Access Management

Abstract

Make centralized access control a straightforward endeavor with Privacera.

Table of Contents

Get started with Access Management

Privacera Access Management works with and extends Apache Ranger to provide data access governance with centralized management of authorization policies and auditing.

Privacera Access Management offers features for policy management (resource and tag policies), data access audits, data user management, and data service management. It also includes reports which provide policy view, security zones which provide means to scope views, and permissions which manage user and group right augmentation.

Using Privacera Access Management, access to your data can be controlled based on:

- User roles
- Tags applied to data elements
- Resources that are connected to Access Management
- Resources that are not yet connected to Access Management

Privacera Access Management has the following features:

- Resource policies [58]: Create and manage *Resource Policies*.
- Tag policies [208]: Create and manage *Tag Policies*.
- Scheme policies: Create and manage *Scheme Policies*.
- Service Explorer [241]: Data resource services, also call connectors, viewing with drilldown through databases, schemas, and tables, annotated with policy defined data user access.
- Users, groups, and roles [13]: Data user access management, supporting creation and management of data access users, groups, and roles from LDAP, Active Directory, and Azure AD.
- **Permissions**: This page manages data access users and groups access to functional modules in the PrivaceraCloud hosted Apache Ranger module.
- Reports [258]: Built-in reports and dashboards for access governance, audit, and compliance.
- Audits [243]: Privacera supports full logging of all access, installation, and configuration events.
- Security zones [213]: Security Policies can be established and maintained based on administrative rights.

Access Management methods

Privacera Access Management works with and extends Apache Ranger to provide data access governance with centralized management of authorization policies and auditing.

There are several approaches to policy enforcement based on the data store and the type of access. All provide consolidated audit logging. And in all cases, data does not have to stream through Privacera's code, so the overhead added by any policy enforcement is kept to a minimum.

Access control using Apache Ranger plug-ins

Where available, Privacera leverages Apache Ranger-style distributed policy enforcement points for access control. Many data processing engines, such as Hive, Spark and Presto, support these plugins. Policies created and managed in the Privacera portal are distributed to and synchronized with these policy enforcement points. Access control decisions happen at the engine, inline with query execution.

For the following plug-ins, the sync interval for retrieving Apache Ranger policies applies:

- Databricks fine-grained access control (FGAC) plug-in: 3 seconds
- Amazon EMR Presto plug-in: 2 seconds
- Amazon EMR Hive plug-in: 2 seconds
- Amazon EMR Trino (previously PrestoSQL): 5 seconds

Access control using Privacera PolicySync

For data sources like RDBMS where Ranger-style access control plugins are not available, access control policies are enforced via policy synchronization. Privacera's PolicySync component translates the configured access control policies into the data source's native access control framework, for example by sending a relational database GRANT/REVOKE statements, generating views where needed for additional layers of access control like data masking and row filtering, and so on. When there are policy changes in Privacera, new or changed objects in the data source, or changes to users, groups and roles, updates are pushed to the data source to keep it aligned with the latest policies.

PolicySync syncs Apache Ranger access policies at 3 second intervals by default, and this interval is configurable per PolicySync connector. In addition to the sync interval, PolicySync reconciles any access policy changes with the data source, and this requires additional time that varies with the complexity of the reconciliation required, such as adding and removing grants.

Access control using the Privacera Data Access Server

For data in object stores like S3 or ADLS, access requests flow through Privacera's Data Access Server for policy enforcement. The Data Access Server integration method redirects data access requests to a Privacera 'authentication broker' inserted into the control and data flow. For requests that are allowed based on authentication and other policy checks, the authentication broker generates a signed URL that the requester can use to fetch the requested data directly from the object store. All access attempts are audited.

Data Access Server syncs Apache Ranger access policies at 5 second intervals.

For details on configuring Data Access Server, see Integrate AWS with Privacera Platform using the Data Access Server.

Users, groups, and roles

The **Access Management > Users/Groups/Roles** page is for managing data access users, groups, and roles.

Identities allowed or denied access are known as *data access* users. *Users* can be part of a *Group*, or be assigned a data access *Role*. Native Privacera individual data access users, along with management of group and roles are defined internally in Privacera and are managed in **Users/Groups/Roles.** Alternatively, data access users and groups can be imported for one-way synchronization with an external directory service or identity service based on LDAP or SCIM. For more information, see Add LDAP UserSync connector [19] or UserSync [19].

Data access users can be allowed or denied use of data stored in connected applications, as opposed to portal users, who are users that can log into your Privacera account.

For more information about users, groups, and roles, see:

- Data access users [13]
- Groups [14]
- Roles [15]

Data access users

Data access users are identified in the creation and definition of Resource Policies. Users may be included or excluded specifically or in groups.

- **User Source** value reflects the method of their creation or import (source).
  - Internal users - created within your Access Management account. Administrative users are Users: 'admin', 'rangerusersync', 'keyadmin', 'rangertagsync', and '{OWNER}' are created by the system.
  - External users:
    - A data access user with the same username as the first 'Administrator'/ Portal user;
    - A 'service' user for each data resource service (e.g. 'hive', 's3', ...);
    - Users imported User Sync with an LDAP or Active Directory.
- **Visibility** indicates if a user is listed when creating or editing a Policy in Access Management: Resource Policies. If a user is Visible, they will be found and selectable under "Select User" column. If a user is Hidden, they will not be selectable. This is useful when your account has been synchronized with a user directory with a large number of users. Visibility may be set by selecting a user object row (on the left side of the table, and using the 'Visibility' action (between +Add and Delete).
- **User Role** here is one of ('User', 'Administrator', or 'Auditor'). This user Role is different from the *custom* Roles defined in the User Management: Roles tab.

Use the Search control to limit displayed objects those matching a specific value. First select a column name, then a value. The table will be filtered to show only those objects that match the value. Users objects may be added, edited, or deleted.

Add data access users

1. From the home page, click **Access Management** > **Users/Groups/Roles**.
2. Select the **Users** tab and click **+Add**. The Add User pop-up displays.
3. Enter the user details.
4. Click **Save**.

Edit users

1. From the home page, click **Access Management** > **Users/Groups/Roles**.
2. Under the **Users** tab, select the **User** and click the pen icon in the **Actions** column.
3. Edit User dialog displays three tabs:
   - Basic Information
   - Change Password
   - Attributes
4. In the **Basic Information** tab, you can modify the user details.
5. In the **Change Password** tab, you can set new password.

NOTE
For external users, you can only edit the user role and password.

6. In the **Attributes** tab, you can add new attributes, delete, or modify existing attributes. For more information about attributes, see User and group attributes [15].
7. Click **Save**.

Add Discovery user for encryption service

To use encryption in the Compliance Workflow policies of the Discovery service, you need to add `privacera_service_discovery` user in the **Users/Groups/Roles** of **Access Management**.

1. From the home page, click **Settings** > **Users Management**.
2. In the **Portal Users** tab, on the **User Management** page, click the edit button next to the **privacera_service_discovery** user.
3. On the **Edit User** page, click **Save**.
4. After saving, verify if the `privacera_service_discovery` has been added. Go to **Access Management** > **Users/Groups/Roles** > **USERS** tab.
5. Add the user in Scheme Policies. See Create scheme policies.
6. Add the user in Ranger KMS. See Provide user access to Ranger KMS.

Groups

Groups are collections of associated users. Users can be members of more than one group. Similar to user objects, groups are used in definition of resource policies. Groups can be included or excluded specifically or in association with other groups for allowed or denied access.

Use groups to manage multiple users with similar data access needs. A user can belong to more than one group.

All functions for users are also available for groups: add/delete, hide/show, and search.

Add groups

1. From the home page, click **Access Management** > **Users/Groups/Roles**.
2. Select the **Groups** tab and click **+Add**. The Add User pop-up displays.
3. Enter the group details.
4. Click **Save**.

Edit groups

To edit the user, use the following steps:

1. From the home page, click **Access Management** > **Users/Groups/Roles**.
2. Select the **Groups** tab.

3. Select the group and click the pen icon in **Actions** column.
   Edit Group dialog displays two tabs:
   - Basic Information
   - Attributes
4. In the **Basic Information** tab, you can edit only a description.
5. In the **Attributes** tab, you can add new attributes, delete, or modify existing attributes. For more information about attributes, see User and group attributes [15].
6. Click **Save**.

Roles

With the *Roles* tab, you can create custom roles used for use when you define data access policies. Custom roles are distinct from data access imported roles and user roles.

Add roles

1. From the home page, click **Access Management** > **Users/Groups/Roles**.
2. Select the **Roles** tab and click **+Add**.
3. Enter the role details and click **Save**.

Edit roles

1. From the home page, click **Access Management** > **Users/Groups/Roles**.
2. Select the **Roles** tab.
3. Select the role and click the pen icon in **Actions** column.
4. Click **Save**.

User and group attributes

Consider the following points when editing User or Group attributes:

- Only Admin users have access to change the user attributes. Other users are unable to view or edit user attributes.
- These modifications are limited to the Ranger DB and have no impact on the source.
- Only the values can be changed. These values are considered as a single string (multiple comma-separate values cannot be added).
- Internal UserSync attributes such as `full_name`, `service_id`, and `sync_source` cannot be changed or removed. If these internal UserSync attributes are added manually through the UI for an internal user, no further modification or deletion will be permitted.
- When Ranger UserSync is restarted, the attributes from the source are overridden, but the custom attributes added from the UI are retained.
- If a user exists in more than one location, such as LDAP and Azure, If you sync that user from both sources, the attributes will be merged, and if there are any common attributes, only the attribute value from the most recent source will be retained.
- If an attribute is deleted from the source or UserSync, it will still be visible in the UI. If it is no longer required, you can delete it manually.

Portal user management

User Management creates and manages portal users. Portal users are identified by username, password, and role and allowed to access your Privacera account.

Portal users are assigned a Role. Each Role establishes a set of permissions.

Role	Description and Permissions
ROLE_ACCOUNT_ADMIN	General administrator for Account. All permissions, including Account User Management.
ROLE_POLICY_ADMIN	Administrator for establishing policy.

Role	Description and Permissions
ROLE_USER	Can access data as established by Policies.
ROLE_POLICY_AUDITOR	Can review Audit information, but can not review data directly.
ROLE_DISCOVERY_ALL	All permissions to Discovery module.
ROLE_DISCOVERY_STEWARDS	All permissions to Discovery module except Delete functionality.
ROLE_DISCOVERY_GOVERNANCE	Read-only permission to Discovery module.
ROLE_DISCOVERY_READ	Read-only permission to Discovery module.
ROLE_DISCOVERY_READ_RESTRICTED	Read-only permission to Discovery module along with hiding sample values of classifications.

Portal users are not the same as data access users who are consumers of information stored in the data repositories.

Data access users are managed in Users, groups, and roles [13].

Your account is created with a single ROLE_ACCOUNT_ADMIN role user with registered user's username and password, referred to as the Account Admin. The Account Admin (and any User with the role ROLE_ACCOUNT_ADMIN) can create additional portal users.

Add Privacera portal users

1. On the Privacera home page, expand the **Settings** menu and click on **User Management** from left menu.
2. Click **+Add**.
3. In the **Add User** dialog, enter the following details:
   - First Name (Mandatory)
   - Last Name
   - Email Id
   - User Name (Mandatory)
   - Select Role (Mandatory)
   - New Password
   - Confirm Password

   NOTE
   Email ID of a user must be unique. No two users can share the same email ID, because the email ID of the second user will appear blank.
4. Click **Save**.

Edit or delete Privacera portal users

1. On the Privacera home page, expand the **Settings** menu and click on **User Management** from left menu.
2. Click **Edit** (pencil icon) for the user.
3. Edit the user details.

   NOTE
   You are not allowed to change the Username once it is created. Hence, Username and editable.
4. Click **Save**.

5. To delete a user, click the **Delete** icon next to the user name.

Edit Privacera portal user profiles

1. On the Privacera home page, click on **Username** and then click on **Profile** on top-right.
2. Edit the profile properties. Profile pop-up displays.
3. Change the password.
   a. Click **Edit** next to the **Old Password**.
   b. Enter the old password.
   c. Enter the new password and confirm it.

Entitlement on Privacera Platform

View policies associated with a user

To see a list of policies associated with a user, follow these steps:

1. On the **Users** tab, select a user to see information about the policies for which the user has permissions.
2. On the **Resources** tab, select a resource to see information about the policies that govern access to that resource.

View user entitlement

On the **Entitlement** page, under the **Users** tab, select the User (e.g. padmin) from the drop-down. This drop-down auto-populates with a list of Access Management users.

Based on the selected user, the Entitlement page displays the relationship between user and policy in a tabular format.

The following are the columns under the Users tab:

- Details: This column allows you to expand and view the policy details with the following sub-columns:
  - Policy ID: This indicates the policy's unique id. On clicking this id, you can drill-down the policy and will be navigated to the edit policy page. You can edit the policy and save it.
  - Policy Name: This indicates the policy name.
  - Policy Labels: This indicates the policy labels.
  - Roles: This indicates the role name (if any).
  - Groups: This indicates the group name (if any).
  - Users: This indicates the user name (if any).
  - Action: This sub-column contains 3 below action items:
    - View: Using this option, you are allowed to view the policy in read-only format.
    - Edit: Using this option, you are allowed to edit the policy and save it.
    - Delete: Using this option, you are allowed to delete the policy.
- Service: This indicates the name of service such as privacera_adls or privacera_hive.
- Resource: This indicates the resource path/name which is associated with the selected user.
- Permissions: This indicates the list of permissions associated with the selected user such as read, write, meta read, delete, etc.
- Policy Count: This indicated the count of direct and indirect policies based on the selected user.

What are direct and indirect relationships?

Direct relationship denotes that the user is attached directly to the policy or resource. Direct relationship legend displays in green color along with the count of Group, Roles, and Policies. Consider the below example:

- Suppose there is a user (Mark), and a role (Project_Alpha) where the user is defined as Mark.

- So now, on the entitlement page if you select Mark as a user then under the Role drop-down Project_Alpha will be listed because it is directly mapped with a user called Mark.

Indirect relationship denotes that the user is mapped indirectly to the policy or resource. Indirect relationship legend displays in yellow color along with the count of Group, Roles, and Policies. Consider the below example:

- Suppose there is a user (Mark), and a role (Project_Beta) that contains 'Role1' as Role.
- So now, on the entitlement page if you select Mark as a user then under the Role drop-down Project_Beta will be listed because 'Role1' is indirectly mapped with a user called Mark. Because under Role1, Mark is defined as a user.

Search options:

- User: This option auto-populates with the list of defined users under Access Management module.
- Group: This option auto-populates with the list of defined groups under Access Management module.
- Role: This option auto-populates with the list of defined roles under Access Management module. This list contains direct or indirect roles associated based on the selected user.
- Zone: This option auto-populates with the list of defined zones under Access Management module.
- Resource: This option allows you to filter the entitlement records by using the resource name. For example: container-1
- Service: This option auto-populates with the list of services that are present under Access Management module.

Permissions

You can assign Access Management permissions to users and groups.

Permission types

- **Admin** has permissions to all items.
- **Auditor** can view all items except those on the '**Permissions**' menu. Auditors also view policies, users, and reports.
- **User** is the default permission type, unless they have been granted Admin or Auditor permissions. A user may be given Admin permission for a Security Zone, which allows them to create data access policies for their security zone.

Set permissions

1. From the home page, click **Access Management > Permissions**.
2. On the Module Permissions page, click the edit icon in the Action column.
3. Add or delete the user or group using the appropriate drop-down.

UserSync

UserSync synchronizes user-related data between external systems and Privacera. The following are the general types of UserSync:

- Synchronization by pulling user data from external systems into Privacera.
- Synchronization by pushing user data from Privacera to external systems.

For pull-based user provisioning, UserSync works with the Lightweight Directory Access Protocol (LDAP) , LDAP-SSL, and System for Cross-domain Identity Management (SCIM) protocols and with applications built on those prototocols, such as Active Directory (AD), Azure Active Directory (AAD), and Okta. UserSync pulls an initial set of defined identities from these systems and keeps the set of identities updated with refresh queries, approximately once an hour.

Add UserSync connectors

The documentation links below include instructions for adding various Privacera UserSync connectors.

Add LDAP UserSync connector

You can use UserSync to connect to LDAP for the purpose of connecting, pulling, or serving as data access users.

Add LDAP UserSync connector on Privacera Platform

To add an LDAP UserSync connector on Privacera Platform, follow these steps:

1. Enable Privacera UserSync:

   ```
   cd ~/privacera/privacera-manager
   cp config/sample-vars/vars.privacera-usersync.yml config/custom-vars/
   ```
2. Enable the LDAP connector:

   ```
   cd ~/privacera/privacera-manager
   cp config/sample-vars/vars.privacera-usersync.ldap.yml config/custom-vars/
   vi config/custom-vars/vars.privacera-usersync.ldap.yml
   ```
3. Edit the following properties:
   - `LDAP_CONNECTOR`: The name of the this connector
   - `LDAP_ENABLED`: The enabled status of the connector (true/false)
   - `LDAP_URL`: The service URL
   - `LDAP_BIND_DN`: The bind DN of service
   - `LDAP_BIND_PASSWORD`: The bind password
   - `LDAP_SEARCH_INCREMENTAL_ENABLED`: Enables incremental search (true/false)
   - `LDAP_SEARCH_BASE`: The search base for query
   - `LDAP_SEARCH_USER_BASE`: The search base for querying users
   - `LDAP_SEARCH_USER_FILTER`: The user search filter
   - `LDAP_SEARCH_USER_GROUPONLY`: Syncs only users that are members of groups (true/false)
   - `LDAP_SEARCH_GROUP_BASE`: The search base for querying groups
   - `LDAP_SEARCH_GROUP_FILTER`: The group search filter

   For a full list of properties, see UserSync LDAP connector properties [28].
4. Run the following command:

   ```
   cd ~/privacera/privacera-manager
   ./privacera-manager.sh update
   ```

Add LDAP UserSync connector on PrivaceraCloud

To add an LDAP UserSync connector on Privacera Cloud, follow these steps:

NOTE

Configure Connector - Detect deleted users, groups and cycles by selecting the following fields under **ADVANCED**:

- **Search Deleted Group**
- **Search Deleted User**
- **Search Deleted Cycles** - integer, the default value is 6. Min value is 0 and max value is 100.

1. From the navigation menu, select **Settings** > **Datasource**.
2. Choose a data source, click the dots icon, and select **Add Application**.
3. From the **Application List** section, select **USERSYNC**.
4. From the **Service Type** dropdown, select **LDAP**.
5. In the **Connector Name** field, enter a name for the connector.
6. In the **BASIC** tab, enter the values in the respective fields.
7. From the **Authentication Type** dropdown, select **Simple**.
8. Enable paging for UserSync using LDAP:
   a. Select **Incremental Search**.
   b. In the **Add Custom Properties** field, set the following properties:

      ```
      usersync.connector.results.paged.enabled=true
      usersync.connector.results.paged.size=<Results_Per_Page>
      ```
   c. Click **Next**.
9. Complete each step and advance through the pages of the configuration wizard.
10. Complete all **BASIC** values, then review and update **ADVANCED** values as required.
11. Click **FINISH**.

NOTE

When you update the UserSync configuration, you should restart it. This is to ensure that your updated configuration works properly.

Add SCIM UserSync connector

You can use UserSync to connect to SCIM for the purpose of connecting, pulling, or serving as data access users.

Add SCIM UserSync connector on Privacera Platform

To add an SCIM UserSync connector on Platform, follow these steps:

1. Enable Privacera UserSync:

   ```
   cd ~/privacera/privacera-manager
   cp config/sample-vars/vars.privacera-usersync.yml config/custom-vars/
   ```

2. Enable the SCIM connector:

```
cd ~/privacera/privacera-manager
cp config/sample-vars/vars.privacera-usersync.scim.yml config/custom-vars/
vi config/custom-vars/vars.privacera-usersync.scim.yml
```

3. Edit the following properties:
   - `SCIM_CONNECTOR`: The name of this connector
   - `SCIM_ENABLED`: The enabled status of the connector (true/false)
   - `SCIM_URL`: The SCIM endpoint URL
   - `SCIM_AUTH_TYPE`:
   - `ADMIN_USER_BEARER_TOKEN`:
   - `SCIM_AUTH_USERNAME`
   - `SCIM_AUTH_PASSWORD`
   - `SCIM_SEARCH_USER_GROUPONLY`: Syncs only users that are members of groups (true/false)

   For a full list of properties, see SCIM UserSync connector properties [34].
4. Run the following command:

```
cd ~/privacera/privacera-manager
./privacera-manager.sh update
```

Add SCIM UserSync connector on PrivaceraCloud

Prerequisite: Pull data access users and groups from the generic SCIM 2.0 compliant server.

To add an SCIM UserSync connector on Cloud, follow these steps:

NOTE

Configure Connector - Detect deleted users and groups by selecting the following fields under **ADVANCED**:

- **Search Deleted Group**
- **Search Deleted User**

1. From the navigation menu, select **Settings** > **Datasource**.
2. Choose a data source, click the dots icon, and select **Add Application**.
3. From the **Application List** section, select **USERSYNC**.
4. From the **Service Type** dropdown, select **SCIM**.
5. In the **BASIC** tab, enter **Endpoint URL** and **Bearer Token**.
6. Click **Next**.
7. Complete each step and advance through the pages of the configuration wizard.
8. Complete all **BASIC** values, then review and update **ADVANCED** values as required.
9. Click **FINISH**.

NOTE

When you update the UserSync configuration, you should restart it. This is to ensure that your updated configuration works properly.

Add SCIM Server UserSync connector

You can use UserSync to connect to SCIM Server for the purpose of connecting, pulling, or serving as data access users.

Add SCIM Server UserSync connector on Privacera Platform

To add a SCIM Server UserSync connector on Platform, follow these steps:

1. Enable Privacera UserSync:

   ```
   cd ~/privacera/privacera-manager
   cp config/sample-vars/vars.privacera-usersync.yml config/custom-vars/
   ```
2. Enable the SCIM Server connector:

   ```
   cd ~/privacera/privacera-manager
   cp config/sample-vars/vars.privacera-usersync.scimserver.yml config/custom-vars/
   vi config/custom-vars/vars.privacera-usersync.scimserver.yml
   ```
3. Edit the following properties:
   - `SCIM_SERVER_CONNECTOR`: The name of this connector
   - `SCIM_SERVER_ENABLED`: The enabled status of the connector (true/false)
   - `SCIM_SERVER_USERNAME`: The basic auth username
   - `SCIM_SERVER_PASSWORD`: The basic auth password
   - `SCIM_SERVER_BEARER_TOKEN`: The bearer token for auth to SCIM API

   For a full list of properties, see SCIM Server UserSync connector properties [35].
4. Run the following command:

   ```
   cd ~/privacera/privacera-manager
   ./privacera-manager.sh update
   ```

Add SCIM Server UserSync connector on PrivaceraCloud

Prerequisite: Configure to allow data access users and groups to be provided (pushed) to your PrivaceraCloud account from a SCIM 2.0 client, including push integration with an Okta Identity Provider. See SCIM Server User-Provisioning on PrivaceraCloud [47] for detailed setup instructions.

To add a SCIM Server UserSync connector on Cloud, follow these steps:

1. From the navigation menu, select **Settings** > **Datasource**.
2. Choose a data source, click the dots icon, and select **Add Application**.
3. From the **Application List** section, select **USERSYNC**.
4. From the **Service Type** dropdown, select **SCIM-Server (System for Cross Identity Management - Server Endpoint)**.
5. In the **Connector Name** field, enter a name for the connector.
6. In the **BASIC** tab, enter **Endpoint URL** and **Bearer Token**.
7. Click **Next**.
8. Complete each step and advance through the pages of the configuration wizard.
9. Complete all **BASIC** values, then review and update **ADVANCED** values as required.
10. Click **FINISH**.

NOTE

When you update the UserSync configuration, you should restart it. This is to ensure that your updated configuration works properly.

Add AAD UserSync connector

You can use UserSync to connect to Azure Active Directory (AAD) for the purpose of connecting, pulling, or serving as data access users.

Add AAD UserSync connector on Privacera Platform

To add an AAD UserSync connector on Platform, follow these steps:

1. Enable Privacera UserSync:

   ```
   cd ~/privacera/privacera-manager
   cp config/sample-vars/vars.privacera-usersync.yml config/custom-vars/
   ```
2. Enable the AAD connector:

   ```
   cd ~/privacera/privacera-manager
   cp config/sample-vars/vars.privacera-usersync.azuread.yml config/custom-vars/
   vi config/custom-vars/vars.privacera-usersync.azuread.yml
   ```
3. Edit the following properties:
   - `AZURE_AD_CONNECTOR`: The name of this connector
   - `AZURE_AD_ENABLED`: Enables the connector (true/false)
   - `AZURE_AD_TENANT_ID`: The tenant ID
   - `AZURE_AD_CLIENT_ID`: The client ID
   - `AZURE_AD_CLIENT_SECRET`: The client secret
   - `AZURE_AD_SEARCH_USER_GROUPONLY`: Syncs only the attributes of uses already synced from other services (true/false)
   - `AZURE_AD_SERVICEPRINCIPAL_ENABLED`: Enables the sync of service principals as a user (true/false)

   For a full list of properties, see AAD UserSync connector properties [25].
4. Run the following command:

   ```
   cd ~/privacera/privacera-manager
   ./privacera-manager.sh update
   ```

Add AAD UserSync connector on PrivaceraCloud

To add an AAD UserSync connector on Cloud, follow these steps:

1. From the navigation menu, select **Settings** > **Datasource**.
2. Choose a data source, click the dots icon, and select **Add Application**.
3. From the **Application List** section, select **USERSYNC**.
4. From the **Service Type** dropdown, select **AAD**.
5. In the **Connector Name** field, enter a name for the connector.
6. In the **BASIC** tab, enter the values in the respective fields.
7. From the **Authentication Type** dropdown, select **Simple**.
8. Complete each step and advance through the pages of the configuration wizard.

NOTE

Configure Connector - Detect deleted users and groups by selecting the following fields under **ADVANCED**:

- **Search Deleted Group**
- **Search Deleted User**

Configure Filters - There are the following optional fields for filtering:

- **Include Users By Domain:** Add domain names to include, default value is empty
- **Exclude Users By Domain:** Add domain names to exclude, default value is empty

9. Complete all **BASIC** values, then review and update **ADVANCED** values as required.
10. Click **FINISH**.

NOTE

When you update the UserSync configuration, you should restart it. This is to ensure that your updated configuration works properly.

Add Okta UserSync connector

You can use UserSync to connect to Okta for the purpose of connecting, pulling, or serving as data access users.

Add Okta UserSync connector on Privacera Platform

To add an Okta connector on Platform, follow these steps:

1. Enable Privacera UserSync:

```
cd ~/privacera/privacera-manager
cp config/sample-vars/vars.privacera-usersync.yml config/custom-vars/
```

2. Enable the connector:

```
cd ~/privacera/privacera-manager
cp config/sample-vars/vars.privacera-usersync.okta.yml config/custom-vars/
vi config/custom-vars/vars.privacera-usersync.okta.yml
```

3. Edit the following properties:
   - `OKTA_CONNECTOR`: The name of this connector
   - `OKTA_ENABLED`: The enabled status of the connector (true/false)
   - `OKTA_SERVICE_URL`: The Okta endpoint URL
   - `OKTA_API_TOKEN`: The API token for auth to OKTA API
   - `OKTA_SEARCH_USER_GROUPONLY`: Syncs only users that are members of groups (true/false)

   For a full list of properties, see Okta UserSync connector properties [31].
4. Run the following command:

```
cd ~/privacera/privacera-manager
./privacera-manager.sh update
```

Add Okta UserSync connector on PrivaceraCloud

Prerequisite: Pull data access users and groups from . PrivaceraCloud will use protocols in client-mode to connect to an enabled SCIM-Server. It will synchronize with the targeted server to obtain data access users and groups.

To add an connector on Cloud, follow these steps:

1. From the navigation menu, select **Settings** > **Datasource**.
2. Choose a data source, click the dots icon, and select **Add Application**.
3. From the **Application List** section, select **USERSYNC**.
4. From the **Service Type** dropdown, select **Okta**.
5. In the **Connector Name** field, enter a name for the connector.
6. In the **BASIC** tab, enter **Endpoint URL** and **Bearer Token**.
7. Click **Next**.
8. Complete all **BASIC** values, then review and update **ADVANCED** values as required.
9. Click **FINISH**.

NOTE
When you update the UserSync configuration, you should restart it. This is to ensure that your updated configuration works properly.

UserSync connector properties on Privacera Platform

With UserSync connector properties on Privacera Platform, you can define:

- Authentication to connect to your IdPs.
- The general behavior of the UserSync process.
- The user-related details you want to sync, such as users or groups to include or exclude.
- The interactions among those user-related details, such as deriving group memberships based on user records.
- Other settings.

AAD UserSync connector properties

Property	Description	Example
AAD Basic Info		
`AZURE_AD_CONNECTOR`	Name of the connector.	`AAD1`
`AZURE_AD_ENABLED`	Enabled status of connector. (true/false)	`true`
`AZURE_AD_SERVICE_TYPE`	Service Type	
`AZURE_AD_DATASOURCE_NAME`	Name of the datasource.	
`AZURE_AD_ATTRIBUTE_ONLY`	Sync only the attributes of users already synced from other services.	`false`
`AZURE_AD_SYNC_INTERVAL`	Frequency of usersync pulls and audit records in seconds. Default value is 3600, minimum value is 300.	`3600`
B) Azure AAD Info: (Get the following information from Azure Portal)		
`AZURE_AD_TENANT_ID`	Azure Active Directory Id (Tenant ID)	`1a2b3c4d-azyd-4755-9638-e12xa34p56le`

Property	Description	Example
AZURE_AD_CLIENT_ID	Azure Active Directory application client ID which will be used for accessing Microsoft Graph API.	11111111-1111-1111-1111-111111111111
AZURE_AD_CLIENT_SECRET	Azure Active Directory application client secret which will be used for accessing Microsoft Graph API.	
AZURE_AD_USERNAME	Azure Account username which will be used for getting access token to be used on behalf of Azure AD application.	
AZURE_AD_PASSWORD	Azure Account password which will be used for getting access token to be used on behalf of Azure AD application.	
C) AAD Manage/Ignore List of Users/Groups		
AZURE_AD_MANAGER_USER_LIST	List of users to manage from sync results. If this list is defined, all users not on this list will be ignored.	
AZURE_AD_IGNORE_USER_LIST	List of users to ignore from sync results.	
AZURE_AD_MANAGE_GROUP_LIST	List of groups to manage from sync results. If this list is defined, all groups not on this list will be ignored.	
AZURE_AD_IGNORE_GROUP_LIST	List of groups to ignore from sync results.	
D) AAD Search		
AZURE_AD_SEARCH_SCOPE	Azure AD Application Access Scope	
AZURE_AD_SEARCH_USER_GROUPONLY	Boolean to only load users in groups.	false
AZURE_AD_SEARCH_INCREMENTAL_ENABLED	Enable incremental search. Syncing only changes since last search.	false
AZURE_AD_SEARCH_DETECT_DELETED_USERS_GROUPS	Enables both user and group deleted searches. Default is false.	false
AZURE_AD_SEARCH_DETECT_DELETED_USERS	Override setting for user deleted search. Default value is AZURE_AD_SEARCH_DETECT_DELETED_USERS_GROUPS.	AZURE_AD_SEARCH_DETECT_DELETED_USERS_GROUPS
AZURE_AD_SEARCH_DETECT_DELETED_GROUPS	Override setting for group deleted search. Default value is AZURE_AD_SEARCH_DETECT_DELETED_USERS_GROUPS.	AZURE_AD_SEARCH_DETECT_DELETED_USERS_GROUPS
AZURE_AD_SEARCH_CYCLES_BETWEEN_DELETED_DETECTION	Number of cycles between attempts to detect deleted groups. Only used when deleted users and groups detection is enabled in the AAD connector (see above properties). Default value is 6.	6
E) Azure Service Principal		

NOTE

If **Sync Service Principals as Users** is enabled, AAD does **not** require that displayName of a Service Principal be a unique value. In this case a different attribute (such as appId) should be used as the Service Principal Username.

AZURE_AD_SERVICEPRINCIPAL_ENABLED	Sync Azure service principal to ranger user entity.	false
AZURE_AD_SERVICEPRINCIPAL_USERNAME	Properties to specify from which key to get values of username in case service principal is mapped to Ranger user entity.	displayName
F) AAD User/Group Attributes		

Property	Description	Example
AZURE_AD_ATTRIBUTE_USERNAME	Attribute of a user's name (default: userPrincipalName)	
AZURE_AD_ATTRIBUTE_FIRSTNAME	Attribute of a user's first name (default: givenName)	
AZURE_AD_ATTRIBUTE_LASTNAME	Attribute of a user's last name (default: surname)	
AZURE_AD_ATTRIBUTE_EMAIL	Attribute from user entry that would be treated as email address.	
AZURE_AD_ATTRIBUTE_GROUPNAME	Attribute from group entry that would be treated as group name.	
AZURE_AD_SERVICEPRINCIPAL_USERNAME	Attribute of service principal name.	
G) Username/Group name Attribute Modification		
AZURE_AD_ATTRIBUTE_USERNAME_VALUE_EXTRACTFROMEMAIL	Extract username from an email address. (e.g. username@domain.com -> username) Default is false.	false
AZURE_AD_ATTRIBUTE_USERNAME_VALUE_PREFIX	Prefix to prepend to the username. Default is blank.	
AZURE_AD_ATTRIBUTE_USERNAME_VALUE_POSTFIX	Postfix to append pend to the username. Default is blank.	
AZURE_AD_ATTRIBUTE_USERNAME_VALUE_TOLOWER	Convert the username to lowercase. Default is false.	false
AZURE_AD_ATTRIBUTE_USERNAME_VALUE_TOUPPER	Convert the username to uppercase. Default is false.	false
AZURE_AD_ATTRIBUTE_USERNAME_VALUE_REGEX	Attribute to replace username to matching regex. Default is blank.	
AZURE_AD_ATTRIBUTE_GROUPNAME_VALUE_EXTRACTFROMEMAIL	Extract the group name from an email address. Default is false.	false
AZURE_AD_ATTRIBUTE_GROUPNAME_VALUE_PREFIX	Prefix to prepend to the group's name. Default is blank.	
AZURE_AD_ATTRIBUTE_GROUPNAME_VALUE_POSTFIX	Postfix to append pend to the group's name. Default is blank.	
AZURE_AD_ATTRIBUTE_GROUPNAME_VALUE_TOLOWER	Convert the name to group's name to lower case. Default is false.	false
AZURE_AD_ATTRIBUTE_GROUPNAME_VALUE_TOUPPER	Convert the group's name to uppercase. Default is false.	false
AZURE_AD_ATTRIBUTE_GROUPNAME_VALUE_REGEX	Attribute to replace the group's name to matching regex. Default is blank.	
H) Group Attribute Configuration		
AZURE_AD_GROUP_ATTRIBUTE_LIST	The list of attribute keys to get from synced groups.	
AZURE_AD_GROUP_ATTRIBUTE_VALUE_PREFIX	Append prefix to values of group attributes such as group name.	
AZURE_AD_GROUP_ATTRIBUTE_KEY_PREFIX	Append prefix to key of group attributes such as group name.	
I) Filter Properties		
AZURE_AD_FILTER_USER_LIST	Filter the AAD user list, supported for non-incremental search. When incremental search is enabled delta search does not support filter properties.	abc.def@privacera.com
AZURE_AD_FILTER_SERVICEPRINCIPAL_LIST	Filter the AAD service principal list, supported for non-incremental search. When incremental search is enabled delta search does not support filter properties.	abc-testapp
AZURE_AD_FILTER_GROUP_LIST	Filter the AAD group list, supported for non-incremental search. When incremental search is enabled delta search does not support filter properties.	PRIVACERA-AB-GROUP-00
J) Domain Properties		

Property	Description	Example
AZURE_AD_MANAGE_DOMAIN_LIST	Only users in manage domain list will be synced.	Privacera.US
AZURE_AD_IGNORE_DOMAIN_LIST	Users in ignore domain list will not be synced.	Privacera.US
AZURE_AD_DOMAIN_ATTRIBUTE	Specify the attribute from which you want to compare user domain, email or username are supported. Default is email.	username

UserSync LDAP connector properties

Property	Description	Example
A) LDAP Connector Info		
LDAP_CONNECTOR	Name of the connector.	ad
LDAP_ENABLED	Enabled status of connector: true or false	true
LDAP_SERVICE_TYPE	Set a service type: ldap or ad	ad
LDAP_DATASOURCE_NAME	Name of the datasource: ldap or ad	ad
LDAP_URL	URL of source LDAP.	ldap://example.us:389
LDAP_BIND_DN	Property is used to connect to LDAP and then query for users and groups.	CN=Example User,OU=sales,DC=ad,DC=sales,DC=us
LDAP_BIND_PASSWORD	LDAP bind password for the bind DN specified above.	
LDAP_AUTH_TYPE	Authentication type, the default is simple	simple
LDAP_REFERRAL	Set the LDAP context referral: ignore or follow. Default value is follow.	follow
LDAP_SYNC_INTERVAL	Frequency of UserSync pulls and audit records in seconds. Default value is 3600, minimum value is 300.	3600
B) Enable SSL for LDAP Server		
PRIVACERA_USERSYNC_SYNC_LDAP_SSL_ENABLED	Set this property to enable/disable SSL for Privacera Usersync.	true
PRIVACERA_USERSYNC_SYNC_LDAP_SSL_PM_GEN_TS	Set this property if you want Privacera Manager to generate a truststore for your SSL-enabled LDAP server.	true
PRIVACERA_USERSYNC_AUTH_SSL_ENABLED	Set this property if the other Privacera services are not SSL enabled and you are using SSL-enabled LDAP server.	true
C) LDAP Search		

NOTE

Support Chain SSL - Preview Functionality

Previously Privacera services were only using one SSL certificate of LDAP server even if a chain of certificates was available. Now as a Preview functionality, all the certificates which are available in the chain certificate are imported it into the truststore. This is added for Privacera usersync, Ranger usersync and portal SSL certificates.

Property	Description	Example
LDAP_SEARCH_GROUP_FIRST	Property to enable to search for groups first, before searching for users.	true
LDAP_SEARCH_BASE	Search base for users and groups.	DC=ad,DC=sales,DC=us
LDAP_SEARCH_USER_BASE	Search base for users.	ou=example,dc=ad,dc=sales,dc=us
LDAP_SEARCH_USER_SCOPE	Set the value for search scope for the users: base, one or sub. Default value is sub.	sub
LDAP_SEARCH_USER_FILTER	Optional additional filter constraining the users selected for syncing.	
LDAP_SEARCH_USER_GROUPONLY	Boolean to only load users in groups.	false
LDAP_ATTRIBUTE_ONLY	Sync only the attributes of users already synced from other services.	false
LDAP_SEARCH_INCREMENTAL_ENABLED	Enable incremental search. Syncing changes only since last search.	false
LDAP_PAGED_RESULTS_ENABLED	Enable paged results control for LDAP Searches. Default is true.	true
LDAP_PAGED_CONTROL_CRITICAL	Set paged results control criticality to CRITICAL. Default is true.	true
LDAP_SEARCH_GROUP_BASE	Search base for groups.	ou=example,dc=ad,dc=sales,dc=us
LDAP_SEARCH_GROUP_SCOPE	Set the value for search scope for the groups: base, one or sub. Default value is sub.	sub
LDAP_SEARCH_GROUP_FILTER	Optional additional filter constraining the groups selected for syncing.	
LDAP_SEARCH_CYCLES_BETWEEN_DELETED_DETECTION	Numeric number of cycles between deleted searches. Default value is 6.	6
LDAP_SEARCH_DETECT_DELETED_USERS_GROUPS	Enables both user and group deleted searches. Default is false.	false
LDAP_SEARCH_DETECT_DELETED_USERS	Override setting for user deleted search. Default value is LDAP_SEARCH_DETECT_DELETED_USERS_GROUPS.	LDAP_SEARCH_DETECT_DELETED_USERS_GROUPS
LDAP_SEARCH_DETECT_DELETED_GROUPS	Override setting for group deleted search. Default value is LDAP_SEARCH_DETECT_DELETED_USERS_GROUPS.	LDAP_SEARCH_DETECT_DELETED_USERS_GROUPS
LDAP_SEARCH_READ_TIMEOUT_MS	Set LDAP search read timeout in milliseconds. Default value is 3600000.	3600000
D) LDAP Manage/Ignore List of Users/Groups		
LDAP_MANAGE_USER_LIST	List of users to manage from sync results. If this list is defined, all users not on this list will be ignored.	
LDAP_IGNORE_USER_LIST	List of users to ignore from sync results.	

Property	Description	Example
LDAP_MANAGE_GROUP_LIST	List of groups to manage from sync results. If this list is defined, all groups not on this list will be ignored.	
LDAP_IGNORE_GROUP_LIST	List of groups to ignore from sync results.	
E) LDAP Object Users/Groups Class		
LDAP_OBJECT_USER_CLASS	Objectclass to identify user entries.	user
LDAP_OBJECT_GROUP_CLASS	Objectclass to identify group entries.	group
F) LDAP User/Group Attributes		
LDAP_ATTRIBUTE_USERNAME	Attribute from user entry that would be treated as user name.	SAMAccountName
LDAP_ATTRIBUTE_FIRSTNAME	Attribute of a user's first name. The default is givenName.	givenName
LDAP_ATTRIBUTE_LASTNAME	Attribute of a user's last name.	
LDAP_ATTRIBUTE_EMAIL	Attribute from user entry that would be treated as email address.	mail
LDAP_ATTRIBUTE_GROUPNAMES	List of attributes from group entry that would be treated as group name.	
LDAP_ATTRIBUTE_GROUPNAME	Attribute from group entry that would be treated as group name.	name
LDAP_ATTRIBUTE_GROUP_MEMBER	Attribute from group entry that is list of members.	member
G) Username/Group name Attribute Modification		
LDAP_ATTRIBUTE_USERNAME_VALUE_EXTRACTFROMEMAIL	Extract username from an email address. (e.g. username@domain.com -> username) Default is false.	false
LDAP_ATTRIBUTE_USERNAME_VALUE_PREFIX	Prefix to prepend to the username. Default is blank.	
LDAP_ATTRIBUTE_USERNAME_VALUE_POSTFIX	Postfix to append pend to the username. Default is blank.	
LDAP_ATTRIBUTE_USERNAME_VALUE_TOLOWER	Convert the username to lowercase. Default is false.	false
LDAP_ATTRIBUTE_USERNAME_VALUE_TOUPPER	Convert the username to uppercase. Default is false.	false
LDAP_ATTRIBUTE_USERNAME_VALUE_REGEX	Attribute to replace username to matching regex. Default is blank.	
LDAP_ATTRIBUTE_GROUPNAME_VALUE_EXTRACTFROMEMAIL	Extract the group name from an email address. Default is false.	false
LDAP_ATTRIBUTE_GROUPNAME_VALUE_PREFIX	Prefix to prepend to the group's name. Default is blank.	
LDAP_ATTRIBUTE_GROUPNAME_VALUE_POSTFIX	Postfix to append pend to the group's name. Default is blank.	
LDAP_ATTRIBUTE_GROUPNAME_VALUE_TOLOWER	Convert the name to group's name to lower case. Default is false.	false
LDAP_ATTRIBUTE_GROUPNAME_VALUE_TOUPPER	Convert the group's name to uppercase. Default is false.	false

Property	Description	Example
LDAP_ATTRIBUTE_GROUPNAME_VALUE_REGEX	Attribute to replace the group's name to matching regex. Default is blank.	
H) Group Attribute Configuration		
LDAP_GROUP_ATTRIBUTE_LIST	The list of attribute keys to get from synced groups.	
LDAP_GROUP_ATTRIBUTE_VALUE_PREFIX	Append prefix to values of group attributes such as group name.	
LDAP_GROUP_ATTRIBUTE_KEY_PREFIX	Append prefix to key of group attributes such as group name.	
LDAP_GROUP_LEVELS	Configure Privacera UserSync with AD/LDAP nested group membership.	

Okta UserSync connector properties

Property	Description	Example
A) OKTA Connector Info		
OKTA_CONNECTOR	Name of the connector.	OKTA
OKTA_ENABLED	Enabled status of connector. (true/false)	true
OKTA_SERVICETYPE	Type of service/connector.	okta
OKTA_DATASOURCE_NAME	Unique datasource name, used for identifying source of data and configuring priority list. (Optional)	
OKTA_SERVICE_URL	Connector URL	https://{myOktaDomain}.okta.com
OKTA_API_TOKEN	API token	A8b2c84d-895a-4fea-82dc-401397b8e50c
OKTA_SYNC_INTERVAL	Frequency of usersync pulls and audit records in seconds. Default value is 3600, minimum value is 300.	3600
B) OKTA Manage/Ignore List of Users/Groups		
OKTA_USER_LIST	List of users to manage from sync results. If this list is defined, all users not on this list will be ignored.	
OKTA_IGNORE_USER_LIST	List of users to ignore from sync results.	
OKTA_USER_LIST_STATUS	List of users to manage with status as equal to: STAGED, PROVISIONED,ACTIVE,RECOVERY,PASSWORD_EXPIRED,LOCKED_OUT or DEPROVISIONED. If this list is defined, all users not on this list will be ignored.	ACTIVE,STAGED

Property	Description	Example
OKTA_USER_LIST_LOGIN	List of users to manage with user login name (can contain *). If this list is defined, all users not on this list will be ignored.*	sw;mon,san
OKTA_USER_LIST_PROFILE_FIRSTNAME	List of users to manage with user first name (can contain). If this list is defined, all users not on this list will be ignored.	sw;mon,san
OKTA_USER_LIST_PROFILE_LASTNAME	List of users to manage with user last name (can contain *). If this list is defined, all users not on this list will be ignored.*	sw;mon,san
OKTA_LIST_PROFILE_EMAIL	List of users to manage with user email (can contain). If this list is defined, all users not on this list will be ignored.	sw;mon,san
OKTA_LIST_TYPE	List of groups to manage with group type. If this list is defined, all groups not on this list will be ignored.	APP_GROUP,BUILT_IN,OKTA_GROUP
OKTA_GROUP_LIST	List of groups to manage from sync results. If this list is defined, all groups not on this list will be ignored.	
OKTA_IGNORE_GROUP_LIST	List of groups to ignore from sync results.	
OKTA_GROUP_LIST_SOURCE_ID	List of groups to manage with group source id. If this list is defined, all groups not on this list will be ignored.	0oa2v0el0gP90aqjJ0g7,0oa2v0el0gP90aqjJ0g8,0oa2v0el0gP90aqjJ0g0
OKTA_GROUP_LIST_PROFILE_NAME	List of groups to manage with group name. If this list is defined, all groups not on this list will be ignored.	group1,testGroup,testGroup2
C) OKTA Search		
OKTA_SEARCH_USER_GROUP_ONLY	Boolean to only load users in groups.	false
OKTA_SEARCH_INCREMENTAL_ENABLED	Boolean to enable incremental search, syncing only changes since last search.	false
D) OKTA User/Group Attributes		
OKTA_ATTRIBUTE_USERNAME	Attribute from user entry that would be treated as user name.	login

Property	Description	Example
OKTA_ATTRIBUTE_FIRST-NAME	Attribute from user entry that would be treated as firstname.	firstName
OKTA_ATTRIBUTE_LASTNAME	Attribute from user entry that would be treated as lastname.	lastName
OKTA_ATTRIBUTE_EMAIL	Attribute from user entry that would be treated as email address.	email
OKTA_ATTRIBUTE_GROUPS	Attribute of user's group list.	groups
OKTA_ATTRIBUTE_GROUP-NAME	Attribute of a group's name.	name
OKTA_ATTRIBUTE_ONLY	Sync only the attributes of users already synced from other services. (true/false)	false
E) OKTA Username Attribute Modifications		
OKTA_ATTRIBUTE_USER-NAME_VALUE_EXTRACTFRO-MEMAIL	Extract the user's username from an email address. (e.g. username@domain.com -> username) The default is false.	false
OKTA_ATTRIBUTE_USER-NAME_VALUE_PREFIX	Prefix to prepend to username. The default is blank.	
OKTA_ATTRIBUTE_USER-NAME_VALUE_POSTFIX	Postfix to append to the username. The default is blank.	
OKTA_ATTRIBUTE_USER-NAME_VALUE_TOLOWER	Convert the user's username to lowercase. The default is false.	false
OKTA_ATTRIBUTE_USER-NAME_VALUE_TOUPPER	Convert the user's username to uppercase. The default is false.	false
OKTA_ATTRIBUTE_USER-NAME_VALUE_REGEX	Attribute to replace username to matching regex. The default is blank.	
F) OKTA Group Name Attribute Modifications		
OKTA_ATTRIBUTE_GROUP-NAME_VALUE_EXTRACTFRO-MEMAIL	Extract the group's name from an email address (e.g. groupname@domain.com -> groupname). The default is false.	false
OKTA_ATTRIBUTE_GROUP-NAME_VALUE_PREFIX	Prefix to prepend to the group's name. The default is blank.	
OKTA_ATTRIBUTE_GROUP-NAME_VALUE_POSTFIX	Postfix to append to the group's name. The default is blank.	
OKTA_ATTRIBUTE_GROUP-NAME_VALUE_TOLOWER	Convert group's name to lowercase. The default is false.	false
OKTA_ATTRIBUTE_GROUP-NAME_VALUE_TOUPPER	Convert the group's name to uppercase. The default is false.	false

Property	Description	Example
OKTA_ATTRIBUTE_GROUP-NAME_VALUE_REGEX	Attribute to replace group's name to matching regex. The default is blank.	

SCIM UserSync connector properties

Property	Description	Example
A) SCIM Connector Info		
SCIM_CONNECTOR	Name of connector.	DB1
SCIM_ENABLED	Enabled status of connector. (true/false)	true
SCIM_SERVICETYPE	Service Type	scim
SCIM_DATASOURCE_NAME	Name of the datasource.	databricks1
SCIM_URL	Connector URL	
ADMIN_USER_BEARER_TOKEN	Bearer token	
SCIM_SYNC_INTERVAL	Frequency of UserSync pulls and audit records in seconds. Default value is 3600, minimum value is 300.	3600
SCIM_SEARCH_DETECT_DE-LETED_USERS_GROUPS	Enable search of deleted users/groups.	false
B) SCIM Manage/Ignore List of Users/Groups		
SCIM_MANAGE_USER_LIST	List of users to manage from sync results. If this list is defined, all users not on this list will be ignored	
SCIM_IGNORE_USER_LIST	List of users to ignore from sync results.	
SCIM_MANAGE_GROUP_LIST	List of groups to manage from sync results. If this list is defined, all groups not on this list will be ignored.	
SCIM_IGNORE_GROUP_LIST	List of groups to ignore from sync results.	
C) SCIM User/Group Attributes		
SCIM_ATTRIBUTE_USERNAME	Attribute from user entry that would be treated as user name.	userName
SCIM_ATTRIBUTE_FIRSTNAME	Attribute from user entry that would be treated as first-name.	name.givenName
SCIM_ATTRIBUTE_LASTNAME	Attribute from user entry that would be treated as last-name.	name.familyName
SCIM_ATTRIBUTE_EMAIL	Attribute from user entry that would be treated as email address.	emails[primary-true].value
SCIM_ATTRIBUTE_ONLY	Sync only the attributes of users already synced from other services. (true/false)	false
SCIM_ATTRIBUTE_GROUPS	Attribute of user's group list.	groups
SCIM_ATTRIBUTE_GROUPNAME	Attribute from group entry that would be treated as group name.	displayName
SCIM_ATTRIBUTE_GROUP_MEMBER	Attribute from group entry that is list of members.	members
D) SCIM Server Username Attribute Modifications		
SCIM_ATTRIBUTE_USERNAME_VAL-UE_EXTRACTFROMEMAIL	Extract the user's username from an email address. (e.g. username@domain.com -> username) The default is false.	false
SCIM_ATTRIBUTE_USERNAME_VAL-UE_PREFIX	Prefix to prepend to username. The default is blank.	
SCIM_ATTRIBUTE_USERNAME_VAL-UE_POSTFIX	Postfix to append to the username. The default is blank.	
SCIM_ATTRIBUTE_USERNAME_VAL-UE_TOLOWER	Convert the user's username to lowercase. The default is false.	false
SCIM_ATTRIBUTE_USERNAME_VAL-UE_TOUPPER	Convert the user's username to uppercase. The default is false.	false
SCIM_ATTRIBUTE_USERNAME_VAL-UE_REGEX	Attribute to replace username to matching regex. The default is blank.	
E) SCIM Server Group Name Attribute Modifications		

Property	Description	Example
SCIM_ATTRIBUTE_GROUP-NAME_VALUE_EXTRACTFROMEMAIL	Extract the group's name from an email address (e.g. groupname@domain.com -> groupname). The default is false.	false
SCIM_ATTRIBUTE_GROUP-NAME_VALUE_PREFIX	Prefix to prepend to the group's name. The default is blank.	
SCIM_ATTRIBUTE_GROUP-NAME_VALUE_POSTFIX	Postfix to append to the group's name. The default is blank.	
SCIM_ATTRIBUTE_GROUP-NAME_VALUE_TOLOWER	Convert group's name to lowercase. The default is false.	false
SCIM_ATTRIBUTE_GROUP-NAME_VALUE_TOUPPER	Convert the group's name to uppercase. The default is false.	false
SCIM_ATTRIBUTE_GROUP-NAME_VALUE_REGEX	Attribute to replace group's name to matching regex. The default is blank.	
F) Group Attribute Configuration		
SCIM_GROUP_ATTRIBUTE_LIST	The list of attribute keys to get from synced groups.	
SCIM_GROUP_ATTRIBUTE_VAL-UE_PREFIX	Append prefix to values of group attributes such as group name.	
SCIM_GROUP_ATTRIB-UTE_KEY_PREFIX	Append prefix to key of group attributes such as group name.	
SCIM_APPLY_MEMBER-SHIPS_FROM_USER	Defines group membership based on the user record's group attributes. • Default: true • Allowable values: true, false.	false

SCIM Server UserSync connector properties

Property	Description	Example
A) SCIM Server Connector Info		
SCIMSERVER_CONNECTOR	Identifying name of this connector.	DB1
SCIMSERVER_ENABLED	Enabled status of connector. (true/false)	true
SCIMSERVER_SERVICETYPE	Type of service/connector.	scimserver
SCIMSERVER_DATASOURCE_NAME	Unique datasource name. Used for identifying source of data and configuring priority list. (Optional)	databricks1
SCIMSERVER_ATTRIBUTE_ONLY	Sync only the attributes of users already synced from other services. (true/false)	
SCIMSERVER_BEARER_TOKEN	Bearer token for auth to SCIM API. When set, SCIM requests with this token will be allowed access.	
SCIMSERVER_USERNAME	Basic auth username, when set SCIM requests with this username will be allowed access. (Password also required)	
SCIMSERVER_PASSWORD	Basic auth password, when set SCIM requests with this password will be allowed access. (Username also required)	
SCIMSERVER_SYNC_INTERVAL	Frequency of usersync audit records in seconds. Default value is 3600, minimum value is 300.	3600
B) SCIM Server Manage/Ignore List of Users/Groups		
SCIMSERVER_MANAGE_USER_LIST	List of users to manage from sync results. If this list is defined, all users not on this list will be ignored.	
SCIMSERVER_IGNORE_USER_LIST	List of users to ignore from sync results.	
SCIMSERVER_MANAGE_GROUP_LIST	List of groups to manage from sync results. If this list is defined, all groups not on this list will be ignored.	
SCIMSERVER_IGNORE_GROUP_LIST	List of groups to ignore from sync results.	
C) SCIM Server Attributes		
SCIMSERVER_ATTRIBUTE_USERNAME	Attribute of a user's name.	userName
SCIMSERVER_ATTRIBUTE_FIRSTNAME	Attribute of a user's first name.	name.givenName
SCIMSERVER_ATTRIBUTE_LASTNAME	Attribute of a user's last/family name.	name.familyName

Property	Description	Example
SCIMSERVER_ATTRIBUTE_EMAIL	Attribute of a user's email.	emails[primary-true].value
SCIMSERVER_ATTRIBUTE_GROUPS	Attribute of a user's group list.	groups
SCIMSERVER_ATTRIBUTE_GROUPNAME	Attribute of a group's name.	displayName
SCIMSERVER_ATTRIB-UTE_GROUP_MEMBER	Attribute from group entry that is the list of members.	members
D) SCIM Server Username Attribute Modifications		
SCIMSERVER_ATTRIBUTE_USER-NAME_VALUE_EXTRACTFROMEMAIL	Extract the user's username from an email address. (e.g. username@domain.com -> username) The default is false.	false
SCIMSERVER_ATTRIBUTE_USER-NAME_VALUE_PREFIX	Prefix to prepend to username. The default is blank.	
SCIMSERVER_ATTRIBUTE_USER-NAME_VALUE_POSTFIX	Postfix to append to the username. The default is blank.	
SCIMSERVER_ATTRIBUTE_USER-NAME_VALUE_TOLOWER	Convert the user's username to lowercase. The default is false.	false
SCIMSERVER_ATTRIBUTE_USER-NAME_VALUE_TOUPPER	Convert the user's username to uppercase. The default is false.	false
SCIMSERVER_ATTRIBUTE_USER-NAME_VALUE_REGEX	Attribute to replace username to matching regex. The default is blank.	
E) SCIM Server Group Name Attribute Modifications		
SCIMSERVER_ATTRIBUTE_GROUP-NAME_VALUE_EXTRACTFROMEMAIL	Extract the group's name from an email address (e.g. groupname@domain.com -> groupname). The default is false.	false
SCIMSERVER_ATTRIBUTE_GROUP-NAME_VALUE_PREFIX	Prefix to prepend to the group's name. The default is blank.	
SCIMSERVER_ATTRIBUTE_GROUP-NAME_VALUE_POSTFIX	Postfix to append to the group's name. The default is blank.	
SCIMSERVER_ATTRIBUTE_GROUP-NAME_VALUE_TOLOWER	Convert group's name to lowercase. The default is false.	false
SCIMSERVER_ATTRIBUTE_GROUP-NAME_VALUE_TOUPPER	Convert the group's name to uppercase. The default is false.	false
SCIMSERVER_ATTRIBUTE_GROUP-NAME_VALUE_REGEX	Attribute to replace group's name to matching regex. The default is blank.	
F) Group Attribute Configuration		
SCIMSERVER_GROUP_ATTRIB-UTE_LIST	The list of attribute keys to get from synced groups.	
SCIMSERVER_GROUP_ATTRIB-UTE_VALUE_PREFIX	Append prefix to values of group attributes such as group name.	
SCIMSERVER_GROUP_ATTRIB-UTE_KEY_PREFIX	Append prefix to key of group attributes such as group name.	
SCIM_SERVER_APPLY_MEMBER-SHIPS_FROM_USER	Defines group membership based on the user record's group attributes. • Default: false. • Allowable values: true, false.	true

UserSync connector fields on PrivaceraCloud

With UserSync connector fields on PrivaceraCloud, you can define:

- Authentication to connect to your IdPs.
- The general behavior of the UserSync process.
- The user-related details you want to sync, such as users or groups to include or exclude.
- The interactions among those user-related details, such as deriving group memberships based on user records.
- Other settings.

LDAP/AD fields for UserSync on PrivaceraCloud

These are descriptions of fields for configuring PrivaceraCloud UserSync for LDAP and Active Directory.

Field name	Description	Tab in application set-up
Name	Identifying name of this connector.	
Group Only	Sync only users that are members of groups. Allowable values: true or false	
Attribute Only	Sync only the attributes of users already synced from other services. Allowable values: true or false	
Incremental	Enable incremental search. Syncing only changes since last search. Allowable values: true or false	
Paged Results	Enable paged results control for LDAP Searches. Allowable values: true or false Default: true	
Paged Results Control Critical	Set paged results control criticality to CRITICAL. Allowable values: true or false Default: true	
Manage Lists		
Include Users	List of users to include from sync results. If this list is defined, all users not on this list are ignored.	
Exclude Users	List of users to ignore from sync results.	
Include Groups	List of groups to include from sync results. If this list is defined, all groups not on this list are ignored.	
Exclude Groups	List of groups to exclude from sync results.	
Connection		
Service URL	Service URL	
Bind DN	Bind DN of service.	
Bind Password	Bind password .	
Authentication Type	Authentication type. Default: simple	Advanced
Follow Referral	Follow LDAP Referral. Default: true Allowable values: true or false	Advanced
Search Base	Search base for query.	
	User Search	
User Search Base	Search base for querying users.	
User Search Scope	User search scope. Default: 2.	
User Search Filter	User search filter.	
User Object Class	User object class. Default: user	Advanced
Group Search		
Group Search Base	Search base for querying groups.	
Group Search Scope	Group search scope. Default: 2.	
Group Search Filter	Group search filter.	
Group Object Class	Group object class. Default: group	Advanced
Nested Group Levels	Number of levels to evaluate nested groups.	Advanced
Attributes		
Username	Attribute of a user's username. Default: sAMAccountName.	
First Name	Attribute of a user's first name. Default: givenName.	
Last Name	Attribute of a user's last name. Default: sn.	
Email	Attribute of a user's email. Default: email.	
Group Name	Attribute of a group's name. Default: sAMAccountName.	
Group Members	Attribute listing a group's members. Default: member.	
Username Attribute Modification		
Extract From Email	Extract the user's username from an email address. Example: username@domain.com extracts username. Default: false	Advanced

Field name	Description	Tab in application set-up
Prefix	Prefix to prepend to the user's username. No default.	Advanced
Postfix	Postfix to append to the user's username. No default.	Advanced
To Lowercase	Convert the user's username to lowercase. Default: false	Advanced
To Uppercase	Convert the user's username to uppercase. Default: false	Advanced
Regex	Attribute to replace user's username to matching regex No default.	Advanced
Group Name Attribute Modification		
Extract From Email	Extract the group's name from an email address. Example: groupname@domain.com extracts groupname. Default: false	Advanced
Prefix	Prefix to prepend to the group's name. No default.	Advanced
Postfix	Postfix to append to the group's name. No default.	Advanced
To Lowercase	Convert the group's name to lowercase. Default: false	Advanced
To Uppercase	Convert the group's name to uppercase. Default: false	Advanced
Regex	Replace group's name to matching regex. No default.	Advanced

SCIM Server fields for UserSync on PrivaceraCloud

These are descriptions of fields for configuring PrivaceraCloud UserSync for a SCIM server.

Field name	Description	Tab in application set-up
Name	Identifying name of this connector.	
Attribute Only	Sync only the attributes of users already synced from other services. Allowable values: true or false	
Manage Lists		
Include Users	List of users to include from sync results. If this list is defined, all users not on this list are ignored.	
Exclude Users	List of users to exclude from sync results.	
Include Groups	List of groups to include from sync results. If this list is defined, all groups not on this list are ignored.	
Exclude Groups	List of groups to exclude from sync results.	
Connection		
Endpoint URL	SCIM endpoint URL Base URL for SCIM endpoints (readOnly) /API/pus/public/scim/v2/{connectorName}	
Bearer Token	Bearer token for auth to SCIM API. When set, SCIM requests with this token are allowed access.	

Field name	Description	Tab in application set-up
Username	Basic auth username. When set, SCIM requests with this username are allowed access. Password is required.	
Password	Basic Auth Password. When set, SCIM requests with this password are allowed access. Username is required.	
Attributes		
User Name	Attribute of a user's name Default: userName	Advanced
First Name	Attribute of a user's first name Default: name.givenName	Advanced
Family Name	Attribute of a user's family name. Default: name.familyName	Advanced
User Email	Attribute of a user's email. Default: emails[primary-true].value	Advanced
User Groups	Attribute of user's group list. Default: groups	Advanced
Group Name	Attribute of a group's name. Default: displayName	Advanced
Group Members	Attribute of a group's member list. Default: members	Advanced
Username Attribute Modification		
Extract From Email	Extract the user's username from an email address. Example:. username@domain.com extracts username. Default: false	Advanced
Prefix	Prefix to prepend to the user's username. No default.	Advanced
Postfix	Postfix to append to the user's username. No default.	Advanced
To Lowercase	Convert the user's username to lowercase. Default: false	Advanced
To Uppercase	Convert the user's username to uppercase. Default: false	Advanced
Regex	Attribute to replace user's username to matching regex No default.	Advanced
Group Name Attribute Modification		
Extract From Email	Extract the group's name from an email address. Example: groupname@domain.com extracts groupname. Default: false	Advanced
Prefix	Prefix to prepend to the group's name. No default.	Advanced
Postfix	Postfix to append to the group's name. No default.	Advanced
To Lowercase	Convert the group's name to lowercase. Default: false	Advanced
To Uppercase	Convert the group's name to uppercase. Default: false	Advanced
Regex	Replace group's name to matching regex. No default.	Advanced

Okta fields for UserSync on PrivaceraCloud

These are descriptions of fields for configuring PrivaceraCloud UserSync for Okta.

Field name	Description	Tab in application set-up
Name	Identifying name of this connector.	
Group Only	Sync only users that are members of groups. Allowable values: true or false	
Attribute Only	Sync only the attributes of users already synced from other services. Allowable values: true or false	
Incremental	Enable incremental search. Syncing only changes since last search. Allowable values: true or false	
Manage Lists		
Include Users	List of users to include from sync results. If this list is defined, all users not on this list are ignored.	
Exclude Users	List of users to ignore from sync results.	
Include Users	List of users to manage with status as equal to STAGED, PROVISIONED,ACTIVE,RECOVERY,PASSWORD_EXPIRED,LOCKED_OUT,DEPROVISIONED . If this list is defined, all users not on this list are ignored. Example: eq;ACTIVE,STAGED	
Include Users	List of users to manage with user login name. Wildcard (*) is allowed. Format filterOperator;login,login2,login3 #sw for start with . If this list is defined, all users not on this list are ignored. Example: sw;mon,san	
Include Users	List of users to manage with user first name. Wildcard (*) is allowed. Format filterOperator;firstName,firstName2,firstName3 #sw for start with . If this list is defined, all users not on this list are ignored. Example: sw;mon,san	
Include Users	List of users to manage with user last name. Wildcard (*) is allowed. Format filterOperator;lastName,lastName2,lastName3 #sw for start with . If this list is defined, all users not on this list are ignored. Example: sw;mon,san	
Include Users	List of users to manage with user email. Wildcard (*) is allowed. Format filterOperator;email,email2,email3 #sw for start with If this list is defined, all users not on this list are ignored. Sample value: sw;mon,san	
Include Groups	List of groups to manage from sync results. If this list is defined, all groups not on this list are ignored.	
Exclude Groups	List of groups to exclude from sync results.	
Include Groups	List of groups to manage with group type. Format filterOperator;groupType,groupType2,groupType3 If this list is defined, all groups not on this list are ignored. Sample: e q;APP_GROUP,BUILT_IN, Okta _GROUP	
Include Groups	List of groups to manage with group names. Format filterOperator;groupName,groupName2,groupName3 If this list is defined, all groups not on this list are ignored. Example: e q;group1,testGroup,testGroup2	
Include Groups	List of groups to manage with group samAccountName. Format filterOperator;samAccountName,samAccountName2,samAccountName3 If this list is defined, all groups not on this list are ignored. Example: e q;sam1,sam2,sam3	

Field name	Description	Tab in application set-up
Include Groups	List of groups to manage with group source id. Format filterOperator;sourceId,sourceId2,sourceId3 If this list is defined, all groups not on this list are ignored. Example: eq;0oa2v0el0gP90aqjJ0g7 ,0oa2v0el0gP90aqjJ0g8 ,0oa2v0el0gP90aqjJ0g0	
Connection		
Endpoint URL	Okta endpoint URL	
Bearer Token	API token for auth to Okta API	
Attributes		
User Name	Attribute of a user's name Default: login	Advanced
First Name	Attribute of a user's first name Default: firstName	Advanced
Family Name	Attribute of a users's family name. Default: lastName	Advanced
User Email	Attribute of a user's email Default: email.	Advanced
User Groups	Attribute of user's group list. Default: groups	Advanced
Group Name	Attribute of a group's name Default: name	Advanced
Username Attribute Modification		
Extract From Email	Extract the user's username from an email address. Example: username@domain.com extracts username. Default: false	Advanced
Prefix	Prefix to prepend to the user's username. No default.	Advanced
Postfix	Postfix to append to the user's username. No default.	Advanced
To Lowercase	Convert the user's username to lowercase. Default: false	Advanced
To Uppercase	Convert the user's username to uppercase. Default: false	Advanced
Regex	Attribute to replace user's username to matching regex. Default : Blank	Advanced
Group Name Attribute Modification		
Extract From Email	Extract the group's name from an email address.Example: groupname@domain.com extracts groupname. Default: false	Advanced
Prefix	Prefix to prepend to the group's name. Default : Blank	Advanced
Postfix	Postfix to append to the group's name. No default.	Advanced
To Lowercase	Convert the group's name to lowercase. Default: false	Advanced
To Uppercase	Convert the group's name to uppercase. Default: false	Advanced
Regex	Replace group's name to matching regex. No default.	Advanced

Azure Active Directory fields for UserSync on PrivaceraCloud

These are descriptions of fields for configuringPrivaceraCloud UserSync for Azure Active Directory.

Field name	Description	Tab in application set-up
Name	Identifying name of this connector.	
Group Only	Sync only users that are members of groups. Allowable values: true or false	
Attribute Only	Sync only the attributes of users already synced from other services. Allowable values: true or false	
Incremental	Enable incremental search. Syncing only changes since last search. Allowable values: true or false	
Service Principals as Users	Enable sync of service principals as a User. Allowable values: true or false Default: false	
Manage Lists		
Include Users	List of users to include from sync results. If this list is defined, all users not on this list are ignored.	
Exclude Users	List of users to ignore from sync results.	
Include Groups	List of groups to include from sync results. If this list is defined, all groups not on this list are ignored.	
Exclude Groups	List of groups to exclude from sync results.	
Connection		
Account ID	Account ID	
Client ID	Client ID	
Client Secret	Client secret	
Attributes		
User Name	Attribute of a user's name. Default: userPrincipalName.	
First Name	Attribute of a user's first name. Default: givenName.	
Last Name	Attribute of a user's last name. Default: surname.	
User Email	Attribute of a user's email. Default: userPrincipalName.	
Group Name	Attribute of a group's name. Default: displayName	
Service Principal Name	Attribute of service principal name. Default: displayName	
Username Attribute Modification		
Extract From Email	Extract the user's username from an email address.Example: username@domain.com extracts username. Default: false	Advanced
Prefix	Prefix to prepend to the user's username. No default.	Advanced
Postfix	Postfix to append to the user's username. No default.	Advanced
To Lowercase	Convert the user's username to lowercase. Default: false	Advanced
To Uppercase	Convert the user's username to uppercase. Default: false	Advanced
Regex	Attribute to replace user's username to matching regex No default.	Advanced
Group Name Attribute Modification		
Extract From Email	Extract the group's name from an email address. Example: groupname@domain.com extracts groupname. Default: false	Advanced
Prefix	Prefix to prepend to the group's name. No default.	Advanced
Postfix	Postfix to append to the group's name. No default.	Advanced

Field name	Description	Tab in application set-up
To Lowercase	Convert the group's name to lowercase. Default: false	Advanced
To Uppercase	Convert the group's name to uppercase. Default: false	Advanced
Regex	Replace group's name to matching regex. No default.	Advanced

SCIM fields for UserSync on PrivaceraCloud

These are descriptions of fields for configuring PrivaceraCloud UserSync for SCIM.

Field name	Description	Tab in application set-up
Name	Identifying name of this connector.	
Group Only	Sync only users that are members of groups. Allowable values: true or false	Advanced
Attribute Only	Sync only the attributes of users already synced from other services. Allowable values: true or false	
	Manage Lists	
Include Users	List of users to manage from sync results. If this list is defined, all users not on this list are ignored.	
Include Users	List of users to ignore from sync results.	
Include Groups	List of groups to include from sync results. If this list is defined, all groups not on this list are ignored.	
Include Groups	List of groups to ignore from sync results.	
	Connection	
Endpoint URL	SCIM endpoint URL	
Bearer Token	Bearer token for auth to SCIM API	
	Attributes	
User Name	Attribute of a user's name Default: userName	Advanced
First Name	Attribute of a user's first name Default: name.givenName	Advanced
Family Name	Attribute of a user's family name. Default: name.familyName	Advanced
User Email	Attribute of a user's email Default: emails[primary-true].value	Advanced
User Groups	Attribute of user's group list. Default: groups	Advanced
Group Name	Attribute of a group's name. Default: displayName	Advanced
Group Members	Attribute of a group's member list. Default: members	Advanced
	Username Attribute Modification	
Extract From Email	Extract the user's username from an email address. Example: username@domain.com extracts username. Default: false	Advanced
Prefix	Prefix to prepend to the user's username. No default.	Advanced
Postfix	Postfix to append to the user's username. No default.	Advanced
To Lowercase	Convert the user's username to lowercase. Default: false	Advanced
To Uppercase	Convert the user's username to uppercase. Default: false	Advanced
Regex	Attribute to replace user's username to matching regex No default.	Advanced
	Group Name Attribute Modification	

Field name	Description	Tab in application set-up
Extract From Email	Extract the group's name from an email address. Example: groupname@domain.com extracts groupname. Default: false	Advanced
Prefix	Prefix to prepend to the group's name. No default.	Advanced
Postfix	Postfix to append to the group's name. No default.	Advanced
To Lowercase	Convert the group's name to lowercase. Default: false	Advanced
To Uppercase	Convert the group's name to uppercase. Default: false	Advanced
Regex	Replace group's name to matching regex. No default.	Advanced

UserSync system properties on Privacera Platform

UserSync property	Description	Property	Default
`PRIVACERA_USERSYNC_RANGER_URL`	Address of Ranger instance.	`ranger.url`	`http://ranger:6080`
`PRIVACERA_USERSYNC_RANGER_USERNAME`	Username of Ranger user.	`ranger.username`	`admin`
`PRIVACERA_USERSYNC_RANGER_PASSWORD`	Password of Ranger user.	`ranger.password`	`welcome1`

`PRIVACERA_USERSYNC_CONTEXT_CLASS`	Implementation class used for USContext. Storage of synced Users and Groups.	`usersync.context.class`	`com.privacera.usersync.context.USContextRocksDB`
			Options:
			`com.privacera.usersync.context.USContextRocksDB`
			`com.privacera.usersync.context.USContextMemory`

`PRIVACERA_USERSYNC_CONTEXT_DATASOURCE_PRIORITY_LIST`	Priority list of configured datasources. Sources nearest the beginning of the list will be used over sources later in the list.	`usersync.context.datasource.priority.list`	

UserSync property	Description	Property	Default
PRIVACERA_USERSYNC_DETECT_CACHE_DIFFERENCES_ENABLED	To enable the cache synchronization. While UserSync reads data from an IdP, for performance, the incoming user data is kept in cache and periodically compared to user data already synced to the Privacera portal. From cache, UserSync pushes user data from the IdP that has been reconciled with the Privacera portal to the connected applications. .	usersync.detect.DifferencesBetweenCacheAndRangerForUserAndGroup.enabled	true
PRIVACERA_USERSYNC_DETECT_CACHE_INTERVAL_SECONDS	Frequency of cache synchronization in seconds.	usersync.ranger.compare.interval.seconds	43200
PRIVACERA_USERSYNC_LOADER_BULK_ENABLED	Load users to Portal in batches.	usersync.user.loader.bulk.enabled	TRUE
PRIVACERA_USERSYNC_LOADER_BULK_BATCHSIZE	Size of batches to load Users into Portal.	usersync.user.loader.bulk.batchsize	100
PRIVACERA_USERSYNC_UPDATE_GROUP_MEMBERSHIPS_BATCH_ENABLE	Load group memberships to Portal in batches.	usersync.user.loader.update.group.memberships.batch.enable	FALSE
PRIVACERA_USERSYNC_UPDATE_GROUP_MEMBERSHIPS_BATCHSIZE	Size of batches to load Group memberships into Portal.	usersync.user.loader.update.group.memberships.batchsize	1000
PRIVACERA_USERSYNC_STARTUP_PERFORM_OPERATIONS_ENABLED	Scan for and perform any pending operations in cache (User/Group objects) at service start-up	usersync.startup.performoperations.enabled	TRUE
PRIVACERA_USERSYNC_LOADER_PROCESS_THREAD_MIN	Minimum threads for processing user/group updates (<=0 will use a cached thread pool)	usersync.user.loader.process.thread.min	1
PRIVACERA_USERSYNC_LOADER_PROCESS_THREAD_MAX	Maximum threads for processing user/group updates (if min is <= 0, this has no effect)	usersync.user.loader.process.thread.max	1
PRIVACERA_USERSYNC_LOADER_PROCESS_THREAD_KEEPALIVE	Keep alive value for threads in pool.	usersync.user.loader.process.thread.keepalive	30
	JCEKS KeyStore File Paths	privacera.usersync.keystore.files	
	JCEKS KeyStore Files Passwords	privacera.usersync.keystore.passwords	
	Secure keys alias prefix	privacera.usersync.secure.key.prefix	jceks
PRIVACERA_USERSYNC_AUTH_SSL_TRUSTSTORE_FILE	SSL Truststore path	ssl.truststore	
PRIVACERA_USERSYNC_AUTH_SSL_TRUSTSTORE_PASSWORD	SSL Truststore password	ssl.truststore.password	
PRIVACERA_USERSYNC_RANGER_INIT_RETRY_INTERVAL_IN_MILLIS	Delay in milliseconds between retry attempts for initializing Ranger user loader.	usersync.user.loader.ranger.init.retryinterval.ms	30000

UserSync property	Description	Property	Default
PRIVACERA_USERSYNC_RANGER_INIT_RETRY_LIMIT	Maximum retry attempts for initializing Ranger user loader. (<0 indicates unlimited retries)	usersync.user.loader.ranger.init.retrylimit	-1
PRIVACERA_USERSYNC_RANGER_REQUEST_RETRY_INTERVAL_IN_MILLIS	Delay in milliseconds between retry attempts for requests to Ranger	ranger.request.retryinterval.ms	10000
PRIVACERA_USERSYNC_RANGER_REQUEST_RETRY_LIMIT	Maximum retry attempts for requests to Ranger	ranger.request.retrylimit	3
PRIVACERA_USERSYNC_UPDATE_GROUP_MEMBERSHIPS_BULK_ENABLED	Enable bulk update of group memberships to Ranger	usersync.user.loader.update.group.memberships.bulk.enabled	TRUE
PRIVACERA_USERSYNC_CONTEXT_OPEN_MAX_RETRY	Maximum retry attempts to open RocksDB cache.	usersync.context.rocksdb.open.max.retry	5
PRIVACERA_USERSYNC_CONTEXT_OPEN_DESTROY_ON_FAIL	Enable automatic destroy of RocksDB cache if unable to open (corrupted). Cache will be rebuilt.	usersync.context.rocksdb.open.destroyonfail	TRUE
PRIVACERA_USERSYNC_LOADER_ASSIGN_ROLE_PRIORITY_LIST	Priority list of roles if a user has multiple roles mapped. Highest priority role will be applied to the user.	usersync.user.loader.assign.role.priority.list	ROLE_SYS_ADMIN,ROLE_ADMIN_AUDITOR
PRIVACERA_USERSYNC_API_SECURITY_USER_NAME	If configured, Usersync REST APIs are available with basic auth.	usersync.api.security.user.name	
PRIVACERA_USERSYNC_API_SECURITY_USER_PASSWORD	If configured, Usersync REST APIs are available with basic auth.	usersync.api.security.user.password	

About Ranger UserSync

The Ranger UserSync service syncs users, groups, and group memberships from various idP sources.

Customize user details on sync

You can use advanced settings to perform a variety of transformations on user-related data from your Identity Provider (IdP).

- On the **Base User Attributes** page, go to the **ADVANCED** tab.
  Then on the **Customize Base LDAP Source Attribute Keys** dropdown:
  - **Apply to Attribute(s)**: Specify the names of LDAP attributes to which these customizations apply.
  - **Extract from email**: Extract the username portion of an email address value from the username attribute field. The username then becomes the value to the left of the `@-sign` of the email address.

CAUTION

Duplicate usernames map to a single username.

Usernames in email addresses (the left-hand side of the @-sign) that are identical even if they are different domains (the right-hand side of the @-sign) are considered the same user on import.

For example these email addresses with different domains result in the same user on :

- BillSmith@gmail.com
- BillSmith@yourcompany.com

- **No Conversion**: If you select any of the following conversions, this radio button is unselected.
- **Convert attribute values to lowercase**. Allowable values: `true` or `false`.
- **Convert attribute values to uppercase**. Allowable values: `true` or `false`.
- **Prefix to Prepend**. Allowable values: String to prepend.
- **Postfix to Append**. Allowable values: String to append.
- **Regex Replace Expression**. Allowable values: Substitute any string represented by a regular expression with another string. Requires Linux-editor-style `s` command, with optional `g` for global replacement. Examples:
  - Regexp `s/ch/AAA/g`: `ch` (a simple string) is globally replaced with `AAA`.
  - Regexp `s/[123]//`: The first occurrence of the number `1` or `2` or `3` is removed.
  - Regexp `s/a[bc]/z/`: The first occurence of the letter `a` optionally followed by either `b` or `c` is replaced by `z`.

UserSync integrations

Privacera UserSync can be integrated with many IdPs.

SCIM Server User-Provisioning on PrivaceraCloud

NOTE

Contact Privacera Support to request enabling this feature.

PrivaceraCloud can be configured to use the System for Cross-Domain Identity Management (SCIM) 2.0 protocol.

This allows external management and synchronization of your PrivaceraCloud data access users and groups.

After you connect your Identity Provider to your PrivaceraCloud account via SCIM, data user attributes and group memberships are managed in the Identity Provider, not in PrivaceraCloud.

NOTE

Data access users, groups, and roles can still be added in PrivaceraCloud, but users, groups, or roles created in PrivaceraCloud are *not* synchronized back to the Identity Provider.

SCIM server user provisioning does not apply to portal users (those who login to the portal).

Enable SCIM Server in PrivaceraCloud

1. In your PrivaceraCloud account, navigate to **Settings > Datasource**
2. Choose **UserSync**.
3. Enter a name to identify the connector. Click **Next**.
4. Copy the **Endpoint URL**.
   Enter a value for **Username** and **Password**. Save all three values: Endpoint URL, Username, and Password values as they will be used by your SCIM Client or Okta SCIM Client.
   Click **Next**.
5. Users or groups specified in **Inclusions** are added to filters. Specify any user or group **Exclusions** and click **Next**.
6. **Base User Attributes** maps identity attributes in the SCIM payloads to Privacera data access user attributes.
   - On the right the source fields for PrivaceraCloud users.
   - These value can be available from your Okta or other identity provider.

Okta Identity Provider Integration

For Okta SCIM client operations supported by PrivaceraCloud, see Supported Okta SCIM Client Operations [50].

Prerequisites

1. Obtain user provisioning functionality for your Okta account. For details, see Okta Lifecycle Management.
2. Resolve group name conflicts before syncing to your PrivaceraCloud account. Rename groups that have the same name in your identity provider and in your PrivaceraCloud account.

Recommendation: Before integrating your production users and groups, create a test account and group in Okta, such as `privacera-test-users`, and use those test values to confirm integration. When you are satisfied with the results, repeat the process using live production users and groups.

Integration Steps

Step 1. Enable SCIM API Integration in Okta

Be sure you have the **Endpoint URL** and the username and password you specified from Enable SCIM Server in PrivaceraCloud [48].

1. Log in to Okta and add the **PrivaceraCloud** application.
2. From the application, click the **Provisioning** tab.
3. Click **Configure API integration**.
4. Select **Enable API integration**.
5. Enter the **Endpoint URL** that was generated for you, and the **Username/Password** you provided in your PrivaceraCloud account in Enable SCIM Server in PrivaceraCloud [48].
6. Click **Test API** Credentials. If the test passes, click **Save**.
7. Under **Settings**, click **To App**.
8. Click **Edit** and select **Enable** for required options.
   Use this step to map user attributes or leave them with default settings.
9. Click **Save** to apply the integration settings.

Step 2: Activate application features

1. From the application, click the **Provisioning** tab.
2. Click **Edit**.

3. Click the **Enable** checkbox for **Create Users** and **Update User Attributes**.
4. Click **Save**

Step 3. Verify Email Addresses

User provisioning in Okta relies on an email address to identify a user in PrivaceraCloud and consequently create or update a Privacera Cloud account. If the email address attribute for a user is inconsistent between the SAML SSO setting and the SCIM user provisioning setting in Okta, the user might end up with duplicate PrivaceraCloud accounts.

To avoid duplicate accounts, verify the email address attribute that maps to a user account is correct and used for both SAML SSO and SCIM user provisioning:

1. From the **User provisioning** tab in Okta, note the field that maps to the **Primary email** attribute. The default is **email**,.
2. Click the **Sign on** tab. From the **Credentials details** section, look for the **Application username format** setting. Okta passes this field from a user's account as the SSO email address when creating or linking an account.
   If **Application username format** specifies an old value (for example, the old email address is `user1OLD@example.com` for the specified attribute) but you have another attribute that stores the same user's email address `user1NEW@example.com`, the user might end up with duplicate accounts. Troubleshoot as follows: - *Before you complete this step*, ask the user to log in with their PrivaceraCloud account at least once. - If the user still ends up with duplicate accounts, contact PrivaceraCloud support with the user's email addresses.
3. Make sure the **Application username format** is set to the same attribute specified as **Primary email** in the previous step.
4. Make sure that **Update application username** is set to **Create and update**. Click **Save** to apply your changes.
5. Click **Update Now** to push the change faster than the Okta automatic update.

Step 4. Push Groups

Privacera recommends using the group synchronization feature to automatically manage user privileges and licenses from your directory, instead of manually managing these from the organization.

This section describes how to configure group-based management.

Pushing a group to your PrivaceraCloud account pushes only the detail about a group, not details about users who are part of a group.

1. In Okta, click the **Push Groups** tab and then **By name**.
   Select the group name, and click **Save**.
2. Review to make sure all required groups have been pushed.

Step 5. Assign Users to the PrivaceraCloud Application in Okta

1. In Okta, click the **Assignments** tab of the PrivaceraCloud application.
2. Click **Assign**, then **Groups**. Select the group to assign.
3. In the displayed dialog, these default values are used only if the user profile does not have them. All fields are optional.
   When you are done with this step, click **Save and Go Back**.
4. In PrivaceraCloud, to verify that users and groups are synced, navigate to **Access Manager > Users Groups and Roles**.

Step 6. Write a Policy for Provisioned Users or Groups

Create a data access policy for the provisioned users or groups through PrivaceraCloud to finalize the integration verification.

1. From your PrivaceraCloud Account, navigate to **Access Management > Resource Policies**.
2. Select an application to write a policy over, such as Privacera Hive.
3. Click **Add Policy** and enter the details as shown below.
4. Verify the policy is in effect in your downstream application.

Supported Okta SCIM Client Operations

User Operations

The Privacera SCIM Server supports the following operations:

Operation	Notes
Create new user	A data access user is created in PrivaceraCloud **Access Management -> Users**. This account cannot be edited directly in PrivaceraCloud.
Link an existing user	If a user already exists in your PrivaceraCloud Account, it is automatically linked to the user in Okta. This account can no longer be edited directly in PrivaceraCloud.
Update user details	In PrivaceraCloud, you can update the display name and email address user attributes from your identity provider. By default, a user's first and last names are combined to create the **Display name**. If Any display name entered in PrivaceraCloud overwrites the first and last name combination.
Deactivate user	Deactivate is also sometimes called "soft delete". Deactivation has the following effects: • The user is marked as hidden. • The user is removed from all groups. **Important**: the Privacera administrator must manually remove the deactivated user from all user-based policies. If your policies are based on groups (not specific users), no changes to policies are needed, because the user has been removed from the groups. You need to manually change a policy only if the user is explicitly named in it.
Delete user	Delete the user manually in PrivaceraCloud.

Group Operations

Groups created manually and by default (for example, public) in your PrivaceraCloud account cannot be managed via SCIM.

You can only manage groups synced from Okta via SCIM.

Operation	Notes	Troubleshooting
Create group	The group is created as an read-only external group in PrivaceraCloud.	
Update group membership	PrivaceraCloud external data access users in your PrivaceraCloud account are modified to support the group membership changes reflected in this SCIM operation.	If a synced group is empty, when pushing a group, make sure that the synchronized group does not have the same name as a default or manually created group in PrivaceraCloud.
Push group	An error will result if the group name already exists in your PrivaceraCloud account. For example, the group named "public" might conflict.	In your identity provider, rename the conflicting group with a name different from the PrivaceraCloud built-in group name and attempt to re-sync.
Delete group	Manually delete the user account in PrivaceraCloud. Not implemented via SCIM	

Okta SCIM Server - Configure custom user attributes

PrivaceraCloud Configuration

These steps will configure the Usersync connector to accept additional user attribute(s).

1. In the PrivaceraCloud portal, navigate to **Settings** -> **Datasources** -> **Usersync connector**.

2. In the “Custom User Attributes” section, add the custom attribute(s).
3. Save the configuration,

Okta Configuration

In Okta you will need to add application profile attributes and mappings.

1. Login to Okta. Navigate to **Applications**.
2. Select your PrivaceraCloud application.
3. Then select "Provisioning (To App)".
4. In the "Local Attribute Mappings" section, click "Go to Profile Editor".
5. Click **Add Attribute**.

Table 1. Attribute examples:

Data type	string
Display name	Title
Variable name	title
External name	title
External namespace	urn:ietf:params:scim:schemas:core:2.0:User
Description	User Title
Enum	
Attribute length	
Attribute required	
Scope	User Personal

6. Click **Mappings**.
7. Select "Okta User to PrivaceraCloud". For example:

user.title	title

For additional information, see the Okta documentation.

SCIM Server API

SCIM 2.0 clients can also connect directly with the Privacera SCIM Server via the REST API.

Follow the steps in Enable SCIM Server in PrivaceraCloud [48] to obtain the direct URL, username, and password.

Use Basic Authentication (base64 encoded username and password) to authenticate each API request.

See SCIM 2.0 Specification for specific protocol and call schemas.

Supported SCIM REST API Requests

PrivaceraCloud supports the following requests:

```
GET - /Users
    - Params
        - startIndex
        - count
        - filter (Supports single filter for ‘eq’ operator on the following attributes:

GET - /Users/{userId}

POST - /Users
    - Params
        - user (SCIM User JSON)
```

```
PUT - /Users/{userId}
    - Params
        - user (SCIM User JSON)

GET - /Groups
    - Params
        - startIndex
        - count
        - filter (Supports single filter for ‘eq’ operator on the following attributes:

GET - /Groups/{groupId}

POST - /Groups
    - Params
        - group (SCIM Group JSON)

PATCH - /Groups/{groupId}
    - Params
        - operations  (SCIM Operation list)   Supported Operations:
                                op: replace
                                path:members
                                value:[{value: userId} … ]

                                op: remove
                                path: members[value eq “userId”]

                                op: add
                                path: members
                                value: [{value:userId} … ]
```

Azure Active Directory UserSync integration on Privacera Platform

This topic covers how you can synchronize users, groups, and service principals from your existing Azure Active Directory (AAD) domain.

Prerequisites

Ensure the following pre-requisites are met:

- Create an Azure AD application.
- Get the values for the following Azure properties: Application (client) ID, Client secrets

Procedure

1. SSH to the instance as ${USER}.
2. Run the following commands.

   ```
   cd ~/privacera/privacera-manager
   cp config/sample-vars/vars.usersync.azuread.yml config/custom-vars/
   vi config/custom-vars/vars.usersync.azuread.yml
   ```
3. Edit the following properties. For property details and description, refer to the **Configuration Properties** below.

   ```
   USERSYNC_AZUREAD_TENANT_ID: "<PLEASE_CHANGE>"
   USERSYNC_AZUREAD_CLIENT_ID: "<PLEASE_CHANGE>"
   USERSYNC_AZUREAD_CLIENT_SECRET: "<PLEASE_CHANGE>"
   USERSYNC_AZUREAD_DOMAINS: "<PLEASE_CHANGE>"
   USERSYNC_AZUREAD_GROUPS: "<PLEASE_CHANGE>"
   ```

```
USERSYNC_ENABLE: "true"
USERSYNC_SOURCE: "azuread"
USERSYNC_AZUREAD_USE_GROUP_LOOKUP_FIRST: "true"
USERSYNC_SYNC_AZUREAD_USERNAME_RETRIVAL_FROM: "userPrincipalName"
USERSYNC_SYNC_AZUREAD_EMAIL_RETRIVAL_FROM: "userPrincipalName"
USERSYNC_SYNC_AZUREAD_GROUP_RETRIVAL_FROM: "displayName"
SYNC_AZUREAD_USER_SERVICE_PRINCIPAL_ENABLED: "false"
SYNC_AZUREAD_USER_SERVICE_PRINCIPAL_USERNAME_RETRIVAL_FROM: "appId"
```

4. Run the following commands.

```
cd ~/privacera/privacera-manager
./privacera-manager.sh update
```

AAD UserSync configuration properties

Property Name	Description	Example
USERSYNC_AZUREAD_TENANT_ID	To get the value for this property, Go to Azure portal > Azure Active Directory > Properties > Tenant ID	5a5cxxx-xxxx-xxxx-xxxx-c3172b33xxxx
USERSYNC_AZUREAD_CLIENT_ID	Get the value by following the **Pre-requisites** section above.	8a08xxxx-xxxx-xxxx-xxxx-6c0c95a0xxxx
USERSYNC_AZUREAD_CLIENT_SECRET	Get the value by following the **Pre-requisites** section above.	${CLIENT_SECRET}
USERSYNC_AZUREAD_DOMAINS	To get the value for this property, Go to Azure portal > Azure Active Directory > Domains	componydomain1.com,componydomain2.com
USERSYNC_AZUREAD_GROUPS	To get the value for this property, Go to Azure portal > Azure Active Directory > Groups	GROUP1,GROUP2",GROUP3
USERSYNC_ENABLE	Set to true to enable usersync.	true
USERSYNC_SOURCE	Source from which users/groups are synced. **Values**: unix, ldap, azuread	azuread
USERSYNC_AZUREAD_USE_GROUP_LOOKUP_FIRST	Set to true if you want to first sync all groups and then all the users within those groups.	true
USERSYNC_SYNC_AZUREAD_USERNAME_RETRIVAL_FROM	Azure provides the user info in a JSON format. Assign a JSON attribute that is unique. This would be the name of the user in Ranger.	userPrincipalName
USERSYNC_SYNC_AZUREAD_EMAIL_RETRIVAL_FROM	Azure provides the user info in a JSON format. Set the email from the JSON attribute of the Azure user entity.	userPrincipalName
USERSYNC_SYNC_AZUREAD_GROUP_RETRIVAL_FROM	Azure provides the user info in a JSON format. Use the JSON attribute to retrieve group information for the user.	displayName
SYNC_AZUREAD_USER_SERVICE_PRINCIPAL_ENABLED	Set to true to sync Azure service principal to the Ranger user entity	false
SYNC_AZUREAD_USER_SERVICE_PRINCIPAL_USERNAME_RETRIVAL_FROM	Azure provides the service principal info in a JSON format. Assign a JSON attribute that is unique. This would be the name of the user in Ranger.	appId

LDAP UserSync integration on Privacera Platform

This topic covers how you can configure Privacera Platform to attach and import users and groups defined in an external Active Directory (AD), LDAP, or LDAPS (LDAP over SSL)) directory as *data access* users and groups.

Prerequisites

Before starting these steps, prepare the following. You need to configure various Privacera properties with these values, as detailed in Configuration.

Determine the following LDAP values:

- The FQDN and protocol (http or https) of your LDAP server
- DN
- Complete Bind DN
- Bind DN password
- Top-level search base
- User search base

To configure an SSL-enabled LDAP-S server, Privacera requires an SSL certificate. You have these alternatives:

- Set the Privacera property `USERSYNC_SYNC_LDAP_SSL_ENABLED: "true"`.
- Allow Privacera Manager to download and create the certificate based on the LDAP-S server URL. Set the Privacera property `USERSYNC_SYNC_LDAP_SSL_PM_GEN_TS: "true"`.
- Manually configure a truststore on the Privacera server that contains the certificate of the LDAP-S server. Set the Privacera property `USERSYNC_SYNC_LDAP_SSL_PM_GEN_TS: "false"`.

Procedure

1. SSH to instance as `${USER}`.
2. Run the following commands.

```
USERSYNC_SYNC_LDAP_URL: "<PLEASE_CHANGE>"
USERSYNC_SYNC_LDAP_BIND_DN: "<PLEASE_CHANGE>"
USERSYNC_SYNC_LDAP_BIND_PASSWORD: "<PLEASE_CHANGE>"
USERSYNC_SYNC_LDAP_SEARCH_BASE: "<PLEASE_CHANGE>"
USERSYNC_SYNC_LDAP_USER_SEARCH_BASE: "<PLEASE_CHANGE>"
USERSYNC_SYNC_LDAP_SSL_ENABLED: "true"
USERSYNC_SYNC_LDAP_SSL_PM_GEN_TS: "true"
```

3. Run Privacera Manager update.

```
cd ~/privacera/privacera-manager
./privacera-manager.sh update
```

LDAP UserSync configuration properties

Property	Description	Example
`USERSYNC_SYNC_LDAP_URL`		`"ldap://dir.ldap.us:389"` (when NonSSL) or `"ldaps://dir.ldap.us:636"` (when SSL)
`USERSYNC_SYNC_LDAP_BIND_DN`		`CN=Bind User,OU=example,DC=ad,DC=example,DC=com`
`USERSYNC_SYNC_LDAP_BIND_PASSWORD`		
`USERSYNC_SYNC_LDAP_SEARCH_BASE`		`OU=example,DC=ad,DC=example,DC=com`
`USERSYNC_SYNC_LDAP_USER_SEARCH_BASE`		
`USERSYNC_SYNC_LDAP_SSL_ENABLED`	Set this to true if SSL is enabled on the LDAP server.	`true`

Property	Description	Example
USERSYNC_SYNC_LDAP_SSL_PM_GEN_TS	Set this to true if you want Privacera Manager to generate the truststore certificate. Set this to false if you want to manually provide the truststore certificate. To learn how to upload SSL certificates, [click here](../pm-ig/upload_custom_cert.md).	true

Policies

Policies are rule sets for usage and access. Each rule specifies a scope of control, access type, a set of user identities allowed or denied use, along with enforcement periods. Access control list schemes support both “Allow”, and “Deny” access as well as “Exclude from Allow” and “Exclude from Deny”.

Resource Policies and **Tag Policies** are concerned with access to data.

Controlled access datasets are subsets of connected data repositories and databases, defined by any combination of database, table, and column access (*wildcards supported*) or for filesystem/object stores based on object, file, folder names (with *wildcard support* for paths).

For Resource Policies [58], the scope of control is data accessible through the connected data repositories as defined by resource paths.

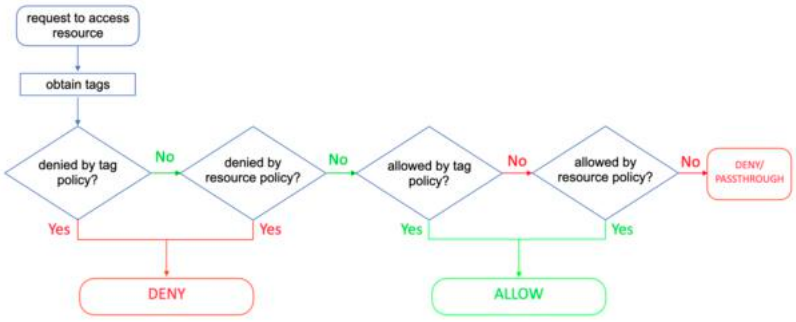
- For databases structured resources, scope is specified in terms of database, table, and column access. The type of access is defined based on the actions that can be performed on that particular type of database, such as “Select”, “Update”, “Create”, “Drop”, “Alter”, etc.
- For filesystem/object stores, the scope is defined in terms of access to entities such as a blob, object, file, or folder. Permission rules will be in the form of file actions such as "Read”, “Write”, and "Delete".

For Tag Policies [208], the scope of control are those data elements that have been assigned a metadata label or *tag*. Tag Policies are defined in terms of the *tags* rather than the tagged data itself or the location of the data itself.

Scheme Policies are used to specify user access to encryption and decryption services provided by the Privacera Encryption Gateway (PEG). The *Scheme Policies* page is enabled when the PEG service is connected.

How polices are evaluated

To authorize access to a resource, the system evaluates the resource- and tag-based policies associated with the resource as shown in the following diagram.



Tag-based policies

The system's policy engine evaluates the tag-based policies applicable to the tags:

- If a policy results in a deny, access is denied.
- If none of the tags is denied, and if a policy allows for one of the tags, access is allowed.
- If there is no result for any tag, or if there are no tags for the resource, the policy engine then evaluates the resource-based policies (as opposed to the tag-based policies).

Using tags in conditions

You can use custom conditions for evaluating authorization policies. The system policy engine makes various request details – such as user, groups, resource, and context – available to the conditions. Tags

in the request context are available to the conditions and can be used to influence the authorization decision.

The default policy in tag service instances, the EXPIRES_ON tag, uses such conditions to verify that the request date is later than the value specified in tag attribute expiry_date. This default policy does not work unless an EXPIRES_ON tag has been created, which the Privacera Platform defaults to "never".

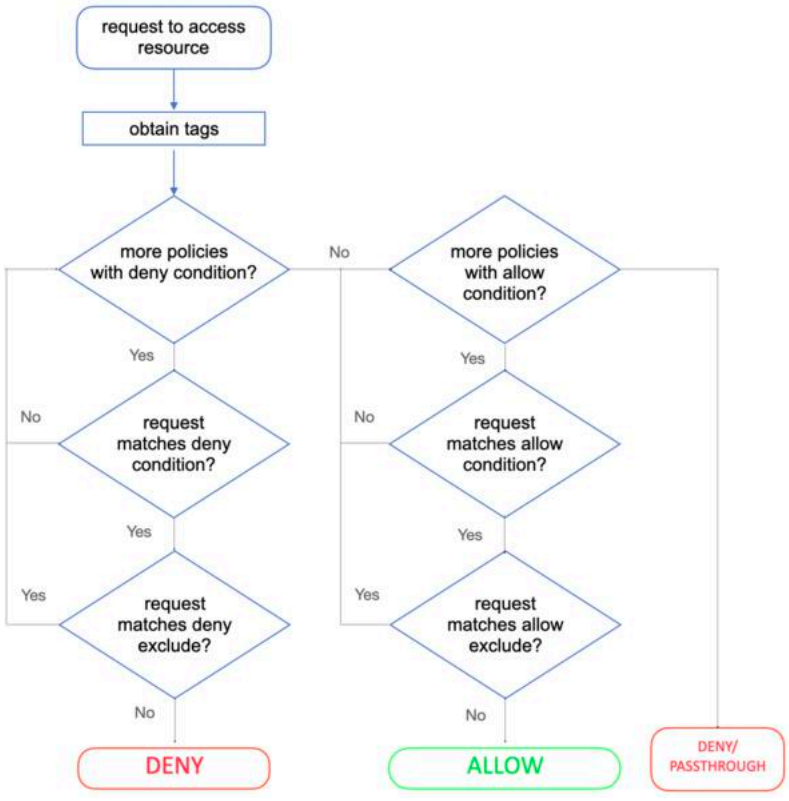
Allow deny and exclude conditions

The system supports the following access conditions:

- Allow
- Exclude from Allow
- Deny
- Exclude from Deny

These access conditions are for fine-grained access control policies.

For example, you can allow access to a "manufacturing" database to all users in the "manufacturing" group, but deny access to all users in the "design" group. Suppose that "scott", a member of the "design" group, must work on an assignment that requires access to the "manufacturing" database. In that case, you can add an **Exclude from Deny** condition so user "scott" can access the "manufacturing" database. The following diagram shows how this policy can be defined in the system:



Normal vs override priority

You can define the priority of a resource-based and tag-based policy as normal (the default) or override.

A matching override policy takes precedence over all normal-priority policies.

If there are multiple override policies the relationship among them is governed by the same flow normal policies.

- Any applicable override policy overrides all normal policies.
- Multiple override policies interact the same rules as multiple normal policies.

Evaluation order for row filtering and column masking

Within a single row-filter/column-masking policy, the defined order of filters/masks determines the choice of masking type.

For example, consider this logical representation of a masking policy.

- The protected data are database `db1`, table `tb1`, `col1`.
- The user `user1` is in `group1` and all users, including `user1`, are in group `public`:

```
resource: database=db1, table=tb1, column=col1 - users:[user1] maskType=NO_MASK - grou
```

1. When `user1` accesses column `db1.tbl1.col1`, `user1` can see the unmasked value because `maskType NO_MASK` is applied.
2. For other users in `group1`, `maskType HASH` is applied and so the column is displayed but the values are not visible.
3. For all other users `NULL` is returned: access is denied and the column is not displayed.

General approach to validating policy

For all datasources you have connected, to validate that your policies protecting the datasource are working, you have several general approaches:

- Use the datasource's own features such as filters for a particular user to examine the results of the policy or the datasource's own mechanisms for accessing the protected data. For examples of validation using a datasource's native mechanisms, see Example: Create basic policies for table access [223].
- In Privacera, look at the audit logs [244] and narrow the search of audit info [254] by datasource, user, or group.
- Examine specific categories of Privacera audits [243], such as the following, depending on how the datasource was connected, by Privacera plug-in or Privacera PolicySync:
  - **Access Management > Audit > Plugin**
  - **Access Management > Audit > PolicysSync**

Resource policies

The Resource Policies page displays a list of resource service groups and resource services.

A resource service represents:

- Connection to one or more data repositories.
- A set of policies.

The first default service in each service group is assigned a name using the form "privacera_<service_type>".

Each resource service contains a set of resource policies, which, in turn, contain access rules for this data resource or subset.

About service groups on PrivaceraCloud

A resource service group is a collection of services sharing similar attributes and configuration parameter requirements. On PrivaceraCloud, a service group and its first default service is created in **Settings** > **Applications**. For more information about application, see Manage applications on PrivaceraCloud.

Service/Service group global actions

- **Refresh button**: updates service groups and resource services.
- **Security Zone filter**: Filter service groups and services to display only those associated with selected Security Zones. For more information about Security Zones, see Security zones [213].
- **EXPORT button**: You can export all service types, and services in the service group will be pre-loaded. You have the option of removing the service type and the service name. Click the **Save** button, then all policies in the selected elements will be exported to a JSON formatted policy set.
- **IMPORT button**: You can import previously exported policy set. Browse the file and and then Click the **IMPORT** button. If the **Override Policy** checkbox is selected then it will allow the import to overwrite existing destination service policies. Click the **IMPORT** button to initiate the import.
- Click the three vertical dots in the service group to see the following actions:
  - **Add Service**: a new resource-based service, click the Add 'icon in the applicable box on the Resource Policies page. Enter the required configuration details, then click Save. Different service types have different attributes but all service types include a Service Name (required), Description (optional), optional associated Tag Service and accept a Username, Common Name for Certificate, and optional Key/Value pairs.
  - **Export** : You can export one or more services in the service group. By default, all services in a group are listed in the dialog but can be deselected. All policies in the selected services will be exported to JSON formatted policy set. Click Save to initiate a file browser and save dialog.
  - **Import** : You can import previously exported policy set. Browse the file and then click the **IMPORT** button. If the **Override Policy** checkbox is selected then it will allow the import to overwrite existing destination service policies. Click the **IMPORT** button to initiate the import.

Service actions

In front of each service type, you will see the following action buttons:

- **View** button: View the service details in read-only format.
- **Edit** button: Edit the configuration details.
- **Delete** button: Delete a resource-based service.

Policy definition

Click a service name (for example, *privacera_hive*) to open to the Policy definition and management page for this service . The page will display the existing polices for this service along with an **Add New Policy** button.

Each Policy definition row shows key attributes (**Policy ID**, **Policy Name**, **Policy Labels**, **Roles**, **Groups**, **Users**, and **Action**).

Under the Action column are three action icons:

- Preview button
- Edit button
- Delete button

To see an individual policy detail, either click the **Policy ID** number or **Edit** button. **Policy Detail** page will be displayed.

The Policy Details page contains the following fields:

- **Policy Type**: The basis for controlling access. For example, a policy can be based on the resource, on a tag, or on a scheme.
- **Policy Id:** Each policy is assigned an immutable numeric identifier. These ids are monotonically incremented and unique within each PrivaceraCloud account. Policy identifiers are referenced in the audit trail event messages, so that action taken and recorded to the audit trail is associated with a specific policy.

- **Policy Name:** Polices are assigned a name, either by the system or when created by a portal user. Default, system-created policies can be renamed.
- **ADD VALIDITY PERIOD:** A policy can be defined as being effective only for a period of time. Start and end dates and times (defined to the minute), as well as a time zone selection
- **Policy Label:** Policies can be assigned a new or existing label. Labels assist in filtering and with search reports.
- **Add Validity Period**: A policy can be defined as being effective only for a period of time. Start and end dates and times (defined to the minute), as well as a time zone selection.
- **Resource Specifier:** Underneath the Policy Label field are the Resource specifiers. These will be different for each type of resource, and the set of specifiers will change depending on the top down choices. For example, by default a Hive resource will display fields for **database**, **table**, and **column**. However, each prompt field, is a drop-down menu list with other options. Click the down-arrow in the database prompt field and there will be two other options: **url** and **global**. Select url to specify a URL as the Hive resource. Note that table and column are not relevant to specifying a URL, so those choices are removed.
- **Description**: This field required description of policy which can be used to identify among others policies.
- **Audit Logging**: Enable/disable Audit Logging. Toggle to **No**, if this policy doesn't need to be audited. By default, it is selected as **Yes**.
- **Condition Sets:** These are the rules that are used to determine allowed or denied access to the identified resource(s). Each is defined in terms of a set of data access permissions and data access individual users, user groups, or user roles. The permission selection list is specific to the type of service. For example, for the ADLS service, the permission set is read, write, delete, metadata read, metadata write, and admin. The following access conditions are available:
  - **Allow Conditions**
  - **Exclude from Allow Conditions**
  - **Deny Conditions**
  - **Exclude from Deny Conditions**

At least one rule must be defined. Rules for the other condition sets can be omitted.

Any service named "privacera_<service type>" automatically creates one or more default **all...** policies. (The policy names vary depending on the service. For example, the **all** policy for hive services is **all - database**. The default policy name for database repository services is **all - database, schema, table, column**, etc.).

Create resource policies: general steps

Using Access Management, you can create and configure policies that control access to specific resources.

To create a resource policy, follow these steps:

1. From the home page, click **Access Management** > **Resource Policies**.
2. Click a service in one of the service groups.
3. Click **Add New Policy**.
4. Define the new resource policy.

The exact definition of the policy depends on the application and the nature of the data you want to protect.

About secure database views

Many connected applications do not have the native ability to enforce some kinds of resource policies directly in the associated tables. For example, Databricks SQL does not have the native capability to create column masks or row filters.

For this reason, Privacera creates a secure view of the original database and applies policy to that secure view. The name of a secure view is:

*originalDatabaseName*_secure

In Privacera, the access policy itself must always specify the name of the *original* database, not the secure view.

In the access policy, make sure you remove the **Data Admin** permission for the user. Otherwise, the user can see the original, unprotected database.

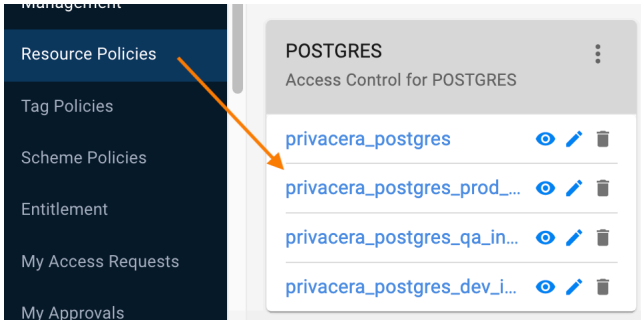
NOTE
You should tell users the name of the secure view for their queries and that access to the original database is no longer allowed.

PolicySync design on Privacera Platform

PolicySync design and configuration on Privacera Platform

To help you reach compliance, Privacera PolicySync distributes your defined access management policies to the third-party datasources you connect to Privacera.

Policies are created in Privacera at **Access Management > Resource Policies [58]** or **Access Management > Tag Policies [208]** in policy repositories:



Some of the characteristics of a PolicySync connector are:

- Authentication to connect to the third-party system, such as JDBC URL, username, and password,
- Polling intervals for syncing policies or auditing and other network-related characteristics.
- Multiple instances of a connector to distribute work across Kubernetes pods and Docker containers.

Relationships: policy repository, connector, and datasource

The mechanics of PolicySync involve a Privacera *connector* with its Privacera *policy repository* to distribute your defined policies from that repository to a third-party *datasource*:

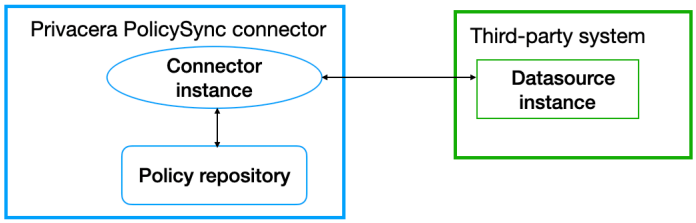
- An internal-to-Privacera *policy repository* stores the access management policies you create via the Privacera UI at **Access Management > Resource Policies**. These are the policies distributed by the connector to the datasource.
- An internal-to-Privacera instance of a *connector* to that third-party system is configured by YAML files on Privacera Platform that contain *properties* or fields in PrivaceraCloud. Properties and fields are name/value pairs to set various features or characteristics of the connector and third-party system.
- An external-to-Privacera instance of a third-party application's database or file or other object is called a *datasource*. This is your application.

PolicySync topologies

Connector instances, policy repository instances, and datasource instances can be configured in various topologies.

Required basic PolicySync topology

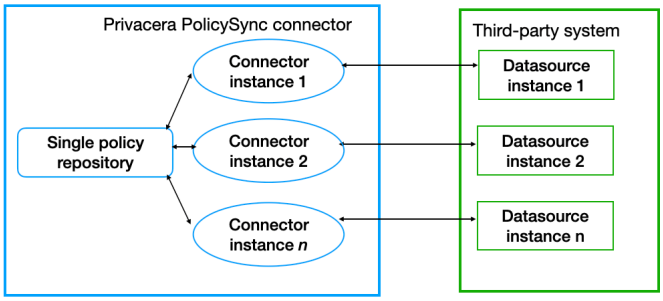
The default, basic, required topology is a single connector instance, a single policy repository, a single third-party system datasource.



Creating this basic topology is detailed in Required basic PolicySync topology: always at least one connector instance [64].

PolicySync topology: multiple connectors and datasources, single policy repository

An alternative to the basic topology is a single Privacera policy repository with multiple Privacera connector instances to distribute the same policies to several different datasources.

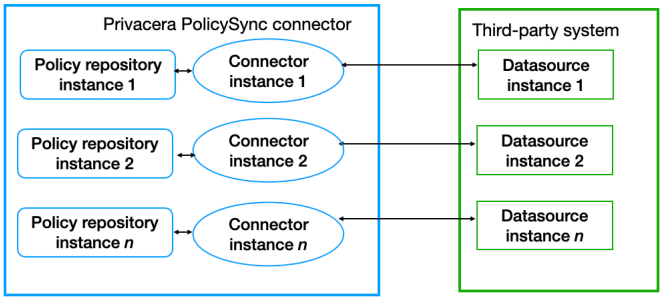


Creating this configuration is detailed in Optional topology: multiple connector instances for Kubernetes pods and Docker containers [65].

PolicySync topology: unique connectors, datasources, and policy repositories

The optimal topology is one connector instance with one policy repository instance for a single datasource instance.

For distinct access management policies for distinct organizational groups that require distinct access rights, Privacera recommends the PolicySync topology of a separate, unique policy repository for each connector instance and its associated datasource instance.



Creating this configuration is detailed in Recommended PolicySync topology: individual policy repositories for individual connectors [65]

Connector instance directory/file structure

PolicySync works with configured connector instances for each instance of the third-party system datasource you want to manage.

You configure connectors in the following directories in Privacera Manager, with a YAML properties file for each instance of the connector:

```
~/privacera/privacera-manager/config/custom-vars/connectors/<ConnectorName>/<someInstanc
```

where the following sections define this syntax.

Directory naming conventions for PolicySync

`~/privacera/privacera-manager/config/custom-vars/connectors/` is the base directory for all connector instances.

`<ConnectorName>` is a subdirectory with the lowercase, reserved-word name of a third-party system to connect to, such as `databricks-sql-analytics`, `postgres`, or `mssql`.

These names are reserved words defined by Privacera for each supported third-party system detailed in the PolicySync documentation for each.

- `biquery [109]` for Google BigQuery.
- `databricks-sql-analytics [69]` for Databricks SQL.
- `dremio [77]` for Dremio.
- `mssql [116]` for Microsoft SQL Server.
- `postgres [127]` for PostgreSQL on several different platforms.
- `powerbi [137]` for Power BI.
- `redshift` for Redshift and Redshift Spectrum connector for PolicySync [141]
- `snowflake [150]` for Snowflake.

`<someInstanceName>` is a subdirectory whose name you create to indicate a single instance of a connector for the third-party datasource.

`<someInstanceName>` must be composed of only alphanumeric characters and hyphens.

For example, suppose you have three different Kubernetes pods for three different instances of PostgreSQL. Let's name the `<someInstanceName>` subdirectories for these instances as follows:

- `postgres-dev-instance` for Software Development.
- `postgres-qa-instance` for Quality Assurance.
- `postgres-prod-instance` for Production.

YAML file with connector-specific properties

`vars.connector.<ConnectorName>.yml` is a YAML file of name-value pairs you complete for a connector instance, including authentication, features, and other details.

These property names are defined by Privacera for each third-party system and are detailed in the PolicySync documentation for each.

- `vars.connector.bigquery.yml`: see BigQuery connector properties for PolicySync on Privacera Platform [112].
- `vars.connector.databricks.sql.analytics.yml`: see Databricks SQL connector properties for PolicySync on Privacera Platform [70].
- `vars.connector.dremio.yml`: see Dremio connector properties for PolicySync on Privacera Platform [78].
- `vars.connector.mssql.yml`: see Microsoft SQL connector properties for PolicySync on Privacera Platform [118].

- `vars.connector.postgres.yml`: see PostgreSQL connector properties for PolicySync on Privacera Platform [128].
- `vars.connector.powerbi.yml`: see Power BI connector properties for PolicySync on Privacera Platform [138].
- `vars.connector.redshift.yml`: see Redshift and Redshift Spectrum connector properties for PolicySync on Privacera Platform [143].
- `vars.connector.snowflake.yml`: see Snowflake connector properties for PolicySync on Privacera Platform [153].

Required basic PolicySync topology: always at least one connector instance

For each third-party system datasource, you must always create at least one PolicySync connector instance with the directory structure and file naming described in Connector instance directory/file structure [63].

For example, suppose you have a single instance of PostgreSQL in production. Let's call this instance `postgres-prod-instance`.

Follow these steps to create the PolicySync configuration for it.

```
cd ~/privacera/privacera-manager
#
# Make the connector subdirectory
# for the PostgreSQL production instance
# called postgres-prod-instance
mkdir -p config/custom-vars/connectors/postgres/postgres-prod-instance
#
# Copy the template PostgreSQL connector properties .yml file
# to the postgres-prod-instance subdirectory
# for the production instance
cp config/sample-vars/vars.connector.postgres.yml \
config/custom-vars/connectors/postgres/postgres-prod-instance
#
# Set the values of the properties in the YAML file
# called vars.connector.postgres.yml
# for this example postgres-prod-instance instance
#
# Properties are defined in the documentation
# for each of the individual third-party systems.
#
vi config/custom-vars/connectors/postgres/postgres-prod-instance/vars.connector.postgres
#
# Save the file
#
# Update the running configuration
./privacera-manager.sh update
```

NOTE

Every time you save changes to a configuration file, you must run `privacera-manager.sh update` to apply those changes to the running configuration.

Optional topology: multiple connector instances for Kubernetes pods and Docker containers

Suppose you have three different Kubernetes pods for three different instances of PostgreSQL databases. Let's name the subdirectories for these instances as follows:

- `postgres-dev-instance` for Software Development.
- `postgres-qa-instance` for Quality Assurance.
- `postgres-prod-instance` for Production.

For each connector instance, create the directory/file structure following the details in Required basic PolicySync topology: always at least one connector instance [64] and define the PolicySync properties for each instance in its `vars.connector.<ConnectorName>.yml` YAML file.

You will have a directory structure like this:

```
cd ~/privacera/privacera-manager
ls -R config/custom-vars/connectors

config/custom-vars/connectors/postgres:
postgres-dev-instance    postgres-prod-instance    postgres-qa-instance

config/custom-vars/connectors/postgres/postgres-dev-instance:
vars.connector.postgres.yml

config/custom-vars/connectors/postgres/postgres-prod-instance:
vars.connector.postgres.yml

config/custom-vars/connectors/postgres/postgres-qa-instance:
vars.connector.postgres.yml
```

Recommended PolicySync topology: individual policy repositories for individual connectors

By default, when you configure multiple connectors, policies are stored in the connector's default single PolicySync policy repository. But you can define an individual policy repository for each individual connector instance and datasource instance.

For distinct access management policies for distinct organizational groups that require distinct access rights, Privacera recommends the PolicySync topology of a separate, unique policy repository for each connector instance and its associated datasource instance.

When you configure a connector's first instance, the following properties for the policy repository are added to your connector instance's YAML file, in this example, `~/privacera/privacera-manager/config/custom-vars/connectors/postgres/postgres-prod-instance/vars.connector.postgres.yml`.

Any new connector instances share that same single, common policy repository. The example below shows this shared repository: `privacera_postgres`.

```
ranger.policysync.connector.0.ranger.service.name=privacera_postgres
ranger.policysync.connector.0.ranger.service.appid=privacera_postgres
```

Setting up individual repositories per connector is a two-step process:

- In Privacera, go to **Access Management > Resource Policies**, create the individual repositories for the planned connectors.
- Set up the per-connector properties to point to the individual policy repositories.

Prerequisites

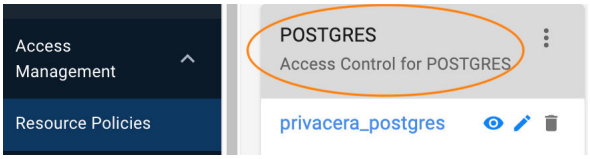
Make sure you have the following details ready:

1. Decide which connector to a third-party datasource to create the unique policy repository for. In this example, we work with PostgreSQL.
2. Decide names for the policy repository instances that correspond to the connector instances and their datasources.
   Use `privacera_` as a prefix for the repository instance to help distinguish the repository instance from the connector instance it is associated with.
   In this example, we decide to use the connector instance names we have already defined, each with the `privacera_` prefix:
   - `privacera_postgres_dev_instance` for Software Development.
   - `privacera_postgres_qa_instance` for Quality Assurance.
   - `privacera_postgres_prod_instance` for Production.
3. Be sure to have created at least one connector for the third-party system datasource, as detailed in Required basic PolicySync topology: always at least one connector instance [64].

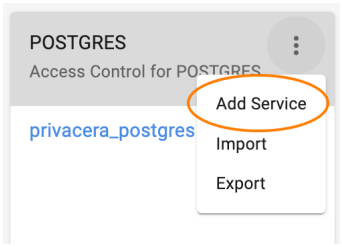
Create policy repository for each connector instance in Privacera Access Management

In this example, create the `privacera_postgres_prod_instance`. policy repository.

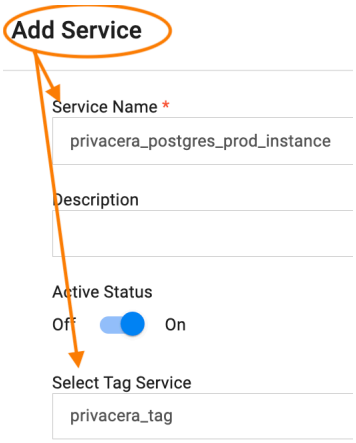
1. As administrator, go to **Access Management > Resource Policies**.
2. Locate the third-party connector already available. In this example, the single shared repository for PostgreSQL is shown:



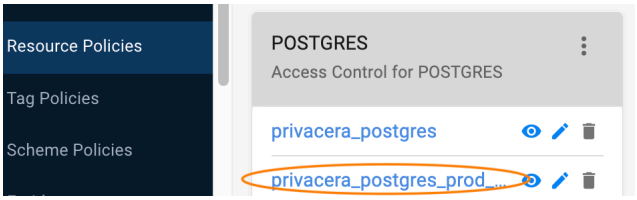
3. From the three vertical dots on the upper right, select **Add Service**:



4. Enter the **Service Name**. Follow your naming convention described in 2 [66]. In this example, `privacera_postgres_prod_instance`.
5. Under **Select Tag Service**, choose `privacera_tag`.

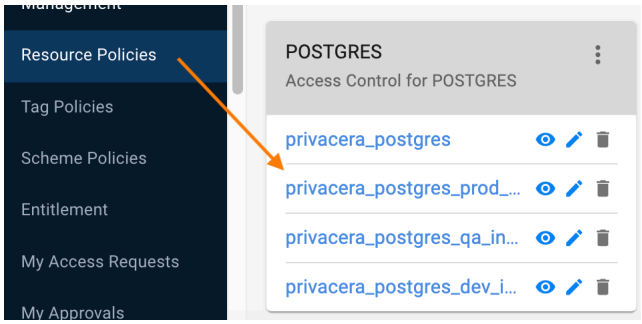


6. Click **Save**.
   The new repository has been created.



7. Repeat these steps for as many other policy repositories you want.

In this example, the three separate repositories are ready:



Configuring properties to point to individual policy repositories

When you create other connector instances, each points to the single common repository.

To define a separate unique repository, for each connector instance, create a `connector-custom.properties` file to replace the name of the shared repository with the name of the unique repository.

The `connector-custom.properties` file must be in the same directory as the connector instance's YAML file, as described in Connector instance directory/file structure [63].

```
~/privacera/privacera-manager/config/custom-vars/connectors/<ConnectorName>/custom/conne
```

Following the examples in this discussion, for each connector instance, specify the name of instance's policy repository on the following lines in `connector-custom.properties`:

- Connector instance for Software Development: `custom-vars/connectors/postgres/postgres-dev-instance/custom/connector-custom.properties`:

  ```
  ranger.policysync.connector.0.ranger.service.name=privacera_postgres_dev_instance
  ranger.policysync.connector.0.ranger.service.appid=privacera_postgres_dev_instance
  ```
- Connector instance for Quality Assurance: `custom-vars/connectors/postgres/postgres-qa-instance/custom/connector-custom.properties`:

  ```
  ranger.policysync.connector.0.ranger.service.name=privacera_postgres_qa_instance
  ranger.policysync.connector.0.ranger.service.appid=privacera_postgres_qa_instance
  ```
- Connector instance for Production: `custom-vars/connectors/postgres/postgres-prod-instance/custom/connector-custom.properties`:

  ```
  ranger.policysync.connector.0.ranger.service.name=privacera_postgres_prod_instance
  ranger.policysync.connector.0.ranger.service.appid=privacera_postgres_prod_instance
  ```

These three separate connector instances are now associated with their respective policy repositories viewable at **Access Management > Resource Policies [58]** or **Access Management > Tag Policies [208]**.

Optional encryption of property values

To protect sensitive values in your YAML files, PolicySync can encrypt them.

To encrypt property values, create the following file and specify the property names whose values to encrypt:

`~/privacera/privacera-manager/config/custom-vars/vars.encrypt.secrets.yml`

The contents of `vars.encrypt.secrets.yml` must look like this:

```
CONNECTOR_ENCRYPT_PROPS_LIST:
  - <PROPERTY_NAME1>
  - <PROPERTY_NAME2>
  - <PROPERTY_NAME3>
```

where:

- `CONNECTOR_ENCRYPT_PROPS_LIST:` is exactly as shown.
- The indented <PROPERTY_NAME1>, <PROPERTY_NAME2>, and <PROPERTY_NAME3> are names of connector properties whose values to encrypt.
  The property names can be for any connector.
  There is no limit to the number of property names you can specify.

Example: In the following example, the file `~/privacera/privacera-manager/config/custom-vars/vars.encrypt.secrets.yml` specifies encryption of the PostgreSQL connector's property values of JDBC username, JDBC password, and JDBC database and the Google BigQuery connector's property values of JDBC URL and organization ID:

```
CONNECTOR_ENCRYPT_PROPS_LIST:
  - CONNECTOR_POSTGRES_JDBC_USERNAME
  - CONNECTOR_POSTGRES_JDBC_PASSWORD
  - CONNECTOR_POSTGRES_JDBC_DB
  - CONNECTOR_BIGQUERY_JDBC_URL
  - CONNECTOR_BIGQUERY_ORGANIZATION_ID
```

Migration to PolicySync v2 on Privacera Platform 7.2

The PolicySync configuration has been significantly enhanced. The new configuration is sometimes called "PolicySync version 2" or "v2".

You can now configure multiple connector instances for a single application, such as Databricks SQL or PostgreSQL. Each connector instance can distribute policies to a separate instance of the third-party system. In addition, each connector can be configured to distribute policies from its own separate policy repository.

Steps to migrate old PolicySync properties to the new framework

The general process for migrating from older versions of PolicySync to the new framework is to create new subdirectories in Privacera Manager specifically for connectors with new YAML files and copy the values of your old properties to the new property names in the new YAML files.

1. Become familiar with the new PolicySync directory structure described at Connector instance directory/file structure [63].
2. Make a backup copy of all your current, unmigrated properties files:

   ```
   cd ~/privacera/privacera-manager
   mkdir policysync-vars-backup
   mv config/custom-vars/vars.policysync.*.yml policysync-vars-backup/
   ```
3. *For each of your connectors*, create the new required basic directory structure following the steps detailed at Required basic PolicySync topology: always at least one connector instance [64].

In this example, we create a production connector instance directory for PostgreSQL and change directory to it:

```
cd ~/privacera/privacera-manager/config/custom-vars/connectors
mkdir postgres/postgres-prod-instance
cd postgres/postgres-prod-instance
```

4. Copy the skeleton properties YAML file from `~/privacera/privacera-manager/config/sample-vars` to the new instance directory properties YAML file.

> **NOTE**
> The name of the new file must be as shown below: `vars.connector.<ConnectorName>.yml`, in this example `vars.connector.postgres.yml`.

```
cp ~/privacera/privacera-manager/config/sample-vars/vars.connector.postgres.yml
```

The complete path to the new properties file is now:

```
~/privacera/privacera-manager/config/custom-vars/connectors/postgres/postgres-prod-in
```

5. Edit the new properties file (in this example, `vars.connector.postgres.yml`) to bring forward the values for the new property names.
   You can compare the property values in your backup files to the new property names and cut-and-paste the old values into the new property setting.
   The new property names have the string `CONNECTOR_` prepended to the old property name. For example:
   - **Old property name**: `POSTGRES_JDBC_URL`
   - **New property name**: `CONNECTOR_POSTGRES_JDBC_URL`

   Repeat bringing forward the desired values for all connectors you want to configure.
6. Enable PolicySync for this connector instance by setting the following property:

```
CONNECTOR_POSTGRES_ENABLE: true
```

7. Save the YAML file.
8. Stop the previous PolicySync deployment.
   - On Kubernetes:

```
# Check existing running policysync deployment
kubectl get deployment -n <namespace>
# Scale down existing deployment
kubectl scale --replicas=0 deployment/policysyncv2 -n <namespace>
```

   - On Docker:

```
cd ~/privacera/docker
./privacera_services stop policysyncv2
```

9. Restart Privacera Manager to put the new configuration into effect:

```
cd ~/privacera/privacera-manager
./privacera-manager.sh update
```

10. If you're running on Kubernetes, after successfully updating Privacera Manager, delete the existing deployment:

```
kubectl delete deploy policysyncv2 -n <namespace>
```

Databricks SQL connector for PolicySync on Privacera Platform

This topic shows how to configure access control for Databricks SQL.

Generalized approach for implementing PolicySync

To help you reach compliance, Privacera PolicySync distributes your defined access management policies to the third-party datasources you connect to Privacera.

Use this generalized approach for implementing PolicySync.

1. Understand how PolicySync works and how it is configured. See PolicySync design and configuration on Privacera Platform [61].
2. Decide which PolicySync topology best suits your needs:
   - Required basic PolicySync topology: always at least one connector instance [64]
   - Optional topology: multiple connector instances for Kubernetes pods and Docker containers [65]
   - Recommended PolicySync topology: individual policy repositories for individual connectors [65]
3. Create the required, basic PolicySync configuration. See Required basic PolicySync topology: always at least one connector instance [64]
4. Examine the BASIC and ADVANCED properties, decide which features you want to implement, and set the necessary values in the YAML property file.

Connector name: databricks-sql-analytics

When you create the connector as detailed in PolicySync design and configuration on Privacera Platform [61], use the following reserved word for the name of the connector:

```
databricks-sql-analytics
```

In formal syntax shown in Connector instance directory/file structure [63] replace `<ConnectorName>` with the above and in the example in Required basic PolicySync topology: always at least one connector instance [64], replace `postgres` with the above.

Prerequisites

Ensure the following prerequisites are met:

- Create an SQL warehouse in Databricks SQL with a user having admin privileges. For more information, see Databricks documentation.
- As you configure the warehouse make a note of the following values, which you to specify in the PolicySync properties for Databricks SQL :
  - Host URL
  - JDBC URL
  - SQL endpoint token
  - Database List

Databricks SQL connector properties for PolicySync on Privacera Platform

These Databricks SQL connector properties can be set for PolicySync in Privacera Platform.

The properties are grouped by general function, such as JDBC connection properties, properties for user, group, and role management, and other functions.

The properties are also categorized as **BASIC** or **ADVANCED**:

- **BASIC** pertains to the most fundamental aspects of the connector, such as authentication.
- **ADVANCED** indicates additional features beyond the **BASIC**s, such as row-filtering or group member handling.

Start by setting the **BASIC** fields described here and then examine the **ADVANCED** fields to determine which of these features you might want to enable.

For a general process to migrate values from old YAML files to the new YAML files, see Migration to PolicySync v2 on Privacera Platform 7.2 [68].

Category	Property name	Description	Default	Allowable values
JDBC configuration properties				
BASIC	CONNECTOR_DATABRICKS_SQL_ANALYTICS_JDBC_URL	This property is used to set jdbc jdbc url which can be used to connect to Databricks sql endpoint. JDBC URL should follow below convention `jdbc:spark://<WORKSPACE_URL>:443/<DATABASE>;transportMode=http;ssl=1;AuthMech=3;httpPath=/sql/1.0/endpoints/1234567890` **Example:** `jdbc:spark://example.cloud.databricks.com:443/default;transportMode=http;ssl=1;AuthMech=3;httpPath=/sql/1.0/endpoints/1234567890`		
BASIC	CONNECTOR_DATABRICKS_SQL_ANALYTICS_JDBC_USERNAME	This property is used to set jdbc Username to be used to make connection to Databricks sql endpoint. This is just an email used to log in to Databricks to manage the SQL endpoint. (ex. eric.yuan@privacera.com)		
BASIC	CONNECTOR_DATABRICKS_SQL_ANALYTICS_JDBC_PASSWORD	This is a personal access token used to access the Databricks sql endpoint, that is created in Databricks by going to SQL endpoints and then "Create a personal access token". It should look like this: `dapi71f8493d87bce9847d09e17a10ba8d53`		
BASIC	CONNECTOR_DATABRICKS_SQL_ANALYTICS_JDBC_DB	This property is used to set jdbc database to be used to make initial connection to Databricks sql endpoint.		
BASIC	CONNECTOR_DATABRICKS_SQL_ANALYTICS_OWNER_ROLE	This property is used to set ownership for all the resources managed by PolicySync. The specified user will become owner for all managed resources and will have full control on those resources.We support changing owners of database, tables and views. **NOTE** If owner role is kept as blank, then ownership will not change and users who creates tables/views or any other object will be the owner of those objects and PolicySync won't be able to do access control on that object		
BASIC	CONNECTOR_DATABRICKS_SQL_ANALYTICS_HOST_URL	This property is used to make a call to SQL analytics API for users/groups/audits. It should simply be the base url of Databricks, for example `https://db1.cloud.databricks.com/`		
BASIC	CONNECTOR_DATABRICKS_SQL_ANALYTICS_DEFAULT_USER_PASSWORD	This property is used to set password which will be used for every new user creation by PolicySync.		
Resources management				

Category	Property name	Description	Default	Allowable values
BASIC	`CONNECTOR_DATABRICKS_SQL_ANALYTICS_MANAGE_DATABASE_LIST`	This property is used to set comma separated database names which access control should be managed by PolicySync. If you want to manage all databases then you can skip specifying this property. This supports wildcards as well. The ignore database list has precedence over manage database list. Eg. `testdb1,testdb2,sales_db*`. **NOTE** Values for this property are case-sensitive.		
ADVANCED	`CONNECTOR_DATABRICKS_SQL_ANALYTICS_MANAGE_TABLE_LIST`	This property is used to set comma separated table/view Fqdn which access control should be managed by PolicySync. If you want to manage all tables/views then you can skip specifying this property. This supports wildcards as well. The ignore table list has precedence over manage table list. Example: `testdb1.schema1.table1,testdb2.schema2.view2,sales_db*.sales*.*`. **NOTE** Values for this property are case-sensitive.		
ADVANCED	`CONNECTOR_DATABRICKS_SQL_ANALYTICS_IGNORE_DATABASE_LIST`	This property is used to set comma separated database names which access control you don't want to be managed by PolicySync. If you don't want to ignore any database then you can skip specifying this property. This supports wildcards as well. This has precedence over manage database list. Example: `testdb1,testdb2,sales_db*`. **NOTE** Values for this property are case-sensitive.		

Category	Property name	Description	Default	Allowable values
ADVANCED	CONNECTOR_DATABRICKS_SQL_ANALYTICS_IGNORE_TABLE_LIST	This property is used to set comma separated table/view Fqdn which access control you don't want to be managed by PolicySync. If you don't want to ignore any tables/views then you can skip specifying this property. This supports wildcards as well. This has precedence over manage table list. Example: `testdb1.schema1.table1,testdb2.schema2.view2,sales_db*.sales*.*`. **NOTE** Values for this property are case-sensitive.		
Users/Groups/Roles management				
ADVANCED	CONNECTOR_DATABRICKS_SQL_ANALYTICS_NAME_REPLACE_FROM_REGEX	This takes the regular expression as input and finds the matching characters in a user name and replaces them with the characters specified in property. If kept blank, no find and replace operation is performed.	`[~`$&+:;=?@#|'<>.^*()_%\\\\[\\\\]!\\\\\-\\\\/\\\\\\\\{}]`	
ADVANCED	CONNECTOR_DATABRICKS_SQL_ANALYTICS_NAME_REPLACE_TO_STRING	The value specified in this property is used to replace the characters found by the regex specified user name regex property. If kept blank, no find and replace operation is performed.	_	
ADVANCED	CONNECTOR_DATABRICKS_SQL_ANALYTICS_USER_NAME_REPLACE_FROM_REGEX	This takes the regular expression as input and finds the matching characters in a user name and replaces them with the characters specified in property. If kept blank, no find and replace operation is performed.	`[~`$&+:;=?@#|'<>.^*()_%\\\\[\\\\]!\\\\\-\\\\/\\\\\\\\{}]`	
ADVANCED	CONNECTOR_DATABRICKS_SQL_ANALYTICS_USER_NAME_REPLACE_TO_STRING	The value specified in this property is used to replace the characters found by the regex specified user name regex property. If kept blank, no find and replace operation is performed.	_	
ADVANCED	CONNECTOR_DATABRICKS_SQL_ANALYTICS_GROUP_NAME_REPLACE_FROM_REGEX	This takes the regular expression as input and finds the matching characters in a group name and replaces them with the characters specified in property. If kept blank, no find and replace operation is performed.	`[~`$&+:;=?@#|'<>.^*()_%\\\\[\\\\]!\\\\\-\\\\/\\\\\\\\{}]`	
ADVANCED	CONNECTOR_DATABRICKS_SQL_ANALYTICS_GROUP_NAME_REPLACE_TO_STRING	The value specified in this property is used to replace the characters found by the regex specified group name regex property. If kept blank, no find and replace operation is performed.	_	
ADVANCED	CONNECTOR_DATABRICKS_SQL_ANALYTICS_ROLE_NAME_REPLACE_FROM_REGEX	This takes the regular expression as input and finds the matching characters in a role name and replaces them with the characters specified in property. If kept blank, no find and replace operation is performed.	`[~`$&+:;=?@#|'<>.^*()_%\\\\[\\\\]!\\\\\-\\\\/\\\\\\\\{}]`	
ADVANCED	CONNECTOR_DATABRICKS_SQL_ANALYTICS_ROLE_NAME_REPLACE_TO_STRING	The value specified in this property is used to replace the characters found by the regex specified role name regex property. If kept blank, no find and replace operation is performed.	_	

Category	Property name	Description	Default	Allowable values
ADVANCED	CONNECTOR_DATABRICKS_SQL_ANALYTICS_USER_NAME_PERSIST_CASE_SENSITIVITY	After loading user from Ranger API's all are converted into lowercase, but in some cases, you would need to have the users in the same case as they are in Ranger. When setting this value to true, it will maintain the case sensitivity of names as they are in Ranger.	false	
ADVANCED	CONNECTOR_DATABRICKS_SQL_ANALYTICS_GROUP_NAME_PERSIST_CASE_SENSITIVITY	After loading group from Ranger API's all are converted into lowercase, but in some cases, you would need to have the users in the same case as they are in Ranger. When setting this value to true, it will maintain the case sensitivity of names as they are in Ranger.	false	
ADVANCED	CONNECTOR_DATABRICKS_SQL_ANALYTICS_ROLE_NAME_PERSIST_CASE_SENSITIVITY	After loading role from Ranger API's all are converted into lowercase, but in some cases, you would need to have the users in the same case as they are in Ranger. When setting this value to true, it will maintain the case sensitivity of names as they are in Ranger.	false	
ADVANCED	CONNECTOR_DATABRICKS_SQL_ANALYTICS_CREATE_USER	This property controls whether we should create user in Databricks sql endpoint for users fetched from ranger.	true	
ADVANCED	CONNECTOR_DATABRICKS_SQL_ANALYTICS_MANAGE_USERS	This property controls whether we should create role in Databricks sql endpoint for users fetched from ranger.	true	
ADVANCED	CONNECTOR_DATABRICKS_SQL_ANALYTICS_DELETE_USERS	When this property is set to true, PolicySync will delete users in Databricks when they are deleted in Portal. The property is set to false by default, as deleting users in Databricks wipes out their access tokens and other info.	false	
ADVANCED	CONNECTOR_DATABRICKS_SQL_ANALYTICS_MANAGE_GROUPS	This property controls whether we should create role in Databricks sql endpoint for groups fetched from ranger.	true	
ADVANCED	CONNECTOR_DATABRICKS_SQL_ANALYTICS_MANAGE_ROLES	This property controls whether we should create role in Databricks sql endpoint for roles fetched from ranger.	true	
ADVANCED	CONNECTOR_DATABRICKS_SQL_ANALYTICS_MANAGE_USER_LIST	This property is used to set comma separated user names which access control should be managed by PolicySync. If you want to manage all users then you can skip specifying this property. This supports wildcards as well. The ignore users list has precedence over manage users list. Eg. `user1,user2,dev_user*`		
ADVANCED	CONNECTOR_DATABRICKS_SQL_ANALYTICS_MANAGE_GROUP_LIST	This property is used to set comma separated group names which access control should be managed by PolicySync. If you want to manage all group then you can skip specifying this property. This supports wildcards as well. The ignore group list has precedence over manage group list. Eg. `group1,group2,dev_group*`		
ADVANCED	CONNECTOR_DATABRICKS_SQL_ANALYTICS_MANAGE_ROLE_LIST	This property is used to set comma separated role names which access control should be managed by PolicySync. If you want to manage all role then you can skip specifying this property. This supports wildcards as well. The ignore role list has precedence over manage role list. Eg. `role1,role2,dev_role*`		

Category	Property name	Description	Default	Allowable values
ADVANCED	CONNECTOR_DATABRICKS_SQL_ANALYTICS_IGNORE_USER_LIST	This property is used to set comma separated user names which access control you don't want to be managed by PolicySync. If you don't want to ignore any users then you can skip specifying this property. This supports wildcards as well. This has precedence over manage users list. Eg. `user1,user2,dev_user*`		
ADVANCED	CONNECTOR_DATABRICKS_SQL_ANALYTICS_IGNORE_GROUP_LIST	This property is used to set comma separated group names which access control you don't want to be managed by PolicySync. If you don't want to ignore any groups then you can skip specifying this property. This supports wildcards as well. This has precedence over manage groups list. Eg. `group1,group2,dev_group*`		
ADVANCED	CONNECTOR_DATABRICKS_SQL_ANALYTICS_IGNORE_ROLE_LIST	This property is used to set comma separated role names which access control you don't want to be managed by PolicySync. If you don't want to ignore any roles then you can skip specifying this property. This supports wildcards as well. This has precedence over manage roles list. Eg. `role1,role2,dev_role*`		
ADVANCED	CONNECTOR_DATABRICKS_SQL_ANALYTICS_GROUP_ROLE_PREFIX	This property is used to set a prefix for role which we will be creating in Databricks sql endpoint for group from ranger. For example if you have group named dev in ranger and you have defined prefix as test_group_ then the role which we create for dev in Databricks sql endpoint will have name test_group_dev. **NOTE** This property does not exist for users because users are using emails instead of usernames to log in.	`priv_group_`	
ADVANCED	CONNECTOR_DATABRICKS_SQL_ANALYTICS_ROLE_ROLE_PREFIX	This property is used to set a prefix for role which we will be creating in Databricks sql endpoint for role from ranger. For example if you have role named finance in ranger and you have defined prefix as test_role_ then the role which we create for finance in Databricks sql endpoint will have name test_role_finance. **NOTE** This property does not exist for users because users are using emails instead of usernames to log in.	`priv_role_`	
ADVANCED	CONNECTOR_DATABRICKS_SQL_ANALYTICS_USE_NATIVE_PUBLIC_GROUP	Set this property to true, if you want PolicySync to use the "public" group in Databricks for access grants whenever there is policy created referring to public group inside it.	`true`	

Category	Property name	Description	Default	Allowable values
ADVANCED	CONNECTOR_DATABRICKS_SQL_ANALYTICS_MANAGE_USER_FILTERBY_GROUP	Set this property to true, if you want to manage only the users who belongs to the groups defined in manage groups list property.	false	
ADVANCED	CONNECTOR_DATABRICKS_SQL_ANALYTICS_MANAGE_USER_FILTERBY_ROLE	Set this property to true, if you want to manage only the users who belongs to the roles defined in manage roles list property.	false	
ADVANCED	CONNECTOR_DATABRICKS_SQL_ANALYTICS_USER_USE_EMAIL_AS_SERVICE_NAME	This Property is used to map the username as email address while grant/revoke	true	
Access control management				
ADVANCED	CONNECTOR_DATABRICKS_SQL_ANALYTICS_ENABLE_VIEW_BASED_MASKING	Set this property to true, if you want to enable secure view based masking in Databricks PolicySync. **NOTE** Databricks does not support native masking, so it is recommended to use view based masking.	true	
ADVANCED	CONNECTOR_DATABRICKS_SQL_ANALYTICS_ENABLE_VIEW_BASED_ROW_FILTER	Set this property to true, if you want to enable secure view based tr filter in Databricks PolicySync. **NOTE** Databricks does not support native tr filters,so it is recommended to use view based tr filters.	true	
ADVANCED	CONNECTOR_DATABRICKS_SQL_ANALYTICS_SECURE_VIEW_CREATE_FOR_ALL	Set this property to true, if you want to create secure view for all tables as well all view which were created by end users. This will create secure view for resource regardless whether there any masking/tr filter policy exists in ranger.	true	
ADVANCED	CONNECTOR_DATABRICKS_SQL_ANALYTICS_MASKED_NUMBER_VALUE	This property is used to specify the default masking value for numeric columns	0	
ADVANCED	CONNECTOR_DATABRICKS_SQL_ANALYTICS_MASKED_TEXT_VALUE	This property is used to specify the default masking value for text/string columns	<MASKED>'	
ADVANCED	CONNECTOR_DATABRICKS_SQL_ANALYTICS_SECURE_VIEW_NAME_PREFIX	By default view-based tr filter and masking related secure views have the same name as the table name with postfixed by _secure. If you want to change the secure view name prefix and postfix, that can be done with these properties. After prefix and postfix is specified the view name will be in this format : {prefix}{table_name}{postfix}		

Category	Property name	Description	Default	Allowable values
ADVANCED	CONNECTOR_DATABRICKS_SQL_ANALYTICS_SECURE_VIEW_NAME_POSTFIX	By default view-based tr filter and masking related secure views have the same name as the table name with postfixed by _secure. If you want to change the secure view name prefix and postfix, that can be done with these properties. After prefix and postfix is specified the view name will be in this format : {prefix}{table_name}{postfix}		
ADVANCED	CONNECTOR_DATABRICKS_SQL_ANALYTICS_SECURE_VIEW_DATABASE_NAME_PREFIX	By default view-based tr filter and masking related secure views have the same schema name as the table database name. If you want to change the secure view database name prefix and postfix, that can be done with these properties. After prefix and postfix is specified the view database name will be in this format : {prefix}{view_database_name}{postfix}		
ADVANCED	CONNECTOR_DATABRICKS_SQL_ANALYTICS_SECURE_VIEW_DATABASE_NAME_POSTFIX	By default view-based tr filter and masking related secure views have the same database name as the table database name. If you want to change the secure view database name prefix and postfix, that can be done with these properties. After prefix and postfix is specified the view database name will be in this format : {prefix}{view_database_name}{postfix}	_secure	
BASIC	CONNECTOR_DATABRICKS_SQL_ANALYTICS_GRANT_UPDATES	This property controls whether actual grant/revoke and create/update/delete queries for user/group/role should be run on Databricks sql endpoint.	true	
ADVANCED	CONNECTOR_DATABRICKS_SQL_ANALYTICS_ENABLE_DATA_ADMIN	This property is used to enable data admin feature, with data admin feature enabled you can create all the policies on table/native view and by default respective grants will be made on secure view of table table or native view. And this secure view will have tr filter and masking capability as well. In case if you need permission on table then you can select the permission you want plus dataadmin in the policy, In this case that permissions will be granted on both, the table/native view and its secure view as well	true	
Access audits management				
BASIC	CONNECTOR_DATABRICKS_SQL_ANALYTICS_AUDIT_ENABLE	This property is used to enable access audit fetching from Databricks sql endpoint	true	
ADVANCED	CONNECTOR_DATABRICKS_SQL_ANALYTICS_AUDIT_EXCLUDED_USERS	This property is used to exclude the users while pushing the audits logs to ranger access audits. Recommended to set this as JDBC user name as there will be audits from PolicySync application.	{{DATABRICKS_SQL_ANALYTICS_JDBC_USERNAME}}	

Dremio connector for PolicySync on Privacera Platform

This topic shows how to configure access control for Dremio

Generalized approach for implementing PolicySync

To help you reach compliance, Privacera PolicySync distributes your defined access management policies to the third-party datasources you connect to Privacera.

Use this generalized approach for implementing PolicySync.

1. Understand how PolicySync works and how it is configured. See PolicySync design and configuration on Privacera Platform [61].
2. Decide which PolicySync topology best suits your needs:
   - Required basic PolicySync topology: always at least one connector instance [64]
   - Optional topology: multiple connector instances for Kubernetes pods and Docker containers [65]
   - Recommended PolicySync topology: individual policy repositories for individual connectors [65]
3. Create the required, basic PolicySync configuration. See Required basic PolicySync topology: always at least one connector instance [64]
4. Examine the BASIC and ADVANCED properties, decide which features you want to implement, and set the necessary values in the YAML property file.

Connector name: dremio

When you create the connector as detailed in PolicySync design and configuration on Privacera Platform [61], use the following reserved word for the name of the connector:

```
dremio
```

In formal syntax shown in Connector instance directory/file structure [63] replace `<ConnectorName>` with the above and in the example in Required basic PolicySync topology: always at least one connector instance [64], replace `postgres` with the above.

Dremio connector properties for PolicySync on Privacera Platform

These Dremio connector properties can be set for PolicySync in Privacera Platform.

The properties are grouped by general function, such as JDBC connection properties, properties for user, group, and role management, and other functions.

The properties are also categorized as **BASIC** or **ADVANCED**:

- **BASIC** pertains to the most fundamental aspects of the connector, such as authentication.
- **ADVANCED** indicates additional features beyond the **BASIC**s, such as row-filtering or group member handling.

Start by setting the **BASIC** properties and then examine the **ADVANCED** properties to determine which of these features you might want to enable.

For a general process to migrate values from old YAML files to the new YAML files, see Migration to PolicySync v2 on Privacera Platform 7.2 [68].

Category	Property name	Description	Default
JDBC configuration properties			
BASIC	`CONNECTOR_DREMIO_JDBC_URL`	This property is used to set JDBC url which can be used to create a direct connection to a Dremio coordinator node. JDBC URL should follow below convention `jdbc:dremio:direct=<DREMIO_COORDINATOR>:31010` Example :- `jdbc:dremio:direct=example-12345fg.us-east-1.elb.amazonaws.com:31010`	
BASIC	`CONNECTOR_DREMIO_JDBC_USERNAME`	This property is used to set JDBC Username to be used to make connection to Dremio coordinator. This is just an username used to log in to Dremio to manage the Authorization. (ex. adminUser)	
BASIC	`CONNECTOR_DREMIO_JDBC_PASSWORD`	This property is used to set JDBC user's password to be used to make connection to Dremio coordinator.	

Category	Property name	Description	Default
BASIC	CONNECTOR_DREMIO_OWNER_ROLE	This property is used to set ownership for all the resources managed by policysync. The specified user will become owner for all managed resources and will have full control on those resources. We support changing owners of source, space, source folders, space folders, physical datasets and virtual datasets. Note :- If owner role is kept as blank, then ownership will not change and users who creates resources or any other object will be the owner of those objects and policysync won't be able to do access control on that objects.	
BASIC	CONNECTOR_DREMIO_URL	This property is used to make a call to Dremio API for catalogs/users/groups/audits. For example `https://internal-dummy.us-east-1.elb.amazonaws.com:9047`	
BASIC	CONNECTOR_DREMIO_DEFAULT_USER_PASSWORD	This property is used to set password which will be used as default password for every new user created by policysync.	
Resources management			
BASIC	CONNECTOR_DREMIO_MANAGE_SPACE_LIST	This property is used to set comma separated space names for which access control should be managed by policysync. If you want to manage all spaces then you can skip specifying this property. This supports wildcards as well. The ignore space list has precedence over manage space list. Eg. `testspace1,testspace2,space*` Note :- values for this property are case-sensitive.	
BASIC	CONNECTOR_DREMIO_MANAGE_SOURCE_LIST	This property is used to set comma separated source names which access control should be managed by policysync. If you want to manage all sources then you can skip specifying this property. This supports wildcards as well. The ignore source list has precedence over manage source list. Eg. `testsource1,testsource2,source*` Note :- values for this property are case-sensitive.	
ADVANCED	CONNECTOR_DREMIO_MANAGE_SOURCE_FOLDER_LIST	This property is used to set comma separated source folder for which access control should be managed by policysync. If you want to manage all source folder then you can skip specifying this property. This supports wildcards as well. The ignore source folder list has precedence over manage source folder list. Eg. `source.sourcefolder,source*.sales*` Note :- values for this property are case-sensitive.	
ADVANCED	CONNECTOR_DREMIO_MANAGE_SPACE_FOLDER_LIST	This property is used to set comma separated space folder for which access control should be managed by policysync. If you want to manage all space folders then you can skip specifying this property. This supports wildcards as well. The ignore space folder list has precedence over manage space folder list. Eg. `space.spacefolder,space*.sales*` Note :- values for this property are case-sensitive.	
ADVANCED	CONNECTOR_DREMIO_MANAGE_PHYSICAL_DATASET_LIST	This property is used to set comma separated physical datasets for which access control should be managed by policysync. If you want to manage all physical datasets then you can skip specifying this property. This supports wildcards as well. The ignore physical dataset list has precedence over manage physical dataset list. Eg. `source.folder1.physicaldataset,source*.sales*.*` Note :- values for this property are case-sensitive.	

Category	Property name	Description	Default
ADVANCED	CONNECTOR_DREMIO_MANAGE_VIRTUAL_DATASET_LIST	This property is used to set comma separated virtual datasets for which access control should be managed by policysync. If you want to manage all virtual datasets then you can skip specifying this property. This supports wildcards as well. The ignore virtual dataset list has precedence over manage virtual dataset list. Eg. `space.folder1.virtualdataset,space*.sales*.*` Note :- values for this property are case-sensitive.	
ADVANCED	CONNECTOR_DREMIO_IGNORE_SOURCE_LIST	This property is used to set comma separated source names for which you don't want access control to be managed by policysync. If you don't want to ignore any source then you can skip specifying this property. This supports wildcards as well. This has precedence over manage source list. Eg. `testsource,sales*` Note :- values for this property are case-sensitive.	
ADVANCED	CONNECTOR_DREMIO_IGNORE_SPACE_LIST	This property is used to set comma separated space names for which you don't want access control to be managed by policysync. If you don't want to ignore any space then you can skip specifying this property. This supports wildcards as well. This has precedence over manage space list. Eg. `testspace,sales*` Note :- values for this property are case-sensitive.	
ADVANCED	CONNECTOR_DREMIO_IGNORE_SOURCE_FOLDER_LIST	This property is used to set comma separated source folder names for which you don't want access control to be managed by policysync. If you don't want to ignore any source folder then you can skip specifying this property. This supports wildcards as well. This has precedence over manage source folder list. Eg. `source.sourcefolder,source*.sales*` Note :- values for this property are case-sensitive.	
ADVANCED	CONNECTOR_DREMIO_IGNORE_SPACE_FOLDER_LIST	This property is used to set comma separated space folder names for which you don't want access control to be managed by policysync. If you don't want to ignore any space folder then you can skip specifying this property. This supports wildcards as well. This has precedence over manage space folder list. Eg. `space.spacefolder,space*.sales*` Note :- values for this property are case-sensitive.	
ADVANCED	CONNECTOR_DREMIO_IGNORE_PHYSICAL_DATASET_LIST	This property is used to set comma separated physical dataset names for which you don't want access control to be managed by policysync. If you don't want to ignore any physical dataset then you can skip specifying this property. This supports wildcards as well. This has precedence over manage physical dataset list. Eg. `source.folder1.physicaldataset,source*.sales*.*` Note :- values for this property are case-sensitive.	
ADVANCED	CONNECTOR_DREMIO_IGNORE_VIRTUAL_DATASET_LIST	This property is used to set comma separated virtual dataset names for which you don't want access control to be managed by policysync. If you don't want to ignore any virtual dataset then you can skip specifying this property. This supports wildcards as well. This has precedence over manage virtual dataset list. Eg. `source.folder1.virtualdataset,source*.sales*.*` Note :- values for this property are case-sensitive.	
Users/Groups/Roles management			

Category	Property name	Description	Default
ADVANCED	CONNECTOR_DREMIO_USER_NAME_REPLACE_FROM_REGEX	This takes the regular expression as input and finds the matching characters in a user name and replaces them with the characters specified in property. If kept blank, no find and replace operation is performed.	[~`$&+:;=?@#|'<>.\\\ \s^*()_%\\\\[\ \\\]!\\\\-\\\ \/\\\\\\\\{}]
ADVANCED	CONNECTOR_DREMIO_USER_NAME_REPLACE_TO_STRING	The value specified in this property is used to replace the characters found by the regex specified user name regex property. If kept blank, no find and replace operation is performed.	_
ADVANCED	CONNECTOR_DREMIO_GROUP_NAME_REPLACE_FROM_REGEX	This takes the regular expression as input and finds the matching characters in a group name and replaces them with the characters specified in property. If kept blank, no find and replace operation is performed.	[~`$&+:;=?@#|'<>.\\\ \s^*()_%\\\\[\ \\\]!\\\\-\\\ \/\\\\\\\\{}]
ADVANCED	CONNECTOR_DREMIO_GROUP_NAME_REPLACE_TO_STRING	The value specified in this property is used to replace the characters found by the regex specified group name regex property. If kept blank, no find and replace operation is performed.	_
ADVANCED	CONNECTOR_DREMIO_ROLE_NAME_REPLACE_FROM_REGEX	This takes the regular expression as input and finds the matching characters in a role name and replaces them with the characters specified in property. If kept blank, no find and replace operation is performed.	[~`$&+:;=?@#|'<>.\\\ \s^*()_%\\\\[\ \\\]!\\\\-\\\ \/\\\\\\\\{}]
ADVANCED	CONNECTOR_DREMIO_ROLE_NAME_REPLACE_TO_STRING	The value specified in this property is used to replace the characters found by the regex specified role name regex property. If kept blank, no find and replace operation is performed.	_
ADVANCED	CONNECTOR_DREMIO_USER_NAME_PERSIST_CASE_SENSITIVITY	After loading user from Ranger API's all users are converted into lowercase, but in some cases, you would need to have the users in the same case as they are in Ranger. When setting this value to true, it will maintain the case sensitivity of names as they are in Ranger.	false
ADVANCED	CONNECTOR_DREMIO_GROUP_NAME_PERSIST_CASE_SENSITIVITY	After loading group from Ranger API's all groups are converted into lowercase, but in some cases, you would need to have the groups in the same case as they are in Ranger. When setting this value to true, it will maintain the case sensitivity of names as they are in Ranger.	false
ADVANCED	CONNECTOR_DREMIO_ROLE_NAME_PERSIST_CASE_SENSITIVITY	After loading role from Ranger API's all roles are converted into lowercase, but in some cases, you would need to have the roles in the same case as they are in Ranger. When setting this value to true, it will maintain the case sensitivity of names as they are in Ranger.	false
	CONNECTOR_DREMIO_USER_NAME_CASE_CONVERSION	This property only applicable if USER NAME PERSIST CASE SENSITIVITY is set to false. Managed users name would be treated as lowercase by default. If the value is set to "upper" then the user name would be treated as uppercase. If the value is set to "none" then the user name case is preserved as present in ranger.	lower
ADVANCED	CONNECTOR_DREMIO_CREATE_USER	This property controls whether we should create user in dremio for users fetched from ranger.	true
ADVANCED	CONNECTOR_DREMIO_CREATE_USER_ROLE	This property controls whether we should create role over the end user in dremio for users fetched from ranger.	true
ADVANCED	CONNECTOR_DREMIO_MANAGE_USERS	This property controls whether we should create role in dremio for users fetched from ranger.	true
ADVANCED	CONNECTOR_DREMIO_MANAGE_GROUPS	This property controls whether we should create role in dremio for groups fetched from ranger.	true
CUSTOM	CONNECTOR_DREMIO_MANAGE_GROUP_MEMBERS		true
ADVANCED	CONNECTOR_DREMIO_MANAGE_ROLES	This property controls whether we should create role in dremio for roles fetched from ranger.	true

Category	Property name	Description	Default
CUSTOM	CONNECTOR_DREMIO_MANAGE_GROUP_MEMBERS		true
ADVANCED	CONNECTOR_DREMIO_MANAGE_USER_LIST	This property is used to set comma separated user names which access control should be managed by policysync. If you want to manage all users then you can skip specifying this property. This supports wildcards as well. The ignore users list has precedence over manage users list. Eg. user1,user2,dev_user*	
ADVANCED	CONNECTOR_DREMIO_MANAGE_GROUP_LIST	This property is used to set comma separated group names which access control should be managed by policysync. If you want to manage all group then you can skip specifying this property. This supports wildcards as well. The ignore group list has precedence over manage group list. Eg. group1,group2,dev_group*	
ADVANCED	CONNECTOR_DREMIO_MANAGE_ROLE_LIST	This property is used to set comma separated role names which access control should be managed by policysync. If you want to manage all role then you can skip specifying this property. This supports wildcards as well. The ignore role list has precedence over manage role list. Eg. role1,role2,dev_role*	
ADVANCED	CONNECTOR_DREMIO_IGNORE_USER_LIST	This property is used to set comma separated user names which access control you don't want to be managed by policysync. If you don't want to ignore any users then you can skip specifying this property. This supports wildcards as well. This has precedence over manage users list. Eg. user1,user2,dev_user*	
ADVANCED	CONNECTOR_DREMIO_IGNORE_GROUP_LIST	This property is used to set comma separated group names which access control you don't want to be managed by policysync. If you don't want to ignore any groups then you can skip specifying this property. This supports wildcards as well. This has precedence over manage groups list. Eg. group1,group2,dev_group*	
ADVANCED	CONNECTOR_DREMIO_IGNORE_ROLE_LIST	This property is used to set comma separated role names which access control you don't want to be managed by policysync. If you don't want to ignore any roles then you can skip specifying this property. This supports wildcards as well. This has precedence over manage roles list. Eg. role1,role2,dev_role*	
ADVANCED	CONNECTOR_DREMIO_USER_ROLE_PREFIX	This property is used to set a prefix for role which we will be creating in Dremio for user from ranger. For example if you have user named john in ranger and you have defined prefix as test_user_ then the role which we create for john in Dremio will have name test_user_john	priv_user_
ADVANCED	CONNECTOR_DREMIO_GROUP_ROLE_PREFIX	This property is used to set a prefix for role which we will be creating in Dremio for group from ranger. For example if you have group named dev in ranger and you have defined prefix as test_group_ then the role which we create for dev in Dremio will have name test_group_dev.	priv_group_
ADVANCED	CONNECTOR_DREMIO_ROLE_ROLE_PREFIX	This property is used to set a prefix for role which we will be creating in Dremio for role from ranger. For example if you have role named finance in ranger and you have defined prefix as test_role_ then the role which we create for finance in Dremio will have name test_role_finance.	priv_role_
ADVANCED	CONNECTOR_DREMIO_USE_NATIVE_PUBLIC_GROUP	Set this property to true, if you want policysync to use the "public" group from Dremio for access grants whenever there is policy created referring to public group inside it.	true

Category	Property name	Description	Default
ADVANCED	CONNECTOR_DREMIO_MANAGE_USER_FILTERBY_GROUP	Set this property to true, if you want to manage only the users who belongs the the groups defined in manage groups list property.	false
ADVANCED	CONNECTOR_DREMIO_MANAGE_USER_FILTERBY_ROLE	Set this property to true, if you want to manage only the users who belongs the the roles defined in manage roles list property.	false
Access control management			
ADVANCED	CONNECTOR_DREMIO_ENABLE_VIEW_BASED_MASKING	Set this property to true, if you want to enable secure view based masking in Dremio policysync.	false
ADVANCED	CONNECTOR_DREMIO_ENABLE_VIEW_BASED_ROW_FILTER	Set this property to true, if you want to enable secure view based tr filter in Dremio policysync.	false
ADVANCED	CONNECTOR_DREMIO_SECURE_VIEW_CREATE_FOR_ALL	Set this property to true, if you want to create secure view for all datasets which were created by end users. This will create secure view for datasets regardless whether there any masking/tr filter policy exists in ranger.	false
ADVANCED	CONNECTOR_DREMIO_ENABLE_ROW_FILTER	This property controls whether to enable native tr filter policy creation functionality in policysync.	true
ADVANCED	CONNECTOR_DREMIO_ENABLE_MASKING	This property controls whether to enable native masking policy creation functionality in policysync.	true
ADVANCED	CONNECTOR_DREMIO_MASKED_NUMBER_VALUE	This property is used to specify the default masking value for numeric columns	0
ADVANCED	CONNECTOR_DREMIO_MASKED_DOUBLE_VALUE	This property is used to specify the default masking value for double datatype columns	0
ADVANCED	CONNECTOR_DREMIO_MASKED_TEXT_VALUE	This property is used to specify the default masking value for text/string columns	<MASKED>'
ADVANCED	CONNECTOR_DREMIO_SECURE_SPACE_PREFIX	By default view-based tr filter and masking related secure views have the same space name as the table/view source/space name. If you want to change the secure view space name prefix and postfix, that can be done with these properties. After prefix and postfix is specified the view space name will be in this format : {prefix}{view_space_name}{postfix}	
ADVANCED	CONNECTOR_DREMIO_SECURE_SPACE_POSTFIX	By default view-based tr filter and masking related secure views have the same space name as the table/view source/space name. If you want to change the secure view space name prefix and postfix, that can be done with these properties. After prefix and postfix is specified the view space name will be in this format : {prefix}{view_space_name}{postfix}	_secure
BASIC	CONNECTOR_DREMIO_GRANT_UPDATES	This property controls whether actual grant/revoke and create/update/delete queries for user/group/role should be run on Dremio.	true
ADVANCED	CONNECTOR_DREMIO_ENABLE_DATA_ADMIN	This property is used to enable data admin feature, with data admin feature enabled you can create all the policies on table/native view and by default perspective grants will be maid on secure view of table table or native view. And this secure view will have tr filter and masking capability as well. In case if you need permission on table then you can select the permission you want plus DataAdmin in the policy, In this case that permissions will be granted on both, the table/native view and its secure view as well.	false
Access audits management			
BASIC	CONNECTOR_DREMIO_AUDIT_ENABLE	This property is used to enable access audit fetching from Dremio.	true
ADVANCED	CONNECTOR_DREMIO_AUDIT_EXCLUDED_USERS	This property is used to exclude the users while pushing the audits logs to ranger access audits. Recommended to set this as JDBC user name as there will be audits from policysync application.	{{DREMIO_JDBC_USERNAME}}

Configure AWS Lake Formation on Privacera Platform

Get started with AWS Lake Formation

NOTE
Privacera support for AWS Lake Formation is a public preview. All interested customers are encouraged to try it.

AWS Lake Formation is a fully managed service that makes it easy to build, secure, and manage data lakes. Lake Formation provides its own permissions model that augments the IAM permissions model. This centrally defined permissions model enables fine-grained access to data stored in data lakes through a simple grant or revoke mechanism, much like a relational database management system (RDMS). Lake Formation permissions are enforced using granular controls at the column, row, and cell-levels across AWS services, including Amazon Athena, Amazon EMR, and Amazon Redshift.

AWS Lake Formation makes it easier for you to build, secure, and manage data lakes. Lake Formation helps you define granular data access policies for the metadata and data through a grant/revoke permissions model.

Why Lake Formation with Privacera?

Privacera offers a data access governance platform that allows for secure data sharing across hybrid environments and cloud services. When Privacera is configured with a Lake Formation connector, it syncs policies from Lake Formation for databases in the AWS Glue data Catalog and can be used to enforce those policies on Databricks, Databricks SQL, Trino, Starburst, and Dremio. Similarly, these policies can be enforced on other data sources, such as Amazon Redshift, Amazon RDS for PostgreSQL, Amazon Aurora, Snowflake, and Amazon RDS. So, for access control policies, Lake Formation will be the source of truth, and whatever policies are defined in Lake Formation can be applied to or enforced on a variety of other data sources.

Using Lake Formation with Privacera allows you to gain access control over a wide range of cloud services and improves access control from Lake Formation.

Advantages of using Lake Formation with Privacera

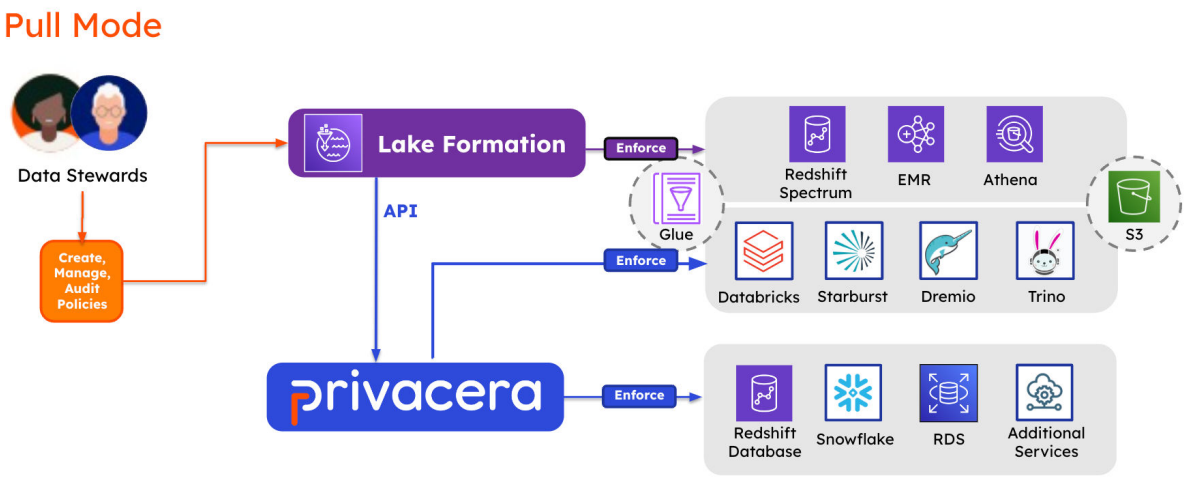
- Centralizes fine-grained data access control policies over a wide range of cloud services and various types of data sources.
- Provides centralized, detailed audit information about data access.
- Customizes reports and dashboards to support compliance, audit, and governance.

Connector configuration modes

The following two modes are available for configuring the Lake Formation connector with Privacera.

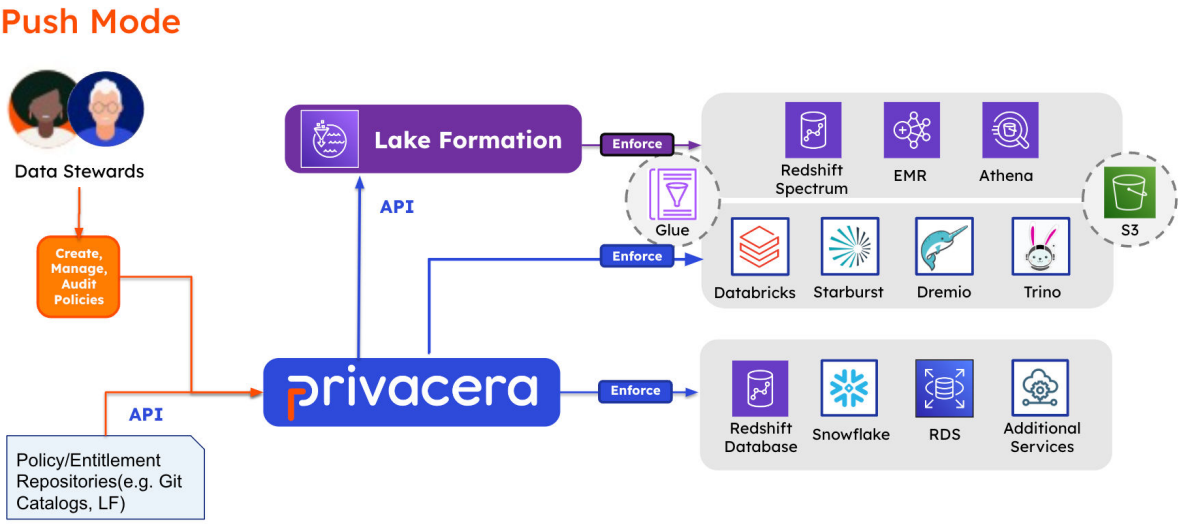
Pull mode

In this mode, the Lake Formation is the source of truth for access control. The access control policies are pulled from the Lake Formation at specific time intervals. From Privacera, and then these policies get enforced on various data sources defined by the configuration provided.



Push mode

In this mode, Privacera is the source of truth for access control policies. The access control policies are defined in the Privacera and then these policies will be pushed to AWS Lake Formation. From there, these policies will be enforced for Lake Formation-supported services like Amazon Redshift Spectrum, Amazon EMR, and Amazon Athena.



Create IAM Role for Lake Formation connector for Platform

This IAM Role creation is needed for PrivaceraCloud to pull the access control policies from Lake Formation into Privacera so that these policies can be enforced on various data sources.

Basically, this IAM role will have some set of permissions to read resources from AWS Glue and read access control policies on those resources from AWS Lake Formation.

When this role is created, it needs to be attached to PrivaceraCloud so that it will assume this role and get access to pull the access control policies from Lake Formation into Privacera.

Create IAM policy

1. Log in to the AWS Account.
2. Navigate to **IAM** → **Policies** → **Create policy**.
3. Select **JSON** and copy the following configuration in the text box:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "GlueReadPermissions",
            "Effect": "Allow",
            "Action": [
                "glue:GetTables",
                "glue:GetTableVersions",
                "glue:GetDatabases",
                "glue:GetTable",
                "glue:GetDatabase",
                "glue:GetTableVersion",
                "glue:GetColumnStatisticsForTable"
            ],
            "Resource": "*"
        },
        {
            "Sid": "LFReadPermissions",
            "Effect": "Allow",
            "Action": [
                "lakeformation:ListDataCellsFilter",
                "lakeformation:GetEffectivePermissionsForPath",
                "lakeformation:ListLFTags",
                "lakeformation:GetLFTag",
                "lakeformation:ListPermissions",
                "lakeformation:GetResourceLFTags",
                "lakeformation:DescribeResource",
                "lakeformation:ListResources",
                "lakeformation:GetTableObjects"
            ],
            "Resource": "*"
        },
        {
            "Sid": "IAMRolesReadPermissions",
            "Effect": "Allow",
            "Action": [
                "iam:ListRoles"
            ],
            "Resource": "*"
        }
    ]
}
```

4. Click **Next: Tags**;
5. Add any tags, then click **Next: Review**.
6. Add the policy name as `privacera-lf-access-policy`.
7. Click **Create policy**.

Create IAM Policy to Perform Grant/Revokes (Only for Push mode)

NOTE
This setup is only needed if you want to sync policies from Privacera Ranger to Lake Formation.

1. Login to AWS Account and navigate to **IAM** → **Policies** → **Create policy**.
2. Click **JSON** and paste below JSON content in the text box, and then click **Next: Tags**.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "LFWritePermission",
            "Effect": "Allow",
            "Action": [
                "lakeformation:BatchGrantPermissions",
                "lakeformation:GrantPermissions",
                "lakeformation:DeleteDataCellsFilter",
                "lakeformation:RevokePermissions",
                "lakeformation:CreateDataCellsFilter",
                "lakeformation:BatchRevokePermissions"
            ],
            "Resource": "*"
        }
    ]
}
```

3. Add any tags if you need, and then click **Next: Review**.
4. Add the policy name like `privacera-lf-write-access-policy`, and then click **Create policy**.

Create and attach IAM Role for Platform

1. Log in to AWS Account.
2. Navigate to **IAM** → **Roles** → **Create role**.
3. In **Select trusted entity**:
   - **Trusted entity type** select **AWS Service**
   - **Common use cases** select use case as **EC2**
4. Click **Next**.
5. **Add permissions**: In Permission policies, search for the policy `privacera-lf-access-policy` and select the policy checkbox. Click **Next**.
6. For the **Role name** add name as `PrivaceraLakeformationAccessRole` and click **Create role**.
7. For Docker based installation: *(Kubernates installation instructions currently not available)*
   1. Navigate to **EC2** → **Instances**.
   2. Search for your EC2 Instance and select the checkbox next to it.
   3. From the right corner, click on **Actions** → **Security** → **Modify IAM role** and select the IAM Role previously created `PrivaceraLakeformationAccessRole`. Then click **Update IAM role**.

Configure Lake Formation administrator

1. Log in to AWS Account and navigate to **AWS Lake Formation → Permissions → Administrative roles and tasks**.
2. Click on **Choose administrators** and for the **IAM users and roles** select the previously created role `PrivaceraLakeformationAccessRole` .
3. Click **Save**.

Configure Lake Formation connector on Privacera Platform

This topic describes how to enable the AWS Lake Formation connector with Privacera Platform. There are two ways to connect to the Lake Formation application: push mode and pull mode.

Implement PolicySync

Overview of steps of PolicySync Setup

1. Understand how PolicySync works and how it is configured. See PolicySync design and configuration on Privacera Platform [61] to learn more.
2. Decide which PolicySync topology best suits your needs:
   - Required basic PolicySync topology: [64]always at least one connector instance
   - Optional topology [65]: multiple connector instances for Kubernetes pods and Docker containers
   - Recommended PolicySync topology [65]: individual policy repositories for individual connectors
3. Create the PolicySync configuration of at least one connector.
4. Review the **BASIC** and **ADVANCED** properties [101] and decide which features you want to implement. Then set the values in the YAML property file.

NOTE

When you create the connector, use `lakeformation` as the connector name. This is a reserved term.

In formal syntax (shown in the Connector instance directory/file structure [63]) replace `<ConnectorName>` with `lakeformation` .

Configure Lake Formation connector using Push mode

Prerequisites

- The AWS Account ID.
- IAM Role. For more information, see Create IAM Role for Lake Formation connector.

Configure connector

1. SSH to the instance where Privacera is installed.
2. Navigate to the */config* directory.

   ```
   cd ~/privacera/privacera-manager/config
   ```
3. Create a new directory with this command:

   ```
   mkdir -p custom-vars/connectors/lakeformation/instance11
   ```
4. Copy the sample vars with the following command:

   ```
   cp sample-vars/vars.connector.lakeformation.yml custom-vars/connectors/lakeformation/
   ```

5. Open the *.yml* file to be edited.

   ```
   vi custom-vars/connectors/lakeformation/instance1/vars.connector.lakeformation.yml
   ```

6. Modify the following properties:
   - `CONNECTOR_LAKEFORMATION_AWS_ACCOUNT_ID` - Enter the AWS Account ID of the account you will be running the lake formation connector.
   - `CONNECTOR_LAKEFORMATION_AWS_REGION` - Set AWS region to connect to your lake formation instance.
   - Lake formation permissions sink type - `CONNECTOR_LAKEFORMATION_PERMISSION_SINK_TYPE: "sink"`
   - Enable policy enforcement and user/group/role management - `CONNECTOR_LAKEFORMATION_GRANT_UPDATES: "true"`
   - Enable policy processing when policies are updated in Privacera Ranger - `CONNECTOR_LAKEFORMATION_PERMISSION_RANGER_POLICY_CHANGE_PROCESSOR_ENABLE: "true"`
   - Enforce lake formation native row filter - `CONNECTOR_LAKEFORMATION_ENABLE_ROW_FILTER: "true"`
   - Enable sync service policy flag - `SYNC_SERVICEPOLICY_ENABLE: "true"`

   NOTE
   The # (hash) symbol before any property indicates it is commented out. You need have to remove the # sign and space if you want to uncomment that property. Click here to see more on property details and description. [101]

7. Once the properties are configured, run the following commands to update your Privacera Manager platform instance:

   ```
   cd ~/privacera/privacera-manager
   ./privacera-manager.sh update
   ```

The following points are to be considered when synchronizing policies in Ranger for Lake Formation:

Synchronizing Ranger Column Exclude Policies to Lake formation Column Exclude Policies

When the ranger policies with exclude columns synchronized into Lake formation, it converts it into permissions with include column policies.

For example:

Consider a table with the columns `country`, `id`, `region`, `sales_amount`, `city`, and `name`. Ranger policy with SELECT permission on columns excluding `city` and `name` columns is converted into Lake Formation policy with `SELECT` permission on columns including `country`, `id`, `region`, and `sales_amount`.

Synchronizing Ranger Tag Policies to Lake formation Tag Policies

For the tag policy created in Ranger, it internally fetches the resource attached to that tag and applies the permissions on the actual resource in Lake Formation.

For example

You have attached the `PII` tag to a table resource called `sales_data`, and then you created a tag based policy with `SELECT` permission for user `emily` on the `PII` tag. This internally gets the resource (i.e., table resource `sales_data`) attached to the `PII` tag and applies `SELECT` permission to user `emily` on the actual table `sales_data` in the Lake Formation.

Multiple Row Filter Policy Items Enforcement Behavior difference in Lake Formation

When you add a row filter policy in Ranger with a row filter condition, then it creates a data filter with the same row filter condition inside the lake formation. If you add multiple row filter items with different row filter conditions inside the row filter policy in Ranger, it creates those many data filters inside the lake formation and applies permissions on top of that.

For example:

If you create a Row-Level Filter policy on the `sales_data` table with two row filter items as below:

- `SELECT` permission with row filter condition `country='US'` to user `emily`
- `SELECT` permission with row filter condition `id=4` to user `emily`

When the Lake Formation engine enforces these permissions, when the user `emily` queries the `sales_data` table from Athena, it gets the result that is the intersection of both data filters. That is, it only gets one row with `id=4` and `country='US'` in the result.

```
select * from sales_data;
```

```
4,'US','Mountain','Palmertown','Sarah','50771.9'
```

No Access control on IAM Groups

Lake Formation does not support IAM groups, but it does support AD groups, and you can add policies to AD groups.

When granting access to IAM Role, it assumes the IAM role is present on the AWS console

When you create a policy for a role in Ranger, you assume that the IAM role is present on the AWS console with the same name as the role name in Ranger and just try to assign permissions to that role. It doesn't create any role if it's not present in the Lake Formation.

Configure Lake Formation connector using pull mode

Prerequisites

- The AWS Account ID.
- Create IAM Role for Lake Formation connector
- In AWS, ensure that databases and tables are in AWS Glue managed by Lake Formation.
- To sync permissions for IAM Users/Groups, then these users/groups should be present in Privacera. Ideally, these are synchronized from AD/LDAP or Okta into Privacera, but can also be added manually in Privacera. If the users/groups are not in Privacera Portal, then these permissions will not be synchronized.

NOTE

For IAM Roles, Privacera will automatically sync the IAM Roles as Apache Ranger Roles into Privacera.

- Setup Tag Policy Repository for Lake Formation connector [92]

Configure connector

1. SSH to the instance where Privacera is installed.
2. Navigate to the */config* directory.

```
cd ~/privacera/privacera-manager/config
```

3. Create a new directory with this command:

   ```
   mkdir -p custom-vars/connectors/lakeformation/instance11
   ```
4. Copy the sample vars with the following command:

   ```
   cp sample-vars/vars.connector.lakeformation.yml custom-vars/connectors/lakeformation/
   ```
5. Open the *.yml* file to be edited.

   ```
   vi custom-vars/connectors/lakeformation/instance1/vars.connector.lakeformation.yml
   ```
6. Modify the following properties:
   - `CONNECTOR_LAKEFORMATION_AWS_ACCOUNT_ID` - Enter the AWS Account ID of the account you will be running the lake formation connector.
   - `CONNECTOR_LAKEFORMATION_AWS_REGION` - Set AWS region to connect to your lake formation instance.

   NOTE
   The # (hash) symbol before any property indicates it is commented out. You need have to remove the # sign and space if you want to uncomment that property. Click here to see more on property details and description. [101]
7. Once the properties are configured, run the following commands to update your Privacera Manager platform instance:

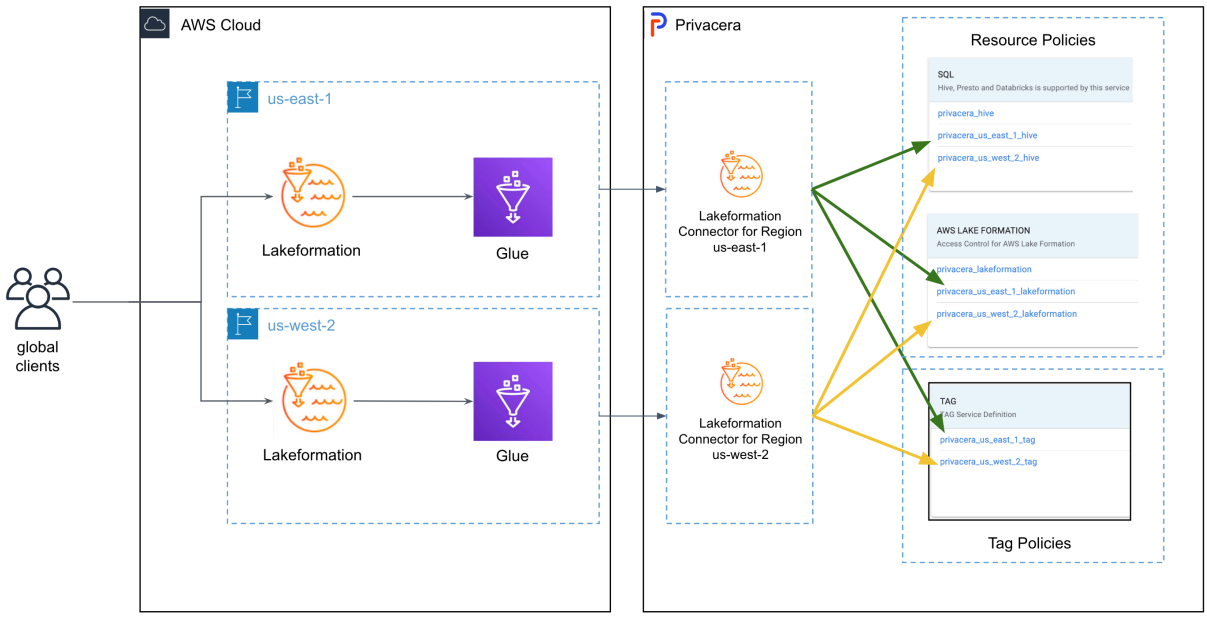
   ```
   cd ~/privacera/privacera-manager
   ./privacera-manager.sh update
   ```

Create Lake Formation connectors for multiple AWS regions for Platform

This topic describes how to setup Lake Formation connectors for multiple AWS regions.

Architecture

The below diagram shows the architecture of Lake Formation connectors with multiple AWS regions.



- **Left Panel**: AWS Cloud with 2 regions `us-east-1` and `us-west-2`.
- **Right Panel**: Two separate Lake Formation connectors configured within Privacera. Each connector is responsible for syncing policies from `us-east-1` and `us-west-2` region respectively.

- **Resource Policies** : This block contains the Hive and Lake Formation policy repositories within Privacera. We have `privacera_us_east_1_hive / privacera_us_east_1_lakeformation` and `privacera_us_west_2_hive / privacera_us_west_2_lakeformation` for `us-east-1` and `us-west-2` region policies.
- **Tag Policies**: This block contains the tag policy repositories within Privacera. We can have tag based Lake Formation policies inside policy repository `privacera_us_east_1_tag` and `privacera_us_west_2_tag` for `us-east-1` and `us-west-2` region.

Set up Lake Formation connectors with multiple regions for Platform

NOTICE

Follow the steps outlined in Configure Lake Formation connector on Privacera Platform [88] to setup the connectors for each region.

- Make sure to create different configuration folder for each region *custom-vars/connectors/lakeformation/*. For example `us-east-1` region connector *.yml* file can be inside *custom-vars/connectors/lakeformation/instance1* and the `us-west-2` region connector *.yml* file can be inside *custom-vars/connectors/lakeformation/instance2*.
- Set the `CONNECTOR_LAKEFORMATION_AWS_ACCOUNT_ID` and `CONNECTOR_LAKEFORMATION_AWS_REGION` variable values in the configuration file.
- Set the `CONNECTOR_LAKEFORMATION_SINK_HIVE_SERVICE_APP_ID` and `CONNECTOR_LAKEFORMATION_SINK_LAKEFORMATION_SERVICE_APP_ID` variable values with the policy repository names you have configured for each region.
- Create policy repositories for multiple AWS regions [92]

Create policy repositories for multiple AWS regions

Create a policy repository for tags

- Follow steps in Setup Tag Policy Repository for Lake Formation connector [92] to setup tag repositories for each AWS region. Here you can use `privacera_us_east_1_tag` and `privacera_us_west_2_tag` as a service name for `us-east-1` and `us-west-2` regions respectively.

Create a policy repository for Hive

- Follow steps in Setup access policy repository for Hive [94] to setup hive resource policy repositories for each AWS region. Here you can use `privacera_us_east_1_hive` and `privacera_us_west_2_hive` as a service name for `us-east-1` and `us-west-2` regions respectively.

Create a policy repository for Lake Formation

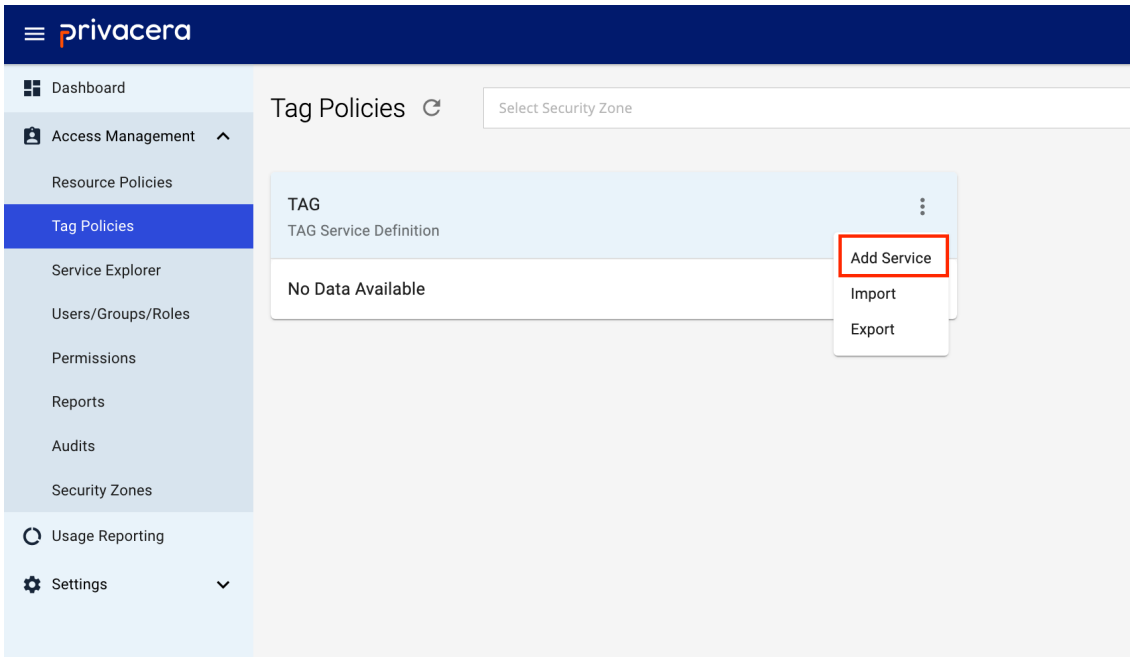
- Follow steps in Setup access policy repository for Lake Formation [94] to setup lake formation resource policy repositories for each AWS region. Here you can use `privacera_us_east_1_lakeformation` and `privacera_us_west_2_lakeformation` as a service name for `us-east-1` and `us-west-2` regions respectively.

Setup Tag Policy Repository for Lake Formation connector

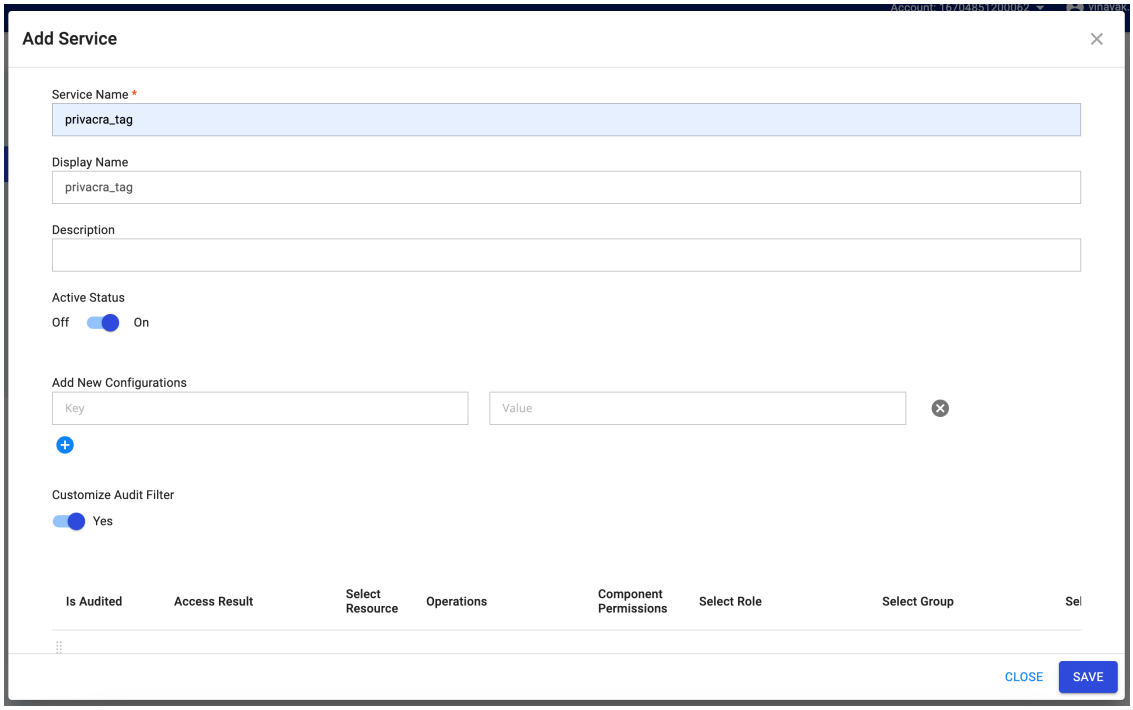
If a `privacera_tag` policy repository has already been created for Lake Formation, you do not need to follow the steps here.

1. Go to **Access Management** → **Tag Policies**.
2. On the **Tag Policies** tab, under the TAG window, if you are able to add `privacera_tag` as a repository, go to step 3 below. Else:

1. Click on three dots in the right corner and select the **Add Service** option.



2. Enter the **Service Name** as `privacera_tag` then click **SAVE**.



3. Go to **Resource Policies** click on edit button of `privacera_hive` and `privacera_lakeformation` , update the **Select Tag Service** with `privacera_tag`as the repository name.

Setup access policy repository for Lake Formation

1. Go to **Access Management** → **Resource Policies**.
2. On the **Resource Policies** screen, under the **AWS LAKE FORMATION** window, click on three dots in the right corner. Then select the **Add Service** option.
3. Enter the below details in the policy repository configuration:
   - **Service Name** - Enter `privacera_lakeformation` or the name of your choice.
   - **Username** - Enter `lakeformation` as the username in the configuration. This is required to create some default policies in the Lake Formation connector.
   - **Select Tag Service** - Enter `privacera_tag` as the Tag Service name or choose the name of the tag repository you have created.
4. Click the **SAVE** button.

Setup access policy repository for Hive

1. Go to **Access Management** → **Resource Policies**.
2. On the **Resource Policies** screen under the **SQL** window, click on 3 dots in the right corner then select the **Add Service** option.
3. Enter the below details in the policy repository configuration:

- **Service Name** - Enter `privacera_hive` or a name of your choice.
- **Username** - Enter `hive` as the Username in the configuration. This is required to create some default policies in the Lake Formation connector.
- **Password** - Enter the password of your choice.
- **jdbc.driverClassname** - Enter the Hive JDBC driver class name as `org.apache.hive.jdbc.HiveDriver`
- **JDBC URL** - Enter the Hive JDBC url as: `jdbc:hive2://localhost:10000`
- **Select Tag Service** - Enter `privacera_tag` as the service name or choose the name of the tag repository you have created.

4. Click **SAVE**.

Setup audit logs for Lake Formation on Platform

IMPORTANT

Prerequisite: Before performing the below steps you must have an S3 bucket for storing the audit logs.

Steps to enable audit logs in AWS Lake Formation:

1. Create Trail in AWS cloud trail.
    1. Go to AWS Cloud trail service → **Dashboard** → **Create Trail**.
    2. On the **General Details** page, provide the **Trail name** (such as `LF_Cloud_Trail`) of your choice.
    3. Provide the S3 storage location of your S3 bucket.
    Provide the S3 storage location of your S3 bucket.

4. Uncheck the **Log file SSE-KMS encryption** and **Log file validation** checkboxes. Click the **Next** button.
5. On next page under **Choose Log Events** enable the checkbox for **Management Events** with the checkboxes for Read and Write as checked.
6. Enable the checkbox for **Log Events**. In **Data Events** section, select the **Data Event type** as Lake formation and **Log Selector template** as Log All Events.
7. Click **Next**. Confirm the values and click on **Create Trail**.

2. Enable the audit log flag in Privacera Platform:
   Set the `CONNECTOR_LAKEFORMATION_AUDIT_ENABLE` flag to true.
3. Create a database and table for storing audit logs query results. For this, you can use AWS Athena.
   1. Create database:

      ```
      CREATE DATABASE lf_audit_db;
      ```

   2. Create table:
      (In the below query, the LOCATION is the S3 bucket location of the cloud trail logs.)

      ```
      CREATE EXTERNAL TABLE lf_audit_db.cloudtrail_logs (
          eventVersion STRING,
          userIdentity STRUCT<
              type: STRING,
              principalId: STRING,
              arn: STRING,
              accountId: STRING,
              invokedBy: STRING,
              accessKeyId: STRING,
              userName: STRING,
              sessionContext: STRUCT<
                  attributes: STRUCT<
                      mfaAuthenticated: STRING,
                      creationDate: STRING>,
                  sessionIssuer: STRUCT<
                      type: STRING,
                      principalId: STRING,
                      arn: STRING,
                      accountId: STRING,
                      userName: STRING>>>,
          eventTime STRING,
          eventSource STRING,
          eventName STRING,
          awsRegion STRING,
          sourceIpAddress STRING,
          userAgent STRING,
          errorCode STRING,
          errorMessage STRING,
          requestParameters STRING,
          responseElements STRING,
          additionalEventData STRING,
          requestId STRING,
          eventId STRING,
          resources ARRAY<STRUCT<
              arn: STRING,
              accountId: STRING,
              type: STRING>>,
          eventType STRING,
      ```

```
    apiVersion STRING,
    readOnly STRING,
    recipientAccountId STRING,
    serviceEventDetails STRING,
    sharedEventID STRING,
    vpcEndpointId STRING
)
COMMENT 'CloudTrail table for lakeformation audit logs'
ROW FORMAT SERDE 'com.amazon.emr.hive.serde.CloudTrailSerde'
STORED AS INPUTFORMAT 'com.amazon.emr.cloudtrail.CloudTrailInputFormat'
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION 's3://privacera-dev-bucket/AWSLogs/587946681758'
TBLPROPERTIES ('classification'='cloudtrail');
```

4. In the configuration file for the connector instance you created for Lake Formation (which is `~/privacera/privacera-manager//config/custom-vars/connectors/lakeformation/<your_instance_name>/vars.connector.lakeformation.yml`), set the following properties:

Property	Description	PrivaceraCloud custom property	Privacera Platform property
AWS Athena Region	Specifies AWS Athena region to create JDBC connection for Lake Formation audit logs database. If not specified, will default to use the first region from AWS Regions property.	`aws.athena.region=<athena_region_name>`	`CONNECTOR_LAKEFORMATION_AWS_ATHENA_REGION="<athena_region_name>"`
AWS Athena Endpoint	Specifies AWS Athena endpoint to create JDBC connection for Lake Formation audit logs database. If not specified, will create an endpoint with default region from the region property.	`aws.athena.endpoint=<athena_endpoint>`	`CONNECTOR_LAKEFORMATION_AWS_ATHENA_ENDPOINT="<athena_endpoint>"`
AWS Athena Workgroup	Specifies AWS Athena workgroup to create JDBC connection for Lake Formation audit logs database.	`aws.athena.workgroup=<athena_workgroup>`	`CONNECTOR_LAKEFORMATION_AWS_ATHENA_WORKGROUP="<athena_workgroup>"`
Audit Database Name	Specifies AWS audit database to store Lake Formation audit logs.	`audit.db.name=<audit_db_name>`	`CONNECTOR_LAKEFORMATION_AUDIT_DB_NAME="<audit_db_name>"`
Audit Table Name	Specifies AWS audit table to store Lake formation audit logs.	`audit.table.name=<audit_table_name>`	`CONNECTOR_LAKEFORMATION_AUDIT_TABLE_NAME="<audit_db_name>"`
AWS Athena S3 output location	Specifies S3 location to store the access audit logs query results.	`aws.athena.s3.output.location=<aws_athena_s3_output_location>`	`CONNECTOR_LAKEFORMATION_AUDIT_ATHENA_S3_OUTPUT_LOCATION="<aws_athena_s3_output_location>"`

5. There are two additional optional properties used to restrict collecting audit logs for excluding specific users and specific access types:

Property	Description	PrivaceraCloud custom property	Privacera Platform property
Audit Excluded Users	Specifies a list of users to exclude when fetching access audits.	`audit.excluded.users=<audit_excluded_users>`	`CONNECTOR_LAKEFORMATION_AUDIT_EXCLUDED_USERS="<audit_excluded_users>"`

Property	Description	PrivaceraCloud custom property	Privacera Platform property
Audit Excluded Access type	Specifies a list of list of access types to exclude when fetching access audits. For example: `StartQueryExecution`, `GetTable`,`DeleteTable`, `CreateTable`, `CreateDatabase`.	`audit.excluded.access.types=<audit_excluded_access_types>`	`CONNECTOR_LAKEFORMATION_AUDIT_EXCLUDED_ACCESS_TYPES="<aws_athena_s3_output_location>"`

How to validate a Lake Formation connector

By performing the below steps you can confirm that the configuration of your Lake Formation connector is valid.

- In PrivaceraCloud portal, go to **Access Management** → **Audit** → **Plugin**. There you should see your service name: `privacera_lakeformation` (or the service name you have chosen). If you don't see the connector plugin listed, then part of the configuration is incorrect, try rechecking your configuration.
- Click on the **Policy Sync** tab to see connector audit logs for this connector, this means your connector is functional. This shows policies/permissions in Lake Formation. If there are no logs for this connector then most likely your IAM role is incorrect or the cross-account trust was not configured properly.
- The Privacera Lake Formation connector will automatically pull the IAM Roles and add them to Apache Ranger. You can check this by going to **Access Management** → **Users/Groups/Roles** → **Roles**. If you don't see any IAM Roles in Apache Ranger, then most likely your IAM role is incorrect or the cross-account trust was not configured properly.
- If you already have policies in Lake Formation, then they will have synced in Privacera. Verify at **Access Management** → **Resource Policies** → `privacera_lakeformation` (or the name of your repo). In the **ACCESS** tab, you should see your policies. Here you can see the label for policies fetched from Lake Formation will be marked as `Connector: LakeFormation`. Also, you will see only the "Preview" option for the policy because by default these are read-only policies, not able to be edited or deleted from Privacera. This option can be changed by turning off the read-only flag in the Lake Formation connector configuration, but note this not recommended as the Policy creator should be only AWS Lake Formation.
  - If you click on the "Preview" button of a policy, it will show details for the resource. Such as the user/group/role and the permission associated with it.
- You can also check the **ROW LEVEL FILTER** tab for "Data filter" policies.
  - If you click on the "Preview" button of the policy you can see the row filter condition which is loaded for Lake formation "Data Filter".
  - For each Lake formation "Data filter" permission, it will create two policies.
    1. One policy in **ROW LEVEL FILTER** for specifying for which user and which resource has what type of access, and the filter condition.
    2. One in **Access Policy** for specifying the column level access. This shows what the user has access to and which columns of the table have `SELECT` access. Some columns can be excluded while providing access to other columns.
- **Access Management** → **Tag Policies** → `privacera_tag` (or the name of your tag policy repo). This shows your tag-based policies created in Privacera by reading from AWS Lake Formation.
  - Click on the "Preview" button to display the detail tag name, tag attributes, and policies attached to those tags.
- You can also check the Lake Formation Data location policies by going to **Access Management** → **Resource Policies** → `privacera_lakeformation` (or the name of your repo). In the **ACCESS** tab, you should see your policies prefixed with `data_location` and labeled with `connector: Lakeformation`.
  - Click on the "Preview" button to see the data location and the policy granted for which users/group/roles.

Lake Formation FAQs for Pull mode

Does the Privacera Lake Formation pull mode integration work with the following solutions?

- **Unity Catalog**- No, this solution has not yet been tested.
- **EMR Spark plugin**- Yes.
- **Databricks Spark plugin**- Yes.
- **Databricks SQL**- No, this solution has not yet been tested.

What are the prerequisites for setup?

- For PrivaceraCloud, see Connect Lake Formation application on PrivaceraCloud.
- For Privacera Platform, see Prerequisites [88].

Are Tag Policies supported?

- Yes, the tag policies are supported. Tags and Tag policies created in AWS Lake Formation will be pulled into Privacera in pull mode, and while pushing the same policies into the Hive repo, it transforms them into an access policy.

Can Lake Formation polices be modified?

- No, you can't modify policies through the Privacera Portal UI. The AWS Lake Formation will always be the source of truth, and those policies are imported into Ranger.

Can an audit log be enabled?

- Yes, audit logging can be enabled. To enable audit logs, you need to establish an S3 bucket and set up a cloud trail. For more information, see Configuring audit logs for Lake Formation on PrivaceraCloud. To enable this flag, do the following:
  PrivaceraCloud Steps:
  - Use the toggle to `Enable access audits`

  Privacera Platform steps:
  - Set the `CONNECTOR_LAKEFORMATION_AUDIT_ENABLE` flag to true.

NOTE
When enabling access audits for Lake formation connector, you must enable CloudTrail from AWS. For more information, see AWS guidelines for cost management with CloudTrail.

Is S3 data location supported?

- Yes.

Is Row Level Filtering supported?

- Yes.

Is Column Masking supported?

- No, column masking is not supported by the AWS Lake Formation.

Are “Include/exclude column policies” supported?

- Yes.

Are database resource link/table resource link supported?

- Yes.

What Privacera features are supported in the AWS Lake Formation pull mode?

- See the supported feature matrix in the table below.

TIP
Table abbreviations:

GA: General Availability

NA: Not Applicable

NS: Not Supported

MP: Mission Possible

Table 2. Privacera Features support Matrix for pull mode

Feature	Availability
Database Access Control	
Catalog Level	GA
Database Level	GA
Table Level	GA
View Level	GA
Native Column Level	GA
Other objects	
Data locations	GA
Database Resource links	GA
Table Resource links	GA
Cross account Resource links	GA
Tag	NA
Row Filter	
Native Row Filter on Table	GA
Native Row Filter on View	NS
Masking	
Native Masking on Table	NA
Native Masking on View	NA
Tag Based Access Control	
Allow Condition	GA
Tag Based Masking	
Allow Condition	NA
Attribute Based Access Control (ABAC)	
Allow Condition	NS
Audits	
Access Audits	GA
Principals	
IAM Users	GA
IAM Role	GA
SAML Users	GA
SAML Groups	GA
External Accounts	MP
Native Public Group	GA

Feature	Availability
Extented Privacera Plugin Support	
Hive Plugin	GA
Spark Plugin	GA
Databricks SQL Analytics with Glue Metastore	GA

Lake Formation Connector Properties

The properties described in this topic are necessary for setting up LakeFornation with PrivaceraCloud and Platform.

Table 3. AWS Account Configuration

PrivaceraCloud property field name	PM connector property name	Mandatory (to set by user)	Property variable name	PrivaceraCloud property type
AWS Account ID	`CONNECTOR_LAKEFORMATION_AWS_ACCOUNT_ID`	`TRUE`	`aws.account.id`	BASIC
AWS Assume IAM Role ARN	`CONNECTOR_LAKEFORMATION_AWS_ACCOUNT_ID`	`TRUE`	`aws.assume.iam.role.arn`	BASIC
AWS Assume IAM Role External ID	`CONNECTOR_LAKEFORMATION_AWS_ASSUME_IAM_ROLE_EXTERNAL_ID`	`FALSE`	`aws.assume.iam.role.external.id`	BASIC
AWS Access Key	`CONNECTOR_LAKEFORMATION_AWS_ACCESS_KEY`	`FALSE`	`aws.access.key`	CUSTOM
AWS Secret Key	`CONNECTOR_LAKEFORMATION_AWS_SECRET_KEY`	`FALSE`	`aws.secret.key`	CUSTOM
AWS Session Token	`CONNECTOR_LAKEFORMATION_AWS_SESSION_TOKEN`	`FALSE`	`aws.session.token`	CUSTOM
AWS Region	`CONNECTOR_LAKEFORMATION_AWS_REGION`	`TRUE`	`aws.region`	BASIC
SAML Provider ARN	`CONNECTOR_LAKEFORMATION_SAML_PROVIDER_ARN`	`TRUE`	`saml.provider.arn`	BASIC

Table 4. Load keys and intervals

PrivaceraCloud property field name	PM connector property name	Default value	Mandatory	Property variable name	PrivaceraCloud property type
Enable Resources Sync	`CONNECTOR_LAKEFORMATION_RESOURCE_SYNC_ENABLE`	TRUE	FALSE	`sync.resource.enable`	CUSTOM
Resource sync interval time in seconds	`CONNECTOR_LAKEFORMATION_RESOURCE_SYNC_INTERVAL`	60	FALSE	`sync.interval.sec`	CUSTOM
Enable Tagdefs Sync	`CONNECTOR_LAKEFORMATION_SYNC_TAGDEF_ENABLE`	TRUE	FALSE	`sync.tagdef.enable`	CUSTOM
Enable Tag Resource mapping Sync	`CONNECTOR_LAKEFORMATION_SYNC_RESOURCE_TAG_ENABLE`	TRUE	FALSE	`sync.resourcetag.enable`	CUSTOM
Enable Resource Permissions Sync	`CONNECTOR_LAKEFORMATION_SYNC_RESOURCE_POLICY_ENABLE`	TRUE	FALSE	`sync.resourcepolicy.enable`	CUSTOM
Enable Tag Permissions Sync	`CONNECTOR_LAKEFORMATION_SYNC_TAG_POLICY_ENABLE`	TRUE	FALSE	`sync.tagpolicy.enable`	CUSTOM
Enable AWS IAM Roles Sync	`CONNECTOR_LAKEFORMATION_SYNC_IAM_ROLE_ENABLE`	TRUE	FALSE	`sync.iam.role.enable`	CUSTOM

Privacera-Cloud property field name	PM connector property name	Default value	Mandatory	Property variable name	Privacera-Cloud property type
Enable Reconcile Resource Policies from Ranger	CONNECTOR_LAKEFORMATION_RECONCILE_RESOURCE_POLICY_ENABLE	TRUE	FALSE	reconcile.resource-policy.enable	CUSTOM
Enable Reconcile Tag Policies from Ranger	CONNECTOR_LAKEFORMATION_RECONCILE_TAG_POLICY_ENABLE	TRUE	FALSE	reconcile.tagpolicy.enable	CUSTOM
Tagdef sync interval time in seconds	CONNECTOR_LAKEFORMATION_TAGDEF_SYNC_INTERVAL	60	FALSE	tagdef.interval.sec	CUSTOM
Tag Resource mapping sync interval time in seconds	CONNECTOR_LAKEFORMATION_RESOURCE_TAG_SYNC_INTERVAL	60	FALSE	resourcetag.interval.sec	CUSTOM
Resource Permissions sync interval time in seconds	CONNECTOR_LAKEFORMATION_RESOURCE_POLICY_SYNC_INTERVAL	60	FALSE	resourcepolicy.interval.sec	CUSTOM
Tag Permissions sync interval time in seconds	CONNECTOR_LAKEFORMATION_TAG_POLICY_SYNC_INTERVAL	60	FALSE	tagpolicy.interval.sec	CUSTOM
AWS IAM Role sync interval time in seconds	CONNECTOR_LAKEFORMATION_IAM_ROLE_SYNC_INTERVAL	60	FALSE	iam.role.interval.sec	CUSTOM
Reconcile Resource Policies interval time in seconds	CONNECTOR_LAKEFORMATION_RECONCILE_RESOURCE_POLICY_INTERVAL	300	FALSE	reconcile.resource-policy.interval.sec	CUSTOM
Reconcile Tag Policies interval time in seconds	CONNECTOR_LAKEFORMATION_RECONCILE_TAG_POLICY_INTERVAL	300	FALSE	reconcile.tagpolicy.interval.sec	CUSTOM

Table 5. Resource Management

PrivaceraCloud property field name	PM Connector property name	Description	Mandatory	PrivaceraCloud property type
Catalogs to set access control policies	CONNECTOR_LAKEFORMATION_MANAGE_CATALOG_LIST	Specifies a comma-separated list of AWS Catalogs for which PolicySync manages access control. If unset, access control is managed for all catalogs. If specified, use the following format. You can use wildcards. Names are case-sensitive. Example list of catalogs: `123456789XXX, 987654321XXX, 1234*` If specified, Catalogs to ignore while setting access control policies takes precedence over this setting.	FALSE	BASIC

Privacera-aCloud property field name	PM Connector property name	Description	Mandatory	Privacera-aCloud property type
Data Locations to set access control policies	`CONNECTOR_LAKEFORMATION_MANAGE_DATA_LOCATION_LIST`	Specifies a comma-separated list of Data locations for which PolicySync manages access control. If unset, access control is managed for all data locations. If specified, use the following format. You can use wildcards. Names are case-sensitive. Example list of data locations: `123456789XXX.us-east-1.demo-s3-bucket/test_data*` If specified, Data locations to ignore while setting access control policies takes precedence over this setting.	FALSE	ADVANCED
Databases to set access control policies	`CONNECTOR_LAKEFORMATION_MANAGE_DATABASE_LIST`	Specifies a comma-separated list of database names for which PolicySync manages access control. If unset, access control is managed for all databases. If specified, use the following format. You can use wildcards. Names are case-sensitive. Syntax: `<CATALOG_ID>.<REGION>.<DATABASE>` Example list of databases: `123456789XXX.us-east-1.testdb1, 123456789XXX.us-east-1.testdb2, 123456789XXX.us-east-1.sales_db*` If specified, Databases to ignore while setting access control policies takes precedence over this setting.	FALSE	CUSTOM
Database Resource Links to set access control policies	`CONNECTOR_LAKEFORMATION_MANAGE_DATABASE_RESOURCE_LINK_LIST`	Specifies a comma-separated list of database resource links for which PolicySync manages access control. If unset, access control is managed for all database resource links. If specified, use the following format. You can use wildcards. Names are case-sensitive. Syntax: `<CATALOG_ID>.<REGION>.<DATABASE_RESOURCE_LINK>` Example list of database resource links: `123456789XXX.us-east-1.testdb1, 123456789XXX.us-east-1.testdb2, 123456789XXX.us-east-1.sales_db*` If specified, Databases resource links to ignore while setting access control policies takes precedence over this setting.	FALSE	ADVANCED
Tables to set access control policies	`CONNECTOR_LAKEFORMATION_MANAGE_TABLE_LIST`	Specifies a comma-separated list of table names for which PolicySync manages access control. If unset, access control is managed for all tables. If specified, use the following format. You can use wildcards. Names are case-sensitive. Syntax: `<CATALOG_ID>.<REGION>.<DATABASE>.<TABLE>` Example list of tables: `123456789XXX.us-east-1.testdb1.test_table1, 123456789XXX.us-east-1.testdb2.test_table2, 123456789XXX.us-east-1.sales_db.sales_data*` If specified, tables to ignore while setting access control policies takes precedence over this setting.	FALSE	CUSTOM

Privacera-aCloud property field name	PM Connector property name	Description	Mandatory	Privacera-aCloud property type
Table Resource Links to set access control policies	CONNECTOR_LAKEFORMATION_MANAGE_TABLE_RESOURCE_LINK_LIST	Specifies a comma-separated list of table resource links for which PolicySync manages access control. If unset, access control is managed for all table resource links. If specified, use the following format. You can use wildcards. Names are case-sensitive. Syntax: <CATALOG_ID>.<REGION>.<DATABASE>.<TABLE_RESOURCE_LINK> Example list of table resource links: 123456789XXX.us-east-1.testdb1.test_table1, 123456789XXX.us-east-1.testdb2.test_table2, 123456789XXX.us-east-1.sales_db.sales_data* If specified, Table resource links to ignore while setting access control policies takes precedence over this setting.	FALSE	CUSTOM
Tags to set access control policies	CONNECTOR_LAKEFORMATION_MANAGE_TAG_LIST	Specifies a comma-separated list of tags for which PolicySync manages access control. If unset, access control is managed for all tags. If specified, use the following format. You can use wildcards. Names are case-sensitive. Syntax:<CATALOG_ID>.<REGION>.<TAG> Example list of tags: 123456789XXX.us-east-1.test_tag1, 123456789XXX.us-east-1.test_tag2, 123456789XXX.us-east-1.sales_db_tag* If specified, tags to ignore while setting access control policies takes precedence over this setting.	FALSE	CUSTOM
Catalogs to ignore for access control policies	CONNECTOR_LAKEFORMATION_IGNORE_CATALOG_LIST	Specifies a comma-separated list of AWS catalog ids that PolicySync does not provide access control for. You can specify wildcards. Names are case-sensitive. If not specified, all catalogs from manage catalog list are subject to access control. Example: 123456789XXX, 987654321XXX, 1234* This setting supersedes any values specified by manage catalog list to set access control policies.	FALSE	ADVANCED
Data Locations to ignore for access control policies	CONNECTOR_LAKEFORMATION_IGNORE_DATA_LOCATION_LIST	Specifies a comma-separated list of data locations that PolicySync does not provide access control for. You can specify wildcards. Names are case-sensitive. If not specified, all data locations specified in manage data locations list are subject to access control. Example: 123456789XXX.us-east-1.demo-s3-bucket/test_data*. This setting supersedes any values specified by manage data location list to set access control policies.	FALSE	ADVANCED
Databases to ignore for access control policies	CONNECTOR_LAKEFORMATION_IGNORE_DATABASE_LIST	Specifies a comma-separated list of database names that PolicySync does not provide access control for. You can specify wildcards. Names are case-sensitive. If not specified, all manage database list are subject to access control. Syntax: <CATALOG_ID>.<REGION>.<DATABASE> Example: 123456789XXX.us-east-1.testdb1, 123456789XXX.us-east-1.testdb2, 123456789XXX.us-east-1.sales_db* This setting supersedes any values specified by manage database list to set access control policies.	FALSE	CUSTOM

Privacera-aCloud property field name	PM Connector property name	Description	Mandatory	Privacera-aCloud property type
Database Resource Links to ignore for access control policies	`CONNECTOR_LAKEFORMATION_IGNORE_DATABASE_RESOURCE_LINK_LIST`	Specifies a comma-separated list of database resource links that PolicySync does not provide access control for. You can specify wildcards. Names are case-sensitive. If not specified, all from manage database resource link list are subject to access control. Example: `123456789XXX.us-east-1.testdb1, 123456789XXX.us-east-1.testdb2, 123456789XXX.us-east-1.sales_db*` This setting supersedes any values specified by manage database resource link list to set access control policies.	FALSE	ADVANCED
Tables to ignore for access control policies	`CONNECTOR_LAKEFORMATION_IGNORE_TABLE_LIST`	Specifies a comma-separated list of table names that PolicySync does not provide access control for. You can specify wildcards. Names are case-sensitive. If not specified, all tables are subject to access control. Syntax: `<CATALOG_ID>.<REGION>.<DATABASE>.<TABLE>` Example: `123456789XXX.us-east-1.testdb1.test_table1, 123456789XXX.us-east-1.testdb2.test_table2, 123456789XXX.us-east-1.sales_db.sales_data*.` This setting supersedes any values specified by Tables to set access control policies.	FALSE	CUSTOM
Tables resource links to ignore for access control policies	`CONNECTOR_LAKEFORMATION_IGNORE_TABLE_RESOURCE_LINK_LIST`	Specifies a comma-separated list of table resource links that PolicySync does not provide access control for. You can specify wildcards. Names are case-sensitive. If not specified, all table resource links are subject to access control. Example: `123456789XXX.us-east-1.testdb1.test_table1, 123456789XXX.us-east-1.testdb2.test_table2, 123456789XXX.us-east-1.sales_db.sales_data*` This setting supersedes any values specified by Table resource links to set access control policies.	FALSE	CUSTOM
Tags to ignore for access control policies	`CONNECTOR_LAKEFORMATION_IGNORE_TAG_LIST`	Specifies a comma-separated list of tag names that PolicySync does not provide access control for. You can specify wildcards. Names are case-sensitive. If not specified, all tags are subject to access control. Syntax: `<CATALOG_ID>.<REGION>.<AG>` Example: `123456789XXX.us-east-1.test_tag1, 123456789XXX.us-east-1.test_tag2, 123456789XXX.us-east-1.sales_db_tag*` This setting supersedes any values specified by Tags to set access control policies.	FALSE	CUSTOM
Database names to set access control policies	`CONNECTOR_LAKEFORMATION_LF_MANAGE_DATABASE_LIST`	Specifies a comma-separated list of database names for which PolicySync manages access control across the specified regions. If unset, access control is managed for all databases. If specified, use the following format. You can use wildcards. Names are case-sensitive. Example list of databases: `testdb1, testdb2, sales_db*` If specified, Databases to ignore while setting access control policies takes precedence over this setting.	FALSE	BASIC

Privacera-aCloud property field name	PM Connector property name	Description	Mandatory	Privacera-aCloud property type
Database names to ignore for access control policies	`CONNECTOR_LAKEFORMATION_LF_IGNORE_DATABASE_LIST`	Specifies a comma-separated list of database names that PolicySync does not provide access control across specified regions. You can specify wildcards. Names are case-sensitive. If not specified, all manage database list are subject to access control. Example list of databases: `testdb1, testdb2, sales_db*` This setting supersedes any values specified by manage database list to set access control policies.	FALSE	ADVANCED

Table 6. User/Group Management

Privacera-Cloud property field name	PM Connector property name	Description	Mandatory	Privacera-aCloud property type
Manage users from portal	`CONNECTOR_LAKEFORMATION_MANAGE_SERVICE_USER`	Set to true for PolicySync to handle Lake Formation roles create/update/delete based on portal roles create/update/delete.	FALSE	CUSTOM
Manage groups from portal	`CONNECTOR_LAKEFORMATION_MANAGE_SERVICE_GROUP`	Set to true for PolicySync to handle Lake Formation role members create/update/delete based on portal role members create/update/delete.	FALSE	CUSTOM
Manage group members from portal	`CONNECTOR_LAKEFORMATION_MANAGE_SERVICE_GROUP_MEMBERS`	Specifies a comma-separated list of user names for which PolicySync manages access control. You can use wildcards. Names are case-sensitive. Example user list: `user1,user2,dev_user*` If not specified, PolicySync manages access control for all users. If specified, Users to be ignored by access control policies takes precedence over this setting.	FALSE	CUSTOM
Manage roles from portal	`CONNECTOR_LAKEFORMATION_MANAGE_SERVICE_ROLE`	Specifies a comma-separated list of group names for which PolicySync manages access control. You can use wildcards. Names are case-sensitive. Example group list: `group1, group2, dev_group*` If not specified, PolicySync manages access control for all groups. If specified, Groups to be ignored by access control policies takes precedence over this setting.	FALSE	CUSTOM
Manage role members from portal	`CONNECTOR_LAKEFORMATION_MANAGE_SERVICE_ROLE_MEMBERS`	Specifies a comma-separated list of role names for which PolicySync manages access control. You can use wildcards. Names are case-sensitive. Example role list: `role1, role2, dev_role*` If not specified, PolicySync manages access control for all roles. If specified, Roles to be ignored by access control policies takes precedence over this setting.	FALSE	CUSTOM
Users to set access control policies	`CONNECTOR_LAKEFORMATION_MANAGE_USER_LIST`	Specifies a comma-separated list of user names of which PolicySync does not provide access control. You can specify wildcards. Names are case-sensitive. If not specified, all from manage user list are subject to access control. An example user list: `user1,user2,dev_user*` This setting supersedes any values specified by Users to set access control policies.	FALSE	ADVANCED

Privacera-Cloud property field name	PM Connector property name	Description	Mandatory	PrivaceraCloud property type
Groups to set access control policies	`CONNECTOR_LAKEFORMATION_MANAGE_GROUP_LIST`	Specifies a comma-separated list of group names of which PolicySync does not provide access control. You can specify wildcards. Names are case-sensitive. If not specified, all from manage group list are subject to access control. Example group list: `group1, group2, dev_group*` This setting supersedes any values specified by Groups to set access control policies.	FALSE	ADVANCED
Roles to set access control policies	`CONNECTOR_LAKEFORMATION_MANAGE_ROLE_LIST`	Specifies a comma-separated list of role names of which PolicySync does not provide access control. You can specify wildcards. Names are case-sensitive. If not specified, all from manage role list are subject to access control. Example role list: `role1, role2, dev_role*` This setting supersedes any values specified by Roles to set access control policies.	FALSE	ADVANCED
Users to be ignored by access control policies	`CONNECTOR_LAKEFORMATION_IGNORE_USER_LIST`	Specifies a comma-separated list of user names of which PolicySync does not provide access control. You can specify wildcards. Names are case-sensitive. If not specified, all from manage user list are subject to access control. Example user list might resemble the following: `user1,user2,dev_user*` This setting supersedes any values specified by Users to set access control policies.	FALSE	ADVANCED
Groups to be ignored by access control policies	`CONNECTOR_LAKEFORMATION_IGNORE_GROUP_LIST`	Specifies a comma-separated list of group names of which PolicySync does not provide access control. You can specify wildcards. Names are case-sensitive. If not specified, all from manage group list are subject to access control. An example group list: `group1, group2, dev_group*` This setting supersedes any values specified by Groups to set access control policies.	FALSE	ADVANCED
Roles to be ignored by access control policies	`CONNECTOR_LAKEFORMATION_IGNORE_ROLE_LIST`	Specifies a comma-separated list of role names of which PolicySync does not provide access control. You can specify wildcards. Names are case-sensitive. If not specified, all from manage role list are subject to access control. An example role list: `role1, role2, dev_role*` This setting supersedes any values specified by Roles to set access control policies.	FALSE	ADVANCED
Use lake formation native public group for public group access policies	`CONNECTOR_LAKEFORMATION_USE_NATIVE_PUBLIC_GROUP`	Set this property to true if you want PolicySync to use Lake Formation's native public group for access grants whenever there is a policy created referring to the public group inside it. The native public group of the Lake Formation has `<aws-account-id>:IAMPrincipals` as ARN.	TRUE	ADVANCED

Table 7. Access control management

PrivaceraCloud property field name	PM Connector property name	Description	PM Default value	Mandatory	PrivaceraCloud property type
Enforce lakeformation native row filter	`CONNECTOR_LAKEFORMATION_ENABLE_ROW_FILTER`	Specifies whether to use secure view based row filtering. The default value is true.	TRUE	FALSE	ADVANCED
Enable policy enforcements and user/group/role management	`CONNECTOR_LAKEFORMATION_ENABLE_GRANT_UPDATES`	Specifies whether PolicySync performs grants and revokes for access control and creates, updates, and deletes api calls for users, groups, and roles. The default value is false.	FALSE	FALSE	BASIC

Table 8. Access audits mangement

PrivaceraCloud property field name	PM Connector property	Description	Mandatory	PrivaceraCloud property type
Enable access audits	`CONNECTOR_LAKEFORMATION_ENABLE_ACCESS_AUDITS`	Specifies whether Privacera should retrieve access audit data from the Lake Formation. Default value is `false`.	FALSE	BASIC
Users to exclude when fetching access audits	`CONNECTOR_LAKEFORMATION_AUDIT_EXCLUDED_USERS`	Specifies a list of list of users to exclude when fetching access audits.	FALSE	ADVANCED
Access Types to exclude when fetching access audits	`CONNECTOR_LAKEFORMATION_AUDIT_EXCLUDED_ACCESS_TYPES`	Specifies a list of list of access types to exclude when fetching access audits. Example: `StartQueryExecution, GetTable,DeleteTable, CreateTable, CreateDatabase`	FALSE	CUSTOM
AWS Athena region for JDBC connection to audit logs database	`CONNECTOR_LAKEFORMATION_AWS_ATHENA_REGION`	Specifies AWS Athena region to create JDBC connection for Lake formation audit logs database. If not specified, it will default to use the first region from AWS Regions property.	FALSE	CUSTOM
AWS Athena endpoint for JDBC connection to audit logs database	`CONNECTOR_LAKEFORMATION_AWS_ATHENA_ENDPOINT`	Specifies AWS Athena endpoint to create JDBC connection for Lake Formation audit logs database. If not specified, by default it will create endpoint with specified/default region from region property . Example: `athena.region.amazonaws.com:443`	FALSE	CUSTOM
AWS Athena workgroup for JDBC connection to audit logs database	`CONNECTOR_LAKEFORMATION_AWS_ATHENA_WORKGROUP`	Specifies AWS Athena workgroup to create JDBC connection for Lake Formation audit logs database.	FALSE	CUSTOM
Lake formation audit logs database	`CONNECTOR_LAKEFORMATION_AUDIT_DB_NAME`	Specifies AWS audit database to store Lake Formation audit logs	FALSE	BASIC
Lake Formation audit logs table name	`CONNECTOR_LAKEFORMATION_AUDIT_TABLE_NAME`	Specifies AWS audit table to store Lake Formation audit logs	FALSE	BASIC

PrivaceraCloud property field name	PM Connector property	Description	Mandatory	Privacera-Cloud property type
S3 output location for access audit logs query results	`CONNECTOR_LAKEFORMATION_AUDIT_ATHENA_S3_OUTPUT_LOCATION`	Specifies S3 location to store the access audit logs query results. Example: `s3://privacera-dev-XXX/<LF_audit_logs_folder>/athena_query_results/`	FALSE	BASIC

Table 9. Reverse sink properties

Privacera-Cloud property field name	PM Connector property name	Description	PM Default value	Mandatory	Privacera-Cloud property type
Lake Formation permissions sink type	`CONNECTOR_LAKEFORMATION_PERMISSION_SINK_TYPE`	Specifies Lake Formation permissions sink type. `reverse_sink` - all policies from lake formation will be loaded into Ranger within configured services(Lake formation, Hive or Unity Catalog). `sink` - policies from Privacera Ranger service will be applied into AWS Lake Formation.	`sink`	TRUE	BASIC
Lake Formation sink max index	`CONNECTOR_LAKEFORMATION_SINK_MAX_INDEX`	Specifies Lake Formation sink max index is the number of max services. Can be configured for reverse sink	`10`	FALSE	CUSTOM
Push Lake Formation permissions to Hive	`CONNECTOR_LAKEFORMATION_SINK_HIVE_ENABLED`	Specifies whether to enable policy sink to Hive service.	FALSE	FALSE	ADVANCED
Policy repository name for Hive service	`CONNECTOR_LAKEFORMATION_SINK_HIVE_SERVICE_APP_ID`	Specifies the policy repository name for Hive service.	`privacera_hive`	FALSE	ADVANCED
Enable read-only policy creation	`CONNECTOR_LAKEFORMATION_ENABLE_CREATE_READONLY_POLICIES`	Specifies whether policies created by PolicySync are editable or read-only.	FALSE	FALSE	ADVANCED

Google BigQuery connector for PolicySync on Privacera Platform

Google BigQuery provides fine-grained access control on datasets. This includes:

- Table-level Access Control
- Column-level Access Control
- Native/Dynamic secure view-based Row Filter
- Masking With Dynamic secure views created using PolicySync

Generalized approach for implementing PolicySync

To help you reach compliance, Privacera PolicySync distributes your defined access management policies to the third-party datasources you connect to Privacera.

Use this generalized approach for implementing PolicySync.

1. Understand how PolicySync works and how it is configured. See PolicySync design and configuration on Privacera Platform [61].

2. Decide which PolicySync topology best suits your needs:
   - Required basic PolicySync topology: always at least one connector instance [64]
   - Optional topology: multiple connector instances for Kubernetes pods and Docker containers [65]
   - Recommended PolicySync topology: individual policy repositories for individual connectors [65]
3. Create the required, basic PolicySync configuration. See Required basic PolicySync topology: always at least one connector instance [64]
4. Examine the BASIC and ADVANCED properties, decide which features you want to implement, and set the necessary values in the YAML property file.

Connector name: bigquery

When you create the connector as detailed in PolicySync design and configuration on Privacera Platform [61], use the following reserved word for the name of the connector:

```
bigquery
```

In formal syntax shown in Connector instance directory/file structure [63] replace `<ConnectorName>` with the above and in the example in Required basic PolicySync topology: always at least one connector instance [64], replace `postgres` with the above.

Prerequisites

Create PrivaceraPolicySyncRole IAM Role

You need to give Privacera PolicySync basic access to GCP. To grant that access, create `PrivaceraPolicySyncRole` IAM role in your GCP project or GCP organization using the following commands on Google Cloud's shell (gcloud). The shell can be installed and accessed locally or through Google Console.

Run the following command to create the file containing the permissions required for the `PrivaceraPolicySyncRole` role:

```
ROLE_NAME="PrivaceraPolicySyncRole"

cat << EOF > ${ROLE_NAME}.yaml
title: "${ROLE_NAME}"
description: "${ROLE_NAME}"
stage: "ALPHA"
includedPermissions:
- resourcemanager.projects.get
- resourcemanager.projects.getIamPolicy
- resourcemanager.projects.setIamPolicy
- iam.roles.list
- iam.roles.get
- iam.roles.create
- iam.roles.update
- bigquery.jobs.create
- bigquery.datasets.get
- bigquery.datasets.create
- bigquery.datasets.update
- bigquery.datasets.delete
- bigquery.datasets.getIamPolicy
- bigquery.datasets.setIamPolicy
- bigquery.tables.list
- bigquery.tables.get
- bigquery.tables.getData
- bigquery.tables.create
- bigquery.tables.update
```

```
- bigquery.tables.delete
- bigquery.tables.getIamPolicy
- bigquery.tables.setIamPolicy
- bigquery.rowAccessPolicies.list
- bigquery.rowAccessPolicies.create
- bigquery.rowAccessPolicies.update
- bigquery.rowAccessPolicies.delete
- bigquery.rowAccessPolicies.getIamPolicy
- bigquery.rowAccessPolicies.setIamPolicy

EOF
```

GCP Project-level access

NOTE
If you have multiple projects in your GCP organization and want them to be managed by a single BigQuery connector, then repeat the steps below for each project. Assign the role to the same service account which will be used across multiple projects.

1. Run the following command. Replace <GCP_PROJECT_ID> with your GCP project ID.

   ```
   PROJECT_ID="<GCP_PROJECT_ID>"
   ```

2. To create PrivaceraPolicySyncRole role in your GCP project, run the following command.

   ```
   gcloud iam roles create ${ROLE_NAME} --project=${PROJECT_ID} --file=${ROLE_NAME}.yaml
   ```

GCP Organization-level access

1. Run the following command. Replace <GCP_ORGANIZATION_ID> with your GCP organization ID.

   ```
   ORGANIZATION_ID="<GCP_ORGANIZATION_ID>"
   ```

2. To create PrivaceraPolicySyncRole role in your GCP organization, run the following command.

   ```
   gcloud iam roles create ${ROLE_NAME} --organization=${ORGANIZATION_ID} --file=${ROLE_
   ```

Attach IAM Role to Service Account

To attach the PrivaceraPolicySyncRole IAM role created above, do the following steps:

1. Log in to your GCP console.
2. Select **IAM & admin > Service accounts** and click **+ CREATE SERVICE ACCOUNT**.
3. Enter values in the fields and click **CREATE**.
4. In **Grant this service account access to project**, select the role as PrivaceraPolicySyncRole.
5. On the Services Account Page, find the newly created service account and copy the email address of the new service account for use in a later step.

NOTE
This email will be the Service Account Email for configuring PolicySync in Privacera Manager.

6. If you are using a Google VM machine to configure Google BigQuery for PolicySync, then you can attach the service account created above to your VM machine and skip below steps.
7. On the Services Account Page, go to the **Keys** tab and click **Add Key** and select **Create New Key**.
8. Select the **JSON** key type, and click **CREATE**. A JSON key file downloads to your system. Store the file at an accessible location. It will be used for configuring PolicySync in Privacera Manager.

See the Google documentation for a detailed information on creating a service account.

Configure Logs for Auditing

A sink is required to collect all the logs from Google BigQuery. To create a sink, do the following steps:

1. In the search bar, search for *Logging*, and then click **Logs Router**, and click **Create Sink**.
2. Enter the sink name as *PolicySyncBigQueryAuditSink*, and then click **Next**.
3. Enter the sink destination.
   a. In the **Select sink service**, select **BigQuery**.
   b. In **SelectBigQuerydataset**, click **Create newBigQuerydataset**.
   c. Enter the **Dataset ID** as *bigquery_audits* and click **Create Dataset**.
   d. Click **Next**.
4. Add theBigQuerylogs in the sink:
   In the **Build an inclusion filter**, add the following line:

   ```
   resource.type="bigquery_resource"
   ```
5. Click **Create Sink**.

See the Google documentation for a detailed information on creating a sink.

Optional Basic Authentication for PolicySync

To optionally enable basic authenticate for PolicySync toGoogle BigQuery you can create a JSON file in your connector instance subdirectory.

The name of the file must be `XXX.json`.

An example of the contents of `XXX.json`.:

```
 {
  "type": "service_account",
  "project_id": "your_project_id",
  "private_key_id": "autogenerated_value",
  "private_key": "-----BEGIN PRIVATE KEY-----autogenerated_value-----END PRIVATE KEY----
  "client_email": "autogenerated_value",
  "client_id": "autogenerated_value",
  "auth_uri": "https://accounts.google.com/o/oauth2/auth",
  "token_uri": "https://oauth2.googleapis.com/token",
  "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
  "client_x509_cert_url": "https://www.googleapis.com/robot/v1/metadata/x509/autogenerat
}
```

BigQuery connector properties for PolicySync on Privacera Platform

These BigQuery connector properties can be set for PolicySync in Privacera Platform.

The properties are grouped by general function, such as JDBC connection properties, properties for user, group, and role management, and other functions.

The properties are also categorized as **BASIC** or **ADVANCED**:

- **BASIC** pertains to the most fundamental aspects of the connector, such as authentication.
- **ADVANCED** indicates additional features beyond the **BASIC**s, such as row-filtering or group member handling.

Start by setting the **BASIC** properties and then examine the **ADVANCED** properties to determine which of these features you might want to enable.

For a general process to migrate values from old YAML files to the new YAML files, see Migration to PolicySync v2 on Privacera Platform 7.2 [68].

Category	Property name	
JDBC configuration properties		
BASIC	`CONNECTOR_BIG-QUERY_PROJECT_LOCATION`	The BigQuery data location. We can check any of the table details and we can see its data location va
BASIC	`CONNECTOR_BIG-QUERY_PROJECT_ID`	This property is used to set the project id which can be used for initial interaction with BigQuery APIs to
	`CONNECTOR_BIG-QUERY_JDBC_URL`	This property is used to set JDBC url which can be used to connect to BigQuery server.
	`CONNECTOR_BIG-QUERY_USE_VM_CREDENTIALS`	This is property is used to set whether if you want to use service account attached to your VM for getti
BASIC	`CONNECTOR_BIG-QUERY_OAUTH_SERVICE_AC-COUNT_EMAIL`	This property is used to set service account email address which is to be used for PolicySync to conne
	`CONNECTOR_BIG-QUERY_OAUTH_PRI-VATE_KEY_PATH`	This property is used to set the path of service account credential json file downloaded from google se
BASIC	`CONNECTOR_BIG-QUERY_OAUTH_PRI-VATE_KEY_FILE_NAME`	This property is used to set the credential JSON file name which you have copied inside your connecto
BASIC	NA	
Custom IAM Roles		
AD-VANCED	`CONNECTOR_BIGQUERY_CRE-ATE_CUSTOM_IAM_ROLES`	Enable this property if you want PolicySync to create custom IAM roles automatically in your GCP proj
AD-VANCED	`CONNECTOR_BIGQUERY_CUS-TOM_IAM_ROLES_SCOPE`	BigQuery PolicySync used custom IAM roles for fine grained access control. These roles can be create
AD-VANCED	`CONNECTOR_BIGQUERY_OR-GANIZATION_ID`	This is property is used to set your GCP organization id if you decide to use the custom IAM roles at th
AD-VANCED	`CONNECTOR_BIGQUERY_CUS-TOM_IAM_ROLES_NAME_MAP-PING`	BigQuery PolicySync used custom IAM roles for fine grained access control. These roles are created v `PrivaceraGBQProjectListRole:CustomBigQueryProjectListRole,PrivaceraGBQJobLi`
Load keys and intervals		
	`CONNECTOR_BIGQUERY_AU-DIT_SYNC_INTERVAL`	This property is used to set the interval in seconds for getting access audits process. Access audits pr
BASIC	`CONNECTOR_BIGQUERY_MAN-AGE_PROJECT_LIST`	This property is used to set comma separated project names for which access control should be mana

NOTE
Values for this property are case-sensitive.

Category	Property name	
ADVANCED	CONNECTOR_BIGQUERY_MANAGE_DATASET_LIST	This property is used to set comma separated dataset fqdn for which access control should be manage
		 **NOTE** Values for this property are case-sensitive.
ADVANCED	CONNECTOR_BIGQUERY_MANAGE_TABLE_LIST	This property is used to set comma separated table/view fqdn for which access control should be mana
		 **NOTE** Values for this property are case-sensitive.
ADVANCED	CONNECTOR_BIGQUERY_IGNORE_PROJECT_LIST	This property is used to set comma separated project names for which you don't want access control t
		 **NOTE** Values for this property are case-sensitive.
ADVANCED	CONNECTOR_BIGQUERY_IGNORE_DATASET_LIST	This property is used to set comma separated dataset fqdn for which you don't want access control to
		 **NOTE** Values for this property are case-sensitive.
ADVANCED	CONNECTOR_BIGQUERY_IGNORE_TABLE_LIST	This property is used to set comma separated table/view fqdn for which you don't want access control
		 **NOTE** Values for this property are case-sensitive.
Users/Groups/Roles management		
ADVANCED	CONNECTOR_BIGQUERY_MANAGE_USER_LIST	This property is used to set comma separated user names for which access control should be manage
ADVANCED	CONNECTOR_BIGQUERY_MANAGE_GROUP_LIST	This property is used to set comma separated group names for which access control should be manag
ADVANCED	CONNECTOR_BIGQUERY_IGNORE_USER_LIST	This property is used to set comma separated user names for which you don't want access control to b
ADVANCED	CONNECTOR_BIGQUERY_IGNORE_GROUP_LIST	This property is used to set comma separated group names for which you don't want access control to
BASIC	CONNECTOR_BIGQUERY_NATIVE_PUBLIC_GROUP_IDENTITY_NAME	Set this property to your preferred value, PolicySync uses this native public group for access grants wh
ADVANCED	CONNECTOR_BIGQUERY_MANAGE_USER_FILTERBY_GROUP	Set this property to true, if you want to manage only the users who belongs the the groups defined in m
Access control management		
ADVANCED	CONNECTOR_BIGQUERY_COLUMN_ACCESS_CONTROL_TYPE	This property is used to set the method of column level access control to be used by PolicySync.
ADVANCED	CONNECTOR_BIGQUERY_POLICY_NAME_SEPARATOR	This property is used to set separator, this separator is used while creating name for native tr filter polic

Category	Property name	
ADVANCED	CONNECTOR_BIGQUERY_ROW_FILTER_POLICY_NAME_TEMPLATE	This property is used to set template, which will be used to create native tr filter policy names. For exa
ADVANCED	CONNECTOR_BIGQUERY_ENABLE_ROW_FILTER	Set this property to true, if you want to enable native tr filter functionality. This is not recommended to
ADVANCED	CONNECTOR_BIGQUERY_ENABLE_VIEW_BASED_MASKING	Set this property to true, if you want to enable secure view based masking in BigQuery PolicySync.
		 **NOTE** BigQuery don't support native masking yet, thus recommended to use view based m
ADVANCED	CONNECTOR_BIGQUERY_ENABLE_VIEW_BASED_ROW_FILTER	Set this property to true, if you want to enable secure view based tr filter in BigQuery PolicySync.
		 **NOTE** BigQuery support native tr filters, but due to its some limitations we recommended to
ADVANCED	CONNECTOR_BIGQUERY_SECURE_VIEW_CREATE_FOR_ALL	Set this property to true, if you want to create secure view for all tables as well all views which were cr
	CONNECTOR_BIGQUERY_MASKING_FUNCTIONS_DATASET	This property is used to set the name of the dataset in the gcp project which can be used to create cus
ADVANCED	CONNECTOR_BIGQUERY_MASKED_NUMBER_VALUE	This property is used to specify the default masking value for numeric columns
ADVANCED	CONNECTOR_BIGQUERY_MASKED_TEXT_VALUE	This property is used to specify the default masking value for text/string columns
ADVANCED	CONNECTOR_BIGQUERY_SECURE_VIEW_NAME_PREFIX	By default view-based tr filter and masking related secure views have the same name as the table nam
ADVANCED	CONNECTOR_BIGQUERY_SECURE_VIEW_NAME_POSTFIX	By default view-based tr filter and masking related secure views have the same name as the table nam
ADVANCED	CONNECTOR_BIGQUERY_SECURE_VIEW_DATASET_NAME_PREFIX	By default view-based tr filter and masking related secure views have the same schema name as the t
ADVANCED	CONNECTOR_BIGQUERY_SECURE_VIEW_DATASET_NAME_POSTFIX	By default view-based tr filter and masking related secure views have the same schema name as the t
ADVANCED	CONNECTOR_BIGQUERY_SECURE_VIEW_NAME_REMOVE_SUFFIX_LIST	You can remove any unwanted suffix attached at the end of a table/view name. For example, if the tab
ADVANCED	CONNECTOR_BIGQUERY_SECURE_VIEW_DATASET_NAME_REMOVE_SUFFIX_LIST	You can remove any unwanted suffix attached at the end of a schema name. For example, if the schen
	CONNECTOR_BIGQUERY_ENABLE_AUTHORIZED_VIEW_ACL_UPDATER	This property is used to enable unsynchronized authorized view ACLs updater thread, it updates the d
ADVANCED	CONNECTOR_BIGQUERY_GRANT_UPDATES	This property controls whether actual grant/revoke should be run on BigQuery.
ADVANCED	CONNECTOR_BIGQUERY_ENABLE_DATA_ADMIN	This property is used to enable data admin feature, with data admin feature enabled you can create all
Access audits management		
BASIC	CONNECTOR_BIGQUERY_AUDIT_ENABLE	This property is used to enable access audit fetching from BigQuery.
ADVANCED	CONNECTOR_BIGQUERY_AUDIT_EXCLUDED_USERS	This property is used to set the list of users whose access audits should be ignored by PolicySync. It ta
ADVANCED	CONNECTOR_BIGQUERY_AUDIT_PROJECT_ID	This property is used to set the project id which should be used when running query to fetch the audits

Category	Property name	
ADVANCED	CONNECTOR_BIGQUERY_AUDIT_DATASET_NAME	This property is used to set the dataset name which should be used when running query to fetch the a

Microsoft SQL Server connector for PolicySync on Privacera Platform

These instructions enable and configure Privacera Microsoft SQL Server database connector to an existing Microsoft SQL Server database running on the Azure cloud platform or the AWS Relational Database Service (RDS). This connector uses the PolicySync method in which access policies defined in Privacera are mapped to and synchronized to the 'native' access controls in MS SQL.

The PolicySync approach has several benefits and advantages:

- Fine-grained access control - at the database, schema, table, view, and column levels.
- Column level masking
- Dynamic row-level filters on tables and views

If you assign the `CreateTable` permission to a user, group, or role in Access Manager then PolicySync grants both the `CREATE TABLE` permission on the grantee and the `ALTER ON SCHEMA` permission on the grantee for the underlying schema in MS SQL Server. The `ALTER ON SCHEMA` permission is required to create a table in a schema. However, this permission also allows the grantee to delete tables from a schema. The ability to delete tables that the `ALTER ON SCHEMA` permission confers is inherent in the SQL Server permissions implementation.

Generalized approach for implementing PolicySync

To help you reach compliance, Privacera PolicySync distributes your defined access management policies to the third-party datasources you connect to Privacera.

Use this generalized approach for implementing PolicySync.

1. Understand how PolicySync works and how it is configured. See PolicySync design and configuration on Privacera Platform [61].
2. Decide which PolicySync topology best suits your needs:
   - Required basic PolicySync topology: always at least one connector instance [64]
   - Optional topology: multiple connector instances for Kubernetes pods and Docker containers [65]
   - Recommended PolicySync topology: individual policy repositories for individual connectors [65]
3. Create the required, basic PolicySync configuration. See Required basic PolicySync topology: always at least one connector instance [64]
4. Examine the BASIC and ADVANCED properties, decide which features you want to implement, and set the necessary values in the YAML property file.

Connector name: mssql

When you create the connector as detailed in PolicySync design and configuration on Privacera Platform [61], use the following reserved word for the name of the connector:

```
mssql
```

In formal syntax shown in Connector instance directory/file structure [63] replace `<ConnectorName>` with the above and in the example in Required basic PolicySync topology: always at least one connector instance [64], replace `postgres` with the above.

Prerequisites

The Microsoft SQL Server must already be installed and running.

If you are installing an evaluation, you may need to install and configure an Microsoft SQL Server with one or more databases to test against.

1) Target Database Access

The Microsoft SQL Server must also be accessible from the Privacera Platform hosts. The standard inbound port for Microsoft SQL Server access is TCP 1433. Make sure that is open 'outbound' from Privacera Platform hosts and inbound to your target Microsoft SQL Server.

2) Access Control by Privacera Service Account

Privacera Platform requires access to the target database and the service account must be established in a 'loginmanager' role. This can be configured in three ways: (1) Access Control on Azure AD Users; (2) Access Control on Local Database Users; or (3) Access Control on both Azure AD user and local users.

Access Control on Azure AD Users

1. Confirm the Microsoft SQL Server is configured to work with Azure AD Users.
2. In your Azure AD, create a Privacera 'service' user to be used by Privacera for the Policy access control synchronization. For this example we'll assume the name is 'privacera_policysync@example.com' but set the value appropriately for your domain(s). Keep note of the username and password as we'll use both later.
3. For each targeted database:
   a. Log on to the target database with an Admin role account.
   b. Execute the following:

```
IF DATABASE_PRINCIPAL_ID('privacera_policysync@example.com') IS NULL BEGIN
  CREATE USER [privacera_policysync@example.com] FROM EXTERNAL PROVIDER;
  END;

-- Grant full control on database to privacera_policysync@example.com user
GRANT CONTROL ON DATABASE::${YOUR_DATABASE} TO [privacera_policysync@example.com];
```

Access Control on Local Database Users

1. Create a Privacera 'service' user in the master database to be used by Privacera for the Policy access control synchronization. For this example, we'll assume the name is 'privacera_policysync' but set the value appropriately for your domain(s). Keep note of the username and password as we'll use both later.

```
IF NOT EXISTS (SELECT name  FROM sys.sql_logins WHERE name = 'privacera_policysync')
  CREATE LOGIN [privacera_policysync] WITH PASSWORD = '${PASSWORD}'
  END;

IF DATABASE_PRINCIPAL_ID('privacera_policysync') IS NULL BEGIN
  CREATE USER [privacera_policysync] FROM LOGIN [privacera_policysync];
  END;

EXEC sp_addrolemember [loginmanager], [privacera_policysync];
```

2. For each targeted database:
   a. Log on to the target database with an Admin role account.
   b. Execute the following:

```
IF DATABASE_PRINCIPAL_ID('privacera_policysync') IS NULL BEGIN
  CREATE USER [privacera_policysync] FROM LOGIN [privacera_policysync];
  END;

-- Grant full control on database to privacera_policysync user
GRANT CONTROL ON DATABASE::${YOUR_DATABASE} TO [privacera_policysync];
```

Access Control on Azure AD and Local Database Users

1. Confirm the Microsoft SQL Serveris configured to work with Azure AD Users.
2. In your Azure AD, create a Privacera 'service' user to be used by Privacera for the Policy access control synchronization. For this example, we'll assume the name is 'privacera_policysync@example.com' but set the value appropriately for your domain(s). Keep note of the username and password as we'll use both later.
3. Create a Privacera 'service' user in the master database to be used by Privacera for the Policy access control synchronization. For this example, we'll assume the name is 'privacera_policysync@example.com' but set the value appropriately for your domain(s). Keep note of the username and password as we'll use both later.

   ```
   IF DATABASE_PRINCIPAL_ID('privacera_policysync@example.com') IS NULL BEGIN
     CREATE USER [privacera_policysync@example.com] FROM EXTERNAL PROVIDER;
     END;

   EXEC sp_addrolemember [loginmanager], [privacera_policysync@example.com];
   ```
4. For each targeted database:
   a. Log on to the target database with an Admin role account.
   b. Execute the following:

      ```
      IF DATABASE_PRINCIPAL_ID('privacera_policysync@example.com') IS NULL BEGIN
        CREATE USER [privacera_policysync@example.com] FROM EXTERNAL PROVIDER;
        END;

      -- Grant full control on database to privacera_policysync@example.com user
      GRANT CONTROL ON DATABASE::${YOUR_DATABASE} TO [privacera_policysync@example.com]
      ```

3) Create or Identify an ADLS Gen2 storage used to store Microsoft SQL Server Audits

1. See Configure Microsoft SQL server for database synapse audits [251].
2. Using information from that article obtain the Audit storage URL. This will be used in the Privacera Microsoft SQL Server PolicySync configuration.

4) Create an MS SQL server in AWS RDS to store MSSQL Server Audits

1. Consult the following article SQL Server Audit.
2. Using information from that article obtain the Audit storage URL. This will be used in the Privacera Microsoft SQL Server PolicySync configuration.

Microsoft SQL connector properties for PolicySync on Privacera Platform

These Microsoft SQL connector properties can be set for PolicySync in Privacera Platform.

The properties are grouped by general function, such as JDBC connection properties, properties for user, group, and role management, and other functions.

The properties are also categorized as **BASIC** or **ADVANCED**:

- **BASIC** pertains to the most fundamental aspects of the connector, such as authentication.
- **ADVANCED** indicates additional features beyond the **BASIC**s, such as row-filtering or group member handling.

Start by setting the **BASIC** properties and then examine the **ADVANCED** properties to determine which of these features you might want to enable.

For a general process to migrate values from old YAML files to the new YAML files, see Migration to PolicySync v2 on Privacera Platform 7.2 [68].

Category	Property name	Description	Default	Allowable values
JDBC configuration properties				
BASIC	CONNECTOR_MSSQL_JDBC_URL	This property is used to set jdbc url which can be used to connect to Mssql server. JDBC URL should follow below convention jdbc:sqlserver://<MSSQL_SERVER_HOST>:<MSSQL_SERVER_PORT> Example : jdbc:sqlserver://example.database.windows.net:1433		
BASIC	CONNECTOR_MSSQL_JDBC_USERNAME	This property is used to set jdbc Username to be used to make connection to MSSQL.		
BASIC	CONNECTOR_MSSQL_JDBC_PASSWORD	This property is used to set jdbc username password to be used to make connection to MSSQL.		
BASIC	CONNECTOR_MSSQL_MASTER_DB	This property is used to set jdbc master database to be used to make initial connection to Mssql.	master	
BASIC	CONNECTOR_MSSQL_AUTHENTICATION_TYPE	If MSSQL_JDBC_USERNAME is a 'local user', set value as below: MSSQL_AUTHENTICATION_TYPE: "SqlPassword" If MSSQL_JDBC_USERNAME is an Azure AD user, then set as below: MSSQL_AUTHENTICATION_TYPE: "ActiveDirectoryPassword".	SqlPassword	SqlPassword/ ActiveDirectory-Password
BASIC	CONNECTOR_MSSQL_DEFAULT_USER_PASSWORD	This property is used to set password to every new user created by policysync in MSSQL.		
BASIC	CONNECTOR_MSSQL_OWNER_ROLE	This property is used to set ownership for all the resources managed by policysync. The specified role will become owner for all managed resources and will have full control on those resources.We support changing owners of database, schema, tables and views. Note :- If owner role is kept as blank, then ownership will not change and users who creates table/view or any other object he will be owner of those objects and policysync won't be able to do access control on that object		
Resources management				

Category	Property name	Description	Default	Allowable values
ADVANCED	`CONNECTOR_MSSQL_MANAGE_DATABASE_LIST`	This property is used to set database name for which access control should be managed by policysync. The ignore database list has precedence over manage database list. Eg. `testdb1`. **NOTE** For mssql connector we support only 1 database at a time for access control, providing the multiple databases list here will not work, value for this property are case-sensitive.		
ADVANCED	`CONNECTOR_MSSQL_MANAGE_SCHEMA_LIST`	This property is used to set list of comma separated schema names/wildcards for schema names of database to be managed by policysync. Not specifying this property indicates manage all schemas. Schemas should be specified in format of `<database>.<schema>`. **NOTE** Values for this property are case-sensitive.		

Category	Property name	Description	Default	Allowable values
ADVANCED	CONNECTOR_MSSQL_MANAGE_TABLE_LIST	This property is used to set list of comma separated tables names/ wildcards for table names of the managed schema to be managed by policysync. Not specifying this property indicates manage all tables. tables should be specified in format of <database>.<schema>.<table> . **NOTE** Values for this property are case-sensitive.		
ADVANCED	CONNECTOR_MSSQL_IGNORE_DATABASE_LIST	This property is used to set comma separated database names which access control you don't want to be managed by policysync. If you don't want to ignore any database then you can skip specifying this property. This supports wildcards as well. This has precedence over manage database list. Eg. testdb1,testdb2,sales_db* . **NOTE** Values for this property are case-sensitive.		
ADVANCED	CONNECTOR_MSSQL_IGNORE_SCHEMA_LIST	This property is used to set comma separated schema fqdn which access control you don't want to be managed by policysync. If you don't want to ignore any schema then you can skip specifying this property. This supports wildcards as well. This has precedence over manage schema list. Eg. testdb1.schema1,testdb2.schema2,sales_db*.sales*. **NOTE** Values for this property are case-sensitive.		
Users/Groups/Roles management				

Category	Property name	Description	Default	Allowable values
ADVANCED	CONNECTOR_MSSQL_USER_NAME_REPLACE_FROM_REGEX	This takes the regular expression as input and finds the matching characters in a user name and replaces them with the characters specified in property. If kept blank, no find and replace operation is performed.	[~`$&+:;=?@#|'<>.^*()_%\\\\[\\\\]!\\\\-\\\\/\\\\\\\\{}]	
ADVANCED	CONNECTOR_MSSQL_USER_NAME_REPLACE_TO_STRING	The value specified in this property is used to replace the characters found by the regex specified user name regex property. If kept blank, no find and replace operation is performed.	_	
ADVANCED	CONNECTOR_MSSQL_GROUP_NAME_REPLACE_FROM_REGEX	This takes the regular expression as input and finds the matching characters in a group name and replaces them with the characters specified in property. If kept blank, no find and replace operation is performed.	[~`$&+:;=?@#|'<>.^*()_%\\\\[\\\\]!\\\\-\\\\/\\\\\\\\{}]	
ADVANCED	CONNECTOR_MSSQL_GROUP_NAME_REPLACE_TO_STRING	The value specified in this property is used to replace the characters found by the regex specified group name regex property. If kept blank, no find and replace operation is performed.	_	
ADVANCED	CONNECTOR_MSSQL_ROLE_NAME_REPLACE_FROM_REGEX	This takes the regular expression as input and finds the matching characters in a role name and replaces them with the characters specified in property. If kept blank, no find and replace operation is performed.	[~`$&+:;=?@#|'<>.^*()_%\\\\[\\\\]!\\\\-\\\\/\\\\\\\\{}]	
ADVANCED	CONNECTOR_MSSQL_ROLE_NAME_REPLACE_TO_STRING	The value specified in this property is used to replace the characters found by the regex specified role name regex property. If kept blank, no find and replace operation is performed.	_	
ADVANCED	CONNECTOR_MSSQL_USER_NAME_PERSIST_CASE_SENSITIVITY	After loading user from Ranger API's all are converted into lowercase, but in some cases, you would need to have the users in the same case as they are in Ranger. When setting this value to true, it will maintain the case sensitivity of names as they are in Ranger.	false	true, false
ADVANCED	CONNECTOR_MSSQL_GROUP_NAME_PERSIST_CASE_SENSITIVITY	After loading group from Ranger API's all are converted into lowercase, but in some cases, you would need to have the groups in the same case as they are in Ranger. When setting this value to true, it will maintain the case sensitivity of names as they are in Ranger.	false	true, false
ADVANCED	CONNECTOR_MSSQL_ROLE_NAME_PERSIST_CASE_SENSITIVITY	After loading role from Ranger API's all are converted into lowercase, but in some cases, you would need to have the roles in the same case as they are in Ranger. When setting this value to true, it will maintain the case sensitivity of names as they are in Ranger.	false	true, false

Category	Property name	Description	Default	Allowable values
	CONNECTOR_MSSQL_USER_NAME_CASE_CONVERSION	This property only applicable if USER NAME PERSIST CASE SENSITIVITY is set to false. Managed users name would be treated as lowercase by default. If the value is set to "upper" then the user name would be treated as uppercase. If the value is set to "none" then the user name case is preserved.	lower	none, lower, upper
ADVANCED	CONNECTOR_MSSQL_MANAGE_USERS	This property controls whether policysync should manage the membership between user and user role.	false	true, false
ADVANCED	CONNECTOR_MSSQL_MANAGE_GROUPS	This property controls whether we should create role in MSSQL for groups fetched from ranger.	false	true, false
ADVANCED	CONNECTOR_MSSQL_MANAGE_ROLES	This property controls whether we should create role in MSSQL for roles fetched from ranger.	false	true, false
ADVANCED	CONNECTOR_MSSQL_MANAGE_USER_LIST	This property is used to set comma separated user names which access control should be managed by policysync. If you want to manage all users then you can skip specifying this property. This supports wildcards as well. The ignore users list has precedence over manage users list. Eg. user1,user2,dev_user*		
ADVANCED	CONNECTOR_MSSQL_MANAGE_GROUP_LIST	This property is used to set comma separated group names which access control should be managed by policysync. If you want to manage all group then you can skip specifying this property. This supports wildcards as well. The ignore group list has precedence over manage group list. Eg. group1,group2,dev_group*		
ADVANCED	CONNECTOR_MSSQL_MANAGE_ROLE_LIST	This property is used to set comma separated role names which access control should be managed by policysync. If you want to manage all role then you can skip specifying this property. This supports wildcards as well. The ignore role list has precedence over manage role list. Eg. role1,role2,dev_role*.		
ADVANCED	CONNECTOR_MSSQL_IGNORE_USER_LIST	This property is used to set comma separated user names which access control you don't want to be managed by policysync. If you don't want to ignore any users then you can skip specifying this property. This supports wildcards as well. This has precedence over manage users list. Eg. user1,user2,dev_user*.		

Category	Property name	Description	Default	Allowable values
ADVANCED	CONNECTOR_MSSQL_IGNORE_GROUP_LIST	This property is used to set comma separated group names which access control you don't want to be managed by policysync. If you don't want to ignore any groups then you can skip specifying this property. This supports wildcards as well. This has precedence over manage groups list. Eg. `group1,group2,dev_group*`		
ADVANCED	CONNECTOR_MSSQL_IGNORE_ROLE_LIST	This property is used to set comma separated role names which access control you don't want to be managed by policysync. If you don't want to ignore any roles then you can skip specifying this property. This supports wildcards as well. This has precedence over manage roles list. Eg. `role1,role2,dev_role*`		
ADVANCED	CONNECTOR_MSSQL_USER_ROLE_PREFIX	This property is used to set a prefix for role which we will be creating in MSSQL for user from ranger. For example if you have user named john in ranger and you have defined prefix as test_user_ then the role which we create for john in Mssql will have name `test_user_john`.	`priv_user_`	
ADVANCED	CONNECTOR_MSSQL_GROUP_ROLE_PREFIX	This property is used to set a prefix for role which we will be creating in MSSQL for group from ranger. For example if you have group named dev in ranger and you have defined prefix as test_group_ then the role which we create for dev in Mssql will have name `test_group_dev`.	`priv_group_`	
ADVANCED	CONNECTOR_MSSQL_ROLE_ROLE_PREFIX	This property is used to set a prefix for role which we will be creating in MSSQL for role from ranger. For example if you have role named finance in ranger and you have defined prefix as test_role_ then the role which we create for finance in Mssql will have name `test_role_finance`.	`priv_role_`	
ADVANCED	CONNECTOR_MSSQL_USE_NATIVE_PUBLIC_GROUP	Set this property to true, if you want PolicySync to use mssql native public group for access grants whenever there is policy created referring to public group inside it.	`true`	`false`, `true`
ADVANCED	CONNECTOR_MSSQL_MANAGE_USER_FILTERBY_GROUP	Set this property to true, if you want to manage only the users who belongs the the groups defined in manage groups list property.	`false`	`true`, `false`
ADVANCED	CONNECTOR_MSSQL_MANAGE_USER_FILTERBY_ROLE	Set this property to true, if you want to manage only the users who belongs the the roles defined in manage roles list property.	`false`	`true`, `false`
Access control management				

Category	Property name	Description	Default	Allowable values
ADVANCED	CONNECTOR_MSSQL_ENABLE_ROW_FILTER	Set this property to true, if you want to enable native tr filter functionality. This is not recommended to use, since the native tr filters can only be created on tables, they can't be created on views.	true	false, true
ADVANCED	CONNECTOR_MSSQL_ENABLE_VIEW_BASED_MASKING	Set this property to true, if you want to enable secure view based masking in Mssql policysync. Note :- Mssql don't support native masking yet, thus recommended to use view based masking.	true	true, false
ADVANCED	CONNECTOR_MSSQL_ENABLE_VIEW_BASED_ROW_FILTER	Set this property to true, if you want to enable secure view based tr filter in Mssql policysync. Note :- Mssql support native tr filters, but due to its some limitations we recommended to use view based tr filter.	false	true, false
ADVANCED	CONNECTOR_MSSQL_SECURE_VIEW_CREATE_FOR_ALL	Set this property to true, if you want to create secure view for all tables as well all view which were created by end users. This will create secure view for resource regardless whether there any masking/tr filter policy exists in ranger.	false	true, false
ADVANCED	CONNECTOR_MSSQL_MASKED_NUMBER_VALUE	This property is used to specify the default masking value for numeric columns	0	
ADVANCED	CONNECTOR_MSSQL_MASKED_TEXT_VALUE	This property is used to specify the default masking value for text/string columns	<MASKED>'	
ADVANCED	CONNECTOR_MSSQL_SECURE_VIEW_NAME_PREFIX	By default view-based tr filter and masking related secure views have the same name as the table name with postfixed by _secure. If you want to change the secure view name prefix and postfix, that can be done with these properties. After prefix and postfix is specified the view name will be in this format : {prefix}{table_name}{postfix}.		
ADVANCED	CONNECTOR_MSSQL_SECURE_VIEW_NAME_POSTFIX	By default view-based tr filter and masking related secure views have the same name as the table name with postfixed by _secure. If you want to change the secure view name prefix and postfix, that can be done with these properties. After prefix and postfix is specified the view name will be in this format : {prefix}{table_name}{postfix}.	_secure	

Category	Property name	Description	Default	Allowable values
ADVANCED	CONNECTOR_MSSQL_SECURE_VIEW_SCHEMA_NAME_PREFIX	By default view-based tr filter and masking related secure views have the same schema name as the table schema name. If you want to change the secure view schema name prefix and postfix, that can be done with these properties. After prefix and postfix is specified the view schema name will be in this format : {prefix}{view_schema_name}{postfix} . For {view_schema_name} refer to variable MSSQL_SECURE_VIEW_SCHEMA_NAME		
ADVANCED	CONNECTOR_MSSQL_SECURE_VIEW_SCHEMA_NAME_POSTFIX	By default view-based tr filter and masking related secure views have the same schema name as the table schema name. If you want to change the secure view schema name prefix and postfix, that can be done with these properties. After prefix and postfix is specified the view schema name will be in this format : {prefix}{view_schema_name}{postfix}. For {view_schema_name} refer to variable Mssql_SECURE_VIEW_SCHEMA_NAME		
BASIC	CONNECTOR_MSSQL_GRANT_UPDATES	This property controls whether actual grant/revoke and create/update/delete queries for user/group/role should be run on Mssql.	true	true, false
	CONNECTOR_MSSQL_GRANT_UPDATES_MAX_RETRY_ATTEMPTS	This property is used to set max retry attempts to be made for granting or revoking the access in case if any failure due to database connection errors.	2	
ADVANCED	CONNECTOR_MSSQL_ENABLE_DATA_ADMIN	This property is used to enable data admin feature, with data admin feature enabled you can create all the policies on table/native view and by default respective grants will be made on secure view of table table or native view. And this secure view will have tr filter and masking capability as well. In case if you need permission on table then you can select the permission you want plus dataadmin in the policy, In this case that permissions will be granted on both, the table/native view and its secure view as well	true	true, false
Access audits management				
BASIC	CONNECTOR_MSSQL_AUDIT_ENABLE	This property is used to enable access audit fetching from Mssql	false	
BASIC-CONDITIONAL	CONNECTOR_MSSQL_AUDIT_STORAGE_URL	If this parameter is left empty or blank, Privacera Platform will target all databases attached to the MSSQL Server. If one or more database names are listed (comma separated values), only those databases will be controlled by Privacera Platform.		

Category	Property name	Description	Default	Allowable values
ADVANCED	CONNECTOR_MSSQL_AUDIT_EXCLUDED_USERS	This property is used to set the list of users whose access audits we want to ignore. It takes list of comma separated users.	CONNECTOR_MSSQL_JDBC_USERNAME	

PostgreSQL connector for PolicySync on Privacera Platform

This topic covers how you can configure PostgreSQL PolicySync access control using Privacera Manager. Privacera supports the following PostgreSQL implementations:

- Amazon RDS PostgreSQL
- Amazon Aurora in PostgreSQL mode
- Google Cloud PostgreSQL
- PostgreSQL

Generalized approach for implementing PolicySync

To help you reach compliance, Privacera PolicySync distributes your defined access management policies to the third-party datasources you connect to Privacera.

Use this generalized approach for implementing PolicySync.

1. Understand how PolicySync works and how it is configured. See PolicySync design and configuration on Privacera Platform [61].
2. Decide which PolicySync topology best suits your needs:
   - Required basic PolicySync topology: always at least one connector instance [64]
   - Optional topology: multiple connector instances for Kubernetes pods and Docker containers [65]
   - Recommended PolicySync topology: individual policy repositories for individual connectors [65]
3. Create the required, basic PolicySync configuration. See Required basic PolicySync topology: always at least one connector instance [64]
4. Examine the BASIC and ADVANCED properties, decide which features you want to implement, and set the necessary values in the YAML property file.

Connector name: postgres

When you create the connector as detailed in PolicySync design and configuration on Privacera Platform [61], use the following reserved word for the name of the connector:

```
postgres
```

In formal syntax shown in Connector instance directory/file structure [63] replace `<ConnectorName>` with the above.

Prerequisites

- Create a database in PostgreSQL and get the database name and its URL:
  - On Amazon RDS, see Creating a PostgreSQL DB instance.
  - On Google Cloud Platform, see the documentation for your PostgreSQL implementation:
    - Google Cloud AlloyDB for PostgreSQL
    - Google Cloud SQL PostgreSQL
- Create a database user granting all privileges to fully access the database, and then get the user credentials to connect to the database.

If you choose to enable audits for PolicySync, ensure the following prerequisites are met:

- On AWS, see Configure AWS RDS PostgreSQL instance for access audits [246]. If you are using multiple AWS accounts, see Access cross-account SQS queue for PostgreSQL audits on Privacera-Cloud.

- On GCP, see Accessing PostgreSQL Audits in GCP [250].

Optional Basic Authentication for PolicySync

To optionally enable basic authenticate for PolicySync to Google Cloud PostgreSQL you can create a JSON file in your connector instance subdirectory.

The name of the file must be `XXX.json`.

An example of the contents of `XXX.json`.:

```
 {
  "type": "service_account",
  "project_id": "your_project_id",
  "private_key_id": "autogenerated_value",
  "private_key": "-----BEGIN PRIVATE KEY-----autogenerated_value-----END PRIVATE KEY----
  "client_email": "autogenerated_value",
  "client_id": "autogenerated_value",
  "auth_uri": "https://accounts.google.com/o/oauth2/auth",
  "token_uri": "https://oauth2.googleapis.com/token",
  "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
  "client_x509_cert_url": "https://www.googleapis.com/robot/v1/metadata/x509/autogenerat
}
```

PostgreSQL connector properties for PolicySync on Privacera Platform

These PostgreSQL connector properties can be set for PolicySync in Privacera Platform.

The properties are grouped by general function, such as JDBC connection properties, properties for user, group, and role management, and other functions.

The properties are also categorized as **BASIC** or **ADVANCED**:

- **BASIC** pertains to the most fundamental aspects of the connector, such as authentication.
- **ADVANCED** indicates additional features beyond the **BASIC**s, such as row-filtering or group member handling.

Start by setting the **BASIC** properties and then examine the **ADVANCED** properties to determine which of these features you might want to enable.

For a general process to migrate values from old YAML files to the new YAML files, see Migration to PolicySync v2 on Privacera Platform 7.2 [68].

Category	Property name	Description	Default Value	Allowable values
JDBC configuration properties				
BASIC	`CONNECTOR_POSTGRES_JDBC_URL`	This property is used to set jdbc jdbc url which can be used to connect to postgres server. JDBC URL should follow below convention `jdbc:postgresql://<POSTGRES_SERVER_HOST>:<POSTGRES_SERVER_PORT>` Example :- `jdbc:postgresql://testdb.cxwi0ttzd22i.us-east-1.rds.amazonaws.com:5432`		
BASIC	`CONNECTOR_POSTGRES_JDBC_USERNAME`	This property is used to set jdbc Username to be used to make connection to postgres.		

Category	Property name	Description	Default Value	Allowable values
BASIC	CONNECTOR_POSTGRES_JDBC_PASSWORD	This property is used to set jdbc user's password to be used to make connection to postgres.		
BASIC	CONNECTOR_POSTGRES_JDBC_DB	This property is used to set jdbc database to be used to make initial connection to postgres.		
BASIC	CONNECTOR_POSTGRES_DEFAULT_USER_PASSWORD	This property is used to set password to every new user creation in postgres by PolicySync.		
BASIC	CONNECTOR_POSTGRES_OWNER_ROLE	This property is used to set ownership for all the resources managed by PolicySync. The specified role will become owner for all managed resources and will have full control on those resources.We support changing owners of database, schema, tables and views. **NOTE** If owner role is kept as blank, then ownership will not change and users who creates table/ view or any other object he will be owner of those objects and PolicySync won't be able to do access control on that object.		
Resources management				
BASIC	CONNECTOR_POSTGRES_MANAGE_DATABASE_LIST	This property is used to set comma separated database names which access control should be managed by PolicySync. If you want to manage all databases then you can skip specifying this property. This supports wildcards as well. The ignore database list has precedance over manage database list. For example: `testdb1,testdb2,sales_db*`. **NOTE** Values for this property are case-sensitive.		

Category	Property name	Description	Default Value	Allowable values
ADVANCED	CONNECTOR_POSTGRES_MANAGE_SCHEMA_LIST	This property is used to set comma separated schema Fqdn which access control should be managed by PolicySync. If you want to manage all schemas then you can skip specifying this property. This supports wildcards as well. The ignore schema list has precedance over manage schema list. For example: `testdb1.schema1,testdb2.schema2,sales_db*.sales*`. **NOTE** Values for this property are case-sensitive.		
ADVANCED	CONNECTOR_POSTGRES_MANAGE_TABLE_LIST	This property is used to set comma separated table/view Fqdn which access control should be managed by PolicySync. If you want to manage all tables/views then you can skip specifying this property. This supports wildcards as well. The ignore table list has precedance over manage table list. For example: `testdb1.schema1.table1,testdb2.schema2.view2,sales_db*.sales*.*` **NOTE** Values for this property are case-sensitive.		
ADVANCED	CONNECTOR_POSTGRES_IGNORE_DATABASE_LIST	This property is used to set comma separated database names which access control you don't want to be managed by PolicySync. If you don't want to ignore any database then you can skip specifying this property. This supports wildcards as well. This has precedance over manage database list. For example: `testdb1,testdb2,sales_db*`. **NOTE** Values for this property are case-sensitive.		

Category	Property name	Description	Default Value	Allowable values
ADVANCED	CONNECTOR_POSTGRES_IGNORE_SCHEMA_LIST	This property is used to set comma separated schema Fqdn which access control you don't want to be managed by PolicySync. If you don't want to ignore any schema then you can skip specifying this property. This supports wildcards as well. This has precedance over manage schema list. For example: `testdb1.schema1,testdb2.schema2,sales_db*.sales*`. **NOTE** Values for this property are case-sensitive.		
Users/Groups/Roles management				
ADVANCED	CONNECTOR_POSTGRES_USER_NAME_REPLACE_FROM_REGEX	This takes the regular expression as input and finds the matching characters in a user name and replaces them with the characters specified in property. If kept blank, no find and replace operation is performed.	`[~`$&+:;=?@#|'<>.^*()_%\\[\\]!\\-\\/\\\\{}]`	
ADVANCED	CONNECTOR_POSTGRES_USER_NAME_REPLACE_TO_STRING	The value specified in this property is used to replace the characters found by the regex specified user name regex property. If kept blank, no find and replace operation is performed.	_	
ADVANCED	CONNECTOR_POSTGRES_GROUP_NAME_REPLACE_FROM_REGEX	This takes the regular expression as input and finds the matching characters in a group name and replaces them with the characters specified in property. If kept blank, no find and replace operation is performed.	`[~`$&+:;=?@#|'<>.^*()_%\\[\\]!\\-\\/\\\\{}]`	
ADVANCED	CONNECTOR_POSTGRES_GROUP_NAME_REPLACE_TO_STRING	The value specified in this property is used to replace the characters found by the regex specified group name regex property. If kept blank, no find and replace operation is performed.	_	
ADVANCED	CONNECTOR_POSTGRES_ROLE_NAME_REPLACE_FROM_REGEX	This takes the regular expression as input and finds the matching characters in a role name and replaces them with the characters specified in property. If kept blank, no find and replace operation is performed.	`[~`$&+:;=?@#|'<>.^*()_%\\[\\]!\\-\\/\\\\{}]`	
ADVANCED	CONNECTOR_POSTGRES_ROLE_NAME_REPLACE_TO_STRING	The value specified in this property is used to replace the characters found by the regex specified role name regex property. If kept blank, no find and replace operation is performed.	_	
ADVANCED	CONNECTOR_POSTGRES_USER_NAME_PERSIST_CASE_SENSITIVITY	After loading user from Ranger API's all are converted into lowercase, but in some cases, you would need to have the users in the same case as they are in Ranger. When setting this value to true, it will maintain the case sensitivity of names as they are in Ranger.	`false`	`false`, `true`

Category	Property name	Description	Default Value	Allowable values
ADVANCED	CONNECTOR_POSTGRES_GROUP_NAME_PERSIST_CASE_SENSITIVITY	After loading group from Ranger API's all are converted into lowercase, but in some cases, you would need to have the users in the same case as they are in Ranger. When setting this value to true, it will maintain the case sensitivity of names as they are in Ranger.	false	true, false
ADVANCED	CONNECTOR_POSTGRES_ROLE_NAME_PERSIST_CASE_SENSITIVITY	After loading role from Ranger API's all are converted into lowercase, but in some cases, you would need to have the users in the same case as they are in Ranger. When setting this value to true, it will maintain the case sensitivity of names as they are in Ranger.	false	false, true
ADVANCED	CONNECTOR_POSTGRES_CREATE_USER	This property controls whether we should create user in postgres for users fecthed from ranger.	true	true, false
ADVANCED	CONNECTOR_POSTGRES_CREATE_USER_ROLE	This property controls whether we should create role over the end user in postgres for users fecthed from ranger.	true	false, true
ADVANCED	CONNECTOR_POSTGRES_MANAGE_USERS	This property controls whether PolicySync should manage the membership between user and user role.	true	true, false
ADVANCED	CONNECTOR_POSTGRES_MANAGE_GROUPS	This property controls whether we should create role in postgres for groups fecthed from ranger.	true	true, false
ADVANCED	CONNECTOR_POSTGRES_MANAGE_GROUP_MEMBERS	This property controls whether we should update the members of groups in Postgres for groups fetched from ranger.	true	true, false
ADVANCED	CONNECTOR_POSTGRES_MANAGE_ROLES	This property controls whether we should create role in postgres for roles fecthed from ranger.	true	false, true
ADVANCED	CONNECTOR_POSTGRES_MANAGE_ROLE_MEMBERS	This property controls whether we should update the members of roles in Postgres for roles fetched from ranger. For example,, if CONNECTOR_POSTGRES_MANAGE_ROLES is set to true, but CONNECTOR_POSTGRES_MANAGE_ROLE_MEMBERS is set to false, then PolicySync will create roles, but it won't add or remove members from those roles.	true	true, false
ADVANCED	CONNECTOR_POSTGRES_MANAGE_USER_LIST	This property is used to set comma separated user names which access control should be managed by PolicySync. If you want to manage all users then you can skip specifying this property. This supports wildcards as well. The ignore users list has precedance over manage users list. For example: user1,user2,dev_user*		
ADVANCED	CONNECTOR_POSTGRES_MANAGE_GROUP_LIST	This property is used to set comma separated group names which access control should be managed by PolicySync. If you want to manage all group then you can skip specifying this property. This supports wildcards as well. The ignore group list has precedance over manage group list. For example: group1,group2,dev_group*		

Category	Property name	Description	Default Value	Allowable values
ADVANCED	CONNECTOR_POSTGRES_MANAGE_ROLE_LIST	This property is used to set comma separated role names which access control should be managed by PolicySync. If you want to manage all role then you can skip specifying this property. This supports wildcards as well. The ignore role list has precedance over manage role list. For example: role1,role2,dev_role*		
ADVANCED	CONNECTOR_POSTGRES_IGNORE_USER_LIST	This property is used to set comma separated user names which access control you don't want to be managed by PolicySync. If you don't want to ignore any users then you can skip specifying this property. This supports wildcards as well. This has precedance over manage users list. For example: user1,user2,dev_user*		
ADVANCED	CONNECTOR_POSTGRES_IGNORE_GROUP_LIST	This property is used to set comma separated group names which access control you don't want to be managed by PolicySync. If you don't want to ignore any groups then you can skip specifying this property. This supports wildcards as well. This has precedance over manage groups list. For example: group1,group2,dev_group*		
ADVANCED	CONNECTOR_POSTGRES_IGNORE_ROLE_LIST	This property is used to set comma separated role names which access control you don't want to be managed by PolicySync. If you don't want to ignore any roles then you can skip specifying this property. This supports wildcards as well. This has precedance over manage roles list. For example: role1,role2,dev_role*		
ADVANCED	CONNECTOR_POSTGRES_USER_ROLE_PREFIX	This property is used to set a prefix for role which we will be creating in postgres for user from ranger. For example, if you have user named john in ranger and you have defined prefix as test_user_ then the role which we create for john in postgres will have name test_user_john	priv_user_	
ADVANCED	CONNECTOR_POSTGRES_GROUP_ROLE_PREFIX	This property is used to set a prefix for role which we will be creating in postgres for group from ranger. For example, if you have group named dev in ranger and you have defined prefix as test_group_ then the role which we create for dev in postgres will have name test_group_dev.	priv_group_	
ADVANCED	CONNECTOR_POSTGRES_ROLE_ROLE_PREFIX	This property is used to set a prefix for role which we will be creating in postgres for role from ranger. For example, if you have role named finance in ranger and you have defined prefix as test_role_ then the role which we create for finance in postgres will have name test_role_finance.	priv_role_	
ADVANCED	CONNECTOR_POSTGRES_USE_NATIVE_PUBLIC_GROUP	Set this property to true, if you want PolicySync to use porstgres native public group for access grants whenever there is policy created referring to public group inside it.	true	true, false

Category	Property name	Description	Default Value	Allowable values
ADVANCED	CONNECTOR_POSTGRES_MANAGE_USER_FILTERBY_GROUP	Set this property to true, if you want to manage only the users who belongs the the groups defined in manage groups list property.	false	true, false
ADVANCED	CONNECTOR_POSTGRES_MANAGE_USER_FILTERBY_ROLE	Set this property to true, if you want to manage only the users who belongs the roles defined in manage roles list property.	false	true, false
Access control management				
ADVANCED	CONNECTOR_POSTGRES_ENABLE_ROW_FILTER	Set this property to true, if you want to enable native tr filter functionality. This is not recommend to use, since the native tr filters can only be created on tables, they can't be created on views.	false	true, false
ADVANCED	CONNECTOR_POSTGRES_ENABLE_VIEW_BASED_MASKING	Set this property to true, if you want to enable secure view based masking in postgres PolicySync. **NOTE** Postgres don't support native masking yet, thus recommended to use view based masking.	true	true, false
ADVANCED	CONNECTOR_POSTGRES_ENABLE_VIEW_BASED_ROW_FILTER	Set this property to true, if you want to enable secure view based tr filter in postgres PolicySync. **NOTE** Postgres support native tr filters, but due to its some limitations we recommened to use view based tr filter.	true	true, false
ADVANCED	CONNECTOR_POSTGRES_SECURE_VIEW_CREATE_FOR_ALL	Set this property to true, if you want to create secure view for all tables as well all view which were created by end users. This will create secure view for resource regardless whether there any masking/tr filter policy exists in ranger.	true	true, false
ADVANCED	CONNECTOR_POSTGRES_MASKED_NUMBER_VALUE	This property is used to specify the default masking value for numeric columns	0	
ADVANCED	CONNECTOR_POSTGRES_MASKED_TEXT_VALUE	This property is used to specify the default masking value for text/string columns	<MASKED>'	

Category	Property name	Description	Default Value	Allowable values
ADVANCED	CONNECTOR_POSTGRES_SECURE_VIEW_NAME_PREFIX	By default view-based tr filter and masking related secure views have the same name as the table name with postfixed by _secure. If you want to change the secure view name prefix and postfix, that can be done with these properties. After prefix and postfix is specified the view name will be in this format : {prefix}{table_name}{postfix}		
ADVANCED	CONNECTOR_POSTGRES_SECURE_VIEW_NAME_POSTFIX	By default view-based tr filter and masking related secure views have the same name as the table name with postfixed by _secure. If you want to change the secure view name prefix and postfix, that can be done with these properties. After prefix and postfix is specified the view name will be in this format : {prefix}{table_name}{postfix}	_secure	
ADVANCED	CONNECTOR_POSTGRES_SECURE_VIEW_SCHEMA_NAME_PREFIX	By default view-based tr filter and masking related secure views have the same schema name as the table schema name. If you want to change the secure view schema name prefix and postfix, that can be done with these properties. After prefix and postfix is specified the view schema name will be in this format : {prefix}{view_schema_name}{postfix} For {view_schema_name} refer to variable CONNECTOR_POSTGRES_SECURE_VIEW_SCHEMA_NAME.		
ADVANCED	CONNECTOR_POSTGRES_SECURE_VIEW_DATABASE_NAME_PREFIX	By default view-based tr filter and masking related secure views have the same database name as the table database name. If you want to change the secure view database name prefix and postfix, that can be done with these properties. After prefix and postfix is specified the view database name will be in this format : {prefix}{view_database_name}{postfix} For {view_database_name} refer to variable CONNECTOR_POSTGRES_SECURE_VIEW_DATABASE_NAME.		
ADVANCED	CONNECTOR_POSTGRES_SECURE_VIEW_SCHEMA_NAME_POSTFIX	By default view-based tr filter and masking related secure views have the same schema name as the table schema name. If you want to change the secure view schema name prefix and postfix, that can be done with these properties. After prefix and postfix is specified the view schema name will be in this format : {prefix}{view_schema_name}{postfix} For {view_schema_name} refer to variable POSTGRES_SECURE_VIEW_SCHEMA_NAME.		

Category	Property name	Description	Default Value	Allowable values
ADVANCED	CONNECTOR_POSTGRES_SECURE_VIEW_DATABASE_NAME_POSTFIX	By default view-based tr filter and masking related secure views have the same database name as the table database name. If you want to change the secure view database name prefix and postfix, that can be done with these properties. After prefix and postfix is specified the view database name will be in this format : {prefix}{view_database_name}{postfix} For {view_schema_name} refer to variable POSTGRES_SECURE_VIEW_DATABASE_NAME.		
BASIC	CONNECTOR_POSTGRES_GRANT_UPDATES	This property controls whether actual grant/revoke and create/update/delete queries for user/group/role should be run on postgres.	true	true, false
	CONNECTOR_POSTGRES_GRANT_UPDATES_MAX_RETRY_ATTEMPTS	This property is used to set max retry attemps to be made for granting or revoking the access in case if any failure due to database connection errors.	2	
ADVANCED	CONNECTOR_POSTGRES_ENABLE_DATA_ADMIN	This propery is used to enable data admin feature, with data admin feature enabled you can create all the policies on table/native view and by default repspective grants will be made on secure view of table or native view. This secure view will have tr filter and masking capability as well. In case if you need permission on table then you can select the permission you want plus dataadmin in the policy, in this case that permissions will be granted on both, the table/native view and its secure view as well.	true	true, false
Access audits management				
BASIC	CONNECTOR_POSTGRES_AUDIT_ENABLE	This property is used to enable access audit fetching from postgres.	false	false, true
ADVANCED	CONNECTOR_POSTGRES_AUDIT_EXCLUDED_USERS	This property is used to set the list of users whose access audits we want to ignore. It takes list of comma separated users.		
BASIC	CONNECTOR_POSTGRES_AUDIT_SOURCE	This property is used to set the mode through which we should be loading access audits, this depends on the your Postgres server deployment. If you are using AWS RDS Postgres then you need to set this value to sqs. If you are using GCP Postgres then you need to set this to gcp_pgaudit. If you are using Azure Postgres then you need to set this to azure_audit.	sqs	sqs/ gcp_pgaudit

Category	Property name	Description	Default Value	Allowable values
ADVANCED	CONNECTOR_POSTGRES_AUDIT_ENABLE_RESOURCE_FILTER	This property is used to filter access audit by managed resources (database,schema list) NOTE: Policysync is using sql parser to extract resourceName to decide if it's manage or not and there is chance sql query parseser can break in case of complex query and it will end up without logging that audit.	FALSE	true, false
BASICCONDITIONAL	CONNECTOR_POSTGRES_AWS_ACCESS_KEY	This property is used to set the aws access key which will be used to created iam client in order to access sqs queue to get access audits. This should be used only if your deployment machine don't have IAM role associated with it with required permissions		
BASICCONDITIONAL	CONNECTOR_POSTGRES_AWS_SECRET_KEY	This property is used to set the aws secret access key which will be used to created iam client in order to access sqs queue to get access audits. This should be used only if your deployment machine don't have IAM role associated with it with required permissions		
BASICCONDITIONAL	CONNECTOR_POSTGRES_AWS_REGION	This property is used to set the aws region in which sqs queue resides in.	us-east-1	
BASICCONDITIONAL	CONNECTOR_POSTGRES_AWS_SQS_QUEUE_NAME	This property is used to set aws sqs queue name from which we need to get access audits.		
BASICCONDITIONAL	CONNECTOR_POSTGRES_GCP_AUDIT_SOURCE_INSTANCE_ID	This property is used to set the instance id of gcp cloudsql postgres server which will be used to get the access audits. This instance id value needs to be in the format project_id:db_instance_id. For example, demo-project:postgres-demo-server		
BASICCONDITIONAL	CONNECTOR_POSTGRES_AZURE_CLIENT_ID	This property is used to build client credential in order to run *kusto* query		
BASICCONDITIONAL	CONNECTOR_POSTGRES_AZURE_TENANT_ID	This property is used to build client credential in order to run *kusto* query		
BASICCONDITIONAL	CONNECTOR_POSTGRES_AZURE_CLIENT_SECRET_VALUE	This property is used to build client credential in order to run *kusto* query		
BASICCONDITIONAL	CONNECTOR_POSTGRES_AZURE_WORKSPACE_ID	This property is used to specify workspace where *kusto* query needs to be executed		

Power BI connector for PolicySync

This section covers how to enable configure Privacera Power BI connector for workspace fine-grained access-control in Power BI running in Azure. You can set permissions in a Privacera policy depending on the workspace roles : Admin, Member, Contributor, Viewer. Only users and groups from the Azure Active Directory are allowed in Azure Power BI.

Generalized approach for implementing PolicySync

To help you reach compliance, Privacera PolicySync distributes your defined access management policies to the third-party datasources you connect to Privacera.

Use this generalized approach for implementing PolicySync.

1. Understand how PolicySync works and how it is configured. See PolicySync design and configuration on Privacera Platform [61].
2. Decide which PolicySync topology best suits your needs:
   - Required basic PolicySync topology: always at least one connector instance [64]
   - Optional topology: multiple connector instances for Kubernetes pods and Docker containers [65]
   - Recommended PolicySync topology: individual policy repositories for individual connectors [65]
3. Create the required, basic PolicySync configuration. See Required basic PolicySync topology: always at least one connector instance [64]
4. Examine the BASIC and ADVANCED properties, decide which features you want to implement, and set the necessary values in the YAML property file.

Connector name: powerbi

When you create the connector as detailed in PolicySync design and configuration on Privacera Platform [61], use the following reserved word for the name of the connector:

```
powerbi
```

In formal syntax shown in Connector instance directory/file structure [63] replace `<ConnectorName>` with the above and in the example in Required basic PolicySync topology: always at least one connector instance [64], replace `postgres` with the above.

Prerequisites

Ensure that the following prerequisites are met:

1. Create a *service principal* and *application secret* for the Power BI, and get the following information from Azure Portal. For more information, refer the Microsoft Azure documentation.
   - Application (client) ID
   - Directory (tenant) ID
   - Client Secret
2. Create a group to assign your created Power BI application to it. This is required because the Power BI *Admin API* allows only the *service principal* to be an Azure AD Group.
   Follow the steps in the link given above, and configure the following to create a group and add Power BI as a member:
   a. On the **New Group** dialog, select `security` in the **Group type**, and then add the required group details.
   b. Click **Create**.
   c. On the **+Add members** dialog, select your Power BI application.
3. Configure Power BI Tenant to allow Power BI *service principals* to read the *REST API*.
   Follow the steps in the link given above and configure the following:
   a. In the **Developer settings**, enable **Allow service principals to use Power BI APIs**.
   b. Select **Specific security groups (Recommended)**, and then add the Power BI group you created above.
   c. In the **Admin API Settings**, enable **Allow service principals to use read-only Power BI admin APIs (Preview)**. For more information, see the Microsoft Azure documentation - click here.
   d. Select **Specific security groups**, and then add the Power BI group you created above.
4. Enable Privacera UserSync for AAD to pull groups *attribute ID* via the `AZURE_AD_ATTRIBUTE_GROUPNAME` property described in AAD UserSync connector properties [25].

Power BI connector properties for PolicySync on Privacera Platform

These Power BI connector properties can be set for PolicySync in Privacera Platform.

The properties are grouped by general function, such as JDBC connection properties, properties for user, group, and role management, and other functions.

The properties are also categorized as **BASIC** or **ADVANCED**:

- **BASIC** pertains to the most fundamental aspects of the connector, such as authentication.
- **ADVANCED** indicates additional features beyond the **BASIC**s, such as row-filtering or group member handling.

Start by setting the **BASIC** properties and then examine the **ADVANCED** properties to determine which of these features you might want to enable.

For a general process to migrate values from old YAML files to the new YAML files, see Migration to PolicySync v2 on Privacera Platform 7.2 [68].

Category	Property name	Description	Default	Allowable values
Connection configuration properties				
BASIC	`CONNECTOR_POWER_BI_USERNAME`	Username for authentication with Power BI. For authentication either username/password or client secret is needed		
BASIC	`CONNECTOR_POWER_BI_PASSWORD`	Password for authentication with Power BI. For authentication either username/password or client secret is needed		
BASIC	`CONNECTOR_POWER_BI_TENANT_ID`	Tenant Id is needed for authentication. To get the value for this property in the Azure portal, go to **Azure Active Directory > Properties > Tenant ID**.		
BASIC	`CONNECTOR_POWER_BI_CLIENT_ID`	Client Id is needed For authentication. To get the value for this property in the Azure portal, go to **Azure Active Directory > Properties > Client (Application) ID**.		
BASIC	`CONNECTOR_POWER_BI_CLIENT_SECRET`	Client Secret for authentication with Power BI.		
Resources management				
BASIC	`CONNECTOR_POWER_BI_MANAGE_WORKSPACE_LIST`	Add the names of the workspaces to be managed. Only these workspaces will be provided with access control in a Power BI policy. Regular expression can be used. For example: `demo*` (This will manage all the workspaces named as demo1 ,demo2, etc). **NOTE** Values for this property are case-sensitive.		
ADVANCED	`CONNECTOR_POWER_BI_IGNORE_WORKSPACE_LIST`	Add the names of the workspaces to be ignored. These workspaces will not be provided with access control in a Power BI policy. **NOTE** Values for this property are case-sensitive.		
Users/Groups/Roles management				
ADVANCED	`CONNECTOR_POWER_BI_USER_NAME_REPLACE_FROM_REGEX`	This takes the regular expression as input and finds the matching characters in a user name and replaces them with the characters specified in property. If kept blank, no find and replace operation is performed.	`[^a-zA-Z0-9@.\\\\s]`	

Category	Property name	Description	Default	Allowable values
ADVANCED	CONNECTOR_POWER_BI_USER_NAME_REPLACE_TO_STRING	The value specified in this property is used to replace the characters found by the regex specified user name regex property. If kept blank, no find and replace operation is performed.	_	
ADVANCED	CONNECTOR_POWER_BI_GROUP_NAME_REPLACE_FROM_REGEX	This takes the regular expression as input and finds the matching characters in a group name and replaces them with the characters specified in property. If kept blank, no find and replace operation is performed.	[^a-zA-Z0-9@.\\\\s]	
ADVANCED	CONNECTOR_POWER_BI_GROUP_NAME_REPLACE_TO_STRING	The value specified in this property is used to replace the characters found by the regex specified group name regex property. If kept blank, no find and replace operation is performed.	_	
ADVANCED	CONNECTOR_POWER_BI_USER_NAME_PERSIST_CASE_SENSITIVITY	After loading users from Ranger API's all are converted into lowercase, but in some cases, you would need to have the users in the same case as they are in Ranger. When setting this value to true, it will maintain the case sensitivity of names as they are in Ranger.	false	
ADVANCED	CONNECTOR_POWER_BI_GROUP_NAME_PERSIST_CASE_SENSITIVITY	After loading groups from Ranger API's all are converted into lowercase, but in some cases, you would need to have the users in the same case as they are in Ranger. When setting this value to true, it will maintain the case sensitivity of names as they are in Ranger.	false	
ADVANCED	CONNECTOR_POWER_BI_MANAGE_USER_LIST	This property is used to set comma separated user names which access control should be managed by policysync. If you want to manage all users then you can skip specifying this property. This supports wildcards as well. The ignore users list has precedence over manage users list. For example: user1,user2,dev_user*		
ADVANCED	CONNECTOR_POWER_BI_MANAGE_GROUP_LIST	This property is used to set comma separated group names which access control should be managed by policysync. If you want to manage all groups then you can skip specifying this property. This supports wildcards as well. The ignore group list has precedence over manage group list. For example: group1,group2,dev_group*		
ADVANCED	CONNECTOR_POWER_BI_IGNORE_USER_LIST	This property is used to set comma separated user names which access control you don't want to be managed by policysync. If you don't want to ignore any users then you can skip specifying this property. This supports wildcards as well. This has precedence over manage users list. For example: user1,user2,dev_user*		
ADVANCED	CONNECTOR_POWER_BI_IGNORE_GROUP_LIST	This property is used to set comma separated group names which access control you don't want to be managed by policysync. If you don't want to ignore any groups then you can skip specifying this property. This supports wildcards as well. This has precedence over manage groups list. For example: group1,group2,dev_group*		
	CONNECTOR_POWER_BI_USER_FILTER_WITH_EMAIL	Set this property to true, if you want to manage only the users who contain the email field not blank.	false	
ADVANCED	CONNECTOR_POWER_BI_MANAGE_USER_FILTERBY_GROUP	Set this property to true, if you want to manage only the users who belong to the groups defined in manage groups list property.	false	
Access control management				
BASIC	CONNECTOR_POWER_BI_GRANT_UPDATES	This property controls whether actual grant/revoke and create/update/delete queries for user/group/role should be run on power bi.	true	

Category	Property name	Description	Default	Allowable values
Access audits management				
BASIC	`CONNECTOR_POWER_BI_ENABLE_AUDIT`	This property is used to enable access audit fetching from power bi.	`false`	

Redshift and Redshift Spectrum connector for PolicySync

This topic details how to configure PolicySync for Redshift and .

Generalized approach for implementing PolicySync

Use this generalized approach for implementing PolicySync.

1. Understand how PolicySync works and how it is configured. See PolicySync design and configuration on Privacera Platform [61].
2. Decide which PolicySync topology best suits your needs:
   - Required basic PolicySync topology: always at least one connector instance [64]
   - Optional topology: multiple connector instances for Kubernetes pods and Docker containers [65]
   - Recommended PolicySync topology: individual policy repositories for individual connectors [65]
3. Create the required, basic PolicySync configuration. See Required basic PolicySync topology: always at least one connector instance [64]
4. Examine the BASIC and ADVANCED properties, decide which features you want to implement, and set the necessary values in the YAML property file.

Connector name: redshift

When you create the connector as detailed in PolicySync design and configuration on Privacera Platform [61], use the following reserved word for the name of the connector:

```
redshift
```

In formal syntax shown in Connector instance directory/file structure [63] replace `<ConnectorName>` with the above and in the example in Required basic PolicySync topology: always at least one connector instance [64], replace `postgres` with the above.

Redshift Spectrum configuration and security considerations

Redshift Spectrum configuration is similar to Redshift configuration.

Privacera supports access control for Redshift Spectrum only on the following:

- Create Database
- Usage Schema

Prerequisites

The following prerequisites must be met to use Redshift Spectrum :

1. You will require an Amazon Redshift cluster and a SQL client connected to the cluster.
2. The AWS Region in which the Amazon Redshift cluster and Amazon S3 bucket are located must be the same.

Set-up in AWS for Redshift Spectrum

Redshift Spectrum supports the creation of external tables in a Redshift cluster. To configure external tables in Redshift Spectrum, see the following AWS documentation:

1. Create an IAM role for Amazon Redshift.

2. Associate the IAM role with your Amazon Redshift cluster.
3. Create an external schema and an external table.
4. Query your data in Amazon S3.

Important security considerations for external tables and schemas

CAUTION

Be advised that row-level filter and column masking via secure views on EXTERNAL SCHEMA gives a user direct access to the EXTERNAL TABLE. If a user queries the original external table, row-level filter and column masking are not applied.

Redshift does not natively support access control lists (ACLs) on EXTERNAL TABLES. To restrict access to the data. You must set USAGE schema permission on an associated EXTERNAL SCHEMA.

On an EXTERNAL TABLE, Privacera supports row-level filter and column masking to a limited extent. Privacera cannot manage external tables. By default, Privacera manages permissions for external schemas only at the schema level.

- Because Redshift views inside external schemas cannot be created, instead of creating a table, Privacera creates a secure view with the name of the schema with a default postfix `_secure`. For example, if the original view is named CUSTOMER, Privacera creates a secure view named `CUSTOMER_secure`.
- To GRANT access to the secure view, Privacera must grant USAGE permission to the source schema because the secure view schema is separated from the EXTERNAL SCHEMA. As a result, permission is granted to the original source table.
- Only SELECT permission to the EXTERNAL TABLE is supported. `DataAdmin` permission is ineffective because USAGE permission to EXTERNAL SCHEMA allows direct access to EXTERNAL TABLE.

Enable EXTERNAL SCHEMA in Privacera

NOTE

Because of security concerns for EXTERNAL SCHEMA, Privacera does not recommend enabling row-level filter and column masking. Be sure you understand the Important security considerations for external tables and schemas [142].

Property	Description	Default Value	Example
`CONNECTOR_REDSHIFT_ENABLE_EXTERNAL_SCHEMA_SUPPORT`	Set this property to `true` to enable row-level filter and column masking policies on secure views after reading the limitations.	`false`	`true/false`

The values of the following properties must be left blank:

```
CONNECTOR_REDSHIFT_SECURE_VIEW_NAME_PREFIX: ""
CONNECTOR_REDSHIFT_SECURE_VIEW_NAME_POSTFIX: ""
```

The values of the following properties must be set:

```
CONNECTOR_REDSHIFT_SECURE_VIEW_SCHEMA_NAME_PREFIX: ""
CONNECTOR_REDSHIFT_SECURE_VIEW_SCHEMA_NAME_POSTFIX: "_secure"
```

Redshift and Redshift Spectrum connector properties for PolicySync on Privacera Platform

These Redshift and Redshift Spectrum connector properties can be set for PolicySync in Privacera Platform.

The properties are grouped by general function, such as JDBC connection properties, properties for user, group, and role management, and other functions.

The properties are also categorized as **BASIC** or **ADVANCED**:

- **BASIC** pertains to the most fundamental aspects of the connector, such as authentication.
- **ADVANCED** indicates additional features beyond the **BASIC**s, such as row-filtering or group member handling.

Start by setting the **BASIC** properties and then examine the **ADVANCED** properties to determine which of these features you might want to enable.

For a general process to migrate values from old YAML files to the new YAML files, see Migration to PolicySync v2 on Privacera Platform 7.2 [68].

Category	Property name	Description	Default Value	Allowable values
JDBC configuration properties				
BASIC	CONNECTOR_REDSHIFT_JDBC_URL	This property is used to set jdbc jdbc url which can be used to connect to Redshift server. JDBC URL should follow below convention: jdbc:redshift://<REDSHIFT_SERVER_HOST>:<REDSHIFT_SERVER_PORT>. For example: jdbc:redshift://dev-redshift.xxxxxxxxxxx.us-east-1.redshift.amazonaws.com:5439.		
BASIC	CONNECTOR_REDSHIFT_JDBC_USERNAME	This property is used to set jdbc Username to be used to make connection to Redshift.		
BASIC	CONNECTOR_REDSHIFT_JDBC_PASSWORD	This property is used to set jdbc username password to be used to make connection to Redshift.		
BASIC	CONNECTOR_REDSHIFT_JDBC_DB	This property is used to set jdbc database to be used to make initial connection to Redshift.		
BASIC	CONNECTOR_REDSHIFT_DEFAULT_USER_PASSWORD	This property is used to set password which will be used for every new user creation by PolicySync.		
BASIC	CONNECTOR_REDSHIFT_OWNER_ROLE	This property is used to set ownership for all the resources managed by PolicySync. The specified role will become owner for all managed resources and will have full control on those resources. We support changing owners of database, schema, tables and views. Note :- If owner role is kept as blank, then ownership will not change and users who creates table/view or any other object he will be owner of those objects and PolicySync won't be able to do access control on that object		

Category	Property name	Description	Default Value	Allowable values
Resources management				
BASIC	`CONNECTOR_REDSHIFT_MANAGE_DATABASE_LIST`	This property is used to set comma separated database names which access control should be managed by PolicySync. If you want to manage all databases then you can skip specifying this property. This supports wildcards as well. The ignore database list has precedence over manage database list. For example: `testdb1,testdb2,sales_db*`. In V1, if you want to manage all databases you have to set this property to blank, this is property behaviour change from V1 to V2. **NOTE** Values for this property are case-sensitive.		
ADVANCED	`CONNECTOR_REDSHIFT_MANAGE_SCHEMA_LIST`	This property is used to set comma separated schema Fqdn which access control should be managed by PolicySync. If you want to manage all schemas then you can skip specifying this property. This supports wildcards as well. The ignore schema list has precedence over manage schema list. For example: `testdb1.schema1,testdb2.schema2,sales_db*.sales*`. In V1, if you want to manage all schemas you have to set this property to blank, this is property behaviour change from V1 to V2. **NOTE** Values for this property are case-sensitive.		

Category	Property name	Description	Default Value	Allowable values
ADVANCED	CONNECTOR_REDSHIFT_MANAGE_TABLE_LIST	This property is used to set comma separated table/view Fqdn which access control should be managed by PolicySync. If you want to manage all tables/views then you can skip specifying this property. This supports wildcards as well. The ignore table list has precedence over manage table list. For example: `testdb1.schema1.table1,testdb2.schema2.view2,sales_db*.sales*.*`. In V1, if you want to manage all tables/views you have to set this property to blank, this is property behaviour change from V1 to V2. **NOTE** Values for this property are case-sensitive.		
ADVANCED	CONNECTOR_REDSHIFT_IGNORE_DATABASE_LIST	This property is used to set comma separated database names which access control you don't want to be managed by PolicySync. If you don't want to ignore any database then you can skip specifying this property. This supports wildcards as well. This has precedence over manage database list. For example: `testdb1,testdb2,sales_db*`. **NOTE** Values for this property are case-sensitive.		
ADVANCED	CONNECTOR_REDSHIFT_IGNORE_SCHEMA_LIST	This property is used to set comma separated schema fqdn which access control you don't want to be managed by PolicySync. If you don't want to ignore any schema then you can skip specifying this property. This supports wildcards as well. This has precedence over manage schema list. For example: `testdb1.schema1,testdb2.schema2,sales_db*.sales*`. **NOTE** Values for this property are case-sensitive.		
Users/Groups/Roles management				

Category	Property name	Description	Default Value	Allowable values
ADVANCED	CONNECTOR_REDSHIFT_USER_NAME_REPLACE_FROM_REGEX	This takes the regular expression as input and finds the matching characters in a user name and replaces them with the characters specified in property. If kept blank, no find and replace operation is performed.	[~`$&+:;=?@#|'<>.^*_%\\[\\]!\\-\\/\\\\{}]	
ADVANCED	CONNECTOR_REDSHIFT_USER_NAME_REPLACE_TO_STRING	The value specified in this property is used to replace the characters found by the regex specified user name regex property. If kept blank, no find and replace operation is performed.	_	
ADVANCED	CONNECTOR_REDSHIFT_GROUP_NAME_REPLACE_FROM_REGEX	This takes the regular expression as input and finds the matching characters in a group name and replaces them with the characters specified in property. If kept blank, no find and replace operation is performed.	[~`$&+:;=?@#|'<>.^*_%\\[\\]!\\-\\/\\\\{}]	
ADVANCED	CONNECTOR_REDSHIFT_GROUP_NAME_REPLACE_TO_STRING	The value specified in this property is used to replace the characters found by the regex specified group name regex property. If kept blank, no find and replace operation is performed.	_	
ADVANCED	CONNECTOR_REDSHIFT_ROLE_NAME_REPLACE_FROM_REGEX	This takes the regular expression as input and finds the matching characters in a role name and replaces them with the characters specified in property. If kept blank, no find and replace operation is performed.	[~`$&+:;=?@#|'<>.^*_%\\[\\]!\\-\\/\\\\{}]	
ADVANCED	CONNECTOR_REDSHIFT_ROLE_NAME_REPLACE_TO_STRING	The value specified in this property is used to replace the characters found by the regex specified role name regex property. If kept blank, no find and replace operation is performed.	_	
ADVANCED	CONNECTOR_REDSHIFT_USER_NAME_PERSIST_CASE_SENSITIVITY	After loading user from Ranger API's all are converted into lowercase, but in some cases, you would need to have the users in the same case as they are in Ranger. When setting this value to true, it will maintain the case sensitivity of names as they are in Ranger.	false	
ADVANCED	CONNECTOR_REDSHIFT_GROUP_NAME_PERSIST_CASE_SENSITIVITY	After loading group from Ranger API's all are converted into lowercase, but in some cases, you would need to have the groups in the same case as they are in Ranger. When setting this value to true, it will maintain the case sensitivity of names as they are in Ranger.	false	
ADVANCED	CONNECTOR_REDSHIFT_ROLE_NAME_PERSIST_CASE_SENSITIVITY	After loading role from Ranger API's all are converted into lowercase, but in some cases, you would need to have the roles in the same case as they are in Ranger. When setting this value to true, it will maintain the case sensitivity of names as they are in Ranger.	false	
ADVANCED	CONNECTOR_REDSHIFT_ENABLE_CASE_SENSITIVE_IDENTIFIER	By default the Redshift converts all the resources to lower case, to persist the case we have a property to set on each and every connection. If you follow the same please set this value as true.	false	
ADVANCED	CONNECTOR_REDSHIFT_ENABLE_CASE_SENSITIVE_IDENTIFIER_QUERY	If CONNECTOR_REDSHIFT_ENABLE_CASE_SENSITIVE_IDENTIFIER is set to true the query will be fired at first in each and every connection.	SET enable_case_sensitive_identifier=true;	

Category	Property name	Description	Default Value	Allowable values
ADVANCED	CONNECTOR_REDSHIFT_CREATE_USER	This property controls whether we should create user in Redshift for users fetched from ranger.	true	
ADVANCED	CONNECTOR_REDSHIFT_CREATE_USER_ROLE	This property controls whether we should create role over the end user in Redshift for users fetched from ranger.	true	
ADVANCED	CONNECTOR_REDSHIFT_MANAGE_USERS	This property controls whether PolicySync should manage the membership between user and user role.	true	
ADVANCED	CONNECTOR_REDSHIFT_MANAGE_GROUPS	This property controls whether we should create role in Redshift for groups fetched from ranger.	true	
ADVANCED	CONNECTOR_REDSHIFT_MANAGE_GROUP_MEMBERS	This property controls whether we should update the members of groups in Redshift for groups fetched from ranger.	true	
ADVANCED	CONNECTOR_REDSHIFT_MANAGE_ROLES	This property controls whether we should create role in Redshift for roles fetched from ranger.	true	
ADVANCED	CONNECTOR_REDSHIFT_MANAGE_ROLE_MEMBERS	This property controls whether we should update the members of roles in Redshift for roles fetched from ranger.	true	
ADVANCED	CONNECTOR_REDSHIFT_MANAGE_USER_LIST	This property is used to set comma separated user names which access control should be managed by PolicySync. If you want to manage all users then you can skip specifying this property. This supports wildcards as well. The ignore users list has precedence over manage users list. For example: `user1,user2,dev_user*`. In V1, if you want to manage all users you have to set this property to blank, this is property behaviour change from V1 to V2."		
ADVANCED	CONNECTOR_REDSHIFT_MANAGE_GROUP_LIST	This property is used to set comma separated group names which access control should be managed by PolicySync. If you want to manage all group then you can skip specifying this property. This supports wildcards as well. The ignore group list has precedence over manage group list. For example: `group1,group2,dev_group*` In V1, if you want to manage all group you have to set this property to blank, this is property behaviour change from V1 to V2.		
ADVANCED	CONNECTOR_REDSHIFT_MANAGE_ROLE_LIST	This property is used to set comma separated role names which access control should be managed by PolicySync. If you want to manage all role then you can skip specifying this property. This supports wildcards as well. The ignore role list has precedence over manage role list. For example: `role1,role2,dev_role*` In V1, if you want to manage all roles you have to set this property to blank, this is property behaviour change from V1 to V2.		

Category	Property name	Description	Default Value	Allowable values
ADVANCED	CONNECTOR_REDSHIFT_IGNORE_USER_LIST	This property is used to set comma separated user names which access control you don't want to be managed by PolicySync. If you don't want to ignore any users then you can skip specifying this property. This supports wildcards as well. This has precedence over manage users list. For example: `user1,user2,dev_user*`		
ADVANCED	CONNECTOR_REDSHIFT_IGNORE_GROUP_LIST	This property is used to set comma separated group names which access control you don't want to be managed by PolicySync. If you don't want to ignore any groups then you can skip specifying this property. This supports wildcards as well. This has precedence over manage groups list. For example: `group1,group2,dev_group*`.		
ADVANCED	CONNECTOR_REDSHIFT_IGNORE_ROLE_LIST	This property is used to set comma separated role names which access control you don't want to be managed by PolicySync. If you don't want to ignore any roles then you can skip specifying this property. This supports wildcards as well. This has precedence over manage roles list. For example: `role1,role2,dev_role*`.		
ADVANCED	CONNECTOR_REDSHIFT_USER_ROLE_PREFIX	This property is used to set a prefix for role which we will be creating in Redshift for user from ranger. For example if you have user named john in ranger and you have defined prefix as test_user_ then the role which we create for john in Redshift will have name `test_user_john`.	`priv_user_`	
ADVANCED	CONNECTOR_REDSHIFT_GROUP_ROLE_PREFIX	This property is used to set a prefix for role which we will be creating in Redshift for group from ranger. For example if you have group named dev in ranger and you have defined prefix as test_group_ then the role which we create for dev in Redshift will have name `test_group_dev`.	`priv_group_`	
ADVANCED	CONNECTOR_REDSHIFT_ROLE_ROLE_PREFIX	This property is used to set a prefix for role which we will be creating in Redshift for role from ranger. For example if you have role named finance in ranger and you have defined prefix as `test_role_` then the role which we create for finance in Redshift will have name `test_role_finance`.	`priv_role_`	
ADVANCED	CONNECTOR_REDSHIFT_USE_NATIVE_PUBLIC_GROUP	Set this property to true, if you want PolicySync to use Redshift native public group for access grants whenever there is policy created referring to public group inside it.	`true`	
ADVANCED	CONNECTOR_REDSHIFT_MANAGE_USER_FILTERBY_GROUP	Set this property to true, if you want to manage only the users who belongs the the groups defined in manage groups list property.	`false`	
ADVANCED	CONNECTOR_REDSHIFT_MANAGE_USER_FILTERBY_ROLE	Set this property to true, if you want to manage only the users who belongs the the roles defined in manage roles list property.	`false`	
Access control management				

Category	Property name	Description	Default Value	Allowable values
ADVANCED	CONNECTOR_REDSHIFT_ENABLE_VIEW_BASED_MASKING	Set this property to true, if you want to enable secure view based masking in postgres PolicySync. **NOTE** Redshift don't support native masking yet, thus recommended to use view based masking.	true	
ADVANCED	CONNECTOR_REDSHIFT_ENABLE_VIEW_BASED_ROW_FILTER	Set this property to true, if you want to enable secure view based tr filter in Redshift PolicySync. **NOTE** Redshift support native tr filters, but due to its some limitations we recommended to use view based tr filter.	true	
ADVANCED	CONNECTOR_REDSHIFT_SECURE_VIEW_CREATE_FOR_ALL	Set this property to true, if you want to create secure view for all tables as well all view which were created by end users. This will create secure view for resource regardless whether there any masking/tr filter policy exists in ranger.	true	
ADVANCED	CONNECTOR_REDSHIFT_MASKED_NUMBER_VALUE	This property is used to specify the default masking value for numeric columns	0	
ADVANCED	CONNECTOR_REDSHIFT_MASKED_TEXT_VALUE	This property is used to specify the default masking value for text/string columns	<MASKED>'	
ADVANCED	CONNECTOR_REDSHIFT_SECURE_VIEW_NAME_PREFIX	By default view-based tr filter and masking related secure views have the same name as the table name with postfixed by _secure. If you want to change the secure view name prefix and postfix, that can be done with these properties. After prefix and postfix is specified the view name will be in this format : {prefix}{table_name}{postfix}.		
ADVANCED	CONNECTOR_REDSHIFT_SECURE_VIEW_NAME_POSTFIX	By default view-based tr filter and masking related secure views have the same name as the table name with postfixed by _secure. If you want to change the secure view name prefix and postfix, that can be done with these properties. After prefix and postfix is specified the view name will be in this format : {prefix}{table_name}{postfix}.	_secure	

Category	Property name	Description	Default Value	Allowable values
ADVANCED	CONNECTOR_REDSHIFT_SECURE_VIEW_SCHEMA_NAME_PREFIX	By default view-based tr filter and masking related secure views have the same schema name as the table schema name. If you want to change the secure view schema name prefix and postfix, that can be done with these properties. After prefix and postfix is specified the view schema name will be in this format : {prefix}{view_schema_name}{postfix}.		
ADVANCED	CONNECTOR_REDSHIFT_SECURE_VIEW_SCHEMA_NAME_POSTFIX	By default view-based tr filter and masking related secure views have the same schema name as the table schema name. If you want to change the secure view schema name prefix and postfix, that can be done with these properties. After prefix and postfix is specified the view schema name will be in this format : {prefix}{view_schema_name}{postfix}.		
BASIC	CONNECTOR_REDSHIFT_GRANT_UPDATES	This property controls whether actual grant/revoke and create/update/delete queries for user/group/role should be run on Redshift.	false	
ADVANCED	CONNECTOR_REDSHIFT_ENABLE_DATA_ADMIN	This property is used to enable data admin feature, with data admin feature enabled you can create all the policies on table/native view and by default respective grants will be maid on secure view of table table or native view. And this secure view will have tr filter and masking capability as well. In case if you need permission on table then you can select the permission you want plus dataadmin in the policy, In this case that permissions will be granted on both, the table/native view and its secure view as well.	true	
Access audits management				
BASIC	CONNECTOR_REDSHIFT_AUDIT_ENABLE	This property is used to enable access audit fetching from Redshift.	false	
ADVANCED	CONNECTOR_REDSHIFT_AUDIT_EXCLUDED_USERS	This property is used to set the list of users whose access audits we want to ignore. It takes list of comma separated users.		
ADVANCED	CONNECTOR_REDSHIFT_AUDIT_INITIAL_PULL_MINUTES	When the audits are enabled for the first time, the value will decide the number of minutes we need to pull the access audits from the database, default value is 30 mins.	30	

Snowflake connector for PolicySync on Privacera Platform

This topic covers how to configure Snowflake PolicySync access control using Privacera Manager

Snowflake PolicySync prerequisites: users, roles, warehouse, permissions, and UDFs

Before configuring Snowflake, you must first manually create the Snowflake warehouse, database, users, and roles required by PolicySync. All of this can be accomplished by manually executing SQL queries.

NOTE
Log in to Snowflake as a user with ACCOUNTADMIN privileges.

Creating PolicySync role

The PRIVACERA_POLICYSYNC_ROLE role, which we will create in this step, will be used in the SNOWFLAKE_ROLE_TO_USE property when configuring Snowflake with Privacera Manager.

1. Drop a role.

   ```
   DROP ROLE IF EXISTS "PRIVACERA_POLICYSYNC_ROLE";
   ```
2. Create a role.

   ```
   CREATE ROLE IF NOT EXISTS "PRIVACERA_POLICYSYNC_ROLE";
   ```
3. Grant this role permission to users to create/update/delete roles.

   ```
   GRANT ROLE USERADMIN TO ROLE "PRIVACERA_POLICYSYNC_ROLE";
   ```
4. Grant this permission to the role, allowing them to provide grants/revokes privileges on user/roles to create warehouse/database on account.

   ```
   GRANT ROLE SYSADMIN TO ROLE "PRIVACERA_POLICYSYNC_ROLE";
   ```
5. Grant this permission to the role so that it can manage grants for snowflake resources.

   ```
   GRANT MANAGE GRANTS ON ACCOUNT TO ROLE "PRIVACERA_POLICYSYNC_ROLE";
   ```
6. Grant this permission to the role so that it can create native Masking policies.

   ```
   GRANT APPLY MASKING POLICY ON ACCOUNT TO ROLE "PRIVACERA_POLICYSYNC_ROLE";
   ```
7. Grant this permission to the role so that it can create native row filter policies.

   ```
   GRANT APPLY ROW ACCESS POLICY ON ACCOUNT TO ROLE "PRIVACERA_POLICYSYNC_ROLE";
   ```

Creating a warehouse

The PRIVACERA_POLICYSYNC_WH warehouse, which we will create in this step, will be used in the SNOWFLAKE_WAREHOUSE_TO_USE property when configuring Snowflake with Privacera Manager.

Create a warehouse for PolicySync. Change the warehouse size according to deployment.

```
CREATE WAREHOUSE IF NOT EXISTS "PRIVACERA_POLICYSYNC_WH" WITH WAREHOUSE_SIZE='XSMALL'WAR
```

Granting role permission to read access audits

To get read access audit permission on the Snowflake database, follow the steps below.

1. Grant warehouse usage access so we can query the snowflake database and get the Access Audits.

   ```
   GRANT USAGE ON WAREHOUSE "PRIVACERA_POLICYSYNC_WH" TO ROLE "PRIVACERA_POLICYSYNC_ROLE
   ```
2. Grant our role PRIVACERA_POLICYSYNC_ROLE to read Access Audits in the snowflake database.

   ```
   GRANT IMPORTED PRIVILEGES ON DATABASE snowflake TO ROLE "PRIVACERA_POLICYSYNC_ROLE";
   ```

Creating database for Privacera UDFs

The database name PRIVACERA_DB will be used in the SNOWFLAKE_JDBC_DB property when configuring Snowflake with Privacera Manager.

1. This step is optional. If you already have the database and want to use it, you can skip this step.

```
CREATE DATABASE IF NOT EXISTS "PRIVACERA_DB";
```

2. Grant our role `PRIVACERA_POLICYSYNC_ROLE` database access so that we can create UDFs in the database.

```
GRANT ALL ON DATABASE "PRIVACERA_DB" TO ROLE "PRIVACERA_POLICYSYNC_ROLE";

GRANT ALL ON ALL SCHEMAS IN DATABASE "PRIVACERA_DB" TO ROLE "PRIVACERA_POLICYSYNC_ROL
```

Creating user

The user which we will create in this step will be used in the `SNOWFLAKE_JDBC_USERNAME` and `SNOWFLAKE_JDBC_PASSWORD` properties when configuring Snowflake with Privacera Manager.

1. Create a user

```
CREATE USER IF NOT EXISTS "PRIVACERA_POLICYSYNC_USER"PASSWORD='<PLEASE_CHANGE>'MUST_C
```

2. Grant the user the `PRIVACERA_POLICYSYNC_ROLE` role.

```
GRANT ROLE "PRIVACERA_POLICYSYNC_ROLE" TO USER "PRIVACERA_POLICYSYNC_USER";
```

Creating owner role

By configuring the following property in `vars.policysync.snowflake.yml`, PolicySync can take ownership of all objects managed by it. PolicySync requires this in order to create row-filtering and column-Masking policies.

```
SNOWFLAKE_OWNER_ROLE:"PRIVACERA_POLICYSYNC_ROLE"
```

Note

If PolicySync is not configured to take ownership of all objects managed by PolicySync, keep the property value blank.

```
SNOWFLAKE_OWNER_ROLE:""
```

Masking and row level filtering

To run the Masking and Row Level Filter, the following permissions must be granted to each database managed by PolicySync. <DATABASE_NAME> must be replaced with the specific value.

```
GRANT ALL ON DATABASE "<DATABASE_NAME>" TO ROLE "PRIVACERA_POLICYSYNC_ROLE";
GRANT ALL ON ALL SCHEMAS IN DATABASE "<DATABASE_NAME>" TO ROLE "PRIVACERA_POLICYSYNC_ROL
GRANT ALL ON FUTURE SCHEMAS IN DATABASE "<DATABASE_NAME>" TO ROLE "PRIVACERA_POLICYSYNC_
GRANT ALL ON ALL TABLES IN DATABASE "<DATABASE_NAME>" TO ROLE "PRIVACERA_POLICYSYNC_ROLE
GRANT ALL ON FUTURE TABLES IN DATABASE "<DATABASE_NAME>" TO ROLE "PRIVACERA_POLICYSYNC_R
GRANT ALL ON ALL VIEWS IN DATABASE "<DATABASE_NAME>" TO ROLE "PRIVACERA_POLICYSYNC_ROLE"
GRANT ALL ON FUTURE VIEWS IN DATABASE "<DATABASE_NAME>" TO ROLE "PRIVACERA_POLICYSYNC_RO
```

Generalized approach for implementing PolicySync

Use this generalized approach for implementing PolicySync.

1. Understand how PolicySync works and how it is configured. See PolicySync design and configuration on Privacera Platform [61].
2. Decide which PolicySync topology best suits your needs:
   - Required basic PolicySync topology: always at least one connector instance [64]
   - Optional topology: multiple connector instances for Kubernetes pods and Docker containers [65]
   - Recommended PolicySync topology: individual policy repositories for individual connectors [65]
3. Create the required, basic PolicySync configuration. See Required basic PolicySync topology: always at least one connector instance [64]

4. Examine the BASIC and ADVANCED properties, decide which features you want to implement, and set the necessary values in the YAML property file.

Connector name: snowflake

When you create the connector as detailed in PolicySync design and configuration on Privacera Platform [61], use the following reserved word for the name of the connector:

`snowflake`

In formal syntax shown in Connector instance directory/file structure [63] replace `<ConnectorName>` with the above and in the example in Required basic PolicySync topology: always at least one connector instance [64], replace `postgres` with the above.

Optional Basic Authentication for PolicySync

To optionally enable basic authenticate for PolicySync to Snowflake you can create a JSON file in your connector instance subdirectory.

The name of the file must be `XXX.json`.

An example of the contents of `XXX.json`.:

```
 {
  "type": "service_account",
  "project_id": "your_project_id",
  "private_key_id": "autogenerated_value",
  "private_key": "-----BEGIN PRIVATE KEY-----autogenerated_value-----END PRIVATE KEY----
  "client_email": "autogenerated_value",
  "client_id": "autogenerated_value",
  "auth_uri": "https://accounts.google.com/o/oauth2/auth",
  "token_uri": "https://oauth2.googleapis.com/token",
  "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
  "client_x509_cert_url": "https://www.googleapis.com/robot/v1/metadata/x509/autogenerat
}
```

Snowflake connector properties for PolicySync on Privacera Platform

These Snowflake connector properties can be set for PolicySync in Privacera Platform.

The properties are grouped by general function, such as JDBC connection properties, properties for user, group, and role management, and other functions.

The properties are also categorized as **BASIC** or **ADVANCED**:

- **BASIC** pertains to the most fundamental aspects of the connector, such as authentication.
- **ADVANCED** indicates additional features beyond the **BASIC**s, such as row-filtering or group member handling.

Start by setting the **BASIC** properties and then examine the **ADVANCED** properties to determine which of these features you might want to enable.

For a general process to migrate values from old YAML files to the new YAML files, see Migration to PolicySync v2 on Privacera Platform 7.2 [68].

Category	Property name	Description	Default	Allowable values
JDBC configuration properties				
BASIC	CONNECTOR_SNOWFLAKE_JDBC_URL	This property is used to set the JDBC URL, which can be used to connect to the Snowflake server. The JDBC URL should be formatted as follows: jdbc:snowflake://testsnowflake.snowflakecomputing.com/?warehouse=<WAREHOUSE_TO_USE>&role=<ROLE_TO_USE> Example :- jdbc:snowflake://testsnowflake.snowflakecomputing.com/?warehouse=PRIVACERA_POLICYSYNC_WH&role=PRIVACERA_SYNC_ROLE		
BASIC	CONNECTOR_SNOWFLAKE_JDBC_USERNAME	This property is used to specify the JDBC username that will be used to connect to Snowflake.		
BASIC	CONNECTOR_SNOWFLAKE_JDBC_PASSWORD	This property is used to specify the JDBC username password that will be used to connect to Snowflake.		
BASIC	CONNECTOR_SNOWFLAKE_WAREHOUSE_TO_USE	This property is used to specify which JDBC warehouse will be used to establish a connection in order to run SQL queries on Snowflake.		
BASIC	CONNECTOR_SNOWFLAKE_ROLE_TO_USE	This property is used to specify the Snowflake role that will be used to run SQL queries on Snowflake.		
BASIC	CONNECTOR_SNOWFLAKE_OWNER_ROLE	This property is used to specify who owns all of the resources managed by policysync. The specified role will become the owner of all managed resources and will have complete control over those resources. We support changing the owner of a database, schema, tables, and views.Generally value of this should be same as SNOWFLAKE_ROLE_TO_USE property value to be used to execute queries. NOTE: If the owner role is left blank, ownership will not change, and users who create tables/views or other objects will be the owner of those objects, and policysync will not be able to control access to those objects.		
BASIC	CONNECTOR_SNOWFLAKE_MANAGE_WAREHOUSE_LIST	This property is used to specify the names of comma-separated warehouses for which policysync should manage access control. If you want to manage all warehouses, you can skip this property. This also works with wildcards. The ignore warehouses list takes precedence over the manage warehouses list. For example, testdb1warehouse, testdb2warehouse, sales dbwarehouse*. **NOTE: values for this property are case-sensitive.		
BASIC	CONNECTOR_SNOWFLAKE_MANAGE_DATABASE_LIST	This property is used to specify comma-separated database names for which policysync should manage access control. If you want to manage all databases, you can skip this property. This also accepts wildcards. The manage database list takes precedence over the ignore database list. For example, testdb1, testdb2, sales db* **NOTE: values for this property are case-sensitive.		

Category	Property name	Description	Default	Allowable values
ADVANCED	CONNECTOR_SNOWFLAKE_MANAGE_SCHEMA_LIST	This property is used to set comma separated schema Fqdn which access control should be managed by policysync. If you want to manage all schemas then you can skip specifying this property. This supports wildcards as well. The ignore schema list has precedence over manage schema list. Eg. testdb1.schema1,testdb2.schema2,sales_db*.sales* **NOTE: values for this property are case-sensitive.		
ADVANCED	CONNECTOR_SNOWFLAKE_MANAGE_TABLE_LIST	This property is used to set comma separated table/view Fqdn which access control should be managed by policysync. If you want to manage all tables/views then you can skip specifying this property. This supports wildcards as well. The ignore table list has precedence over manage table list. Eg. testdb1.schema1.table1,testdb2.schema2.view2,sales_db*.sales*.* **NOTE: values for this property are case-sensitive.		
ADVANCED	CONNECTOR_SNOWFLAKE_MANAGE_STREAM_LIST	This property is used to set comma separated streams Fqdn which access control should be managed by policysync. If you want to manage all streams then you can skip specifying this property. This supports wildcards as well. The ignore streams list has precedence over manage streams list. Eg. testdb1.schema1.streams1,testdb2.schema2.streams2,sales_db*.sales*.* **NOTE: values for this property are case-sensitive.		
ADVANCED	CONNECTOR_SNOWFLAKE_MANAGE_FUNCTION_LIST	This property is used to set comma separated functions Fqdn which access control should be managed by policysync. If you want to manage all functions then you can skip specifying this property. This supports wildcards as well. The ignore functions list has precedence over manage functions list. Eg. testdb1.schema1.functions1,testdb2.schema2.functions2,sales_db*.sales*.* **NOTE: values for this property are case-sensitive.		
ADVANCED	CONNECTOR_SNOWFLAKE_MANAGE_PROCEDURE_LIST	This property is used to set comma separated procedures Fqdn which access control should be managed by policysync. If you want to manage all procedures then you can skip specifying this property. This supports wildcards as well. The ignore procedures list has precedence over manage procedures list. Eg. testdb1.schema1.procedures1,testdb2.schema2.procedures2,sales_db*.sales*.* **NOTE: values for this property are case-sensitive.		
ADVANCED	CONNECTOR_SNOWFLAKE_MANAGE_SEQUENCE_LIST	This property is used to set comma separated sequences Fqdn which access control should be managed by policysync. If you want to manage all sequences then you can skip specifying this property. This supports wildcards as well. The ignore sequences list has precedence over manage sequences list. Eg. testdb1.schema1.sequence1,testdb2.schema2.sequence2,sales_db*.sales*.* **NOTE: values for this property are case-sensitive.		

Category	Property name	Description	Default	Allowable values
ADVANCED	CONNECTOR_SNOWFLAKE_MANAGE_FILE_FORMAT_LIST	This property is used to set comma separated FileFormats Fqdn which access control should be managed by policysync. If you want to manage all FileFormats then you can skip specifying this property. This supports wildcards as well. The ignore FileFormats list has precedence over manage FileFormats list. Eg. testdb1.schema1.FileFormat1,testdb2.schema2.FileFormat2,sales_db*.sales*.* **NOTE: values for this property are case-sensitive.		
ADVANCED	CONNECTOR_SNOWFLAKE_MANAGE_PIPE_LIST	This property is used to set comma separated pipes Fqdn which access control should be managed by policysync. If you want to manage all pipes then you can skip specifying this property. This supports wildcards as well. The ignore pipes list has precedence over manage pipes list. Eg. testdb1.schema1.pipe1,testdb2.pipe2.FileFormat2,sales_db*.sales*.* **NOTE: values for this property are case-sensitive.		
ADVANCED	CONNECTOR_SNOWFLAKE_MANAGE_EXTERNAL_STAGE_LIST	This property is used to set comma separated ExternalStage Fqdn which access control should be managed by policysync. If you want to manage all ExternalStage then you can skip specifying this property. This supports wildcards as well. The ignore ExternalStage list has precedence over manage ExternalStage list. Eg. testdb1.schema1.ExternalStage1,testdb2.ExternalStage2.FileFormat2,sales_db*.sales*.* **NOTE: values for this property are case-sensitive.		
ADVANCED	CONNECTOR_SNOWFLAKE_MANAGE_INTERNAL_STAGE_LIST	This property is used to set comma separated InternalStage Fqdn which access control should be managed by policysync. If you want to manage all InternalStage then you can skip specifying this property. This supports wildcards as well. The ignore InternalStage list has precedence over manage InternalStage list. Eg. testdb1.schema1.InternalStage1,testdb2.InternalStage2.FileFormat2,sales_db*.sales*.* **NOTE: values for this property are case-sensitive.		
ADVANCED	CONNECTOR_SNOWFLAKE_IGNORE_WAREHOUSE_LIST	This property is used to set comma separated warehouse names which access control you don't want to be managed by policysync. If you don't want to ignore any warehouse then you can skip specifying this property. This supports wildcards as well. This has precedence over manage warehouse list. Eg. testdb1warehouse,testdb2warehouse,sales_dbwarehouse* **NOTE: values for this property are case-sensitive.		
ADVANCED	CONNECTOR_SNOWFLAKE_IGNORE_DATABASE_LIST	This property is used to set comma separated database names which access control you don't want to be managed by policysync. If you don't want to ignore any database then you can skip specifying this property. This supports wildcards as well. This has precedence over manage database list. Eg. testdb1,testdb2,sales_db* **NOTE: values for this property are case-sensitive.		

Category	Property name	Description	Default	Allowable values
ADVANCED	CONNECTOR_SNOWFLAKE_IGNORE_SCHEMA_LIST	This property is used to set comma separated schema fqdn which access control you don't want to be managed by policysync. If you don't want to ignore any schema then you can skip specifying this property. This supports wildcards as well. This has precedence over manage schema list. Eg. testdb1.schema1,testdb2.schema2,sales_db*.sales* **NOTE: values for this property are case-sensitive.		
ADVANCED	CONNECTOR_SNOWFLAKE_MANAGE_USERS	This property controls whether policysync should manage the membership between user and user role.	SNOWFLAKE_MANAGE_ENTITIES	
ADVANCED	CONNECTOR_SNOWFLAKE_MANAGE_GROUPS	This property controls whether we should create role in snowflake for groups fetched from ranger.	SNOWFLAKE_MANAGE_ENTITIES	
ADVANCED	CONNECTOR_SNOWFLAKE_MANAGE_ROLES	This property controls whether we should create role in snowflake for roles fetched from ranger.	SNOWFLAKE_MANAGE_ENTITIES	
ADVANCED	CONNECTOR_SNOWFLAKE_MANAGE_USER_LIST	This property is used to set comma separated user names which access control should be managed by policysync. If you want to manage all users then you can skip specifying this property. This supports wildcards as well. The ignore users list has precedence over manage users list. Eg. user1,user2,dev_user*		
ADVANCED	CONNECTOR_SNOWFLAKE_MANAGE_GROUP_LIST	This property is used to set comma separated group names which access control should be managed by policysync. If you want to manage all group then you can skip specifying this property. This supports wildcards as well. The ignore group list has precedence over manage group list. Eg. group1,group2,dev_group*		
ADVANCED	CONNECTOR_SNOWFLAKE_MANAGE_ROLE_LIST	This property is used to set comma separated role names which access control should be managed by policysync. If you want to manage all role then you can skip specifying this property. This supports wildcards as well. The ignore role list has precedence over manage role list. Eg. role1,role2,dev_role*		
ADVANCED	CONNECTOR_SNOWFLAKE_IGNORE_USER_LIST	This property is used to set comma separated user names which access control you don't want to be managed by policysync. If you don't want to ignore any users then you can skip specifying this property. This supports wildcards as well. This has precedence over manage users list. Eg. user1,user2,dev_user*		
ADVANCED	CONNECTOR_SNOWFLAKE_IGNORE_GROUP_LIST	This property is used to set comma separated group names which access control you don't want to be managed by policysync. If you don't want to ignore any groups then you can skip specifying this property. This supports wildcards as well. This has precedence over manage groups list. Eg. group1,group2,dev_group*		

Category	Property name	Description	Default	Allowable values
ADVANCED	CONNECTOR_SNOWFLAKE_IGNORE_ROLE_LIST	This property is used to set comma separated role names which access control you don't want to be managed by policy-sync. If you don't want to ignore any roles then you can skip specifying this property. This supports wildcards as well. This has precedence over manage roles list. Eg. role1,role2,dev_role*		
ADVANCED	CONNECTOR_SNOWFLAKE_USER_NAME_REPLACE_FROM_REGEX	This takes the regular expression as input and finds the matching characters in a user name and replaces them with the characters specified in property. If kept blank, no find and replace operation is performed.	[~`$&+:;=?@#|'<>.^*()_%\\\\[\\\\]!\\\\-\\\\/\\\\\\\\{}]	
ADVANCED	CONNECTOR_SNOWFLAKE_USER_NAME_REPLACE_TO_STRING	The value specified in this property is used to replace the characters found by the regex specified user name regex property. If kept blank, no find and replace operation is performed.	_	
ADVANCED	CONNECTOR_SNOWFLAKE_GROUP_NAME_REPLACE_FROM_REGEX	This takes the regular expression as input and finds the matching characters in a group name and replaces them with the characters specified in property. If kept blank, no find and replace operation is performed.	[~`$&+:;=?@#|'<>.^*()_%\\\\[\\\\]!\\\\-\\\\/\\\\\\\\{}]	
ADVANCED	CONNECTOR_SNOWFLAKE_GROUP_NAME_REPLACE_TO_STRING	The value specified in this property is used to replace the characters found by the regex specified group name regex property. If kept blank, no find and replace operation is performed.	_	
ADVANCED	CONNECTOR_SNOWFLAKE_ROLE_NAME_REPLACE_FROM_REGEX	This takes the regular expression as input and finds the matching characters in a role name and replaces them with the characters specified in property. If kept blank, no find and replace operation is performed.	[~`$&+:;=?@#|'<>.^*()_%\\\\[\\\\]!\\\\-\\\\/\\\\\\\\{}]	
ADVANCED	CONNECTOR_SNOWFLAKE_ROLE_NAME_REPLACE_TO_STRING	The value specified in this property is used to replace the characters found by the regex specified role name regex property. If kept blank, no find and replace operation is performed.	_	
ADVANCED	CONNECTOR_SNOWFLAKE_USER_NAME_PERSIST_CASE_SENSITIVITY	After loading user from Ranger API's all are converted into lowercase, but in some cases, you would need to have the users in the same case as they are in Ranger. When setting this value to true, it will maintain the case sensitivity of names as they are in Ranger.	false	true/false
ADVANCED	CONNECTOR_SNOWFLAKE_GROUP_NAME_PERSIST_CASE_SENSITIVITY	After loading group from Ranger API's all are converted into lowercase, but in some cases, you would need to have the users in the same case as they are in Ranger. When setting this value to true, it will maintain the case sensitivity of names as they are in Ranger.	false	true/false
ADVANCED	CONNECTOR_SNOWFLAKE_ROLE_NAME_PERSIST_CASE_SENSITIVITY	After loading role from Ranger API's all are converted into lowercase, but in some cases, you would need to have the users in the same case as they are in Ranger. When setting this value to true, it will maintain the case sensitivity of names as they are in Ranger.	false	true/false
ADVANCED	CONNECTOR_SNOWFLAKE_MANAGE_USER_FILTERBY_GROUP	Set this property to true, if you want to manage only the users who belongs to the groups defined in manage groups list property.	false	true/false
ADVANCED	CONNECTOR_SNOWFLAKE_MANAGE_USER_FILTERBY_ROLE	Set this property to true, if you want to manage only the users who belongs to the roles defined in manage roles list property.	false	true/false

Category	Property name	Description	Default	Allowable values
BASIC	CONNECTOR_SNOWFLAKE_GRANT_UPDATES	This property controls whether actual grant/revoke and create/update/delete queries for user/group/role should be run on snowflake.	true	true/false
ADVANCED	CONNECTOR_SNOWFLAKE_ENABLE_MASKING	This property controls whether to enable native masking policy creation functionality in policysync.	true	
ADVANCED	CONNECTOR_SNOWFLAKE_MASKING_POLICY_DB_NAME	This property used to set the database name in which policysync should create custom masking policies		
ADVANCED	CONNECTOR_SNOWFLAKE_MASKING_POLICY_SCHEMA_NAME	This property used to set the schema name in which policysync should create all native masking policies, if this is kept as blank then it will consider the resource schema as masking policy schema	public	
BASIC		This property used to set the database name in which policysync should create custom masking functions		
ADVANCED	CONNECTOR_SNOWFLAKE_ENABLE_ROW_FILTER	This property controls whether to enable native tr filter policy creation functionality in policysync.	true	
ADVANCED	CONNECTOR_SNOWFLAKE_ROW_FILTER_POLICY_DB_NAME	This property used to set the database name in which policysync should create all native tr filter policies, if this is kept as blank then it will consider the resource database as tr filter policy database		
ADVANCED	CONNECTOR_SNOWFLAKE_ROW_FILTER_POLICY_SCHEMA_NAME	This property used to set the schema name in which policysync should create all native tr filter policies, if this is kept as blank then it will consider the resource schema as tr filter policy schema	PUBLIC	
ADVANCED	CONNECTOR_SNOWFLAKE_ENABLE_VIEW_BASED_ROW_FILTER	Set this property to true, if you want to enable secure view based tr filter in postgres policysync. Note :- Postgres support native tr filters, but due to its some limitations we recommended to use view based tr filter.	FALSE	
ADVANCED	CONNECTOR_SNOWFLAKE_ENABLE_VIEW_BASED_MASKING	Set this property to true, if you want to enable secure view based masking in postgres policysync. Note :- Postgres don't support native masking yet, thus recommended to use view based masking.	FALSE	
ADVANCED	CONNECTOR_SNOWFLAKE_SECURE_VIEW_SCHEMA_NAME_PREFIX	By default view-based tr filter and masking related secure views have the same schema name as the table schema name. If you want to change the secure view schema name prefix and postfix, that can be done with these properties. After prefix and postfix is specified the view schema name will be in this format : {prefix}{view_schema_name}{postfix} For {view_schema_name} refer to variable POSTGRES_SECURE_VIEW_SCHEMA_NAME		
ADVANCED	CONNECTOR_SNOWFLAKE_SECURE_VIEW_SCHEMA_NAME_POSTFIX	By default view-based tr filter and masking related secure views have the same schema name as the table schema name. If you want to change the secure view schema name prefix and postfix, that can be done with these properties. After prefix and postfix is specified the view schema name will be in this format : {prefix}{view_schema_name}{postfix} For {view_schema_name} refer to variable POSTGRES_SECURE_VIEW_SCHEMA_NAME		

Category	Property name	Description	Default	Allowable values
ADVANCED	CONNECTOR_SNOWFLAKE_SECURE_VIEW_NAME_PREFIX	By default view-based tr filter and masking related secure views have the same name as the table name with postfixed by _secure. If you want to change the secure view name prefix and postfix, that can be done with these properties. After prefix and postfix is specified the view name will be in this format : {prefix}{table_name}{postfix}		
ADVANCED	CONNECTOR_SNOWFLAKE_SECURE_VIEW_NAME_POSTFIX	By default view-based tr filter and masking related secure views have the same name as the table name with postfixed by _secure. If you want to change the secure view name prefix and postfix, that can be done with these properties. After prefix and postfix is specified the view name will be in this format : {prefix}{table_name}{postfix}	_SECURE	
ADVANCED	CONNECTOR_SNOWFLAKE_SECURE_VIEW_CREATE_FOR_ALL	Set this property to true, if you want to create secure view for all tables as well all view which were created by end users. This will create secure view for resource regardless whether there any masking/tr filter policy exists in ranger.	false	
ADVANCED	CONNECTOR_SNOWFLAKE_MASKED_NUMBER_VALUE	This property used to set the value of the masked column of datatype number	0	
ADVANCED	CONNECTOR_SNOWFLAKE_MASKED_TEXT_VALUE	This property used to set the value of the masked column of datatype text	'<MASKED>'	

Configure resource policies

You can create policies that control access to your connected datasources to protect their resources, such as tables, columns, rows, files, and file paths).resource policies.

Configure ADLS resource policies

ADLS supports access policies.

- Account Name: A storage account name or * for all storage accounts.
- Container Name: A container name or * for all containers.
- Object Path: A specific object path or * for all object paths.
- Allow Conditions:
  - Permissions:
    - Read: READ permission on the URL permits the user to perform HiveServer2 operations which use S3 as a data source for Hive tables.
    - Write: WRITE permission on the URL permits the user to perform HiveServer2 operations which write data to the specified S3 location.
    - Delete: DELETE permission allows you to delete the resource.
    - Metadata Read: METADATA READ permission allows you to run HEAD operation on objects. Also, this permission list buckets, list objects and retrieves objects metadata.
    - Metadata Write: METADATA WRITE permission allows you to modify object's metadata and object's ACL, Tagging, Cros, etc.
    - Admin: Administrators can edit or delete the policy, and can also create child policies based on the original policy.
  - Delegate Admin: Select 'Delegate Admin' to assign administrator rights to the roles, groups, or users specified in the policy. The administrator can edit or delete the policy, and can also create child policies based on the original policy.
  - For Select Group: either **public** or a specific group

Configure AWS S3 resource policies

AWS S3 supports access policies.

- Bucket Name: Specify the bucket name. For example: aws-athena-query-result
  Note: Wildcard characters such as '*' are allowed if you want to give access to all buckets. |
- Object Path: Specify the object path. It accepts wildcard character such as '*'.
  - Recursive: This allows you to view multiple folders based on the mentioned object path.
  - Non-recursive: This allows you to view specific folders based on the mentioned object path.

Example:

If the Bucket name is {*bucket-AWS}* and the Object path is {*path1},*

- Sample 1: *s3://bucket-AWS/path1/*
- Sample 2: *s3://bucket-name/path1/path2/*

If the **Recursive** toggle button is enabled [the default behavior], you can view all files within the `path1` and `path2` folders.

If the **Recursive** toggle button is disabled, you won't be able to view any files in the `path1` folder.

- Allow Conditions:
  - Permissions:
    - Read: READ permission on the URL permits the user to perform HiveServer2 operations which use S3 as a data source for Hive tables.
    - Write: WRITE permission on the URL permits the user to perform HiveServer2 operations which write data to the specified S3 location.
    - Delete: DELETE permission allows you to delete the resource.
    - Metadata Read: METADATA READ permission allows you to run HEAD operation on objects. Also, this permission list buckets, list objects and retrieves objects metadata.
    - Metadata Write: METADATA WRITE permission allows you to modify object's metadata and object's ACL, Tagging, Cros, etc.
    - Admin: Administrators can edit or delete the policy, and can also create child policies based on the original policy.

Configure Athena resource policies

Athena supports access policies.

- Workgroup: Specify the workgroup name of Athena.
  - Data source: Specify the name of the data source.
  - Database: Specify the name of the database.
  - Table: Specify the name of the table.
  - Column: Specify the name of the column.
- URL: Specify the cloud storage path. For example - s3a://user/poc/sales.txt where the end-user permission is needed to access the data from/to a cloud storage path.
- Allow Conditions:
  - Permissions:
    - BatchGetNamedQuery
    - BatchGetQueryExecution
    - CreateNamedQuery
    - CreateWorkGroup
    - DeleteNamedQuery
    - DeleteWorkGroup
    - GetNamedQuery

- GetQueryExecution
- GetQueryResults
- GetWorkGroup
- ListNamedQueries
- ListQueryExecutions
- ListTagsForResource
- ListWorkGroups
- StartQueryExecution
- StopQueryExecution
- TagResource
- UntagResource
- UpdateWorkGroup
- Alter
- Create
- Describe
- Drop
- Insert
- MSCK Repair
- Select
- Show
- ListDataCatalogs

- Delegate Admin: Select 'Delegate Admin' to assign administrator rights to the roles, groups, or users specified in the policy. The administrator can edit or delete the policy, and can also create child policies based on the original policy.

Configure Databricks resource policies

By default, Databricks File System (DBFS) is protected by Privacera. This blocks common tasks like adding jars/libraries into the cluster. For example, when you try to install a library into a protected DBFS cluster, the following exception will be displayed:

Exception:

Exception while installing a Jar in Databricks Cluster with Plugin enabled? java.lang.RuntimeException: ManagedLibraryInstallFailed: java.security.AccessControlException: Access denied for resource [dbfs:/local_disk0/tmp/addedFile4604599454488620309privacera_crypto_jar_with_dependencies-eba20.jar] action [READ] for library:JavaJarId(dbfs:/privacera/crypto/jars/privacera-crypto-jar-with-dependencies.jar,,NONE),isSharedLibrary=false

To grant permissions to read/write on DBFS, you need to create an access policy. Access to DBFS will be audited.

To create an access policy for Databricks, do the following:

1. Go to **Access Management > Resource Policies > privacera_files**.
2. Click **Add New Policy**.
3. Enter the following details:
   a. Policy Name: Access to Temporary Folder for adding libraries
   b. Resource: dbfs:/local_disk0/tmp

      NOTE
      Make sure the recursive box next to the **Resource** field is checked.

   c. Group: public

d. Permission: read & write

NOTE
The above policy gives permission to all the users. If you want to restrict to only certain users, then instead of giving permission to the group **public**, provide it to appropriate users or groups.

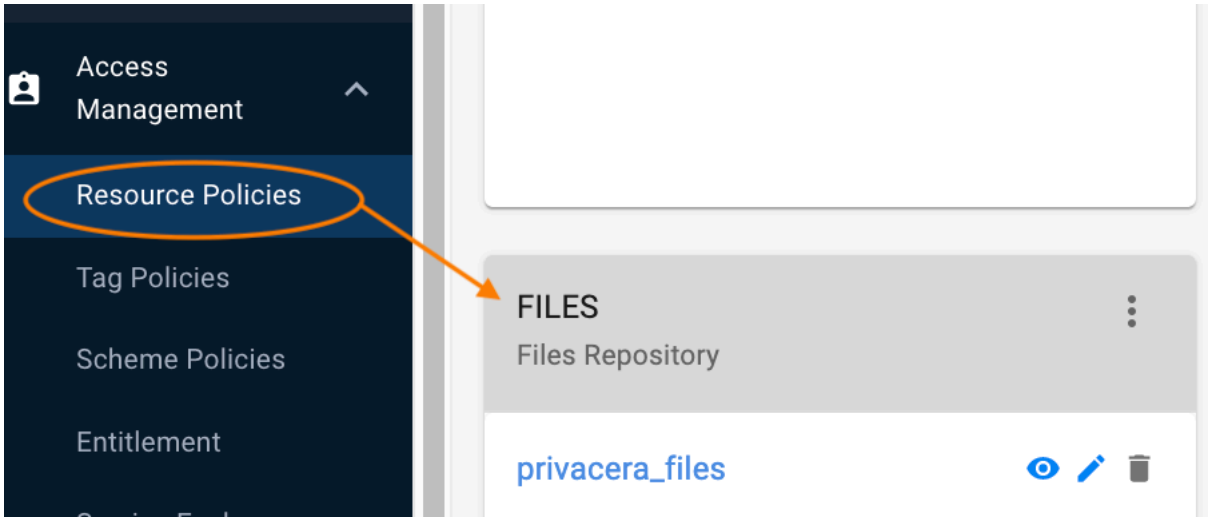
Configure DynamoDB resource policies

DynamoDB supports access policies.

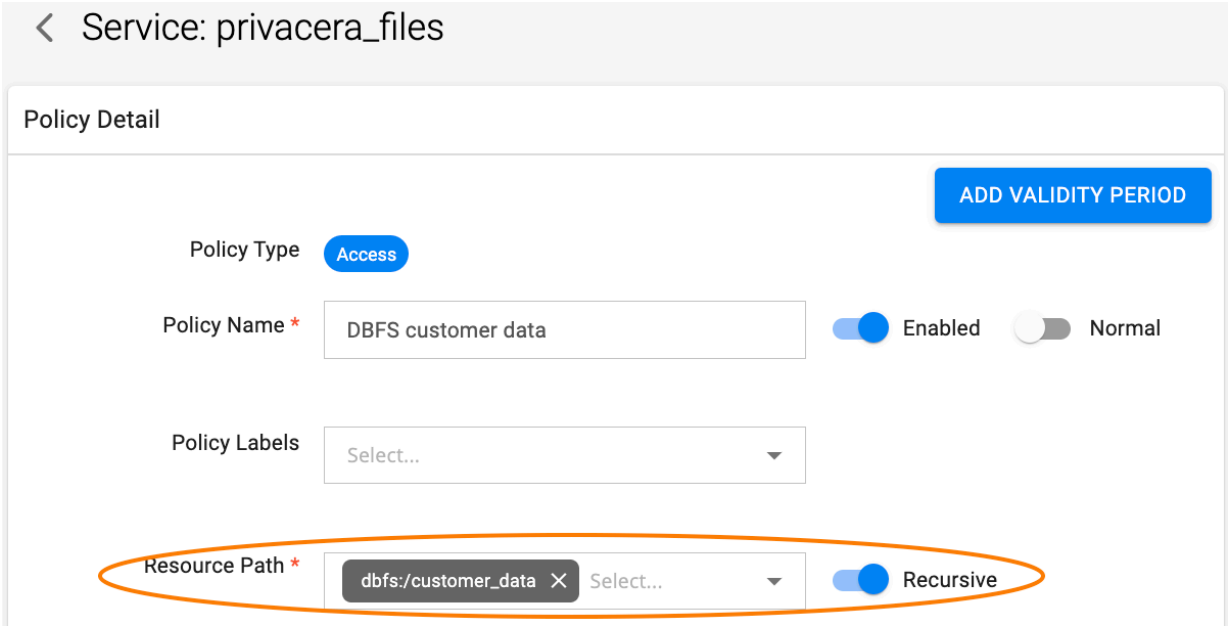
- Table: Specify the table name.
- Attribute: Specify the attribute name.
- Allow Conditions
  - Permissions:
    - Read
    - Write
    - Create
    - Delete
    - List tables
    - Admin
- Delegate Admin: Select 'Delegate Admin' to assign administrator rights to the roles, groups, or users specified in the policy. The administrator can edit or delete the policy, and can also create child policies based on the original policy.

Configure Files resource policies

The **FILES** resource policy repository (also shown as **privacera_files**) holds policies you create to control access to specific files and filesystem directories.



The most important item for a **privacera_files** resource policy is the resource path, as shown here. (This is not a complete policy, just focused on the resource path.) Note that by default the policy applies recursively to the specified path.



Policies can specify many different kinds of resource paths. For example:

- Azure Data Lake Storage Gen2: abfss://path_to_resource
- Azure (non-ADLS Gen2): wasbs://path_to_resource
- Databricks File System: dbfs:/path_to_resource
- Local file: file://path_to_resource
- S3: s3://path_to_resource
- S3 up to 5TB: s3a://path_to_resource
- S3 up to 5GB: s3n://path_to_resource

Description of fields for a files resource policy

- Resource Path
  - Recursive/Non-Recursive:
- Allow Conditions
  - Permissions
    - Read
    - Write
  - Delegate Admin: Select 'Delegate Admin' to assign administrator rights to the roles, groups, or users specified in the policy. The administrator can edit or delete the policy, and can also create child policies based on the original policy.

Configure GBQ resource policies

GBQ supports access policies.

- Project ID
- Dataset Name
- TableName
- Column Name
- Allow Conditions
  - Permissions
    - CreateTable
    - CreateTableAsSelect
    - CreateView

    - Delete
    - DropTable
    - DropView
    - Insert
    - Query
    - Update
  - Delegate Admin: Select 'Delegate Admin' to assign administrator rights to the roles, groups, or users specified in the policy. The administrator can edit or delete the policy, and can also create child policies based on the original policy.

Configure GCS resource policies

GCS supports access policies.

- Project ID
- Bucket Name
- Object Path
  - Recursive/Non-recursive:
- Allow Conditions
  - Permissions:
    - Read: READ permission on the URL permits the user to perform HiveServer2 operations which use S3 as a data source for Hive tables.
    - Write: WRITE permission on the URL permits the user to perform HiveServer2 operations which write data to the specified S3 location.
    - Delete: DELETE permission allows you to delete the resource.
    - Metadata Read: METADATA READ permission allows you to run HEAD operation on objects. Also, this permission list buckets, list objects, and retrieves objects metadata.
    - Metadata Write: METADATA WRITE permission allows you to modify object's metadata and object's ACL, Tagging, Cros, etc.
    - Admin: Administrators can edit or delete the policy, and can also create child policies based on the original policy.
  - Delegate Admin: Select 'Delegate Admin' to assign administrator rights to the roles, groups, or users specified in the policy. The administrator can edit or delete the policy, and can also create child policies based on the original policy.

Configure Glue resource policies

Glue supports access policies.

- Database: Specify the database name.
- Table: Specify the table name.
  Note: You are allowed to enter wildcard character such as '*'. in the above fields.
- Allow Conditions:
  - Permissions:
    - GetCatalogImportStatus
    - GetDatabases
    - GetDatabase
    - GetTables
    - GetTable
    - CreateTable
    - CreateDatabase
    - DeleteDatabase
    - DeleteTable
  - Delegate Admin: Select 'Delegate Admin' to assign administrator rights to the roles, groups, or users specified in the policy. The administrator can edit or delete the policy, and can also create child policies based on the original policy.

Configure Hive resource policy

This section describes how to configure Hive resource policy, including the **Accessed Together** and **Not Accessed Together** policy conditions.

On the **Policy Details** page, do the following:

- **Database**: Specify the database name.
  - **Table/UDF**: Specify the table or udf name
  - **Column**: Specify the column name.

> **NOTE**
> By default the 'Include' option is selected to allow access for all the above fields. In case you want to deny access, toggle to the 'Exclude' option.

- **URL**: Specify the cloud storage path. For example - s3a://user/poc/sales.txt where the end-user permission is needed to read/write the Hive data from/to a cloud storage path.
  - Recursive
  - Non-recursive
- **Global**: Specify global dataset.
- **Allow Conditions**: In this section, you can specify the policy conditions and permissions for resources.
  - **Policy Conditions**: This option allows a user to add custom conditions while evaluating authorization requests. Click the Add Conditions button. In the pop-up, you can see the **Accessed Together ?** and **Non Accessed Together ?** conditions.
    - **Accessed Together ?**: This option allows you to access a specified request (minimum two columns) in the query format.
      For example:

      ```
      default.employeepersonalview.EMP_SSN, default.employeepersonalview.CC
      ```

      Above query allows user to access EMP_SSN & CC columns only when both are mentioned together in the query else it will give denied permission error.
    - **Not Accessed Together?**: This option denies specified requests (minimum two columns) in the query format.
      For example:

      ```
      default.employeepersonalview.EMP_SSN, default.employeepersonalview.CC
      ```

      Above query deny user to view EMP_SSN & CC columns data when both are mentioned together in the query and give denied permission error.
  - **Permission**: Permissions are common for all the resources, add them as per your requirement.
    The list of permissions are:
    - Select
    - Update
    - Create
    - Drop
    - Alter
    - Index
    - Lock
    - All
    - Read
    - Write

    - Data_admin

Configure Lambda resource policies

Lambda supports access policies.

- Function: Specify the function name of Lambda.
- Layer: Specify the layer name of Lambda.
  Note: You are allowed to enter wildcard characters such as '*'.
- Allow Conditions:
  - Permissions:
    - ListAliases
    - ListEventSourceMappings
    - ListFunctionEventInvokeConfigs
    - ListFunctions
    - ListLayers
    - ListLayerVersions
    - ListProvisionedConcurrencyConfigs
    - ListVersionsByFunction
    - GetAccountSettings
    - GetAlias
    - GetEventSourceMapping
    - GetFunction
    - GetFunctionConcurrency
    - GetFunctionConfiguration
    - GetFunctionEventInvokeConfig
    - GetLayerVersion
    - GetLayerVersionByArn
    - GetLayerVersionPolicy
    - GetPolicy
    - GetProvisionedConcurrencyConfig
    - ListTags
    - CreateAlias
    - CreateEventSourceMapping
    - CreateFunction
    - DeleteAlias
    - DeleteEventSourceMapping
    - DeleteFunction
    - DeleteFunctionConcurrency
    - DeleteFunctionEventInvokeConfig
    - DeleteLayerVersion
    - DeleteProvisionedConcurrencyConfig
    - InvokeFunction
    - PublishLayerVersion
    - PublishVersion
    - PutFunctionConcurrency
    - PutFunctionEventInvokeConfig
    - PutProvisionedConcurrencyConfig
    - TagResource
    - UntagResource
    - UpdateAlias
    - UpdateEventSourceMapping
    - UpdateFunctionCode

    - UpdateFunctionConfiguration
    - UpdateFunctionEventInvokeConfig
    - AddLayerVersionPermission
    - AddPermission
    - RemoveLayerVersionPermission
    - RemovePermission
  - Delegate Admin: Select 'Delegate Admin' to assign administrator rights to the roles, groups, or users specified in the policy. The administrator can edit or delete the policy, and can also create child policies based on the original policy.

Configure Kafka resource policies

Kafka supports access policies.

- Topic
- Transactionalid
- Cluster
- Delegationtoken
- Consumergroup
- Policy Conditions
  - Add Conditions
- Allow Conditions:
  - Policy Conditions
    - Add Conditions
  - Permissions
    - Consume
    - Describe
    - Delete
  - Delegate Admin: Select 'Delegate Admin' to assign administrator rights to the roles, groups, or users specified in the policy. The administrator can edit or delete the policy, and can also create child policies based on the original policy.

Configure Kinesis resource policies

Kinesis supports access policies.

- Kinesis_Datastream: Specify the datastream name.
- Kinesis_Firehose: Specify the firehose name.
- Allow Conditions:
  - Permissions:
    - PutRecord
    - CreateDeliveryStream
    - DeleteDeliveryStream
    - DeleteDeliveryStream
    - ListDeliveryStreams
    - UpdateDestination
    - PutRecordBatch
    - ListTagsForDeliveryStream
    - StartDeliveryStreamEncryption
    - StopDeliveryStreamEncryption
    - TagDeliveryStream
    - UntagDeliveryStream
  - Delegate Admin: Select 'Delegate Admin' to assign administrator rights to the roles, groups, or users specified in the policy. The administrator can edit or delete the policy, and can also create child policies based on the original policy.

Configure MSSQL resource policies

MSSQL supports access, masking, and row level filter policies.

- Database
- Schema
- Table
- Column
- Allow Conditions:
  - Permissions
    - Create Database
    - Create Schema
    - Create Table
    - Select
    - Insert
    - Update
    - Delete
  - Delegate Admin: Select 'Delegate Admin' to assign administrator rights to the roles, groups, or users specified in the policy. The administrator can edit or delete the policy, and can also create child policies based on the original policy.

Configure MSSQL masking policies

MSSQL supports access, masking, and row level filter policies.

- Database
- Schema
- Table
- Column
- Masking Conditions:
  - Permissions
    - Select
  - Select Masking Options:
    - Default
    - Nullify: This option replaces all the characters with NULL value.
    - Unmasked: This option is used when no masking is required.
    - Custom: Using this option you need to mention a custom masked value or expression.

Configure MSSQL row level filter policies

- Database
- Schema
- Table
- Row Level Conditions:
  - Permissions: Click the Add Permissions and tick as 'Select'. At present, only 'Select' permission is available.
  - Row Level Filter: Click the Add Row Filter and enter the valid SQL predicate for whom the policy will be applied based on selected role/groups/users. Note: Row level filtering works by adding the predicate to the query. If the query is not valid, it will fail.

Configure PowerBI resource policies

PowerBI supports access policies.

- Workspace
- Allow Conditions:

- Permissions
  - Contributor
  - Member
  - Admin
  - None
- Delegate Admin: Select 'Delegate Admin' to assign administrator rights to the roles, groups, or users specified in the policy. The administrator can edit or delete the policy, and can also create child policies based on the original policy.

Configure Presto resource policies

Presto supports access, masking, and row level filter policies.

- Catalog: Specify the catalog name.
  - Schema: Specify the schema name.
  - Sessionproperty: Specify the session property.
  - Table: Specify the table name.
  - Procedure: Specify the procedure name.
  - Column: Specify the column name.
- Prestouser:
- Systemproperty:
- Function:
- Allow Conditions:
  - Permissions:
    - Select
    - Insert
    - Create
    - Drop
    - Delete
    - Use
    - Alter
    - Grant
    - Revoke
    - Show
    - Impersonate
    - All
    - Execute
    - Create View
- Delegate Admin: Assign administrator rights to the roles, groups, or users specified in the policy. The administrator can edit or delete the policy, and can also create child policies based on the original policy.

Configure Presto masking policies

- Presto Catalog
- Presto Schema
- Presto Table
- Presto Column
- Masking Conditions:
  - Permissions
    - Select: Tick the permission as 'Select'. At present, only 'Select' permission is available.
  - Select Masking Option: You are allowed to select only one masking option from the below list.
    - Redact: This option masks all the alphabetic characters with 'x' and all numeric characters with 'n'.

- Partial mask: show last 4 – This option shows only the last four characters.
- Partial mask: show first 4 – This option shows only the first four characters.
- Hash: This option replaces all the characters with '#' of the entire cell value.
- Nullify: This option replaces all the characters with NULL value.
- Unmasked (retain original value): This option is used when no masking is required.
- Date: show only year: This option shows only the year portion of a date string and default the month and day to 01/01.
- Custom: Using this option you need to mention a custom masked value or expression.

Configure Presto row level filter policies

- Presto Catalog
- Presto Schema
- Presto Table
- Row Level Conditions:
  - Permissions: Click the Add Permissions and tick as 'Select'. At present, only 'Select' permission is available.
  - Row Level Filter: Click the Add Row Filter and enter the valid SQL predicate to which the policy will be applied based on selected role/groups/users. Note: Row level filtering works by adding the predicate to the query. If the query is not valid, it will fail.

Configure Postgres resource policies

Postgres supports access, masking, and row level filter policies.

- Global
- Database
  - Schema
  - Table
  - Column
- Allow Conditions:
- Permissions:
  - Create Database
  - Connect Database
  - Create Schema
  - Usage Schema
  - Create Table
  - Select
  - Insert
  - Update
  - Delete
  - Truncate
- Delegate Admin: Select 'Delegate Admin' to assign administrator rights to the roles, groups, or users specified in the policy. The administrator can edit or delete the policy, and can also create child policies based on the original policy.

Configure Postgres masking policies

- Database
- Schema
- Table
- Column
- Masking Conditions:
  - Permissions

    - Select
  - Select Masking Option:
    - Default:
    - Nullify: This option replaces all the characters with NULL value.
    - Unmasked: This option is used when no masking is required.
    - Custom: Using this option you need to mention a custom masked value or expression.

Configure Postgres row level filter

- Database
- Schema
- Table
- Row Level Conditions:
  - Permissions: Click the Add Permissions and tick as 'Select'. At present, only 'Select' permission is available.
  - Row Level Filter: Click the Add Row Filter and enter the valid SQL predicate for whom the policy will be applied based on selected role/groups/users. Note: Row level filtering works by adding the predicate to the query. If the query is not valid, it will fail.

Configure Redshift resource policies

Redshift supports access policies.

- Global: Specify the Redshift hosted IP. To get Redshift hosted ip, connect with Redshift environment and run this query: *SELECT inet_server_addr() as host, inet_server_port() as port*
- Database: Specify the database name.
  - Schema: Specify the schema name.
  - Table: Specify the table name.
  - Column: Specify the column name.
- Cluster: Specify the cluster ip.
- Allow Condition:
  - Permissions:
    - Create Database
    - Create Schema
    - Usage Schema
    - Create Table
    - Select
    - Insert
    - Update
    - Delete
    - ListClusters
    - CreateCluster
    - UpdateCluster
    - DeleteCluster
    - ResizeCluster
    - PauseCluster
    - RebootCluster
    - CreateSnapshot
    - RestoreSnapshot
  - Delegate Admin: Select 'Delegate Admin' to assign administrator rights to the roles, groups, or users specified in the policy. The administrator can edit or delete the policy, and can also create child policies based on the original policy.

Configure Snowflake resource policies

Snowflake supports access, masking, and row level filter policies.

- Warehouse: Specify the warehouse name of Snowflake.
  When you select `warehouse`, the following warehouse permissions will be displayed in the **Allow Conditions** > **Permissions** section.
  - Operate
  - UseWarehouse
  - Monitor
  - Modify
- Database: Specify the database name.
  When you select `database`, the following database permissions will be displayed in the **Allow Conditions** > **Permissions** section.
  - CreateSchema
  - UseDB
- Schema: Specify the schema name.
  When you select `schema` along with `database`, the following schema permissions will be displayed in the **Allow Conditions** > **Permissions** section.
  - CreateTmpTable
  - CreateTable
  - UseSchema
  - CreateStream
  - CreateFunction
  - CreateProcedure
  - CreateSequence
  - CreatePipe
  - CreateFileFormat
  - CreateStage
  - CreateExternalTable
- Table: Specify the table name.
  When you select `table` along with `database` and `schema`, the following table permissions will be displayed in the **Allow Conditions** > **Permissions** section.
  - Select
  - Insert
  - Update
  - Delete
  - Truncate
  - References
- Stream: Specify the stream that you have created over standard tables.
  When you select `stream` along with `database` and `schema`, the following stream permission will be displayed in the **Allow Conditions** > **Permissions** section.
  - Select
- Function: Specify the function.
  When you select `function` along with `database` and `schema`, the following function permission will be displayed in the **Allow Conditions** > **Permissions** section.
  - Usage
- Procedure: Specify Snowflake stored procedure.
  When you select `procedure` along with `database` and `schema`, the following procedure permission will be displayed in the **Allow Conditions** > **Permissions** section.
  - Usage
- File_Format: Specify the file format for SQL statement.
  When you select `file_format` along with `database` and `schema`, the following file_format permission will be displayed in the **Allow Conditions** > **Permissions** section.
  - Usage
- Pipe: Specify pipe objects that are created and managed to load data using Snowpipe.

When you select `pipe` along with `database` and `schema`, the following pipe permissions will be displayed in the **Allow Conditions** > **Permissions** section.
  - Operate
  - Monitor
- External_stage: Specify external storage, which is the object storage of the cloud platform.
  When you select `external_stage` along with `database` and `schema`, the following external_stage permission will be displayed in the **Allow Conditions** > **Permissions** section.
  - Usage
- Internal_stage: Specify internal storage, which is the database storage.
  When you select `internal_stage` along with `database` and `schema`, the following Internal_stage permissions will be displayed in the **Allow Conditions** > **Permissions** section.
  - Read
  - Write
- Sequence: Specify Snowflake sequence objects.
  When you select `sequence` along with `database` and `schema`, the following sequence permission will be displayed in the **Allow Conditions** > **Permissions** section.
  - Usage
- Column: Specify the column name.
  When you select `column` along with `database`, `schema` and `table`, the following `column` permissions will be displayed in the **Allow Conditions** > **Permissions** section.
  - Select
  - Insert
  - Update
  - Delete
  - Truncate
  - References
- Global: Specify the snowflake account name. To get the snowflake account name, connect with Snowflake environment and run this query: *select current_account() as account*
  When you select `global`, the following global permissions will be displayed in the **Allow Conditions** > **Permissions** section.
  - CreateWarehouse
  - CreateDatabase
- Delegate Admin: Select the `Delegate Admin` checkbox to assign administrator rights to the roles, groups, or users specified in the policy. The administrator can edit or delete the policy, and can also create child policies based on the original policy.

NOTE
When you create a policy for a table with UPDATE and DELETE permissions granted to a user/group/role, you must choose the SELECT permission along with it.

Configure Snowflake masking policies

- Database: Specify the database name.
- Schema: Specify the schema name.
- Table/View: Specify the table or view name.
- Column: Specify the column name.
- Masking Conditions:
  - Permissions: Tick the permission as 'Select'. At present, only 'Select' permission is available.

- Select Masking Option: If a masking option is applied to a data type that is not supported, then the default masking value is applied. You are allowed to select only one masking option from the following list:
  - Default: This option masks column with default value specified by its datatype's property.
    The following are the default data type property values:
    - `SNOWFLAKE_MASKED_NUMBER_VALUE=0`
    - `SNOWFLAKE_MASKED_DOUBLE_VALUE=0`
    - `SNOWFLAKE_MASKED_TEXT_VALUE='{{MASKED}}'`
  - Hash: Returns a hex-encoded string containing the N-bit SHA-2 of the volume in the column, where N is the specified output digest size.
    Internal Function: SHA2({col})
    Supported Data Type: Text
    For more information see SHA2, SHA2_HEX.
  - Nullify: This option replaces all the characters with NULL value.
    Supported Data Type: All Data Types
  - Unmasked (retain original value): This option is used when no masking is required.
    Supported Data Type: All Data Types
  - Regular expression:
    Internal Function: regexp_replace({col},'{value_or_expr}','{replace_value}')
    Supported Data Type: Text
    For more information see REGEX_REPLACE.
  - Literal mask: This option replaces entire cell value with given character.
    Supported Data Type: Text
  - Partial mask: show last 4 - This option shows only the last four characters.
    Internal Function: regexp_replace({col},'(..)(.{4})(.)','***\2')
    Supported Data Type: Text
    For more information see REGEX_REPLACE.
  - Partial mask: show first 4 - This option shows only the first four characters.
    Internal Function: regexp_replace({col},'.','*','5')
    Supported Data Type: Text
    For more information see REGEX_REPLACE.
  - Protect:
    Supported Data Type: Text
    For more information see PEG REST API on Privacera Platform.
  - Unprotect:
    Supported Data Type: Text
    For more information see PEG REST API on Privacera Platform.
  - Custom: Using this option you need to mention a custom masked value or expression.

Configure Snowflake row level filter policies

- Database: Specify the database name.
- Schema: Specify the schema name.
- Table: Specify the table name.
- Row Level Conditions:
  - Permissions: Click the Add Permissions and tick as 'Select'. At present, only 'Select' permission is available.
  - Row Level Filter: Click the Add Row Filter and enter the valid SQL predicate for whom the policy will be applied based on selected role/groups/users. Note: Row level filtering works by adding the predicate to the query. If the query is not valid, it will fail.

Configure Policy with Attribute-Based Access Control (ABAC) on Privacera-Cloud

PrivaceraCloud enables use of user, group, resource, classification, and environment attributes in authorization policies. Attribute-Based Access Control (ABAC) makes it possible to express authorization policies without prior knowledge of specific resources or specific users, which helps avoid the need for new policies as new resources or users are introduced.

Overview

The following scenarios can be accomplished with ABAC configuration:

ABAC in row filter expressions

By using Attribute-Based Access Control (ABAC) for row filters, user can define attribute based conditions in row filter expressions. For example, user/group attributes can be referenced in a row filter expression:

```
dept = ${{USER.dept}}
state in ( ${{GET_UG_ATTR_Q_CSV('state')}} )
```

ABAC in resource definitions

Attributes can also be included in resource names:

```
path: /home/${{USER._name}}
path: /departments/${{USER.dept}}
database: dept_${{USER.dept}}p
```

ABAC in policy conditions in Resource based access policies

With the ABAC feature, you can configure resource policies based on user attributes from your LDAP or AD service.

You can assign attributes to users, groups and tags in policies. You can also implement logical conditions on the user attributes for the resource policies.

Attributes can be referenced using expressions:

```
USER.employeeType != 'intern'

TAG.piiType == 'email'

TAG.sensitivityLevel <= USER.allowedSensitivityLevel
```

NOTE

ABAC for policy conditions in resource-based access policies is supported for the following data sources:

- Databricks/EMR Hive, Spark, and all services using `privacera_hive` service definitions.
- PolicySync Snowflake

Prerequisites

- Import the users from the LDAP or AD directory to the Privacera Ranger database. See Connect users to PrivaceraCloud to learn more.

- Determine the resources you want to protect with ABAC-based policies.

Setup User/Group Attributes

The users and groups with their attributes can be synced with Usersync from LDAP or AD. With users, the user attributes are also synced from source (such as LDAP or AD).

Add/Edit Attributes in the PrivaceraCloud Portal

See Users, groups, and roles [13].

Example policy with ABAC

1. Let's say we have a user with the following attribute: `dept` : `IT`
2. We create a resource access policy such as:
   Policy Details

Policy Name	access	Normal Enabled
Hive Database	hive_test	Include
Hive Table	employee	Include
Hive Column	*	Include
Description		
Audit Log	Yes	
Policy Labels	--	

   Allow Conditions

Select Role	Select Group	Select User	Policy Conditions	Permissions	Delegate Admin
		tejas, sally		select	(unchecked)

3. We created a **Row Filter Policy** as below:

Policy Name	employee_rlf	Normal Enabled
Hive Database	hive_test	
Hive Table	employee	
Description		
Audit Log	Yes	
Policy Labels	--	

   Row Level Conditions

Select Role	Select Group	Select User	Policy Conditions	Access Types	Row Level Filter
	public			select	dept == '${{USER.dept}}'

   Note that the row filter expression used here is: `dept == '${{USER.dept}}'`
4. Now the setup is done on EMR Hive. The table `hive_test.employee` is available on the EMR Hive cluster, with data as below:

```
| employee.id  |  employee.name  |  employee.location |  employee.dept  |
+--------------+-----------------+--------------------+-----------------+
| 1            |  manny          |  mumbai            | IT
| 2            |  novak          |  NY                | HR
| 3            |  Pablo          |  Barcelona         | HR
| 4            |  Carlos         |  Mexico            | Support
+--------------+-----------------+--------------------+-----------------+
4 rows selected
```

5. The user `tejas` is available on the EMR cluster to run the query. When select query is executed by `tejas`:

```
| employee.id  |  employee.name  |  employee.location |  employee.dept  |
+--------------+-----------------+--------------------+-----------------+
| 1            |  manny          |  mumbai            | IT
+--------------+-----------------+--------------------+-----------------+
1 row selected
```

Attribute-based access control (ABAC) macros

Attribute-based access control (ABAC) supports a number of macros to make it easier to write frequently-used conditions.

The following table lists macros provided by Privacera for ABAC:

Table 10.

Name	Description	Sample Usage
USER	User accessing the resource.	`USER.dept == 'finance'/department/${{USER.dept}}`
TAG	Current tag - use only in tag-based policy	TAG.piiType == 'email'
UGNAMES	Name of groups the user belongs to	UGNAMES.indexOf('interns') == -1
URNAMES	Name of roles the user belongs to	URNAMES.indexOf('admin') != -1
TAGNAMES	Name of tags associated with accessed resource	TAGNAMES.indexOf('PII') != -1 TAGNAMES.indexOf('FINANCE')
UG_NAMES_Q_CSV	Quoted name of groups the user belong to, separated by comma. For example: 'grp1','grp2'	Row filter:group_name in ($ {{UG_NAMES_Q_CSV}})
UR_NAMES_Q_CSV	Quoted name of roles the user belong to, separated by comma. For example: 'role1','role2'	Row filter:role_name in ($ {{UR_NAMES_Q_CSV}})
GET_UG_ATTR_Q_CSV	Quoted attribute values of groups the user belongs to, separated by comma. For example: 'store1','store2'	Row filter:store_name in ($ {{GET_UG_ATTR_Q_CSV('managesStore'}})
IS_IN_GROUP	User accessing the resource belongs to a specific group	IS_IN_GROUP('sales')
IS_IN_ROLE	User accessing the resource belongs to a specific role	IS_IN_ROLE('accounts')
HAS_TAG	Resource being access has a specific tag	(HAS_TAG('PERSON_NAME'))
HAS_USER_ATTR	User accessing the resource has a specific user attribute	HAS_USER_ATTR('activities')
HAS_UG_ATTR	User accessing the resource has a specific group attribute	HAS_UG_ATTR('marketing')
HAS_TAG_ATTR	Resource being access has a specific tag attribute	(HAS_TAG_ATTR('identification'))

It is sometimes necessary to setup permissions for users who do or don't belong to any group or any role. The following macros will make it easier to create those permissions:

Table 11.

Name	Description	Sample usage
IS_IN_ANY_GROUP	This macro can be used in policy conditions to ALLOW/DENY policy items. If the user who is accessing the resource is a member of any group, it returns true.	IS_IN_ANY_GROUP
IS_IN_ANY_ROLE	This macro can be used in policy conditions to ALLOW/DENY policy items If the user who is accessing the resource has any role, it returns true.	IS_IN_ANY_ROLE

Name	Description	Sample usage
IS_NOT_IN_ANY_GROUP	This macro can be used in policy conditions to ALLOW/DENY policy items If the user who is accessing the resource does not belong to any groups, it returns true.	IS_NOT_IN_ANY_GROUP
IS_NOT_IN_ANY_ROLE	This macro can be used in policy conditions to ALLOW/DENY policy items If the user who is accessing the resource does not have any roles, it returns true.	IS_NOT_IN_ANY_ROLE

The following macros will make it easier to check if current resource has any tags or not

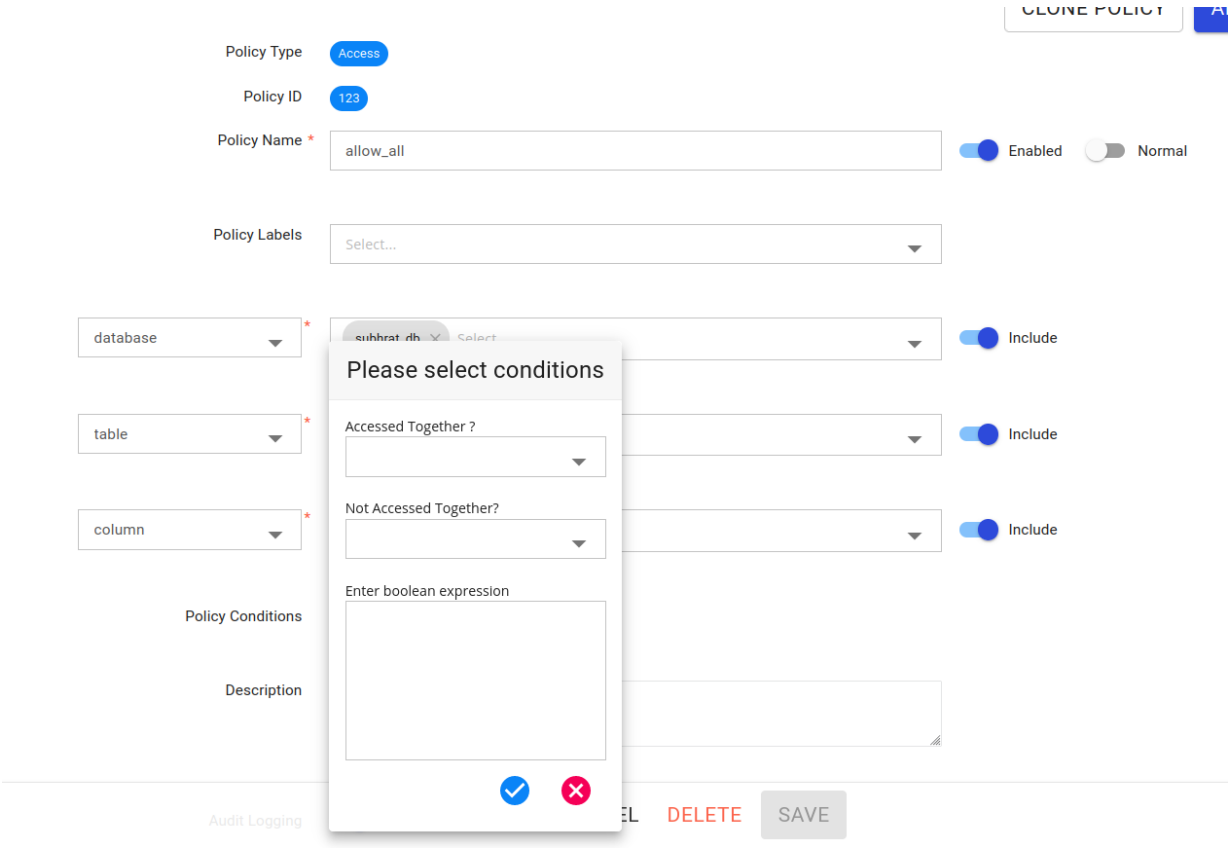
Table 12.

Name	Description	Sample usage
HAS_ANY_TAG	This macro can be used in policy conditions to ALLOW/DENY policy items If the user who is accessing the resource has any tags, this method returns true.	HAS_ANY_TAG
HAS_NO_TAG	This macro can be used in policy conditions to ALLOW/DENY policy items If the user who is accessing the resource does not have any tags, it returns true.	HAS_NO_TAG

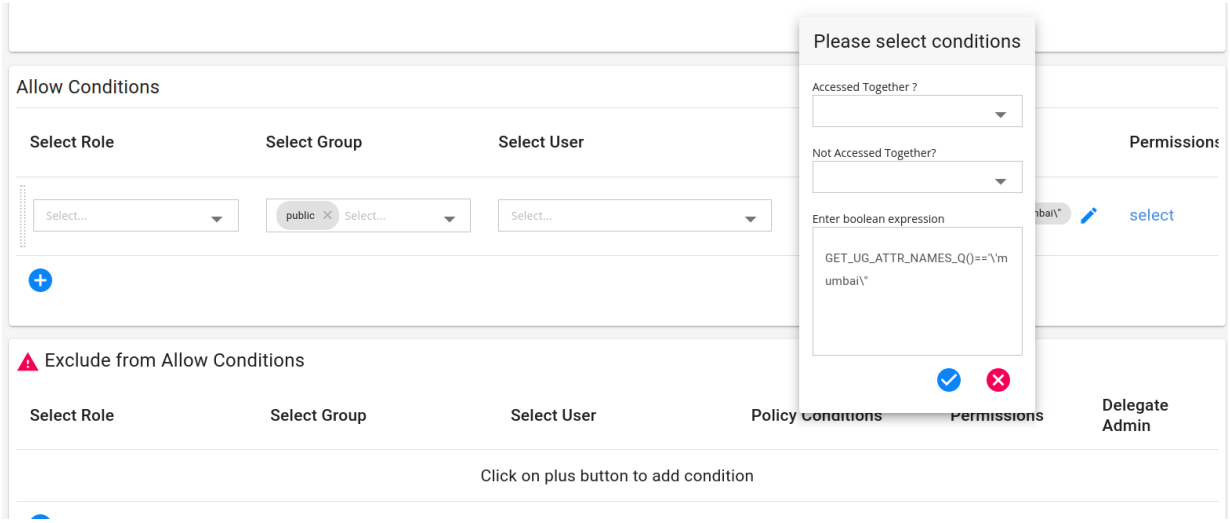
Additional ABAC macros

NOTE

- When policy condition is mentioned, it means policy conditions text box (that expects boolean expression) in following places in policy definitions:
- Policy condition inPolicy Detail in all policy types i.e. Resource Access Policy, Masking Policy, Row Filter Policy and Tag Policy.



- Policy condition inAllow/Deny Policy Items in all policy types i.e. Resource Access Policy, Masking Policy, Row Filter Policy and Tag Policy.



- Some macros (e.g. IS_IN_ANY_GROUP – not described in this document) return boolean value i.e. true or false, they can be directly used in policy conditions, which expect boolean expression.
- Some macros (all the macros in this document) return non-boolean values i.e. list of Strings with or without quotes separated by comma. These macros can be used in row filter expressions, masking condition expressions and policy conditions also (with some limitations).
- Policy conditions do not allow use of the Javascript comparison operator “IN”. So if the macros (which return a list of Strings) are to be used in Policy Conditions, they need to be used with the equality operator i.e. “==”.

Summary

Macro	With default value	Description	Example return value
GET_UG_NAMES_Q()	GET_UG_NAMES_Q('none')	Names of groups (in single quotes) the user belongs to, separated by a comma	‘analyst’,’manager’
GET_UG_NAMES()	GET_UG_NAMES('none')	Names of groups the user belongs to, separated by a comma	analyst,manager
GET_UG_ATTR_NAMES_Q()	GET_UG_ATTR_NAMES_Q('none')	Names of all attributes (in single quotes) in groups the user belongs to, separated by a comma	‘dept’,’site’
GET_UG_ATTR_NAMES()	GET_UG_ATTR_NAMES('none')	Names of all attributes in groups the user belongs to, separated by a comma	dept,site
GET_UG_ATTR_Q('site')	GET_UG_ATTR_Q('site', 'none')	Attribute value (in single quotes) in groups the user belongs to, separated by a comma	‘10’,’20’

GET_UG_ATTR('site')	GET_UG_ATTR('site', 'none')	Attribute value in groups the user belongs to, separated by a comma	10,20
GET_USER_ATTR_NAMES()	GET_USER_ATTR_NAMES('none')	Names of all attributes of the user, separated by a comma	name,email
GET_USER_ATTR_NAMES_Q()	GET_USER_ATTR_NAMES_Q('none')	Names of all attributes (in single quotes) of the user, separated by a comma	‘name’,’email’
GET_USER_ATTR('email')	GET_USER_ATTR('email', 'none')	Value of user attribute	name@domain
GET_USER_ATTR_Q('email')	GET_USER_ATTR_Q('email', 'none')	Value of user attribute (in single quotes)	‘name@domain’
GET_UR_NAMES()	GET_UR_NAMES('none')	Names of roles assigned to the user, separated by a comma	data-steward,admin
GET_UR_NAMES_Q()	GET_UR_NAMES_Q('none')	Names of roles (in single quotes) assigned to the user, separated by a comma	‘data-steward’,’admin’

Groups

GET_UG_NAMES_Q()

Description: returns the list of user groups (with each group name in quotes) to which the querying user is a member. Can be used for comparison of character values.

Example:

- The user joe is a member of groups admin and manager,GET_UG_NAMES_Q() will return‘admin’,‘manager’.

Usage: can be used in:

- Row Level Filter Policies - row level filters expressions
- Example:location in (${{GET_UG_NAMES_Q()}})
- Masking policy - Custom masking condition
- Example:CASE WHEN location in (${{GET_UG_NAMES_Q()}}) THEN {col} ELSE 'NONE' END
- Policy conditions:
- Example:GET_UG_NAMES_Q()=='\’managers\"

With default Value - GET_UG_NAMES_Q(‘default_value’)

Description: returns the list of user groups (with each group name in quotes) to which the querying user is a member. If the user is not a member of any group, ‘default_value’ will be returned. Can be used for comparison of character values.

Example:

- The user is a member of groups admin and manager,GET_UG_NAMES_Q(‘none’) will return‘admin’,‘manager’.
- The user is not a member of any group,GET_UG_NAMES_Q(‘none’) will return‘none’.

Usage: can be used in:

- Row Level Filter Policies - row level filters expressions
- Example:location in (${{GET_UG_NAMES_Q(‘none’)}})
- Masking policy - Custom masking condition
- Example:CASE WHEN location in (${{GET_UG_NAMES_Q(‘none’)}}) THEN {col} ELSE 'NONE' END
- Policy conditions
- Example:GET_UG_NAMES_Q(‘operator’)=='\’managers\"

GET_UG_NAMES()

Description: returns the list of user groups (without quotes) to which the querying user is a member. Can be used for comparison of numeric values.

Example:

- The user joe is a member of groups admin and manager,GET_UG_NAMES() will returnadmin,manager.

Usage: can be used in:

- Row Level Filter Policies - row level filters expressions
- Example:dept_id in (${{GET_UG_NAMES()}})
- Masking policy - Custom masking condition
- Example:CASE WHEN dept_id in (${{GET_UG_NAMES()}}) THEN {col} ELSE '0' END
- Policy condition
- Example:GET_UG_NAMES()==’managers'

With default Value - GET_UG_NAMES(‘default_value’)

Description: returns the list of user groups to which the querying user is a member. If the user is not a member of any group, ‘default_value’ will be returned. Can be used for comparison of numeric values.

Example:

- The user is a member of groupsadmin andmanager,GET_UG_NAMES(‘none’) will returnadmin,manager.
- The user is not a member of any group,GET_UG_NAMES(‘none’) will returnnone.

Usage: can be used in:

- Row Level Filter Policies - row level filters expressions
- Example:dept_id in (${{GET_UG_NAMES(‘0’)}})
- Masking policy - Custom masking condition
- Example:CASE WHEN dept_id in (${{GET_UG_NAMES(‘0’)}}) THEN {col} ELSE '0' END
- Policy condition
- Example:GET_UG_NAMES(‘operators’)==’managers'

GET_UG_ATTR_NAMES_Q()

Description: returns the list of attributes mapped to user groups (with each attribute name in single quotes) to which the querying user is a member. Can be used for comparison of character values.

Example:

- The user joe is a member of the groupsadmin(attribute: access=auditor) andmanager (attribute: location=NY, access=admin).
- GET_UG_ATTR_NAMES_Q() will return‘access’,‘location’.

Usage: can be used in:

- Row Level Filter Policies - row level filters expressions
- Example:data_module in (${{GET_UG_ATTR_NAMES_Q()}})
- Masking policy - Custom masking condition
- Example:CASE WHEN data_module in (${{GET_UG_ATTR_NAMES_Q()}}) THEN {col} ELSE 'NONE' END
- Policy conditions
- Example:GET_UG_ATTR_NAMES_Q()=='\’location\"

With default Value - GET_UG_ATTR_NAMES_Q(‘default_value’)

Description: returns the list of attributes mapped to the user groups (with each attribute name in single quotes) to which the querying user is a member. If the user is not a member of any group or if the group to which the user is a member does not have any attributes, ‘default_value’ will be returned. Can be used for comparison of character values.

Example:

- The user joe is a member of the groupsadmin(attribute: access=auditor) andmanager (attribute: location=NY, access=admin). GET_UG_ATTR_NAMES_Q(‘none’) will return‘access’,‘location’.
- The user is a member of groups admin and manager (both the groups don’t have any attributes),GET_UG_ATTR_NAMES_Q(‘none’) will return‘none’.
- The user is not a member of any group,GET_UG_ATTR_NAMES_Q(‘none’) will return‘none’.

Usage: can be used in:

- Row Level Filter Policies - row level filters expressions
- Example:location in (${{GET_UG_ATTR_NAMES_Q(‘none’)}})
- Masking policy - Custom masking condition
- Example:CASE WHEN location in (${{GET_UG_ATTR_NAMES_Q(‘none’)}}) THEN {col} ELSE '0' END
- Policy conditions:
- Example: GET_UG_ATTR_NAMES_Q('dept')=='\’location\"

GET_UG_ATTR_NAMES()

Description: returns the list of attributes keys mapped to the user groups to which the querying user is a member. Can be used for comparison of numeric values.

Example:

- The user joe is a member of the groupadmin(attribute: 10=auditor).
- GET_UG_ATTR_NAMES() will return10.

Usage: can be used in:

- Row Level Filter Policies - row level filters expressions
- Example:dept_id in (${{GET_UG_ATTR_NAMES()}})
- Masking policy - Custom masking condition
- Example:CASE WHEN dept_id in (${{GET_UG_ATTR_NAMES()}}) THEN {col} ELSE '0' END
- Policy conditions:
- Example: GET_UG_ATTR_NAMES()=='location'

With default Value - GET_UG_ATTR_NAMES(‘default_value’)

Description: returns the list of attributes keys mapped to the user groups to which the querying user is a member. If the user is not a member of any group or the group of which the user is a member does not have any attribute, ‘default_value’ will be returned. Can be used for comparison of numeric values.

Example:

- The user is a member of the groupadmin(attribute: 10=auditor), GET_UG_ATTR_NAMES(‘20’) will return10.
- The user is a member of groupadmin(without any attributes),GET_UG_ATTR_NAMES(‘20’) will return‘20’.
- The user is not a member of any group,GET_UG_ATTR_NAMES(‘20’) will return20.

Usage: can be used in:

- Row Level Filter Policies - row level filters expressions
- Example:dept_id in (${{GET_UG_ATTR_NAMES(‘20’)}})
- Masking policy - Custom masking condition
- Example:CASE WHEN dept_id in (${{GET_UG_ATTR_NAMES(‘20’)}}) THEN {col} ELSE '0' END
- Policy conditions:
- Example: GET_UG_ATTR_NAMES('dept')=='location'

GET_UG_ATTR_Q(‘attribute_key’)

Description: returns the list of values of attribute (attribute_key) mapped to user groups (with each attribute name in single quotes) to which the querying user is a member. Can be used for comparison of character values.

Example:

- The user joe is a member of the groupadmin(attribute: location=mumbai).
- GET_UG_ATTR_Q(‘location’) will return‘mumbai’.

Usage: can be used in:

- Row Level Filter Policies - row level filters expressions
- Example:location in ${{GET_UG_ATTR_Q('location')}}

- Masking policy - Custom masking condition
- Example:CASE WHEN location in (${{GET_UG_ATTR_Q(‘location’)}}) THEN {col} ELSE 'NONE' END
- Policy condition
- Example:GET_UG_ATTR_Q('location')==’\'mumbai\''

With default Value - GET_UG_ATTR_Q(‘attribute_key’,‘default_value’)

Description: returns the list of attributes mapped to the user groups (with each attribute name in single quotes) to which the querying user is a member. If the groups mapped to the user does not have an attribute‘attribute_key’ or if the user is not mapped to any group, ‘default_value’ will be returned. Can be used for comparison of character values.

Example:

- The user joe is a member of the groupsadmin(attribute: access=auditor) andmanager (attribute: location=NY, access=admin). GET_UG_ATTR_Q(‘location’,‘none’) will return‘mumbai’.
- The user is a member of groups admin and manager (both the groups don’t have any attributes),GET_UG_ATTR_Q(‘location’,‘none’) will return‘none’.
- The user is not a member of any group,GET_UG_ATTR_Q(‘location’,‘none’) will return‘none’.

Usage: can be used in:

- Row Level Filter Policies - row level filters expressions
- Example:location in (${{GET_UG_ATTR_Q(‘location’,‘none’)}})
- Masking policy - Custom masking condition
- Example:CASE WHEN location in (${{GET_UG_ATTR_Q(‘location’,‘none’)}}) THEN {col} ELSE 'none' END
- Policy condition
- Example:GET_UG_ATTR_Q('location','NY’)==’\'mumbai\''

GET_UG_ATTR(‘attribute_key’)

Description: returns the list of attributes (separated by comma) mapped to the user groups to which the querying user is a member. Can be used for comparison of numeric values.

Example:

- The user joe is a member of the groupsadmin(attribute: dept=10) andmanager (attribute: dept=20), GET_UG_ATTR(‘dept’) will return10,20.

Usage: can be used in:

- Row Level Filter Policies - row level filters expressions
- Example:dept_id in (${{GET_UG_ATTR(‘dept’)}})
- Masking policy - Custom masking condition
- Example:CASE WHEN dept_id in (${{GET_UG_ATTR(‘dept’)}}) THEN {col} ELSE '0' END
- Policy condition

- Example:GET_UG_ATTR('location')=='mumbai'

With default Value - GET_UG_ATTR(‘attribute_key’,‘default_value’)

Description: returns the list of user groups to which the querying user is a member. If the user is not a member of any group, ‘default_value’ will be returned. Can be used for comparison of numeric values.

Example:

- The user joe is a member of the groupsadmin(attribute: dept=10) andmanager (attribute: dept=20), GET_UG_ATTR(‘dept’,’30’) will return10,20.
- The user is a member of groupsadmin andmanager(without any attributes),GET_UG_ATTR(‘dept’,’30’) will return30.
- The user is not a member of any group,GET_UG_ATTR(‘dept’,’30’) will return30.

Usage: can be used in:

- Row Level Filter Policies - row level filters expressions
- Example:dept_id in (${{GET_UG_NAMES_Q(‘dept’,’30’)}})
- Masking policy - Custom masking condition
- Example:CASE WHEN dept_id in (${{GET_UG_NAMES_Q(‘dept’,’30’)}}) THEN {col} ELSE '0' END
- Policy condition
- Example:GET_UG_ATTR('location',’NY’)=='mumbai'

Users

GET_USER_ATTR_NAMES()

Description: returns the list of user attribute keys. Can be used for comparison of numeric values.

Example:

- The user has an attribute: 10=admin,GET_USER_ATTR_NAMES() will return10.

Usage: can be used in:

- Row Level Filter Policies - row level filters expressions
- Example:dept_id in (${{GET_USER_ATTR_NAMES()}})
- Masking policy - Custom masking condition
- Example:CASE WHEN dept_id in (${{GET_USER_ATTR_NAMES()}}) THEN {col} ELSE '0' END
- Policy condition
- Example:GET_USER_ATTR_NAMES()=='location'

With default Value - GET_USER_ATTR_NAMES(‘default_value’)

Description: returns the list of attribute keys. If the user does not have any attribute, ‘default_value’ will be returned. Can be used for comparison of numeric values.

Example:

- The user has an attribute: 10=admin,GET_USER_ATTR_NAMES(‘20’) will return10.
- The user does not have any attribute,GET_USER_ATTR_NAMES(‘20’) will return20.

Usage: can be used in:

- Row Level Filter Policies - row level filters expressions
- Example:dept_id in (${{GET_USER_ATTR_NAMES(‘20’)}})
- Masking policy - Custom masking condition
- Example:CASE WHEN dept_id in (${{GET_USER_ATTR_NAMES(‘20’)}}) THEN {col} ELSE 'NONE' END
- Policy condition
- Example:GET_USER_ATTR_NAMES(‘none’)=='location'

GET_USER_ATTR_NAMES_Q()

Description: returns the list of user attribute keys (in single quotes, separated by comma). Can be used for comparison of character values.

Example:

- The user has an attribute: access=auditor, GET_USER_ATTR_NAMES_Q() will return‘access’.

Usage: can be used in:

- Row Level Filter Policies - row level filters expressions
- Example:modules in (${{GET_USER_ATTR_NAMES_Q()}})
- Masking policy - Custom masking condition
- Example:CASE WHEN modules in (${{GET_USER_ATTR_NAMES_Q()}}) THEN {col} ELSE 'NONE' END
- Policy condition
- Example:GET_USER_ATTR_NAMES_Q()=='\'location\''

With default Value - GET_USER_ATTR_NAMES(‘default_value’)

Description: returns the list of attribute keys (in single quotes, separated by comma). If the user does not have any attribute, ‘default_value’ will be returned. Can be used for comparison of character values.

Example:

- The user has an attribute: access=auditor, GET_USER_ATTR_NAMES_Q(‘data’) will return‘access’.
- The user does not have any attribute,GET_USER_ATTR_NAMES_Q(‘data’) will return‘data’.

Usage: can be used in:

- Row Level Filter Policies - row level filters expressions
- Example:modules in (${{GET_USER_ATTR_NAMES_Q(‘data’)}})
- Masking policy - Custom masking condition
- Example:CASE WHEN modules in (${{GET_USER_ATTR_NAMES_Q(‘data’)}}) THEN {col} ELSE 'NONE' END
- Policy condition

- Example:GET_USER_ATTR_NAMES_Q(‘none’)==’\'location\'"

GET_USER_ATTR(‘attribute_key’)

Description: returns the list of attributes mapped to the user (separated by comma). Can be used for comparison of numeric or character values.

Example:

- The user has an attribute: dept=10, GET_USER_ATTR(‘dept’) will return‘NY’.

Usage: can be used in:

- Row Level Filter Policies - row level filters expressions

- Example:dept_id in (${{GET_USER_ATTR(‘dept’)}})
- Example:location=’${{GET_USER_ATTR(‘location’)}}’

- Masking policy - Custom masking condition

- Example:CASE WHEN dept_id in (${{GET_USER_ATTR(‘dept’)}}) THEN {col} ELSE '0' END

- Policy condition

- Example:GET_USER_ATTR(‘location’)=='mumbai'

With default Value - GET_UG_ATTR(‘attribute_key’,‘default_value’)

Description: returns the list of user groups to which the querying user is a member. If the user is not a member of any group, ‘default_value’ will be returned. Can be used for comparison of numeric or character values.

Example:

- The user has an attribute: dept=10,GET_USER_ATTR(‘dept’,’20’) will return‘10’.
- The user does not have any attribute or does not have a‘dept’ attribute,GET_USER_ATTR(‘dept’,’20’) will return‘20’.

Usage: can be used in:

- Row Level Filter Policies - row level filters expressions

- Example:dept_id in (${{GET_USER_ATTR(‘dept’,’20’)}})
- Example:location=’${{GET_USER_ATTR(‘location’,’NY’)}}’

- Masking policy - Custom masking condition

- Example:CASE WHEN dept_id in (${{GET_USER_ATTR(‘dept’,’20’)}}) THEN {col} ELSE '0' END

- Policy condition

- Example:GET_USER_ATTR(‘location’,‘NY’)=='mumbai'

GET_USER_ATTR_Q(‘attribute_key’)

Description: returns the list of values of attribute (attribute_key) mapped to the user (with each attribute name in single quotes, separated by comma). Can be used for comparison of character values.

Example:

- The user joe is a member of the groupadmin(attribute: location=mumbai), GET_USER_ATTR_Q(‘location’) will return‘mumbai’.

Usage: can be used in:

- Row Level Filter Policies - row level filters expressions
- Example:location in ${{GET_USER_ATTR_Q('location')}}
- Masking policy - Custom masking condition
- Example:CASE WHEN location in (${{GET_USER_ATTR_Q('location')}}) THEN {col} ELSE 'NONE' END
- Policy condition
- Example:GET_USER_ATTR_Q(‘location’)==’\'mumbai\''

With default Value - GET_USER_ATTR_Q(‘attribute_key’,‘default_value’)

Description: returns the list of attributes mapped to the user (with each attribute name in single quotes, separated by comma). If the user does not have an attribute‘attribute_key’, ‘default_value’ will be returned. Can be used for comparison of character values.

Example:

- The user has an attribute: access=mumbai,GET_USER_ATTR_Q(‘location’,‘NY’) will return‘mumbai’.
- The user does not have an attribute,GET_USER_ATTR_Q(‘location’,‘NY’) will return‘NY’.

Usage: can be used in:

- Row Level Filter Policies - row level filters expressions
- Example:location in (${{GET_USER_ATTR_Q(‘location’,‘NY’)}})
- Masking policy - Custom masking condition
- Example:CASE WHEN location in (${{GET_USER_ATTR_Q(‘location’,‘NY’)}}) THEN {col} ELSE 'none' END
- Policy condition
- Example:GET_USER_ATTR_Q(‘location’,‘NY’)==’\'mumbai\''

Roles

GET_UR_NAMES()

Description: returns the list of user roles (without quotes, separated by comma) to which the querying user is a member. Can be used for comparison of numeric or character values.

Example:

- The user is a member of the role admin and manager,GET_UR_NAMES() will returnadmin, manager.

Usage: can be used in:

- Row Level Filter Policies - row level filters expressions
- Example:dept_id in (${{GET_UR_NAMES()}})
- Example:location=’${{GET_UR_NAMES()}}’
- Masking policy - Custom masking condition
- Example:CASE WHEN dept_id in (${{GET_UR_NAMES()}}) THEN {col} ELSE '0' END

- Policy condition
- Example:GET_UR_NAMES()=='admin'

With default Value - GET_UR_NAMES(‘default_value’)

Description: returns the list of user roles (without quotes, separated by comma) to which the querying user is a member. If the user is not a member of any role, ‘default_value’ will be returned. Can be used for comparison of numeric or character values.

Example:

- The user is a member of rolesadmin andmanager,GET_UR_NAMES(‘none’) will returnadmin,manager.
- The user is not a member of any role,GET_UR_NAMES(‘none’) will returnnone.

Usage: can be used in:

- Row Level Filter Policies - row level filters expressions
- Example:dept_id in (${{GET_UR_NAMES(‘20’)}})
- Example:location=’${{GET_UR_NAMES(‘NY’)}}’
- Masking policy - Custom masking condition
- Example:CASE WHEN dept_id in (${{GET_UR_NAMES(‘20’)}}) THEN {col} ELSE '0' END
- Policy condition
- Example:GET_UR_NAMES(‘operator’)==’admin’

GET_UR_NAMES_Q()

Description: returns the list of user roles (with each role name in quotes, separated by comma) to which the querying user is a member. Can be used for comparison of character values.

Example:

- The user is a member of rolesadmin andmanager,GET_UR_NAMES_Q() will return‘admin’,‘manager’.

Usage: can be used in:

- Row Level Filter Policies - row level filters expressions
- Example:emp_roles in (${{GET_UR_NAMES_Q()}})
- Masking policy - Custom masking condition
- Example:CASE WHEN emp_roles in (${{GET_UR_NAMES_Q()}}) THEN {col} ELSE 'NONE' END
- Policy condition
- Example:GET_UR_NAMES_Q()==’\’admin\’’

With default Value - GET_UG_NAMES_Q(‘default_value’)

Description: returns the list of user groups (with each group name in quotes) to which the querying user is a member. If the user is not a member of any group, ‘default_value’ will be returned. Can be used for comparison of character values.

Example:

- The user is a member of groups admin and manager,GET_UR_NAMES_Q(‘executive’) will return‘admin’,‘manager’.
- The user is not a member of any role,GET_UR_NAMES_Q(‘executive’) will return‘executive’.

Usage: can be used in:

- Row Level Filter Policies - row level filters expressions
- Example:location in (${{GET_UR_NAMES_Q(‘executive’)}})
- Masking policy - Custom masking condition
- Example:CASE WHEN location in (${{GET_UR_NAMES_Q(‘executive’)}}) THEN {col} ELSE 'NONE' END
- Policy condition
- Example:GET_UR_NAMES_Q(‘operator’)==’\’admin\’’

Configure access policies for AWS services on Privacera Platform

Use Access Management to configure resource-based services and add access policies to them.

Set up a data access server environment

1. From the home page, click **Launch Pad**.
2. Click **AWS Cli** and follow the prompts.

Set up proxy for user

- From a terminal prompt, enable the proxy.

  ```
  ~/privacera_aws.sh --enable-proxy
  ```

Use S3 with data access server

1. From a terminal prompt, list the contents of *test-bucket.* (${test-bucket}) is mentioned as an example across the document. You can change the bucket name as per your choice.

   ```
   aws s3 ls s3://test-bucket
   ```
2. From a terminal prompt, copy a local file to test-bucket.

   ```
   aws s3 cp srcFile.txt s3://test-bucket/dstFile.txt
   ```

   It will show the following result: **upload failed: ./srcFile.txt to s3://test-bucket/dstFile.txt An error occurred (403) when calling the PutObject operation: Forbidden**. This indicates that the current user doesn’t have permission to perform this operation.

Set S3 policy in Privacera

Create a policy to allow the user access to test-bucket for READ and WRITE operations.

1. From the home page, click **Access Management** > **Resource Policies**.
2. On the **Resource Policies** page, click **privacera_s3** > **Add New Policy**.
3. Enter the details.
   - Policy Name: s3_test_policy
   - Bucket Name: test-bucket (S3 Bucket Name)
   - Object Path: * (File/Directory/Object Path Inside Bucket)
   - Under Allow Conditions, click '**+**' and select:

- User: User's username to which you want to allow access.
- Add Permission as: read, metadata read, write, metadata write

4. Click **Save**.

Copy a file to S3

1. From a terminal prompt, copy the local file to **test-bucket**.

```
aws s3 cp srcFile.txt s3://test-bucket/dstFile.txt
```

2. Verify that the copy was successful.

```
aws s3 ls s3://test-bucket/
```

Configure policy with conditional masking on Privacera Platform

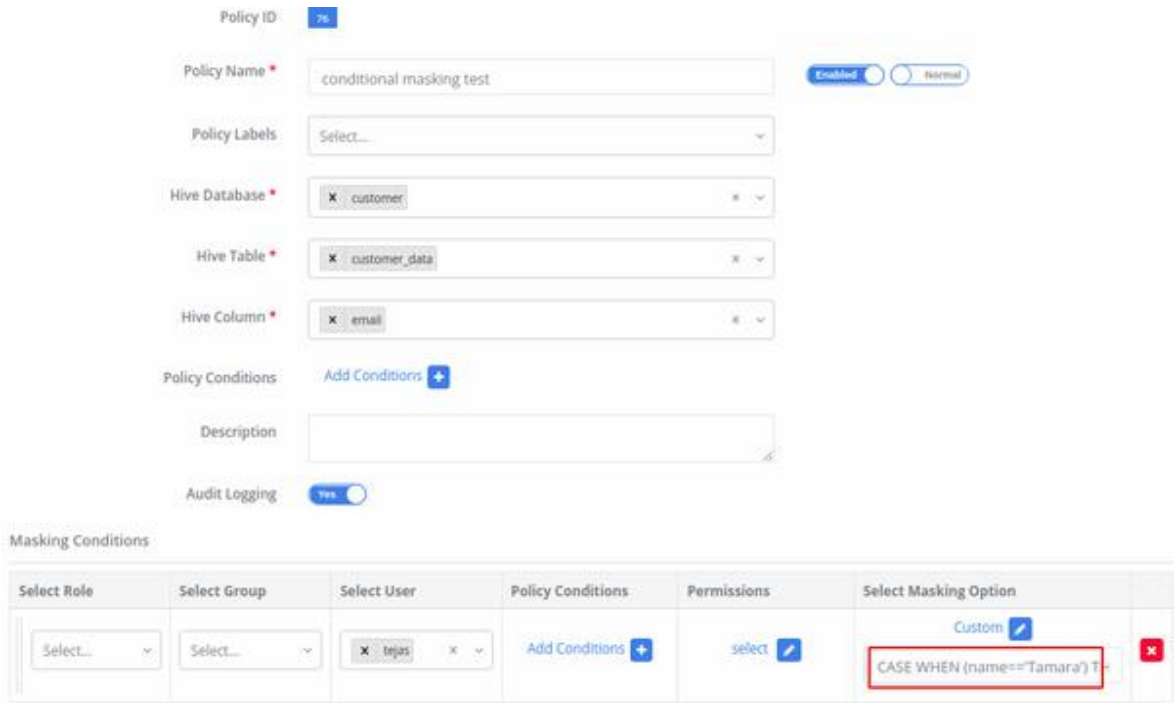
Conditional masking is a masking of a column based on the condition applied on a different column. For example, a condition is applied on column A to mask column B.

Conditional masking is supported for the following systems:

- Hive with EMR
- Hive with Databricks
- Presto SQL with EMR
- Trino

To configure a conditional masking in a policy, do the following:

1. Add Policy. For more details, see Create resource policies: general steps [60].
2. Add the database, table, and column.
3. In the **Select Masking Option** of **Masking Conditions**, select **Custom**. A text appears where you can enter your conditional expression.



Examples

1. Conditional Masking using Single Column
   When the column **name** has *Tamara*, then the column **email** will be masked.

Conditional Expression: `CASE WHEN (name=='Tamara') THEN mask(email) ELSE email END`

customer_data.id	customer_data.name	customer_data.phone	customer_data.email	customer_data.address
NULL	NULL	NULL	NULL	NULL
1	Tamara	898453744	xxxxxxxxx@xxxx.xxxx	16245 Kelly Roads
2	Richard	065511350	vreynolds@gmail.com	0602 Roberts Drives
3	Tanya	634090950	harringtonwilliam@diaz-king.com	4871 Robin Light
4	Richard	829439881	martinvalerie@yahoo.com	512 Gregory Crossroad Apt. 202
5	Raymond	227804351	sarachavez@yahoo.com	97963 Mendez Circle Apt. 857
6	Melissa	553465892	kevinwillis@gmail.com	36654 Rogers Street Apt. 806
7	Deborah	782539839	brittney24@yahoo.com	019 Castro Drive Suite 231
8	Rodney	515337130	jenniferkelly@davis-bond.biz	65083 Joseph Spring Suite 651
9	Katherine	137057143	jperkins@gmail.com	4822 Brittany Ferry Apt. 343
10	David	432941241	wmccann@hotmail.com	406 Jerome Track Apt. 934

2. Conditional Masking using Multiple Columns

   Conditional Expression: `CASE WHEN (name=='Tamara' OR address like '%Robin%') THEN mask(email) ELSE email END`

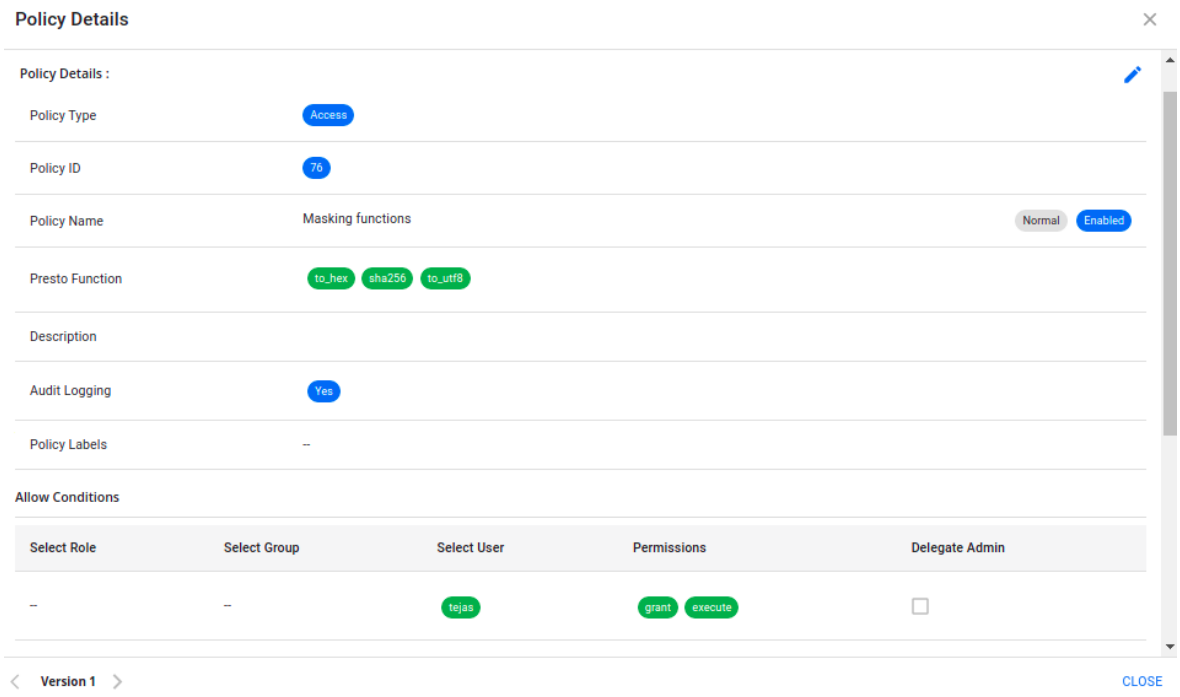
customer_data.id	customer_data.name	customer_data.phone	customer_data.email	customer_data.address
NULL	NULL	NULL	NULL	NULL
1	Tamara	898453744	xxxxxxxxx@xxxx.xxxx	16245 Kelly Roads
2	Richard	065511350	vreynolds@gmail.com	0602 Roberts Drives
3	Tanya	634090950	xxxxxxxxxxxxxxxxx@xxxx-xxxx.xxx	4871 Robin Light
4	Richard	829439881	martinvalerie@yahoo.com	512 Gregory Crossroad Apt. 202
5	Raymond	227804351	sarachavez@yahoo.com	97963 Mendez Circle Apt. 857
6	Melissa	553465892	kevinwillis@gmail.com	36654 Rogers Street Apt. 806
7	Deborah	782539839	brittney24@yahoo.com	019 Castro Drive Suite 231
8	Rodney	515337130	jenniferkelly@davis-bond.biz	65083 Joseph Spring Suite 651
9	Katherine	137057143	jperkins@gmail.com	4822 Brittany Ferry Apt. 343
10	David	432941241	wmccann@hotmail.com	406 Jerome Track Apt. 934
NULL	NULL	NULL	NULL	NULL

3. Conditional Masking in PrestoSQL

   The examples above are applicable for data sources supporting SQL syntax expressions. For PrestoSQL, the syntax changes.

   You need to create an access policy in the **privacera_presto** service which gives access to the following Presto functions for the respective users:

   - to_hex
   - sha256
   - to_utf8



After creating the access policy, you can use the functions in defining the following conditional expression:

Conditional Expression: `if(name='Richard', to_hex(sha256(to_utf8("address"))), "address")`

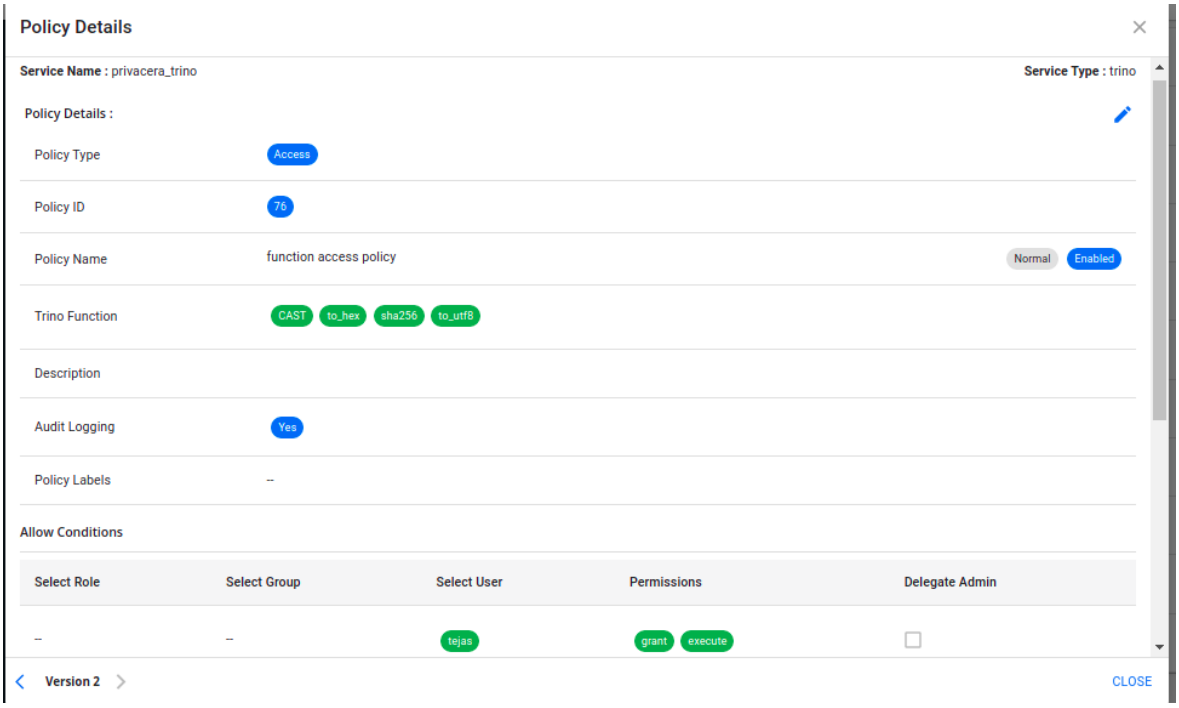
```
presto:customer> select * from customer_data;
  id  |   name    |   phone   |              email               |                                                                  address
------+-----------+-----------+----------------------------------+-----------------------------------------------------------------------------------
 NULL | NULL      | NULL      | NULL                             | NULL
    1 | Tamara    | 898453744 | aphillips@vang.info              | 16245 Kelly Roads
    2 | Richard   | 065511350 | vreynolds@gmail.com              | FE0B0D99B09357082AE254250E5CF85276CEAE33F9F6DF81B2EE6865C7605318
    3 | Tanya     | 634090950 | harringtonwilliam@diaz-king.com  | 4871 Robin Light
    4 | Richard   | 829439881 | martinvalerie@yahoo.com          | 0514EC1A66D3B86B85AB00711E0731D6CA5391280F8F12824689A578F15A3F5C
    5 | Raymond   | 227804351 | sarachavez@yahoo.com             | 97963 Mendez Circle Apt. 857
    6 | Melissa   | 553465892 | kevinwillis@gmail.com            | 36654 Rogers Street Apt. 806
    7 | Deborah   | 782539839 | brittney24@yahoo.com             | 019 Castro Drive Suite 231
    8 | Rodney    | 515337130 | jenniferkelly@davis-bond.biz     | 65083 Joseph Spring Suite 651
    9 | Katherine | 137057143 | jperkins@gmail.com               | 4822 Brittany Ferry Apt. 343
   10 | David     | 432941241 | wmccann@hotmail.com              | 406 Jerome Track Apt. 934
 NULL | NULL      | NULL      | NULL                             | NULL
 NULL | NULL      | NULL      | NULL                             | NULL
    1 | Tamara    | 898453744 | aphillips@vang.info              | 16245 Kelly Roads
    2 | Richard   | 065511350 | vreynolds@gmail.com              | FE0B0D99B09357082AE254250E5CF85276CEAE33F9F6DF81B2EE6865C7605318
    3 | Tanya     | 634090950 | harringtonwilliam@diaz-king.com  | 4871 Robin Light
    4 | Richard   | 829439881 | martinvalerie@yahoo.com          | 0514EC1A66D3B86B85AB00711E0731D6CA5391280F8F12824689A578F15A3F5C
    5 | Raymond   | 227804351 | sarachavez@yahoo.com             | 97963 Mendez Circle Apt. 857
    6 | Melissa   | 553465892 | kevinwillis@gmail.com            | 36654 Rogers Street Apt. 806
    7 | Deborah   | 782539839 | brittney24@yahoo.com             | 019 Castro Drive Suite 231
    8 | Rodney    | 515337130 | jenniferkelly@davis-bond.biz     | 65083 Joseph Spring Suite 651
    9 | Katherine | 137057143 | jperkins@gmail.com               | 4822 Brittany Ferry Apt. 343
   10 | David     | 432941241 | wmccann@hotmail.com              | 406 Jerome Track Apt. 934
 NULL | NULL      | NULL      | NULL                             | NULL
    1 | jp        | jp        | jpa                              | jpad
(25 rows)
```

4. Conditional Masking in Trino

   For conditional masking in Trino, you need to cast/convert the masked column to its appropriate datatype.

   You need to create an access policy in the **privacera_trino** service which gives access to the following Trino functions for the respective users:

   - CAST
   - to_hex
   - sha256
   - to_utf8



After creating the access policy, you can use the functions in defining the following conditional expression:

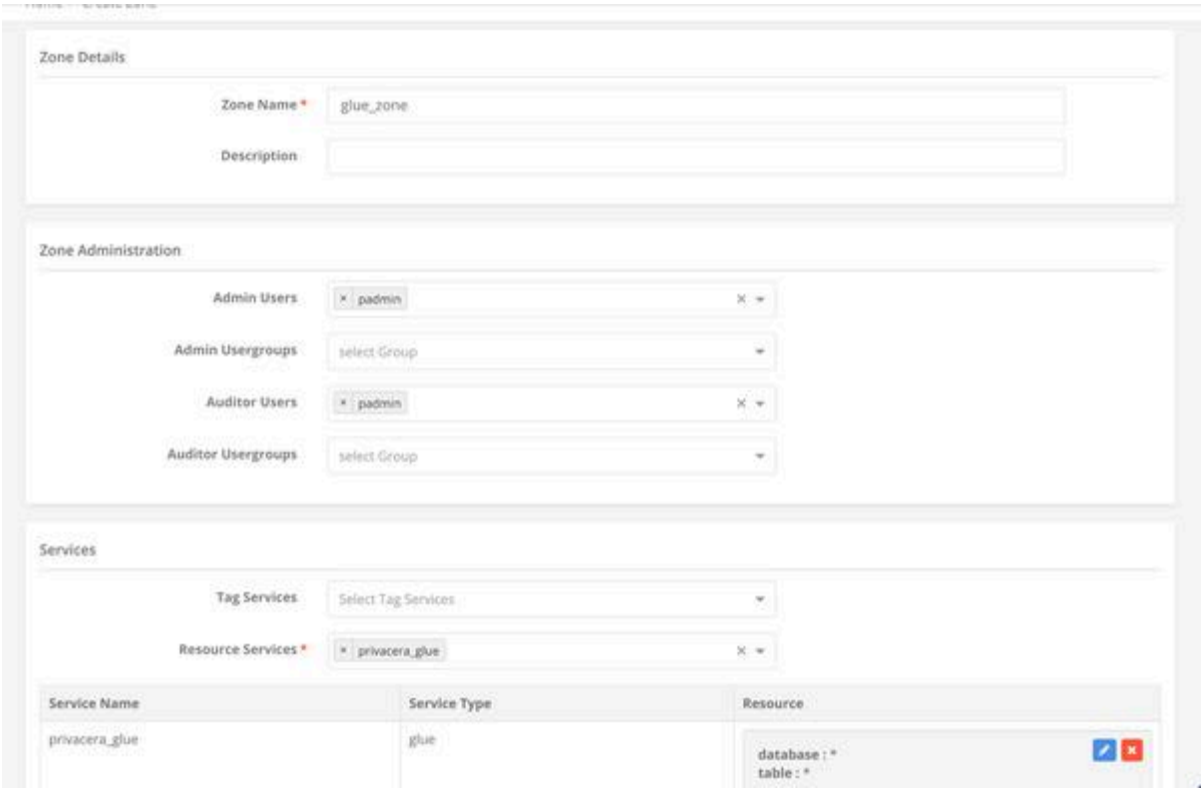
Conditional Expression: `CASE WHEN person_name='Pearlene' THEN (CAST(to_hex(sha256(to_utf8(email_address))) as varchar(100))) ELSE email_address END`

```
trino> select * from customer_catalog.customer_schema.customer_data;
 id | person_name |                          email_address                           |     ssn     | country |   us_phone   |         address
----+-------------+----------------------------------------------------------------------+-------------+---------+--------------+-----------------
  1 | Nancy       | nancy@yahoo.com                                                      | 201-99-5532 | US      | 856-232-9702 | 939 Park Avenue
  2 | Gene        | gene@google.us                                                       | 202-99-5532 | UK      | 954-583-0575 | 303 Johnston Bl
  3 | Edward      | edward@facebook.com                                                  | 203-99-5532 | US      | 209-626-9041 | 130 Hollister
  4 | Pearlene    | 941865CE014AB331DC7AA454D9035046BC2EBA881117EB50840CD5AB078E1C44     | 204-99-5532 | US      | 708-471-6810 | 17 Warren Rd
  5 | James       | james@cuvox.de                                                       | 205-99-5532 | US      | 661-338-6787 | 898 Newport Gra
  6 | Pamela      | pamela@cuvox.de                                                      | 206-99-5532 | UK      | 650-526-5259 | 861 Strick Rd
  7 | Donna       | donna@fleckens.hu                                                    | 207-99-5532 | US      | 303-239-4282 | 1784 S Shore Dr
  8 | Amy         | amy@gustr.com                                                        | 208-99-5532 | US      | 774-553-4736 | 9522 Apple Vall
  9 | Adam        | adam@teleworm.us                                                     | 209-99-5532 | UK      | 651-297-1448 | 745 Old Springv
 10 | Lucille     | lucille@armyspy.com                                                  | 210-99-5532 | US      | 740-320-1270 | 4223  Midway Ro
 11 | Edard       | edu@gustr.com                                                        | 211-99-5532 | UK      | 702-257-8796 | 3659  Dye Stree
 12 | Nick        | nick@jourrapide.com                                                  | 212-99-5532 | US      | 414-483-8638 | 2966  Nutters B
 13 | Brian       | brian@einrot.com                                                     | 213-99-5532 | US      | 479-872-9783 | 3300  Worthingt
 14 | Stella      | stella@jourrapide.com                                                | 214-99-5532 | US      | 818-596-6681 | 1893  Ingram Ro
 15 | Leona       | leona@dayrep.com                                                     | 215-99-5532 | UK      | 256-250-5413 | 4244  Burnside
(15 rows)
```

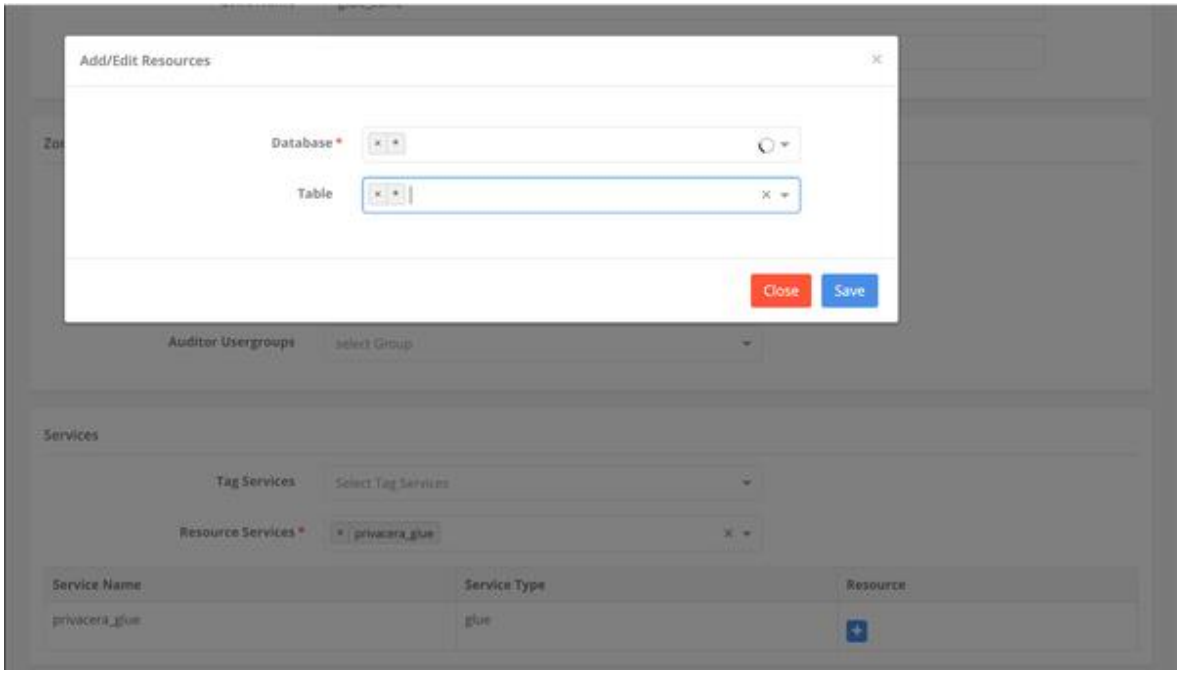
Create access policies for Databricks on Privacera Platform

Create row-level and column-level filters for a Databricks data source.

- Log in to the Privacera Portal.
- On the Privacera Portal home page, from the left menu, expand **Access Management** and click the **Resource Policies**.
- On the **Resource Policies** page, go to **privacera_hive** and click **Row Level Filter** tab.
- Click **Add New Policy**.



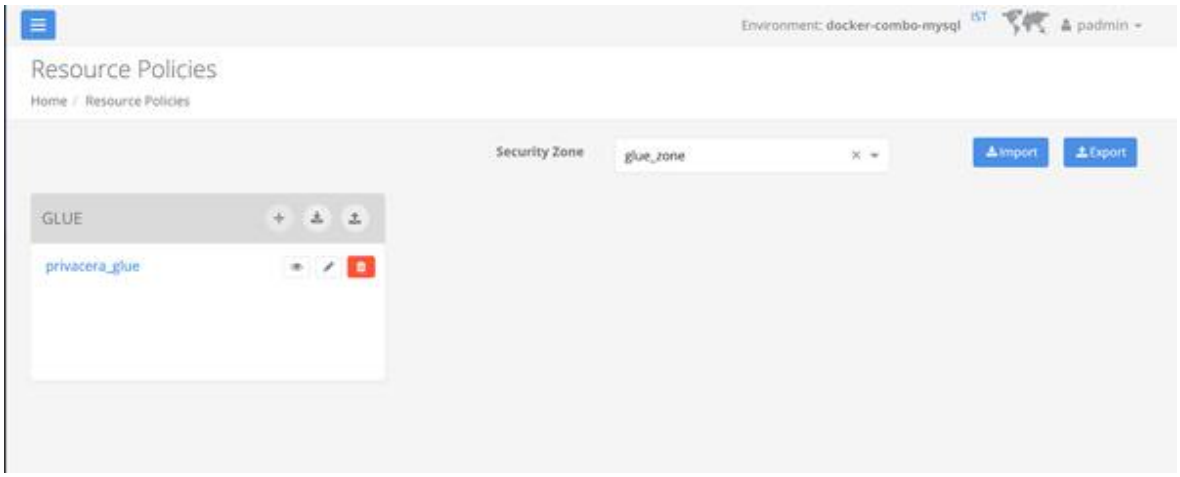
- Enter the Policy Name. E.g. Sales by Country
- Enter the Hive Database name. E.g. sales (This is a filter condition)
- Enter the Hive Table name. E.g. sales_data (This is a filter condition)
- Under Row Level Conditions:
  - Select the User.
  - Set the permission as 'Select'.



- Click **Save**. The Row Level filter for Databricks is added successfully.

Column level access control

- Login to the Privacera Portal.
- On the Privacera Portal home page, from the left menu, expand **Access Management** and click the **Resource Policies**.
- On the **Resource Policies** page, go to **privacera_hive** and click **Add New Policy** to create column level filter for Databricks.
- Create a policy for explicit permission for columns to users.
  - Enter the Policy Name.
  - Select the Database, Table, and Column where you need to apply this filter.



- Click **Save**. The Column Level filter for Databricks policy is added successfully.

Check audit for the above executed command in Privacera Access Manager using the below steps:

- On the Privacera Portal home page, from the left menu, expand **Access Management** and click the **Audit**.

Order of precedence in PolicySync filter

If Filter by Group is set to true, then the value of Filter By Role is ignored. Filter By Role is considered only if Filter by Group is false.

For example, with PostgreSQL:

- When `POSTGRES_MANAGE_USER_FILTERBY_GROUP` is set to `"true"`, Privacera checks if the user belongs to the groups in `POSTGRES_MANAGE_GROUP_LIST`.
- When `POSTGRES_MANAGE_USER_FILTERBY_ROLE` is set to `"true"` and `POSTGRES_MANAGE_USER_FILTERBY_GROUP` is set to `"false"`, Privacera checks if the user is assigned to a role in `POSTGRES_MANAGE_ROLE_LIST`.

This order of precedence applies to the following connectors:

- Databricks SQL
- Google BigQuery
- Microsoft SQL Server
- PostgreSQL
- Redshift
- Snowflake

Example: Manage access to Databricks SQL with Privacera

Now that you have Privacera installed, this document helps you to start working with Privacera to set up access controls for Databricks SQL. Common uses of Privacera to create policies and enforce those policies in Databricks SQL are detailed.

You want to ensure that your data access is consistent and controlled across your computing ecosystem.

Implementing data access policies through Privacera Access Management gives you a single pane of glass to see what data your users can access.

By default in Privacera, your data is not accessible. You can define data access policies that permit only authorized users to access your data. When you define a policy, you apply it to a particular Privacera-connected system that holds your data.

Privacera has several different kinds of policies, such as tag policies and resource policies. Creating resource policies is detailed here, because they are the most basic and easiest to create.

About the data in these examples

These examples rely on TPC DS. The table involved is `CUSTOMER`.

About users, groups, and roles

These examples show a user named Emily (with username **emily**), who is in the **data_analysts** group and has the Privacera role **DATA_ANALYST**.

What has already been set up

These examples assume that the following prerequisites are already set up:

- Your account administrator has connected Databricks SQL.
- Users, groups, and roles have already been created.

Enable access to entire table for a user

Your users need access to the TPC DS `CUSTOMER` table. They are currently prevented from seeing it. By default in Privacera, your data is not accessible.

The Privacera audit log also shows that they do not have access:

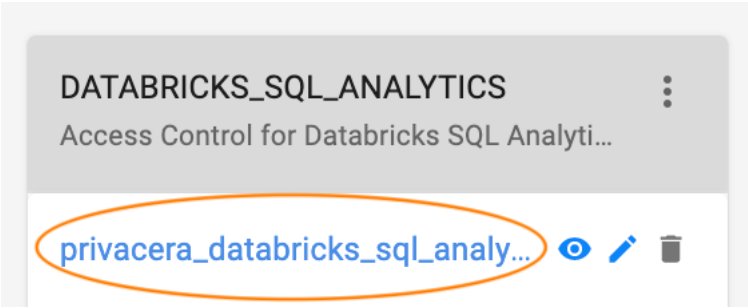
You need to create a resource policy in Privacera that gives Emily all access to the `CUSTOMER` table.

In this example, you should have the following information ready:

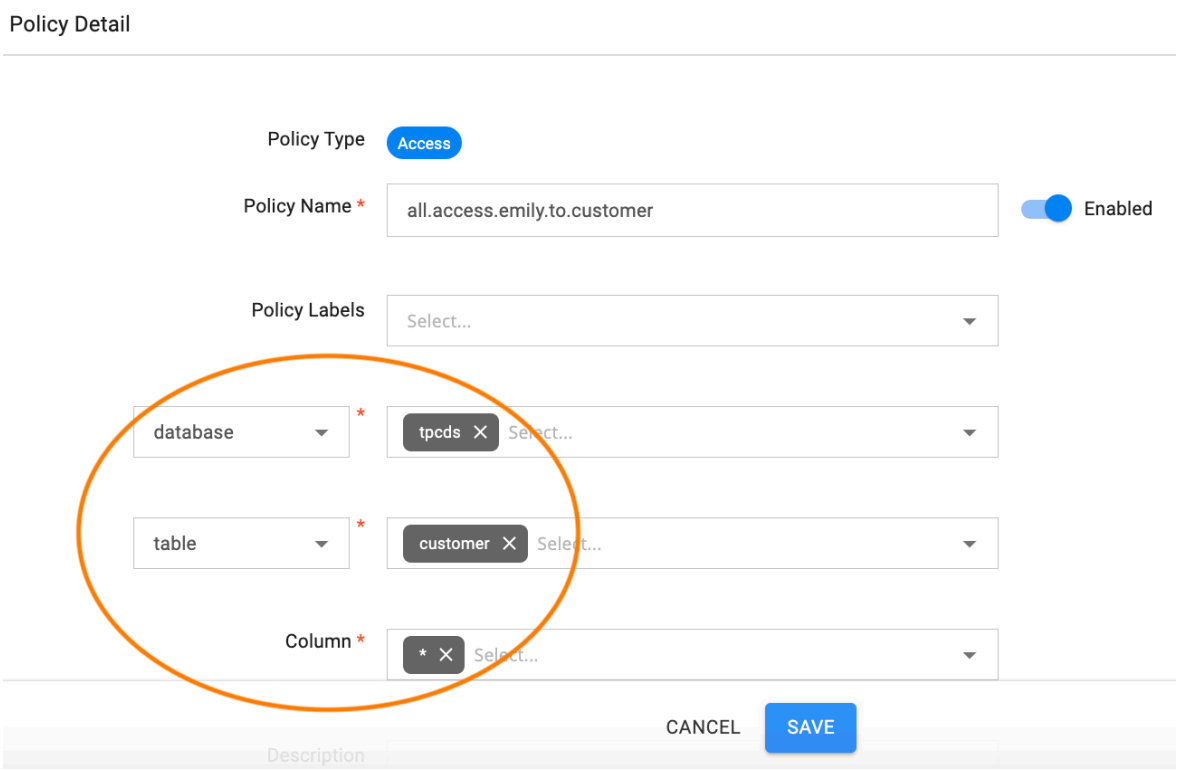
- Name of the user to give access to
- Name of the database
- Name of the table
- Name of the column

Create a policy to give a user access to a table

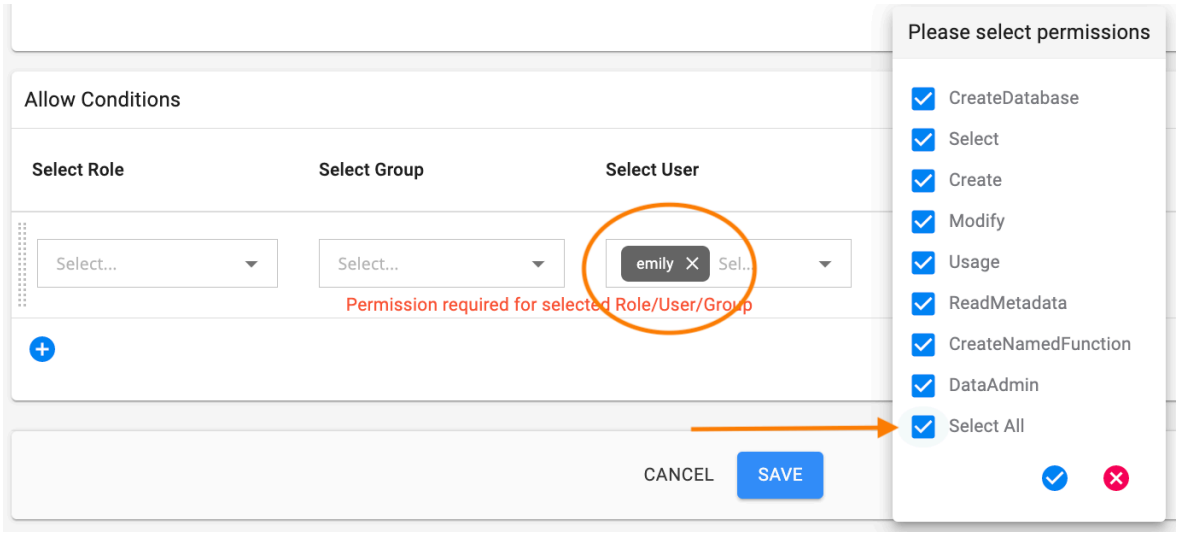
1. In Privacera, expand **Access Management** and click **Resource Policies**.
2. Under **DATABRICKS_SQL_ANALYTICS**, click the **privacera_databricks_sql_analytics** link.

   Lists of policies are displayed on following tabs:
   - **ACCESS**
   - **MASKING**
   - **ROW LEVEL FILTER**
3. On the **ACCESS** tab, click **Add New Policy**.
   The **Policy Detail** page is displayed.
4. Enter a unique policy name.
   Enabled/Disabled: Accept the default, which is **Enabled**.
   Normal/Override: Accept the default, which is **Normal**.
5. Enter a descriptive policy label that helps you find this policy when searching for policies and filtering policy lists.
6. From the **global** pulldown, select **database**.
7. Enter the required database name.
   This example uses **tpcds**.
8. Enter the required table name.
   In this example, we apply policy to the `CUSTOMER` table.
9. Enter the required column names.
   Because this policy is to give access to the entire table, this example uses the * wildcard, which indicates all columns.



10. Ignore **Add Conditions**.
11. Enter a description of the policy to identify it among other policies.
12. **Enable/disable Audit Logging**: Accept the default, which is **Yes**.
13. In the **Allow Conditions** section, under **Select User**, enter the name of the user and add permissions for that user. This example is for user **emily** and all permissions.



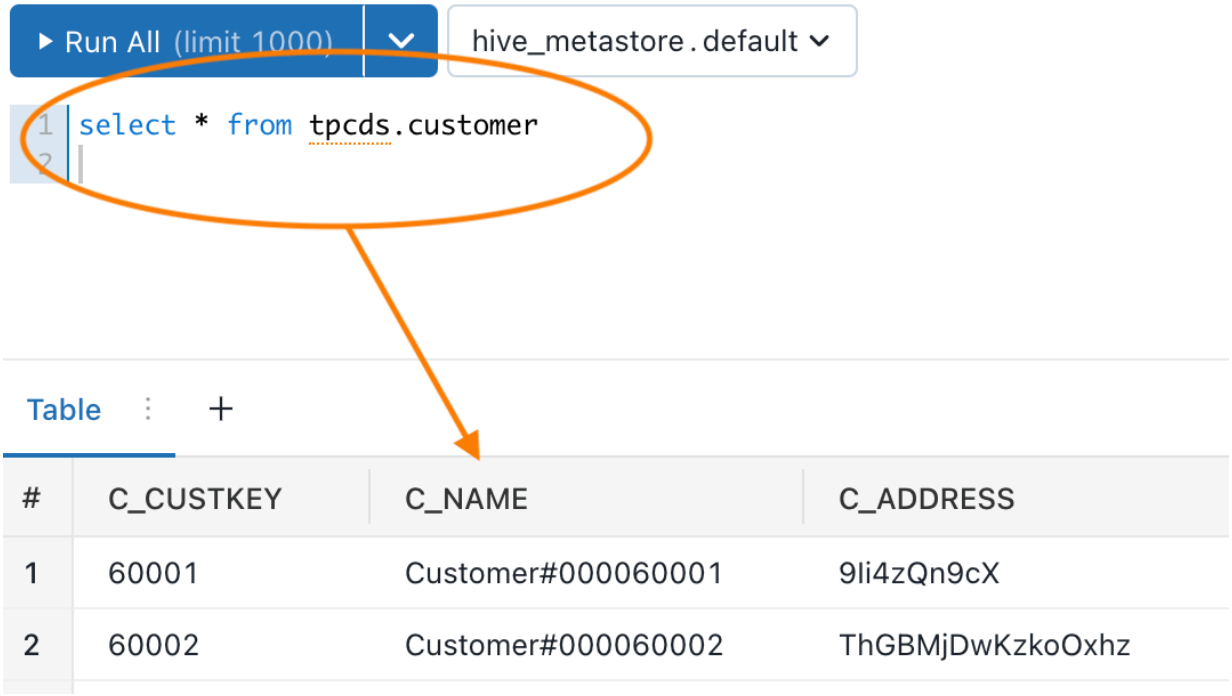
14. Click **Save** to complete the new policy.

Verify the policy in Databricks SQL

To see if the policy has been applied, log in to the database and enter the following SQL command:

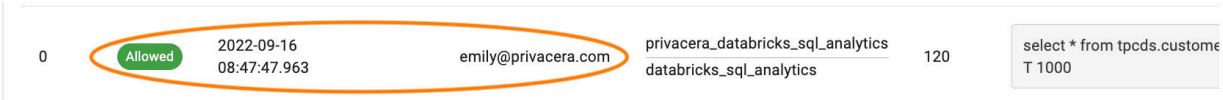
```
select * from tpcds.customer
```

The results show success. The entire table is now visible.



#	C_CUSTKEY	C_NAME	C_ADDRESS
1	60001	Customer#000060001	9li4zQn9cX
2	60002	Customer#000060002	ThGBMjDwKzkoOxhz

The Privacera audit log also shows success:

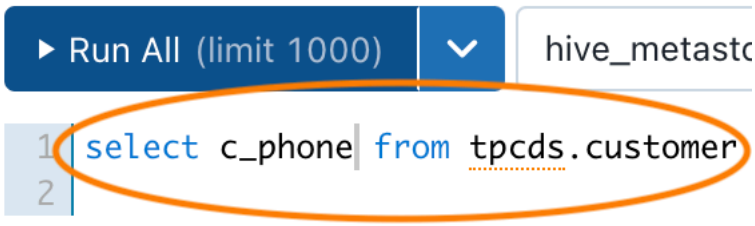


Hide a column from a group of users

The data analysts group needs access to the `CUSTOMER` table, but they should not be able to see customers' phone numbers.

This organization has an internal directive that requires that customer phone numbers and other PII must be protected and shielded from employee view. This internal directive is to comply with regulatory requirements.

The values of the `C_PHONE` column are currently visible and must be hidden:



Table

#	c_phone
1	24-678-784-9652
2	25-782-500-8435
3	26-859-847-7640
4	20-573-674-7999

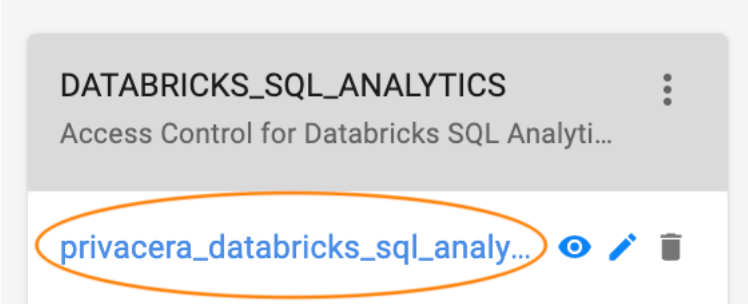
You need to create a policy that masks the values in the `C_PHONE` column from the data analysts group.

In this example, you should have the following information ready:

- Name of the group to give access.
  In this example, the group is **data_analysts**.
- Name of the database.
  In this example, the database name is **tpcds**.
- Name of the table.
  In this example, the table is **CUSTOMER**.
- Name of the column.
  In this example, the column is **C_PHONE**.

Create a policy to mask a column from a group

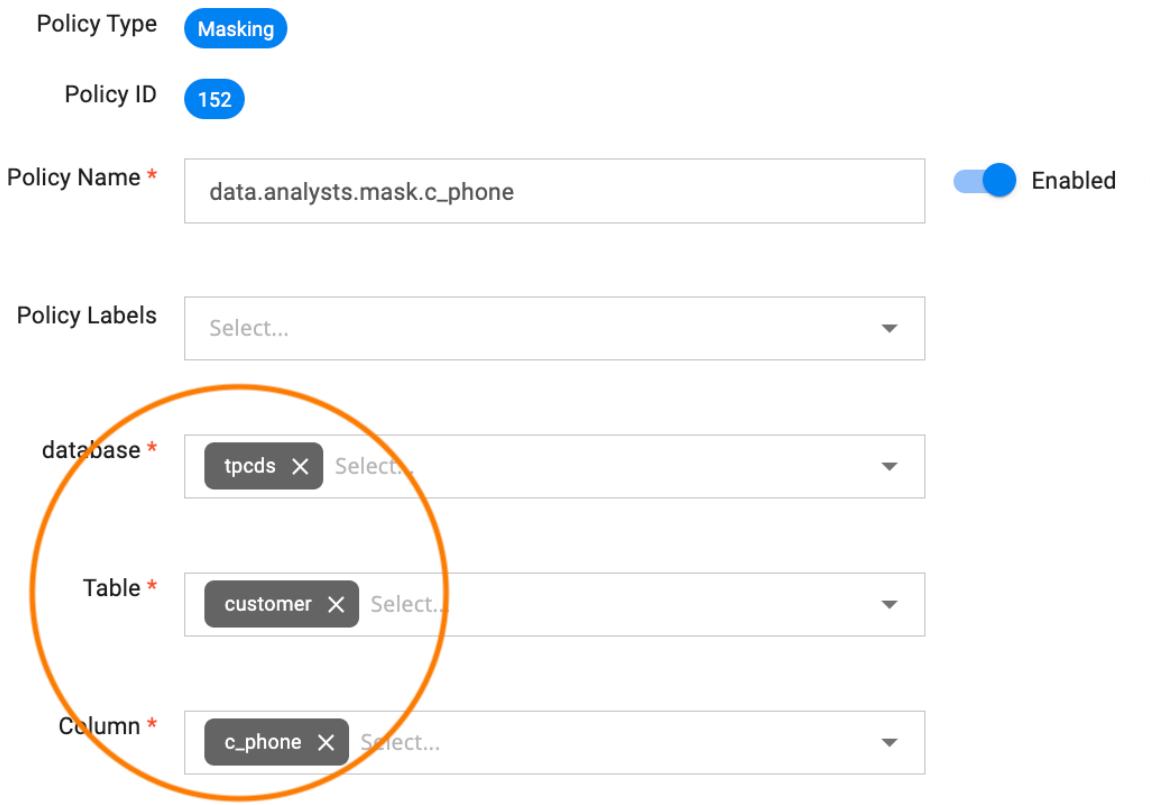
1. In Privacera, expand **Access Management** and click **Resource Policies**.
2. Under **DATABRICKS_SQL_ANALYTICS**, click the **privacera_databricks_sql_analytics** link.



Lists of policies are displayed on the following tabs:

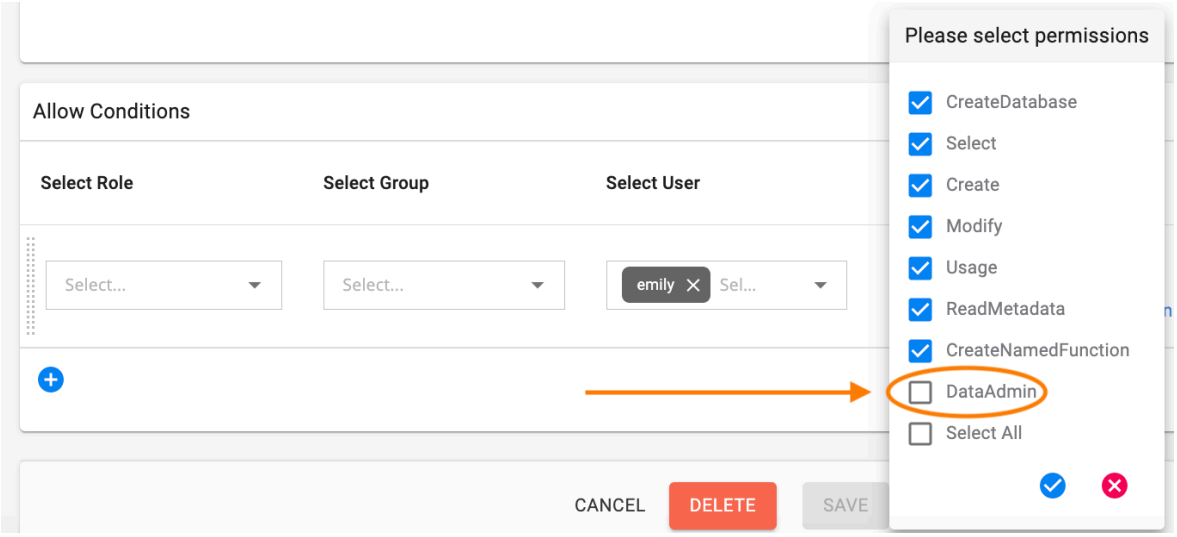
- **ACCESS**
- **MASKING**
- **ROW LEVEL FILTER**

3. On the **MASKING** tab, click **Add New Policy**.
   The **Policy Detail** page is displayed.
4. Enter a unique policy name.
   Enabled/Disabled: Accept the default, which is **Enabled**.
   Normal/Override: Accept the default, which is **Normal**.
5. Enter a descriptive policy label that helps you find this policy when searching for policies and filtering policy lists.
6. Select the required database.
   In this example, the database name is **tpcds**.
7. Select the required table.
   In this example, the table is **CUSTOMER**.
8. Select the required column.
   In this example, the column is **C_PHONE**.



9. Ignore **Add Conditions**.
10. Enter a description of the policy to identify it among other policies.

11. **Enable/disable Audit Logging**: Accept the default, which is **Yes**.
12. In the **Allow Conditions** section, select a group and add permissions for that group.
13. Click **Save** to complete the new policy details.
14. Privacera displays a message that you must have an access policy to accompany the policy you are creating. See Enable access to entire table for a user [197] for how to create an access policy. In that access policy, make sure you remove the **Data Admin** permission for the user. Otherwise, the user can see the original, unprotected database.



Secure database view created by Privacera

Databricks SQL does not have the native capability to create column masks or row filters.

For this reason, Privacera creates a secure view of the original database and applies policy to that secure view. The name of a secure view is:

originalDatabaseName `_secure`

In this example, the name of the secure view is `tpcds_secure`.

In Privacera, the access policy itself must always specify the name of the *original* database, not the secure view.

NOTE

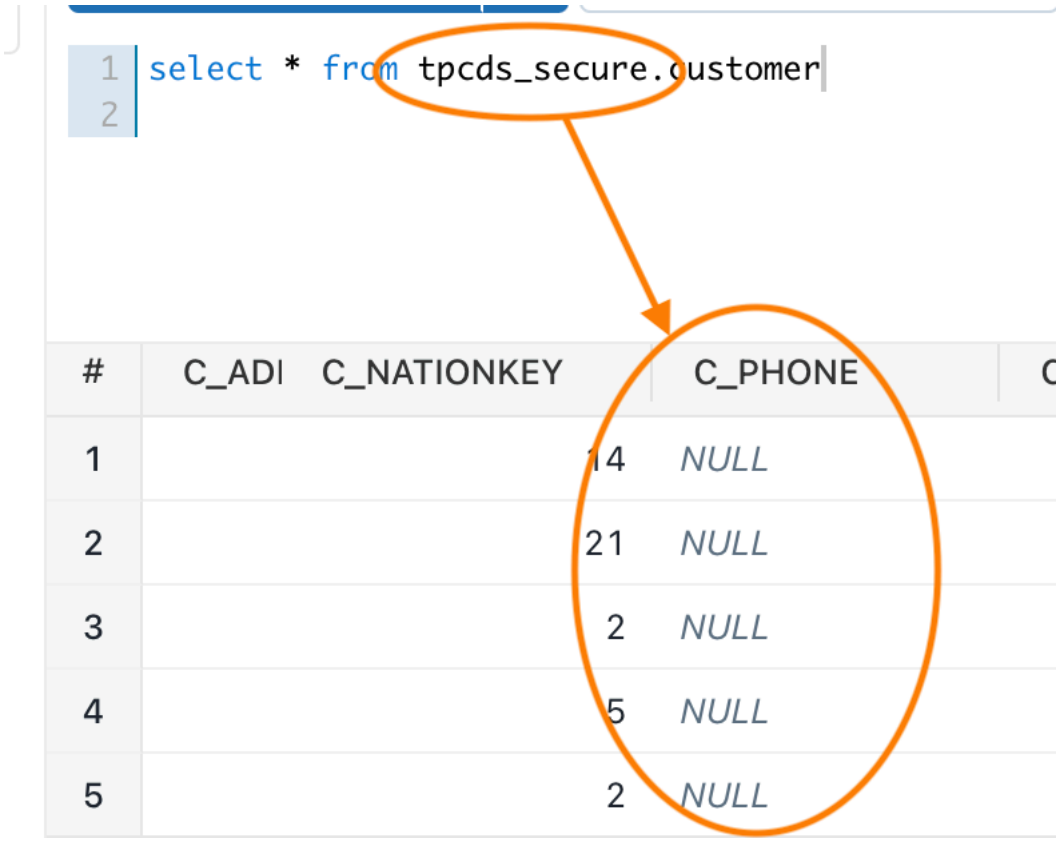
You should tell users the name of the secure view for their queries and that access to the original database is no longer allowed.

Verify the policy in Databricks SQL

To see if the policy has been applied, log in to the database and enter the following SQL command against the secure view `tpcds_secure`:

```
select * from tpcds_secure.CUSTOMER
```

The results show success. The values of the `C_PHONE` column are not visible:



Display only rows with a specific value to a user role

The `CUSTOMER` table includes a column named `C_MKTSEGMENT`. The data analyst group's users need to see only rows that relate to the `HOUSEHOLD` market segment.

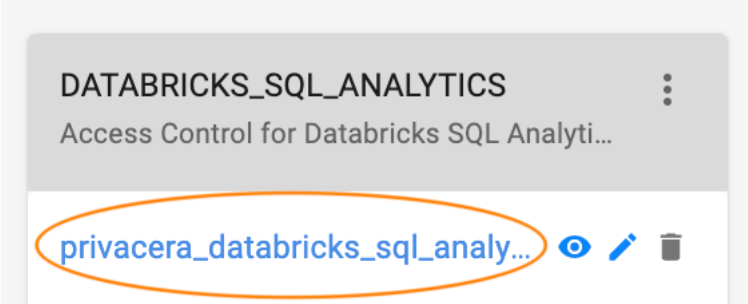
For users with the Privacera role DATA_ANALYST, you need to create a policy to filter out only rows whose `C_MKTSEGMENT` column has the value `HOUSEHOLD`.

In this example, you should have the following information ready:

- Name of the pertinent role.
  In this example, the role is **DATA_ANALYST**. This is a custom role name.
- Name of the database.
  In this example, the database name is **tpcds**.
- Name of the table.
  In this example, the table is **CUSTOMER**.
- Name of the column and its value to filter.
  In this example, the column is **C_MKTSEGMENT** and the value for the row filter is **HOUSEHOLD**.

Create a new policy for a role

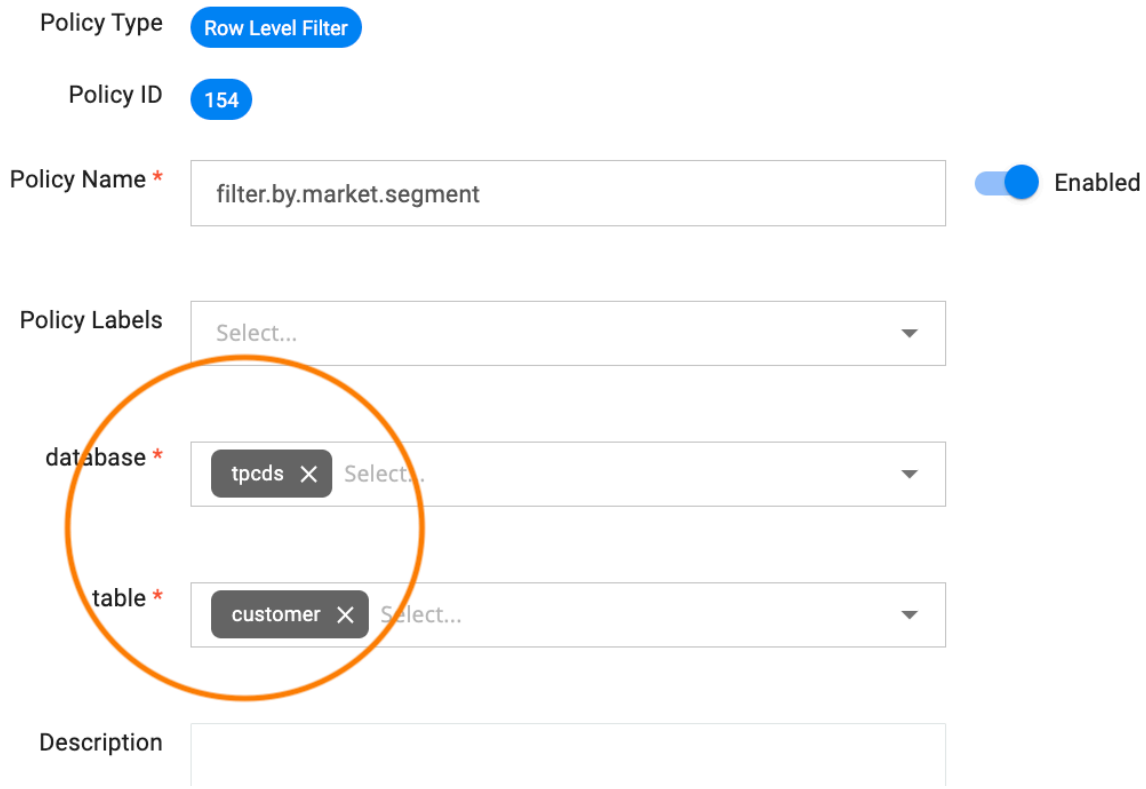
1. In Privacera, expand **Access Management** and click **Resource Policies**.
2. Under **DATABRICKS_SQL_ANALYTICS**, click the **privacera_databricks_sql_analytics** link.



Lists of policies are displayed on the following tabs:

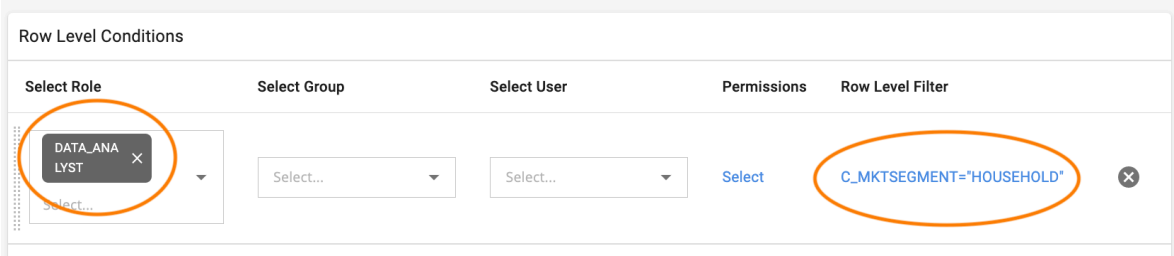
- **ACCESS**
- **MASKING**
- **ROW LEVEL FILTER**

3. On the **ROW LEVEL FILTER** tab, click **Add New Policy**.
   The **Policy Detail** page is displayed.
4. Enter a unique policy name.
   Enabled/Disabled: Accept the default, which is **Enabled**.
   Normal/Override: Accept the default, which is **Normal**.
5. Enter a descriptive policy label that helps you find this policy when searching for policies and filtering policy lists.
6. Select the required database.
   In this example, the database is **tpcds**.
7. Select the required table.
   In this example, the table is **CUSTOMER**..

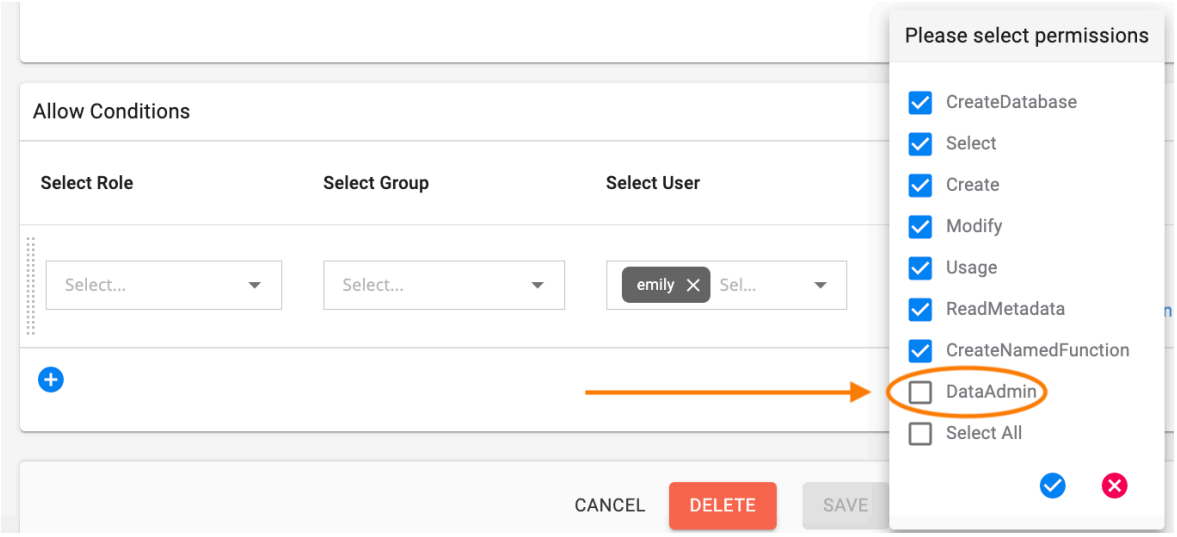


8. Ignore **Add Conditions**.
9. Enter a description of the policy to identify it among other policies.
10. **Enable/disable Audit Logging**: Accept the default, which is **Yes**.
11. In the **Allow Conditions** section, select a role and the row level filter for that role.
    The role in this example is **DATA_ANALYST**.

The filter should be entered as follows. The value is quoted:
columnName =" *value* "
In this example, the filter is: **C_MKTSEGMENT="HOUSEHOLD"**.



12. Click **Save** to complete the new policy details.
13. Privacera displays a message that you must have an access policy to accompany the policy you are creating. See Enable access to entire table for a user [197] for how to create an access policy. In that access policy, make sure you remove the **Data Admin** permission for the user. Otherwise, the user can see the original, unprotected database.



Secure database view created by Privacera

Databricks SQL does not have the native capability to create column masks or row filters.

For this reason, Privacera creates a secure view of the original database and applies policy to that secure view. The name of a secure view is:

originalDatabaseName `_secure`

In this example, the name of the secure view is `tpcds_secure`.

In Privacera, the access policy itself must always specify the name of the *original* database, not the secure view.

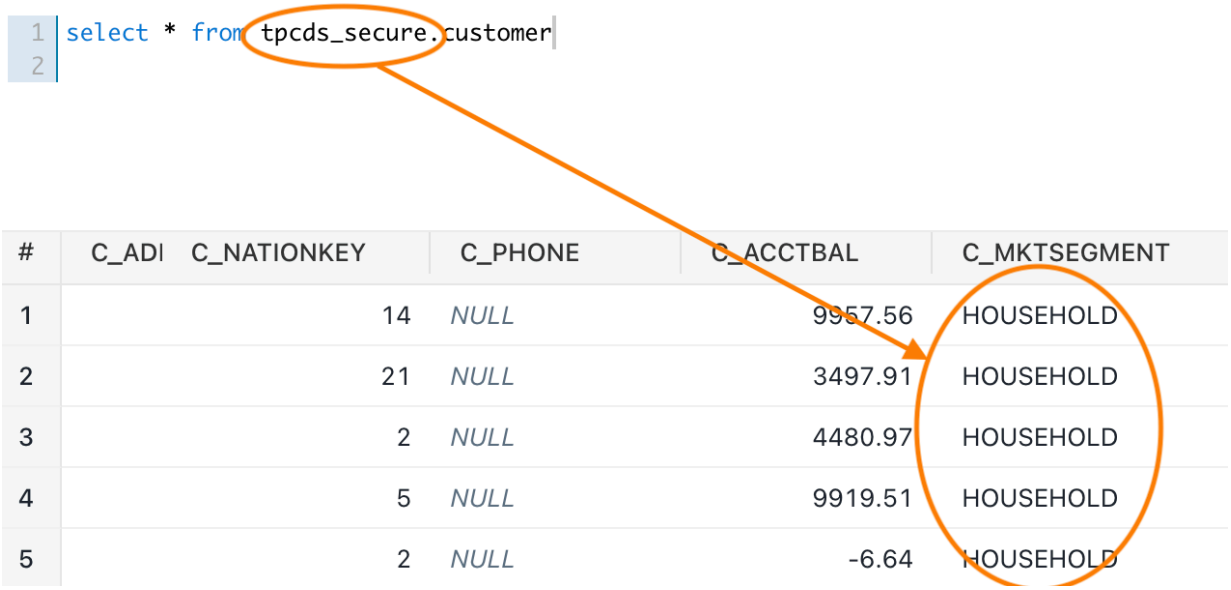
NOTE
You should tell users the name of the secure view for their queries and that access to the original database is no longer allowed.

Verify the policy in Databricks SQL

To see if the policy has been applied, log in to the database and enter the following SQL command against the secure view `tpcds_secure`:

```
select * from tpcds_seecure.CUSTOMER
```

The results show success. Only rows with market segment `HOUSEHOLD` are displayed:



#	C_ADI	C_NATIONKEY	C_PHONE	C_ACCTBAL	C_MKTSEGMENT
1		14	*NULL*	9957.56	HOUSEHOLD
2		21	*NULL*	3497.91	HOUSEHOLD
3		2	*NULL*	4480.97	HOUSEHOLD
4		5	*NULL*	9919.51	HOUSEHOLD
5		2	*NULL*	-6.64	HOUSEHOLD

Service/service group global actions on the Resource Policies page

On the Resource Policies page, you can filter the view and import/export policies.

Add a new resource-based service. Service types have some common attributes as well as attributes specific to that service type.

Export services in JSON-formatted policy sets.

Import a previously exported policy set.

View policy details

Click a service to open to the Policy definition and management page. Each policy definition row shows key attributes:

- **Policy ID:** Each policy is assigned a numeric identifier. These ids are monotonically incremented and unique within each PrivaceraCloud account.Policy identifiers are referenced in the audit trail event messages, so that action taken and recorded to the audit trail is associated with a specific policy.
- **Policy Name:** Policies are assigned a name, either by the system or by a user. System-created policy names can be changed.
- **Validity Period:** A policy can be defined to be effective only for a period of time. Start and End date/times may be defined (to the minute) with a selectable Time Zone. Use the **Add Validity Period** button in the upper right to set a validity period for this policy.
- **Policy Label:** Policies may be assigned a new or existing label. Labels assist in filtering and with search reports.
- **Resource Specifier:** Underneath the Policy Label field are the Resource specifiers. These will be different for each type of resource, and the set of specifiers will change depending on the top down choices. For example, by default a Hive resource will display fields for 'database', 'table', and 'column'.
  The Autocomplete feature is available to add your resources. When you enter the first character in the resource field, the autocomplete feature displays the resources (databases, tables, or columns)

available in the data source. The autocomplete feature supports the **Wildcard character "*"** to add the resources.
Note
Autocomplete feature is supported on the resource fields of the PolicySync connectors only.

- **Condition Sets:** The rules used to allow or deny access to resources. Condition sets are made up of permissions, users, groups, and roles. The permission selection list will be specific to the type of service. (For example, for the ADLS service, the permission set is {read, write, delete, metadata read, metadata write, admin}.) There are four sets of access conditions (rules):
  - **Allow Conditions**
  - **Exclude from Allow Conditions**
  - **Deny Conditions**
  - **Exclude from Deny Conditions**

At least one rule should be defined. Rules for the other condition sets may be omitted.

One or more default 'all...' policies are automatically created for any default created services (those named as "privacera_<service_type>"). (The actual policy names are adjusted for each type of service. For example, for 'hive' services, the 'all' policy is named 'all - database'. For database repository oriented services, the default policy name is: 'all - database, schema, table, column', and so on.).

Tag policies

A **tag** is user-assigned metadata label classifying a resource such as a column, table, or file. Access policies can then be defined by referencing tags values that apply to the tagged data. A single tag can be applied to data resources sourced across multiple data sources or types of data. This allows data managers to create and manage data across scattered resources, based on a user-assigned attribute, reducing the need to define policies for otherwise disjoint resources.

For example, database columns called Email or Phone_Number found in multiple databases can be *tagged* as PII. Policies can then be written for that PII tagged data.

Tags are defined and assigned using the PrivaceraCloud Apache Ranger API. See Apache Ranger API on PrivaceraCloud for general information on using this API interface.

Each PrivaceraCloud account is assigned a unique URL to access and control account resources. This URL is obtained via the API Key on PrivaceraCloud.

Tags can also be defined and assigned using Privacera Discovery. These *tag* definition sets can then be subsequently exported from Privacera Discovery and imported into PrivaceraCloud.

See Apache Ranger Tag APIs for the full set of Ranger Apache REST Tag APIs.

Example: Tag assignment using the Apache Ranger API

In the following example a tag named "PERSON_NAME" is applied to a Hive database named: "sales" with a table named: "sales_data", and a column named : "name".

A PUT is targeted to the endpoint tags/importservicetags of the Apache Ranger API on PrivaceraCloud with access credentials (username: "RangerAPI-Auth"; password: "ranger1234#"). The local file `atlas_tag.json` containst the request the body.

Tag assignment using the Apache Ranger API on Privacera Platform

```
curl -u RangerAPI-Auth:ranger1234# \
-H "Content-type: application/json" \
-d @atlas_tag.json \
-X PUT \
https://api.privaceracloud.com/api/13afxxxxxx6b981fxxxxxx2dc7cdd7xxxxxxa921636xxxxxx2d18
```

where `atlas_tag.json` is:

```
{
 "op": "replace",
 "serviceName": "privacera_hive",
 "tagVersion": 0,
 "tagDefinitions": {
   "0": {
     "name": "PERSON_NAME",
     "source": "Atlas",
     "attributeDefs": [],
     "id": 0,
     "isEnabled": true
   }
 },
 "tags": {
   "0": {
     "type": "PERSON_NAME",
     "owner": 0,
     "attributes": {},
     "id": 0,
     "isEnabled": true
   }
 },
 "serviceResources": [
   {
     "serviceName": "privacera_hive",
     "resourceElements": {
       "database": {
         "values": [
           "sales"
         ],
         "isExcludes": false,
         "isRecursive": false
       },
       "column": {
         "values": [
           "name"
         ],
         "isExcludes": false,
         "isRecursive": false
       },
       "table": {
         "values": [
           "sales_data"
         ],
         "isExcludes": false,
         "isRecursive": false
       }
     },
     "id": 0,
     "isEnabled": true
   }
 ],
 "resourceToTagIds": {
   "0": [
```

```
        0
      ]
    }
  }
```

Tag assignment using the Apache Ranger API on PrivaceraCloud

```
curl -u RangerAPI-Auth:ranger1234# \
-H "Content-type: application/json" \
-d @atlas_tag.json \
-X PUT \
https://api.privaceracloud.com/api/13afxxxxxx6b981fxxxxxx2dc7cdd7xxxxxxa921636xxxxxx2d18
```

where `atlas_tag.json` is:

```
  {
    "op": "replace",
    "serviceName": "privacera_hive",
    "tagVersion": 0,
    "tagDefinitions": {
      "0": {
        "name": "PERSON_NAME",
        "source": "Atlas",
        "attributeDefs": [],
        "id": 0,
        "isEnabled": true
      }
    },
    "tags": {
      "0": {
        "type": "PERSON_NAME",
        "owner": 0,
        "attributes": {},
        "id": 0,
        "isEnabled": true
      }
    },
    "serviceResources": [
      {
        "serviceName": "privacera_hive",
        "resourceElements": {
          "database": {
            "values": [
              "sales"
            ],
            "isExcludes": false,
            "isRecursive": false
          },
          "column": {
            "values": [
              "name"
            ],
            "isExcludes": false,
            "isRecursive": false
          },
          "table": {
```

```
          "values": [
            "sales_data"
          ],
          "isExcludes": false,
          "isRecursive": false
        }
      },
      "id": 0,
      "isEnabled": true
    }
  ],
  "resourceToTagIds": {
    "0": [
      0
    ]
  }
}
```

Add services for tag policies

1. From the homepage, click **Access Management** > **Tag Policies**.
2. Click **Add service** at the top of a service group panel.
3. Type a **Service Name** and **Description**.
4. Set the **Active Status**.
5. Enter the Key and Value in **Add New Configuration** . You can add multiple configurations.
6. To verify the configuration, click **Test Connection**.

Create tag masking policies

Conditions are evaluated sequentially as listed in the policy.

1. From the homepage, click **Access Management** > **Tag Policies**.
2. On the Tag Policies page, click a service in a service group panel.
3. Select the **Masking** tab.
4. Click **Add New Policy**.
5. Configure the masking policy general settings.
   - **Policy Type:** Accept the default value (**Access**).
   - **Policy Name:** Must be unique among all policies.
   - **Normal/Override:** If you select **Override**, this policy takes precedence over other policies.
   - **Add Validity Period:** Select the start and end time of the policy along with the timezone and save.
   - **Policy Labels:** Enter the label for this policy. This helps during search reports and filter policies based on the labels.
   - **Tag:** Enter the applicable tag name.
   - **Policy Conditions:** Click **Add Conditions+** to add policy conditions (This is applied at the policy level).
   - **Audit Logging:** Enable/disable Audit Logging. Toggle to 'No', if this policy doesn't need to be audited. By default, it is selected as 'Yes'.
6. Apply masking conditions.
   a. Under **Masking Conditions**, click **Add** (+).
   b. Select the roles to which this policy applies. To assign a role as an Administrator for the resource, add component permissions and define admin permissions. The administrator can create sub-policies based on the existing policies.
   c. Select the groups to which this policy applies. To assign a group as an Administrator for the resource, add component permissions and define admin permissions. The administrator can

create sub-policies based on the existing policies. The public group contains all users, so setting a condition for the public group applies to all users.

d. Select the users to which this policy applies. To assign a user as an Administrator for the resource, add component permissions, and define admin permissions. The administrator can create sub-policies based on the existing policies.
e. Click **Add Conditions+** and configure the policy conditions.
    i. Set **Accessed after...** to *Yes* or *No* and click **Syntax Check**.
    ii. Enter a boolean expression. This option is applicable to allow or deny conditions on tag-based policies.
f. Click **Add Permissions+** and configure the Component Permissions.
g. Click **Select Masking Option** and select a masking type.
h. Default: Accept the masking scheme applied by the system.
i. Custom: Enter a custom masked value or expression. Custom masking can use any valid Hive UDF (Hive that returns the same data type as the data type in the column being masked).

Add the privacera_tag Service

1. Open **Access Management** > **Tag Policies** > **Add Service**.
2. In the TAG service group, select **privacera_tag** by clicking its name.
3. Save the service.

Create tag access policies

1. From the homepage, click **Access Management** > **Tag Policies**.
2. On the Tag Policies page, click a service in a service group panel.
3. Select the **Access** tab.
4. Click **Add New Policy**.
5. Configure the policy.
    - **Policy Type:** Accept the default value (**Access**).
    - **Policy Name:** Must be unique among all policies.
    - **Normal/Override:** If you select **Override**, this policy takes precedence over other policies.
    - **Policy Labels:** Enter the label for this policy. This helps during search reports and filter policies based on the labels.
    - **Tag:** Enter the applicable tag name.
    - **Policy Conditions:** Click **Add Conditions+** to add policy conditions (This is applied at the policy level).
    - **Audit Logging:** When enabled, an event is entered in the audit log when this policy is applied.

Policy configuration settings

Policies contain access rules associated with a particular data source or a subset of it. Specific policy attributes differ depending on the policy type, but all policies contain the following attributes:

- **Policy Type:** The basis for controlling access. For example, a policy can be based on the resource, on a tag, or on a scheme.
- **Policy Name:** Policies are assigned a name, either by the system or when created by a portal user. Default, system-created policies can be renamed. The policy name should be unique and can not be duplicated across the system.
- **Normal/Override:** This option allows you to select policy type whether it is a 'Normal' or 'Override' policy. If you select 'Override', access permissions in the policy override the access permissions in existing policies.
- **Enable/Disable:** By default, the policy is enabled. If the policy is not required, you can disable it by switching to 'Disabled' mode.

- **Policy Id:** Each policy is assigned a numeric identifier. These IDs are incremented and unique within each account. Policy identifiers are referenced in the audit trail event messages, so that action taken and recorded to the audit trail is associated with a specific policy.
- **Policy Label:** A descriptive label that helps users find this policy when searching for policies and filtering policy lists.
- **Resource Specifier:** These will be different for each type of resource, and the set of specifiers will change depending on the top down choices.
  The *autocomplete feature* is available only if you have defined PolicySync connectors for the following services:
  - Postgres
  - Redshift
  - MSSQL
  - Snowflake
  - Databricks SQL
- **Validity Period**: A policy can be defined to be effective only for a period of time. Start and End date/times (defined to the minute), with a selectable Time zone.
- **Description:** This field required description of policy which can be used to identify among others policies.
- **Audit Logging:** Enable/disable Audit Logging. Toggle to 'No', if this policy doesn't need to be audited. By default, it is selected as 'Yes'.
- **Condition Sets:** The rules that allow or deny access to a resource. Available permissions are specific to the type of service. There are four access conditions:
  - **Allow Conditions**
  - **Exclude from Allow Conditions**
  - **Deny Conditions**
  - **Exclude from Deny Conditions**

At least one rule must be defined. One or more default 'all...' policies are automatically created for any default created services (those named as "privacera_<service_type>"). Policy names reflect the type of service.

Security zones

Security zone administrators can create and update policies for resources in their security zone.

A security zone includes the following:

- A collection of resource services.
- In a resource service, one or more data repository objects (database, bucket, or file) and zero or more tag services.

Create security zones

To create a security zone, follow these steps:

1. Go to **Access Management > Security Zones**.
2. In the upper right, click the **+** icon.
3. Complete the following required and optional fields.
4. After all desired fields are complete, click **Save**.

Zone Details:

- Zone Name (required):
- Description

Zone Administration (all fields required):

- Admin Users
- Admin Usergroups
- Auditor Users
- Auditor Usergroups

Services:

- Tag Services
- Resource Services (required)
  - Select a data repository service
  - To add or edit resources under the **Resource** column, use the **+** icon. Select and add resources appropriate to the data repository type.

Edit or view security zones

To view or edit a security zone:

1. Go to **Access Management > Security Zones**.
2. From the displayed list, select the desired security zone.
3. To change the security zone definition, click **Edit**.
4. Modify and save the revised configuration.

Delete security zones

To delete a security zone:

1. Go to **Access Management > Security Zones**.
2. From the displayed list, select the desired security zone.
3. Click **Delete**.
4. Click **Yes** to delete or **Cancel** to discard the change.

Security zone administration on Privacera Platform

Security Zones can only be created, updated, or deleted by a user with the ROLE_SYS_ADMIN role in Access Management.

Users can view, retrieve, and update policies only in security zones in which they have administrator privileges.

Security zones use in authorization on Privacera Platform

When a plugin authorizes an access request, it determines the Security Zone in which the accessed data source resides. If the data source matches a Security Zone, only the policies of that Security Zone are used to authorize the access. If the data source does not match a Security Zone, the policies in the default (unnamed) Security Zone are used to authorize the access.

Manage Databricks policies on Privacera Platform

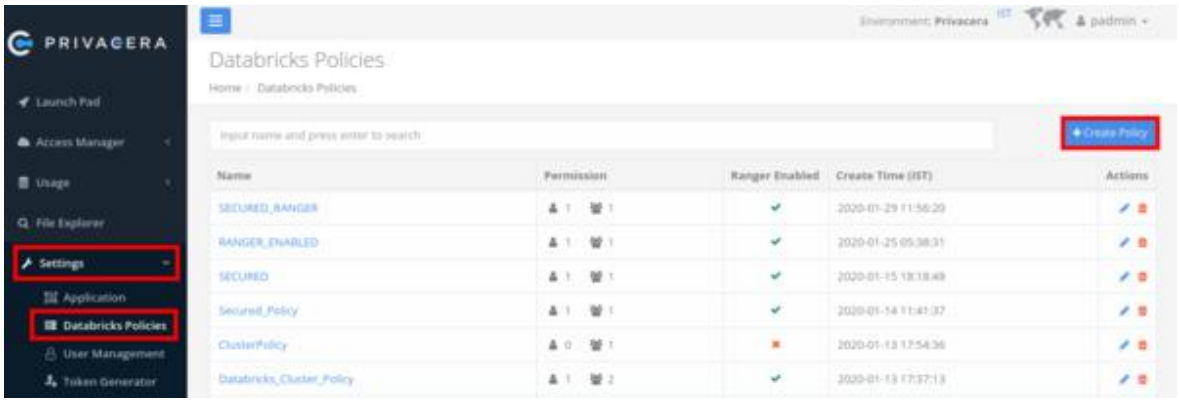
- Databricks AWS Integration can now be done directly using Privacera Manager. See topic Enable AWS CLI on Privacera Platform.
- Databricks Azure Integration can now be done directly using Privacera Manager. See topic Enable Azure CLI on Privacera Platform.

Create Policy in Portal

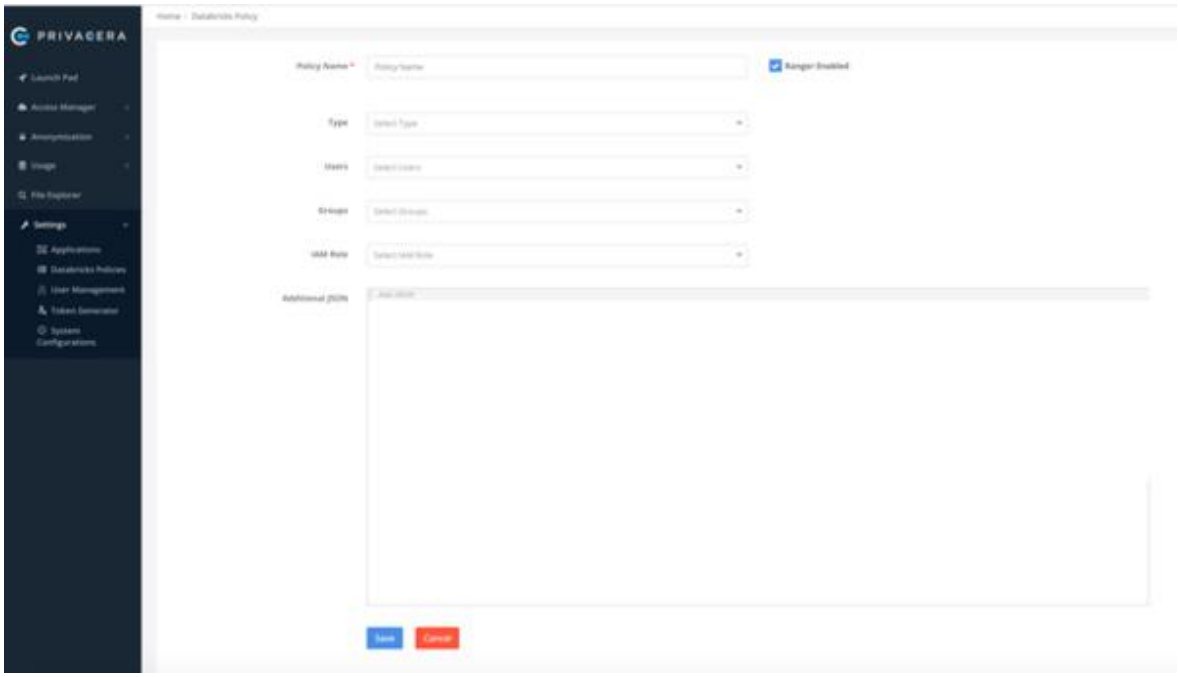
To create a policy in Privacera Portal, use the following steps:

1. Login to Privacera Portal.
2. On the Privacera home page, expand the **Settings** menu and click on **Databricks Policies** from left menu.

3. Click the **+Create Policy**.



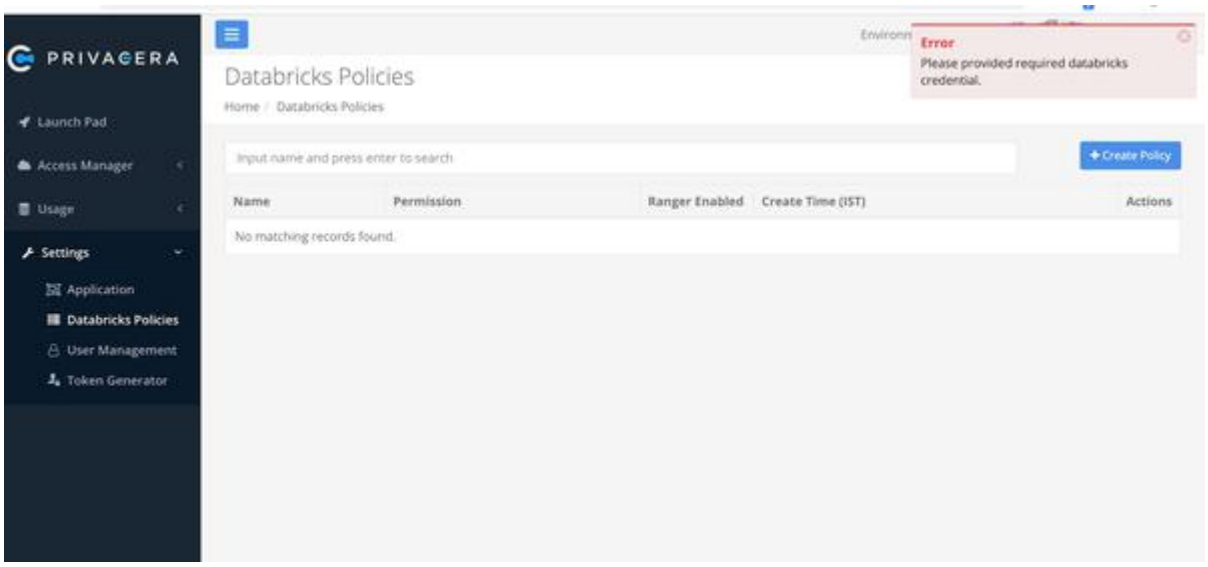
4. Enter the **Policy Name**. (Mandatory)
5. Select the **Users**, **Groups**, **IAM Role** from the drop-down.
   You can select multiple **Users** and **Groups**.
6. Enter the **Additional JSON** (If any). This will append with the existing JSON which will be fetched from backend.



7. Click **Save**.
   The policy is created successfully.

Possible permission error

By default, Admin groups will have permission to all the policies. If you have not configured Databricks properties in Privacera Portal properties file then you will get the following error.



To correct this error:

- The Token should be generated from a user **who is an Admin**.
- Additional JSON that can be used to create policy.

```
{
"autoscale.min_workers": {
        "type": "range",
        "minValue": 1,
        "hidden": false
    },
    "autoscale.max_workers": {
        "type": "range",
        "maxValue": 2
    },
    "cluster_name": {
        "type": "fixed",
        "value": "secured"
    },
    "spark_version": {
        "type": "regex",
        "pattern": "5.5.x-scala2.11"
    },
    "spark_conf.spark.hadoop.hadoop.security.credential.provider.path": {
        "type": "fixed",
        "value": "jceks://dbfs@/${JCEKS_FILE_PATH}",
        "hidden": true
    },
    "spark_conf.spark.databricks.delta.formatCheck.enabled": {
        "type": "fixed",
        "value": "false",
        "hidden": true
    },
    "spark_conf.spark.databricks.delta.preview.enabled": {
        "type": "fixed",
        "value": "true",
        "hidden": true
```

```
    },
    "node_type_id": {
        "type": "regex",
        "pattern": "m4.*"
    },
    "autotermination_minutes": {
        "type": "unlimited",
        "defaultValue": 50
    }
}
```

Create Cluster in Databricks

To create a Cluster in Databricks through policy, use the following steps:

1. Login to Databricks.
2. Click on **Clusters** from left menu.
3. Click on **Create Cluster**.
4. Select the **Policy** from the drop down.
5. Enter the the required details.
6. Click on **Create Cluster**.

The Cluster is created successfully.

Supported actions

Policy

- Create: Setting users and group permissions
- Update:Setting users and group permissions
- Delete:

Form elements

- Ranger Enabled
  - True: Compulsory JSON will be added from backend
  - False:Compulsory JSON will not be added from the backend.

IAM role (Optional)

If selected then the below JSON value will be added from backend.

```
{
   "aws_attributes.instance_profile_arn":{
      "type":"fixed",
      "value":{SELECTED_VALUE},
      "hidden":false
   }
}
```

Use a custom policy repository with Databricks

You can use a custom policy repository with Databricks if you use the fine-grained access control (FGAC) plug-in for access management. A custom policy repository uses a unique prefix, such as `dev`, which you specify as part of your Databricks cluster configuration.

The prefix string is injected into the Apache Ranger access control plug-in through an environment variable named `SERVICE_NAME_PREFIX`.

When configured, the FGAC plug-in can use access policies in custom policy repositories for the following service types:

- Hive
- S3
- Files
- ADLS

Prerequisites

To use Databricks with a custom policy repository, you must first create custom policy repository.

1. Login to Privacera Portal.
2. On the Privacera Portal home page, expand **Access Management**, and then click the **Resource Policies**.
3. Click the three dots menu on the service for which you want to create a custom policy repository.
4. Click **Add Service**.
5. Add the values in the given fields, and then click Save.
   Service Name: name of your service.
   For example, `<prefix>_<service_name>`. Where service name will be hive, s3, files, or so on.
   Select Tag Service: select `privacera_tag` to apply tag based policies for this custom repository.
   Username: It should be service user i.e., `hive`
   Password: xxxxx
   jdbc.driverClassName: dummy or *org.apache.hive.jdbc.HiveDriver*
   Jdb.url: dummy

Complete one of the following procedures to use Databricks with custom policy repositories:

Configure a custom policy repository for all Databricks clusters with cluster policy

You can configure your Databricks cluster policy to inject the `SERVICE_NAME_PREFIX` environment variable for all clusters through cluster policy.

1. Log in to your Databricks account.
2. Define a cluster policy and specify the following JSON configuration:

```
{
  "spark_env_vars.SERVICE_NAME_PREFIX": {
    "type": "fixed",
    "value": "<SERVICE_NAME_PREFIX>"
  }
}
```

   Where:
   - `<SERVICE_NAME_PREFIX>`: Specifies the policy repository prefix, such as `qa`.
3. To apply the new cluster policy, restart each Databricks cluster.

Configure a custom policy repository for a single Databricks cluster with an environment variable

You can configure a specific Databricks cluster to inject the `SERVICE_NAME_PREFIX` environment variable through a cluster environment variable.

1. Log in to your Databricks account.
2. From your list of Databricks clusters, edit the cluster you want to use with a custom policy repository.

3. Update the cluster environment variables to include the following value:

   ```
   SERVICE_NAME_PREFIX=<SERVICE_NAME_PREFIX>
   ```

   Where:
   - <SERVICE_NAME_PREFIX>: Specifies the policy repository prefix, such as qa.
4. To apply the new cluster policy, restart the Databricks cluster.

Configure a custom policy repository for a single Databricks cluster with an Init Script

You can edit the FGAC plug-in Init Script that your Databricks cluster runs such that it sets the SERVICE_NAME_PREFIX environment variable.

1. Log in to the system where you installed Privacera Manager.
2. Locate the privacera_databricks.sh script and open the script with an editor.
3. Modify the script you opened in the previous step and specify the following variable:

   ```
   SERVICE_NAME_PREFIX=<SERVICE_NAME_PREFIX>
   ```

   Where:
   - <SERVICE_NAME_PREFIX>: Specifies the policy repository prefix, such as qa.
4. To update the modified script to your Databricks DBFS, enter the following command:

   ```
   dbfs cp privacera_databricks.sh dbfs:/<PATH>/privacera_databricks.sh
   ```

   Where:
   - <PATH>: Specifies the DBFS path to copy the updated script to.
5. To apply the new cluster policy, restart the Databricks cluster.

Configure policy with Attribute-Based Access Control on Privacera Platform

Privacera enables use of user, group and tag attributes in authorization policies. Attribute-Based Access Control (ABAC) makes it possible to express authorization policies without prior knowledge of specific resources or specific users, which helps avoid the need for new policies as new resources or users are introduced.

For more information, see How polices are evaluated [56].

Overview

The following scenarios can be accomplished with ABAC configuration:

ABAC in row filter expressions

By using Attribute-Based Access Control (ABAC) for row filters, user can define attribute based conditions in row filter expressions. For example, user/group attributes can be referenced in a row filter expression:

```
dept = ${{USER.dept}}
state in ( ${{GET_UG_ATTR_Q_CSV('state')}} )
```

ABAC in resource definitions

Attributes can also be included in resource names:

```
path: /home/${{USER._name}}
path: /departments/${{USER.dept}}
database: dept_${{USER.dept}}p
```

ABAC in policy conditions in Resource based access policies

With the ABAC feature, you can configure resource policies based on user attributes from your LDAP or AD service.

You can assign attributes to users, groups and tags in policies. You can also implement logical conditions on the user attributes for the resource policies.

Attributes can be referenced using expressions:

```
USER.employeeType != 'intern'

TAG.piiType == 'email'

TAG.sensitivityLevel <= USER.allowedSensitivityLevel
```

NOTE

ABAC for policy conditions in resource-based access policies is supported for the following data sources:

- Databricks/EMR Hive, Spark, and all services using `privacera_hive` service definitions.
- PolicySync Snowflake

Prerequisites

Ensure the following prerequisites are met:

- Import the users from the LDAP or AD directory to the Privacera Ranger database.
  If you have not imported LDAP users yet, see LDAP UserSync integration on Privacera Platform [53] for information.
- Determine the resources you want to protect with ABAC-based policies.

Setup User/Group Attributes

The users and groups with their attributes can be synced with Usersync from LDAP or AD. With users, the user attributes are also synced from source (such as LDAP or AD).

NOTE

If any attribute is added or updated to the group public, these attributes cannot be used in ABAC expressions.

Add/Edit Attributes

For more information, see Users, groups, and roles [13]

Example policy with ABAC

1. Let's say we have a user with the following attribute: `dept` : `IT`
2. We create a resource access policy such as:

Policy Details

Policy Name	access	Normal Enabled
Hive Database	hive_test	Include
Hive Table	employee	Include
Hive Column	*	Include
Description		
Audit Log	Yes	
Policy Labels	--	

Allow Conditions

Select Role	Select Group	Select User	Policy Conditions	Permissions	Delegate Admin
		tejas, sally		select	(unchecked)

3. We created a **Row Filter Policy** as below:

Policy Name	employee_rlf	Normal Enabled
Hive Database	hive_test	
Hive Table	employee	
Description		
Audit Log	Yes	
Policy Labels	--	

Row Level Conditions

Select Role	Select Group	Select User	Policy Conditions	Access Types	Row Level Filter
	public			select	dept == '${{USER.dept}}'

Note that the row filter expression used here is: `dept == '${{USER.dept}}'`

4. Now the setup is done on EMR Hive. The table `hive_test.employee` is available on the EMR Hive cluster, with data as below:

```
| employee.id  |  employee.name  |  employee.location |  employee.dept  |
+--------------+-----------------+--------------------+-----------------+
| 1            |  manny          |  mumbai            | IT
| 2            |  novak          |  NY                | HR
| 3            |  Pablo          |  Barcelona         | HR
| 4            |  Carlos         |  Mexico            | Support
+--------------+-----------------+--------------------+-----------------+
4 rows selected
```

5. The user `tejas` is available on the EMR cluster to run the query. When select query is executed by `tejas`:

```
| employee.id  |  employee.name  |  employee.location |  employee.dept  |
+--------------+-----------------+--------------------+-----------------+
| 1            |  manny          |  mumbai            | IT
+--------------+-----------------+--------------------+-----------------+
1 row selected
```

Create Databricks policies on Privacera Platform

To create a Databricks policy in Privacera Portal, follow these steps:

1. Login to Privacera Portal.

2. On the Privacera home page, expand the **Settings** menu and click on **Databricks Policies** from left menu.
3. Click the **+Create Policy**.
   a. Enter the **Policy Name**. (Mandatory)
   b. Select the **Type**, **Users**, **Groups**, **IAM Role** from the respective drop-down.

      NOTE
      You are allowed to select multiple **Users** and **Groups**.

   c. Enter the **Additional JSON** (If any). This will append with the existing JSON which will be fetched from back-end.
4. Click **Save**.
   The policy is created successfully.

IMPORTANT
By default, Admin groups will have permission to all the policies. If you haven't configured Databricks properties in Privacera Portal properties file then you will get the below error.

- The Token should be generated from a user **who is an Admin**.
- Additional JSON that can be used to create policy.

```
{
    "autoscale.min_workers": {
            "type": "range",
            "minValue": 1,
            "hidden": false
        },
        "autoscale.max_workers": {
            "type": "range",
            "maxValue": 2
        },
        "cluster_name": {
            "type": "fixed",
            "value": "secured"
        },
        "spark_version": {
            "type": "regex",
            "pattern": "5.5.x-scala2.11"
        },
        "spark_conf.spark.hadoop.hadoop.security.credential.provider.path": {
            "type": "fixed",
            "value": "jceks://dbfs@/${JCEKS_FILE_PATH}",
            "hidden": true
        },
        "spark_conf.spark.databricks.delta.formatCheck.enabled": {
            "type": "fixed",
            "value": "false",
            "hidden": true
```

```
    },
    "spark_conf.spark.databricks.delta.preview.enabled": {
        "type": "fixed",
        "value": "true",
        "hidden": true
    },
    "node_type_id": {
        "type": "regex",
        "pattern": "m4.*"
    },
    "autotermination_minutes": {
        "type": "unlimited",
        "defaultValue": 50
    }
}
```

To know more about Databricks Policy, refer to Manage Databricks policies on Privacera Platform [214].

Example: Create basic policies for table access

Now that you have Privacera installed, this document helps you to start working with Privacera to set up access controls for Databricks SQL. Common uses of Privacera to create policies and enforce those policies in Databricks SQL are detailed.

You want to ensure that your data access is consistent and controlled across your computing ecosystem.

Implementing data access policies through Privacera Access Management gives you a single pane of glass to see what data your users can access.

By default in Privacera, your data is not accessible. You can define data access policies that permit only authorized users to access your data. When you define a policy, you apply it to a particular Privacera-connected system that holds your data.

Privacera has several different kinds of policies, such as tag policies and resource policies. Creating resource policies is detailed here, because they are the most basic and easiest to create.

About the data in these examples

These examples rely on TPC DS. The table involved is `CUSTOMER`.

About users, groups, and roles

These examples show a user named Emily (with username **emily**), who is in the **data_analysts** group and has the Privacera role **DATA_ANALYST**.

What has already been set up

These examples assume that the following prerequisites are already set up:

- Your account administrator has connected Databricks SQL.
- Users, groups, and roles have already been created.

Enable access to entire table for a user

Your users need access to the TPC DS `CUSTOMER` table. They are currently prevented from seeing it. By default in Privacera, your data is not accessible.

The Privacera audit log also shows that they do not have access:

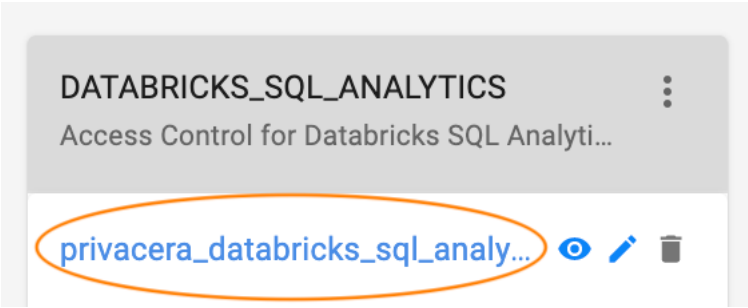
You need to create a resource policy in Privacera that gives Emily all access to the `CUSTOMER` table.

In this example, you should have the following information ready:

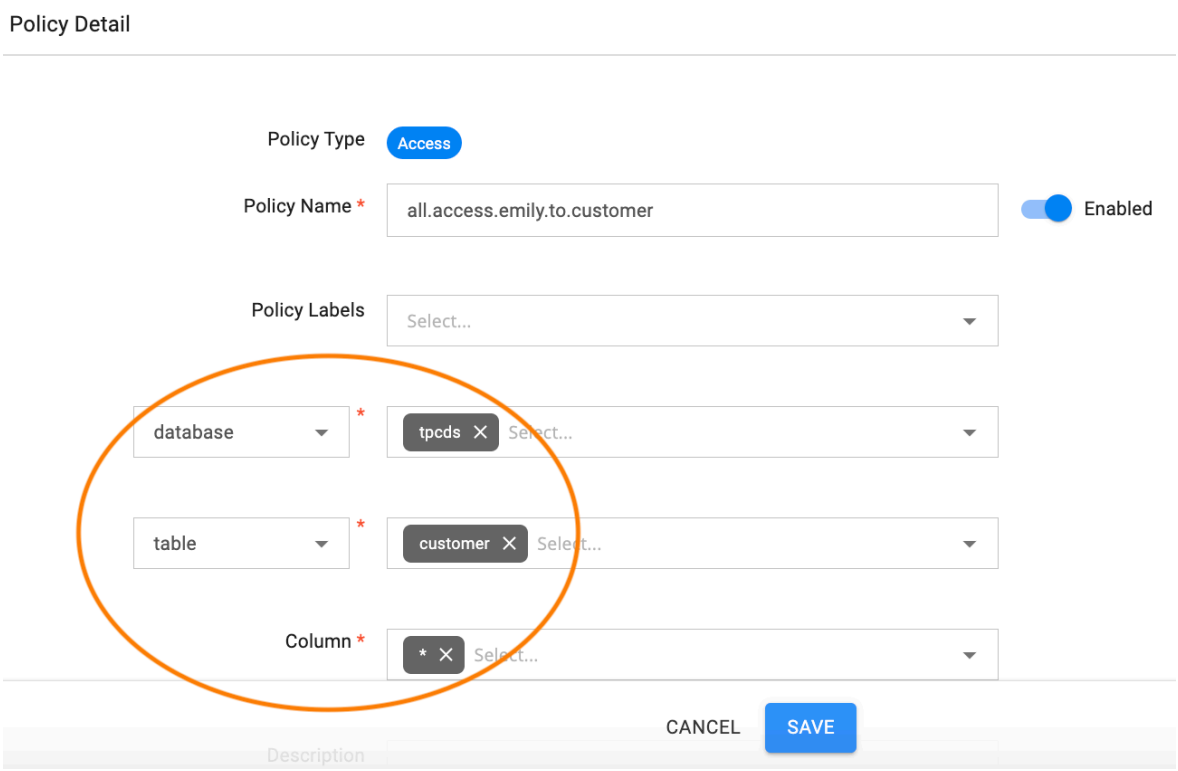
- Name of the user to give access to
- Name of the database
- Name of the table
- Name of the column

Create a policy to give a user access to a table

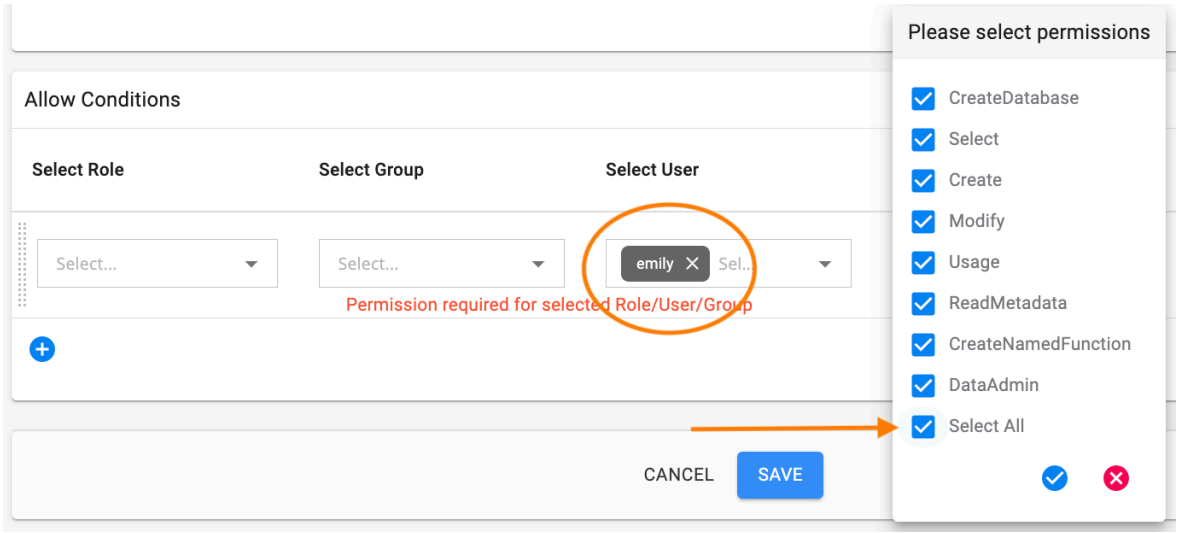
1. In Privacera, expand **Access Management** and click **Resource Policies**.
2. Under **DATABRICKS_SQL_ANALYTICS**, click the **privacera_databricks_sql_analytics** link.

   Lists of policies are displayed on following tabs:
   - **ACCESS**
   - **MASKING**
   - **ROW LEVEL FILTER**
3. On the **ACCESS** tab, click **Add New Policy**.
   The **Policy Detail** page is displayed.
4. Enter a unique policy name.
   Enabled/Disabled: Accept the default, which is **Enabled**.
   Normal/Override: Accept the default, which is **Normal**.
5. Enter a descriptive policy label that helps you find this policy when searching for policies and filtering policy lists.
6. From the **global** pulldown, select **database**.
7. Enter the required database name.
   This example uses **tpcds**.
8. Enter the required table name.
   In this example, we apply policy to the `CUSTOMER` table.
9. Enter the required column names.
   Because this policy is to give access to the entire table, this example uses the * wildcard, which indicates all columns.



10. Ignore **Add Conditions**.
11. Enter a description of the policy to identify it among other policies.
12. **Enable/disable Audit Logging**: Accept the default, which is **Yes**.
13. In the **Allow Conditions** section, under **Select User**, enter the name of the user and add permissions for that user. This example is for user **emily** and all permissions.



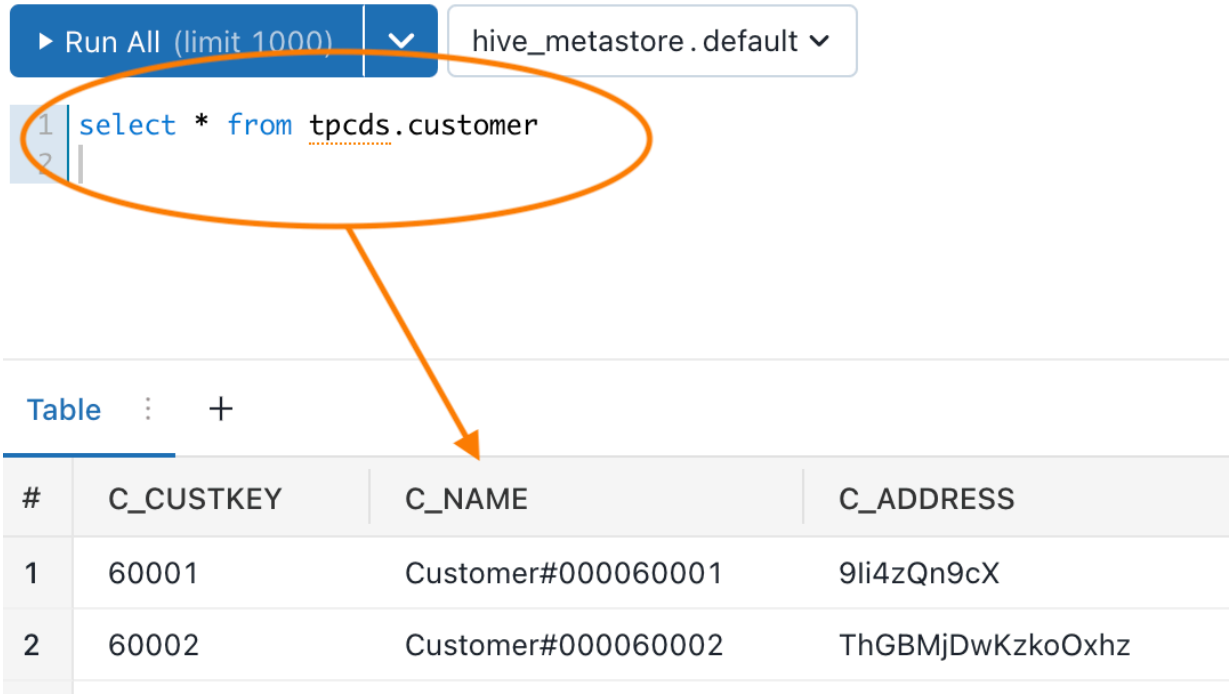
14. Click **Save** to complete the new policy.

Verify the policy in Databricks SQL

To see if the policy has been applied, log in to the database and enter the following SQL command:

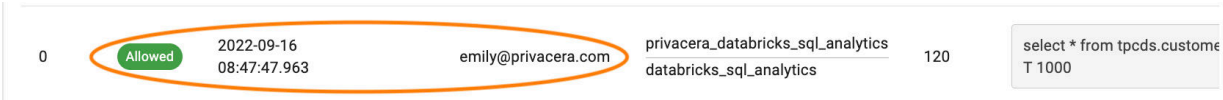
```
select * from tpcds.customer
```

The results show success. The entire table is now visible.



#	C_CUSTKEY	C_NAME	C_ADDRESS
1	60001	Customer#000060001	9li4zQn9cX
2	60002	Customer#000060002	ThGBMjDwKzkoOxhz

The Privacera audit log also shows success:

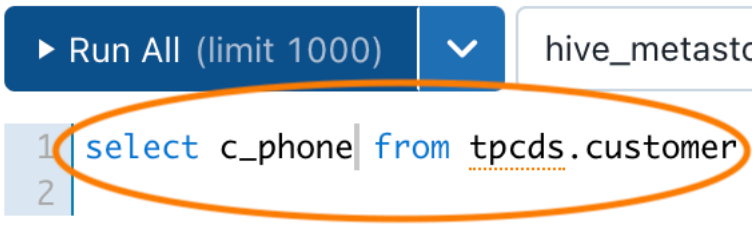


Hide a column from a group of users

The data analysts group needs access to the `CUSTOMER` table, but they should not be able to see customers' phone numbers.

This organization has an internal directive that requires that customer phone numbers and other PII must be protected and shielded from employee view. This internal directive is to comply with regulatory requirements.

The values of the `C_PHONE` column are currently visible and must be hidden:



Table

#	c_phone
1	24-678-784-9652
2	25-782-500-8435
3	26-859-847-7640
4	20-573-674-7999

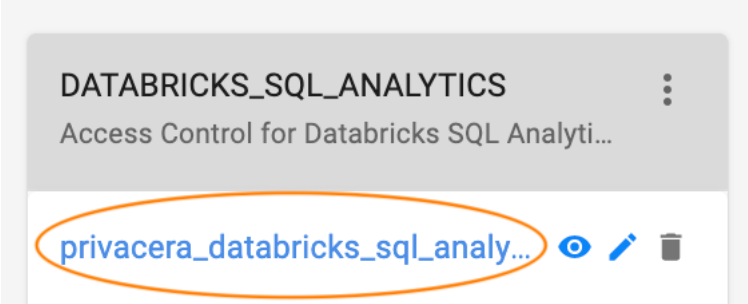
You need to create a policy that masks the values in the `C_PHONE` column from the data analysts group.

In this example, you should have the following information ready:

- Name of the group to give access.
  In this example, the group is **data_analysts**.
- Name of the database.
  In this example, the database name is **tpcds**.
- Name of the table.
  In this example, the table is **CUSTOMER**.
- Name of the column.
  In this example, the column is **C_PHONE**.

Create a policy to mask a column from a group

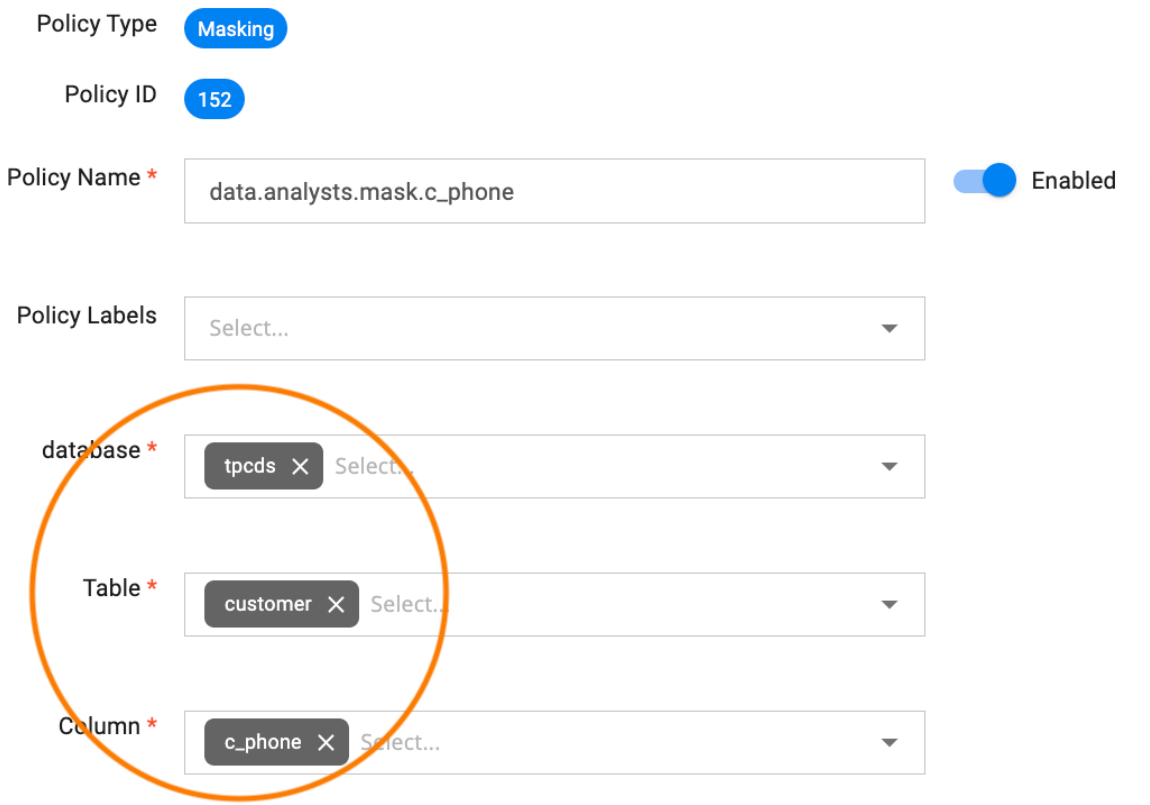
1. In Privacera, expand **Access Management** and click **Resource Policies**.
2. Under **DATABRICKS_SQL_ANALYTICS**, click the **privacera_databricks_sql_analytics** link.



Lists of policies are displayed on the following tabs:

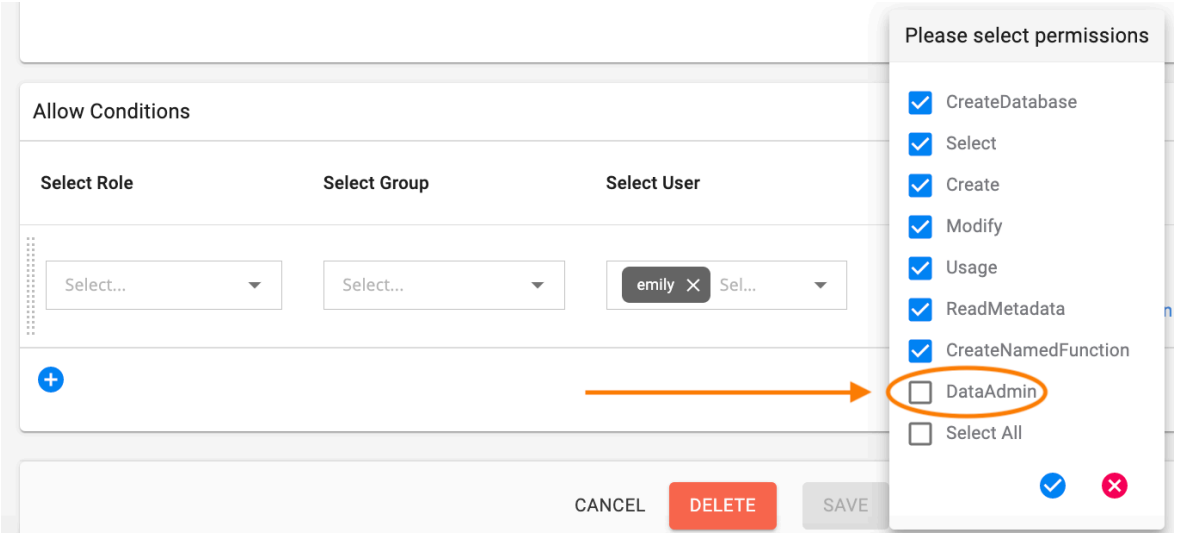
- **ACCESS**
- **MASKING**
- **ROW LEVEL FILTER**

3. On the **MASKING** tab, click **Add New Policy**.
   The **Policy Detail** page is displayed.
4. Enter a unique policy name.
   Enabled/Disabled: Accept the default, which is **Enabled**.
   Normal/Override: Accept the default, which is **Normal**.
5. Enter a descriptive policy label that helps you find this policy when searching for policies and filtering policy lists.
6. Select the required database.
   In this example, the database name is **tpcds**.
7. Select the required table.
   In this example, the table is **CUSTOMER**.
8. Select the required column.
   In this example, the column is **C_PHONE**.



9. Ignore **Add Conditions**.
10. Enter a description of the policy to identify it among other policies.

11. **Enable/disable Audit Logging**: Accept the default, which is **Yes**.
12. In the **Allow Conditions** section, select a group and add permissions for that group.
13. Click **Save** to complete the new policy details.
14. Privacera displays a message that you must have an access policy to accompany the policy you are creating. See Enable access to entire table for a user [223] for how to create an access policy.
    In an access policy, make sure you remove the **Data Admin** permission for the user. Otherwise, the user can see the original, unprotected database.



Secure database view created by Privacera

Databricks SQL does not have the native capability to create column masks or row filters.

For this reason, Privacera creates a secure view of the original database and applies policy to that secure view. The name of a secure view is:

*originalDatabaseName*_secure

In this example, the name of the secure view is `tpcds_secure`.

In Privacera, the access policy itself must always specify the name of the *original* database, not the secure view.

NOTE

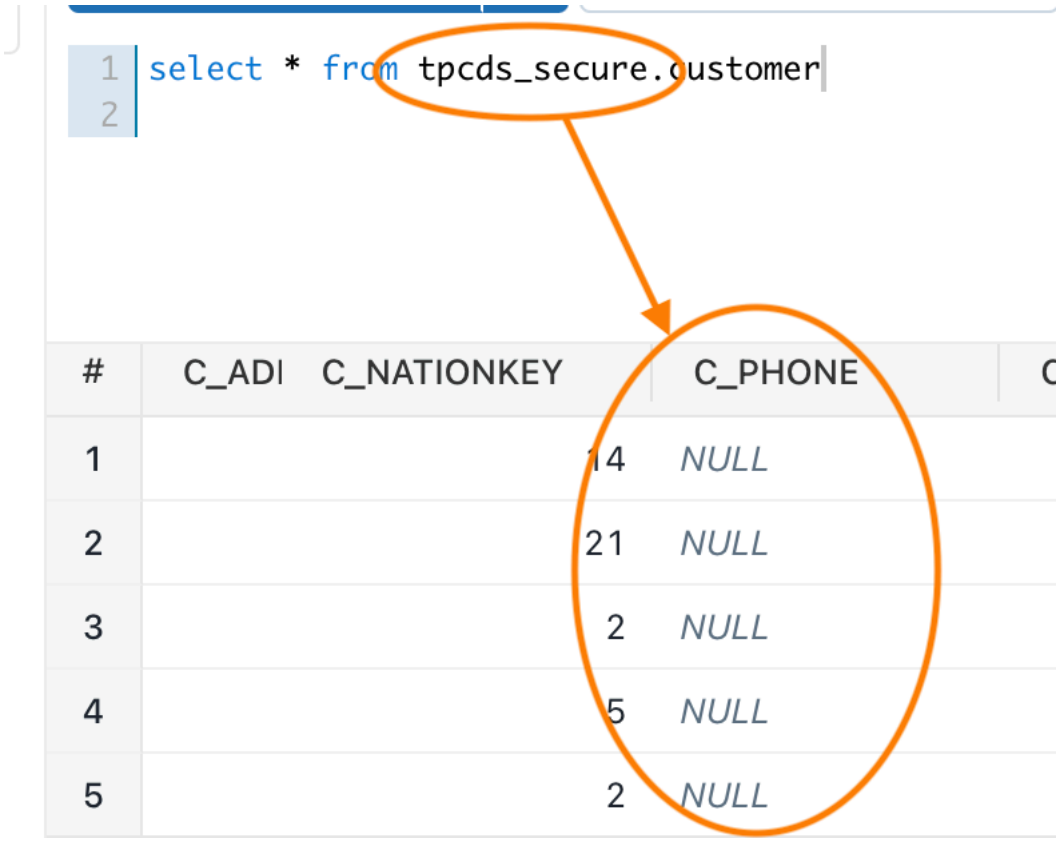
You should tell users the name of the secure view for their queries and that access to the original database is no longer allowed.

Verify the policy in Databricks SQL

To see if the policy has been applied, log in to the database and enter the following SQL command against the secure view `tpcds_secure`:

```
select * from tpcds_secure.CUSTOMER
```

The results show success. The values of the `C_PHONE` column are not visible:



Display only rows with a specific value to a user role

The `CUSTOMER` table includes a column named `C_MKTSEGMENT`. The data analyst group's users need to see only rows that relate to the `HOUSEHOLD` market segment.

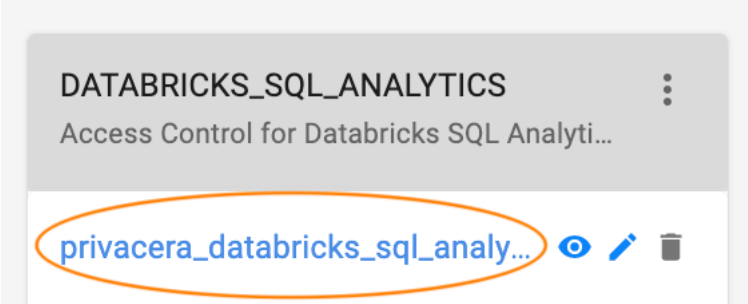
For users with the Privacera role DATA_ANALYST, you need to create a policy to filter out only rows whose `C_MKTSEGMENT` column has the value `HOUSEHOLD`.

In this example, you should have the following information ready:

- Name of the pertinent role.
  In this example, the role is **DATA_ANALYST**. This is a custom role name.
- Name of the database.
  In this example, the database name is **tpcds**.
- Name of the table.
  In this example, the table is **CUSTOMER**.
- Name of the column and its value to filter.
  In this example, the column is **C_MKTSEGMENT** and the value for the row filter is **HOUSEHOLD**.

Create a new policy for a role

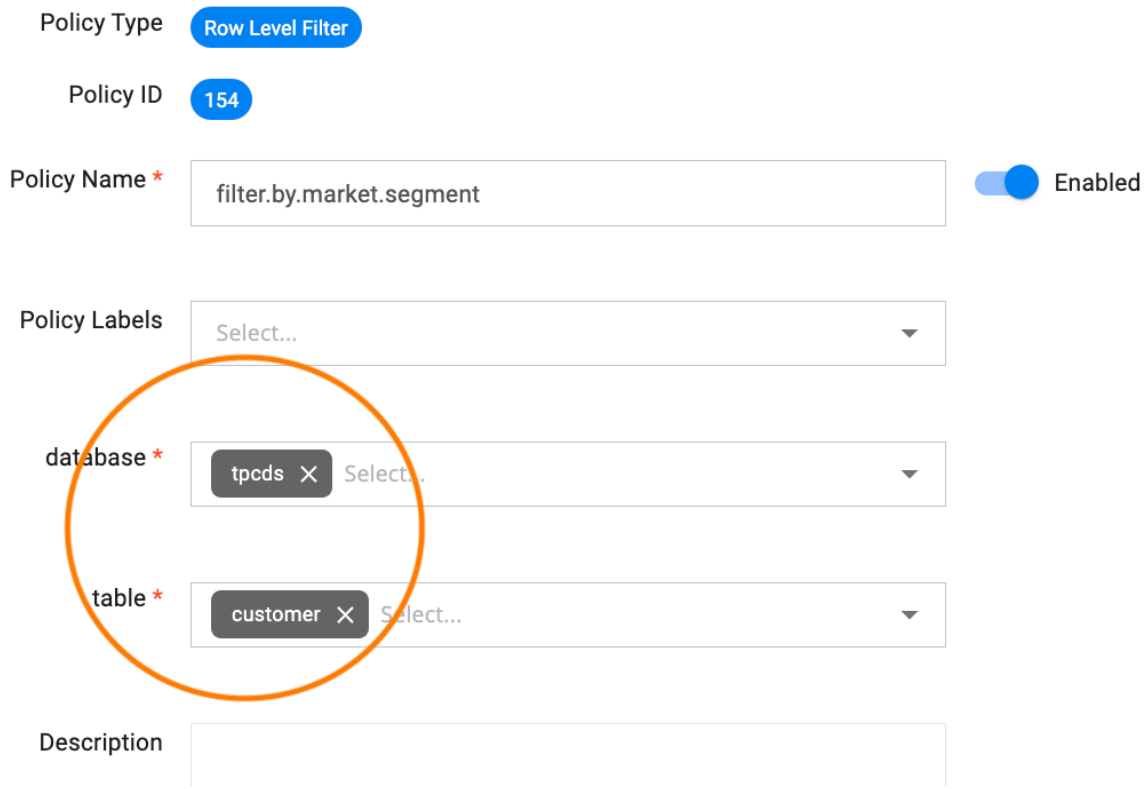
1. In Privacera, expand **Access Management** and click **Resource Policies**.
2. Under **DATABRICKS_SQL_ANALYTICS**, click the **privacera_databricks_sql_analytics** link.



Lists of policies are displayed on the following tabs:

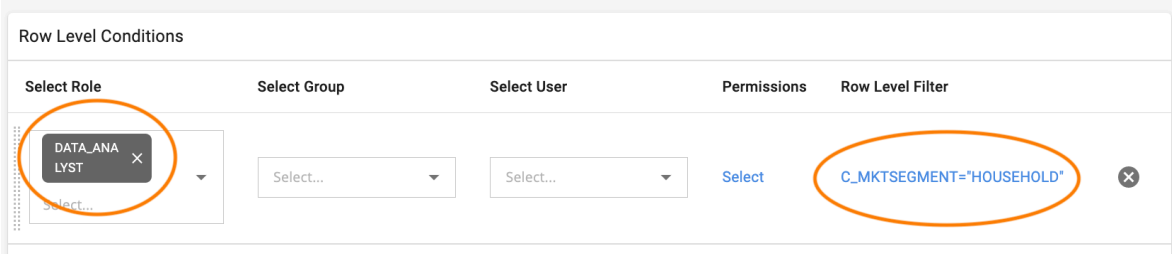
- **ACCESS**
- **MASKING**
- **ROW LEVEL FILTER**

3. On the **ROW LEVEL FILTER** tab, click **Add New Policy**.
   The **Policy Detail** page is displayed.
4. Enter a unique policy name.
   Enabled/Disabled: Accept the default, which is **Enabled**.
   Normal/Override: Accept the default, which is **Normal**.
5. Enter a descriptive policy label that helps you find this policy when searching for policies and filtering policy lists.
6. Select the required database.
   In this example, the database is **tpcds**.
7. Select the required table.
   In this example, the table is **CUSTOMER**..

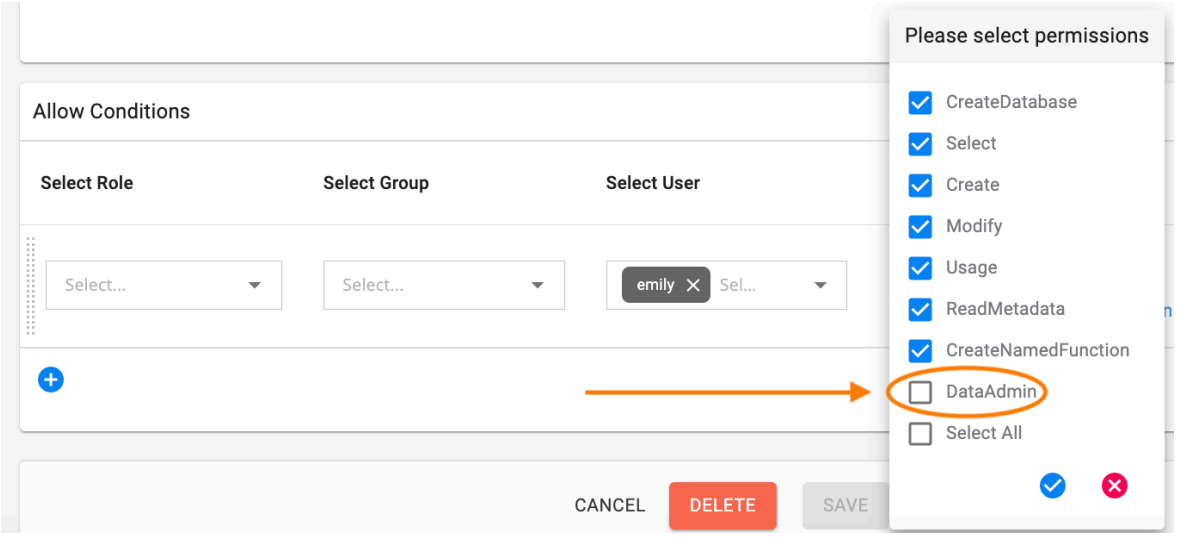


8. Ignore **Add Conditions**.
9. Enter a description of the policy to identify it among other policies.
10. **Enable/disable Audit Logging**: Accept the default, which is **Yes**.
11. In the **Allow Conditions** section, select a role and the row level filter for that role.
    The role in this example is **DATA_ANALYST**.

The filter should be entered as follows. The value is quoted:
columnName =" *value* "
In this example, the filter is: **C_MKTSEGMENT="HOUSEHOLD"**.



12. Click **Save** to complete the new policy details.
13. Privacera displays a message that you must have an access policy to accompany the policy you are creating. See Enable access to entire table for a user [223] for how to create an access policy. In that access policy, make sure you remove the **Data Admin** permission for the user. Otherwise, the user can see the original, unprotected database.



Secure database view created by Privacera

Databricks SQL does not have the native capability to create column masks or row filters.

For this reason, Privacera creates a secure view of the original database and applies policy to that secure view. The name of a secure view is:

originalDatabaseName `_secure`

In this example, the name of the secure view is `tpcds_secure`.

In Privacera, the access policy itself must always specify the name of the *original* database, not the secure view.

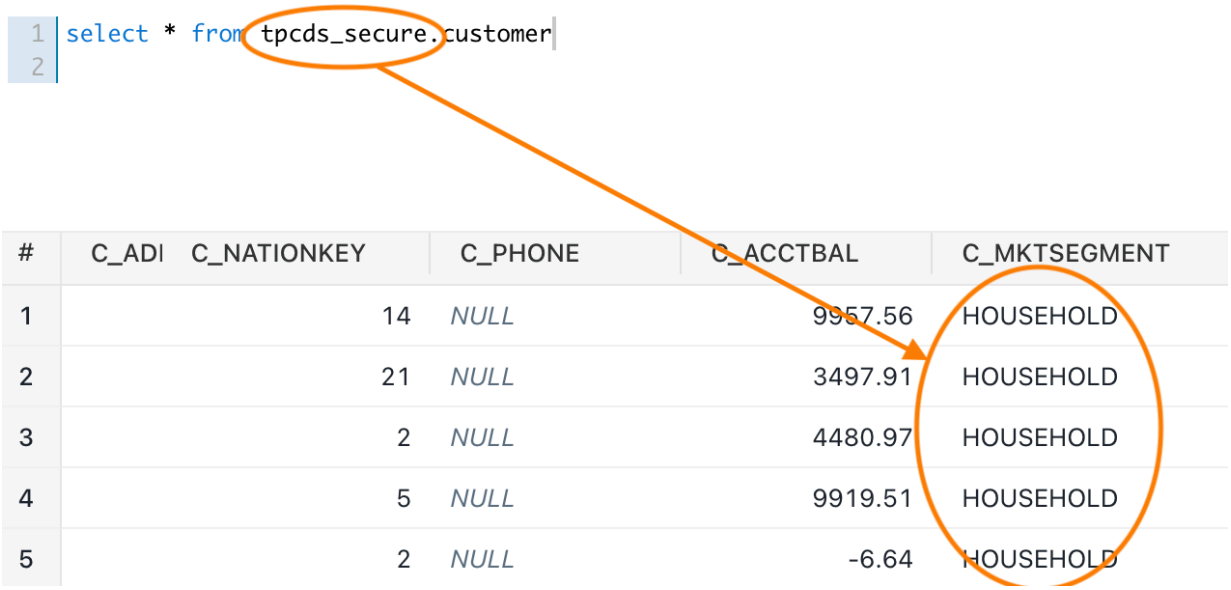
NOTE
You should tell users the name of the secure view for their queries and that access to the original database is no longer allowed.

Verify the policy in Databricks SQL

To see if the policy has been applied, log in to the database and enter the following SQL command against the secure view `tpcds_secure`:

```
select * from tpcds_seecure.CUSTOMER
```

The results show success. Only rows with market segment `HOUSEHOLD` are displayed:



#	C_ADI	C_NATIONKEY	C_PHONE	C_ACCTBAL	C_MKTSEGMENT
1		14	*NULL*	9957.56	HOUSEHOLD
2		21	*NULL*	3497.91	HOUSEHOLD
3		2	*NULL*	4480.97	HOUSEHOLD
4		5	*NULL*	9919.51	HOUSEHOLD
5		2	*NULL*	-6.64	HOUSEHOLD

Examples of access control via programming

Secure S3 via Boto3 in Databricks notebook

This section describes how to use the AWS SDK (Boto3) for PrivaceraCloud to enforce access control on AWS S3 file data through a Privacera Dataserver proxy.

These examples are intended to be run in a Databricks notebook.

Prerequisites

Make sure you have the following ready:

- Your Databricks datasource has been connected to PrivaceraCloud. See Connect Databricks to PrivaceraCloud
- Your PrivaceraCloud access key and secret key to use in the program. See Generate token, access key, and secret key on PrivaceraCloud. In the program, these are represented as `${privacera_access_key}` and `${privacera_secret_key}`.
- Your PrivaceraCloud API key and URL endpoint for use in the program. See API Key on PrivaceraCloud. In the program, this is shown as `${privacera_endpoint_url}`.
- At least one Privacera resource policy that you want to associate with the S3 path and enforce via this program. For a description, see Configure AWS S3 resource policies [161].

Create and run the program

1. Install the AWS Boto3 libraries:

   ```
   pip install boto3
   ```
2. Import the required libraries:

   ```
   import boto3
   ```

3. Access the AWS S3 files:

```
def check_s3_file_exists(bucket, key, access_key, secret_key, endpoint_url, dataserve
  exec_status = False
  access_key = ${privacera_access_key}
  secret_key = ${privacera_secret_key}
  endpoint_url = endpoint_url
  try:
    s3 = boto3.resource(service_name='s3', aws_access_key_id=access_key, aws_secret_a
    print(s3.Object(bucket_name=bucket, key=key).get()['Body'].read().decode('utf-8')
    exec_status = True
  except Exception as e:
    print("Got error: {}".format(e))
  finally:
    return exec_status

def read_s3_file(bucket, key, access_key, secret_key, endpoint_url, dataserver_cert,
  exec_status = False
  access_key = access_key
  secret_key = secret_key
  endpoint_url = endpoint_url
  try:
    s3 = boto3.client(service_name='s3', aws_access_key_id=access_key, aws_secret_acc
    obj = s3.get_object(Bucket=bucket, Key=key)
    print(obj['Body'].read().decode('utf-8'))
    exec_status = True
  except Exception as e:
    print("Got error: {}".format(e))
  finally:
    return exec_status

readFilePath = "file data/data/format=txt/sample/sample_small.txt"
bucket = "infraqa-test"
#saas
access_key = "${privacera_access_key}"
secret_key = "${privacera_secret_key}"
endpoint_url = "https://ds.privaceracloud.com"
dataserver_cert = ""
region_name = "us-east-1"
print(f"got file===== {readFilePath} ============= bucket= {bucket}")
status = check_s3_file_exists(bucket, readFilePath, access_key, secret_key, endpoint_
```

Other Boto3/Pandas examples to secure S3 in Databricks notebook with PrivaceraCloud

This section presents the following example programs using the AWS SDK (Boto3) and the Python Data Analysis Library (Pandas) for PrivaceraCloud to secure S3 file data:

- Read an S3 file
- Read an S3 file and write it to another location

These examples are intended to be run in a Databricks notebook.

Prerequisites

Make sure you have the following ready:

- Your Databricks datasource has been connected to PrivaceraCloud. See Connect Databricks to PrivaceraCloud

- Your PrivaceraCloud API key and URL endpoint for use in the program. See API Key on Privacera-Cloud. In the program, this is shown as `${privacera_endpoint_url}`.
- At least one Privacera resource policy that you want to associate with the S3 path and enforce via this program. For a description, see Configure AWS S3 resource policies [161].
- The examples assume you have a user whose policy allows access to the S3 data and another user who does not have that policy.
- The name of the AWS region where your data is. In the programs, this is shown as `${aws_region}`.
- The path and name of a data file on S3 whose access you want to control. In the example programs, this file is called `sample.csv`.

Create and run programs in a Databricks notebook

Install and upgrade the AWS Boto3 libraries:

```
pip install --upgrade boto3
```

Read the S3 file using Boto3

```
import boto3

access_key = "${privacera_access_key}"
secret_key =  "${privacera_secret_key}"

endpoint_url = "${privacera_endpoint_url}"
region_name = "${aws_region}"

# Create S3 client from this session
s3 = boto3.client(service_name='s3', aws_access_key_id=access_key, aws_secret_access_key

bucket = "infraqa-test-bubble"
key = "file data/data/format=csv/sample/sample.csv"

obj = s3.get_object(Bucket=bucket, Key=key)

print(obj['Body'].read().decode('utf-8'))
```

A user whose Privacera policy allows access successfully sees the desired data, as in this example:

```
EMP_SSN,CC,FIRST_NAME,LAST_NAME,ADDRESS,ZIPCODE,EMAIL,US_PHONE_FORMATTED
749-51-0571,341675287917956,Travis,Zyllah,939 Park Avenue,85297,Aliss.HENDRICK9807@gmail
214-99-5552,372017749988022,CHRYSTTOPHER,zylman,8380 Andell
.
.
.
```

A user who does not have a Privacera policy for access sees a failure message similar to the following. Notice HTTP status code 403 Forbidden.

```
ClientError: An error occurred (403) when calling the GetObject operation: Forbidden
```

Read an S3 file with Pandas

```
import boto3
import pandas as pd

access_key = "${privacera_access_key}"
secret_key =  "${privacera_secret_key}"
```

```
endpoint_url = "${privacera_endpoint_url}"
region_name = "${aws_region}"

# Create S3 client from this session
s3 = boto3.client(service_name='s3', aws_access_key_id=access_key, aws_secret_access_key
```

```
bucket = "infraqa-test-bubble"
key = "file data/data/format=csv/sample/sample.csv"

response = s3.get_object(Bucket=bucket, Key=key)
emp_df = pd.read_csv(response['Body'])
print(emp_df.head(2))
```

Write a copy of a file to a different path

```
import boto3
from io import StringIO

access_key = "${privacera_access_key}"
secret_key =  "${privacera_secret_key}"

endpoint_url = "${privacera_endpoint_url}"
region_name = "${aws_region}"

# Create S3 resource from this session
s3 = boto3.resource(service_name='s3', aws_access_key_id=access_key, aws_secret_access_k

csv_buffer = StringIO()
emp_df.to_csv(csv_buffer, sep="|", index=False)

# Write buffer to S3 object
s3.Object("infraqa-test-bubble","file data/output/format=csv/sample/sales_data/out/sampl
```

A user whose Privacera policy allows access sees a success message similar to the following. Notice HTTP status code 200 Success.

```
Out[6]: {'ResponseMetadata': {'RequestId': 'H40YEBKWX0QSWRC0',
  'HostId': 'Vjt46vIFn1wP4wYzKC5XveBmJKO+g1/y91T17iZdJrBwwJYyyFje04+cPhtNcB1sOmTDATDzJ3o
  'HTTPStatusCode': 200,
  'HTTPHeaders': {'date': 'Tue, 31 Jan 2023 18:24:18 GMT',
  .
  .
  .
```

A user who does not have a Privacera policy for access sees a failure message similar to the following. Notice HTTP status code 403 Forbidden.

```
ClientError: An error occurred (403) when calling the GetObject operation: Forbidden
```

Audit records of access success or failure

For each of the successful or failed attempts to work with the files in these program examples, Privacera records an audit record. See Examples of audit search [254].

Secure Azure file via Azure SDK in Databricks notebook

This section describes how to use the Azure SDK for PrivaceraCloud to access Azure Data Storage/Datalake file data through a Privacera DataServer proxy.

The following commands must be run in a Databricks notebook.

1. Install the Azure SDK libraries:

```
pip install azure-storage-file-datalake
```

2. Import the required libraries:

```
import os, uuid, sys
from azure.storage.filedatalake import DataLakeServiceClient
from azure.core._match_conditions import MatchConditions
from azure.storage.filedatalake._models import ContentSettings
```

3. Initialize the account storage through connection string method:

```
def initialize_storage_account_connect_str(my_connection_string):

    try:
        global service_client
        print(my_connection_string)

        service_client = DataLakeServiceClient.from_connection_string(conn_str=my_cor

    except Exception as e:
        print(e)
```

4. Prepare the connection string:

```
def prepare_connect_str():
    try:

        connect_str = "DefaultEndpointsProtocol=https;AccountName=${privacera_access_

       # sample value is shown below
       #connect_str = "DefaultEndpointsProtocol=https;AccountName=MMTTU5Njg4Njk0MDAwA

        return connect_str
    except Exception as e:
      print(e)
```

5. Define a sample access method to get Azure file and directories:

```
def list_directory_contents(connect_str):
    try:
        initialize_storage_account_connect_str(connect_str)

        file_system_client = service_client.get_file_system_client(file_system="{stor
        #sample values as shown below
        #file_system_client = service_client.get_file_system_client(file_system="infr

        paths = file_system_client.get_paths(path="{directory_path}")
        #sample values as shown below
        #paths = file_system_client.get_paths(path="file data/data/format=csv/sample/

        for path in paths:
            print(path.name + '\n')

    except Exception as e:
      print(e)
```

6. To verify that the proxy is functioning, call the access methods:

```
connect_str = prepare_connect_str()
list_directory_contents(connect_str)
```

Control access to S3 buckets with AWS Lambda function on PrivaceraCloud or Privacera Platform

You can control access to your S3 buckets with a Lambda function to protect them from unauthorized use.

To secure a bucket, you can create a Python function that creates an S3 client to check for proper authorization to access that bucket. If authorization is successful, the desired document is passed to the Privacera dataserver to give the requesting user the access.

A sample Lambda function is provided here.

Prerequisites

You need to make sure of the following:

- Your Privacera connection to S3 must have already been created.
- Decide which S3 buckets you want to protect.
- An S3 resource policy has been set in Privacera..
- You need your Privacera access key, secret key, and Privacera datatserver URL. See Get your access key, secret key, and value of PRIVACERA_DS_ENDPOINT_URL on PrivaceraCloud [238] or Get your access key, secret key, and value of PRIVACERA_DS_ENDPOINT_URL on Privacera Platform [238].
- If your Privacera dataserver uses a self-signed SSL certificate, the dataserver requires SSL validation. You need the absolute path to that dataserver SSL certificate when you create S3 client. The example program provided here shows this absolute path as the variable `certificate_path='/tmp/cert.pem'`.

Get your access key, secret key, and value of PRIVACERA_DS_ENDPOINT_URL on PrivaceraCloud

The values for access key, secret key, and the dataserver URL are included in the `privacera_aws.sh` script, which is downloadable as detailed in Scripts for AWS CLI or Azure CLI for managing connected applications.

In that script, use the value of the `DS_URL_HOST` variable as the value of the `PRIVACERA_DS_ENDPOINT_URL` variable in the Python Lambda function detailed here.

Get your access key, secret key, and value of PRIVACERA_DS_ENDPOINT_URL on Privacera Platform

To get the the values for the access key and secret key, see Generate tokens on Privacera Platform.

For the value of the `PRIVACERA_DS_ENDPOINT_URL` variable used in the example program:

- Go to **Launch Pad**.
- Under the heading **HTTP Proxy Setting**, copy the value displayed for **Host**.

Create Python Lambda function in AWS

The Lambda function needs to create an S3 client object with the Privacera dataserver URL as an endpoint URL for S3 with privacera access key and secret key generated for respective user.

The following example program shows the a sample `lambda_handler()` function to control access to an array of S3 buckets. You can modify this example or create your own based on it.

To create this program in AWS:

1. Follow Amazon's steps to create a Lambda function. See Getting started with Lambda.
2. Call the function **priv_list_bucket**.
3. In the **Create function** dashboard, select **Author from scratch**, and use the following values in creating the function's **Basic** information:
   - Function name: priv_list_bucket
   - Runtime: python 3.7
   - Architecture: x86_64
4. In **Permissions**, for **Execution** role, select **Use an existing role** for the Lambda function. This role must have permissions to execute Lambda functions. Example: AWS_Default_Role.
5. In the displayed **priv_list_bucket** function dashboard, in the **Code source code** field, add your Lambda function in **lambda_function.py** . You can use the example program or your own implementation of it.
6. In the **Runtime** settings, the **Function** name should be `<python_filename>.<function_name>`.
   In our example, we use `lambda_function.lambda_handler`.
7. To create a new test with an empty JSON input, click **Test** and **Save**.
8. If you see the message **Changes not deployed** for the test created in the previous step, click **Deploy**.
9. Click **Test** again.

The result of the test is displayed.

Example Python Lambda for Privacera Platform

If your Privacera dataserver uses a self-signed SSL certificate, the dataserver requires SSL validation. You need the absolute path to that dataserver SSL certificate when you create S3 client. The example program provided here shows this absolute path as the variable `certificate_path='/tmp/cert.pem'`.

```
import boto3
import os
import requests

# Set these variables with the values you
# obtained in the prerequisites.
PRIVACERA_DS_ENDPOINT_URL = ''
PRIVACERA_ACCESS_KEY = ''
PRIVACERA_SECRET_ACCESS_KEY = ''

def lambda_handler(event, context):

# Use the following code block
# if your Privacera dataserver
# relies on a self-signed SSL certificate.
# The value of the following variable must be
# the absolute path to that certificate.
  certificate_path='/tmp/cert.pem'
  certificate_url=PRIVACERA_DS_ENDPOINT_URL+'/services/certificate'
  headers = {
    'connection': 'close',
  }
  response = requests.get(certificate_url, headers=headers, verify=False)
  with open(certificate_path, 'wb') as f:
    f.write(response.content)
```

```
  session = boto3.session.Session()
  s3_client = session.client(
      service_name='s3',
      aws_access_key_id=PRIVACERA_ACCESS_KEY,
      aws_secret_access_key=PRIVACERA_SECRET_ACCESS_KEY,
      endpoint_url=PRIVACERA_DS_ENDPOINT_URL,
      verify=certificate_path
    )
  allBuckets = s3_client.list_buckets()
  data = [bucket["Name"] for bucket in allBuckets['Buckets']]
  return data
```

Example Python Lambda for PrivaceraCloud

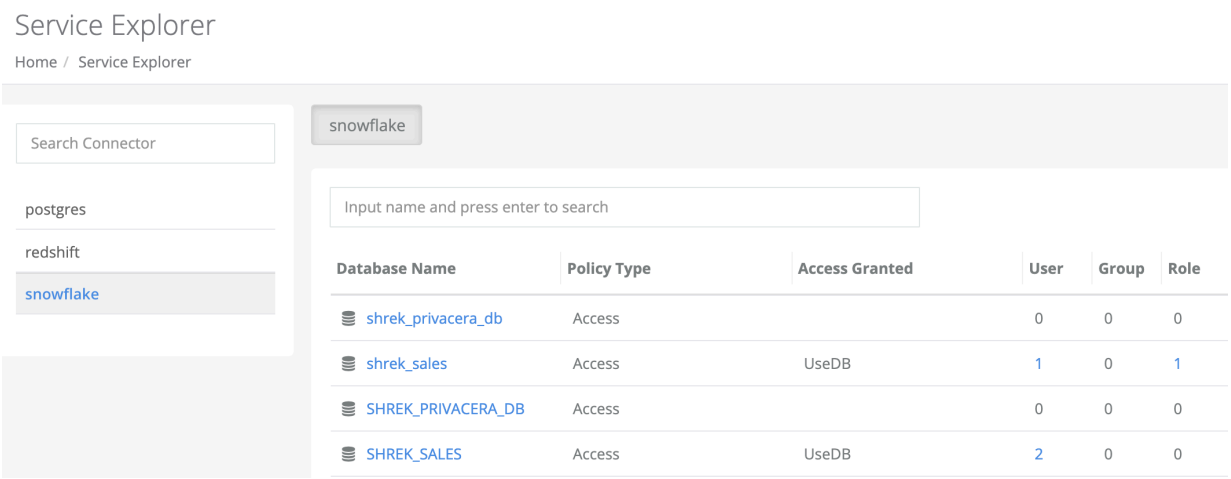
```
import boto3
import os
import requests

# Set these variables with the values you
# obtained in the prerequisites.
PRIVACERA_DS_ENDPOINT_URL = ''
PRIVACERA_ACCESS_KEY =''
PRIVACERA_SECRET_ACCESS_KEY = ''

def lambda_handler(event, context):

  session = boto3.session.Session()
  s3_client = session.client(
      service_name='s3',
      aws_access_key_id=PRIVACERA_ACCESS_KEY,
      aws_secret_access_key=PRIVACERA_SECRET_ACCESS_KEY,
      endpoint_url=PRIVACERA_DS_ENDPOINT_URL
    )
  allBuckets = s3_client.list_buckets()
  data = [bucket["Name"] for bucket in allBuckets['Buckets']]
  return data
```

Service Explorer

Service Explorer provides user, group, and role permission information with respect to a service's data sources, schemas, and tables. Its functionality is specific to data resources that are connected to Privacera using the **PolicySync** method. In Privacera, PolicySync'd data resources are resources, such as PostgreSQL, AWS Redshift, and Snowflake, that have their own native methods of protecting data. Privcera synchronizes resource and tag policy rules onto the target native methods. Connections to PolicySync resources are known as *connectors*.

To view the **Service Explorer** page, select **Access Management** > **Service Explorer** from the navigation menu.

The left side of the Service Explorer page displays a list of connectors in use. The middle of this page supports a drill-down approach. Select a connector to see a table of databases linked to that connector. For each connected database, you'll also see the associated *Policy Type* (such as Access), the *Access Granted* type (e.g. UseDB') and a count (each) of data access User(s), Group(s), and Role(s) provided with access.

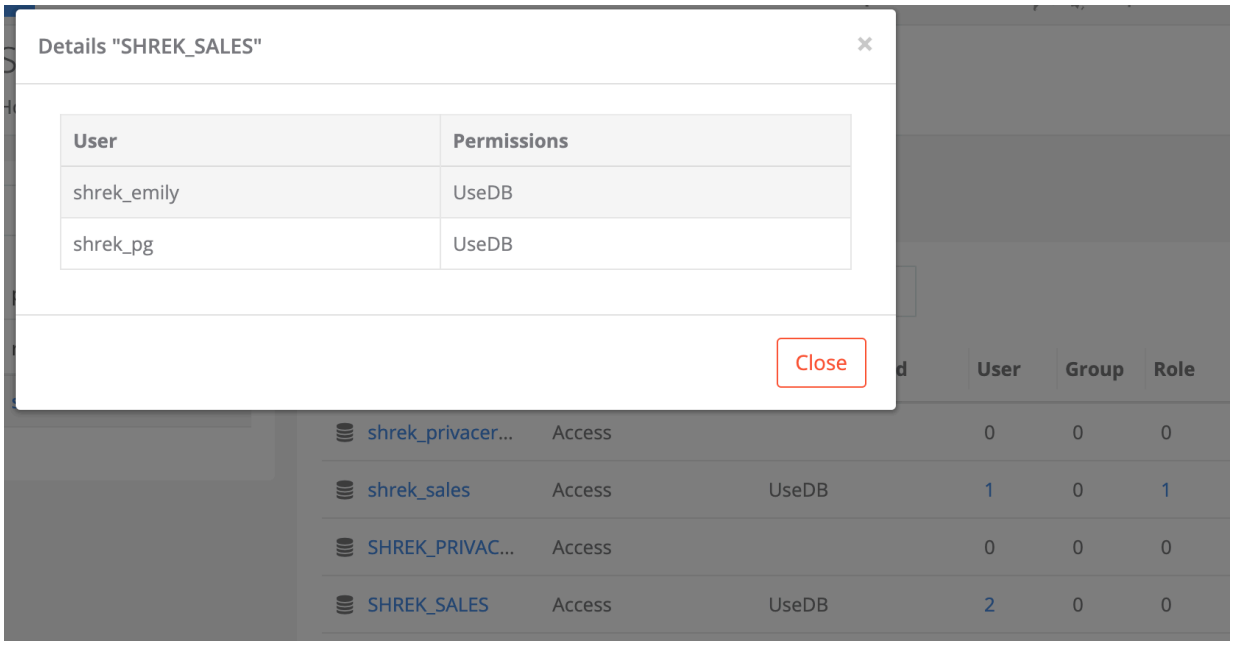


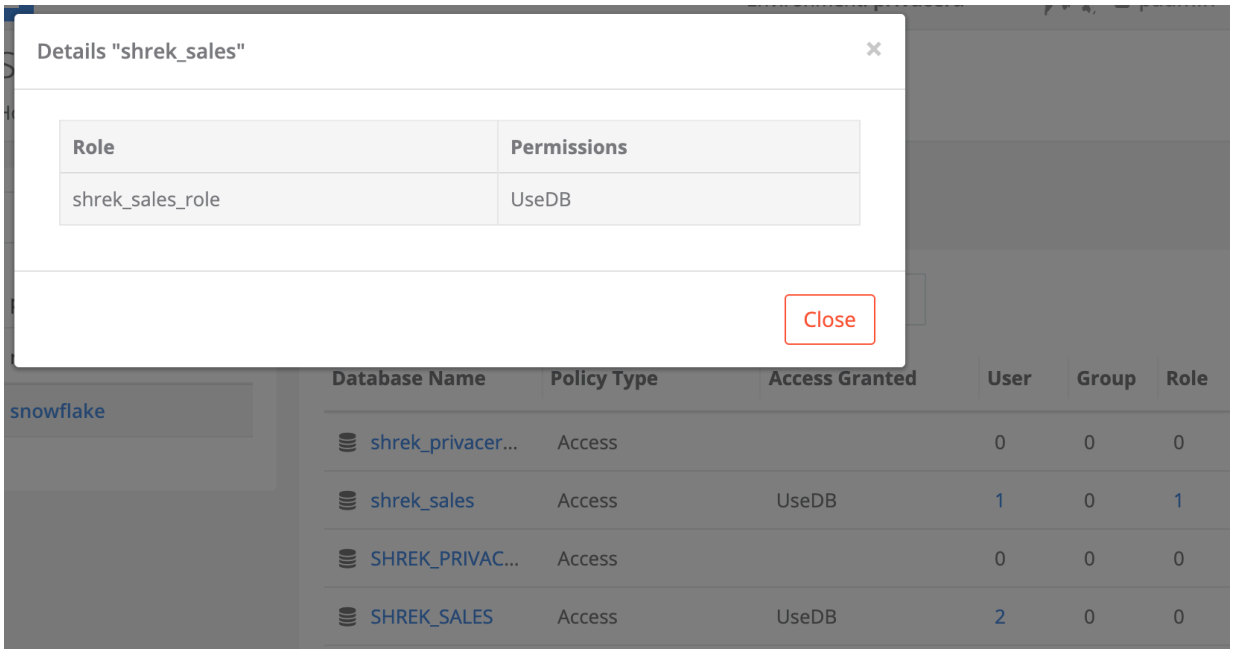
The Service Explorer interface allows drill-down to open each object in the table to show its sub-objects. Databases open to schemas which open to tables, which open to table column names. You can return to a parent view by clicking on its name in the breadcrumb list.

- Click a database to see a list of schemas in that database.
- Click a schema to see a list of tables in that schema.
- Click any of the tables to see a list of columns in that table.

At each level you'll see the *Policy Type* (Access, Deny), type of *Access Granted* (such as Select') and the a count of data access users, groups, and roles affected by that policy.

Click any non-zero users/group/role count, and you'll get a list of those users, groups, or roles associated with that object along with the user/group/role's associated permissions to that object.





Both the Connectors list and data repository object lists can be filtered by entering a character substring in the search/filter box above.

Audits

Privacera Access Management preserves audit records for all data accesses and important access policy-related changes. Administrators can use the built-in audit store, audit browser, and search capabilities to:

- Track recent access control enforcement decisions.
- View recent changes to policies, resources, security principals and entitlements.
- Monitor policy and user synchronization operations across systems under management.

Privacera limits audit record retention to 90 days.

Open access to the underlying Apache Solr audit data store is available, so that audit records can be extracted and forwarded to systems that more closely fit a customer's requirements for long-term audit management.

The Audits page includes information under the following categories:

- **Access:** Each access (or denial) to a managed data repository.
- **Admin:** Portal administrative activity including revisions to policies.
- **Login Sessions:** Logins to your PrivaceraCloud account web portal.
- **Plugin:** Logged status for each synchronization exchange with a data access plug-in component.
- **Plugin Status:** Logged updates with each data access plug-in component.
- **UserSync:** Logged user updates from LDAP/AD service.
- **PolicySync:** Logged queries to data resources integrated using policy sync method.

Categories are using the top navigation tab selected. A date filter is on the upper right. By default it opens to the last 7 days but can be set to other intervals and custom date ranges.

Required permissions to view audit logs on Privacera Platform

To view the audit page, you must be assigned either the `ROLE_ADMIN` or `ROLE_AUDITOR` role.

Anyone who can access the audit page can view all access audit log records for all data objects under management.

The Audits Page reports access to objects in all security zones to any user who has access to the audit page.

Some PolicySync connectors, when collecting audit records, are unable to annotate the audit record with the security zone(s) of tables referenced in each query. Audit records from those connectors do not specify security zone information. It may therefore be impractical to rely on filtering audit records based on security zone.

See the documentation for each connector for details on any audit limitations.

About PolicySync access audit records and policy ID on Privacera Platform

For data sources where Ranger plugins make policy decisions, those plugins can log the specific policy that was enforced, and the Policy ID column is populated with a link to the relevant policy.

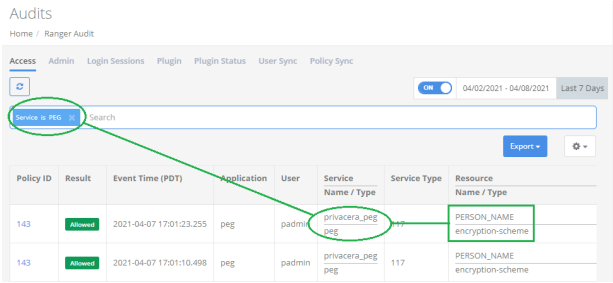
For data sources where Ranger plugins make policy decisions, those plugins can log the specific policy that was enforced, and the Policy ID column is populated with a link to the relevant policy.

View audit logs

1. From the home page, click **Access Management** > **Audits**.
2. Select a tab to see events in the associated category.
   - Access
   - Admin
   - Login Sessions
   - Plugin
   - Plugin Status
   - User Sync
   - Policy Sync
3. (Optional) Select a time range for the events you want to see. The default is seven days.

View PEG API audit logs

On the **Access** tab, use the search filter pulldown menu to see **Service is PEG** (Privacera Encryption Gateway).



This shows access to a PEG encryption key when a PEG REST API request specifies an encryption scheme.

For more information about PEG, see Get started with Encryption.

Generate audit logs using GCS lineage

To generate audit logs using GCS lineage, configure the following setting on the GCP Console to get the events in Google Logs Explorer.

1. Log on to Google Cloud Console and navigate to IAM Admin ->> Audit Logs
2. In the search bar, search for the "Google Cloud Storage" and then select the checkbox in the grid.
3. At right panel, in Google Cloud Storage dialog, select the **Admin Read** and **Data Read** checkboxes and click **Save**.

> **NOTE**
> Perform the above step for all the projects where you will be executing GCS realtime scan and expecting lineage to be generated.

> **NOTE**
> You can configure GCS lineage time using custom properties that are not included by default. See Set custom Discovery properties on Privacera Platform.

Configure Audit Access Settings on PrivaceraCloud

> **NOTE**
> Contact Privacera Support to request enabling this feature.

The access audits in the Audits [243] page are retained for 90 days in the storage of PrivaceraCloud account. If you want to keep the access audit records for much longer, you can copy the audit records from PrivaceraCloud storage to your AWS bucket. The copied audit records in your AWS bucket is the ZIP or TAR format.

When you configure the AWS bucket and region, an ARN Role will be generated automatically by PrivaceraCloud. After configuring this setting, you can see the ARN role in your PrivaceraCloud account. This will be used in the policy of your AWS S3 bucket.

When this feature is enabled, you can see the **Audit Access Settings** section in the **Account** page.

1. In the **Audit Access Settings** section of the **Account** page:
   a. Click the **Backup of Access Audits (AWS)** toggle button.
      The **Privacera Access Audit Configuration** dialog appears.
   b. In the dialog, enter a bucket name or a folder path and bucket region.

      > **NOTE**
      > You cannot modify the parameters after saving the bucket name and region.

   c. Click **Save Settings**. An ARN Role will be generated by PrivaceraCloud.
   d. Click the **SHOW DETAILS** button to get the ARN Role.
      The **Privacera Access Audit Configuration** dialog appears.
   e. Under the **User Role** section, copy the ARN Role.
2. In the AWS console, add the following bucket policy to your AWS S3 bucket:

```
{
"Id": "Policy1645104586202",
"Version": "2012-10-17",
"Statement": [
    {
    "Sid": "Stmt1645104584705",
    "Action": "s3:PutObject",
    "Effect": "Allow",
    "Resource": [
        "arn:aws:s3:::<bucket_name_or_folder_path>",
        "arn:aws:s3:::<bucket_name_or_folder_path>/*"
        ],
    "Principal": {
        "AWS": [
        "<ARN_ROLE>"
        ]
    }
    }
]
}
```

In the policy above, edit the following information:
- `<bucket_name_or_folder_path>`: Add the bucket name or folder path where the audit records will get copied.
- `<ARN_ROLE>`: Add the ARN Role copied from PrivaceraCloud portal.
  For example, `arn:aws:iam::9xxxx56xxxx0:role/PRIVACERA_AUDIT_1xxxxx933xxxx2_ROLE`.

Configure AWS RDS PostgreSQL instance for access audits

You can configure your AWS account to allow Privacera to access your RDS PostgreSQL instance audit logs through Amazon cloudWatch logs. To enable this functionality, you must make the following changes in your account:

- Update the AWS RDS parameter group for the database
- Create an AWS SQS queue
- Specify an AWS Lambda function
- Create an IAM role for an EC2 instance

Update the AWS RDS parameter group for the database

To expose access audit logs, you must update configuration for the data source.

Procedure

1. Log in to your AWS account.
2. To create a role for audits, run the following SQL query with a user with administrative credentials for your data source:

   ```
   CREATE ROLE rds_pgaudit;
   ```
3. Create a new parameter group for your database and specify the following values:
   - **Parameter group family**: Select a database from either the `aurora-postgresql` or `postgres` families.
   - **Type**: Select **DB Parameter Group.**
   - **Group name**: Specify a group name for the parameter group.
   - **Description**: Specify a description for the parameter group.
4. Edit the parameter group that you created in the previous step and set the following values:
   - `pgaudit.log`: Specify `all`, overwriting any existing value.
   - `shared_preload_libraries`: Specify `pg_stat_statements,pgaudit`.
   - `pgaudit.role`: Specify `rds_pgaudit`.
   - `pgaudit.log_rotation = 1` : Specify 1 (true)
5. Associate the parameter group that you created with your database. Modify the configuration for the database instance and make the following changes:
   - **DB parameter group**: Specify the parameter group you created in this procedure.
   - **PostgreSQL log**: Ensure this option is set to enable logging to Amazon cloudWatch logs.
6. When prompted, choose the option to immediately apply the changes you made in the previous step.
7. Restart the database instance.

Verification

To verify that your database instance logs are available, complete the following steps:

1. From the Amazon RDS console, View the logs for your database instance from the RDS console.
2. From the CloudWatch console, complete the following steps:
   a. Find the `/aws/rds/cluster/*` log group that corresponds to your database instance.

b. Click the log group name to confirm that a log stream exists for the database instance, and then click on a log stream name to confirm that log messages are present.

Create an AWS SQS queue

To create an SQS queue used by an AWS Lambda function that you will create later, complete the following steps.

1. From the AWS console, create a new Amazon SQS queue with the default settings. Use the following format when specifying a value for the **Name** field:

   ```
   privacera-postgres-<RDS_CLUSTER_NAME>-audits
   ```

   where:
   - `RDS_CLUSTER_NAME`: Specifies the name of your RDS cluster.
2. After the queue is created save the **URL** of the queue for use later.

Specify an AWS Lambda function

To create an AWS Lambda function to interact with the SQS queue, complete the following steps. In addition to creating the function, you must create a new IAM policy and associate a new IAM role with the function. You need to know your AWS account ID and AWS region to complete this procedure.

1. From the IAM console, create a new IAM policy and input the following JSON:

   ```
   {
       "Version": "2012-10-17",
       "Statement": [
           {
               "Effect": "Allow",
               "Action": "logs:CreateLogGroup",
               "Resource": "arn:aws:logs:<REGION>:<ACCOUNT_ID>:*"
           },
           {
               "Effect": "Allow",
               "Action": [
                   "logs:CreateLogStream",
                   "logs:PutLogEvents"
               ],
               "Resource": [
                   "arn:aws:logs:<REGION>:<ACCOUNT_ID>:log-group:/aws/lambda/<LAMBDA_FUN
               ]
           },
           {
               "Effect": "Allow",
               "Action": "sqs:SendMessage",
               "Resource": "arn:aws:sqs:<REGION>:<ACCOUNT_ID>:<SQS_QUEUE_NAME>"
           }
       ]
   }
   ```

   where:
   - `REGION`: Specify your AWS region.
   - `ACCOUNT_ID`: Specify your AWS account ID.
   - `LAMBDA_FUNCTION_NAME`: Specify the name of the AWS Lambda function, which you will create later. For example: `privacera-postgres-cluster1-audits`
   - `SQS_QUEUE_NAME`: Specify the name of the AWS SQS Queue.
2. Specify a name for the IAM policy, such as `privacera-postgres-audits-lambda-execution-policy`, and then create the policy.

3. From the IAM console, create a new IAM role and choose for the **Use case** the **Lambda** option.
4. Search for the IAM policy that you just created with a name that might be similar to `privacera-postgres-audits-lambda-execution-policy` and select it.
5. Specify a **Role name** for the IAM policy, such as `privacera-postgres-audits-lambda-execution-role`, and then create the role.
6. From the AWS Lambda console, create a new function and specify the following fields:
   - **Function name**: Specify a name for the function, such as `privacera-postgres-cluster1-audits`.
   - **Runtime**: Select **Node.js 12.x** from the list.
   - **Permissions**: Select **Use an existing role** and choose the role created earlier in this procedure, such as `privacera-postgres-audits-lambda-execution-role`.
7. Add a trigger to the function you created in the previous step and select **CloudWatch Logs** from the list, and then specify the following values:
   - **Log group**: Select the log group path for your Amazon RDS database instance, such as `/aws/rds/cluster/database-1/postgresql`.
   - **Filter name**: Specify `auditTrigger`.
8. In the Lambda source code editor, provide the following JavaScript code in the `index.js` file, which is open by default in the editor:

```
var zlib = require('zlib');

// CloudWatch logs encoding
var encoding = process.env.ENCODING || 'utf-8';  // default is utf-8
var awsRegion = process.env.REGION || 'us-east-1';
var sqsQueueURL = process.env.SQS_QUEUE_URL;
var ignoreDatabase = process.env.IGNORE_DATABASE;
var ignoreUsers = process.env.IGNORE_USERS;

var ignoreDatabaseArray = ignoreDatabase.split(',');
var ignoreUsersArray = ignoreUsers.split(',');

// Import the AWS SDK
const AWS = require('aws-sdk');

// Configure the region
AWS.config.update({region: awsRegion});

exports.handler = function (event, context, callback) {

    var zippedInput = Buffer.from(event.awslogs.data, 'base64');

        zlib.gunzip(zippedInput, function (e, buffer) {
        if (e) {
            callback(e);
        }

        var awslogsData = JSON.parse(buffer.toString(encoding));

        // Create an SQS service object
        const sqs = new AWS.SQS({apiVersion: '2012-11-05'});

        console.log(awslogsData);
        if (awslogsData.messageType === 'DATA_MESSAGE') {

            // Chunk log events before posting
```

```
                awslogsData.logEvents.forEach(function (log) {

                    //// Remove any trailing \n
                    console.log(log.message)

                    // Checking if message falls under ignore users/database
                    var sendToSQS = true;

                    if(sendToSQS) {

                        for(var i = 0; i < ignoreDatabaseArray.length; i++) {
                           if(log.message.toLowerCase().indexOf("@" + ignoreDatabaseArray
                                sendToSQS = false;
                                break;
                           }
                        }
                    }

                    if(sendToSQS) {

                        for(var i = 0; i < ignoreUsersArray.length; i++) {
                           if(log.message.toLowerCase().indexOf(ignoreUsersArray[i] + "@"
                                sendToSQS = false;
                                break;
                           }
                        }
                    }

                    if(sendToSQS) {

                        let sqsOrderData = {
                            MessageBody: JSON.stringify(log),
                            MessageDeduplicationId: log.id,
                            MessageGroupId: "Audits",
                            QueueUrl: sqsQueueURL
                        };

                        // Send the order data to the SQS queue
                        let sendSqsMessage = sqs.sendMessage(sqsOrderData).promise();

                        sendSqsMessage.then((data) => {
                            console.log("Sent to SQS");
                        }).catch((err) => {
                            console.log("Error in Sending to SQS = " + err);
                        });

                    }
                });
            }
        });
};
```

9. For the Lambda function, edit the environment variables and create the following environment variables:
   - `REGION`: Specify your AWS region.
   - `SQS_QUEUE_URL`: Specify your AWS SQS queue URL.

- `IGNORE_DATABASE`: Specify `privacera_db`.
- `IGNORE_USERS`: Specify your database administrative user, such as `privacera`.

Create an IAM role for an EC2 instance

To create an IAM role for the AWS EC2 instance where you installed Privacera so that Privacera can read the AWS SQS queue, complete the following steps:

1. From the IAM console, create a new IAM policy and input the following JSON:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sqs:DeleteMessage",
                "sqs:GetQueueUrl",
                "sqs:ListDeadLetterSourceQueues",
                "sqs:ReceiveMessage",
                "sqs:GetQueueAttributes"
            ],
            "Resource": "<SQS_QUEUE_ARN>"
        },
        {
            "Effect": "Allow",
            "Action": "sqs:ListQueues",
            "Resource": "*"
        }
    ]
}
```

   where:
   - `SQS_QUEUE_ARN`: Specifies the AQS SQS Queue ARN identifier for the SQS Queue you created earlier.
2. Specify a name for the IAM policy, such as `postgres-audits-sqs-read-policy`, and create the policy.
3. Attach the IAM policy to the AWS EC2 instance where you installed Privacera.

Accessing PostgreSQL Audits in GCP

Prerequisites

Ensure the following prerequisites are met:

- **gcloud** command-line tool is installed. See gcloud tool overview
- Google **Cloud SDK** is installed. See Installing Cloud SDK

Configuration

1. In GCP:
   a. Run the following commands on Google Cloud's shell (gcloud) by providing `GCP_PROJECT_ID` and `INSTANCE_NAME`.

```
gcloud sql instances patch  {INSTANCE_NAME} --database-flags=cloudsql.enable_pgau
```

   b. Run a SQL command using a compatible psql client to create the pgAudit extension.

```
CREATE EXTENSION pgaudit;
```

c. Create a service account and private key JSON file, which will be used by PolicySync to pull access audits. See Setting up authentication and edit the following fields:
   - **Service account name**: Enter any user-defined name. For example, policysync-postgres-gcp-audit-service-account.
   - **Select a role**: Select **Private Logs Viewer** role.
   - **Create new key**: Create a service account key and download the JSON file in the **custom-vars** folder.

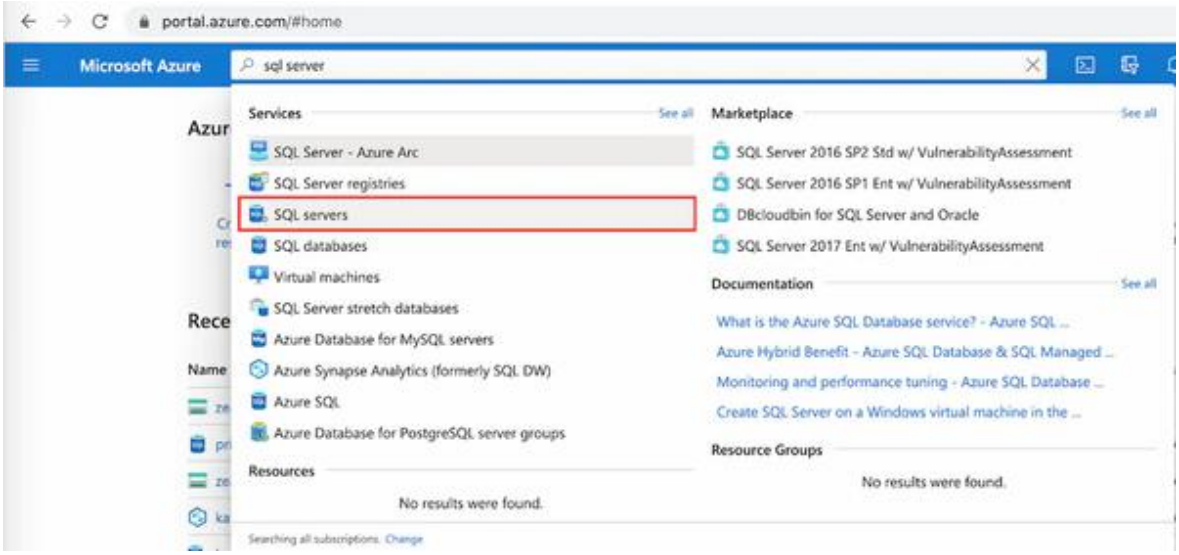
2. In Privacera Manager:
   Add the following properties in `vars.policysync.postgres.yml` file:

```
POSTGRES_AUDIT_SOURCE:"gcp_pgaudit"
POSTGRES_GCP_AUDIT_SOURCE_INSTANCE_ID:""
POSTGRES_OAUTH_PRIVATE_KEY_FILE_NAME:""
```

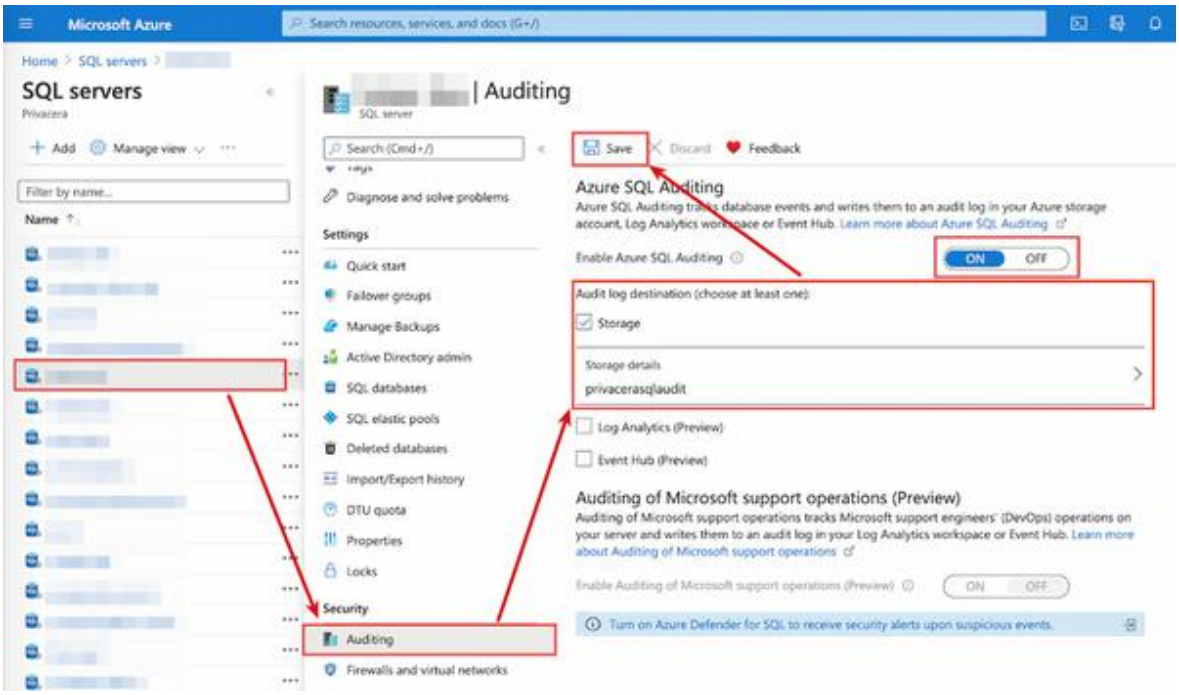
Configure Microsoft SQL server for database synapse audits

To configure MS SQL server for database or Synapse audits, use the following steps:

1. Login to Azure portal.
2. Search for **SQL Servers** in which you want to configure MSSQL for Azure AD users.



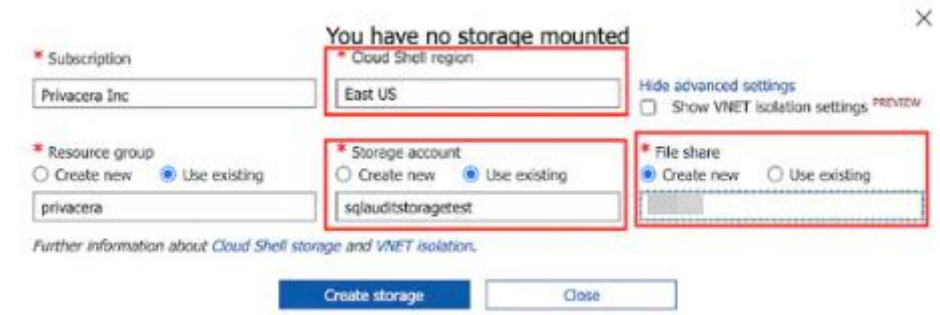
3. Select Azure AD user, and then click **Auditing**.
4. Click the toggle button to **ON** to **Enable Azure SQL Auditing**.
5. Select the **Storage** checkbox to set audit log destination as storage account, and then select your existing storage account from the **Storage Details**.
6. Click the **Save** button.



7. To open the Azure cloud shell, click the shell icon, , on the top menu bar, and then click **PowerShell**

8. Click **Show advanced settings**.

9. In the **Cloud Shell region** text box, enter your region.
10. In the **Storage account**, select **Use existing** to use your existing storage account.
11. In the **File share**, select **Create new**, and then enter name.
12. Click the **Create storage** button.



Cloud powershell window will appear, you can run your commands in the powershell.

13. Run the following command, if you have **Azure MSSQL Database**:

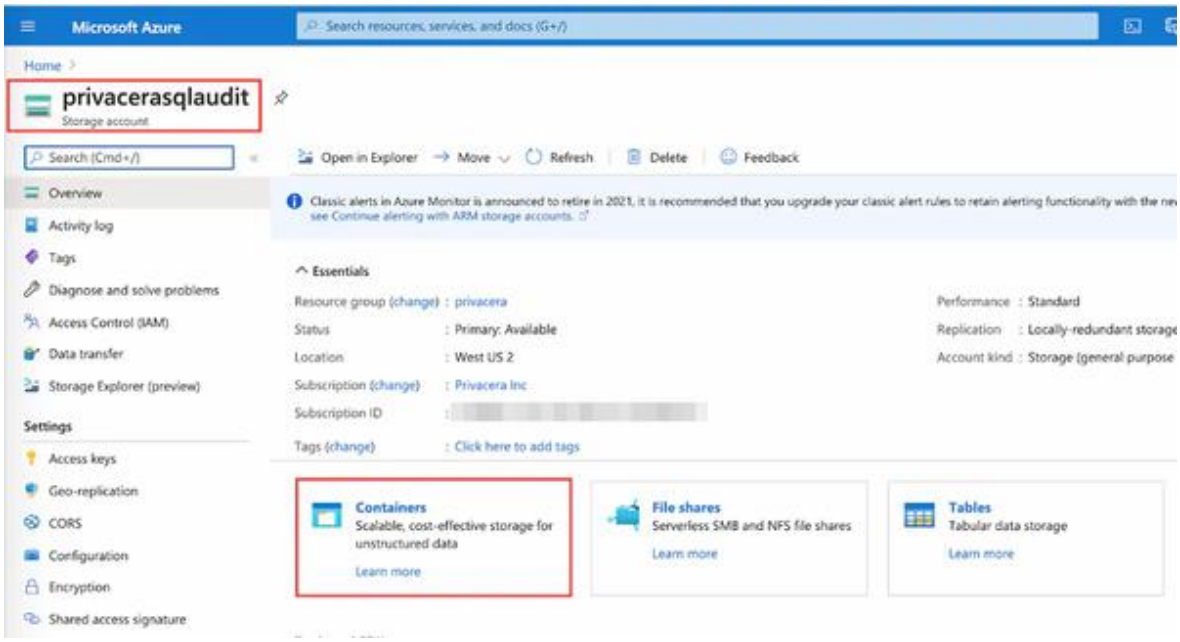
```
Set-AzSqlServerAudit -ResourceGroupName "${RESOURCE_GROUP}" -ServerName "${MSSQL_SERV
SCHEMA_OBJECT_ACCESS_GROUP,DATABASE_OBJECT_CHANGE_GROUP,SCHEMA_OBJECT_CHANGE_GROUP
```

14. Run the following command, if you have **Azure Synapse Database**:

```
Set-AzSqlServerAudit -ResourceGroupName "${RESOURCE_GROUP}" -ServerName "${MSSQL_SERV
```

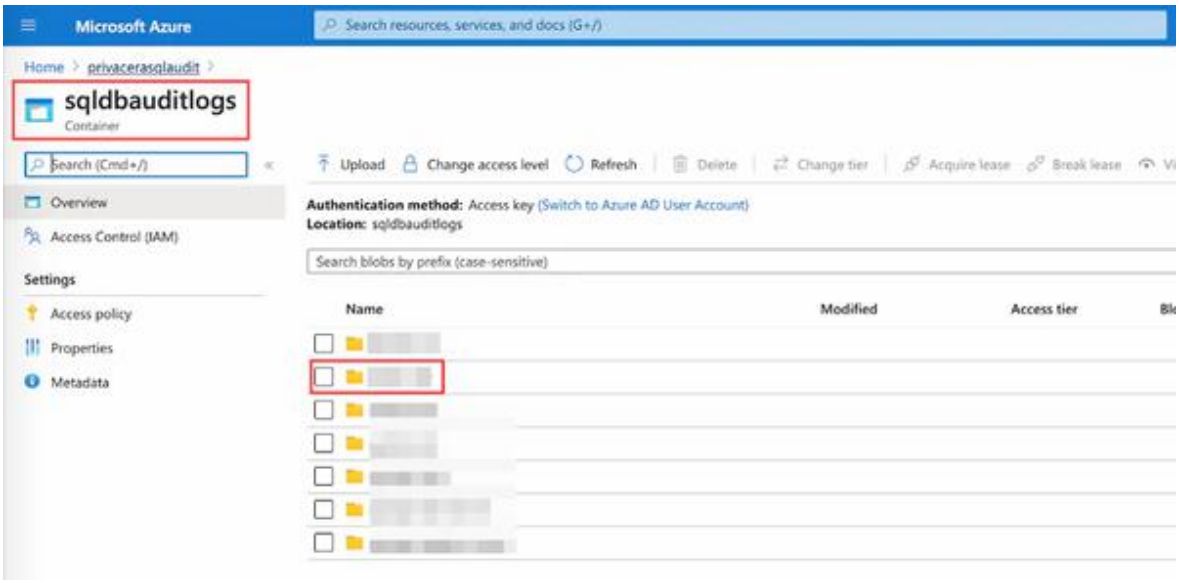
The above queries will take around one or two minutes to be completed.

15. Go to your **Storage account** in which you have configured MSSQL Server for auditing purpose, and then click **Containers**.



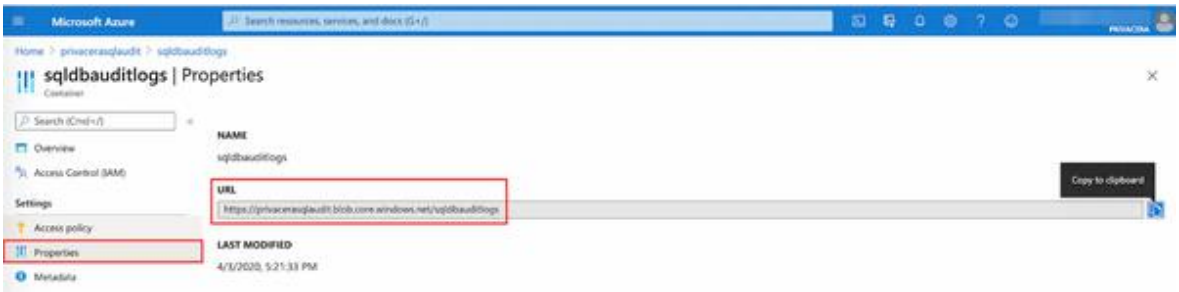
Note

Make sure that your MSSQL Server name directory is visible inside your audit log container. It might take some time to appear inside the container.



Now, you need to form an **Audit Storage URL** for your MSSQL Server.

16. Go to **Properties**, and then copy the container URL.



Now, your Audit storage url will be `${CONTAINER_URL}/${MSSQL_SERVER_NAME}`

Examples of audit search

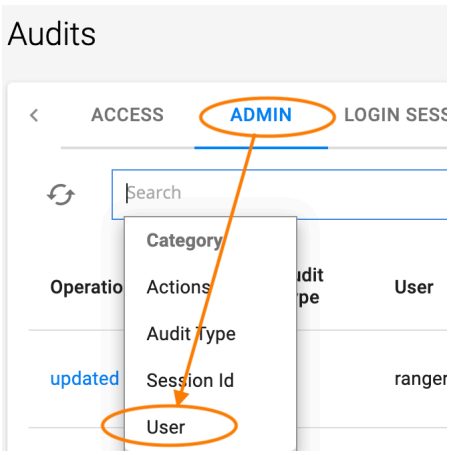
These examples show using **Search** on the **Audits** page to find records of various kinds.

As you position your cursor in the **Search** box, the system guides you through refining your search. You can pick from various displayed menus as you refine.

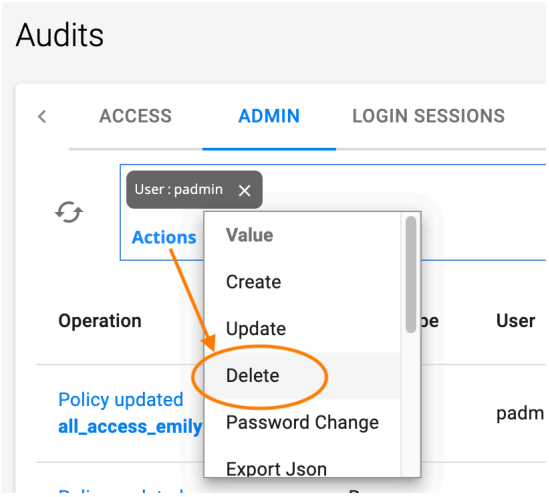
The displayed records indicate either success or failure of the attempted access.

Find policies deleted by an administrator

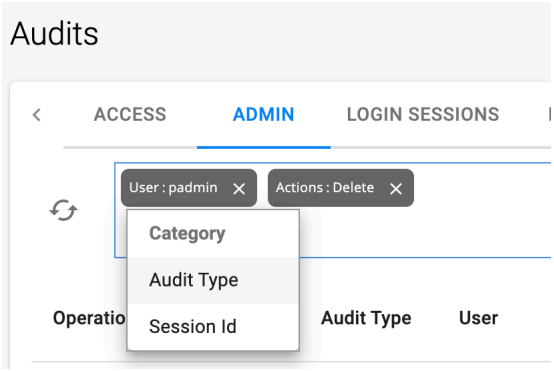
Click the **ADMIN** tab, and in the **Search** box, select **User**. This example shows the default administrative user **padmin**.



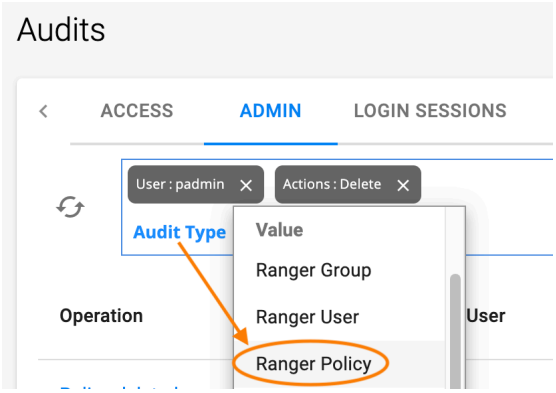
For **Actions**, specify **Delete**.



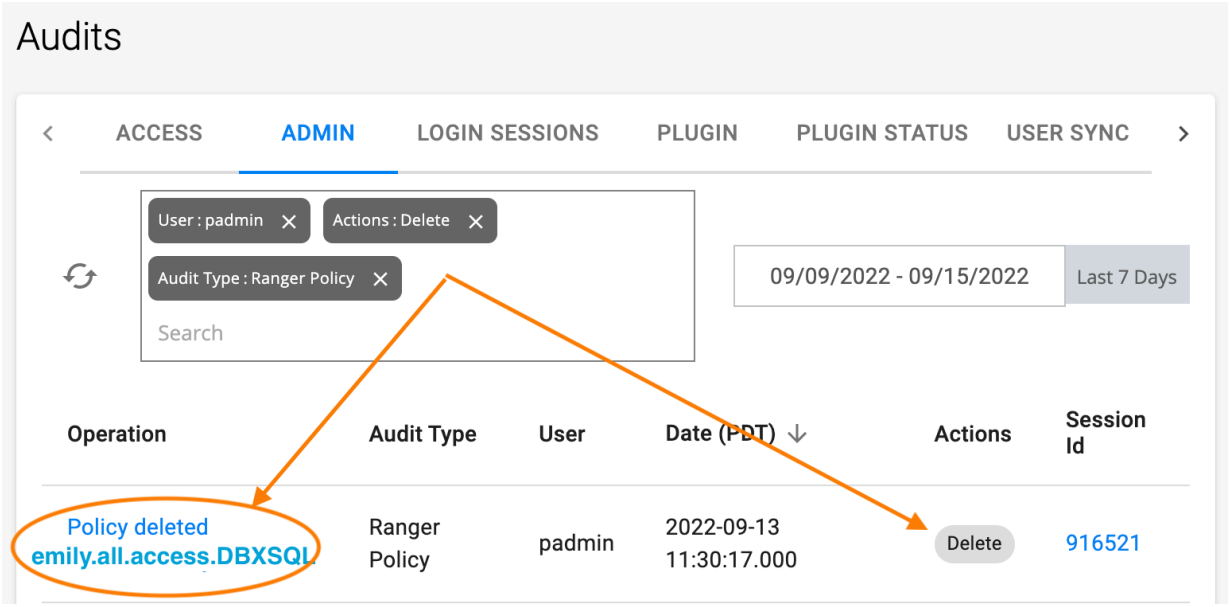
For **Category**, select **Audit Type**.



For **Audit Type**, select **Ranger Policy**.

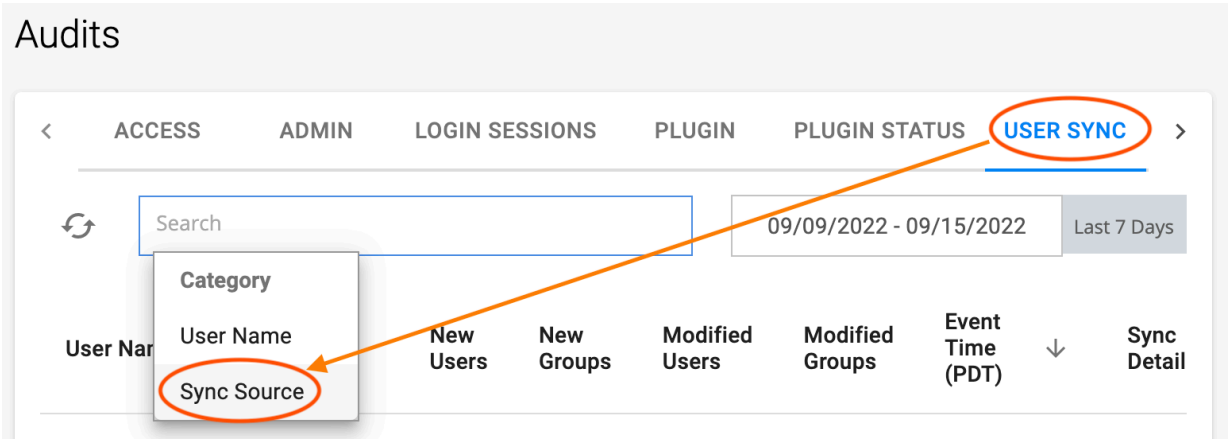


The results are displayed.

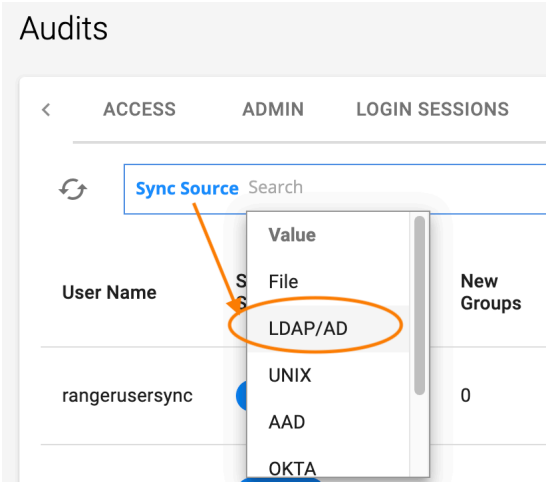


Find statistics of a UserSync from LDAP

Click the **USER SYNC** tab, and in the **Search** box, for **Category**, select **Sync Source**.



For **Sync Source**, select **LDAP/AD**.



The results are displayed. For details about a record, click the eye on the far right.

rangerusersync	LDAP/AD	0	0	0	0	2022-09-14 15:06:02.000
rangerusersync	LDAP/AD	0	0	1	1	2022-09-14 14:06:02.000
rangerusersync	LDAP/AD	0	0	1	1	2022-09-14 13:55:34.000
rangerusersync	LDAP/AD	0	0	1	1	2022-09-14 13:45:35.000

Sync Details

Ldap url	ldap://10.212.1.70:389
User search Enabled	true
Group search First Enabled	false
Total number of groups synced	26
Group hierarchy level	0
Total number of users synced	69

CLOSE

Reports

To view reports, select **Access Management** > **Reports** from the navigation menu.

The **Reports** page displays the following information:

- A summary view of all defined policies.
- Optional export of displayed results.
- An alternate means of locating a policy and opening it for edit.

View policy conditions from the Reports page

- To see the conditions in a policy, click the plus sign in the condition column.

Reports page filters

The Reports page has the following filters:

- **Basic Filters**
  - Policy Type > Select policy type > Access, Mask, or Row Level Filter.
  - Policy Name > Filter by policy name.
- **Advanced Filters:**
  - Component: Specify the resource or tag component. For example: Hive, S3, DynamoDB, tag.
  - Resource: Specify the resource path used while creating the policy.
  - Policy Label: Specify the policy label.
  - Group or Username: Specify the group or user name assigned to the policy.
  - Zone Name: Specify the security zone name.

Edit policies from the Reports page

- To edit a policy, click the **Policy ID**.

Search for policies from the Reports page

1. Type a **Policy Name** and select a policy type.
2. (Optional) Click **More Filters** and select additional search criteria.

Export Policy Reports

1. Click the **Export** button and select the file format.
2. The report will contain the same policies as are displayed in the grid.
3. Export supports the following formats CSV, Excel, and JSON.