Lab 11B. Namespaces, ReplicaSets

In this lab exercise, you are going to learn:

- How to create a Kubernetes namespace and switch to it.
- How to switch Kubernetes contexts.
- How to create a ReplicaSet.
- How to achieve high availability and scalability with ReplicaSets.

Namespaces

Check current config:

```
kubectl config view
```

You can also examine the current configs in the **`cat ~/.kube/config`** file.

Creating a Namespace and Switching to It

Namespaces separate resources running on the same physical infrastructure into virtual clusters. It is typically useful in mid to large scale environments with multiple projects and teams, and need separate scopes. It can also be useful to map to your workflow stages, e.g. dev, stage, prod.

Before you create a namespace, delete all the pods in the default namespace that you may have created earlier and are no longer needed.

Lets create a namespace called **`instavote`**:

```
kubectl get ns
kubectl create namespace instavote
kubectl get ns
```

And switch to it:

```
kubectl config --help
kubectl config get-contexts
kubectl config current-context
kubectl config set-context --help
kubectl config set-context --current --namespace=instavote
kubectl config get-contexts
kubectl config view
```

Go back to the monitoring screen and observe what happens after switching the namespace. For example:

```
kubectl config set-context --current --namespace=default
kubectl config set-context --current --namespace=kube-system
kubectl config set-context --current --namespace=instavote
```

ReplicaSets

To understand how ReplicaSet works with the selectors, let's launch a pod in the new namespace with existing spec.

```
cd k8s-code/pods
kubectl apply -f vote-pod.yaml

kubectl get pods
```

Adding ReplicaSet Configurations

Let's now write spec for the ReplicaSet. This is going to mainly contain:

- Replicas: define the scalability configs here
- Selectors: define configs which provide a base for checking availability
- Template (pod spec)
- `minReadySeconds`: duration to wait after pod is ready till its declared as available (used by Deployment)

From here on, we would switch to the project and environment specific path and work from there.

```
cd projects/instavote/dev
```

Edit file: `vote-rs.yaml`

```
apiVersion: xxx
kind: xxx
```

```
metadata:
  xxx
spec:
  xxx
  template:
    metadata:
      name: vote
      labels:
        app: python
        role: vote
        version: v1
    spec:
      containers:
        - name: app
          image: schoolofdevops/vote:v1
          resources:
            requests:
              memory: "64Mi"
              cpu: "50m"
            limits:
              memory: "128Mi"
              cpu: "250m"
```

The above file already contains spec that you had written for the pod. You can see it has already been added as part of `spec.template` for ReplicaSet.

Let's now add the details specific to ReplicaSet.

File: `vote-rs.yaml`

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: vote
spec:
  replicas: 4
  minReadySeconds: 20
  selector:
    matchLabels:
      role: vote
    matchExpressions:
      - key: version
        operator: Exists
  template:
    metadata:
      name: vote
      labels:
        app: python
        role: vote
        version: v1
    spec:
      containers:
        - name: app
          image: schoolofdevops/vote:v1
          resources:
            requests:
              memory: "64Mi"
              cpu: "50m"
            limits:
              memory: "128Mi"
              cpu: "250m"
```

The complete file will look similar to the one above. Let's now go ahead and apply it.

```
kubectl apply -f vote-rs.yaml --dry-run=client
kubectl apply -f vote-rs.yaml
kubectl get rs
kubectl describe rs vote
kubectl get pods
kubectl get pods --show-labels
```

High Availability

Try to delete pods created by ReplicaSet. Replace **`pod-xxxx`** and **`pod-yyyy`** with actual names of the pods in your environment:

```
kubectl get pods
kubectl delete pods vote-xxxx vote-yyyy
```

Observe as pods are automatically created again.

Let's now delete the pod created independent of ReplicaSet:

```
kubectl get pods
kubectl delete pods vote
```

Does ReplicaSet take any action after deleting the pod created outside of its spec? If so, why?

Scalability

Scaling out and scaling in your application is as simple as running:

```
kubectl scale rs vote --replicas=8
kubectl get pods --show-labels
```

```
kubectl scale rs vote --replicas=25
kubectl get pods --show-labels
```

```
kubectl scale rs vote --replicas=7
kubectl get pods --show-labels
```

Check if the number of replicas:

- Increased to 8
- Increased further to 25
- Dropped to 7

Deploying New Version of the Application

```
kubectl edit rs vote
```

- Update the version of the image from **`schoolofdevops/vote:v1`** to **`schoolofdevops/vote:v2`**
- Update **`template.metadata.labels`** from **`version=v1`** to **`version=v2`**

Save the file. If you are using **`kubectl edit`**, it will apply changes immediately as you save them. There is no need to use the **`kubectl apply`** command.

Did the application get updated? Did updating ReplicaSet launch new pods to deploy new versions?

Summary

Your application is highly available as well as scalable with ReplicaSets. However, ReplicaSet by itself does not have the intelligence to trigger a rollout if you update the version. For that, you are going to need a *deployment* which is something you will learn about in the next chapter.