Operating System Virtualization
UNIT-V-Part-III

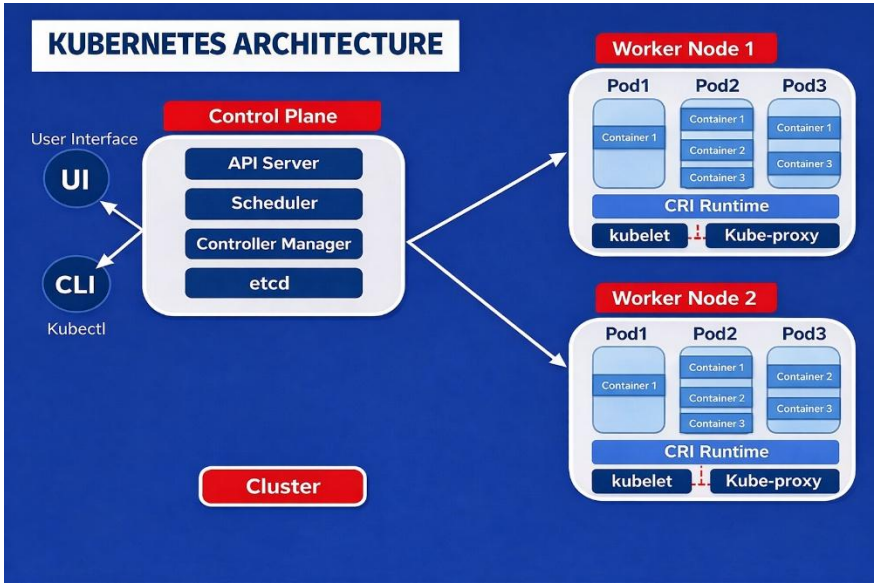
Kubernetes (K8s)

Why Kubernetes

- What Docker gives?
  - Single hosting, K8s gives Cluster
  - No auto healing
  - No auto scaling
  - No support for enterprise applications

What is Kubernetes

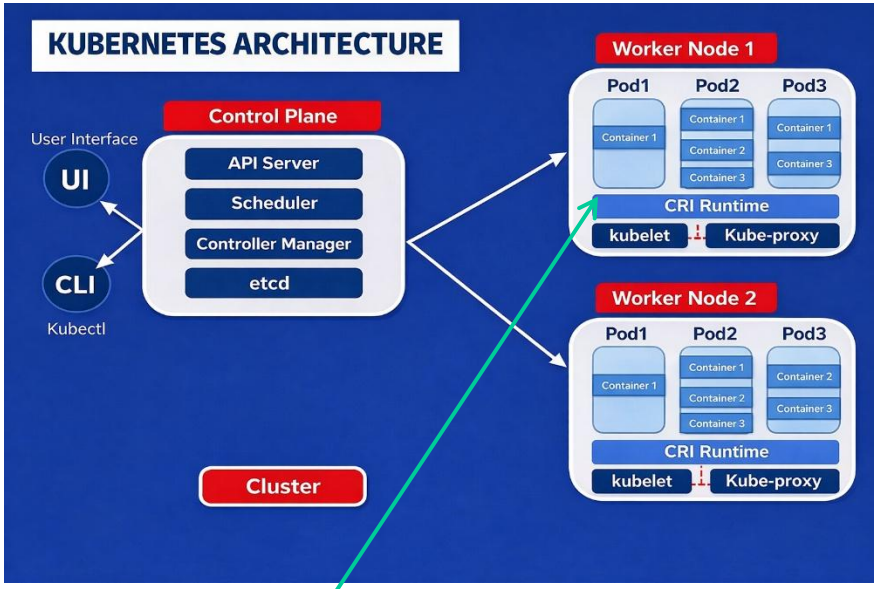
- Containerization simplifies application deployment & management in the cloud.
- Kubernetes automates deploying, scaling, and operations of containers.
- Kubernetes cluster consists of master & worker nodes



Source: https://collabnix.com/5-minutes-to-kubernetes-architecture/

What is Kubernetes

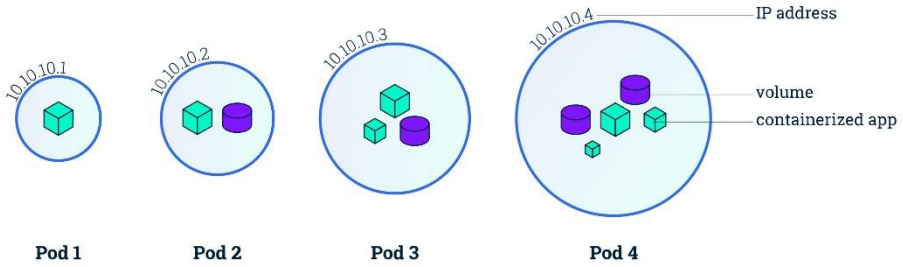
- Kubernetes clusters consist of nodes or servers managed collectively.
- Nodes host containers, while Kubernetes schedules and manages them.



CRI is implemented by container runtimes such as containerd, CRI-O, and others that support the Kubernetes Container Runtime Interface.

Pod in Kubernetes

- **Pod**, is the **smallest unit** where containers run (execution environment) and the most common resource in a Kubernetes cluster.
- Pod's are **created** by a **controller** and assigned to a node by a **scheduler**.
- All containers within a Pod share a network namespace.

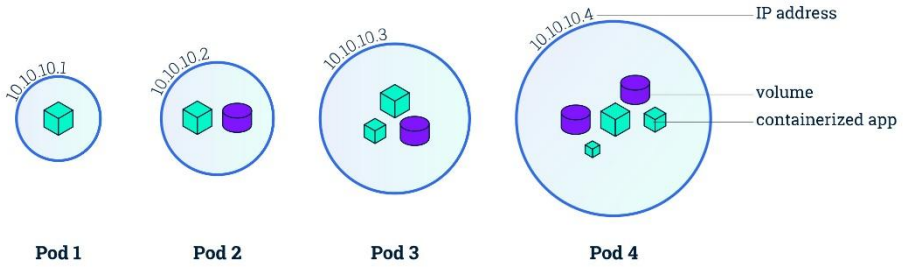


IMPORTANT

Containers should only be scheduled together in a single Pod if they are tightly coupled and need to share resources such as disk.

Pod in Kubernetes

- Pod's are mortal.
- The process of building and running an app on Kubernetes is roughly as follows:
  - Write your app/code.
  - Package it as a container image.
  - Wrap the container image in a Pod.
  - Run it on Kubernetes.

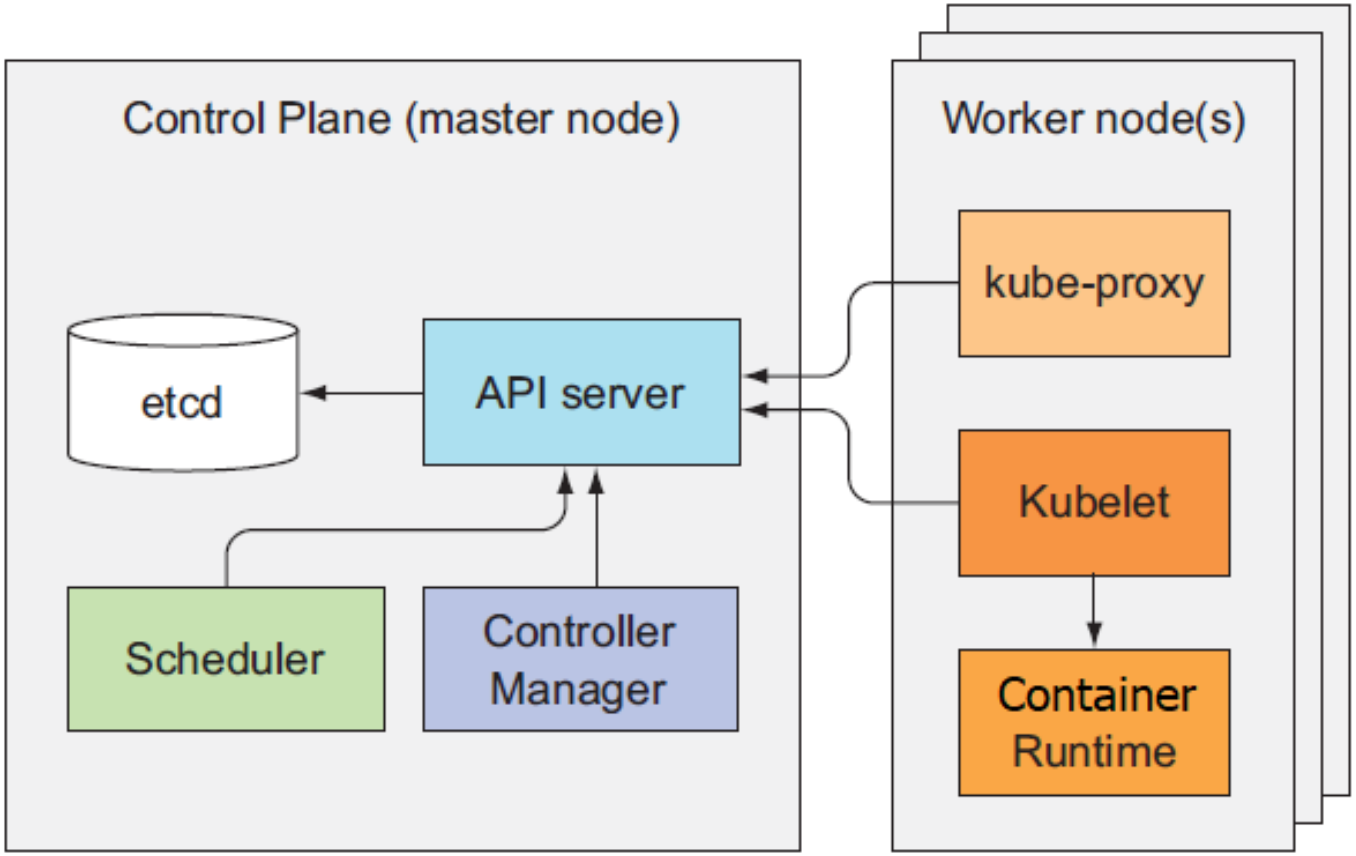


A Pod is not a process, and a container is not strictly a process, but it executes one or more processes in isolation.

Control and Data Plane

- **Control Plane**
- **Purpose:**
  - It is the brain of the Kubernetes cluster.
  - It makes decisions about the overall cluster state, such as which nodes run which Pods, and ensures that the desired state* (as declared by the user) is maintained.

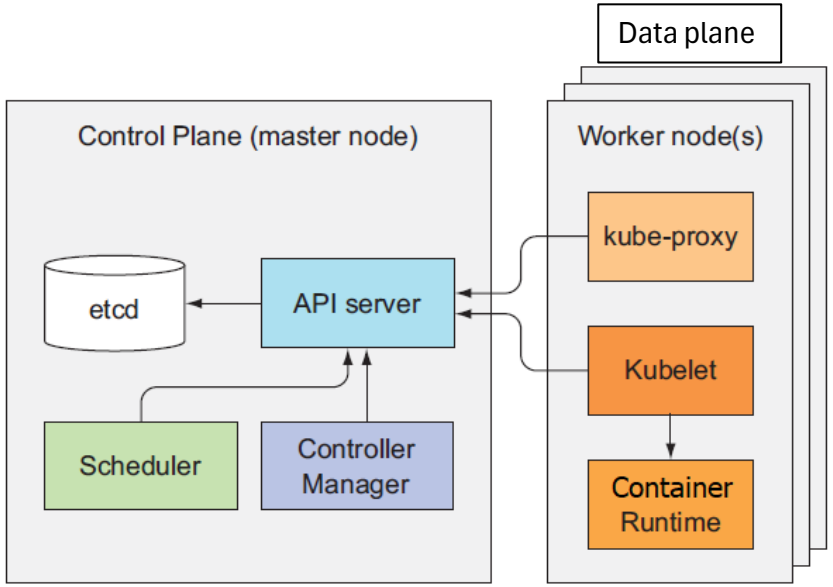
Control Plane and the worker nodes



https://livebook.manning.com/book/kubernetes-in-action

Cont.

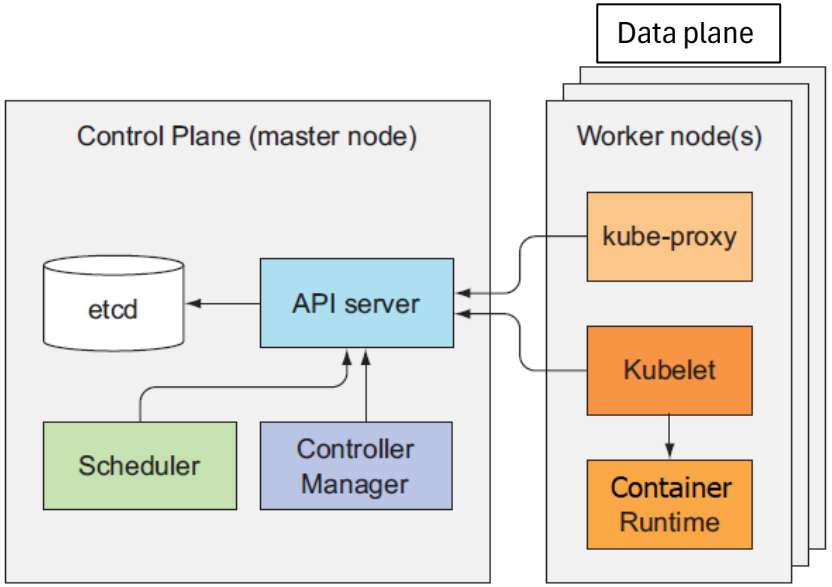
- **Key Components:**
  - **kube-APIserver:**
    The central point that receives API requests (from users, other components, etc.) and communicates the desired state changes.
  - **etcd:**
    A distributed binary key-value database store that holds all the cluster data, including configuration and state.



The name "etcd" combines two concepts: the Unix /etc directory, which stores system configuration files, and "d" for "distributed," reflecting its role as a distributed configuration store.

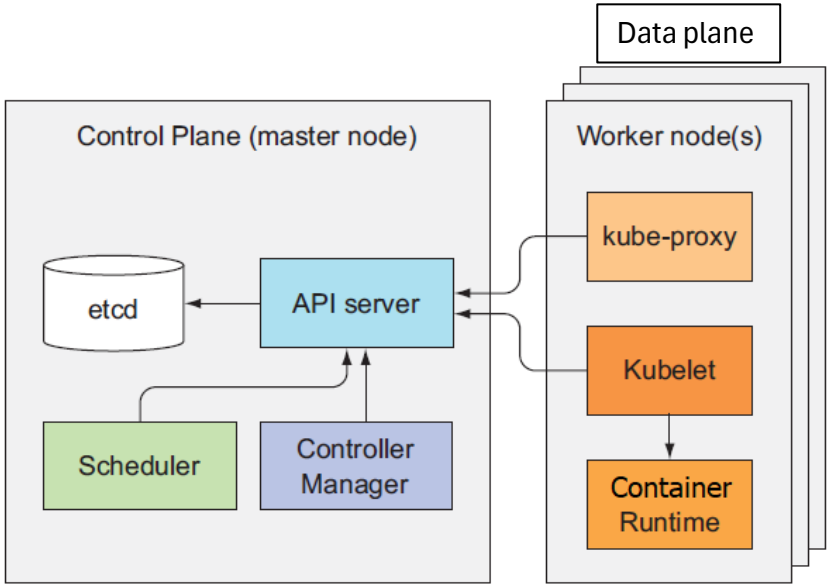
Cont.

- **kube-scheduler:** Determines which available worker node (in the data plane) should run a new Pod based on resource requirements and policies.
- **kube-controller-manager:** Runs various controllers (like the Replica Set, Deployment, and Node controllers) that continuously monitor the cluster and make adjustments to match the desired state.



Cont.

- **Cloud-Controller-Manager (if applicable):**
  Integrates with cloud provider-specific services (such as EKS, load balancers or network configurations).
  One has to write those in CCM so that it can be extended



Cont.

- **Responsibilities:**
  - Monitoring and maintaining cluster state.
  - Scheduling Pods to appropriate nodes.
  - Managing lifecycle events like scaling, updating, and self-healing (restarting failed Pods).

Cont.

- **Data Plane**
- **Purpose:**
  The data plane consists of the **worker nodes** where the actual application workloads run. This is where Pods (which contain one or more containers) are deployed.

Cont.

- **Key Components on Each Worker Node:**
  - **kubelet:**
    An agent that communicates with the control plane, ensuring that containers are running in a **Pod** as specified.
  - **Container Runtime:**
    Software (such as containerd or CRI-O) that pulls container images and runs containers.
  - **kube-proxy:**
    Maintains network rules on nodes to enable communication between Pods, Services, and external traffic.

Cont.

- **Responsibilities of the data plane:**
  - Running application workloads (Pods/containers).
  - Enforcing the network policies and handling service discovery.
  - Reporting node and Pod status back to the control plane.

Kubernetes supports mixed clusters that include both Linux and Windows worker nodes, but with some important caveats.

Limits

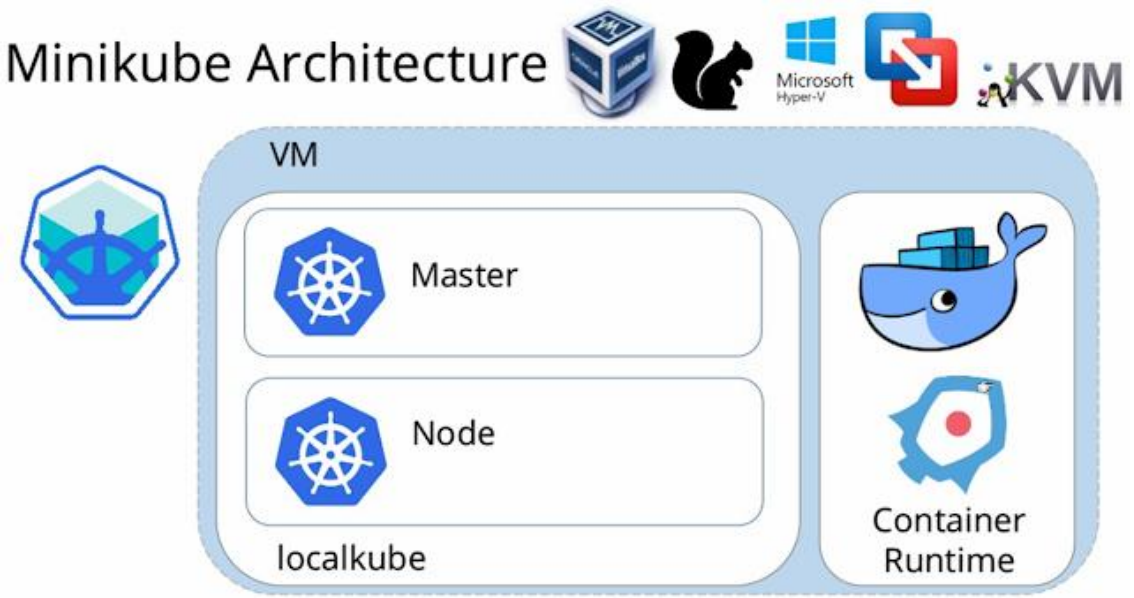
- Kubernetes imposes certain limits to ensure cluster stability:
  - **Pods per Node:** A node can host up to 110 Pods by default.
  - **Total Pods and Containers per Cluster:** Kubernetes is designed to support
    - up to 150,000 total Pods and
    - 300,000 total containers across a cluster.

The kubelet performs **garbage collection** on unused **images every five minutes** and on unused **containers every minute**. You should avoid using external garbage collection tools, as these can break the kubelet behavior and remove containers that should exist.

Courtesy: kubernetes.io

Minikube

- Minikube is a tool that makes it easy to ***run Kubernetes locally*** on your laptop/desktop. Minikube runs a single-node K8s cluster inside a VM on our laptop.



https://blog.codonomics.com/2019/02/loadbalancer-support-with-minikube-for-k8s.html

Deploy a K8s Cluster using Minikube

- Install Minikube
- Install Kubectl

Linux-Only Environment:
Minikube, whether using the Docker driver, VirtualBox, Hyper-V, or WSL 2, creates Linux nodes for your cluster. Therefore, all the pods and containers will run on a Linux OS.

Installing Minikube on a single node

Step 1: Open Your WSL 2 Terminal

- Launch your installed Linux distribution (e.g., Ubuntu) from the Windows Start menu.

  on powershell type *wsl -l –v*

- if Output is

```
      NAME                   STATE           VERSION
  * docker-desktop         Running         2
    docker-desktop-data    Running         2
```

then *Ubuntu* is required to be installed

Installing Minikube on a single node

Step 2: Install Ubuntu (or Another Linux Distribution)

- Open Microsoft Store on Windows.
- Search for Ubuntu (or another preferred Linux distro like Debian).
- Click Install and wait for it to finish.

Installing Minikube on a single node

Step 3: Set Ubuntu as the Default WSL 2 Distribution

on powershell *wsl --set-default Ubuntu*

- Then confirm by checking the installed distributions again:

  on powershell *wsl -l -v*

- You should see

NAME	STATE	VERSION
* Ubuntu	Running	2
docker-desktop	Running	2

Installing Minikube on a single node

Step 4: Open Ubuntu and Install Required Packages

- Open Ubuntu from the Start menu. Run the following commands inside Ubuntu:(it might ask for username/passwd)

$ sudo apt update

$ sudo apt install -y curl

$ curl --version

curl is required because it provides a simple command-line way to send HTTP requests, test services, download resources, and debug networking, which is essential in development, DevOps, and Kubernetes environments.

Installing Minikube on a single node

Step 5: Install Minikube and kubectl in Ubuntu WSL 2

- Now, you can safely follow the normal installation steps: (one by one)

$ curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"

$ chmod +x kubectl

$ sudo mv kubectl /usr/local/bin/

$ kubectl version --client

Installing Minikube on a single node

For Minikube:

$ curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64

$ chmod +x minikube-linux-amd64

$ sudo mv minikube-linux-amd64/usr/local/bin/minikube

$ minikube version

Installing Minikube on a single node

Step 6: Check the Number of Nodes in Minikube

- Run the following command inside your WSL 2 terminal:

$ kubectl get nodes

Expected Output:

- If you started Minikube with a single node (default), you will see something like this:

```
NAME       STATUS   ROLES           AGE   VERSION
minikube   Ready    control-plane,master   10m   v1.28.2
```

Installing Minikube on a single node

Step 7: How to Start Minikube with Multiple Worker Nodes

- If you want to run multiple worker nodes, start Minikube with the --nodes flag:

$ minikube start --nodes=2

Now, check again:

$ kubectl get nodes

Expected Output with 2 Nodes:

NAME	STATUS	ROLES	AGE	VERSION
minikube	Ready	control-plane,master	5m	v1.28.2
minikube-m02	Ready	<none>	5m	v1.28.2

Installing Minikube on a single node

If this expected output is not there then

$ minikube delete

Start Minikube with multiple nodes (e.g., 2 worker nodes):

$ minikube start --nodes=3

Expected Output (With 3 Nodes)

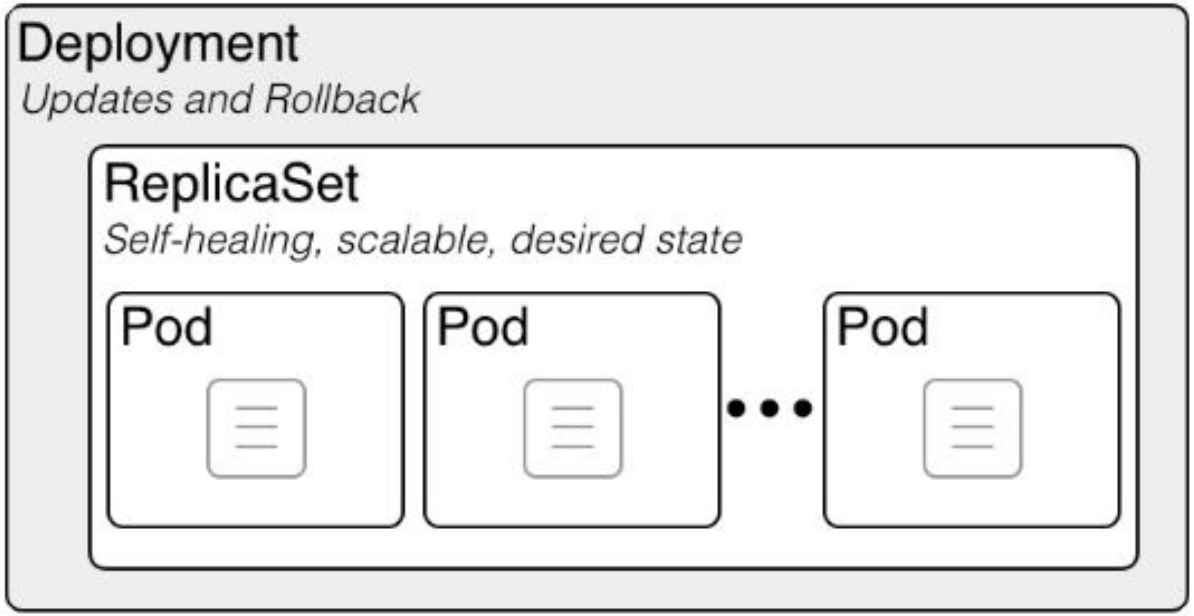
```
NAME           STATUS   ROLES           AGE   VERSION
minikube       Ready    control-plane,master   2m    v1.32.0
minikube-m02   Ready    <none>          2m    v1.32.0
minikube-m03   Ready    <none>          2m    v1.32.0
```

Manifest files

- Kubernetes manifest files are YAML/YML or JSON files that describe objects in our cluster.
- They're the primary way to manage our objects as they let us do version configurations alongside our code, then declaratively apply them to our cluster.

Deployment vs ReplicaSet and Pods

- Scalability, self-healing along with rollbacks and rolling updates are very important feature if we want to use Kubernetes as a cloud



http://wiki.ciscolinux.co.uk/index.php/Kubernetes/Deployment,_ReplicaSet_and_Podery

Auto-healing in action

- We manually delete one of the pods in our deployment and observe how Kubernetes (via the ReplicaSet managed by the Deployment) automatically recreates it to maintain the desired replica count.

Important tips

- Fix permissions in wsl (avoid sudo)
  - *sudo usermod -aG docker $USER*
- If *wsl –l –v* doesn't show minikube.
  - Enter ubuntu and type *minikube version*
- If your directory not seen in wsl type *pwd*
  - It will be either /mnt/c/Users/... OR /home/username
  - If it is second one then no issue otherwise *cs ~* and *pwd*

Auto-healing in action

- Deploy the Application with Multiple Replicas:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
      - name: myapp-container
        image: nginx:latest
        ports:
        - containerPort: 80
```

deployment.yml

Auto-healing in action

$ minikube start --driver=docker --force

If it doesn't work on the other terminal

$ minikube delete

$ kubectl delete deployment my-deployment (if earlier deployment is still alive)

$ nano deployement.yml

$ kubectl apply -f deployment.yml

$ kubectl get pods

You should see some thing like

```
NAME                          READY   STATUS    RESTARTS   AGE
my-deployment-xxxxxxxxxx-abcde   1/1     Running   0          5m
my-deployment-xxxxxxxxxx-fghij   1/1     Running   0          5m
my-deployment-xxxxxxxxxx-klmno   1/1     Running   0          5m
```

Auto-healing in action

$ nano deployement.yml

$ kubectl apply -f deployment.yml

$ kubectl get pods

You should see some thing like

```
NAME                              READY   STATUS    RESTARTS   AGE
my-deployment-xxxxxxxxxx-abcde    1/1     Running   0          5m
my-deployment-xxxxxxxxxx-fghij    1/1     Running   0          5m
my-deployment-xxxxxxxxxx-klmno    1/1     Running   0          5m
```

- Open one more terminal to see how auto healing takes place

Auto-healing in action

Delete One Pod Manually: Choose one pod from the list and delete it:

$ kubectl delete pod my-deployment-xxxxxxxxxx-a

bcde

Observe Auto-Healing: After deleting the pod, check the pods again

$ kubectl get pods

$ minikube stop

Type *$ kubectl get pods -w (watch)* on another open window

Auto-healing in action

ReplicaSet in Action:

- The Deployment manages a ReplicaSet, which is responsible for maintaining the desired number of replicas.
- When you delete a pod, the ReplicaSet notices that the number of running pods is less than desired (3) and automatically creates a new pod to bring the count back to 3.

A Deployment in Kubernetes is an object that manages and maintains a desired number of identical Pods, ensuring they are always running.

Persistent Volume

- **Step 1: Create Volume (PV + PVC):**
  Create pv-pvc.yml file with the following contents then

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-demo
spec:
  capacity:
    storage: 50Mi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: /data/demo

---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-demo
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 50 Mi
```

- *kubectl apply -f pv-pvc.yml* *//unlike docker where we use create here we use apply command –f is file*

Persistent Volume

- **Step 2: Pod 1 (Write data):**
  Create pod1.yml file with the following contents then

```
apiVersion: v1
kind: Pod
metadata:
  name: pod1
spec:
  restartPolicy: Never
  containers:
  - name: c1
    image: busybox
    command: ["sh","-c","echo Hello Students > /data/file.txt"]
    volumeMounts:
    - name: data
      mountPath: /data
  volumes:
  - name: data
    persistentVolumeClaim:
      claimName: pvc-demo
```

- *kubectl apply -f pod1.yml*
- *kubectl get pods //verify*

Persistent Volume

- **Step 2: Pod 1 (Write data):**
  Create pod1.yml file with the following contents then

```
apiVersion: v1
kind: Pod
metadata:
  name: pod1
spec:
  restartPolicy: Never
  containers:
  - name: c1
    image: busybox
    command: ["sh","-c","echo Hello Students > /data/file.txt"]
    volumeMounts:
    - name: data
      mountPath: /data
  volumes:
  - name: data
    persistentVolumeClaim:
      claimName: pvc-demo
```

- *kubectl apply -f pod1.yml*
- *kubectl get pods //verify*
- *kubectl delete pod pod1*

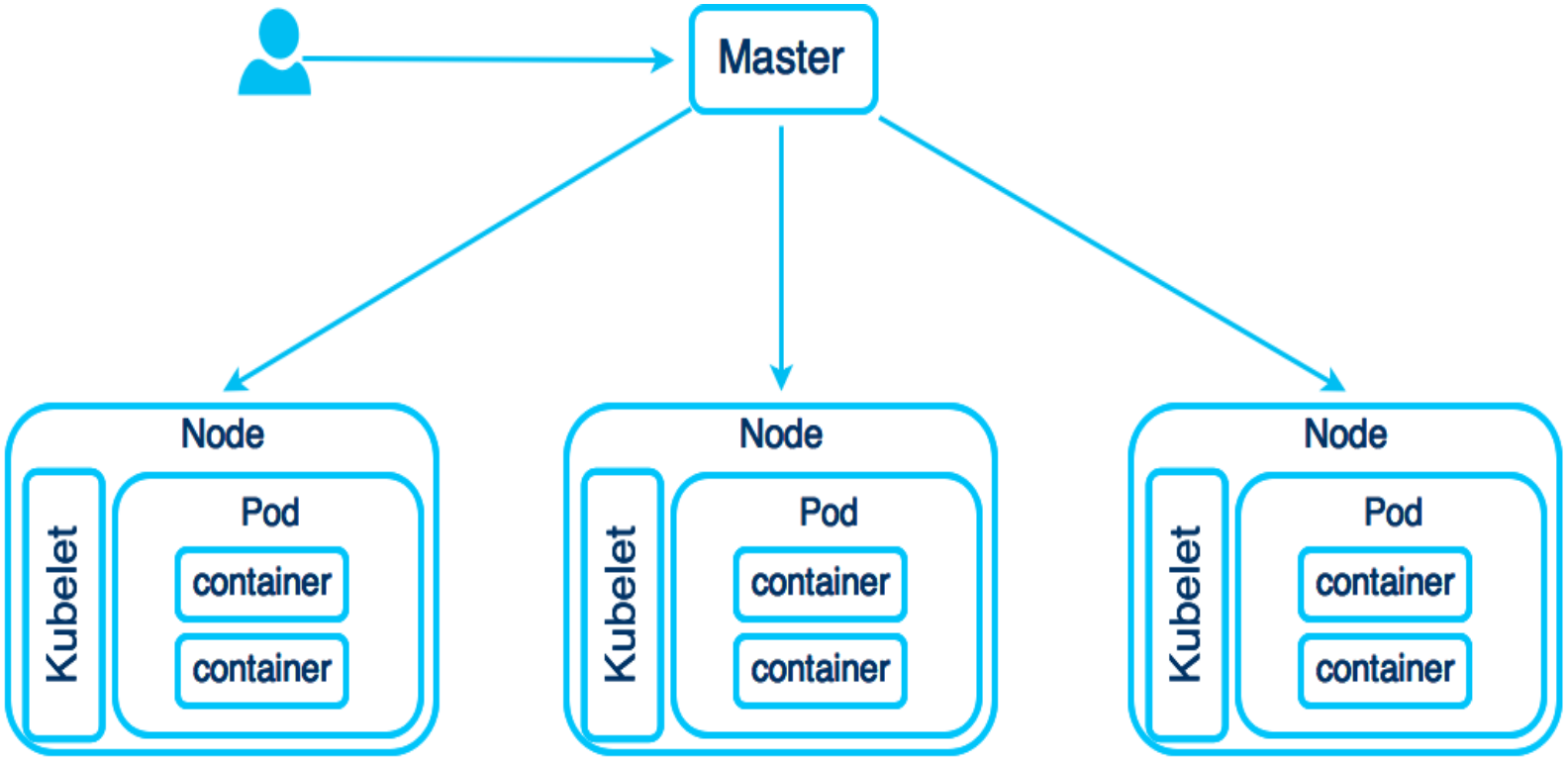
Persistent Volume

- **Step 3: Create Pod 2 (read data):**
- Create pod2.yml file with the following contents then

```
apiVersion: v1
kind: Pod
metadata:
  name: pod2
spec:
  restartPolicy: Never
  containers:
  - name: c2
    image: busybox
    command: ["sh","-c","cat /data/file.txt; sleep 3600"]
    volumeMounts:
    - name: data
      mountPath: /data
  volumes:
  - name: data
    persistentVolumeClaim:
      claimName: pvc-demo
```

- *kubectl logs pod2*
- *kubectl get pods //verify*
- *kubectl delete pod pod2*

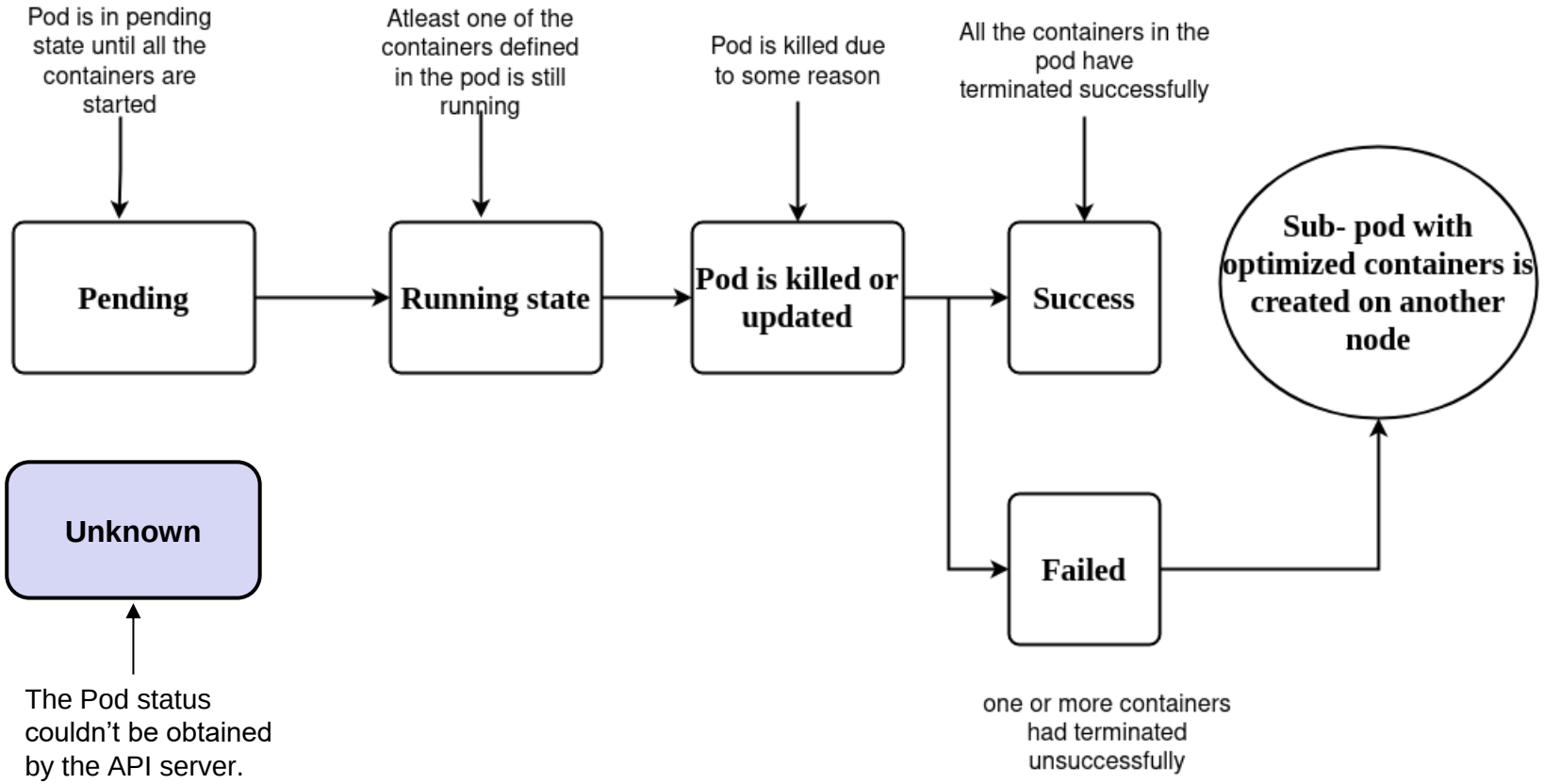
Structure of a Pod with containers in Kubernetes



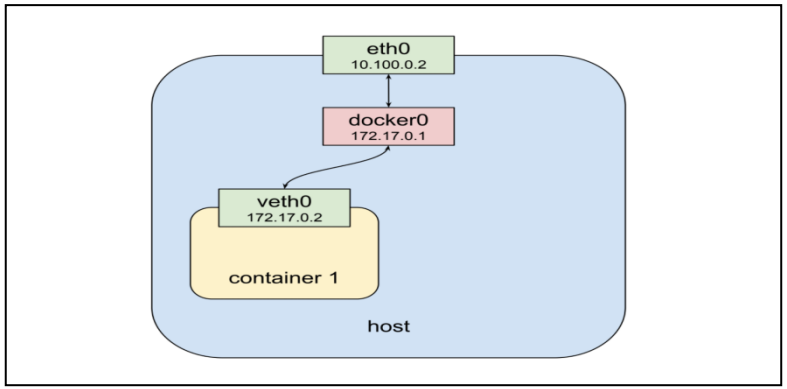
IMPORTANT

Kubelet is a node-level supervisor within a Kubernetes cluster, responsible for the proper execution and management of containers. It ensures that the node maintains the desired state as defined by the Kubernetes active controller.

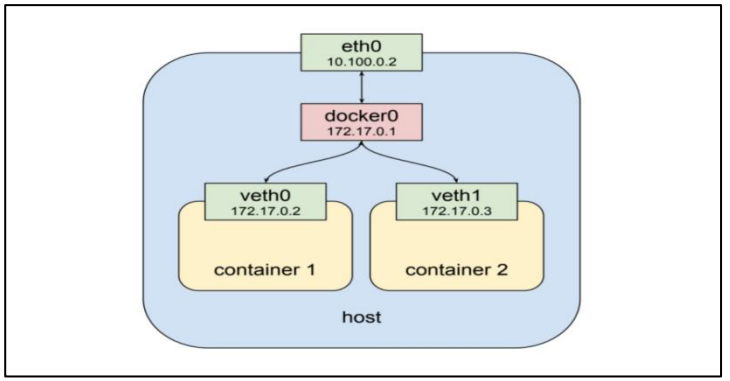
Life cycle of a Pod



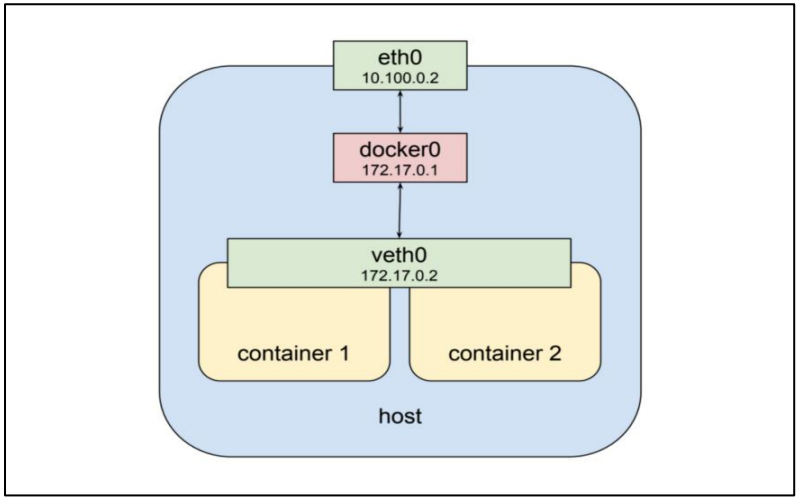
Networking in Pods (linux)



Single pod with 1 container

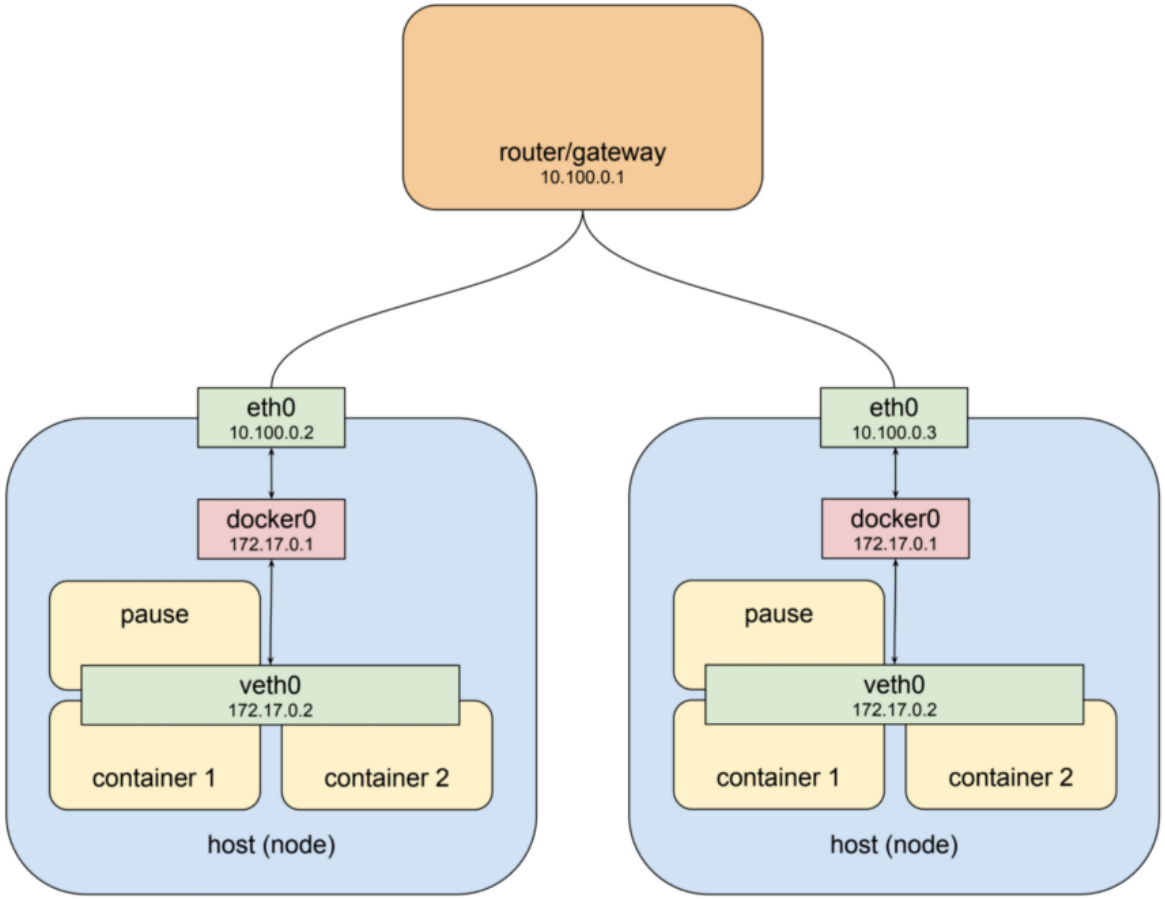


Pod with two containers with own network interface



Pod with two containers with shared interface

Pods with pause containers



https://medium.com/google-cloud/understanding-kubernetes-networking-pods-7117dd28727

How traffic gets into Kubernetes Pod from internet