Backup Solution of K8S cluster(velero)

****

**** Backup Solution on Kubernetes ****

****

There are many backup plans on kubernetes. One of the best practices of kubernetes backups is using velero. Velero gives you tools for backup and restore your Kubernetes cluster resources and persistent volumes. We can run velero with a cloud provider or on-premise.

With velero we can:

- Take backup of our cluster and restore in case of loss.
- Migrate cluster resources to other cluster.
- Replicate our production cluster to development and staging cluster.

Velero consists of:

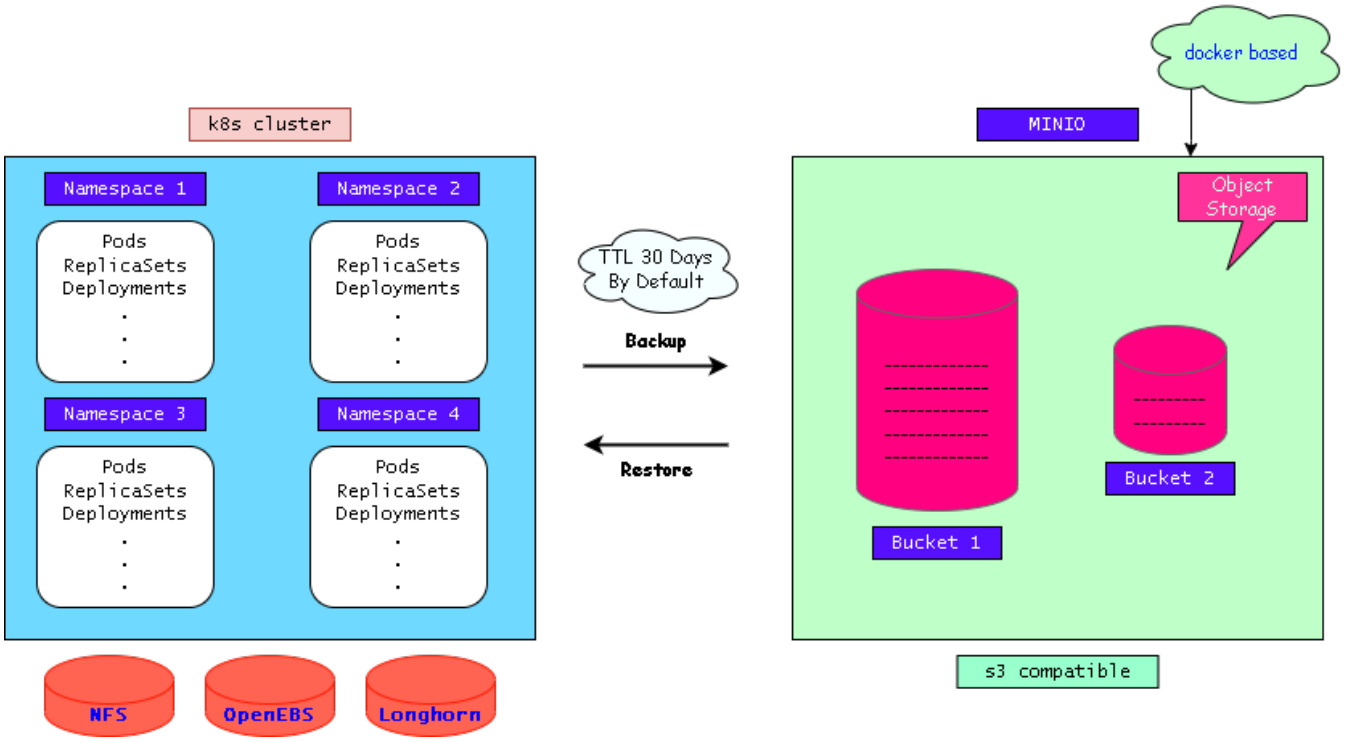
- A server that runs on cluster
- A command-line client that runs locally.

For taking backup of all resources within the cluster, velero needs an S3-like object storage such as MinIO.

Velero allows the user to take snapshot of persistent volumes as part of the backup if we are using cloud providers block storage such as longhorn, openebs, aws ebs,... . but if we are using a volume type that does not have a native snapshot concept, velero integrates with restic to enable it. The downside is that restic does not support *hostPath* volume type. So to make this possible, we chose to use the local-volume type.

In order to easy our management of local-volume types we install the local-volume-provider. It detects the volumes under the chosen path and creates the persistent volume to be used by the application.

In figure below we can see an overall view of our scenario:



As we can see, velero needs an object storage like MinIO to save our resources as json files. So first step is to install and configure MinIO.

MinIO Server

Setting up minio

To setup MinIO we use docker-compose file. The important subject is that **we have to deploy our MinIO server on a server outside the cluster** and preferred to run on another locations. And also **recommend to use ssd disks with high rate of read/write** of disk.

After setting up our physical servers and hard disks, we are ready to deploy MinIO server.

First of all add physical disks to server then create a directory for example “/mnt/minio” and then mount that physical disk to /mnt/minio. After that create a docker-compose file for deploying MinIO:

```
version: !!str 3
services:
  minio1:
    image: quay.io/minio/minio
    command: server /data --console-address ":9001"
    privileged: yes
    ports:
      - "9000:9000"
      - "9001:9001"
    environment:
      - "MINIO_ROOT_USER=<username>"
      - "MINIO_ROOT_PASSWORD=<password>"
    volumes:
      - "/mnt/minio/:/data"
```

After that we can up this server with docker-compose command and we have a MinIO server up and running, to access this server we can login with ip address of the server with port 9001 on browser.

Create Bucket

Next step is create a bucket on minio server for velero backups. So login to minio using ip of the server and port 9001. And create a bucket with name of “kubedemo”

Installing velero client command-line

for working with velero server first we have to download velero command-line tool. this tool must install on k8s master node because it has to install velero server on cluster. For this, first we create directory “/opt/velero” on master node and then following steps below:

```
wget https://github.com/vmware-tanzu/velero/releases/download/v1.7.1/velero-v1.7.1-linux-amd64.tar.gz
tar xvzf velero-v1.7.1-linux-amd64.tar.gz
cd velero-v1.7.1-linux-amd64.tar.gz && mv velero /usr/bin/local
```

Create credentials file to access to MinIO from velero

Now we have to create an access file for velero. First create directory “/opt/velero” on master node and touch a file with name “minio.credentials” with below content:

```
[default]
aws_access_key_id = <minio_username>
aws_secret_access_key = <minio_password>
```

we defined these values on docker-compose as credential of minio server.

Install velero server in k8s cluster using velero command-line

In this step we have to install velero on k8s cluster. for installing velero we have to use some important flags witch is --provider to use a provider, --use-restic to using velero compatible volume snapshoter, --plugins for using compatible plugin for backup volumes and resources,…

to install velero we use command below:

```
velero install --provider aws --use-restic --plugins velero/velero-plugin-for-aws:v1.2.1 --bucket kubedemo --
secret-file /opt/velero/minio.credentials --backup-location-config region=minio,s3ForcePathStyle=true,
s3Url=http://<minio_server_ip_addr>:9000
```

this command will install all the components and resources of velero server under namespace of “velero” on k8s cluster.

Install local-volume-provider

Now we have to install local-volume-provider, that tell to velero that detects volumeMounts on our resources and back them up.

```
git clone https://github.com/kubernetes-sigs/sig-storage-local-static-provisioner.git
cd sig-storage-local-static-provisioner
kubectl create -f deployment/kubernetes/example/default_example_storageclass.yaml
helm template ./helm/provisioner > deployment/kubernetes/provisioner_generated.yaml
```

replace RELEASE_NAME with your favorite name with lower case in provisioner_generated.yaml file.

```
kubectl create -f deployment/kubernetes/provisioner_generated.yaml
```

after deploying local-volume-provider we need to deploy our application and if it has pvc we should be aware of that.

Annotate each pod that we want to backup

Before backup, we should annotate each pod that we want to backup. Each pod with a persistent volume needs to be annotated in order for restic to detect them and back them up. For that run the command below:

```
kubectl -n “namespace” annotate pod/”pod name” backup.velero.io/backup-volumes=YOUR_VOLUME_NAME_1,
YOUR_VOLUME_NAME_2,…
```

YOUR_VOLUME_NAME_1 is the name of the volume found in our manifest of application witch is like following pattern:

```
volumeMounts:
  - name: mysql-persistent-storage
    mountPath: /var/lib/mysql
```

check and verify velero is up and ready

```
velero version
kubectl get ns (velero namespace has been created)
kubectl get all –n velero (velero resources)
kubectl get crds –n velero
```

create backup unsing velero

```
kubectl get backups -n velero # check backup resources, first time there is no backup
velero backup get # check backup resources, first time there is no backup
velero backup create “backup name” # backup from all resources inside k8s cluster.
velero backup create “backup name” --include-namespaces “namespace name” # get a backup from resources inside
specific namespace.
velero backup create “backup name” --include-namespaces “namespace name” --exclude-resource configmap # get
backup from all resources of given namespace except configmap resources
velero backup create “backup name” --exclude-namespaces “namespace name” # backup from all k8s cluster except
given namespace.
```

Verify backups

```
kubectl get backups -n velero
velero backup get
velero backup describe “backup name”
```

restore backup of deleted namespace

```
velero restore create “restore job name” --from-backup “backup name”
velero restore describe “restore job name” # check status of restore job.
kubectl get all -n “deleted namespace” # check that resoruces restored successfully or not.
```

Delete velero restore job after successfully restored.

```
velero restore get # get the name of restore job.
kubectl get restore -n velero # verify that restore is successful.
velero restore delete “restore job name” or --all
```

schedule backups in velero

```
velero schedule create “schedule name” --schedule = “* * * * *” --include-namespaces “namespace name”
velero describe schedule “schedule name”
velero backup get # get the list of backups that runs every minute based on above schedule
kubectl get schedule -n velero
kubectl get backups -n velero
```

delete schedule backup

```
velero schedule delete “schedule name”
```

change TTL value for deleting old backups

```
velero backup create “backup name” --include-namespace “namespace name” --ttl 365d
```