MEP 29 -- Support Role-Based Access Control

Current state: Under Discussion

Summary

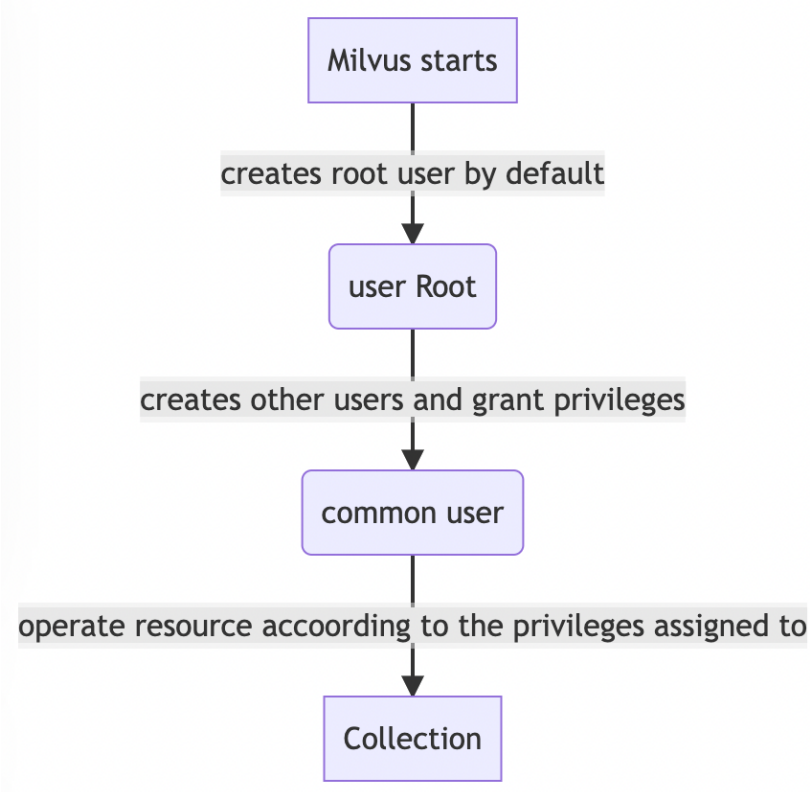
To control users' access to resources like collections in Milvus.

Motivation

There is no basic security model for resource access in Milvus currently. Users can do damage to data either intentionally or unintentionally.

This project aims to support role-based access control. Users can do their operations according to the privileges assigned to them. And administrators of the Milvus cluster can manage users and operations under control.

Design Details



Entities

User: Every user has a unique identifier and is assigned a number of privileges.

Resource Types: Resources defined in Milvus service, like collection, database, etc.

Privilege: Permissions to specific resource.

Role mapping: The mapping between users and roles.

Privilege mapping: The privileges users or roles having for a specific resource.

Additionally, when Milvus service is deployed in cloud, there will be a namespace for each Milvus cluster called tenant.

DB Schema for entities

1User

id	tenant	username	is_super	is_deleted	created_time	updated_time

Attribute is_super is true for root user, meaning the root is a super user.

The root user is created by default for each cluster and has all permissions across the cluster.

The root user does not belong to any role.

2Resource Types

id	resource	created_time

Resource types are globally unique, no need to add attribute tenant for it.

3Privilege

id	tenant	resource_type	privilege	updated_time	is_deleted	created_time

4Role

id	tenant	role_name	updated_time	is_deleted	created_time

5Role mapping

id	tenant	user_id	role_id	is_deleted	created_time

By design, a role inherited from another role is not possible here.

6Privilege grants of resource COLLECTION

id	tenant	grantor_name	principal_name	principal_type	collection_priv	collection_id	is_deleted	created_time

Grantor_name is the user who grants the privileges.
Principal_name is the target which grantor grants privileges to.
The value of principal_type are USER or ROLE.
Collection_priv is the privilege to a collection, like SELECT, INSERT, UPDATE, etc.

For some collections which have alias, it will first get the real collection of the alias. Privilege verification will be based on the real collection not the alias.
Collections stored in the table are also the real collection, not the alias.

The wildcard mode is supported for the collections in the table.

7Privilege grants of resource DATABASE (Not used since database is not supported in Milvus for now)

id	tenant	grantor_name	principal_name	principal_type	db_priv	db_id	is_deleted	created_time

Db_priv is the privilege to a database, like CREATE, DROP, etc.

KV Store Schema

1User

/prefix/credential/users/{tenant}/{username}	{"k1": "v1", "k2": "v2"}

2Resource Types

/prefix/credential/resources/{resourceType}	nil

3Privilege

/prefix/credential/privileges/{tenant}/{resourceType}/{privilege}	nil

4Role

/prefix/credential/roles/{tenant}/{rolename}	nil

5Role mapping

/prefix/credential/user-role-mapping/{tenant}/{username}/{rolename}	nil

6Privilege Grants

/prefix/credential/privilege-grants/{tenant}/{principalType}/{principalName}/{resourceType}/{resourceName}	[{"resource":"SELECT", "grantor":"Alice"}, {"resource":"UPDATE", "grantor":"Bob"}]

Resources & Privileges defined in Milvus

Users/Roles can be granted the following privileges:

Privileges	Resources
ALL	Collection
CREATE	Collection
DROP	Collection
ALTER	Collection
READ	Collection
LOAD	Collection
RELEASE	Collection
COMPACT	Collection
INSERT	Collection
DELETE	Collection

Index-related operations are included in ALTER privilege, like building, dropping index.

Default Roles

There are two default roles: admin, public.

Role admin have ALL the privilege. Role public only has READ ~~and LOAD~~ privileges.

APIs

For every API, parameter tenant is mandatory for avoiding loading too much data to memory.

1Create a role

```
func CreateRole(roleName string) bool
```

Only root user can create roles. Role name cannot be "admin" or "public".

2Grant & revoke privileges

```
func GrantPrivilege(privilege string, resourceType string, resourceName string, principalName string,
principalType string) bool

func RevokePrivilege(privilege string, resourceType string, resourceName string, principalName string,
principalType string) bool
```

Only root user can grant & revoke privileges.

3List grants for a user/role

```
func PrincipalGrantList(principalName string, principalType string, resourceType string, resourceName string) []
PrincipalGrant
```

Output structure:

PrincipalName	PrincipalType	Privilege	ResourceType	ResourceName
Alice	USER	INSERT	Collection	tbl_1

Users can only query the grants for himself. And only root user can query grants for a role.

4Show the role grants

```
func RoleGrantList(roleName string) []RoleGrant
```

Output:

Role	Privilege	ResourceType	ResourceName
role_a	INSERT	COLLECTION	tbl_1
role_a	SELECT	COLLECTION	tbl_1
role_a	CREATE	DATABASE	db_1
role_a	DROP	DATABASE	db_1

The API may query multiple tables depending on how many resource types milvus supporting.

Only root user can use the api.

5Manipulate role membership, includes adding/removing users to a role

```
func AddUserToRole(userName, roleName string) bool

func RemoveUserFromRole(userName, roleName string) bool
```

Only root user can manipulate role membership.

6Drop a role

```
func DropRole(roleName string) bool
```

A role cannot be dropped if it has privileges. Use REVOKE API to remove privileges.

Only root user can drop roles.

7List roles

```
func RoleList() []Role
```

Output:

RoleName
admin
role_a

Only root user can use the api.

8List role memberships

```
func RoleMembershipList(roleName string) []RoleMembership
```

Output:

RoleName	UserName
admin	root

Only root user can use the api.

9Show users

```
func UserList() []User
```

Output:

UserName	Roles
root	[admin, role_a]

Only root user can use the api.

10List roles of a user

```
func rolesOfUser(username string) []string
```

11List all types of resources

```
func ResourceList() []Resource
```

Output:

Resource
COLLECTION
DATABASE

12List all privileges

```
func PrivilegeListOfResource(resourceType string) []Privilege
```

Output:

ResourceType	Privilege
COLLECTION	INSERT
COLLECTION	SELECT

13Delete User

The root user cannot be deleted.

Other Notices

1. There will be initialization program for presetting users, resource types, privileges. Before the Milvus go to service, they are inserted into the meta table.
2. The root user is the only user that has privileges to create/drop/grant/revoke users and privileges.
3. In MEP-27, basic auth is taking effect if there are any existing users. Since root user is created by default once Milvus service starts, it will introduce a toggle to indicate whether the authentication is turned on.
4. Using Casbin for role-based privileges check.

Test Plan

Testing all the APIs listed above.