KUBERNETES REPLICASETS: A BRIEF INTRODUCTION

In this blog post we are going to discuss the ReplicaSet concept in Kubernetes. We will explain:

- What RS is
- What to use ReplicaSets for
- How to create it

This post assumes you have a basic understanding of pods, minikube, and kubectl.

(This article is part of our Kubernetes Guide. Use the right-hand menu to navigate.)

What are ReplicaSets?

A ReplicaSet is one of the Kubernetes controllers that makes sure we have a specified number of pod replicas running. (Remember, a controller in Kubernetes is what takes care of tasks to make sure the desired state of the cluster matches the observed state.)

Without RS, we will have to create multiple manifests for the number of pods we need which is a lot of work to deploy replicas of a single application. In previous versions of Kubernetes, the ReplicaSet was called Replication Controller. The main difference between the two is that ReplicaSets allow us to use something called Label Selector.

Labels are key value pair used to specify attributes of objects that are meaningful and useful to users, so keep in mind that it doesn't change the way the core system works. **Label Selectors** is used to identify a set of objects in Kubernetes.

ReplicaSets allow us to use "*set-based*" label selector (e.g ***environment in (production, qa)*** or ***tier***

notin (frontend, backend)) as opposed to "equality-based"(e.g ***environment = production or tier != frontend***) which is what you use with replication controller.

Operators to use with ReplicaSets

There are three kinds of operator we can use with ReplicaSets:

- In
- Notin
- Exists

ReplicaSets manages all the pods with labels that match the selector. When pods are created, it does not differentiate between all pods so if you create another ReplicaSets with the same label selector, the previous ReplicaSets will think it created those pods therefore causing issues.

This is important to note, therefore make sure label selectors do not match from one ReplicaSets to another.

What does the ReplicaSet manifest look like?

```
apiVersion: apps/v1 # our API version
kind: ReplicaSet   # The kind we are creating
Metadata: # Specify all Metadata like name, labels
  name: some-name
  labels:
    app: some-App
    tier: some-Tier
Spec:
  replicas: 3 # Here is where we tell k8s how many replicas we want
  Selector: # This is our label selector field.
    matchLabels:
      tier: some-Tier
    matchExpressions:
      - {key: tier, operator: In, values: } # we are using the set-based
operators
  template:
    metadata:
      labels:
        app: some-App
        tier: someTier
    Spec: # This spec section should look like spec in a pod definition
      Containers:
```

APIVersion, *kind* and *Metadata* look similar to any other object in Kubernetes but the Spec section look slightly different from other objects. As a matter of fact there are two Spec's in the manifest example above. The first Spec lets you declare what the replicaset should look like and the second spec is for containers. "*.spec.template*" is the only required field of the the first spec section. If you do not specify replicas, it will deploy just 1 pod. It is also worth noting that the

"*.spec.template.metadata.labels*" must match exactly what you have in the "*.spec.selector*"(You can see this in our manifest above), or Kubernetes will not allow it to be created.

ReplicaSet: an example

Ok now that we understand what a replicaset is, let us create one. In this example, we will deploy our replica.yaml file which will create a simple frontend nginx app with 3 replicas.

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: myapp-replicas
  labels:
    app: myapp
    tier: frontend
spec:
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
    matchExpressions:
      - {key: tier, operator: In, values: }
  template:
    metadata:
      labels:
        app: myapp
        tier: frontend
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
```

First we run kubectl create command to apply our manifest

```
$ kubectl create -f replica.yaml
replicaset.apps "myapp-replicas" created
```

Next, we make sure it is created

```
$ kubectl get replicaset
NAME                          DESIRED   CURRENT   READY     AGE
myapp-replicas                3         3         3         15s
```

We see that there are 3 deployed and 3 ready. We can also just check the pods.

```
$ kubectl get pod
NAME                          READY     STATUS    RESTARTS   AGE
myapp-replicas-67rkp          1/1       Running   0          33s
```

```
myapp-replicas-6kfd8            1/1       Running   0          33s
myapp-replicas-s96sg            1/1       Running   0          33s
```

We see all 3 running with 0 restarts which mean our application is not crashing. We can also describe the object which will give us more details about our replicas.

```
$ kubectl describe replicaset myapp-replicas
Name:         myapp-replicas
Namespace:    default
Selector:     tier=frontend,tier in (frontend)
Labels:       app=myapp
              tier=frontend
Annotations:
Replicas:     3 current / 3 desired
Pods Status:  3 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:  app=myapp
           tier=frontend
  Containers:
   nginx:
    Image:        nginx
    Port:         80/TCP
    Host Port:    0/TCP
    Environment:
    Mounts:
  Volumes:
Events:
  Type    Reason            Age   From                   Message
  ----    ------            ----  ----                   -------
  Normal  SuccessfulCreate  12m   replicaset-controller  Created pod: myapp-
replicas-6kfd8
  Normal  SuccessfulCreate  12m   replicaset-controller  Created pod: myapp-
replicas-67rkp
  Normal  SuccessfulCreate  12m   replicaset-controller  Created pod: myapp-
replicas-s96sg
```

ReplicaSet FAQs

Here some quick FAQs about ReplicaSets.

What if we no longer need 3 replicas, we now only need one?

All we have to do is change the replica field to the value we want and k8s will scale it to that number. For example "*replicas: 1*". If we make the change and re apply, we will only have one replica running.

Can we remove the pod from a ReplicaSets?

Yes we can. It is as simple as removing the label from the pod and it will be removed from the Set.

How to clean up RS?

To delete replicaset, all we have to do is run "*kubectl delete replicaset myapp-replicas*". This command will delete the replicasets and the pods.

In general Kubernetes recommend that we use deployment (*a higher-level concept that manages ReplicaSets and provides declarative updates to pods along with a lot of other useful features*) controller instead of ReplicaSets.

Additional resources

For more on Kubernetes, explore these resources:

- Kubernetes Guide, with 20+ articles and tutorials
- BMC DevOps Blog
- The State of Kubernetes in 2020
- Bring Kubernetes to the Serverless Party
- How eBay is Reinventing Their IT with Kubernetes & Replatforming Program