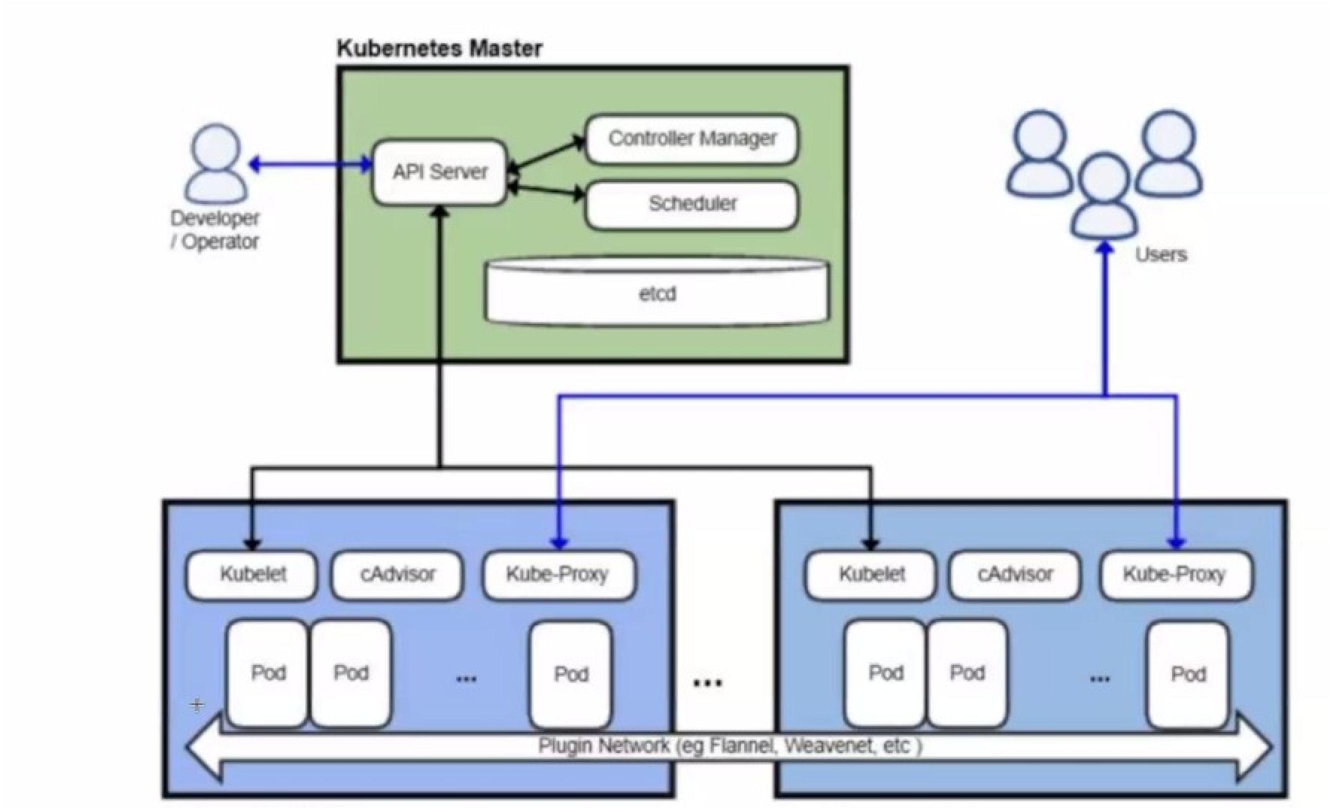


Containerization -

Min nodes required is 3 in cluster
Cluster nodes are added in odd number 3 5 7
Kubernetes - How it works -
Master and worker nodes

Developed by Google - since 2004 is where google was using
Borg
2013-14 – Docker
Docker swarm
Mesos/Marathon -

Yaml is very important to interact.
Kubernetes as managed services.

Amazon services – EKS and ECS. ECR – services
Codes will be written in yaml

Kubernetes works on master and worker

4 core components
Controller
Scheduler
etcd
API server -

API server - Manifest file – YAML file – We first create yaml file first.

Contoller -
The Kubernetes controller manager is a daemon that monitors the state of a Kubernetes cluster and makes changes to keep it running. It's a key component of the Kubernetes control plane.

Scheduler -process by which assigns work load
Resource balancing
Affinity
speading pods – Distribute the application – Workload.

Everything is stored as keyvalue – Metadata is stored.
etcd is an open-source key-value store that's a core component of Kubernetes. It's used to store and manage the state of Kubernetes clusters, including configuration data, metadata, and cluster state.

Kublet – Front end component – Receives instructions
executing work on worker notes.

Kubeproxy – service .
Kube-proxy is a component of Kubernetes that routes traffic for services within a cluster. It runs on each node in the cluster.

cAdvisor (Container Advisor) is an open-source tool that monitors containers in Kubernetes. It collects and exports metrics about CPU, memory, network, and filesystem usage. cAdvisor is integrated into the kubelet binary, so it's easy to use in Kubernetes.

Prometheus is an open-source tool that monitors and alerts for Kubernetes. It's used to collect and store metrics from Kubernetes applications and environments. Prometheus can help with:

Metric server – Cadvisor. All monitoring is done using Cadvisor.

Pod – Single node. Inside pod containers are created.
Pods are made up of one or more containers, such as Docker containers

Containers in a pod share resources like storage, network ports, and a unique IP address

- Containers within a pod communicate with each other using localhost
- Pods can run on a single physical machine, called a node
- Kubernetes can automatically create a new pod if a pod or its node fails

sample example try to understand : 1 box =pod - in 1 pod = more cotntainers - so 1 box = pod and containers- to manage these we need kubernetes.

We need to install kubernetes we need to install network plugin. Flannel – Weaver

Auto pilot mode Kub cluster.

Kubctl – If we want to interact with kubernetes cluster.

Client version and server version should be same.

We create pods -
Default – Docker
Create docker
Deploy using kubernetes.
Run time solution is docker

```
Kubectl get nodes
```

Give more properties of cluster.
Every node will have private and pubic.

```
Kubectl get pods
```

this will show the list of pods.

```
Kubectl run pod1 –nginx
```

This will create container image nginx

```
kubectl describe pod1
```

```
kubectl get pods -o wide
```

```
kubectl exec -it pod1 –bash
```

```
kubectl edit pod
```

We can edit the yaml file and then we can make changes and redeploy
we can change from nginx to httpd.

Docker commands will not run in kubernetes.

List of commands.

```
kubectl version --short
```

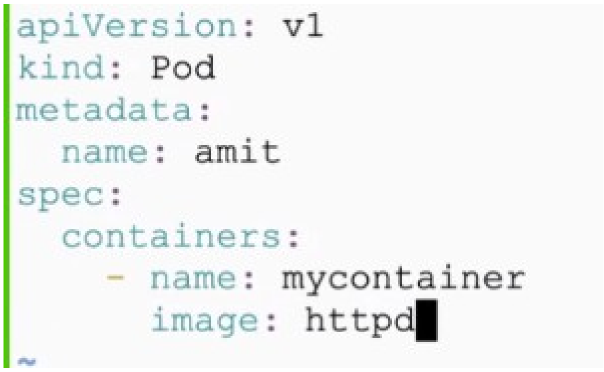
```
 clear
 kubectl get nodes
 kubectl get nodes -o wide
 kubectl get pods
 kubectl run pod1 --image nginx
  kubectl get pods
 kubectl describe pod pod1
  kubectl get pods -o wide
 clear
  kubectl get pods
  kubectl exec -it pod1 -- bash
 kubectl get pods
 kubectl edit pod pod1
  kubectl get pods
   kubectl describe pod pod1
  kubectl get pods
  kubectl delete pod pod1
```

Create pod using yaml file.

4 key properties.

apiversion
Kind
metadata
specs.

Kubernetes commands are case senstive.



```
apiVersion: v1
kind: Pod
metadata:
  name: amit
spec:
  containers:
    - name: mycontainer
      image: httpd
~
```

https://www.yamllint.com/

Used to check if yaml is valid or not, kind of compiler.

Simple yaml file from chat gpt.

```
---
apiVersion: v1
kind: Pod
metadata:
  name: httpd-pod
spec:
  containers:
    - name: httpd-container-1
      image: httpd:latest
      ports:
        - containerPort: 80
    - name: httpd-container-2
      image: httpd:latest
      ports:
        - containerPort: 80
    - name: httpd-container-3
      image: httpd:latest
      ports:
        - containerPort: 80
    - name: httpd-container-4
      image: httpd:latest
      ports:
        - containerPort: 80
```

Namespace

A **namespace** in Kubernetes is a logical partition within a cluster that allows you to organize and manage resources in a way that provides isolation, access control, and easier management of workloads. Namespaces are a way to divide cluster resources between multiple users or applications.

There will be default name space create and we can specify new namespace.

Labels.

In Kubernetes, labels are key-value pairs that attach metadata to objects, such as pods, services, and deployments. Labels help organize and identify resources, and can be used to perform bulk operations.

Deployments and replica set.

Using deployments – Pods will get created. ‘
Create yaml file and deploy it.

In case pod 1 is getting deleted new pod will get added.

We can scale up and scale down using replicas commands.

How to practise kubernetes on ec2? This is big problem for me, can you please advise.
How to practise kubernetes on vbox, please suggest document on how can we do I can do? I will do it myself.
Please share insights on these questions.

Please share detailed steps to download and install kubenetes on vbox. I found few but not working.
Minikube is not working on my machine.

Sol : Use Minikube to practise on Ec2.

Services : This will sit on application to deploy services.
Cluster ip.
Node port
load balancer.

Default cluster ip service – Private ip is created – Application is only accessible with in the cluster.

Node port – Application can be externally be accesisble over some device port.

In Kubernetes, a **NodePort** is a type of **Service** that exposes a pod or set of pods to external traffic by opening a specific port on each node (worker node) in your Kubernetes cluster. It allows access to the application running inside a pod from outside the Kubernetes cluster, using the IP address of any of the worker nodes and a fixed port.

Key Concepts:

- **Service**: A Kubernetes resource that defines how to access a set of pods, typically by defining a policy for accessing them. Services are responsible for routing traffic to the appropriate pods, even if pods are dynamically created or destroyed.
- **NodePort**: A specific type of Service in Kubernetes. It exposes the service on each **node's IP address** at a **static port** (within a range of `30000-32767` by default).

When you create a **NodePort** service, Kubernetes allocates a port from this range on all nodes in the cluster. You can then use the IP address of any node in the cluster and the allocated NodePort to access the service.

How LoadBalancer Works in Kubernetes:

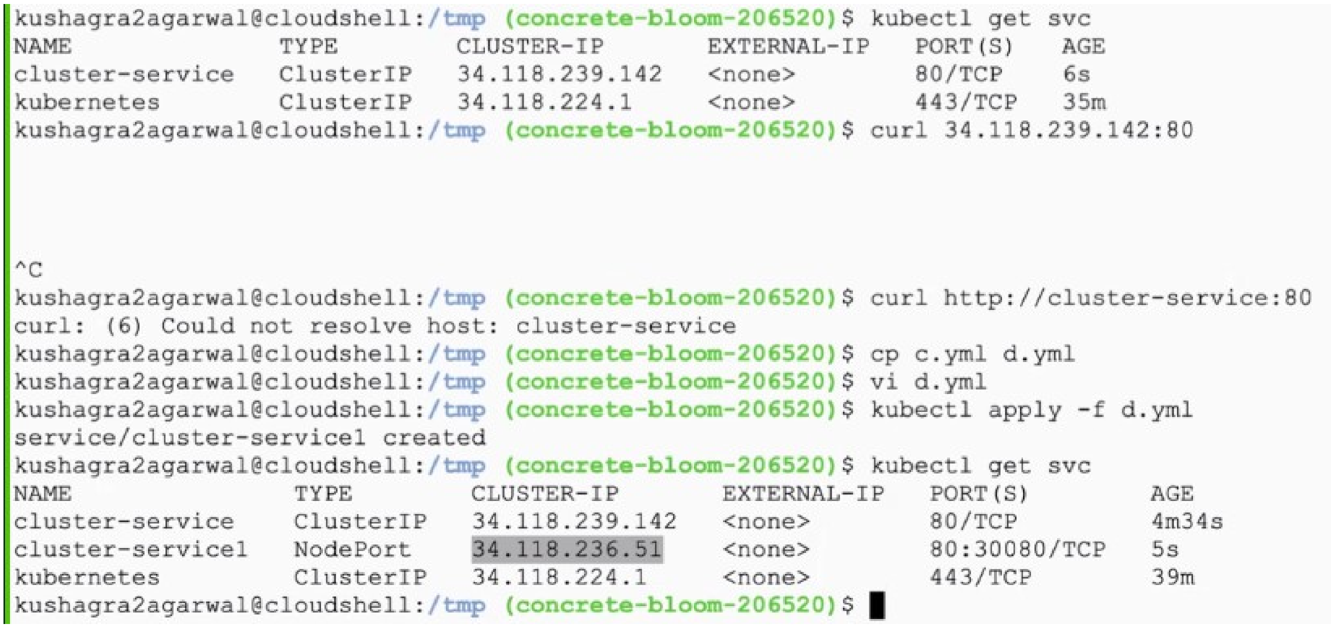
1. **External Access**: A LoadBalancer service gives external clients access to your application, using a public IP address or DNS name provided by the load balancer.
2. **Traffic Distribution**: The external load balancer distributes the incoming traffic to the pods selected by the Kubernetes service. It balances the load across the pods based on the load balancing algorithm (round-robin, least connections, etc.).
3. **Cloud Provider Integration**: When running Kubernetes on a cloud platform (like AWS, GCP, or Azure), Kubernetes interacts with the cloud provider's API to provision the load balancer. The cloud provider manages the external IP address and the necessary configuration for the load balancer.

Explanation:

- `port`: The port on the service that will be exposed to the outside world (e.g., `80`).
- `targetPort`: The port inside the pod where the application is running (e.g., `8080`).
- `type: LoadBalancer`: This makes the service a LoadBalancer type. Kubernetes will request the cloud provider to provision an external load balancer and route traffic to the pods based on the service's selector.

How It Works:

1. **Cloud Provider Creates a Load Balancer**: When you create a LoadBalancer service in Kubernetes on a cloud platform (like AWS, Azure, or GCP), Kubernetes communicates with the cloud provider's API to create an external load balancer (e.g., an Elastic Load Balancer on AWS).
2. **Assigns a Public IP**: The cloud provider assigns a public IP address or DNS name to the load balancer, which can be used by clients to access the service.
3. **Load Balancer Routes Traffic**: The external load balancer forwards the traffic to the Kubernetes nodes, which then forward it to the appropriate pods selected by the Kubernetes service.
4. **Health Checks**: The cloud provider's load balancer typically performs health checks on the pods to ensure that traffic is only sent to healthy pods.



```
kushagra2agarwal@cloudshell:/tmp (concrete-bloom-206520)$ kubectl get svc
NAME              TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)   AGE
cluster-service   ClusterIP   34.118.239.142   <none>        80/TCP    6s
kubernetes        ClusterIP   34.118.224.1     <none>        443/TCP   35m
kushagra2agarwal@cloudshell:/tmp (concrete-bloom-206520)$ curl 34.118.239.142:80




^C
kushagra2agarwal@cloudshell:/tmp (concrete-bloom-206520)$ curl http://cluster-service:80
curl: (6) Could not resolve host: cluster-service
kushagra2agarwal@cloudshell:/tmp (concrete-bloom-206520)$ cp c.yml d.yml
kushagra2agarwal@cloudshell:/tmp (concrete-bloom-206520)$ vi d.yml
kushagra2agarwal@cloudshell:/tmp (concrete-bloom-206520)$ kubectl apply -f d.yml
service/cluster-service1 created
kushagra2agarwal@cloudshell:/tmp (concrete-bloom-206520)$ kubectl get svc
NAME               TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
cluster-service    ClusterIP   34.118.239.142   <none>        80/TCP         4m34s
cluster-service1   NodePort    34.118.236.51    <none>        80:30080/TCP   5s
kubernetes         ClusterIP   34.118.224.1     <none>        443/TCP        39m
kushagra2agarwal@cloudshell:/tmp (concrete-bloom-206520)$ 
```

Secrets

Store passwords eg database details we can save in yaml file.
While creating password we need to give key value.

Give the value in the yaml file so that it will be fetching from password file.

Base 64 encryption and decryption.

Persistant volume. Only admins can create the volume.

So far topics covered.

Pod-to-pod
Deployments
Services
Secrets
Persistent volume.
Delete – Grace period

why deletion of pod take time?

ChatGPT

The deletion of a pod in Kubernetes can take time for several reasons:

1. **Grace Period**: When you delete a pod, Kubernetes doesn’t immediately terminate the pod. Instead, it gives the containers inside the pod a grace period (usually 30 seconds by default) to shut down cleanly. This allows any ongoing processes to complete or save their state. If the pod is not able to terminate within the grace period, Kubernetes forces the deletion.

2. **Resource Cleanup**: The containers within the pod may have resources, like network connections, volumes, or other services that need to be cleaned up before the pod can be fully removed. This can add to the time it takes for the pod to be deleted.
3. **Pod Lifecycle Hooks**: Some pods may have lifecycle hooks such as `preStop` hooks that need to execute before they can be deleted. If these hooks have long-running processes or are waiting for something to complete, it will delay the pod's deletion.
4. **Pod Termination State**: Kubernetes will attempt to gracefully terminate the pod, and this state of termination could take some time depending on the container's exit strategy, the type of workload, or the amount of resources the pod is using.
5. **Node or Cluster Overload**: If the Kubernetes node or the cluster is overloaded or under heavy load, it might take longer to properly clean up resources, communicate the pod's termination, and update the state across the system.
6. **Pending Deletion Events**: Sometimes, the pod might still be in the process of notifying or communicating with other services (like an ingress controller or service). This can sometimes delay the final deletion process.

In summary, Kubernetes works to ensure that pods are terminated in a way that avoids data loss, leaves resources in a clean state, and respects any configured shutdown processes, which can all add to the time it takes to delete the pod.

helm chart.

https://helm.sh/

Helm helps you manage Kubernetes applications — Helm Charts help you define, install, and upgrade even the most complex Kubernetes application.

Charts are easy to create, version, share, and publish — so start using Helm and stop the copy-and-paste.

Helm is a graduated project in the CNCF and is maintained by the Helm community.

Learn more:

Helm Architecture

Quick Start Guide

Video: An Introduction to Helm

https://github.com/yeshwanthlm/Kubeadm-Installation-Guide?tab=readme-ov-file

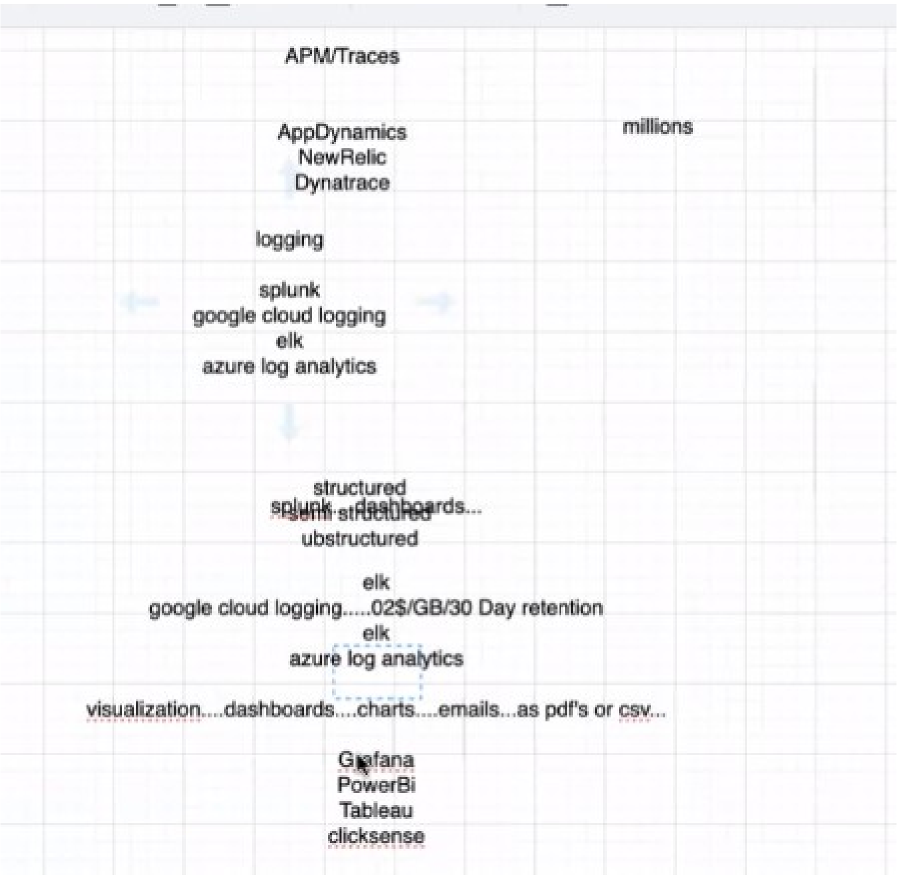
Installation refer above

```
sudo modprobe br_netfilter

echo "1" | sudo tee /proc/sys/net/bridge/bridge-nf-call-iptables

echo "net.ipv4.ip_forward = 1" | sudo tee -a /etc/sysctl.conf

sudo sysctl --system

sudo systemctl restart docker kubelet

kubectl get nodes   (in master)
```

https://helm.sh/docs/intro/install/

Helm helps you manage Kubernetes applications — Helm Charts help you define, install, and upgrade even the most complex Kubernetes application.

Charts are easy to create, version, share, and publish — so start using Helm and stop the copy-and-paste.



APM/Traces

AppDynamics
NewRelic
Dynatrace

millions

logging

splunk
google cloud logging
elk
azure log analytics

structured
splunk...dashboards...
semi structured
ubstructured

elk
google cloud logging.....02$/GB/30 Day retention
elk
azure log analytics

visualization....dashboards....charts....emails...as pdf's or csv...

Grafana
PowerBi
Tableau
clicksense

Monitoring important.

OSI layers.

How do I monitor services, pods containers and services?

Prometheus – Opensource monitoring system.

Epoch timestamp.

My kubernetes

```
sudo kubeadm init --apiserver-advertise-address=172.31.30.134 --pod-network-cidr=192.168.0.0/16 --ignore-preflight-errors=ALL
```

```
kubeadm join 172.31.30.134:6443 --token j4ml17.r4gvwqvgap31fpwu \ --discovery-token-ca-cert-hash
sha256:2eb13908d3e46790c2bc2644489c15178c8381522fa09f82f93be62666c86029
```

new server

```
kubeadm join 172.31.20.33:6443 --token id5alh.swrpd99hon35naiw \
        --discovery-token-ca-cert-hash
sha256:8cbd8148ecdcb99fd3a46482af68d537ff492607018aea25ab474959f3503791
```

```
ssh -i "ubuntukubernetesfeb2025.pem" ubuntu@ec2-54-226-45-56.compute-1.amazonaws.com -
Master
ssh -i "ubuntukubernetesfeb2025.pem" ubuntu@ec2-3-90-104-140.compute-1.amazonaws.com
-Worker
```

```
kubectl taint nodes ip-172-31-20-33 node-role.kubernetes.io/control-plane:NoSchedule-
```

Step 1: Remove the Taint from the Master Node
Use the following command to remove the taint from the master node:

bash
```
kubectl taint nodes <node-name> node-role.kubernetes.io/control-plane:NoSchedule-
```
Replace <node-name> with the actual name of your master node. You can get the node name using the following command:

bash
```
kubectl get nodes
```
Step 2: Verify the Taint Removal
After removing the taint, verify that the taint has been removed:

bash
```
kubectl describe node <node-name>
```
Ensure that the node-role.kubernetes.io/control-plane:NoSchedule taint is no longer listed.

Step 3: Check Pod Status
Check the status of your pods again to see if they have been scheduled:

bash
```
kubectl get pods
```
Summary of Steps:
Remove the taint from the master node:

bash
```
kubectl taint nodes <node-name> node-role.kubernetes.io/control-plane:NoSchedule-
```
Verify the taint removal:

bash

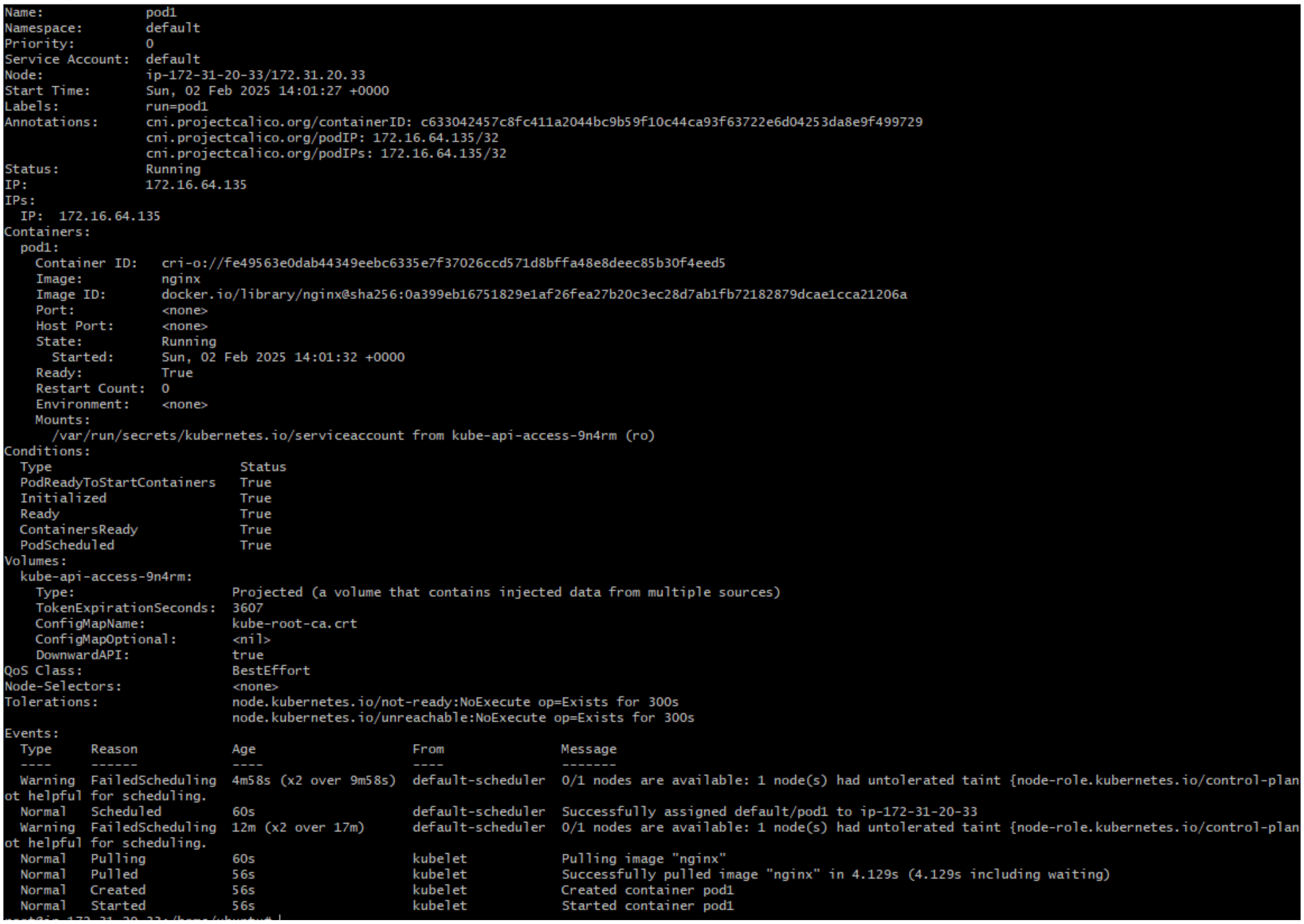
```
kubectl describe node <node-name>
```

Check the status of the pods:

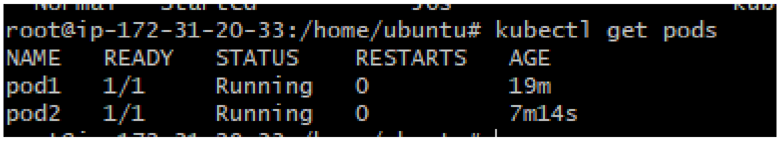
bash

```
kubectl get pods
```

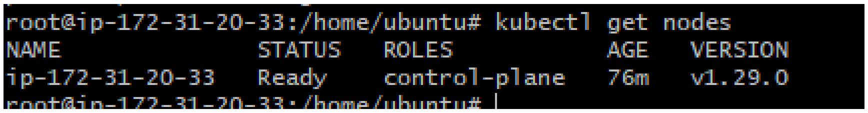
By following these steps, the pods should be able to be scheduled on the master node, resolving the "FailedScheduling" issue. If you encounter any specific error messages or issues during these steps, please share them, and we can troubleshoot further. 😊



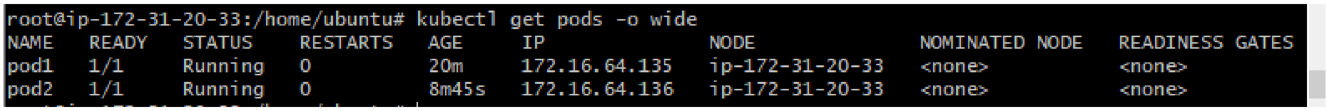
```
Name:             pod1
Namespace:        default
Priority:         0
Service Account:  default
Node:             ip-172-31-20-33/172.31.20.33
Start Time:       Sun, 02 Feb 2025 14:01:27 +0000
Labels:           run=pod1
Annotations:      cni.projectcalico.org/containerID: c633042457c8fc411a2044bc9b59f10c44ca93f63722e6d04253da8e9f499729
                  cni.projectcalico.org/podIP: 172.16.64.135/32
                  cni.projectcalico.org/podIPs: 172.16.64.135/32
Status:           Running
IP:               172.16.64.135
IPs:
  IP:  172.16.64.135
Containers:
  pod1:
    Container ID:   cri-o://fe49563e0dab44349eebc6335e7f37026ccd571d8bffa48e8deec85b30f4eed5
    Image:          nginx
    Image ID:       docker.io/library/nginx@sha256:0a399eb16751829e1af26fea27b20c3ec28d7ab1fb72182879dcae1cca21206a
    Port:           <none>
    Host Port:      <none>
    State:          Running
      Started:      Sun, 02 Feb 2025 14:01:32 +0000
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-9n4rm (ro)
Conditions:
  Type                        Status
  PodReadyToStartContainers   True
  Initialized                 True
  Ready                       True
  ContainersReady             True
  PodScheduled                True
Volumes:
  kube-api-access-9n4rm:
    Type:                    Projected (a volume that contains injected data from multiple sources)
    TokenExpirationSeconds:  3607
    ConfigMapName:           kube-root-ca.crt
    ConfigMapOptional:       <nil>
    DownwardAPI:             true
QoS Class:                   BestEffort
Node-Selectors:              <none>
Tolerations:                 node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                             node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:
  Type     Reason            Age                   From               Message
  ----     ------            ----                  ----               -------
  Warning  FailedScheduling  4m58s (x2 over 9m58s)  default-scheduler  0/1 nodes are available: 1 node(s) had untolerated taint {node-role.kubernetes.io/control-plan
ot helpful for scheduling.
  Normal   Scheduled         60s                   default-scheduler  Successfully assigned default/pod1 to ip-172-31-20-33
  Warning  FailedScheduling  12m (x2 over 17m)     default-scheduler  0/1 nodes are available: 1 node(s) had untolerated taint {node-role.kubernetes.io/control-plan
ot helpful for scheduling.
  Normal   Pulling           60s                   kubelet            Pulling image "nginx"
  Normal   Pulled            56s                   kubelet            Successfully pulled image "nginx" in 4.129s (4.129s including waiting)
  Normal   Created           56s                   kubelet            Created container pod1
  Normal   Started           56s                   kubelet            Started container pod1
```



```
root@ip-172-31-20-33:/home/ubuntu# kubectl get pods
NAME   READY   STATUS    RESTARTS   AGE
pod1   1/1     Running   0          19m
pod2   1/1     Running   0          7m14s
```



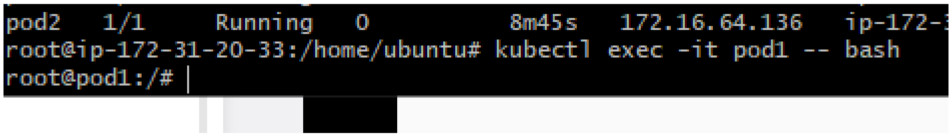
```
root@ip-172-31-20-33:/home/ubuntu# kubectl get nodes
NAME               STATUS   ROLES           AGE   VERSION
ip-172-31-20-33    Ready    control-plane   76m   v1.29.0
root@ip-172-31-20-33:/home/ubuntu# |
```



```
root@ip-172-31-20-33:/home/ubuntu# kubectl get pods -o wide
NAME   READY   STATUS    RESTARTS   AGE     IP              NODE              NOMINATED NODE   READINESS GATES
pod1   1/1     Running   0          20m     172.16.64.135   ip-172-31-20-33   <none>           <none>
pod2   1/1     Running   0          8m45s   172.16.64.136   ip-172-31-20-33   <none>           <none>
```

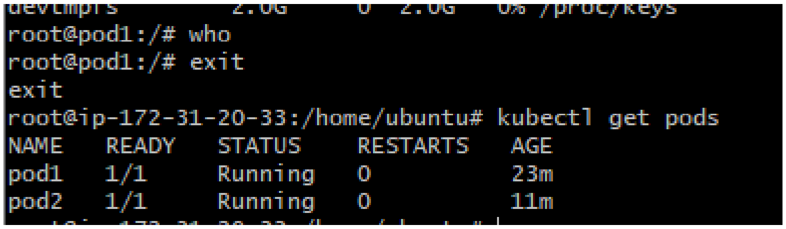
Login inside the pod.

`kubectl get pods -o wide`



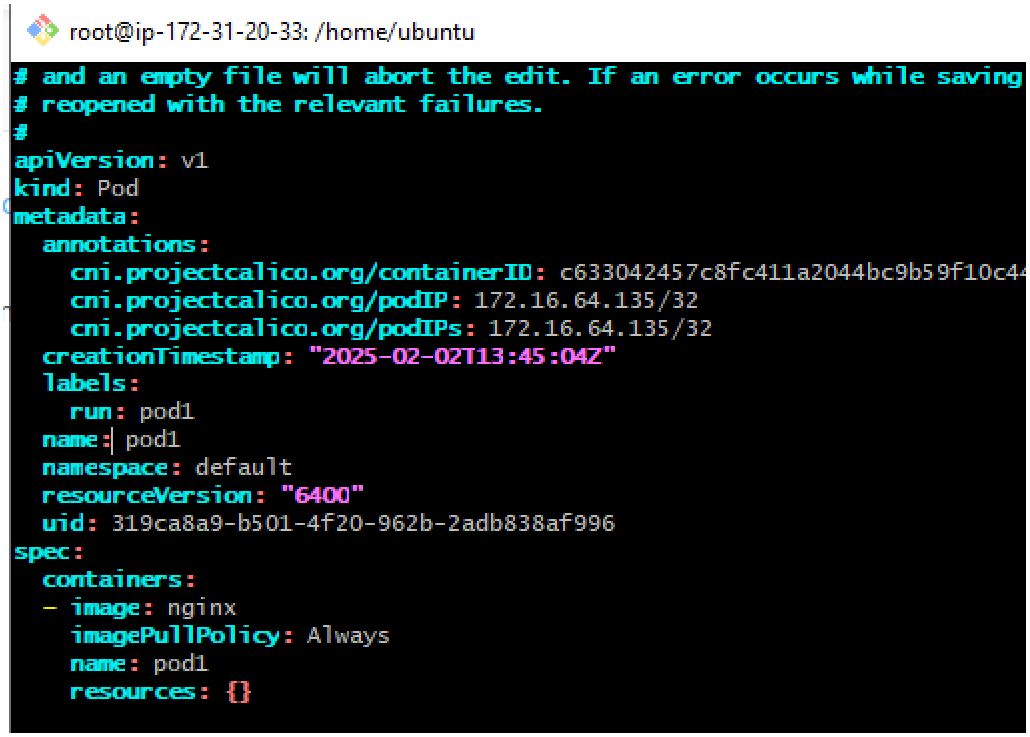
```
pod2   1/1     Running   0          8m45s   172.16.64.136   ip-172-
root@ip-172-31-20-33:/home/ubuntu# kubectl exec -it pod1 -- bash
root@pod1:/# |
```

Even after running pod is running. This was not happening in docker.



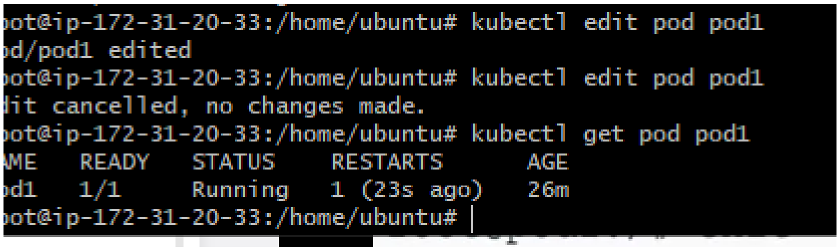
```
root@pod1:/# who
root@pod1:/# exit
exit
root@ip-172-31-20-33:/home/ubuntu# kubectl get pods
NAME   READY   STATUS    RESTARTS   AGE
pod1   1/1     Running   0          23m
pod2   1/1     Running   0          11m
```

We can edit the pod info and change from nginx to httpd.



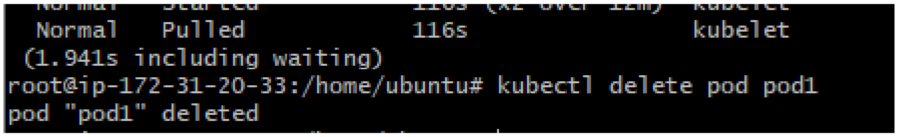
root@ip-172-31-20-33: /home/ubuntu

```
# and an empty file will abort the edit. If an error occurs while saving
# reopened with the relevant failures.
#
apiVersion: v1
kind: Pod
metadata:
  annotations:
    cni.projectcalico.org/containerID: c633042457c8fc411a2044bc9b59f10c4
    cni.projectcalico.org/podIP: 172.16.64.135/32
    cni.projectcalico.org/podIPs: 172.16.64.135/32
  creationTimestamp: "2025-02-02T13:45:04Z"
  labels:
    run: pod1
  name: pod1
  namespace: default
  resourceVersion: "6400"
  uid: 319ca8a9-b501-4f20-962b-2adb838af996
spec:
  containers:
  - image: nginx
    imagePullPolicy: Always
    name: pod1
    resources: {}
```



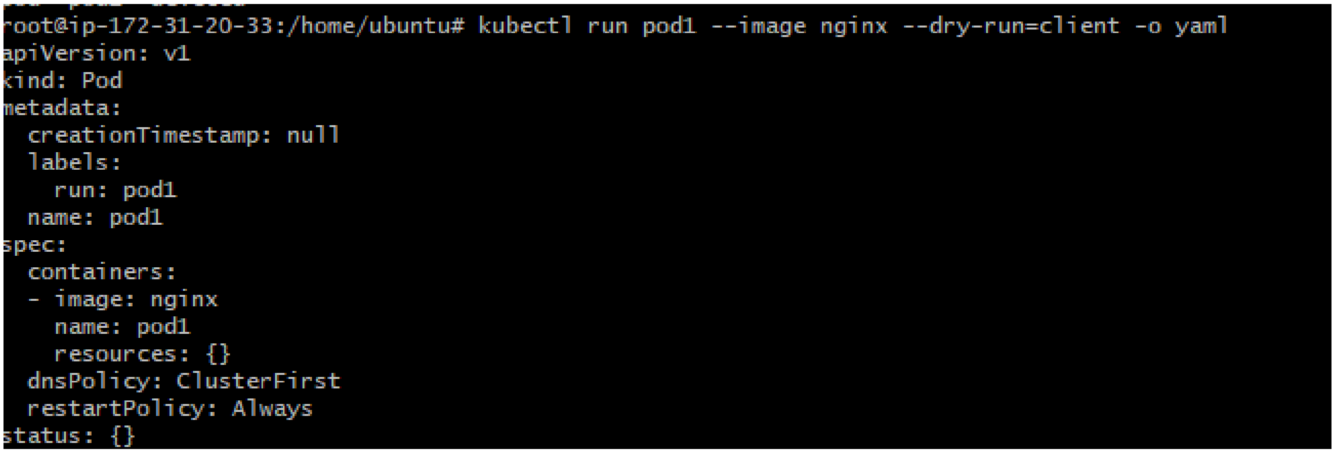
```
ot@ip-172-31-20-33:/home/ubuntu# kubectl edit pod pod1
d/pod1 edited
ot@ip-172-31-20-33:/home/ubuntu# kubectl edit pod pod1
it cancelled, no changes made.
ot@ip-172-31-20-33:/home/ubuntu# kubectl get pod pod1
ME    READY   STATUS    RESTARTS       AGE
d1    1/1     Running   1 (23s ago)    26m
ot@ip-172-31-20-33:/home/ubuntu# |
```

We can delete the pod



```
  Normal   Pulled               116s                kubelet
 (1.941s including waiting)
root@ip-172-31-20-33:/home/ubuntu# kubectl delete pod pod1
pod "pod1" deleted
```

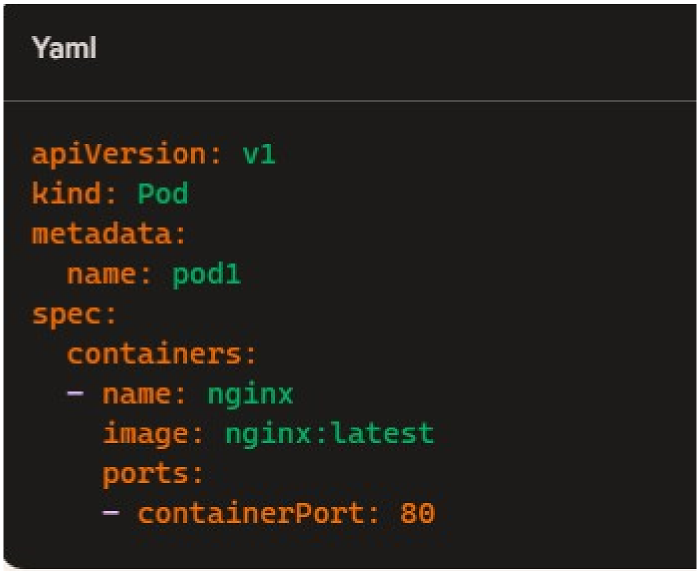
```
root@ip-172-31-20-33:/home/ubuntu# kubectl run pod1 --image nginx --dry-run=client -o yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: pod1
  name: pod1
spec:
  containers:
  - image: nginx
    name: pod1
    resources: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Always
status: {}
```



```
root@ip-172-31-20-33:/home/ubuntu# kubectl run pod1 --image nginx --dry-run=client -o yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: pod1
  name: pod1
spec:
  containers:
  - image: nginx
    name: pod1
    resources: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Always
status: {}
```

Create pod using yaml file

```
apiVersion: v1
kind: Pod
metadata:
  name: pod1
spec:
  containers:
  - name: nginx
    image: nginx:latest
    ports:
    - containerPort: 80
```



```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod1
spec:
  containers:
  - name: nginx
    image: nginx:latest
    ports:
    - containerPort: 80
```