

William Broach Fo

Follow

DevOps Janitor | Recovering SysAdmin | Kubernetes | Docker | Distributed Computing | (@while1eq1)

Jun 15, 2017 · 2 min read

Dynamically Provisioned PV's Using Amazon EFS in a MultiAZ Kubernetes setup



Warning: Dynamically Provisioned PersistentVolumes have a default persistentVolumeReclaimPolicy of DELETE, which means if the PersistentVolumeClaim ever gets unbound from the PV, it will delete all the data. If you want to change this you need to change the persistentVolumeReclaimPolicy of the created PV's to "Retain".

In this guide we're going to setup a PersistentVolume (PV) using Amazon's EFS that will give pods the ability to get rescheduled across Availability Zones (AZ's) and still maintain access to the data in that PV.

Prerequisites: An existing EFS volume and volume mounts in the same VPC as your Kubernetes cluster.

Step 1: Make a directory in your EFS called $\protect\operatorname{\textsc{PVS}}$ (how you do this is up to you)

Step 2: Create a ConfigMap that contains your EFS filesystem ID / AWS Region

1 of 6 11/8/2018, 3:28 PM

```
kubectl create configmap efs-provisioner \
--from-literal=file.system.id=fs-xxxxxx \
--from-literal=aws.region=us-east-1 \
--from-literal=novisioner_name=example_com/aws-efs
```

Step 3: Edit the efs-provisioner deployment to add your EFS filesystem ID and mount point from Step 1. (lines 39 / 40 below)

```
1
      kind: Deployment
 2
      apiVersion: extensions/v1beta1
 3
      metadata:
 4
        name: efs-provisioner
 5
      spec:
6
        replicas: 1
 7
        strategy:
8
          type: Recreate
 9
        template:
10
          metadata:
            labels:
11
12
              app: efs-provisioner
13
          spec:
14
            containers:
15
               - name: efs-provisioner
16
                 image: quay.io/external_storage/efs-provisioner:
17
                 env:
                   - name: FILE_SYSTEM_ID
                     valueFrom:
                       configMapKeyRef:
20
21
                         name: efs-provisioner
22
                         key: file.system.id
23
                   - name: AWS_REGION
24
                     valueFrom:
25
                       configMapKeyRef:
26
                         name: efs-provisioner
```

Step 4: Launch the deployment of the efs-provisioner: kubectl apply

-f efs-provisioner-deployment.yml

Step 5: Create a StorageClass called 'aws-efs' that uses the above

2 of 6 11/8/2018, 3:28 PM

provisioner

```
kind: StorageClass
apiVersion: storage.k8s.io/v1beta1
metadata:
name: aws-efs
```

```
Step 6: kubectl create -f aws-efs-sc.yml
```

Step 7: Create a PersistentVolumeClaim that uses the 'aws-efs' StorageClass you just created in the previous step.

Note: You can make the storage size anything you'd like up to Amazon's maximum of 8 Exabytes.

Note: For every PersistentVolumeClaim you make, it will make a subdirectory in EFS under '/pvs/pvc-<claim_id>'

```
kind: PersistentVolumeClaim
1
2
     apiVersion: v1
3
     metadata:
4
       name: efs
5
       annotations:
         volume.beta.kubernetes.io/storage-class: "aws-efs"
6
7
     spec:
8
       accessModes:
           Doodling + oMany
```

Step 8: kubectl create -f aws-efs-pvc.yml

Your Done!

You can now add the following to any podspec / deployment / etc...

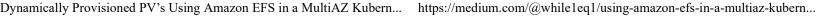
```
volumes:
    - name: efs
persistentVolumeClaim:
```

3 of 6 11/8/2018, 3:28 PM

Full example in a deployment:

```
1
      apiVersion: extensions/v1beta1
2
      kind: Deployment
3
      metadata:
4
       name: efs-test
5
      spec:
6
       replicas: 1
7
        template:
          metadata:
8
9
            labels:
10
              app: efs-test
11
          spec:
12
            containers:
            - name: efs-test
13
14
              image: alpine:latest
```

4 of 6



5 of 6



6 of 6