Kubernetes Controllers

**🌟 Kubernetes Controllers: Overview**

Controllers are part of the **Kubernetes Control Plane**. They:

* Continuously monitor resources (like Pods, Deployments, etc.)
* Compare the actual state to the desired state (declared in YAML)
* Take corrective actions (e.g., start/stop/create/delete resources)

There are several types of controllers:

* **ReplicationController**
* **ReplicaSet**
* **Deployment**
* **StatefulSet**
* **DaemonSet**
* **Job / CronJob**

**A screenshot of a computer

AI-generated content may be incorrect.**

Let’s go back to our first scenario where we had a single POD running our application. For some reason, our application crashes and the POD fails. Then, users will no longer be able to access our application. To prevent users from losing access to our application, we would like to have more than one instance or POD running at the same time.

The replication controller helps us run multiple instances of a single POD in the Kubernetes cluster thus providing High Availability.

**📌 ReplicationController (RC)**

The **ReplicationController** ensures that a specified number of **identical Pods** are running at any given time.

🔁 It will create new pods if there are fewer than desired or delete excess pods if there are more than desired.

A screen shot of a computer

AI-generated content may be incorrect.

**🔍 Key Fields**

| **Field** | **Description** |
| --- | --- |
| replicas | Desired number of pod copies |
| selector | Labels used to identify pods managed by this RC |
| template | Pod template to define the pod to be replicated |

**⚙️ How It Works**

* The RC continuously watches for pods with the matching label.
* If a pod dies (node failure, crash, etc.), the RC creates a new one.
* If someone manually creates a pod with the same label, it counts towards the desired replica count.
* If more pods exist than needed, it deletes extra ones.

Q.) But why would We have to specify what PODs fall under it, if we have provided the contents of the pod-definition file itself in the template?

It’s because, replica set can also manage pods that were not created as part of the ReplicaSet creation. Say for example, there were pods created BEFORE the creation of the ReplicaSet that match the labels specified in the selector, the replica set will also take those pods into consideration when creating the replicas.

The selector is not a REQUIRED field in case of a replication controller, but it is still available. When we skip it, it assumes it to be the same as the labels provided in the pod-definition file.

In case of replica set a user input is required for this property. And it must be written in the form of matchLabels. The matchLabels selector simply matches the labels specified under it to the labels on the PODs. The ReplicaSet selector also provides many other options for matching labels that were not available in a replication controller.

**⚠️ Limitations**

* **Lacks support for set-based label selectors** (uses equality-based selectors only)
* **Deprecated in favor of ReplicaSet** in newer Kubernetes versions
* Not integrated with **Deployments**, which are the recommended way to manage application updates

**✅ ReplicaSet (RS)**

**ReplicaSet** is the **next-generation** replication controller. It does the same thing — maintains a stable set of **replica pods running at any given time** — but with more powerful **label selector** capabilities.

A screen shot of a computer program

AI-generated content may be incorrect.

**🔍 Key Fields**

| **Field** | **Description** |
| --- | --- |
| replicas | Desired number of pod replicas |
| selector.matchLabels | Pods with matching labels will be managed |
| template | Pod configuration |

**⚙️ How It Works**

* Just like RC, it maintains the desired number of Pods.
* Can use more advanced selectors:  
  A screen shot of a computer program

  AI-generated content may be incorrect.

**🚀 Used by Deployments**

* While **ReplicaSet** can be used directly, it is rarely done in practice.
* **Deployments** manage ReplicaSets internally and add features like:
  + Rolling updates
  + Rollbacks
  + Revision history

🔄 Summary: RC vs RS

| **Feature** | **ReplicationController** | **ReplicaSet** |
| --- | --- | --- |
| Status | Legacy | Current standard |
| Label Selectors | Equality-based only | Equality + Set-based |
| Recommended for use? | ❌ No | ✅ Yes |

**Q.) How labels and selectors helps ReplicaSet in Kubernetes?**

**A screenshot of a computer

AI-generated content may be incorrect.**

Say we deployed 3 instances of our frontend web application as 3 PODs. We would like to create a replication controller or replica set to ensure that we have 3 active PODs at any time.   
  
There could be 100s of other PODs in the cluster running different application. By adding selectors, ReplicaSet knows which pods to monitor.

Q.) We have 3 existing PODs that were created already, and we need to create a replica set to monitor the PODs to ensure there are a minimum of 3 always running. When the replication controller is created, it is NOT going to deploy a new instance of POD as 3 of them with matching labels are already created.   
In that case, do we really need to provide a template section in the replica-set specification, since we are not expecting the ReplicaSet to create a new POD on deployment?

Yes, we do, because in case one of the PODs were to fail in the future, the ReplicaSet needs to create a new one to maintain the desired number of PODs. And for the replica set to create a new POD, the template definition section is required.

**Q.) Does that mean we can’t use a replication controller if we plan to have a single POD?**No! Even if we have a single POD, the replication controller can help by automatically bringing up a new POD when the existing one fails. Thus, the replication controller ensures that the specified number of PODs are always running. Even if it’s just 1 or 100.

**A diagram of a system

AI-generated content may be incorrect.**

Another reason we need replication controller is to create multiple PODs to share the load across them.

For example, in this simple scenario we have a single POD serving a set of users. When the number of users increase, we deploy additional POD to balance the load across the two pods. If the demand further increases and if we were to run out of resources on the first node, we could deploy additional PODs across other nodes in the cluster. As We can see, the replication controller spans across multiple nodes in the cluster. It helps us balance the load across multiple pods on different nodes as well as scale our application when the demand increases.