ReplicaSet

Initially, ReplicationControllers were the only Kubernetes component for replicating pods and rescheduling them when nodes failed. Later, a similar resource called a ReplicaSet was introduced. It's a new generation of ReplicationController and replaces it completely (ReplicationControllers will eventually be deprecated).

You usually won't create them directly, but instead have them created automatically when you create the higher-level Deployment resource.

ReplicaSet is the next-generation Replication Controller. The only difference between a ReplicaSet and a Replication Controller right now is the selector support. ReplicaSet supports the new set-based selector requirements as described in the labels user guide whereas a Replication Controller only supports equality-based selector requirements.

Comparing a ReplicaSet to a ReplicationController

A ReplicaSet behaves exactly like a ReplicationController, but it has more expressive pod selectors. Whereas a ReplicationController's label selector only allows matching pods that include a certain label, a ReplicaSet's selector also allows matching pods that lack a certain label or pods that include a certain label key, regardless of its value.

Also, for example, a single ReplicationController can't match pods with the label `env=production` and those with the label `env=devel` at the same time. It can only match either pods with the `env=production` label or pods with the `env=devel` label. But a single ReplicaSet can match both sets of pods and treat them as a single group.

Similarly, a ReplicationController can't match pods based merely on the presence of a label key, regardless of its value, whereas a ReplicaSet can. For example, a ReplicaSet can match all pods that include a label with the key `env`, whatever its actual value is (you can think of it as `env=*`).

Defining a ReplicaSet

```
cat << EOF > kubia-replicaset.yaml
apiVersion: apps/v1beta2
kind: ReplicaSet
metadata:
  name: kubia
spec:
```

```
  replicas: 3
  selector:
    matchLabels:
      app: kubia
  template:
    metadata:
      labels:
        app: kubia
    spec:
      containers:
      - name: kubia
        image: luksa/kubia
EOF
```

The first thing to note is that ReplicaSets aren't part of the `v1` API, so you need to ensure you specify the proper `apiVersion` when creating the resource.

You're creating a resource of type ReplicaSet which has much the same contents as the Replication-Controller. The only difference is in the selector. Instead of listing labels the pods need to have directly under the selector property, you're specifying them under `selector.matchLabels`. This is the simpler (and less expressive) way of defining label selectors in a ReplicaSet. Later, you'll look at the more expressive option, as well.

About the API version attribute

This is your first opportunity to see that the apiVersion property specifies two things:

- The API group (which is `apps` in this case)
- The actual API version (`v1beta2`)

Certain Kubernetes resources are in what's called the core API group, which doesn't need to be specified in the `apiVersion` field (you just specify the version—for example, you've been using `apiVersion: v1` when defining Pod resources). Other resources, which were introduced in later Kubernetes versions, are categorized into several API groups.

Creating and examining a ReplicaSet

Create the ReplicaSet from the YAML file with the `kubectl create` command. After that, you can examine the ReplicaSet with `kubectl get` and `kubectl describe`:

```
kubectl create -f kubia-replicaset.yaml
```

```
replicaset.apps "kubia" created
```

Use `rs` shorthand, which stands for `replicaset`.

```
kubectl get rs
```

```
NAME      DESIRED   CURRENT   READY     AGE
kubia     3         3         3         2m
```

```
kubectl describe rs
```

```
Name:         kubia
Namespace:    default
Selector:     app=kubia
Labels:       app=kubia
Annotations:  <none>
Replicas:     3 current / 3 desired
Pods Status:  3 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:  app=kubia
  Containers:
   kubia:
    Image:        luksa/kubia
    Port:         <none>
    Host Port:    <none>
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Events:
  Type    Reason            Age   From                   Message
  ----    ------            ----  ----                   -------
  Normal  SuccessfulCreate  3m    replicaset-controller  Created pod: kubia-8n5f4
  Normal  SuccessfulCreate  3m    replicaset-controller  Created pod: kubia-rf9fq
  Normal  SuccessfulCreate  3m    replicaset-controller  Created pod: kubia-xgccl
```

As you can see, the ReplicaSet isn't any different from a ReplicationController. It's showing it has three replicas matching the selector.

```
kubectl delete rs kubia
```

```
replicaset.extensions "kubia" deleted
```

Deleting the ReplicaSet should delete all the pods.

```
kubectl get pods
```

```
NAME          READY     STATUS        RESTARTS   AGE
kubia-8n5f4   1/1       Terminating   0          8m
kubia-rf9fq   1/1       Terminating   0          8m
kubia-xgccl   1/1       Terminating   0          8m
```

Using the ReplicaSet's more expressive label selectors

The main improvements of ReplicaSets over ReplicationControllers are their more expressive label selectors. You intentionally used the simpler matchLabels selector in the first ReplicaSet example to see that ReplicaSets are no different from Replication-Controllers. Now, you'll rewrite the selector to use the more powerful `matchExpressions` property, as shown in the following listing.

```
cat << EOF > kubia-replicaset-matchexpressions.yaml
apiVersion: apps/v1beta2
kind: ReplicaSet
metadata:
  name: kubia
spec:
  replicas: 3
  selector:
    matchExpressions:
      - key: app
        operator: In
        values:
         - kubia
  template:
    metadata:
      labels:
        app: kubia
    spec:
      containers:
      - name: kubia
        image: luksa/kubia
EOF
```

You can add additional expressions to the selector. As in the example, each expression must contain a `key`, an `operator`, and possibly (depending on the operator) a list of `values`. You'll see four valid operators:

- `In`—Label's value must match one of the specified values.
- `NotIn`—Label's value must not match any of the specified values.
- `Exists`—Pod must include a label with the specified key (the value isn't important). When using this operator, you shouldn't specify the values field.
- `DoesNotExist`—Pod must not include a label with the specified key. The values property must not be specified.

If you specify multiple expressions, all those expressions must evaluate to true for the selector to match a pod. If you specify both `matchLabels` and `matchExpressions`, all the labels must match and all the expressions must evaluate to true for the pod to match the selector.

```
kubectl create -f kubia-replicaset-matchexpressions.yaml
```

```
replicaset.apps "kubia" created
```

```
kubectl get rs
```

```
NAME      DESIRED   CURRENT   READY     AGE
kubia     3         3         3         18s
```

```
kubectl describe rs
```

```
Name:         kubia
Namespace:    default
Selector:     app in (kubia)
Labels:       app=kubia
Annotations:  <none>
Replicas:     3 current / 3 desired
Pods Status:  3 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:  app=kubia
  Containers:
   kubia:
    Image:        luksa/kubia
    Port:         <none>
    Host Port:    <none>
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Events:
  Type    Reason            Age   From                   Message
  ----    ------            ----  ----                   -------
  Normal  SuccessfulCreate  49s   replicaset-controller  Created pod: kubia-z56zt
  Normal  SuccessfulCreate  49s   replicaset-controller  Created pod: kubia-k9nxp
  Normal  SuccessfulCreate  49s   replicaset-controller  Created pod: kubia-5jnc2
```

```
kubectl delete rs kubia
```

```
replicaset.extensions "kubia" deleted
```