kubernetes-incubator / **external-storage**

Branch: master ▾ external-storage / aws / efs /

Create new file | Upload files | Find file | History

wongma7 Use kubernetes-sigs/sig-storage-lib-external-provisioner — Latest commit `0ddf1e5` 22 days ago

..		
cmd/efs-provisioner	Use kubernetes-sigs/sig-storage-lib-external-provisioner	20 days ago
deploy	Distribute Role+Rolebinding everywhere instead of giving cluster-scop...	3 months ago
.gitignore	Move efs to aws/efs	2 years ago
CHANGELOG.md	Update changelog for efs v0.1.2	a year ago
Dockerfile	Release efs-provisioner v0.1.0	a year ago
Makefile	Use kubernetes-sigs/sig-storage-lib-external-provisioner	20 days ago
OWNERS	Rename OWNERS assignees: to approvers:	a year ago
README.md	Merge pull request #984 from wongma7/kc	2 months ago

README.md

efs-provisioner

container ready

The efs-provisioner allows you to mount EFS storage as PersistentVolumes in kubernetes. It consists of a container that has access to an AWS EFS resource. The container reads a configmap which contains the EFS filesystem ID, the AWS region and the name you want to use for your efs-provisioner. This name will be used later when you create a storage class.

Prerequisites

- An EFS file system in your cluster's region
- Mount targets and security groups such that any node (in any zone in the cluster's region) can mount the EFS file system by its File system DNS name

Getting Started

If you are new to Kubernetes or to PersistentVolumes this quick start will get you up and running with simple defaults.

- Download the manifest file manifest.yaml.

```
wget https://raw.githubusercontent.com/kubernetes-incubator/external-storage/master/aws/efs/deploy
/manifest.yaml
```

- In the configmap section change the `file.system.id:` and `aws.region:` to match the details of the EFS you created.
- In the deployment section change the `server:` to the DNS endpoint of the EFS you created.
- `kubectl apply -f manifest.yaml`
- Now you can use your PersistentVolume in your pod or deployment by referencing your claim name when it's created.

```
kind: Pod
apiVersion: v1
metadata:
  name: test-pod
spec:
  containers:
  - name: test-pod
    image: gcr.io/google_containers/busybox:1.24
    command:
      - "/bin/sh"
    args:
      - "-c"
      - "touch /mnt/SUCCESS && exit 0 || exit 1"
    volumeMounts:
      - name: efs-pvc
        mountPath: "/mnt"
  restartPolicy: "Never"
  volumes:
    - name: efs-pvc
      persistentVolumeClaim:
        claimName: efs
```

If you scale this pod each aditional pod will also be able to read and write the same files. You may also reference the same claimName in another type of pod so your 2 applications can read and write the same files. If you wish to have a second application that uses EFS storage but don't want other pods to access the files, create a new claim using a new name but the same storage class.

Some times you want the replica pods to be on EFS but you do not wish them to share the same files. In those situations it's best to use a StatefulSet. When a StatefulSet scales up it will dynamically create new claims for your pods.

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  ports:
  - port: 80
    name: web
  clusterIP: None
  selector:
    app: nginx
---
apiVersion: apps/v1beta1
kind: StatefulSet
metadata:
  name: web
spec:
  serviceName: "nginx"
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
    spec:
      terminationGracePeriodSeconds: 10
      containers:
      - name: nginx
        image: gcr.io/google_containers/nginx-slim:0.8
        ports:
        - containerPort: 80
          name: web
        volumeMounts:
        - name: efs
          mountPath: /usr/share/nginx/html
  volumeClaimTemplates:
  - metadata:
      name: efs
      annotations:
        volume.beta.kubernetes.io/storage-class: aws-efs
    spec:
      accessModes: [ "ReadWriteOnce" ]
      resources:
        requests:
          storage: 1Mi
```

Note: We do not reference a claim name, instead we give the new claim the name efs and we list the StorageClass we wish to use when creating the claim. I like to use the type of storage, because the name of the pod will be be added to it. This example when ran will create the claim `efs-web-0` and scaling the pod up to 3 will create `efs-web-1` and `efs-web-2` .

If you wish to learn more about efs-provisioner and how to change additional settings, you can continue on in the deployment section. You might also want to check the FAQ at the bottom.

Deployment

Create a configmap containing the File system ID and Amazon EC2 region of the EFS file system you wish to provision NFS PVs from, plus the name of the provisioner, which administrators will specify in the `provisioner` field of their `StorageClass(es)` , e.g. `provisioner: example.com/aws-efs` .

```
$ kubectl create configmap efs-provisioner \
--from-literal=file.system.id=fs-47a2c22e \
--from-literal=aws.region=us-west-2 \
--from-literal=provisioner.name=example.com/aws-efs
```

> See Optional: AWS credentials secret if you want the provisioner to only once at startup check that the EFS file system you specified in the configmap actually exists.

Decide on & set aside a directory within the EFS file system for the provisioner to use. The provisioner will create child directories to back each PV it provisions. Then edit the `volumes` section at the bottom of "deploy/deployment.yaml" so that the `path` refers to the directory you set aside and the `server` is the same EFS file system you specified.

```
    volumes:
      - name: pv-volume
        nfs:
          server: fs-47a2c22e.efs.us-west-2.amazonaws.com
          path: /persistentvolumes
```

You will need to create the directory you use for `path:` on your EFS file system first or the efs-provisioner pod will fail to start.

```
$ kubectl create -f deploy/deployment.yaml
deployment "efs-provisioner" created
```

If you are not using RBAC or OpenShift you can continue to the usage section.

Authorization

If your cluster has RBAC enabled or you are running OpenShift you must authorize the provisioner. If you are in a namespace/project other than "default" edit `deploy/rbac.yaml` .

RBAC

```
# Set the subject of the RBAC objects to the current namespace where the provisioner is being deployed
$ NAMESPACE=`kubectl config get-contexts | grep '^*' | tr -s ' ' | cut -d' ' -f5`
$ sed -i'' "s/namespace:.*/namespace: $NAMESPACE/g" ./deploy/rbac.yaml
$ kubectl create -f deploy/rbac.yaml
```

SELinux

If SELinux is enforcing on the node where the provisioner runs, you must enable writing from a pod to a remote NFS server (EFS in this case) on the node by running:

```
$ setsebool -P virt_use_nfs 1
$ setsebool -P virt_sandbox_use_nfs 1
```

https://docs.okd.io/latest/install_config/persistent_storage/persistent_storage_nfs.html#nfs-selinux

Usage

First a `StorageClass` for claims to ask for needs to be created.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: slow
provisioner: example.com/aws-efs
parameters:
  gidMin: "40000"
  gidMax: "50000"
  gidAllocate: "true"
```

Parameters

- `gidMin` + `gidMax` : A unique value (GID) in this range (`gidMin` - `gidMax`) will be allocated for each dynamically provisioned volume. Each volume will be secured to its allocated GID. Any pod that consumes the claim will be able to read/write the volume because the pod will automatically receive the volume's allocated GID as a supplemental group, but non-pod mounters outside the system will not have read/write access unless they have the GID or root privileges. See here and here for more information. Default to `"2000"` and `"2147483647"` .
- `gidAllocate` : Whether to allocate GIDs to volumes according to the above scheme at all. If `"false"` , dynamically provisioned volumes will not be allocated GIDs, `gidMin` and `gidMax` will be ignored, and anyone will be able to read/write volumes. Defaults to `"true"` .

Once you have finished configuring the class to have the name you chose when deploying the provisioner and the parameters you want, create it.

```
$ kubectl create -f deploy/class.yaml
storageclass "aws-efs" created
```

When you create a claim that asks for the class, a volume will be automatically created.

```
$ kubectl create -f deploy/claim.yaml
persistentvolumeclaim "efs" created
$ kubectl get pv
NAME                                       CAPACITY   ACCESSMODES   RECLAIMPOLICY   STATUS    CLAIM         REASON
pvc-557b4436-ed73-11e6-84b3-06a700dda5f5   1Mi        RWX           Delete          Bound     default/efs
```

Optional: AWS credentials secret

Create a secret containing the AWS credentials of a user assigned the AmazonElasticFileSystemReadOnlyAccess policy. The credentials will be used by the provisioner only once at startup to check that the EFS file system you specified in the configmap actually exists.

```
$ kubectl create secret generic aws-credentials \
  --from-literal=aws-access-key-id=AKIAIOSFODNN7EXAMPLE \
  --from-literal=aws-secret-access-key=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
```

Add a reference to the secret in the deployment yaml.

```
...
        - name: AWS_ACCESS_KEY_ID
          valueFrom:
            secretKeyRef:
              name: aws-credentials
              key: aws-access-key-id
        - name: AWS_SECRET_ACCESS_KEY
          valueFrom:
            secretKeyRef:
              name: aws-credentials
              key: aws-secret-access-key
...
```

FAQ

- Do I have to use a configmap?

Nope, the configmap values are just fed to the container as an environment variables. It is handy though when you create your EFS with Terraform and use Terraform to create the configmap dynamically.

- I noticed the EFS gets mounted directly to the efs-provisioner container, can I do that for my apps?

Yes you can but it's not recommended. You lose the reusability of the StorageClass to be able to dynamically provision new PersistentVolumes for new containers and pods.

- But what if the efs-provisoner pod goes down?

Your containers will continue to have EFS storage as they are mapped directly to EFS but behind the scenes they were mounted to a folder created by the EFS provisioner. New claims will not be provisioned nor deleted while the efs-provisioner pod is not running. When it comes backup it will catchup on any work it has missed.

- Can I scale the efs-provisioner accross my nodes?

You can but it's not needed. You won't see a performance increase and you wont have a storage outage if the underlying node dies.

- Can I have multiple efs-provisioners pointed at multiple EFS mounts?

Yes you can, but would you really need to? EFS is designed to scale across many nodes and the efs-provisioner already has the ability to divide EFS into seperate chunks for your applications.

- I don't like the manual step of mounting and creating the /persistentvolumes directory.

It's not needed but it is helpful if you are going to use your EFS for other things than Kubernetes. In your efs-provisioner deployment set your mounts like this...

```
          volumeMounts:
            - name: pv-volume
              mountPath: /persistentvolumes
      volumes:
        - name: pv-volume
          nfs:
            server: {{ efs_file_system_id }}.efs.{{ aws_region }}.amazonaws.com
            path: /
```

- I noticed when creating the claim it has request for a really small amount of storage?

The storage section size is a requirment because most other PersistentVolumes need it. Every pod accessing EFS will have unlimited storage. I use 1Mi to remind my self it's unlimited.

- Can I omit that part of the claim?

No, you must list a size even though it's not used with EFS.

- Can I create multiple StorageClasses for the same provisioner?

Yes, you can create multiple StorageClasses for the same provisioner, each with their own `parameters` settings. Note that if two StorageClasses enable `gidAllocate` and the `gidMin` / `gidMax` ranges overlap, the same gid could be allocated twice.