Pod:

In Kubernetes, a **Pod** is the smallest and most basic unit of deployment. It represents a single instance of a running application or service. A Pod can contain one or more containers, which share the same network namespace, storage, and other resources.

Functionality of a Pod:

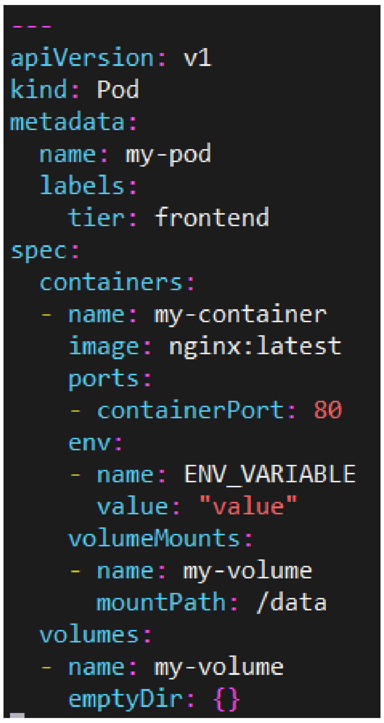
1. **Encapsulation of Containers**: Pods encapsulate one or more containers, providing a logical grouping of related application components.
2. **Shared Network Namespace**: Containers within the same Pod share the same network namespace, allowing them to communicate with each other over localhost.
3. **Shared Storage**: Pods can mount shared volumes, enabling containers within the same Pod to share data.
4. **Pod Lifecycle Management**: Kubernetes manages the lifecycle of Pods, including creation, scaling, termination, and rescheduling.
5. **Service Discovery and Load Balancing**: Pods are assigned a unique IP address and DNS hostname, making it easy for other Pods to discover and communicate with them. Kubernetes also provides built-in load balancing for Pods within a Service.
6. **Health Checking and Self-Healing**: Kubernetes continuously monitors the health of Pods and automatically restarts or reschedules them if they fail or become unresponsive.
7. **Resource Management**: Kubernetes allows you to specify resource requirements and limits for Pods, ensuring fair allocation of CPU and memory resources.

We can create simple pods just by ad-hoc command or from a pod-definition file:

- kubectl run my-pod --image nginx

or we can create a pod-definition file which is written in YAML

POD-DEFINITION FILE IN DETAIL



```yaml
---
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
  labels:
    tier: frontend
spec:
  containers:
  - name: my-container
    image: nginx:latest
    ports:
    - containerPort: 80
    env:
    - name: ENV_VARIABLE
      value: "value"
    volumeMounts:
    - name: my-volume
      mountPath: /data
  volumes:
  - name: my-volume
    emptyDir: {}
```

Example: pod-definition.yaml in current directory

Commands

- kubectl get pods
- kubectl get pods -o wide
- kubectl get pods -n
- kubectl get pods -A
- `kubectl get pods --selector app=nginx`
- kubectl describe pod [pod name]
- kubectl run [pod name] –image [image name]
- kubectl run web-app --image nginx --port 80 --env “ENV_NAME=value” --labels “tier=frontend,end=test”
- kubectl run -it [pod name] --image [image name]
- kubectl logs pod-name
- kubectl create -f pod-definition.yaml
- kubectl apply -f pod-definiton.yaml
- kubectl delete -f pod-definition.yaml
- kubectl edit pod pod-name
-

POD’s Life Cycle

The lifecycle of a pod in Kubernetes consists of several phases, from creation to termination.

1. **Pending Phase**
   - ➔ When we create the pod, it enters the pending phase initially.
   - ➔ During this phase, Kubernetes scheduler looks for a suitable node to schedule the pod into. The pod has been accepted by the kube-apiserver but the necessary resources(CPU, memory) haven’t been allocated yet, and the container image haven’t been pulled onto the node.

2. **Container Creating Phase:**
   - ➔ After the pod is assigned to a node, it enters the container creating phase.
   - ➔ During this phase, the kube-scheduler has assigned the pod to a node and the container runtime (CRI such as containerD) is pulling the container image specified in the pod’s configuration onto the node. This phase ends when all containers in the pod have been created.

3. **Running Phase**
   - ➔ Once all containers in the pod have been created and started, the pod enters the running phase.
   - ➔ During this phase all containers in the pod are running and actively processing requests or performing their specified tasks, if any container in the pod fails or exits, Kubernetes restarts it according to the pod’s restart policy. (i.e. always, onFailure, etc)

4. **Succeeded Phase**
   - ➔ If all containers in the pod exit with a success status, or have terminated in success, and will not be restarted then the pod enters the succeeded phase.
   - ➔ This phase indicated that all containers in the pod have completed their tasks successfully and exited. Pods in this phase are not automatically deleted by default, but they can be configured to be automatically cleaned up.

5. **Failed Phase**
   - ➔ If any container in the pod exits with a failure status or non zero exit code, the pod enters the failed phase.
   - ➔ This phase indicates that at least one container in the pod has encountered an error or failed to start. Pods in this phase are not automatically deleted by default, but they can be configured to be automatically cleaned up.

6. **Unknown Phase**
   - ➔ For some reason the state of the pod could not be obtained. This phase typically occurs due to an error in the communicating with the node where the pod should be running.
   - ➔ This phase indicates that Kubernetes is unable to determine the current state of the pod. Pods in this phase are not deleted by default but they can be configured to be automatically cleaned up.

Container States:

Waiting:

If a container is not in either the Running or Terminated state, it is Waiting. A container in the Waiting state is still running the operations it requires in order to complete start up: for example, pulling the container image from a container image registry, or applying Secret data. When you use kubectl to query a Pod with a container that is Waiting, you also see a Reason field to summarize why the container is in that state.

Running:

The Running status indicates that a container is executing without issues. If there was a postStart hook configured, it has already executed and finished. When you use kubectl to query a Pod with a container that is Running, you also see information about when the container entered the Running state.

Terminated:

A container in the Terminated state began execution and then either ran to completion or failed for some reason. When you use kubectl to query a Pod with a container that is Terminated, you see a reason, an exit code, and the start and finish time for that container's period of execution.If a container has a preStop hook configured, this hook runs before the container enters the Terminated state.