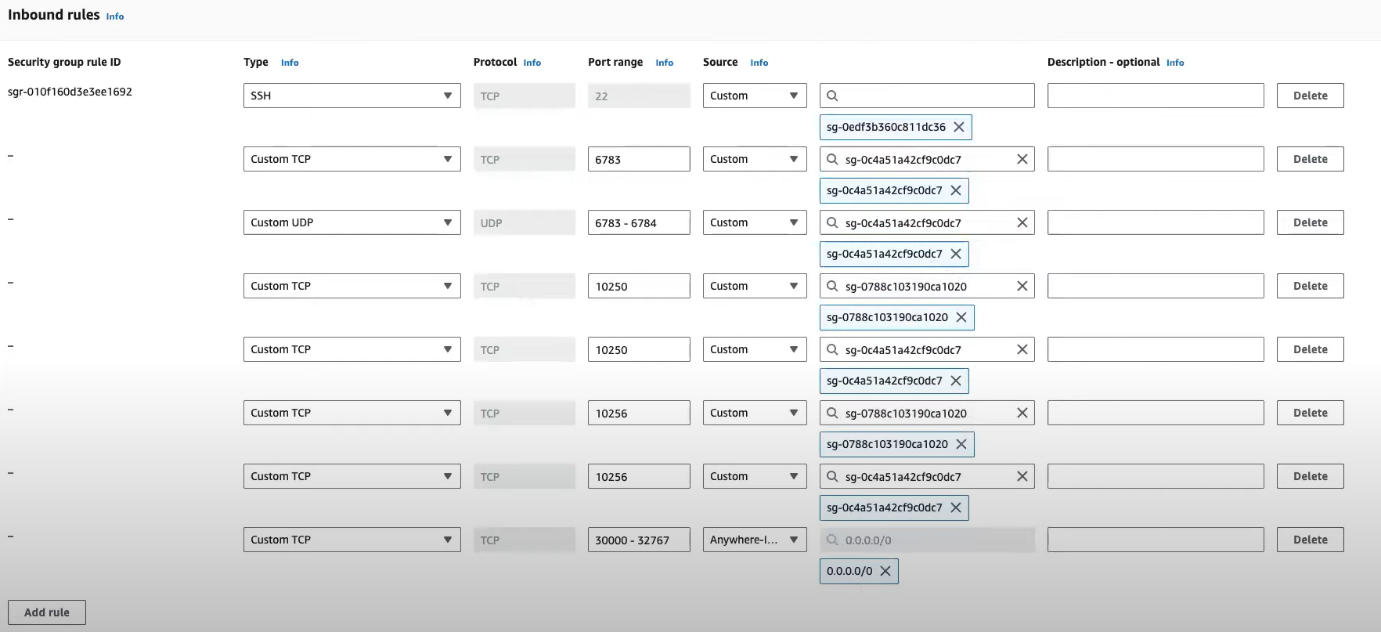
controlplane(ec2 instance) – SG – edit inbound rules



Also add:

* 6783/tcp
* 6783-6784/udp
* We mentioned this in ‘before getting started’ (see reference)

**Open ports on worker nodes**

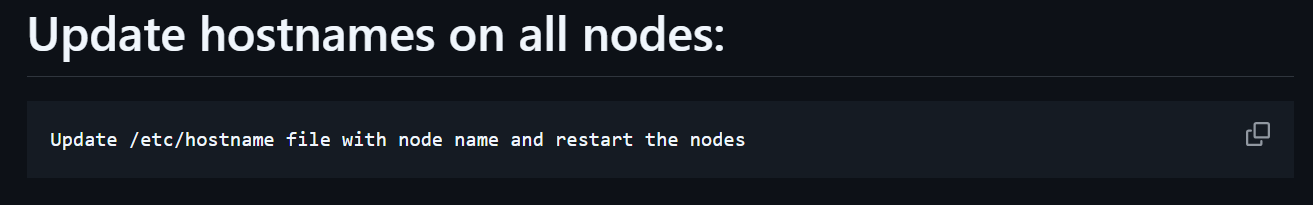


Note:-

10250, 10256 are for both controlplane and workernode SGs:

* 10250 – custom – source: controlplane
* 10250 – custom – source: workernode
* 10256 – custom – source: controlplane
* 10256 – custom – source: workernode

Step-5:



ssh into bastion node

* sudo su
* vi etc/hostname
* remove hostname (ip address) and type ‘bastion’
* :wq
* reboot

we’ll log into controlplane and worker nodes from bastion

Step-6:

In the terminal:

we need to copy bastion.pem and clusternodes.pem to controlplane server

* scp -i bastion.pem clusternode.pem ubuntu@<controlplane ip address>:/home/ubuntu

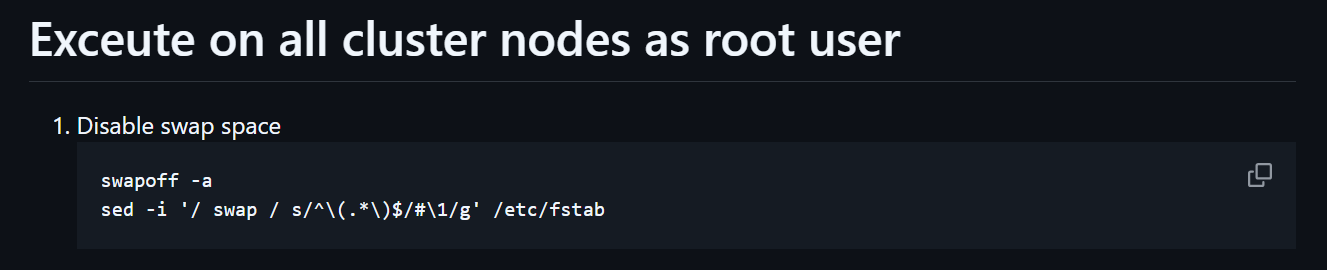
1. ssh into bastion node
2. from here we have to ssh into each of the worker nodes and control plane node:

* ssh -i clusternodes.pem ubuntu@<ip address of workernode-1>
* ssh -i clusternodes.pem ubuntu@<ip address of workernode-2>
* ssh -i clusternodes.pem ubuntu@<ip address of controlplane>

(each on a separate terminal tab)

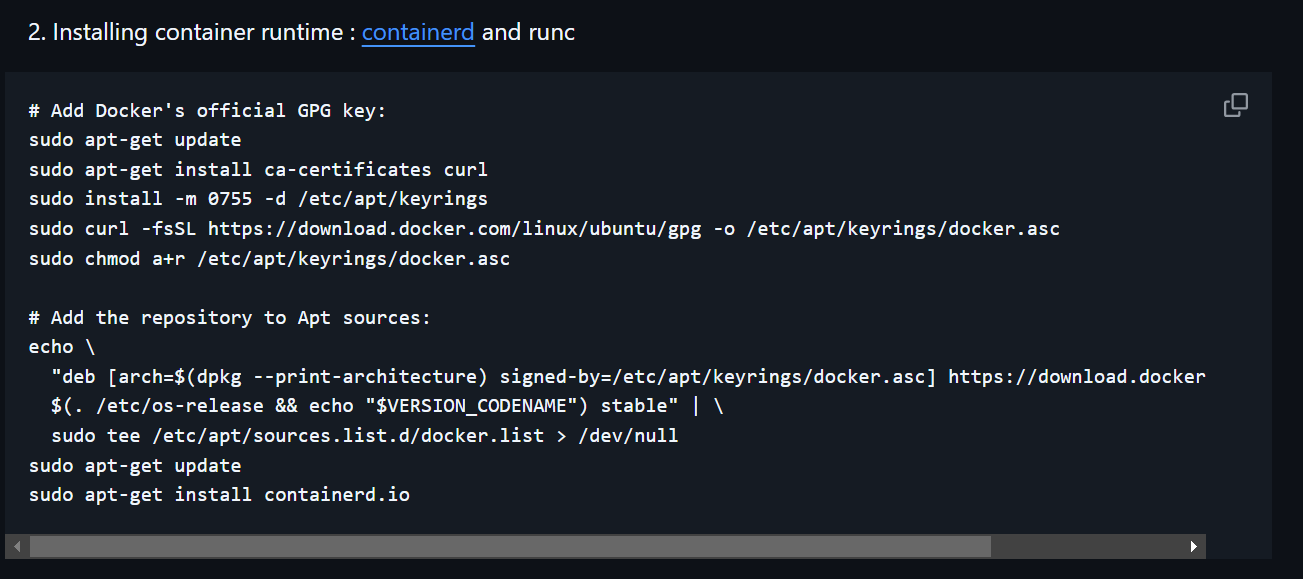
(yes, we used the same pem file for controlplane and workernodes. Confuse kaaku)

1. change hostname for controlplane and workernode like we did for bastion
   1. sudo su
   2. vi etc/hostname
   3. change name - :wq
   4. reboot



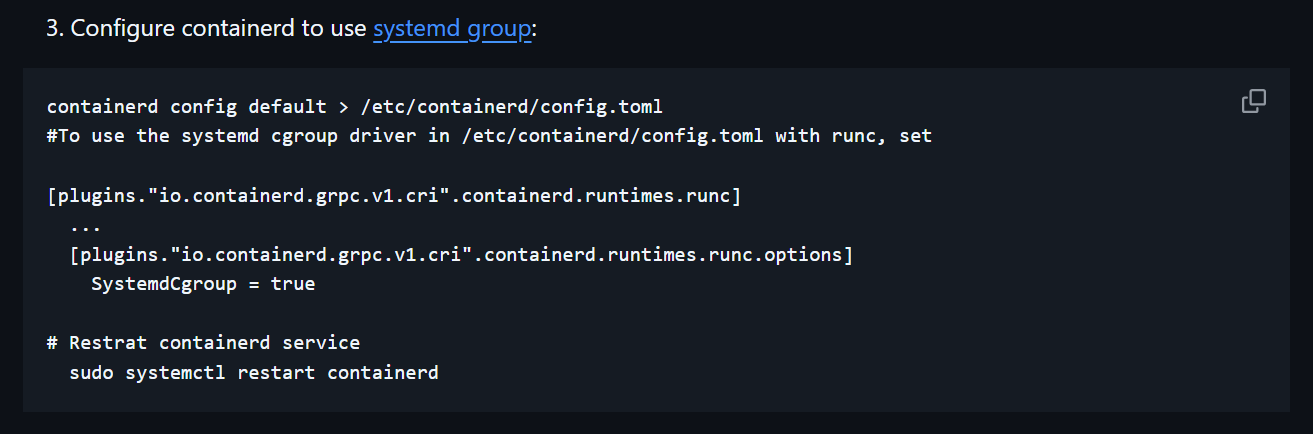
sudo su

Copy paste on all the cluster nodes



Copy paste these on all nodes to install containerd and runc

Check if installed or not using: *systemctl status containerd*

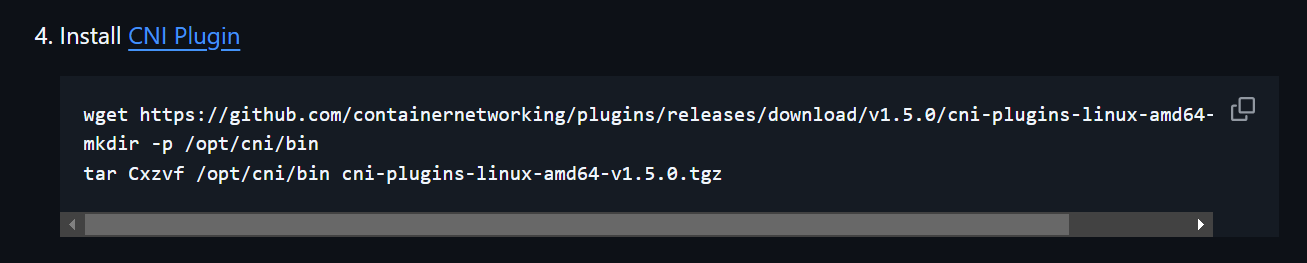


vi etc/containerd/config.toml

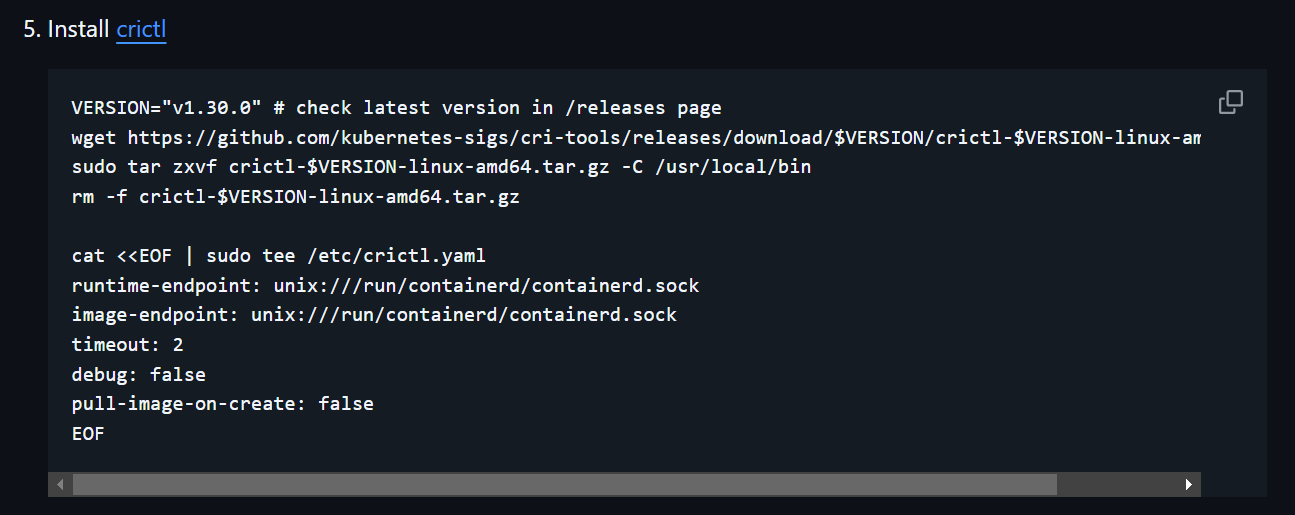
search for SystemdCgroup – by default it will be false. Make it true

(/Systemd --- you can search for text like this)

Restart containerd – systemctl restart containerd



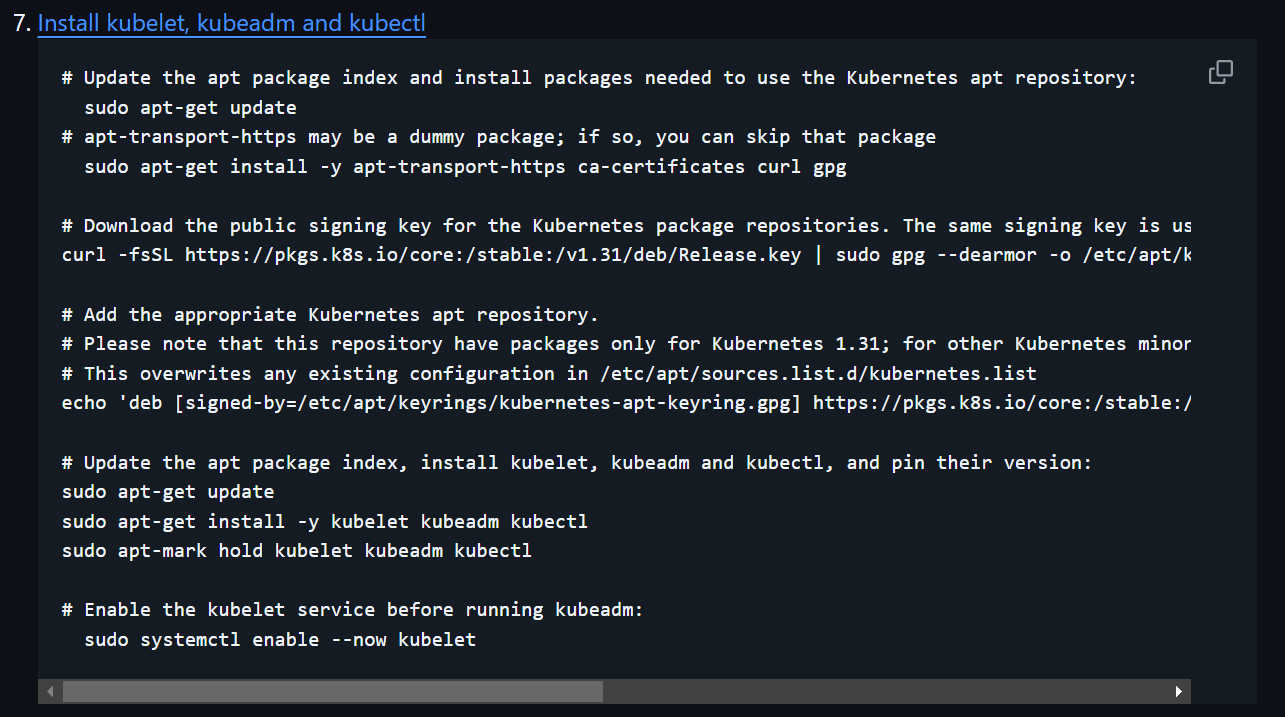
Copy paste



Copy paste

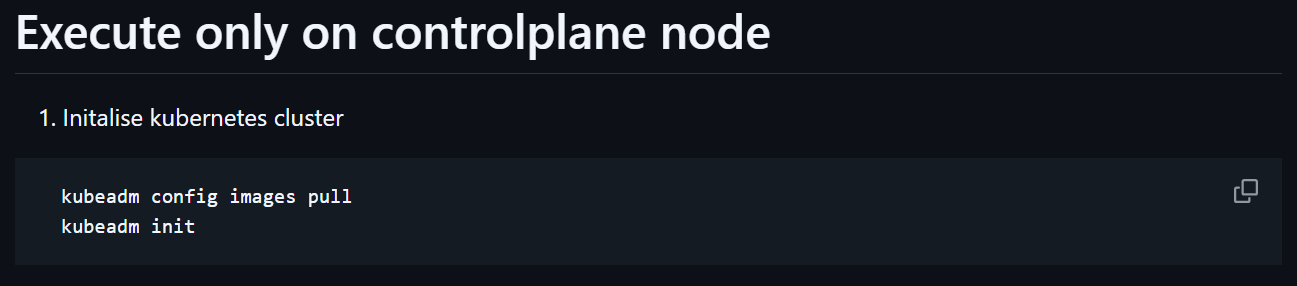


Copy paste

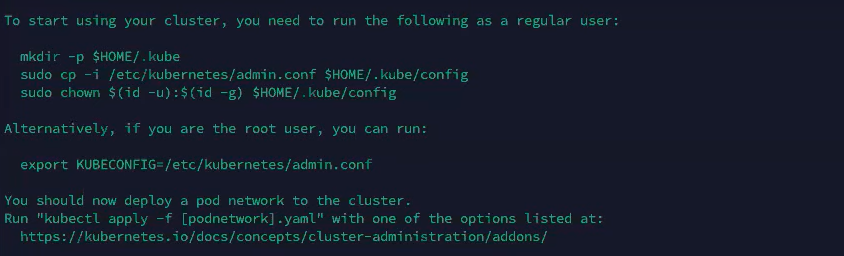


Copy paste

Step-7:



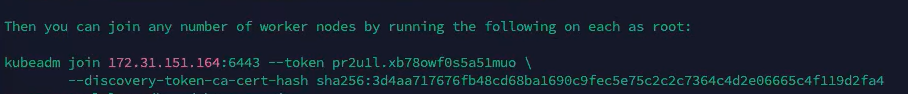
At the end of the output of kubeadm init, you will have some things like this:

1)

In controlplane node:

* sudo su
* export KUBECONFIG = /etc/kubernetes/admin.conf

2)



Copy this and paste in each of the workernodes

This joins the nodes to the cluster

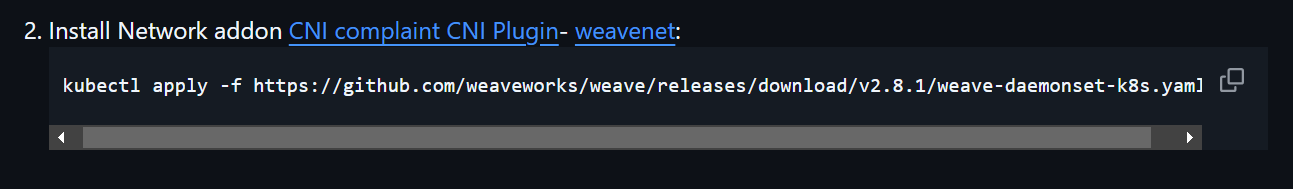
Note:- in worker nodes*, kubectl get nodes* gives an error but not in controlplane node

Because controlplane node has this file:

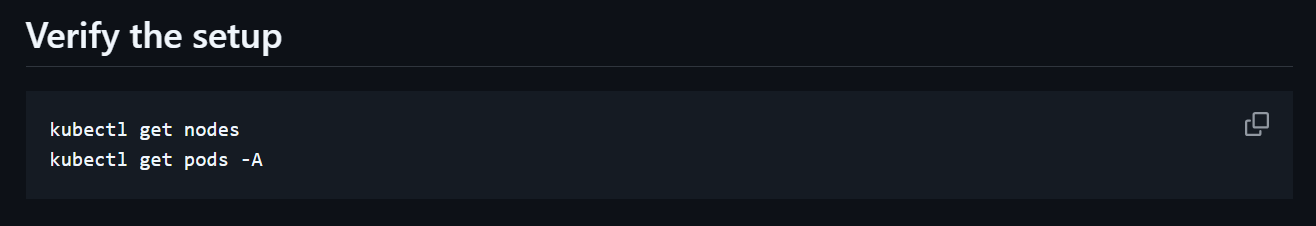


This allows kubectl to interact with the Kubernetes server and API server.

This is not present in worker nodes



Copy paste in controlplane



Verify if all pods are running with these commands