Using Default Storage Classes

In this lesson, we will look into default storage classes and will use them.

WE'LL COVER THE FOLLOWING

Default Storage Classes

Working with dynamic provisioning simplifies a few things. Still, a user needs to know which volume type to use. While in many cases that is an important choice, there are often situations when a user might not want to worry about that. It might be easier to use the cluster administrator's choice for volume types and let all claims that do not specify `storageClassName` get a default volume. We'll try to accomplish that through one of the admission controllers.

DefaultStorageClass Admission Controller

Admission controllers intercept requests to the Kubernetes API server. We won't go into the details of admission controllers since the list of those supported by Kubernetes is relatively big.

We are interested only in the DefaultStorageClass which happens to be already enabled in the cluster we created with kops.

DefaultStorageClass admission controller observes creation of

PersistentVolumeClaims. Through it, those that do not request any specific

storage class are automatically assigned a default storage class. As a result, PersistentVolumeClaims that do not request any special storage class are bound to PersistentVolumes created from the default StorageClass. From a user's perspective, there's no need to care about volume types since they will be provisioned based on the default type unless they choose a specific class.

Available Storage Classes

Let's take a look at the storage classes currently available in our cluster.

```
kubectl get sc
```

The **output** is as follows.

```
NAME            PROVISIONER            AGE
default         kubernetes.io/aws-ebs  56m
gp2 (default)   kubernetes.io/aws-ebs  56m
```

This is not the first time we're listing the storage classes in our cluster. However, we did not discuss that one of the two (`gp2`) is marked as the default StorageClass.

Let's describe the `gp2` class.

```
kubectl describe sc gp2
```

The **output**, limited to the relevant parts, is as follows.

```
Name:            gp2
IsDefaultClass:  Yes
Annotations:     kubectl.kubernetes.io/last-applied-configuration={"apiVersion":"storage.k8s.
,storageclass.beta.kubernetes.io/is-default-class=true
Provisioner:     kubernetes.io/aws-ebs
Parameters:      type=gp2
ReclaimPolicy:   Delete
Events:          <none>
```

The important part lies in the annotations. One of them is `".../is-default-class":"true"`. It sets that StorageClass as default. As a result, it will be used to

create PersistentVolumes by any PersistentVolumeClaim that does not specify a StorageClass name.

Dynamic Volume Provision

Let's try to adapt Jenkins stack to use the ability to dynamically provision a volume associated with the DefaultStorageClass.

Looking into the Definitions

The new Jenkins definition is as follows.

```
cat pv/jenkins-default.yml
```

The **output**, limited to the `PersistentVolumeClaim`, is as follows.

```
...
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: jenkins
  namespace: jenkins
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
...
```

It's hard to spot the difference between that YAML file and the one we used before. It is very small and hard to notice change so we'll execute `diff` to compare the two.

```
diff pv/jenkins-dynamic.yml \
    pv/jenkins-default.yml
```

The **output** is as follows.

```
48d47
<   storageClassName: gp2
```

As you can see, the only difference is that `pv/jenkins-dynamic.yml` doesn't have `storageClassName: gp2`. That field is omitted from the new definition. Our new `PersistentVolumeClaim` does not have an associated StorageClass.

Applying the Definition

Let's `apply` the new definition.

```
kubectl apply \
    -f pv/jenkins-default.yml \
    --record
```

The **output** is as follows.

```
namespace "jenkins" configured
ingress "jenkins" configured
service "jenkins" configured
persistentvolumeclaim "jenkins" created
deployment "jenkins" created
```

Verifying PersistentVolumes

What we're interested in are PersistentVolumes, so let's retrieve them.

```
kubectl get pv
```

```
NAME    CAPACITY ACCESS MODES RECLAIM POLICY STATUS CLAIM           STORAGECLASS REASON AGE
pvc-... 1Gi      RWO          Delete         Bound  jenkins/jenkins gp2                 16s
```

As you can see, even though we did not specify any StorageClass, a volume was created based on the `gp2` class, which happens to be the default one.

Deleting the Resources

We'll delete the `jenkins` Deployment and PersistentVolumeClaim before we explore how we can create our own Storage Classes.

```
kubectl --namespace jenkins \
    delete deploy,pvc jenkins
```

The **output** is as follows.

```
deployment "jenkins" deleted
persistentvolumeclaim "jenkins" deleted
```

In the next lesson, we will explore how to create our own Storage Classes.