How to Backup and Restore Milvus Data Between Standalone Clusters

- Access the first Standalone Cluster and create a pod using `vector-db` namespace.

```
apiVersion: v1
kind: Pod
metadata:
  name: milvus-backup-pod
  namespace: vector-db
spec:
  containers:
  - name: python-container
    image: python:3.8
    command: [ "sleep", "infinity" ]
```

- Exec into the pod using

```
kubectl exec -it milvus-backup-pod -n vector-db -- bash
```

- Install `milvus-cli` and `vim`

```
apt-get update
pip install milvus-cli
apt-get install vim
```

- Create a new file to connect to the Milvus service.

```
vim connect-milvus.py
```

- Paste this script in the new Python file.

```
from pymilvus import connections, utility

# Connect to Milvus
connections.connect(
    alias="default",
    host='prod-milvus-milvus.vector-db.svc.cluster.local',
    port='19530'
)

# Check connection by listing collections
try:
    collections = utility.list_collections()
    print("Successfully connected to Milvus. Collections:")
    for collection in collections:
        print(f"- {collection}")
except Exception as e:
    print(f"Failed to connect to Milvus: {e}")
```

- Run the Python script using

```
python3 connect-milvus.py
```

- Once you run the script, the pod gets connected to the Milvus service and it will list all the collections which are available in the Milvus.
- Now we need to clone the Milvus Backup tool repository to take the backup of Milvus data in Minio.

```
apt-get update
```

```
apt install git -y
apt-get install wget -y
wget https://go.dev/dl/go1.21.4.linux-amd64.tar.gz -O go.tar.gz
tar -xzvf go.tar.gz -C /usr/local
echo export PATH=$HOME/go/bin:/usr/local/go/bin:$PATH >> ~/.profile
source ~/.profile
go version

git clone https://github.com/zilliztech/milvus-backup.git
cd milvus-backup
go get
go build
```

- Edit the backup configuration file

```
cd configs/
vim backup.yaml
```

```
# Configures the system log output.
log:
  level: info # Only supports debug, info, warn, error, panic, or fatal. Default 'info'.
  console: true # whether print log to console
  file:
    rootPath: "logs/backup.log"

http:
  simpleResponse: true

# milvus proxy address, compatible to milvus.yaml
milvus:
  address: prod-milvus-milvus.vector-db.svc.cluster.local
  port: 19530
  authorizationEnabled: false
  # tls mode values [0, 1, 2]
  # 0 is close, 1 is one-way authentication, 2 is two-way authentication.
  tlsMode: 0
  user: "username"
  password: "password"

# Related configuration of minio, which is responsible for data persistence for Milvus.
minio:
  # cloudProvider: "minio" # deprecated use storageType instead
  storageType: "minio" # support storage type: local, minio, s3, aws, gcp, ali(aliyun), azure, tc(tencent)

  address: prod-milvus-minio.vector-db.svc.cluster.local  # Address of MinIO/S3
  port: 9000   # Port of MinIO/S3
  accessKeyID: minioadmin # accessKeyID of MinIO/S3
  secretAccessKey: minioadmin  # MinIO/S3 encryption string
  useSSL: false # Access to MinIO/S3 with SSL
  useIAM: false
  iamEndpoint: ""

  bucketName: "prod-milvus" # Milvus Bucket name in MinIO/S3, make it the same as your milvus instance
  rootPath: "files" # Milvus storage root path in MinIO/S3, make it the same as your milvus instance

  # only for azure
  backupAccessKeyID: # accessKeyID of MinIO/S3
  backupSecretAccessKey:  # MinIO/S3 encryption string

```

```
  backupBucketName: "prod-milvus-new"  # Bucket name to store backup data. Backup data will store to backupBucketName/backupRootPath
  backupRootPath: "backup"  # Rootpath to store backup data. Backup data will store to backupBucketName/backupRootPath

backup:
  maxSegmentGroupSize: 6G

  parallelism:
    # collection level parallelism to backup
    backupCollection: 4
    # thread pool to copy data. reduce it if blocks your storage's network bandwidth
    copydata: 128
    # Collection level parallelism to restore
    restoreCollection: 2

  # keep temporary files during restore, only use to debug
  keepTempFiles: false

    # Pause GC during backup through Milvus Http API.
  gcPause:
    enable: false
    seconds: 7200
    address: http://prod-milvus-milvus.vector-db.svc.cluster.local:9091
```

- Take the backup of Milvus Collection in Minio using

```
cd ..
./milvus-backup create -n prod_milvus_new
```

- Backup of Milvus collection in Minio is successfull.
- Now we will verify in Minio whether the collections are present or not.
- Download all the required packages (minio-client) to connect to Minio through pod.

```
apt-get update
apt install wget -y
wget https://dl.min.io/client/mc/release/linux-amd64/mc
chmod +x mc
mv mc /usr/local/bin/
```

- Connect to minio client using

```
mc alias set myminio http://prod-milvus-minio.vector-db.svc.cluster.local:9000 minioadmin minioadmin
```

- List the minio buckets

```
mc ls myminio
```

- If minio bucket called `prod-milvus-new` is present, then the Data has been successfully backed up in Minio.
- Now we will copy the collections from Minio to S3 buckets.
- In AWS s3 service in `us-east-2` region create an empty bucket with name `prod-milvus-new`
- Now connect to the AWS S3 service inside pod using

```
mc alias set s3 https://s3.us-east-2.amazonaws.com AWS_Access_Key_ID AWS_Access_Secret_Key
```

- Copy the Minio Bucket (`prod-milvus-new`) data to AWS S3 (`prod-milvus-new`) bucket.

```
mc mirror myminio/prod-milvus-new s3/prod-milvus-new
```

- Verify in AWS S3 service UI whether the data has been copied or not in the bucket.

- Now take the access of Second Standalone Cluster where we want to restore the Milvus Data
- Create a pod using `vector-db` namespace.

```
apiVersion: v1
kind: Pod
metadata:
  name: milvus-backup-restore-pod
  namespace: vector-db
spec:
  containers:
  - name: ubuntu-container
    image: ubuntu:20.04
    command: [ "sleep", "infinity" ]
    securityContext:
      runAsUser: 0
      runAsGroup: 0
```

- Exec into the pod using

```
kubectl exec -it milvus-backup-restore-pod -n vector-db -- bash
```

- Install `milvus-cli` and `vim`

```
apt-get update
apt-get install pip -y
pip install milvus-cli
apt-get install vim
```

- Create a new file to connect to the Milvus service.

```
vim connect-milvus.py
```

- Paste this script in the new Python file.

```
from pymilvus import connections, utility

# Connect to Milvus
connections.connect(
    alias="default",
    host='prod-milvus-milvus.vector-db.svc.cluster.local',
    port='19530'
)

# Check connection by listing collections
try:
    collections = utility.list_collections()
    print("Successfully connected to Milvus. Collections:")
    for collection in collections:
        print(f"- {collection}")
except Exception as e:
    print(f"Failed to connect to Milvus: {e}")
```

- Run the Python script using

```
python3 connect-milvus.py
```

- Once you run the script, the pod gets connected to the Milvus service and it will list all the collections which are available in the Milvus.
- Download all the required packages (minio-client) to connect to Minio through pod.

```
apt-get update
apt install wget -y
wget https://dl.min.io/client/mc/release/linux-amd64/mc
chmod +x mc
mv mc /usr/local/bin/
```

- Connect to minio client using

```
mc alias set myminio http://prod-milvus-minio.vector-db.svc.cluster.local:9000 minioadmin minioadmin
```

- Create an empty minio bucket using

```
mc mb myminio/prod-milvus-new
```

- Connect to AWS S3 service from pod using

```
mc alias set s3 https://s3.us-east-2.amazonaws.com AWS_Access_Key_ID AWS_Access_Secret_Key
```

- Copy the S3 bucket (prod-milvus-new) data to the newly created minio bucket (prod-milvus-new) in second standalone cluster using

```
mc mirror s3/prod-milvus-new myminio/prod-milvus-new
```

- Now we need to clone the Milvus Backup tool repository to restore the Milvus Data in second standalone cluster.

```
apt-get update
apt install git -y
apt-get install wget -y
wget https://go.dev/dl/go1.21.4.linux-amd64.tar.gz -O go.tar.gz
tar -xzvf go.tar.gz -C /usr/local
echo export PATH=$HOME/go/bin:/usr/local/go/bin:$PATH >> ~/.profile
source ~/.profile
go version

git clone https://github.com/zilliztech/milvus-backup.git
cd milvus-backup
go get
go build
```

- Edit the backup configuration file

```
cd configs/
vim backup.yaml
```

```
# Configures the system log output.
log:
  level: info # Only supports debug, info, warn, error, panic, or fatal. Default 'info'.
  console: true # whether print log to console
  file:
    rootPath: "logs/backup.log"

http:
  simpleResponse: true

# milvus proxy address, compatible to milvus.yaml
milvus:
  address: prod-milvus-milvus.vector-db.svc.cluster.local
  port: 19530
```

```
  authorizationEnabled: false
  # tls mode values [0, 1, 2]
  # 0 is close, 1 is one-way authentication, 2 is two-way authentication.
  tlsMode: 0
  user: "username"
  password: "password"

# Related configuration of minio, which is responsible for data persistence for Milvus.
minio:
  # cloudProvider: "minio" # deprecated use storageType instead
  storageType: "minio" # support storage type: local, minio, s3, aws, gcp, ali(aliyun), azure, tc(tencent)

  address: prod-milvus-minio.vector-db.svc.cluster.local  # Address of MinIO/S3
  port: 9000   # Port of MinIO/S3
  accessKeyID: minioadmin # accessKeyID of MinIO/S3
  secretAccessKey: minioadmin  # MinIO/S3 encryption string
  useSSL: false # Access to MinIO/S3 with SSL
  useIAM: false
  iamEndpoint: ""

  bucketName: "prod-milvus" # Milvus Bucket name in MinIO/S3, make it the same as your milvus instance
  rootPath: "files" # Milvus storage root path in MinIO/S3, make it the same as your milvus instance

  # only for azure
  backupAccessKeyID: # accessKeyID of MinIO/S3
  backupSecretAccessKey:  # MinIO/S3 encryption string

  backupBucketName: "prod-milvus-new"  # Bucket name to store backup data. Backup data will store to
backupBucketName/backupRootPath
  backupRootPath: "backup"  # Rootpath to store backup data. Backup data will store to
backupBucketName/backupRootPath

backup:
  maxSegmentGroupSize: 6G

  parallelism:
    # collection level parallelism to backup
    backupCollection: 4
    # thread pool to copy data. reduce it if blocks your storage's network bandwidth
    copydata: 128
    # Collection level parallelism to restore
    restoreCollection: 2

  # keep temporary files during restore, only use to debug
  keepTempFiles: false

    # Pause GC during backup through Milvus Http API.
  gcPause:
    enable: false
    seconds: 7200
    address: http://prod-milvus-milvus.vector-db.svc.cluster.local:9091
```

- Now restore the Milvus Data using

```
cd ..
./milvus-backup restore -n prod_milvus_new
```

- Restoration of data is successful in Second Standalone Cluster Milvus
- To verify whether the same collections exist or not in the second Standalone cluster after restoring data, run the python script

```
python3 connect-milvus.py
```

- If the milvus collections from first standalone cluster exactly matches to milvus collections from second standalone then our Backup and Restore procedure was successful.
- We can also check from UI side whether collections with data exist or not in the Second Standalone Cluster.