

MEP 10 -- Support Collection Alias

Current state: Under Discussion

ISSUE: <https://github.com/milvus-io/milvus/issues/4812>

PRs: TODO

Keywords: Collection Alias, Collection Hot Reload

Released: TODO

Summary

1. As the name indicates, CollectionAlias is an alias to an existing collection.
2. The collection alias can be updated to a new collection.
3. Within RootCoordinator, Proxy, and all the key components, CollectionName and CollectionAlias are equal.
e.g. `MetaTable.GetCollectionByName(collectionName string, ts typeutil.Timestamp)` can receive CollectionAlias and return corresponding CollectionInfo.
4. CollectionAlias CollectionName = . CollectionAlias cannot collide with existing CollectionNames.

Motivation

In recommendation systems, there is a need to update the whole collection(e.g. User embedding) periodically(hourly, daily). To update the whole collection, we can:

1. upsert all the items.
 - It will be **slow** if the collection is huge.
2. Insert new collection B data & rename the new collection B as A.
 - In a distributed system, it is pretty **costly**(Performance, Availability, Complexity) to update the state globally.

As CollectionAlias works as an extra pointer to the existing collection in the RootCoordinator, we can implement collection hot reloading at a much lower cost compared to the **1, 2** approaches.

With CollectionAlias, we can implement collection hot reloading functionality as such:

1. Create Collection A & insert data.
2. Create CollectionAlias Z -> A.
3. Create Collection B & insert data.
4. Update CollectionAlias Z -> B.
5. Users can access new data with the Z alias.

Public Interfaces

New Public APIs

There will be CollectionAlias related 3 new APIs.

- CreateAlias, DropAlias, AlterAlias

```
// milvus.proto
message CreateAliasRequest {
    common.MsgBase base = 1;
    string collection_name = 2;
    string alias = 3;
}

message DropAliasRequest {
    common.MsgBase base = 1;
    string alias = 2;
}

message AlterAliasRequest{
    common.MsgBase base = 1;
    string collection_name = 2;
    string alias = 3;
}

service MilvusService {
    // NEW
    rpc CreateAlias(SetAliasRequest) returns (common.Status) {}
    // NEW
    rpc DropAlias(DropAliasRequest) returns (common.Status) {}
    // NEW
    rpc AlterAlias(AlterAliasRequest) returns (common.Status) {}
}
```

Changes to Existing APIs

- Users can't drop the collection if the collection is referenced by an alias .
- DescribeCollection now returns aliases

```
message DescribeCollectionResponse {
    common.Status status = 1;
    schema.CollectionSchema schema = 2;
    int64 collectionID = 3;
    repeated string virtual_channel_names = 4;
    repeated string physical_channel_names = 5;
    uint64 created_timestamp = 6;
    uint64 created_utc_timestamp = 7;

    // NEW
    repeated string aliases = 8;
}

service MilvusService {
    // Users are required to drop the aliases first before dropping the collection.
    rpc DropCollection(DropCollectionRequest) returns (common.Status) {}
    // DescribeCollectionResponse contains `aliases` that refer to this collection.
    rpc DescribeCollection(DescribeCollectionRequest) returns (DescribeCollectionResponse) {}
}
```

Design Details

Changes to the MetaTable

```

type metaTable struct {
    client          kv.SnapShotKV // client of a reliable
kv service, i.e. etcd client
    tenantID2Meta   map[typeutil.UniqueID]pb.TenantMeta // tenant id to tenant
meta
    proxyID2Meta    map[typeutil.UniqueID]pb.ProxyMeta // proxy id to proxy
meta
    collID2Meta     map[typeutil.UniqueID]pb.CollectionInfo // collection_id -> meta
    collName2ID     map[string]typeutil.UniqueID // collection name to
collection id
    // NEW
    collAlias2ID    map[string]typeutil.UniqueID
    ...
}

```

As CollectionAlias & CollectionName are equal, GetCollectionByName also checks metaTable.collAlias2ID when getting the collection by name.

```

func (mt *metaTable) GetCollectionByName(collectionName string, ts typeutil.Timestamp) (*pb.CollectionInfo,
error) {
    mt.ddLock.RLock()
    defer mt.ddLock.RUnlock()

    if ts == 0 {
        vid, ok := mt.collName2ID[collectionName]
        if !ok {
            // NEW
            if vid, ok = mt.collAlias2ID[collectionName]; !ok {
                return nil, fmt.Errorf("can't find collection: " + collectionName)
            }
        }
    }
    ...
}

```

CollectionAlias also have to be persisted in the etcd .

```

const (
    ComponentPrefix      = "root-coord"
    TenantMetaPrefix     = ComponentPrefix + "/tenant"
    ProxyMetaPrefix      = ComponentPrefix + "/proxy"
    CollectionMetaPrefix = ComponentPrefix + "/collection"
    SegmentIndexMetaPrefix = ComponentPrefix + "/segment-index"
    IndexMetaPrefix      = ComponentPrefix + "/index"
    // NEW Additions
    CollectionAliasMetaPrefix = ComponentPrefix + "/collection-alias"
)

```

For persistence in etcd, the **key** will be `fmt.Sprintf("%s/%s", CollectionAliasMetaPrefix, CollectionAlias)` and the **value** be `CollectionID`.

Changes to the RootCoordinator

```

// root_coord.proto
service RootCoord {
  // NEW
  // CreateAlias creates 1 to 1 mapping between `alias` and `collection_name`
  // 1. If there no `alias` in the metaTable:
  //     1.1 new `alias` will be added to the metaTable
  //     1.2 `alias` will be persisted in the `etcd`
  //     1.3 `dd_op` will sent to log broker.
  // 2. If there is `alias/collection` in the metaTable:
  //     2.1 An `alias/collection already exists` error will be returned.
  rpc CreateAlias(milvus.CreateAliasRequest) returns (common.Status) {}

  // NEW
  // 1. DropAlias
  //     1.1 Removes existing mapping from the metaTable
  //     1.2 Removes existing mapping from the `etcd`
  //     1.3 `dd_op` will be sent to the log broker.
  //     1.4 Invalidates proxy caches
  rpc DropAlias(milvus.DropAliasRequest) returns (common.Status) {}

  // NEW
  // 1. AlterAlias
  //     1.1 Existing mapping will be updated in metaTable.
  //     1.2 Existing mapping will be updated in `etcd`
  //     1.3 `dd_op` will be sent to log broker.
  //     1.4 Invalidates proxy caches.
  rpc AlterAlias(milvus.AlterAliasRequest) returns (common.Status) {}

  // UPDATED REQUIRED
  // Collection can't be dropped if it is referenced by an `alias`.
  rpc DropCollection(DropCollectionRequest) returns (common.Status) {}
  // UPDATED REQUIRED
  // DescribeCollection now returns `aliases`
  rpc DescribeCollection(DescribeCollectionRequest) returns (DescribeCollectionResponse) {}
}

```

There will be new tasks as `CreateAliasTask`, `DropAliasTask`, and `AlterAliasTask` that handles the actual implementation within the GRPC methods.

Recovery

The recovery process will handle `CreateAliasDDType`, `DropAliasDDType`, `AlterAliasDDType` types.

```

func (c *Core) reSendDdMsg(ctx context.Context) error {
    flag, err := c.MetaTable.Client.Load(DDMsgSendPrefix, 0)
    if err != nil || flag == "true" {
        log.Debug("No un-successful DdMsg")
        return nil
    }

    ddOpStr, err := c.MetaTable.Client.Load(DDOperationPrefix, 0)
    if err != nil {
        log.Debug("DdOperation key does not exist")
        return nil
    }
    var ddOp DdOperation
    if err = json.Unmarshal([]byte(ddOpStr), &ddOp); err != nil {
        return err
    }

    switch ddOp.Type {
    case CreateCollectionDDType:
        var ddReq = internalpb.CreateCollectionRequest{}
        if err = proto.UnmarshalText(ddOp.Body, &ddReq); err != nil {
            return err
        }
        collInfo, err := c.MetaTable.GetCollectionByName(ddReq.CollectionName, 0)
        if err != nil {
            return err
        }
        if err = c.SendDdCreateCollectionReq(ctx, &ddReq, collInfo.PhysicalChannelNames); err != nil {
            return err
        }
    // NEW
    case CreateAliasDDType:
        ...
    // NEW
    case DropAliasDDType:
        ...
    // NEW
    case AlterAliasDDType:
        ...
    ...
}

```

Compatibility, Deprecation, and Migration Plan

CollectionAlias design is not intrusive, there won't be compatibility issues.

Test Plan

- Unit tests

Rejected Alternatives

- Rejected Physical Collection Rename Design due to:
 - high implementation complexity
 - potential undesirable performance characteristics
- Rejected SetAlias API
 - SetAlias combines CreateAlias & UpdateAlias semantics, but it may have unexpected behaviors for the user. e.g. If the alias was already in the metaTable there is a risk that it may be overridden unexpectedly.

References

- ElasticSearch provides similar abstraction <https://www.elastic.co/guide/en/elasticsearch/reference/current/indices-aliases.html>