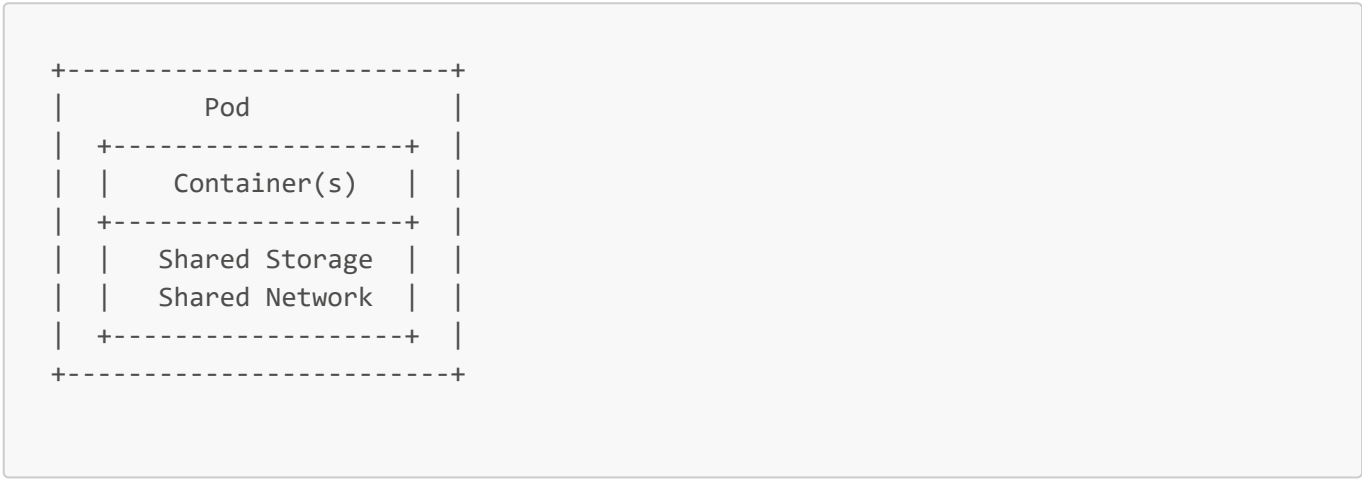
Pods

A Pod is the smallest and simplest deployable unit in Kubernetes.

- Pod = 1 or more containers + storage + network + configuration

So, a Pod is a wrapper around containers (usually just one), giving them:

- A unique IP address
- Shared storage
- Shared networking
- Configuration (like environment variables)



Why are Pods important?

Reason	Explanation
Fundamental Unit	All applications in Kubernetes run inside Pods — whether it's a single container or multiple working together.
Manages Lifecycle	Kubernetes manages the lifecycle of Pods — creating, restarting, or removing them.
Encapsulation	Pods encapsulate all the resources a containerized app needs to run.
Building Block for Higher Resources	Deployments, ReplicaSets, StatefulSets, and DaemonSets all **manage Pods** under the hood.

How Pods are linked to other Kubernetes components

Component	Relationship to Pod
Deployment	Manages ReplicaSets → which manage multiple Pods
ReplicaSet	Ensures a specific number of Pod replicas are running
StatefulSet	Manages Pods with **stable identity** (for databases, etc.)
DaemonSet	Ensures a Pod runs on **every node**

Component	Relationship to Pod
Job / CronJob	Create Pods for batch or scheduled tasks
Service	Exposes Pods on the network (load balances traffic to Pods)
ConfigMap / Secret	Injects config or sensitive data into Pods
PVC / Storage	Mounts volumes inside Pods for persistent data
NetworkPolicy	Controls which Pods can talk to each other
Probes	Check Pod health (liveness/readiness)
ServiceAccount	Provides Pods with permissions (RBAC)

How to write a Pod YAML (Step-by-Step)

1. start with 4 key section

- Every YAML start with 4 key sections

```
apiVersion: v1
kind: Pos
metadata:
spec:
```

Field	Purpose
`apiVersion`	Specifies the version of the Kubernetes API you're using (`v1` for core)
`kind`	Tells Kubernetes what object you're defining (e.g., `Pod`, `Deployment`)
`metadata`	Provides information like name, labels, annotations
`spec`	Describes what the object should do or look like

2. define metadata

```
metadata:
  name: nginx-pod
  labels:
    app: nginx
```

- The name is used when running commands like kubectl get pod nginx-pod
- labels help Services or Deployments select Pods using selectors

Key	Purpose
`name`	A **unique identifier** for the Pod within the namespace

Key	Purpose
`labels`	Key-value pairs used to **organize, group, and select** resources

3. define spec

```
spec:
  containers:
    - name: nginx-container
      image: nginx:latest
      ports:
        - containerPort: 80
```

Field	Purpose
`containers`	A list (even for one container) — you can run multiple inside a Pod
`name`	Unique name of the container (inside this Pod)
`image`	The Docker image to run (`nginx:latest` is pulled from Docker Hub)
`containerPort`	Exposes port **inside the container** to the Pod's network

4. full working pod yaml

```
apiVersion: v1             # API group/version for core objects like Pod
kind: Pod                  # This is a Pod resource
metadata:
  name: nginx-pod          # Unique name of the Pod in the namespace
  labels:
    app: nginx             # Label to group this Pod with others
spec:
  containers:
    - name: nginx-container     # Name of the container (must be unique inside the
Pod)
      image: nginx:latest       # Container image to run (nginx from Docker Hub)
      ports:
        - containerPort: 80     # Container listens on port 80 (usually for web
servers)
```

Example:

```
# Step 1: Define the API version to use
apiVersion: v1
# This tells Kubernetes which version of the API you're using. 'v1' is used for
core resources like Pod.
```

```yaml
# Step 2: Define the kind of resource
kind: Pod
# This tells Kubernetes you're defining a Pod. Other kinds include Deployment,
Service, etc.

# Step 3: Metadata - name and labels for identification
metadata:
  name: my-nginx-pod
  # 'name' must be unique within a namespace. Used to identify the Pod.
  labels:
    app: nginx
    tier: frontend
  # Labels are key-value pairs to tag your Pod. Useful for grouping, selecting, or
organizing.

# Step 4: Spec - Define what the Pod should contain and how it should behave
spec:
  containers:
    - name: nginx-container
      # Name of the container. Must be unique within this Pod.
      image: nginx:latest
      # Docker image to run. Here, using the latest official Nginx image.
      ports:
        - containerPort: 80
        # Port exposed by the container (usually the HTTP port for web servers).
```

Pod Lifecycle

Phase	Description
`Pending`	Pod accepted but not yet running
`Running`	All containers are running
`Succeeded`	All containers exited successfully
`Failed`	One or more containers failed
`Unknown`	State couldn't be determined

Commands to Interact with Pods

Action	Command
List Pods	`kubectl get pods -n <namespace>`
Describe a Pod	`kubectl describe pod <pod-name> -n <namespace>`
View Pod logs	`kubectl logs <pod-name> -n <namespace>`
Exec into Pod shell	`kubectl exec -it <pod-name> -n <namespace> -- /bin/bash`
Delete a Pod	`kubectl delete pod <pod-name> -n <namespace>`

1. `kubectl get pods -n <namespace>`

- list all the pods in specifi namespace

```
#input command
kubectl get pods -n dev

#output
NAME         READY   STATUS    RESTARTS   AGE
nginx-pod    1/1     Running   0          2m
```

2. `kubectl describe pod <pod-name> -n <namespace>`

- Show detailed information about a Pod, including events, IP, node, image, ports, volume mounts, and status.
- This command is imp for debugging issue and show why pod is pending, crashLoopBackoff

```
#input
kubectl describe pod nginx-pod -n dev

# output
- Labels and annotations
- Container image
- Pod IP and node
- Events (warnings, restarts)
```

3. `kubectl logs <pod-name> -n <namespace>`

- Provide log from running container
- Useful for debugging the application
- Able to see the error, request log and startup output

```
kubectl logs nginx-pod -n dev
```

4. `kubectl exec -it <pod-name> -n <namespace> -- /bin/bash`

- Open an interactive shell inside a running container.
- Useful for inspecting files, environment variables, running processes and act like SSH into container.

```
kubectl exec -it nginx-pod -n dev -- /bin/bash
```

5. `kubectl delete pod <pod-name> -n <namespace>`

- Delete specific pod from namespace.

```
kubectl delete pod nginx-pod -n dev
```

---

yaml in k8s : k8s definition file always conatin 4 properties : apiversions, kind, metadata, spec