ReplicaSet — Deep Technical Tutorial

A **ReplicaSet (RS)** is a Kubernetes controller responsible for ensuring a **specified number of Pod replicas** are always running.

A ReplicaSet **continuously compares:**

```
desired state (spec.replicas)
vs
current state (number of pods matching label selector)
```

If mismatch:

- It **creates** Pods.
- It **deletes** excess Pods.
- It **replaces** failed Pods.

It does this via the **Kubernetes control plane**, mostly through:

- kube-apiserver
- kube-controller-manager
- kube-scheduler
- kubelet
- etcd

----

1. Let’s Start With a YAML

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

You run:

```
kubectl apply -f nginx-rs.yaml
```

---

2. What Happens? — Full Control Plane Flow

Step 1: kubectl → API Server

`kubectl` sends the ReplicaSet definition as a REST request to:

```
kube-apiserver:443/apis/apps/v1/replicasets
```

The API server:

1. Authenticates (AuthN)
2. Authorizes (RBAC)
3. Admits (Validating/Mutating Webhooks)
4. Stores the ReplicaSet object in **etcd**

etcd state after storing ReplicaSet:

```
/registry/replicasets/default/nginx-rs
     - spec.replicas = 3
     - selector = app=web
     - pod template stored here
```

---

3. ReplicaSet Controller Starts Acting

Running inside **kube-controller-manager**, the **replicaset controller** watches the API server via a watch stream.

It receives an event:

```
ADDED ReplicaSet "nginx-rs"
```

The controller compares:

```
current pod count = 0
desired pod count = 3
```

So it decides to **create 3 Pods**.

It sends Pod objects to the API server:

```
POST /api/v1/namespaces/default/pods
```

Each Pod uses the template stored inside the ReplicaSet.

----

4. etcd After Pod Creation

Three pod entries appear:

```
/registry/pods/default/web-abc12
/registry/pods/default/web-def34
/registry/pods/default/web-ghi56
```

Within each Pod object:

- `metadata.ownerReferences` points to the ReplicaSet
- status.phase = Pending
- spec.nodeName = "" (no node assigned yet)

----

5. Scheduler Now Gets Involved

The **kube-scheduler** continuously watches the API server for **Pending** pods with no node assigned.

It receives:

```
ADDED Pod web-abc12
ADDED Pod web-def34
ADDED Pod web-ghi56
```

For each pod:

1. Scheduler runs its scheduling logic (fit + priority).
2. Selects a node (example: node1).
3. Writes **binding** back to API server:

```
POST /api/v1/namespaces/default/pods/web-abc12/binding
spec.nodeName = node1
```

etcd after scheduling:

Each pod gets updated:

```
spec.nodeName = node1 (or node2, node3 ...)
status.phase = Scheduled (not Running yet)
```

----

6. Kubelet on Each Node Takes Over

Each worker node's **kubelet** watches:

```
/api/v1/pods?fieldSelector=spec.nodeName=<this node>
```

When kubelet sees the Pod assigned to its node:

1. It pulls the image
2. Creates container via container runtime (Docker, containerd, CRI-O)
3. Starts it
4. Updates the API server with pod status:

```
status.phase = Running
```

This updated status is stored in etcd.

ReplicaSet Controller Continuous

Reconciliation Loop

The ReplicaSet controller ALWAYS runs this loop:

Read from API server:

```
List Pods matching selector app=web
```

Compare:

- desired: 3
- current: 3

Equal → do nothing.

Failure Scenarios — How ReplicaSet Reacts

Pod Crashes

kubelet updates:

```
status.phase = Failed
```

ReplicaSet sees:

```
desired = 3
running = 2
```

→ Creates a new Pod object.

Node Dies

If node stays **NotReady** for ~5 minutes, node controller marks all pods as:

```
status.phase = Unknown
```

Then the ReplicaSet controller deletes those pods and creates replacements.

You manually delete a Pod

If deletion is not with `--cascade=orphan`, the RS sees:

```
2 running
desired 3
```

→ Creates 1 new pod.

🗑️ If You Delete the ReplicaSet Itself

You run:

```
kubectl delete rs nginx-rs
```

The API server marks RS for deletion → stored in etcd.

The ReplicaSet controller receives:

```
DELETED ReplicaSet nginx-rs
```

It deletes all **owned pods**.

All pod entries are removed from etcd.

Putting It All Together — ETCD ILLUSTRATION

Below is a simplified view of how objects exist in etcd:

```
/registry
 ├── replicasets
 │    └── default
 │         └── nginx-rs
 │               - spec.replicas = 3
 │               - selector: app=web
 │               - podTemplate
 │
 ├── pods
 │    └── default
 │         ├── web-abc12
 │         │      - ownerRef = nginx-rs
 │         │      - spec.nodeName = node1
 │         │      - status: Running
 │         ├── web-def34
 │         └── web-ghi56
 │
 └── nodes
      └── node1
      └── node2
      └── node3
```

ReplicaSet Summary (Deep Insight)

Component	Role
kubectl	Sends ReplicaSet YAML to API server
API Server	Validates and stores object in etcd
etcd	Stores desired state for RS and Pods; stores actual pod status
ReplicaSet Controller	Detects pod count mismatch and creates/deletes pods
Scheduler	Assigns pods to nodes
Kubelet	Creates, manages containers on node; reports status
Container Runtime	CRI-O/Docker/containerd actually runs containers

ReplicaSets are purely **controllers** — they never run pods; they only ensure correct count.