milvus-backup tool practice

Preparation

Download the latest binary from milvus-backup repo: https://github.com/zilliztech/milvus-backup/releases

- For Mac, download **milvus-backup_Darwin_arm64.tar.gz** or **milvus-backup_Darwin_x86_64.tar.gz**
- For Linux, download **milvus-backup_Linux_arm64.tar.gz** or **milvus-backup_Linux_x86_64.tar.gz**

Download the configuration file:

```
wget https://github.com/zilliztech/milvus-backup/blob/main/configs/backup.yaml
```

Extract the tar file to a directory. Put backup.yaml to **configs/backup.yaml** under this directory. The file structure looks like this:

```
├── configs
│   └── backup.yaml
├── milvus-backup
└── README.md
```

Command Usage

In the terminal, type command `milvus-backup help` to show the command line usage:

```
milvus-backup is a backup&restore tool for milvus.

Usage:
  milvus-backup [flags]
  milvus-backup [command]

Available Commands:
  check       check if the connects is right.
  create      create subcommand create a backup.
  delete      delete subcommand delete backup by name.
  get         get subcommand get backup by name.
  help        Help about any command
  list        list subcommand shows all backup in the cluster.
  restore     restore subcommand restore a backup.
  server      server subcommand start milvus-backup RESTAPI server.

Flags:
      --config string   config YAML file of milvus (default "backup.yaml")
  -h, --help            help for milvus-backup

Use "milvus-backup [command] --help" for more information about a command.
```

Type command `milvus-backup create --help` to show the usage of "create a backup":

```
Usage:
  milvus-backup create [flags]

Flags:
  -n, --name string                    backup name, if unset will generate a name automatically
  -c, --colls string                   collectionNames to backup, use ',' to connect multiple collections
  -d, --databases string               databases to backup
  -a, --database_collections string   databases and collections to backup, json format: {"db1":["c1", "c2"],"db2":[]}
  -f, --force                          force backup, will skip flush, should make sure data has been stored into disk when using it
      --meta_only                      only backup collection meta instead of data
  -h, --help                           help for create

```

Type command `milvus-backup restore --help` to show the usage of "restore a backup":

```
Usage:
  milvus-backup restore [flags]

```

```
Flags:
  -n, --name string                      backup name to restore
  -c, --collections string               collectionNames to restore
  -s, --suffix string                    add a suffix to collection name to restore
  -r, --rename string                    rename collections to new names, format: db1.collection1:db2.collection1_new,db1.collection2:db2.collection2_new
  -d, --databases string                 databases to restore, if not set, restore all databases
  -a, --database_collections string   databases and collections to restore, json format: {"db1":["c1", "c2"],"db2":[]}
      --meta_only                        if true, restore meta only
      --restore_index                    if true, restore index
      --use_auto_index                   if true, replace vector index with autoindex
      --drop_exist_collection            if true, drop existing target collection before create
      --drop_exist_index                 if true, drop existing index of target collection before create
      --skip_create_collection           if true, will skip collection, use when collection exist, restore index or data
  -h, --help                             help for restore

```

From the usage we know:

- "-c" is to specify a collection's name to backup.
- "-n" defines a name for the backup operation, the backup data files will be copied to a directory that has this name.
- "-s" defines a suffix, the restored collection's name is combined with the origin collection's name and this suffix.

Configurations

In the **configs/backup.yaml** file we can see the following sections:

1. "log" section, config the log behavior of milvus-backup tool

```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"
```

2. "milvus" section, config the address and connection method of Milvus

```yaml
# milvus proxy address, compatible to milvus.yaml
milvus:
  address: localhost
  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: "root"
  password: "Milvus"
```

3. "minio" section, config the address and connection method of MinIO. And the backup/restore target path.

- **minio.backetName** is the bucket name which the Milvus is using as storage.
- **minio.rootPath** is the root path under the **minio.backetName** which the Milvus is using to store data files. Milvus-backup tool copies data files from this path.
- **minio.backupBacketName** is the target bucket in which the milvus-backup stores the backup files.
- **minio.backupRootPath** is the root path under the **minio.backupBacketName** in which the milvus-backup stores the backup files.

```yaml
# 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: localhost # 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: "a-bucket" # 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: minioadmin  # accessKeyID of MinIO/S3
  backupSecretAccessKey: minioadmin # MinIO/S3 encryption string

  backupBucketName: "a-bucket" # 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
```

> For example, assuming there is a collection named "A" in the Milvus. Milvus stores its data files under this path:
>
> `[minio.bucketName]/[minio.rootPath]/insert_log/[ID of collection A]`
>
> When you create a backup for this collection, the milvus-backup copies data files from the above path to the below path:
>
> `[minio.backupBucketName]/[minio.backupRootPath]/[backup name]`
>
> When you restore this backup to a new collection, the milvus-backup calls Milvus bulkinsert interface to import these files. The milvus-backup only sends a S3 related path to the Milvus. In this case, the path is :
>
> `/[minio.rootPath]/insert_log/[ID of collection A]`.
>
> Milvus tries to read files from this path, it requires the `minio.backupBucketName` must be the bucket that the target Milvus is using, because Milvus can only access one S3 bucket at runtime.
>
> After restore, a new collection's data files are stored in this path:
>
> `[minio.bucketName]/[minio.rootPath]/insert_log/[ID of new collection]`

Backup/Restore

Generally, there are 4 types of backup/restore use cases:

1. Copy a collection within a Milvus instance.
2. Copy a collection between two Milvus in one S3, one bucket, and different root paths.
3. Copy a collection between two Milvus in one S3 with different buckets.

4. Copy a collection between two Milvus in different S3.

One Milvus

Purpose

Backup/restore a collection to a new collection in the same Milvus service. Assuming there is a collection named "coll" in the Milvus, we backup/restore it to a new collection named "coll_bak". The Milvus uses the S3 bucket "bucket_A" as storage.

Milvus Configuration

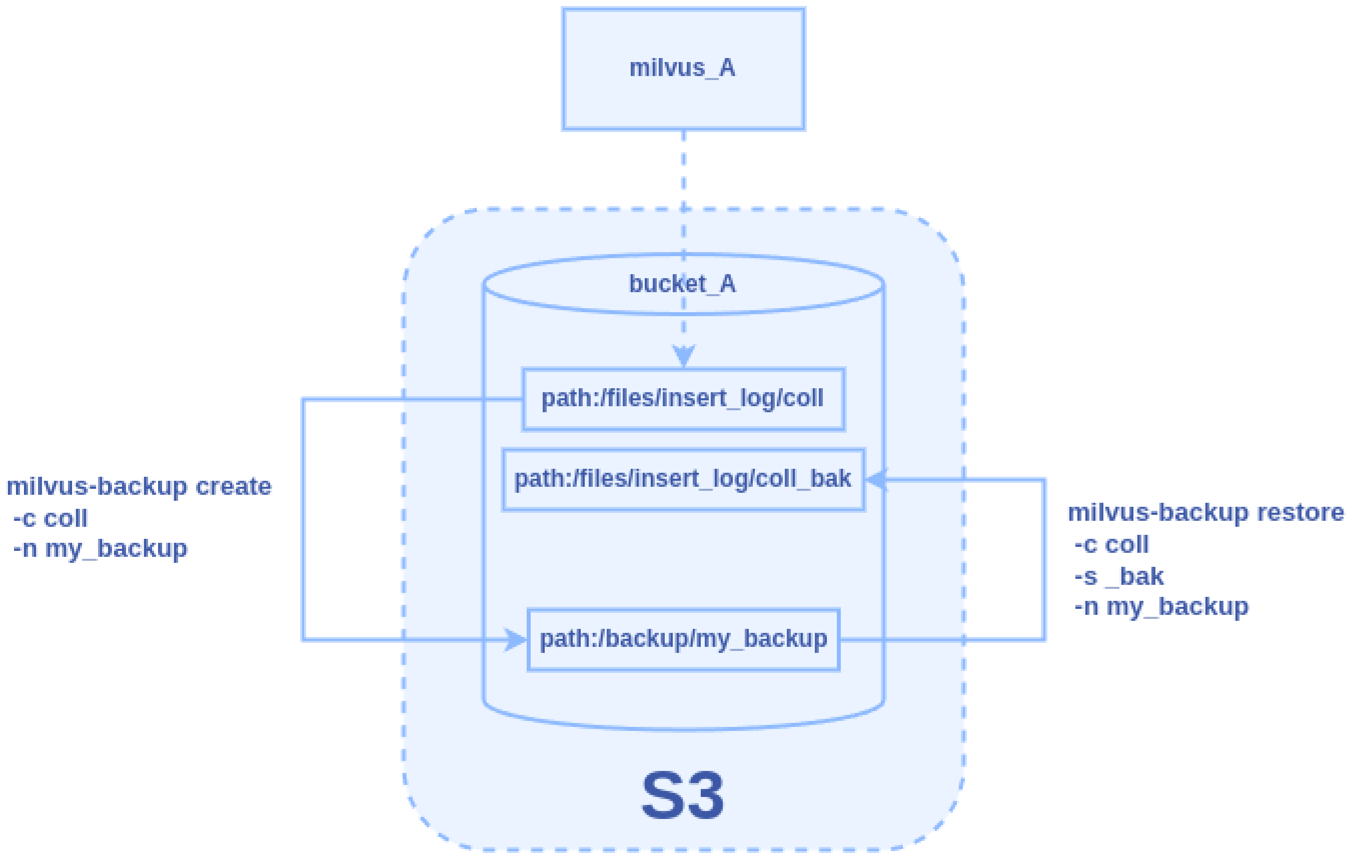
The Milvus is deployed with this configuration in the **milvus.yaml**:

- **minio.address** is "localhost"
- **minio.bucketName** is "bucket_A"
- **minio.rootPath** is "files"

```yaml
minio:
  address: localhost # 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
  ssl:
    tlsCACert: /path/to/public.crt # path to your CACert file, ignore when it is empty
  bucketName: bucket_A # Bucket name in MinIO/S3
  rootPath: files # The root path where the message is stored in MinIO/S3
```

Workflow

> Note that Milvus organizes data path by collection's ID, not collection's name. In this picture, we write the path as collection's name just for easy understanding.



1. In the **configs/backup.yaml**

- Set **milvus.address** to be "localhost".
- Set **minio.bucketName** and **minio.backupBucketName** to be "bucket_A".
- Set **minio.rootPath** to be "files".
- Set **minio.backupRootPath** to a path that is different from the **minio.rootPath** (to avoid contamination of Milvus storage), here we set it to be "backup".

```yaml
# 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: localhost # 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: "bucket_A" # 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: minioadmin  # accessKeyID of MinIO/S3
  backupSecretAccessKey: minioadmin # MinIO/S3 encryption string

  backupBucketName: "bucket_A" # 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
```

2. Use **/create** command to create a backup. The name is "my_backup". After the command succeeds, you will see the path **bucket_A/backup/my_backup** is created in the S3.

```
./milvus-backup create -c coll -n my_backup
```

3. Use **/restore** command to restore the backup to a new collection. After the command succeeds, you will see a new collection named "coll_bak" is created in the Milvus. The new collection's data files are stored in `bucket_A/files/insert_log/[ID of new collection]`

```
./milvus-backup restore -c coll -n my_backup -s _bak
```

Note: if you want to restore the index, you can append **--restore_index** to the command.

Two Milvus share one bucket with different root paths

Purpose

Backup a collection from a Milvus and restore it to another Milvus that shares the same bucket but a different root path. Assuming there is a collection named "coll" in the milvus_A, we backup/restore it to a new collection named "coll_bak" to milvus_B. The two Milvus share the same bucket "bucket_A" as storage, but they have different root paths.

Milvus Configuration

The **milvus.yaml** of milvus_A:

- **minio.address** is "localhost"
- **minio.bucketName** is "bucket_A"
- **minio.rootPath** is "files_A"

```yaml
minio:
  address: localhost # 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
  ssl:
    tlsCACert: /path/to/public.crt # path to your CACert file, ignore when it is empty
  bucketName: bucket_A # Bucket name in MinIO/S3
  rootPath: files_A # The root path where the message is stored in MinIO/S3
```

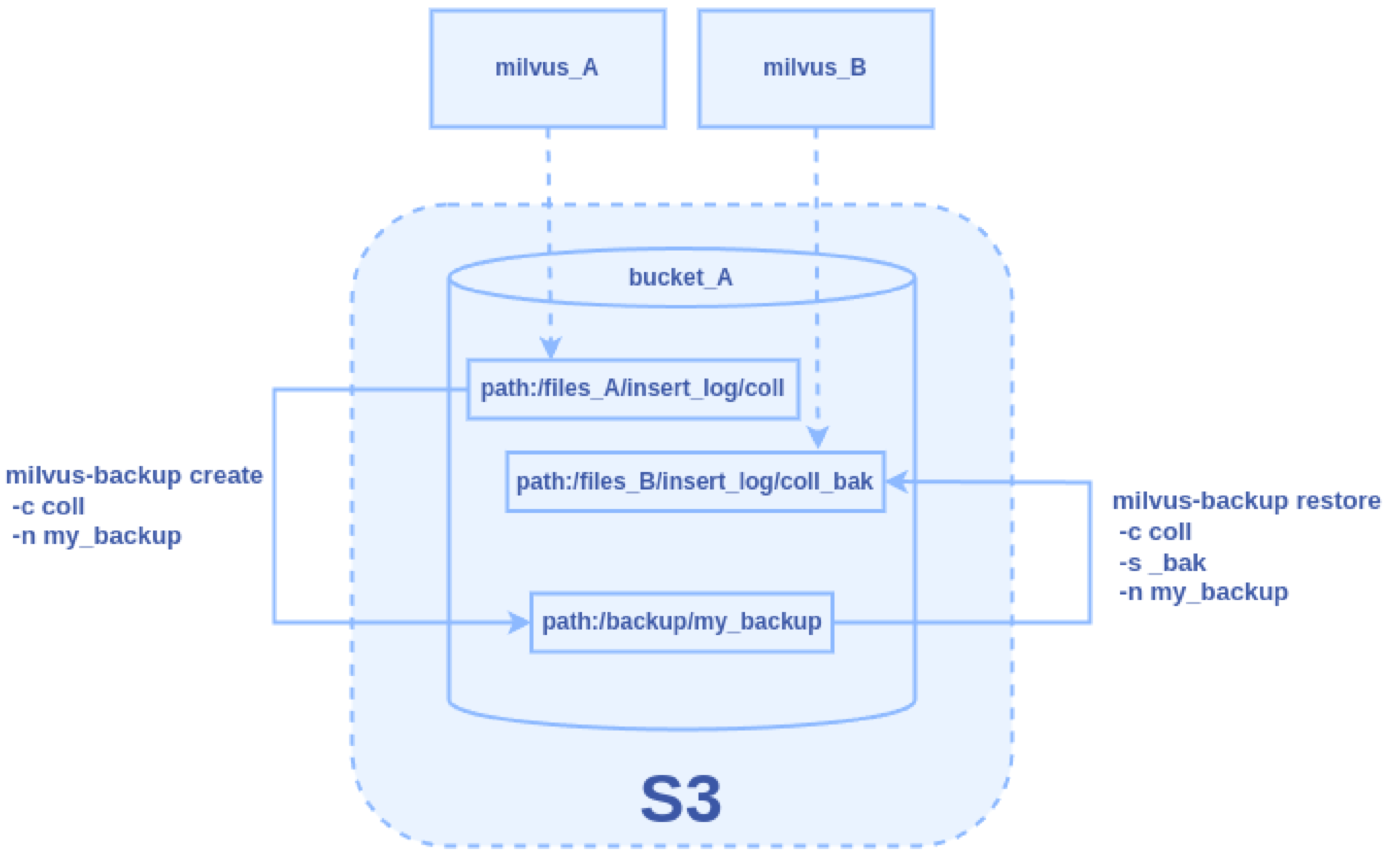
The **milvus.yaml** of milvus_B:

- **minio.address** is "localhost"
- **minio.bucketName** is "bucket_A"
- **minio.rootPath** is "files_B"

```yaml
minio:
  address: localhost # 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
  ssl:
    tlsCACert: /path/to/public.crt # path to your CACert file, ignore when it is empty
  bucketName: bucket_A # Bucket name in MinIO/S3
  rootPath: files_B # The root path where the message is stored in MinIO/S3
```

Workflow

> Note that Milvus organizes data path by collection's ID, not collection's name. In this picture, we write the path as collection's name just for easy understanding the internal workflow.


milvus_A
milvus_B
bucket_A
path:/files_A/insert_log/coll
path:/files_B/insert_log/coll_bak
milvus-backup create -c coll -n my_backup
path:/backup/my_backup
milvus-backup restore -c coll -s _bak -n my_backup
S3

1. In the **configs/backup.yaml**

- Set **milvus.address** to be the address of milvus_A.
- Set **minio.bucketName** and **minio.backupBucketName** to be "bucket_A".
- Set **minio.rootPath** to be "files_A".
- Set **minio.backupRootPath** to a path that is different from the **minio.rootPath** (to avoid contamination of Milvus storage), here we set it to be "backup".

```
# milvus proxy address, compatible to milvus.yaml
milvus:
  address: milvus_A
  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: "root"
  password: "Milvus"

# 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: milvus_A # 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: "bucket_A" # Milvus Bucket name in MinIO/S3, make it the same as your milvus instance
  rootPath: "files_A" # Milvus storage root path in MinIO/S3, make it the same as your milvus instance

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

  backupBucketName: "bucket_A" # 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
```

2. Use **/create** command to create a backup. The name is "my_backup". After the command succeeds, you will see the path **backup_A/backup/my_backup** is created in the S3.

```
./milvus-backup create -c coll -n my_backup
```

3. Modify the **configs/backup.yaml**

- Set **milvus.address** to be the address of milvus_B, so that the restore command can restore the collection to milvus_B.

- Set **milvus.port** to the port of milvus_B.
- Set **minio.rootPath** to be "files_B".

```yaml
# milvus proxy address, compatible to milvus.yaml
milvus:
  address: milvus_B
  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: "root"
  password: "Milvus"

# 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: milvus_B # 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: "bucket_A" # Milvus Bucket name in MinIO/S3, make it the same as your milvus instance
  rootPath: "files_B" # Milvus storage root path in MinIO/S3, make it the same as your milvus instance

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

  backupBucketName: "bucket_A" # 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
```

4. Use **/restore** command to restore the backup to a new collection. After the command succeeds, you will see a new collection named "coll_bak" is created in the milvus_B. The new collection's data files are stored in `bucket_A/files_B/insert_log/[ID of new collection]`

```
./milvus-backup restore -c coll -n my_backup -s _bak
```

Two Milvus in one S3, different buckets

Purpose

Backup a collection from a Milvus and restore it to another Milvus in different buckets, but the two Milvus are in the same S3. Assuming there is a collection named "coll" in the milvus_A, we backup/restore it to a new collection named "coll_bak" to milvus_B. The milvus_A is using bucket "bucket_A" as storage, the milvus_B is using bucket "bucket_B" as storage.

Milvus Configuration

The **milvus.yaml** of milvus_A:

- **minio.address** is "localhost"
- **minio.bucketName** is "bucket_A"
- **minio.rootPath** is "files"

```yaml
minio:
  address: localhost # 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
  ssl:
    tlsCACert: /path/to/public.crt # path to your CACert file, ignore when it is empty
  bucketName: bucket_A # Bucket name in MinIO/S3
  rootPath: files # The root path where the message is stored in MinIO/S3
```

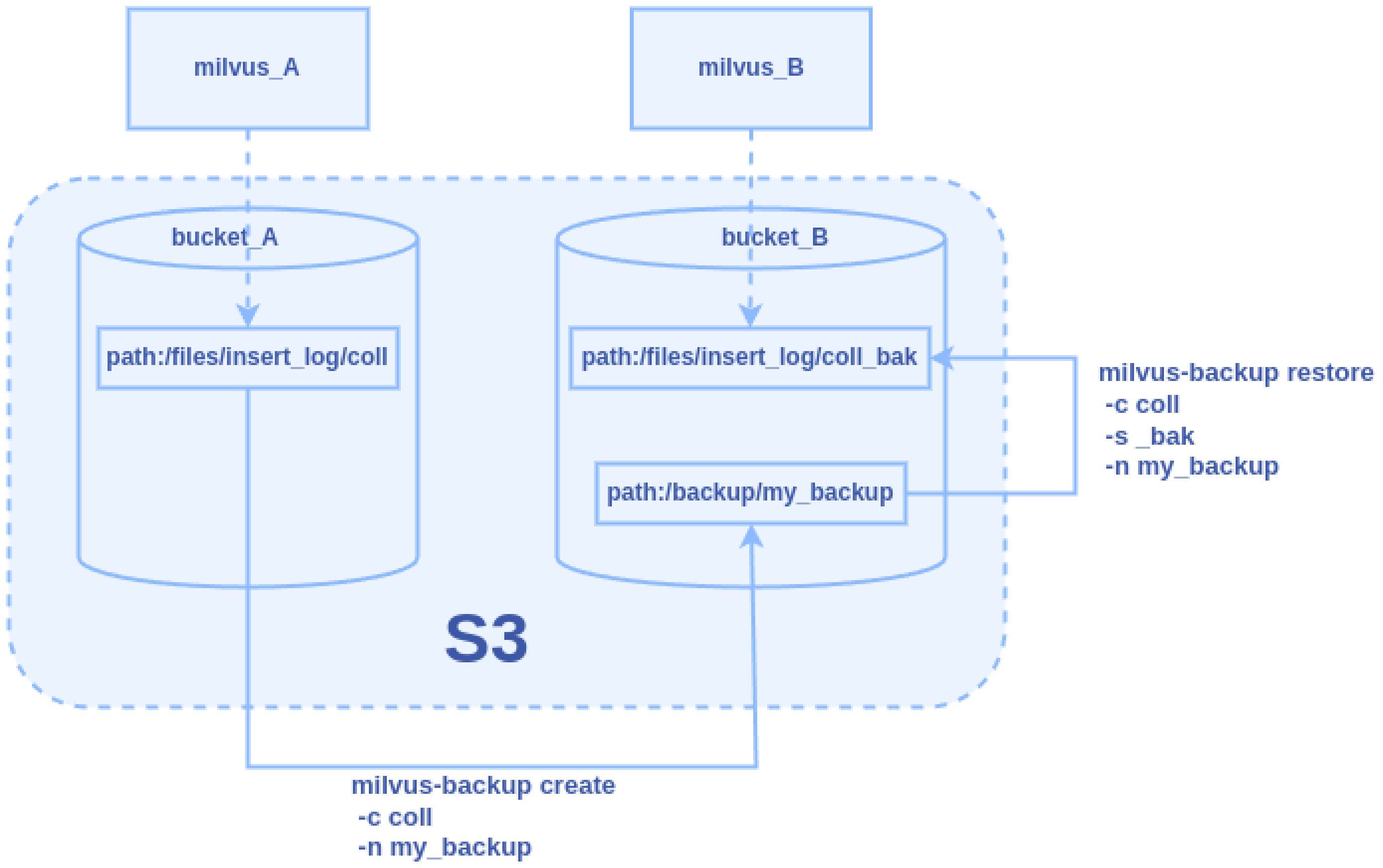
The **milvus.yaml** of milvus_B:

- **minio.address** is "localhost"
- **minio.bucketName** is "bucket_B"
- **minio.rootPath** is "files"

```
minio:
  address: localhost # 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
  ssl:
    tlsCACert: /path/to/public.crt # path to your CACert file, ignore when it is empty
  bucketName: bucket_B # Bucket name in MinIO/S3
  rootPath: files # The root path where the message is stored in MinIO/S3
```

Workflow

> Note that Milvus organizes data path by collection's ID, not collection's name. In this picture, we write the path as collection's name just for easy understanding the internal workflow.



1. In the **configs/backup.yaml**

- Set **milvus.address** to be the address of milvus_A.
- Set **minio.bucketName** to be "bucket_A".
- Set **minio.rootPath** to be "files".
- Set **minio.backupBucketName** to be "bucket_B"
- Set **minio.backupRootPath** to a path that is different from the **minio.rootPath** (to avoid contamination of Milvus storage), here we set it to be "backup".

```yaml
# milvus proxy address, compatible to milvus.yaml
milvus:
  address: milvus_A
  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: "root"
  password: "Milvus"

# 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: localhost # 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: "bucket_A" # 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: minioadmin  # accessKeyID of MinIO/S3
  backupSecretAccessKey: minioadmin # MinIO/S3 encryption string

  backupBucketName: "bucket_B" # 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
```

2. Use **/create** command to create a backup. The name is "my_backup". After the command succeeds, you will see the path **bucket_B/backup/my_backup** is created in the S3.

```
./milvus-backup create -c coll -n my_backup
```

3. Modify the **configs/backup.yaml**

- Set **milvus.address** to be the address of milvus_B, so that the restore command can restore the collection to milvus_B.
- Set **milvus.port** to the port of milvus_B.
- Set **minio.bucketName** to be "bucket_B"
- Set **minio.rootPath** to be "files".

```
# milvus proxy address, compatible to milvus.yaml
milvus:
  address: milvus_B
  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: "root"
  password: "Milvus"

# 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: localhost # Address of MinIO/S3
  port: 9000   # Port of MinIO/S3
  accessKeyID: minioadmin  # accessKeyID of MinIO/S3
  secretAccessKey: minioadmin # MinIO/S3 encryption string
```

```yaml
  useSSL: false # Access to MinIO/S3 with SSL
  useIAM: false
  iamEndpoint: ""

  bucketName: "bucket_B" # 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: minioadmin  # accessKeyID of MinIO/S3
  backupSecretAccessKey: minioadmin # MinIO/S3 encryption string

  backupBucketName: "bucket_B" # 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
```

4. Use **/restore** command to restore the backup to a new collection. After the command succeeds, you will see a new collection named "coll_bak" is created in the milvus_B. The new collection's data files are stored in `bucket_B/files/insert_log/[ID of new collection]`

```
./milvus-backup restore -c coll -n my_backup -s _bak
```

Two Milvus in two S3

Purpose

Backup a collection from a Milvus and restore it to another Milvus in different S3. Assuming there is a collection named "coll" in the milvus_A, we backup/restore it to a new collection named "coll_bak" to milvus_B. The milvus_A is using bucket "bucket_A" as storage, the milvus_B is using bucket "bucket_B" as storage, they are using different S3/MinIO addresses.

Milvus Configuration

The **milvus.yaml** of milvus_A:

- **minio.address** is address of minio_A

- **minio.bucketName** is "bucket_A"
- **minio.rootPath** is "files"

```yaml
minio:
  address: minio_A # 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
  ssl:
    tlsCACert: /path/to/public.crt # path to your CACert file, ignore when it is empty
  bucketName: bucket_A # Bucket name in MinIO/S3
  rootPath: files # The root path where the message is stored in MinIO/S3
```

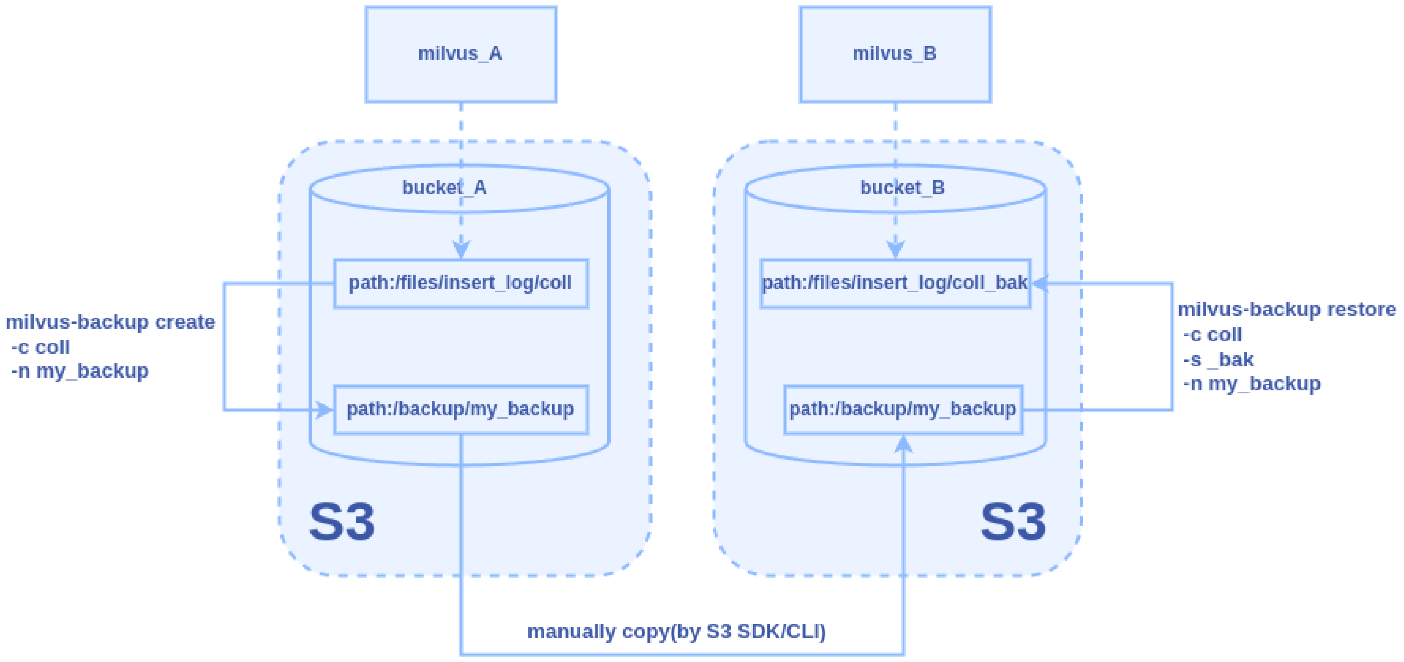
The **milvus.yaml** of milvus_B:

- **minio.address** is address of minio_B
- **minio.bucketName** is "bucket_B"
- **minio.rootPath** is "files"

```yaml
minio:
  address: minio_B # 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
  ssl:
    tlsCACert: /path/to/public.crt # path to your CACert file, ignore when it is empty
  bucketName: bucket_B # Bucket name in MinIO/S3
  rootPath: files # The root path where the message is stored in MinIO/S3
```

Workflow

> Note that Milvus organizes data path by collection's ID, not collection's name. In this picture, we write the path as collection's name just for easy understanding the internal workflow.


milvus_A
milvus_B
bucket_A
bucket_B
path:/files/insert_log/coll
path:/files/insert_log/coll_bak
path:/backup/my_backup
path:/backup/my_backup
milvus-backup create -c coll -n my_backup
milvus-backup restore -c coll -s _bak -n my_backup
S3
S3
manually copy(by S3 SDK/CLI)

1. In the **configs/backup.yaml**

- Set **milvus.address** to be the address of milvus_A.
- Set **minio.address** to be the address of minio_A.
- Set **minio.bucketName** to be "bucket_A".
- Set **minio.rootPath** to be "files".
- Set **minio.backupBucketName** to be "bucket_A"
- Set **minio.backupRootPath** to a path that is different from the **minio.rootPath** (to avoid contamination of Milvus storage), here we set it to be "backup".

```yaml
# milvus proxy address, compatible to milvus.yaml
milvus:
  address: milvus_A
  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: "root"
  password: "Milvus"

# 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: minio_A # 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: "bucket_A" # 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: minioadmin  # accessKeyID of MinIO/S3
  backupSecretAccessKey: minioadmin # MinIO/S3 encryption string

  backupBucketName: "bucket_A" # 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
```

2. Use **/create** command to create a backup. The name is "my_backup". After the command succeeds, you will see the path **bucket_A/backup/my_backup** is created in the minio_A.

```
./milvus-backup create -c coll -n my_backup
```

3. Manually copy the **bucket_A/backup/my_backup** from minio_A to **bucket_B/backup/my_backup** of minio_B. You can use a S3 compatible SDK or Client to do the work.

4. Modify the **configs/backup.yaml**

- Set **milvus.address** to be the address of milvus_B, so that the restore command can restore the collection to milvus_B.
- Set **milvus.port** to the port of milvus_B.

- Set **minio.address** to be the address of minio_B.
- Set **minio.bucketName** to be "bucket_B"
- Set **minio.rootPath** to be "files".
- Set **minio.backupBucketName** to be "bucket_B" and **minio.backupRootPath** to be "backup" since we have copied the backup files to this place.

```yaml
# milvus proxy address, compatible to milvus.yaml
milvus:
  address: milvus_B
  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: "root"
  password: "Milvus"

# 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: minio_B # 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: "bucket_B" # 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: minioadmin  # accessKeyID of MinIO/S3
  backupSecretAccessKey: minioadmin # MinIO/S3 encryption string

  backupBucketName: "bucket_B" # 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
```

5. Use **/restore** command to restore the backup to a new collection. After the command succeeds, you will see a new collection named "coll_bak" is created in the milvus_B. The new collection's data files are stored in `bucket_B/files/insert_log/[ID of new collection]` of minio_B.

```
./milvus-backup restore -c coll -n my_backup -s _bak
```