Star 898 Fork 152 Watch 52 Follow @collabnix 0

# Kubernetes ReplicaSet

Kubernetes - Beginners | Intermediate | Advanced

View on GitHub Join Slack

## Kubernetes ReplicaSet

- ReplicaSets are Kubernetes controllers that are used to maintain the number and running state of pods.
- It uses labels to select pods that it should be managing.
- A pod must labeled with a matching label to the ReplicaSet selector, and it must not be already owned by another controller so that the ReplicaSet can acquire it.
- Pods can be isolated from a ReplicaSet by simply changing their labels so that they no longer match the ReplicaSet's selector.
- ReplicaSets can be deleted with or without deleting their dependent pods.
- You can easily control the number of replicas (pods) the ReplicaSet should maintain through the command line or by directly editing the ReplicaSet configuration on the fly.
- You can also configure the ReplicaSet to autoscale based on the amount of CPU load the node is experiencing.
- You may have read about ReplicationControllers in older Kubernetes documentation, articles or books. ReplicaSets are the successors of ReplicationControllers. They are recommended to be used instead of

ReplicationControllers as they provide more features.

- A Kubernetes pod serves as a deployment unit for the cluster.
- It may contain one or more containers.
- However, containers (and accordingly, pods) are short-lived entities.
- A container hosting a PHP application, for example may experience an unhandled code exception causing the process to fail, effectively crashing the container. Of course, the perfect solution for such a case is to refactor the code to properly handle exceptions.
- But, till that happens we need to keep the application running and the business going. In other words, we need to restart the pod whenever it fails.
- In parallel, developers are monitoring, investigating and fixing any errors that make it crash.
- At some point, a new version of the pod is deployed, monitored and maintained. It's an ongoing process that is part of the DevOps practice.

Another requirement is to keep a predefined number of pods running. If more pods are up, the additional ones are terminated. Similarly, of one or more pods failed, new pods are activated until the desired count is reached.

A Kubernetes ReplicaSet resource was designed to address both of those requirements. It creates and maintains a specific number of similar pods (replicas).

Under this lab, we'll discuss how we can define a ReplicaSet and what are the different options that can be used for fine-tuning it.

## How Does ReplicaSet Manage Pods?

- In order for a ReplicaSet to work, it needs to know which pods it will manage so that it can restart the failing ones or kill the unneeded.
- It also requires to understand how to create new pods from scratch in case it needs to spawn new ones.

- A ReplicaSet uses labels to match the pods that it will manage. It also needs to check whether the target pod is already managed by another controller (like a Deployment or another ReplicaSet). So, for example if we need our ReplicaSet to manage all pods with the label role=webserver, the controller will search for any pod with that label. It will also examine the ownerReferences field of the pod’s metadata to determine whether or not this pod is already owned by another controller. If it isn’t, the ReplicaSet will start controlling it. Subsequently, the ownerReferences field of the target pods will be updated to reflect the new owner’s data.

To be able to create new pods if necessary, the ReplicaSet definition includes a template part containing the definition for new pods.

## Creating Your First ReplicaSet

```
git clone https://github.com/collabnix/dockerlabs
cd dockerlabs/kubernetes/workshop/replicaset101
```

```
kubectl apply -f nginx_replicaset.yaml
```

```
kubectl get rs
```

```
NAME   DESIRED   CURRENT   READY   AGE
web     4        4        4       2m
```

## A Peep into the ReplicaSet definition file

Let’s examine the definition file that was used to create our ReplicaSet:

- The apiVersion for this object is currently app/v1
- The kind of this object is ReplicaSet
- In the metadata part, we define the name by which we can refer to this ReplicaSet. We also define a number of labels through which we can identify it.
- The spec part is mandatory in the ReplicaSet object. It defines:
- The number of replicas this controller should maintain. It default to 1 if it was not specified.
- The selection criteria by which the ReplicaSet will choose its pods. Be careful not to

use a label that is already in use by another controller. Otherwise, another ReplicaSet may acquire the pod(s) first. Also notice that the labels defined in the pod template (spec.template.metadata.label) cannot be different than those defined in the matchLabels part (spec.selector).
- The pod template is used to create (or recreate) new pods. It has its own metadata, and spec where the containers are specified. You can refer to our article for more information about pods.

## Is Our ReplicaSet the Owner of Those Pods?

OK, so we do have four pods running, and our ReplicaSet reports that it is controlling four pods. In a busier environment, you may want to verify that a particular pod is actually managed by this ReplicaSet and not by another controller. By simply querying the pod, you can get this info:

```
kubectl get pods web-6n9cj -o yaml | grep -A 5 owner
```

The first part of the command will get all the pod information, which may be too verbose. Using grep with the -A flag (it takes a number and prints that number of lines after the match) will get us the required information as in the example:

```
ownerReferences:
  - apiVersion: extensions/v1beta1
        blockOwnerDeletion: true
        controller: true
        kind: ReplicaSet
        name: web
```

## Removing a Pod From a ReplicaSet

You can remove (not delete) a pod that is managed by a ReplicaSet by simply changing its label. Let’s isolate one of the pods created in our previous example:

```
kubectl edit pods web-44cjb
```

Then, once the YAML file is opened, change the pod label to be role=isolated or anything different than role=web. In a few moments, run kubectl get pods. You will notice that we have five pods now. That's because the ReplicaSet dutifully created a new pod to reach the desired number of four pods. The isolated one is still running, but it is no longer managed by the ReplicaSet.

## Scaling the Replicas to 5

```
[node1 replicaset101]$ kubectl scale --replicas=5 -f nginx_replicaset.yaml
```

## Scaling and Autoscaling ReplicaSets

You can easily change the number of pods a particular ReplicaSet manages in one of two ways:

- Edit the controllers configuration by using kubectl edit rs ReplicaSet_name and change the replicas count up or down as you desire.
- Use kubectl directly. For example, kubectl scale –replicas=2 rs/web. Here, I'm scaling down the ReplicaSet used in the article's example to manage two pods instead of four. The ReplicaSet will get rid of two pods to maintain the desired count. If you followed the previous section, you may find that the number of running pods is three instead of two; as we isolated one of the pods so it is no longer managed by our ReplicaSet.

```
kubectl autoscale rs web --max=5
```

This will use the Horizontal Pod Autoscaler (HPA) with the ReplicaSet to increase the number of pods when the CPU load gets higher, but it should not exceed five pods. When the load decreases, it cannot have less than the number of pods specified before (two in our example).

## Best Practices

The recommended practice is to always use the ReplicaSet's template for creating and managing pods. However, because of the way ReplicaSets work, if you create a bare pod (not owned by any controller) with a label that matches the ReplicaSet selector, the controller will automatically adopt it. This has a number of undesirable consequences. Let's have a quick lab to demonstrate them.

Deploy a pod by using a definition file like the following:

```
apiVersion: v1
kind: Pod
metadata:
  name: orphan
  labels:
        role: web
spec:
  containers:
  - name: orphan
        image: httpd
```

It looks a lot like the other pods, but it is using Apache (httpd) instead of Nginx for an image. Using kubectl, we can apply this definition like:

```
kubectl apply -f orphan.yaml
```

Give it a few moments for the image to get pulled and the container is spawned then run kubectl get pods. You should see an output that looks like the following:

```
NAME                READY   STATUS          RESTARTS   AGE
orphan              0/1     Terminating   0          1m
web-6n9cj    1/1          Running         0        25m
web-7kqbm    1/1          Running         0        25m
web-9src7    1/1          Running         0        25m
web-fvxzf    1/1          Running         0        25m
```

The pod is being terminated by the ReplicaSet because, by adopting it, the controller has more pods than it was configured to handle. So, it is killing the excess one.

Another scenario where the ReplicaSet won't terminate the bare pod is that the latter gets created before the ReplicaSet does. To demonstrate this case, let's destroy our ReplicaSet:

```
kubectl delete -f nginx_replicaset.yaml
```

Now, let's create it again (our orphan pod is still running):

```
kubectl apply -f nginx_replicaset.yaml
```

Let's have a look at our pods status by running kubectl get pods. The output should resemble the following:

```
orphan          1/1     Running   0             29s
web-44cjb   1/1         Running   0             12s
web-hcr9j   1/1         Running   0             12s
web-kc4r9   1/1         Running   0             12s
```

The situation now is that we're having three pods running Nginx, and one pod running Apache (the httpd image). As far as the ReplicaSet is concerned, it is handling four pods (the desired number), and their labels match its selector. But what if the Apache pod went down?

Let's do just that:

```
kubectl delete pods orphan
```

Now, let's see how the ReplicaSet responded to this event:

```
kubectl get pods
```

The output should be something like:

```
NAME            READY   STATUS                  RESTARTS   AGE
```

```
web-44cjb   1/1         Running                 0        24s
web-5kjwx   0/1         ContainerCreating   0            3s
web-hcr9j   1/1         Running                 0        24s
web-kc4r9   1/1         Running                 0        24s
```

The ReplicaSet is doing what is was programmed to: creating a new pod to reach the desired state using the template that was added in its definition. Obviously, it is creating a new Nginx container instead of the Apache one that was deleted.

So, although the ReplicaSet is supposed to maintain the state of the pods it manages, it failed to respawn the Apache web server. It replaced it with an Nginx one.

The bottom line: you should never create a pod with a label that matches the selector of a controller unless its template matches the pod definition. The more-encouraged procedure is to always use a controller like a ReplicaSet or, even better, a Deployment to create and maintain your pods.

## Deleting Replicaset

```
kubectl delete rs ReplicaSet_name
```

Alternatively, you can also use the file that was used to create the resource (and possibly, other resource definitions as well) to delete all the resources defined in the file as follows:

```
kubectl delete -f definition_file.yaml
```

The above commands will delete the ReplicaSet and all the pods that it manges. But sometimes you may want to just delete the ReplicaSet resource, keeping the pods unowned (orphaned). Maybe you want to manually delete the pods and you don’t want the ReplicaSet to restart them. This can be done using the following command:

```
kubectl delete rs ReplicaSet_name --cascade=false
```

If you run kubectl get rs now you should see that there are no ReplicaSets there. Yet if you run kubectl get pods, you should see all the pods that were managed by the destroyed

ReplicaSet still running.

The only way to get those pods managed by a ReplicaSet again is to create this ReplicaSet with the same selector and pod template as the previous one. If you need a different pod template, you should consider using a Deployment instead, which will handle replacing pods in a controlled way.

Next »

## Join KubeDaily

**8** Members Online

Support

MEMBERS ONLINE

- h3ll_boy
- MEE6
- Nitinkashyap Apex Legends
- ojaswa
- Parmeshwar
- prasad
- trimankaur
- vikas027

Free voice chat from Discord Connect

Tweets by collabnix