Creating Kubernetes Persistent Volumes

In this lesson, we will understand and create three Kubernetes persistent Volumes.

WE'LL COVER THE FOLLOWING

- Understanding Persistent Volumes
  - Looking into the Definition
    - Exploring the Spec Section (Lines 7-15)
  - Exploring Other Storage Platforms
- Creation of the Persistent Volume
  - Verification

Understanding Persistent Volumes

The fact that we have a few EBS volumes available does not mean that Kubernetes knows about their existence. We need to add **PersistentVolumes** that will act as a bridge between our Kubernetes cluster and AWS EBS volumes.

PersistentVolumes allow us to abstract details of how storage is provided (e.g., EBS) from how it is consumed. Just like Volumes, PersistentVolumes are resources in a Kubernetes cluster. The main difference is that their lifecycle is independent of individual Pods that are using them.

Looking into the Definition

Let's take a look at a definition that will create a few PersistentVolumes.

```
cat pv/pv.yml
```

The **output**, limited to the first of the three volumes, is as follows.

```
kind: PersistentVolume
apiVersion: v1
metadata:
  name: manual-ebs-01
  labels:
    type: ebs
spec:
  storageClassName: manual-ebs
  capacity:
    storage: 5Gi
  accessModes:
    - ReadWriteOnce
  awsElasticBlockStore:
    volumeID: REPLACE_ME_1
    fsType: ext4
...
```

Exploring the Spec Section (Lines 7-15)

The `spec` section features a few interesting details.

Line 8: We set `manual-ebs` as the storage class name. We'll see later what its function is. For now, just remember the name.

Line 9-10: We defined that the storage capacity is `5Gi`. It does not need to be the same as the capacity of the EBS we created earlier, as long as it is not bigger. Kubernetes will try to match `PersistentVolume` with, in this case, EBS that has a similar, if not the same capacity. Since we have only one EBS volume with 10GB, it is the closest (and the only) match to the `PersistentVolume` request of `5Gi`. Ideally, persistent volumes capacity should match EBS size, but we wanted to demonstrate that any value equal to or less then the actual size should do.

Line 11-12: We specified that the access mode should be `ReadWriteOnce`. That means that we'll be able to mount the volume as read-write only once. Only one Pod will be able to use it at any given moment. Such a strategy fits us well since EBS cannot be mounted to multiple instances. Our choice of the access mode is not truly a choice, but more an acknowledgment of the way how EBS works. The alternative modes are `ReadOnlyMany` and `ReadWriteMany`. Both modes would result in volumes that could be mounted to multiple Pods, either as read-only or read-write. Those modes would be more suitable for NFS like, for example, EFS, which can be mounted by multiple instances.

Line 13-15: The `spec` fields we explored so far are common to all persistent volume types. Besides those, there are entries specific to the actual volume we are associating with a Kubernetes `PersistentVolume`. Since we're going to use

are associating with a Kubernetes `PersistentVolume`. Since we're going to use EBS, we specified `awsElasticBlockStore` with the volume ID and file system

type. Since we could not know in advance what will be the ID of your EBS volume, the definition has the value set to `REPLACE_ME`. Later on, we'll replace it with the ID of the EBS we created earlier.

Exploring Other Storage Platforms

There are many other types we could have specified instead.

- If this cluster would run on Azure, we could use `azureDisk` or `azureFile`.
- In Google Compute Engine (GCE) it would be `GCEPersistentDisk`.
- We could have setup `Glusterfs`.
- If we would have this cluster running in an on-prem data center, it would probably be `nfs`.

There are quite a few others we could use but, since we're running the cluster in AWS, many would not work, while others could be too difficult to set up.

Since EBS is already available, we'll just roll with it. All in all, this cluster is in AWS, and `awsElasticBlockStore` is the easiest, if not the best choice.

Creation of the Persistent Volume

Now that we have an understanding of the YAML definition, we can proceed and create the `PersistentVolume`.

```
cat pv/pv.yml \
    | sed -e \
    "s@REPLACE_ME_1@$VOLUME_ID_1@g" \
    | sed -e \
    "s@REPLACE_ME_2@$VOLUME_ID_2@g" \
    | sed -e \
    "s@REPLACE_ME_3@$VOLUME_ID_3@g" \
    | kubectl create -f - \
    --save-config --record
```

We used `cat` to output the contents of the `pv/pv.yml` file and pipe it into `sed` commands which, in turn, replaced the `REPLACE_ME_*` strings with the IDs of the EBS volumes we created earlier. The result was sent to the `kubectl create` command that created persistent volumes. As a result, we can see from the

output that all three PersistentVolumes were created.

Verification

Let's take a look at the persistent volumes currently available in our cluster.

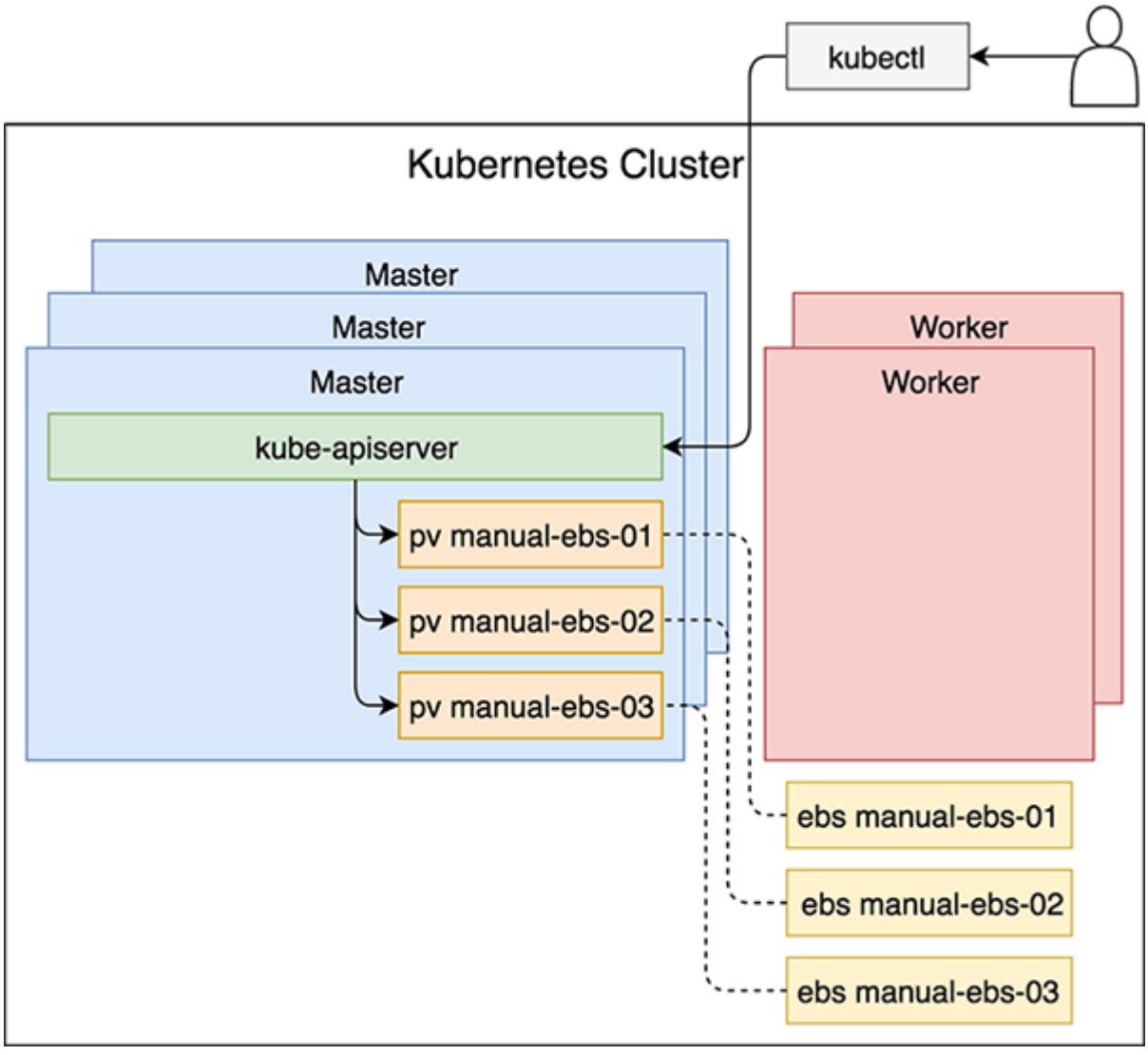
```
kubectl get pv
```

The **output** is as follows.

```
NAME          CAPACITY ACCESS MODES RECLAIM POLICY STATUS    CLAIM STORAGECLASS REASON AGE
manual-ebs-01 5Gi      RWO          Retain         Available       manual-ebs           11s
manual-ebs-02 5Gi      RWO          Retain         Available       manual-ebs           11s
manual-ebs-03 5Gi      RWO          Retain         Available       manual-ebs           11s
```

It should come as no surprise that we have three volumes.

The status column is the most interesting part we are seeing.



Kubernetes persistent volumes tied to EBS volumes

The persistent volumes are `available`. We created them, but no one is using them. They just sit there waiting for someone to claim them.

In the next lesson, we will claim the persistent volumes.