Pod (Basic Workload Unit)

A **Pod** is the smallest deployable unit in Kubernetes.
It:

- Runs one or more containers (usually 1)
- Shares network & storage among containers
- Has a single IP address
- Is *not self-healing* (if it dies, it's gone)

Pod YAML example:

```
apiVersion: v1
kind: Pod
metadata:
  name: simple-pod
spec:
  containers:
  - name: app
    image: nginx
```

Pods are ephemeral

- If a pod crashes → it's gone
- If a Node dies → its pods disappear
- Kubernetes does **not** bring a standalone Pod (created via `kubectl run`) back automatically

This is why ReplicaSets exist.

ReplicaSet (Ensures the number of Pods stays constant)

A **ReplicaSet (RS)** ensures "I always want *X* copies of this Pod running."

If a pod dies → RS creates a new one
If more pods exist than desired → it deletes extras

ReplicaSet **does not** handle:

- rolling updates
- undoing changes
- multi-step rollout strategies

ReplicaSets are therefore *not* usually created directly by engineers — Deployments create them for you.

ReplicaSet YAML example:

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: simple-rs
spec:
  replicas: 3
  selector:
    matchLabels:
      app: demo
  template:
    metadata:
      labels:
        app: demo
    spec:
      containers:
      - name: app
        image: nginx
```

Deployment – The Master Controller

A **Deployment** is the recommended way to manage stateless applications in Kubernetes.

It is responsible for:

✔ Creating & owning ReplicaSets

✔ Rolling updates (seamless version upgrades)

✔ Rollback to previous revisions

✔ Pause/resume rollout for debugging

✔ Self-healing (via RS)

✔ Declarative updates

This makes Deployment the powerhouse controller in Kubernetes.

A Basic Deployment (Start Here)

```
apiVersion: apps/v1
```

```
kind: Deployment
metadata:
  name: webapp
spec:
  replicas: 3
  selector:
    matchLabels:
      app: webapp
  template:
    metadata:
      labels:
        app: webapp
    spec:
      containers:
      - name: app
        image: nginx:1.27
        ports:
        - containerPort: 80
```

This Deployment ensures:

- 3 pods always exist
- When updating the image, a new ReplicaSet is created
- The old ReplicaSet is scaled down & preserved for rollback

Deployment Workflow – What Happens Internally

Let’s analyze what happens when you run:

`kubectl apply -f deployment.yaml`

Kubernetes performs **all** of these steps:

STEP 1 — Request hits API Server

The YAML is submitted to the **kube-apiserver**.
It validates:

- apiVersion
- schema
- RBAC permissions
- admission controllers (OPA, PSP, mutating/validating webhooks)

If valid → stored in **etcd** as the Deployment object.

STEP 2 — Deployment Controller wakes up

Running inside `kube-controller-manager`.

It checks:

- Is there an existing RS for this deployment?
- Does it match the Pod template hash?

If no → create a new ReplicaSet.
If yes → update RS.

STEP 3 — ReplicaSet ensures correct number of Pods exist

- Desired replicas = 3
- RS creates Pods until 3 exist
- If pods die → RS recreates them

STEP 4 — Scheduler assigns Pods to Nodes

Scheduler:

- sees pending pods
- picks optimal node based on CPU, memory, taints, affinity, etc.
- binds pod to node

STEP 5 — Kubelet pulls image & runs container

On the selected node, kubelet:

1. Pulls the image
2. Creates a Pod sandbox (pause container)
3. Starts containers inside the Pod

Deployment Rollout Strategy (Very Important)

Kubernetes supports two rollout strategies:

1. RollingUpdate (Default)

Gradually replace old pods with new ones.

```
maxUnavailable: 25%
maxSurge: 25%
```

2. Recreate

Delete all old pods first, then create new pods.

Used for:

- Stateful apps
- Apps that cannot run two versions simultaneously

How Deployment Data Is Stored in etcd (Deep Internals)

This part is usually ignored in most tutorials — but extremely important if you want to understand Kubernetes at a systems level.

Everything in Kubernetes is stored as key-value pairs in etcd

Deployments, Pods, ReplicaSets, Services, Nodes, Events — everything.

How Deployment Object Is Saved

A Deployment is stored under a path like:

`/registry/deployments/<namespace>/<deployment-name>`

Example:

`/registry/deployments/default/webapp`

The value stored is a serialized **protobuf** (or JSON if using older versions).

It contains:

- Deployment spec
- Rolling update strategy
- Revision number
- Replicas count
- Pod template
- Labels & selectors
- OwnerReferences
- Status subresource (updatedReplicas, availableReplicas, etc.)

ReplicaSets also stored similarly

`/registry/replicasets/default/webapp-7f5d8498d4`

ReplicaSets store:

- Desired replicas
- Template hash
- OwnerReference pointing to the Deployment

- Pods also stored in etcd

`/registry/pods/default/webapp-7f5d8498d4-l29bx`

Each Pod entry includes:

- pod spec
- nodeName (once scheduled)
- podStatus (Ready, Running, ContainerStatuses)
- restart counts
- events (separate resource)

How Rollouts & Rollbacks Are Tracked in

etcd

Deployments maintain **revisions**.
Each revision is stored as an annotation:

```
deployment.kubernetes.io/revision: "3"
```

Every update creates:

- a new ReplicaSet
- with its own revision number
- stored as separate entries in etcd

Rollback simply makes the Deployment point to an older revision and adjust RS scaling.

How Deployment Self-Healing Works

Because everything is stored in etcd:

- Controller loops continuously compare “desired state” (stored in etcd) with “actual state” (from kubelets)
- If mismatch → controller fixes it

For example:

If a Pod disappears:

- Controller sees desired replicas = 3
- Actual replicas = 2
- A new Pod is created immediately

This is Kubernetes' core reconciliation loop.

Updating a Deployment (Hands-On Example)

```
kubectl set image deployment/webapp app=nginx:1.28
```

What happens:

1. New Pod template hash generated

2. New ReplicaSet created
3. Old ReplicaSet scaled down
4. New Pods created gradually
5. All stored in etcd

Check rollout status:

```
kubectl rollout status deployment/webapp
```

View revision history:

```
kubectl rollout history deployment/webapp
```

Rollback:

```
kubectl rollout undo deployment/webapp
```

Pausing a Rollout (Useful for Checking Canary Releases)

```
kubectl rollout pause deployment/webapp
```

Make changes:

```
kubectl set image ...
kubectl edit deployment webapp
```

Resume:

```
kubectl rollout resume deployment/webapp
```

Scaling a Deployment

```
kubectl scale deployment webapp --replicas=10
```

This updates:

- Deployment spec in etcd
- Desired replicas
- ReplicaSet creates extra pods

Deleting a Deployment

```
kubectl delete deployment webapp
```

This triggers:

- RS deletion
- Pod deletion
- All keys removed from etcd