Snowflake Cookbook - Monitoring

Contents

[Architecture of The Snowflake Monitoring 3](#_Toc134601621)

[Setup Account Monitoring 4](#_Toc134601622)

[Create The Metrics Reporting Database 4](#_Toc134601623)

[Create The Metrics Collection Task 6](#_Toc134601624)

[Create The Metrics Combiner Task 10](#_Toc134601625)

[Security Monitoring 11](#_Toc134601626)

[❖ Verify connections are logging in from specific networks 11](#_Toc134601627)

[❖ Monitor Grants Not Made By SECURITYADMIN Role 11](#_Toc134601628)

[❖ Monitor Built-in Admin Roles Activity 11](#_Toc134601629)

[❖ Monitor Users Created 11](#_Toc134601630)

[❖ Monitor User Modification 12](#_Toc134601631)

[❖ Monitor How Users Are Authenticated 12](#_Toc134601632)

[❖ Monitor Users Who Used SSO Before And Now Using Other Authentication Methods 12](#_Toc134601633)

[❖ Monitor For Key Pair Authentication 12](#_Toc134601634)

[❖ Monitor If Key Pair Authentication Users Also Configured For Other Methods 13](#_Toc134601635)

[❖ Monitor For SCIM Access Token Creation 13](#_Toc134601636)

[❖ Monitor Failed Login Attempts 13](#_Toc134601637)

[❖ Monitor Snowflake MFA 14](#_Toc134601638)

[❖ Monitor For Re-Enabling Disabled Users 14](#_Toc134601639)

[❖ Monitor for Enabled Plaintext User Passwords 14](#_Toc134601640)

[❖ Monitor SCIM user-provisioning API calls 14](#_Toc134601641)

[❖ Monitor Highly Privileged Database Objects 15](#_Toc134601642)

[❖ Monitor ACCOUNTADMIN role grants 15](#_Toc134601643)

[❖ Monitor grants to the public role 15](#_Toc134601644)

[Operational Monitoring 16](#_Toc134601645)

[❖ Monitor Network Policies 16](#_Toc134601646)

[❖ Show Network and User-level Policy Settings 16](#_Toc134601647)

[❖ Monitor for long sessions 16](#_Toc134601648)

[PII Monitoring 17](#_Toc134601649)

[❖ Monitor Use of PII Roles 17](#_Toc134601650)

[❖ Monitor Tables Not Accessed 17](#_Toc134601651)

[❖ Monitor Users With PII Access 17](#_Toc134601652)

[Encryption and Key Management Monitoring 18](#_Toc134601653)

[❖ Verify that annual rekeying is enabled 18](#_Toc134601654)

[❖ Monitor for any changes to the periodic\_data\_rekeying setting 18](#_Toc134601655)

[Integration Monitoring 19](#_Toc134601656)

[❖ Show integrations in Snowflake 19](#_Toc134601657)

[❖ Monitor for security integrations that have been created or altered 19](#_Toc134601658)

[Backup and Recovery Monitoring 20](#_Toc134601659)

[❖ Monitor Data Retention 20](#_Toc134601660)

[Snowflake Parameter Monitoring 21](#_Toc134601661)

[❖ Monitor Account-Level Parameters 21](#_Toc134601662)

[❖ Monitor SAML and SSO Account-Level Parameters 21](#_Toc134601663)

[Query Monitoring 22](#_Toc134601664)

[❖ Monitor Query Details 22](#_Toc134601665)

[❖ Monitor Expensive Queries 22](#_Toc134601666)

[❖ Monitor Longest Running Queries 22](#_Toc134601667)

[Monitor Queries Queuing 22](#_Toc134601668)

[Storage Monitoring 23](#_Toc134601669)

[❖ Monitor Storage Summary 23](#_Toc134601670)

[❖ Monitor Storage Details 23](#_Toc134601671)

# Architecture of The Snowflake Monitoring

Since we can have many Snowflake accounts, the ideal method for monitoring account data is to collect the monitoring data into a single common location and point tools like SPLUNK and DATADOG to that location for monitoring. With this design pattern, we can add as many additional Snowflake accounts as we like without changing the monitoring infrastructure.

Graphical user interface

Description automatically generated

Figure : Monitoring Architecture

Within each Snowflake account, a standard task called *HOURLY\_MONITORING\_METRICS\_CAPTURE* runs each hour and copies data required for SPLUNK monitoring. Data from the ***SNOWFLAGE.ACCOUNT\_USAGE*** namespace is copied into the *SNOWFLAKE\_ACCOUNT\_REPORTING.ACCOUNT\_USAGE* namespace. Each table has the same name, and all the columns are the same except an additional column is added to each table that identifies the account the monitoring data is from. The database is shared back to the Organizational account through a Private Data Share.

In the Organizational Account, each shared account monitoring database is given a name *ACCOUNT\_USAGE\_REPORTING\_*<*account*> where <*account*> is the colloquial name for the source account. An hourly task called *HOURLY\_MONITORING\_METRICS\_COMBINE* will consolidate all the individual account databases into a single database called *ORG\_PROD.* SPLUNK will monitor this central database.

Finally, a reverse data share is made from the Organizational account back to all the individual accounts. This reverse share contains all the account usage spend in USD. The purpose to the reverse data share is to allow individual accounts the ability to monitor their spend. This data is not otherwise available for monitoring outside the Organizational Account.

# Setup Account Monitoring

In this section I explain how to setup a new account to upload its monitoring metrics to the organization account.

## Create The Metrics Reporting Database

This database and schema will hold all the metrics that will be shared with the Organization account.

// The account admin owns all the databases on their account  
**USE ROLE** **ACCOUNTADMIN**;

**CREATE OR REPLACE WAREHOUSE** **SNOWFLAKE\_ACCOUNT\_REPORTING**

WAREHOUSE\_SIZE = **SMALL**

MAX\_CLUSTER\_COUNT = **1**

MIN\_CLUSTER\_COUNT = **1**

SCALING\_POLICY = **ECONOMY**

AUTO\_SUSPEND = **120**

AUTO\_RESUME = **TRUE**

INITIALLY\_SUSPENDED = **TRUE**

COMMENT = 'Warehouse For Capturing Monitoring Metrics'

STATEMENT\_TIMEOUT\_IN\_SECONDS = **7200**;

// Create the database  
**CREATE OR REPLACE DATABASE** **SNOWFLAKE\_ACCOUNT\_REPORTING**

DATA\_RETENTION\_TIME\_IN\_DAYS = 30  
 COMMENT = 'Database for sharing account metrics to the organization account';

**CREATE OR REPLACE SCHEMA** **METRICS** WITH MANAGED ACCESS DATA\_RETENTION\_TIME\_IN\_DAYS = **30**

COMMENT = 'Schema for the Snowflake Loader process';

**CREATE OR REPLACE TABLE** **SNOWFLAKE\_ACCOUNT\_REPORTING.METRICS.LOGIN\_HISTORY**

(ACCOUNTNAME VARCHAR(),

EVENT\_ID INT,

EVENT\_TIMESTAMP TIMESTAMP,

EVENT\_TYPE VARCHAR(),

USER\_NAME VARCHAR(),

CLIENT\_IP VARCHAR(),

REPORTED\_CLIENT\_TYPE VARCHAR(),

REPORTED\_CLIENT\_VERSION VARCHAR(),

FIRST\_AUTHENTICATION\_FACTOR VARCHAR(),

SECOND\_AUTHENTICATION\_FACTOR VARCHAR(),

IS\_SUCCESS VARCHAR(),

ERROR\_CODE VARCHAR(),

ERROR\_MESSAGE VARCHAR(),

RELATED\_EVENT\_ID INT,

CONNECTION VARCHAR()

) COPY GRANTS;

**CREATE OR REPLACE TABLE** **SNOWFLAKE\_ACCOUNT\_REPORTING.METRICS.LOGIN\_HISTORY\_FAILS**

(ACCOUNTNAME VARCHAR(),

USER\_NAME VARCHAR(),

FAILED\_LOGINS VARCHAR(),

AVERAGE\_SECONDS\_BETWEEN\_LOGIN\_ATTEMPTS INT );

**CREATE OR REPLACE TABLE** **SNOWFLAKE\_ACCOUNT\_REPORTING.METRICS.QUERY\_HISTORY**

(ACCOUNTNAME VARCHAR(),

QUERY\_ID VARCHAR(),

QUERY\_TEXT VARCHAR(),

DATABASE\_NAME VARCHAR(),

SCHEMA\_NAME VARCHAR(),

QUERY\_TYPE VARCHAR(),

USER\_NAME VARCHAR(),

ROLE\_NAME VARCHAR(),

WAREHOUSE\_NAME VARCHAR(),

WAREHOUSE\_SIZE VARCHAR(),

EXECUTION\_STATUS VARCHAR(),

ERROR\_CODE VARCHAR(),

ERROR\_MESSAGE VARCHAR(),

START\_TIME TIMESTAMP,

END\_TIME TIMESTAMP,

TOTAL\_ELAPSED\_TIME INT,

BYTES\_SCANNED INT,

PERCENTAGE\_SCANNED\_FROM\_CACHE NUMBER(38,3),

BYTES\_WRITTEN INT,

ROWS\_PRODUCED INT,

ROWS\_INSERTED INT,

ROWS\_UPDATED INT,

ROWS\_DELETED INT,

ROWS\_UNLOADED INT,

BYTES\_DELETED INT,

PARTITIONS\_SCANNED INT,

PARTITIONS\_TOTAL INT,

EXECUTION\_TIME INT,

QUEUED\_PROVISIONING\_TIME INT,

QUEUED\_REPAIR\_TIME INT,

QUEUED\_OVERLOAD\_TIME INT,

TRANSACTION\_BLOCKED\_TIME INT,

CREDITS\_USED\_CLOUD\_SERVICES NUMBER(38,8))

COPY GRANTS;

**CREATE OR REPLACE TABLE** **SNOWFLAKE\_ACCOUNT\_REPORTING.METRICS.STORAGE\_USAGE**

(ACCOUNTNAME VARCHAR(),

USAGE\_DATE DATE,

STORAGE\_BYTES BIGINT,

STAGE\_BYTES BIGINT,

FAILSAFE\_BYTES BIGINT) COPY GRANTS;

**CREATE OR REPLACE TABLE** **SNOWFLAKE\_ACCOUNT\_REPORTING.METRICS.USERS**

(ACCOUNTNAME VARCHAR(),

NAME VARCHAR(),

CREATED\_ON TIMESTAMP,

DELETED\_ON TIMESTAMP,

LOGIN\_NAME VARCHAR(),

DISPLAY\_NAME VARCHAR(),

FIRST\_NAME VARCHAR(),

LAST\_NAME VARCHAR(),

EMAIL VARCHAR(),

HAS\_PASSWORD VARCHAR(),

COMMENT VARCHAR(),

DISABLED VARCHAR(),

EXT\_AUTHN\_DUO VARCHAR(),

LAST\_SUCCESS\_LOGIN TIMESTAMP,

HAS\_RSA\_PUBLIC\_KEY VARCHAR(),

PASSWORD\_LAST\_SET\_TIME TIMESTAMP,

OWNER VARCHAR()) COPY GRANTS;

**CREATE OR REPLACE TABLE** **SNOWFLAKE\_ACCOUNT\_REPORTING.METRICS.ACCESS\_HISTORY**

(ACCOUNTNAME VARCHAR(),

QUERY\_ID VARCHAR(),

QUERY\_START\_TIME TIMESTAMP,

USER\_NAME VARCHAR(),

DIRECT\_OBJECTS\_ACCESSED VARIANT,

BASE\_OBJECTS\_ACCESSED VARIANT,

OBJECTS\_MODIFIED VARIANT) COPY GRANTS;

**CREATE OR REPLACE TABLE** **SNOWFLAKE\_ACCOUNT\_REPORTING.METRICS.SCHEMATA**

(ACCOUNTNAME VARCHAR(),

SCHEMA\_ID VARCHAR(),

SCHEMA\_NAME VARCHAR(),

DATABASE\_NAME VARCHAR(),

SCHEMA\_OWNER VARCHAR(),

RETENTION\_TIME INT,

IS\_TRANSIENT VARCHAR(),

IS\_MANAGED\_ACCESS VARCHAR(),

CREATED TIMESTAMP,

LAST\_ALTERED TIMESTAMP,

DELETED TIMESTAMP) COPY GRANTS;

**CREATE OR REPLACE TABLE** **SNOWFLAKE\_ACCOUNT\_REPORTING.METRICS.DATABASES**

(ACCOUNTNAME VARCHAR(),

DATABASE\_NAME VARCHAR(),

DATABASE\_OWNER VARCHAR(),

CREATED TIMESTAMP,

LAST\_ALTERED TIMESTAMP,

DELETED TIMESTAMP,

RETENTION\_TIME INT) COPY GRANTS;

**CREATE OR REPLACE TABLE** **SNOWFLAKE\_ACCOUNT\_REPORTING.METRICS.TABLES**

(ACCOUNTNAME VARCHAR(),

TABLE\_NAME VARCHAR(),

TABLE\_SCHEMA VARCHAR(),

DATABASE\_NAME VARCHAR(),

TABLE\_OWNER VARCHAR(),

TABLE\_TYPE VARCHAR(),

ROW\_COUNT BIGINT,

BYTES BIGINT,

RETENTION\_TIME BIGINT,

CREATED TIMESTAMP,

LAST\_ALTERED TIMESTAMP,

DELETED TIMESTAMP,

AUTO\_CLUSTERING\_ON VARCHAR() )

COPY GRANTS;

**CREATE OR REPLACE TABLE** **SNOWFLAKE\_ACCOUNT\_REPORTING.METRICS.TAG\_REFERENCES**

(ACCOUNTNAME VARCHAR(),

TAG\_DATABASE VARCHAR(),

TAG\_SCHEMA VARCHAR(),

TAG\_NAME VARCHAR(),

TAG\_VALUE VARCHAR(),

OBJECT\_DATABASE VARCHAR(),

OBJECT\_SCHEMA VARCHAR(),

OBJECT\_NAME VARCHAR(),

OBJECT\_DELETED TIMESTAMP,

DOMAIN VARCHAR(),

COLUMN\_NAME VARCHAR()) COPY GRANTS;

**CREATE OR REPLACE TABLE** **SNOWFLAKE\_ACCOUNT\_REPORTING.METRICS.STORAGE\_INTEGRATIONS**

(ACCOUNTNAME VARCHAR(),

NAME VARCHAR(),

TYPE VARCHAR(),

CATEGORY VARCHAR(),

ENABLED VARCHAR(),

PROPERTY VARCHAR(),

PROPERTY\_TYPE VARCHAR(),

PROPERTY\_VALUE VARCHAR(),

PROPERTY\_DEFAULT VARCHAR() )

COPY GRANTS;

**CREATE OR REPLACE TABLE** **SNOWFLAKE\_ACCOUNT\_REPORTING.METRICS.NETWORK\_POLICIES**

(ACCOUNTNAME VARCHAR(),

NAME VARCHAR(),

CREATED\_ON TIMESTAMP,

POLICY\_TYPE VARCHAR(),

VALUE VARCHAR() )

COPY GRANTS;

**CREATE OR REPLACE TABLE** **SNOWFLAKE\_ACCOUNT\_REPORTING.METRICS.ACCOUNT\_PARAMETERS**

(ACCOUNTNAME VARCHAR(),

KEY VARCHAR(),

VALUE VARCHAR(),

TYPE VARCHAR() ) COPY GRANTS;

## Create The Metrics Collection Task

This task runs at the top of every hour.

**USE ROLE** ACCOUNTADMIN;

**USE DATABASE** **SNOWFLAKE\_ACCOUNT\_REPORTING**;

**USE SCHEMA** METRICS;

[**CREATE OR REPLACE TASK**](https://docs.snowflake.com/en/sql-reference/sql/create-task.html) **HOURLY\_MONITORING\_METRICS\_CAPTURE**

WAREHOUSE = **SNOWFLAKE\_ACCOUNT\_REPORTING**

SCHEDULE = 'USING CRON 0 \* \* \* \* America/Chicago'

COMMENT = 'Task to capture the metrics for monitoring'

AS

EXECUTE IMMEDIATE $$

DECLARE

D1 VARCHAR();

D2 VARCHAR();

D3 VARCHAR();

D4 VARCHAR();

BEGIN

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\* Truncate all the stage tables

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TRUNCATE TABLE SNOWFLAKE\_ACCOUNT\_REPORTING.METRICS.LOGIN\_HISTORY;

TRUNCATE TABLE SNOWFLAKE\_ACCOUNT\_REPORTING.METRICS.LOGIN\_HISTORY\_FAILS;

TRUNCATE TABLE SNOWFLAKE\_ACCOUNT\_REPORTING.METRICS.QUERY\_HISTORY;

TRUNCATE TABLE SNOWFLAKE\_ACCOUNT\_REPORTING.METRICS.STORAGE\_USAGE;

TRUNCATE TABLE SNOWFLAKE\_ACCOUNT\_REPORTING.METRICS.USERS;

TRUNCATE TABLE SNOWFLAKE\_ACCOUNT\_REPORTING.METRICS.ACCESS\_HISTORY;

TRUNCATE TABLE SNOWFLAKE\_ACCOUNT\_REPORTING.METRICS.SCHEMATA;

TRUNCATE TABLE SNOWFLAKE\_ACCOUNT\_REPORTING.METRICS.DATABASES;

TRUNCATE TABLE SNOWFLAKE\_ACCOUNT\_REPORTING.METRICS.TABLES;

TRUNCATE TABLE SNOWFLAKE\_ACCOUNT\_REPORTING.METRICS.TAG\_REFERENCES;

TRUNCATE TABLE SNOWFLAKE\_ACCOUNT\_REPORTING.METRICS.STORAGE\_INTEGRATIONS;

TRUNCATE TABLE SNOWFLAKE\_ACCOUNT\_REPORTING.METRICS.NETWORK\_POLICIES;

TRUNCATE TABLE SNOWFLAKE\_ACCOUNT\_REPORTING.METRICS.ACCOUNT\_PARAMETERS;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\* Login History

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

INSERT INTO SNOWFLAKE\_ACCOUNT\_REPORTING.METRICS.LOGIN\_HISTORY

SELECT '<ACCOUNTNAME>',EVENT\_ID, EVENT\_TIMESTAMP, EVENT\_TYPE, USER\_NAME, CLIENT\_IP, REPORTED\_CLIENT\_TYPE, REPORTED\_CLIENT\_VERSION, FIRST\_AUTHENTICATION\_FACTOR,

SECOND\_AUTHENTICATION\_FACTOR, IS\_SUCCESS, ERROR\_CODE, ERROR\_MESSAGE, RELATED\_EVENT\_ID, CONNECTION

FROM SNOWFLAKE.ACCOUNT\_USAGE.LOGIN\_HISTORY

WHERE TIMESTAMPDIFF(minute,EVENT\_TIMESTAMP,current\_timestamp()) <= 60;

SELECT \* FROM SNOWFLAKE\_ACCOUNT\_REPORTING.METRICS.LOGIN\_HISTORY;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\* Login History Fails

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

INSERT INTO SNOWFLAKE\_ACCOUNT\_REPORTING.METRICS.LOGIN\_HISTORY\_FAILS

SELECT ACCOUNTNAME, USER\_NAME, COUNT(\*) AS FAILED\_LOGINS,

AVG(SECONDS\_BETWEEN\_LOGIN\_ATTEMPTS) AS AVERAGE\_SECONDS\_BETWEEN\_LOGIN\_ATTEMPTS

FROM ( SELECT '<ACCOUNTNAME>' AS ACCOUNTNAME, USER\_NAME,

TIMEDIFF(SECONDS, EVENT\_TIMESTAMP, LEAD(EVENT\_TIMESTAMP)

OVER ( PARTITION BY USER\_NAME ORDER BY EVENT\_TIMESTAMP)) AS SECONDS\_BETWEEN\_LOGIN\_ATTEMPTS

FROM SNOWFLAKE.ACCOUNT\_USAGE.LOGIN\_HISTORY

WHERE EVENT\_TIMESTAMP > DATE\_TRUNC( MONTH , CURRENT\_DATE )

AND IS\_SUCCESS = 'NO'

)

GROUP BY 1,2

HAVING COUNT(\*) > 3 //NUMBER OF FAILURES - ADJUST AS NEEDED

AND AVG(SECONDS\_BETWEEN\_LOGIN\_ATTEMPTS) < 5 //AVERAGE SECONDS BETWEEN FAILURES - ADJUST AS NEEDED

ORDER BY AVG(SECONDS\_BETWEEN\_LOGIN\_ATTEMPTS) DESC ;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\* Query History

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

INSERT INTO SNOWFLAKE\_ACCOUNT\_REPORTING.METRICS.QUERY\_HISTORY

SELECT '<ACCOUNTNAME>', QUERY\_ID, QUERY\_TEXT, DATABASE\_NAME, SCHEMA\_NAME, QUERY\_TYPE, USER\_NAME, ROLE\_NAME, WAREHOUSE\_NAME, WAREHOUSE\_SIZE,

EXECUTION\_STATUS, ERROR\_CODE, ERROR\_MESSAGE, START\_TIME, END\_TIME, TOTAL\_ELAPSED\_TIME, BYTES\_SCANNED,

PERCENTAGE\_SCANNED\_FROM\_CACHE, BYTES\_WRITTEN, ROWS\_PRODUCED, ROWS\_INSERTED, ROWS\_UPDATED, ROWS\_DELETED, ROWS\_UNLOADED,

BYTES\_DELETED, PARTITIONS\_SCANNED, PARTITIONS\_TOTAL, EXECUTION\_TIME, QUEUED\_PROVISIONING\_TIME, QUEUED\_REPAIR\_TIME, QUEUED\_OVERLOAD\_TIME,

TRANSACTION\_BLOCKED\_TIME, CREDITS\_USED\_CLOUD\_SERVICES

FROM SNOWFLAKE.ACCOUNT\_USAGE.QUERY\_HISTORY

WHERE TIMESTAMPDIFF(minute,END\_TIME,current\_timestamp()) <= 60;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\* Storage Usage

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

INSERT INTO SNOWFLAKE\_ACCOUNT\_REPORTING.METRICS.STORAGE\_USAGE

SELECT '<ACCOUNTNAME>', USAGE\_DATE, STORAGE\_BYTES, STAGE\_BYTES, FAILSAFE\_BYTES

FROM SNOWFLAKE.ACCOUNT\_USAGE.STORAGE\_USAGE

WHERE TIMESTAMPDIFF(day,USAGE\_DATE, current\_date()) < 2;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\* Users

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

INSERT INTO SNOWFLAKE\_ACCOUNT\_REPORTING.METRICS.USERS

SELECT '<ACCOUNTNAME>', NAME, CREATED\_ON, DELETED\_ON, LOGIN\_NAME, DISPLAY\_NAME, FIRST\_NAME, LAST\_NAME, EMAIL, HAS\_PASSWORD, COMMENT, DISABLED,

EXT\_AUTHN\_DUO, LAST\_SUCCESS\_LOGIN, HAS\_RSA\_PUBLIC\_KEY, PASSWORD\_LAST\_SET\_TIME, OWNER

FROM SNOWFLAKE.ACCOUNT\_USAGE.USERS

WHERE DELETED\_ON IS NULL

AND (TIMESTAMPDIFF(minute,CREATED\_ON, current\_timestamp()) <= 60 OR

TIMESTAMPDIFF(minute,DELETED\_ON, current\_timestamp()) <= 60

);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\* Access History

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

INSERT INTO SNOWFLAKE\_ACCOUNT\_REPORTING.METRICS.ACCESS\_HISTORY

SELECT '<ACCOUNTNAME>', QUERY\_ID, QUERY\_START\_TIME, USER\_NAME, DIRECT\_OBJECTS\_ACCESSED, BASE\_OBJECTS\_ACCESSED, OBJECTS\_MODIFIED

FROM SNOWFLAKE.ACCOUNT\_USAGE.ACCESS\_HISTORY

WHERE TIMESTAMPDIFF(minute,QUERY\_START\_TIME, current\_timestamp()) <= 60;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\* Schemas

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

INSERT INTO SNOWFLAKE\_ACCOUNT\_REPORTING.METRICS.SCHEMATA

SELECT '<ACCOUNTNAME>', SCHEMA\_ID, SCHEMA\_NAME, CATALOG\_NAME, SCHEMA\_OWNER, RETENTION\_TIME, IS\_TRANSIENT, IS\_MANAGED\_ACCESS,

CREATED, LAST\_ALTERED, DELETED

FROM SNOWFLAKE.ACCOUNT\_USAGE.SCHEMATA

WHERE TIMESTAMPDIFF(minute,CREATED, current\_timestamp()) <= 60

OR TIMESTAMPDIFF(minute,LAST\_ALTERED, current\_timestamp()) <= 60

OR TIMESTAMPDIFF(minute,DELETED, current\_timestamp()) <= 60;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\* Databases

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

INSERT INTO SNOWFLAKE\_ACCOUNT\_REPORTING.METRICS.DATABASES

SELECT '<ACCOUNTNAME>', DATABASE\_NAME, DATABASE\_OWNER, CREATED, LAST\_ALTERED, DELETED, RETENTION\_TIME

FROM SNOWFLAKE.ACCOUNT\_USAGE.DATABASES

WHERE DATABASE\_NAME <> 'SNOWFLAKE'

AND (TIMESTAMPDIFF(minute,CREATED, current\_timestamp()) <= 60 OR

TIMESTAMPDIFF(minute,LAST\_ALTERED, current\_timestamp()) <= 60 OR

TIMESTAMPDIFF(minute,DELETED, current\_timestamp()) <= 60

);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\* Tag References

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

INSERT INTO SNOWFLAKE\_ACCOUNT\_REPORTING.METRICS.TAG\_REFERENCES

SELECT '<ACCOUNTNAME>',TAG\_DATABASE, TAG\_SCHEMA, TAG\_NAME, TAG\_VALUE, OBJECT\_DATABASE, OBJECT\_SCHEMA, OBJECT\_NAME, OBJECT\_DELETED, DOMAIN, COLUMN\_NAME

FROM SNOWFLAKE.ACCOUNT\_USAGE.TAG\_REFERENCES;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\* Capture STORAGE INTEGRATIONS

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

SHOW INTEGRATIONS;

LET C1 CURSOR FOR SELECT "name" AS NAME, "type" as TYPE, "category" as CATEGORY, "enabled" as ENABLED

FROM TABLE(RESULT\_SCAN())

WHERE TIMESTAMPDIFF(minute,"created\_on", current\_timestamp()) <= 60;

FOR RECORD IN C1 DO

LET F1 VARCHAR() := RECORD.NAME;

LET F2 VARCHAR() := RECORD.TYPE;

LET F3 VARCHAR() := RECORD.CATEGORY;

LET F4 VARCHAR() := RECORD.ENABLED;

let select\_statement := 'DESC INTEGRATION ' || F1;

execute immediate :select\_statement ;

LET C2 CURSOR FOR SELECT "property" AS PROPERTY, "property\_type" as PROPERTY\_TYPE, "property\_value" as PROPERTY\_VALUE, "property\_default" as PROPERTY\_DEFAULT

FROM TABLE(RESULT\_SCAN()) ;

FOR RECORD2 IN C2 DO

D1 := RECORD2.PROPERTY;

D2 := RECORD2.PROPERTY\_TYPE;

D3 := RECORD2.PROPERTY\_VALUE;

D4 := RECORD2.PROPERTY\_DEFAULT;

INSERT INTO SNOWFLAKE\_ACCOUNT\_REPORTING.METRICS.STORAGE\_INTEGRATIONS VALUES ('<ACCOUNTNAME>',:F1, :F2, :F3, :F4, :D1, :D2, :D3, :D4);

END FOR;

END FOR;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\* Capture NETWORK POLICIES

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

SHOW NETWORK POLICIES;

LET C3 CURSOR FOR SELECT "created\_on" AS CREATED\_ON, "name" as NAME, "comment" as COMMENT,

"entries\_in\_allowed\_ip\_list" as ENTRIES\_IN\_ALLOWED\_IP\_LIST,

"entries\_in\_blocked\_ip\_list" as ENTRIES\_IN\_BLOCKED\_IP\_LIST

FROM TABLE(RESULT\_SCAN() )

WHERE TIMESTAMPDIFF(minute,"created\_on", current\_timestamp()) <= 60;

FOR RECORD3 IN C3 DO

LET F1 TIMESTAMP := RECORD3.CREATED\_ON;

LET F2 VARCHAR() := RECORD3.NAME;

let select\_statement := 'DESC NETWORK POLICY ' || F2;

execute immediate :select\_statement ;

LET C4 CURSOR FOR SELECT "name" AS NAME, "value" as VALUE

FROM TABLE(RESULT\_SCAN()) ;

FOR RECORD4 IN C4 DO

D1 := RECORD4.NAME;

D2 := RECORD4.VALUE;

INSERT INTO SNOWFLAKE\_ACCOUNT\_REPORTING.METRICS.NETWORK\_POLICIES VALUES ('<ACCOUNTNAME>',:F2, :F1, :D1, :D2);

END FOR;

END FOR;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\* Capture Account Parameters

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

SHOW PARAMETERS;

LET C5 CURSOR FOR SELECT "key" AS KEY, "value" as VALUE, "default" as DEFAULT, "level" as LEVEL,

"description" AS DESCRIPTION, "type" AS TYPE

FROM TABLE(RESULT\_SCAN()) ;

FOR RECORD5 IN C5 DO

LET F1 VARCHAR() := RECORD5.KEY;

LET F2 VARCHAR() := RECORD5.VALUE;

LET F3 VARCHAR() := RECORD5.DEFAULT;

LET F4 VARCHAR() := RECORD5.LEVEL;

LET F5 VARCHAR() := RECORD5.DESCRIPTION;

LET F6 VARCHAR() := RECORD5.TYPE;

INSERT INTO SNOWFLAKE\_ACCOUNT\_REPORTING.METRICS.ACCOUNT\_PARAMETERS VALUES ('<ACCOUNTNAME>', :F1, :F2, :F6);

END FOR;

RETURN 'Done';

END;

$$

;

//\* Tasks are initially created suspended. Set to resume

**ALTER TASK** **HOURLY\_MONITORING\_METRICS\_CAPTURE** RESUME;

**EXECUTE TASK HOURLY\_MONITORING\_METRICS\_CAPTURE**;

## Create The Metrics Combiner Task

This task runs at the organization account to combine all the metrics from other accounts.

**USE ROLE** **ACCOUNTADMIN**;

**USE DATABASE** **ORG\_PROD**;

**USE SCHEMA** **METRICS**;

**CREATE OR REPLACE WAREHOUSE** **SNOWFLAKE\_ACCOUNT\_REPORTING**

WAREHOUSE\_SIZE = **SMALL**

MAX\_CLUSTER\_COUNT = **1**

MIN\_CLUSTER\_COUNT = **1**

SCALING\_POLICY = **ECONOMY**

AUTO\_SUSPEND = **120**

AUTO\_RESUME = **TRUE**

INITIALLY\_SUSPENDED = **TRUE**

COMMENT = 'Warehouse For Capturing Monitoring Metrics'

STATEMENT\_TIMEOUT\_IN\_SECONDS = **7200**;

[**CREATE OR REPLACE TASK**](https://docs.snowflake.com/en/sql-reference/sql/create-task.html) **HOURLY\_MONITORING\_METRICS\_COMBINE**

WAREHOUSE = **SNOWFLAKE\_ACCOUNT\_REPORTING**

SCHEDULE = 'USING CRON 30 \* \* \* \* America/Chicago'

COMMENT = 'Task to combine the metrics for monitoring'

AS

EXECUTE IMMEDIATE $$

DECLARE

D1 VARCHAR();

BEGIN

SHOW DATABASES;

LET C1 CURSOR FOR SELECT "created\_on" AS CREATED\_ON, "name" AS NAME, "is\_default" as IS\_DEFAULT, "origin" as ORIGIN

FROM TABLE(RESULT\_SCAN())

WHERE "name" LIKE 'ACCOUNT\_USAGE\_REPORTING%';

FOR RECORD IN C1 DO

LET F1 VARCHAR() := RECORD.NAME;

LET SELECT\_STATEMENT := 'USE DATABASE ' || F1;

EXECUTE IMMEDIATE :SELECT\_STATEMENT ;

DELETE FROM ORG\_PROD.METRICS.TAG\_REFERENCES T1

USING METRICS.TAG\_REFERENCES T2

WHERE T1.TAG\_DATABASE = T2.TAG\_DATABASE

AND T1.TAG\_SCHEMA = T2.TAG\_SCHEMA

AND T1.TAG\_NAME = T2.TAG\_NAME

AND T1.OBJECT\_DATABASE = T2.TAG\_DATABASE

AND T1.OBJECT\_SCHEMA = T2.TAG\_SCHEMA

AND T1.OBJECT\_NAME = T2.TAG\_NAME;

INSERT INTO ORG\_PROD.METRICS.LOGIN\_HISTORY SELECT \* FROM METRICS.LOGIN\_HISTORY ;

INSERT INTO ORG\_PROD.METRICS.LOGIN\_HISTORY\_FAILS SELECT \* FROM METRICS.LOGIN\_HISTORY\_FAILS;

INSERT INTO ORG\_PROD.METRICS.QUERY\_HISTORY SELECT \* FROM METRICS.QUERY\_HISTORY;

INSERT INTO ORG\_PROD.METRICS.STORAGE\_USAGE SELECT \* FROM METRICS.STORAGE\_USAGE;

INSERT INTO ORG\_PROD.METRICS.USERS SELECT \* FROM METRICS.USERS;

INSERT INTO ORG\_PROD.METRICS.ACCESS\_HISTORY SELECT \* FROM METRICS.ACCESS\_HISTORY;

INSERT INTO ORG\_PROD.METRICS.SCHEMATA SELECT \* FROM METRICS.SCHEMATA;

INSERT INTO ORG\_PROD.METRICS.DATABASES SELECT \* FROM METRICS.DATABASES;

INSERT INTO ORG\_PROD.METRICS.TABLES SELECT \* FROM METRICS.TABLES;

INSERT INTO ORG\_PROD.METRICS.TAG\_REFERENCES SELECT \* FROM METRICS.TAG\_REFERENCES ;

INSERT INTO ORG\_PROD.METRICS.STORAGE\_INTEGRATIONS SELECT \* FROM METRICS.STORAGE\_INTEGRATIONS;

INSERT INTO ORG\_PROD.METRICS.NETWORK\_POLICIES SELECT \* FROM METRICS.NETWORK\_POLICIES;

INSERT INTO ORG\_PROD.METRICS.ACCOUNT\_PARAMETERS SELECT \* FROM METRICS.ACCOUNT\_PARAMETERS;

END FOR;

RETURN 'Done';

END;

$$

;

//\* Tasks are initially created suspended. Set to resume

**ALTER TASK** **HOURLY\_MONITORING\_METRICS\_COMBINE** RESUME;

# Security Monitoring

## ❖ **Verify connections are logging in from specific networks**

Verify that connections expected to log in from specific networks are doing so.

**select** event\_timestamp, user\_name, client\_ip, reported\_client\_type

reported\_client\_version, first\_authentication\_factor,

second\_authentication\_factor

from ORG\_PROD.METRICS.login\_history

where error\_message = **'INCOMING\_IP\_BLOCKED'**

AND TIMESTAMPDIFF(minute,EVENT\_TIMESTAMP,current\_timestamp()) <= 60;

orderby event\_timestamp desc ;

## ❖ **Monitor Grants Not Made By SECURITYADMIN Role**

Snowflake recommends using a designated role for all user management tasks. Monitor that all user and role grants originate from this role, and that this role is only granted to appropriate users.

**select** user\_name, role\_name, query\_text

from ORG\_PROD.METRICS.query\_history

where execution\_status = **'SUCCESS'**

and query\_type = **'GRANT'**

and USER\_NAME <> **'SECURITYADMIN'**

AND TIMESTAMPDIFF(minute,END\_TIME,current\_timestamp()) <= 60

order by end\_time desc ;

## ❖ **Monitor Built-in Admin Roles Activity**

Watch the logs for all instances of a user using the default Snowflake admin roles to ensure their use is appropriate. The default admin roles are:

* ACCOUNTADMIN
* SECURITYADMIN
* USERADMIN
* SYSADMIN

**select** end\_time, query\_type, query\_text, user\_name, role\_name

from ORG\_PROD.METRICS.query\_history

where execution\_status = **'SUCCESS'**

and role\_name in ( **'ACCOUNTADMIN' , 'SECURITYADMIN' , 'USERADMIN' , 'SYSADMIN', 'ORGADMIN'** )

AND TIMESTAMPDIFF(minute,END\_TIME,current\_timestamp()) <= 60

order by end\_time desc ;

## ❖ **Monitor Users Created**

Monitor QUERY\_HISTORY for unusual CREATE or REPLACE user activity.

**select** end\_time, query\_type, query\_text, user\_name, role\_name

from ORG\_PROD.METRICS.query\_history

where execution\_status = **'SUCCESS'**

and query\_type = **'CREATE\_USER'**

and query\_text ilike **'%create%user%'**

AND TIMESTAMPDIFF(minute,END\_TIME,current\_timestamp()) <= 60

order by end\_time desc ;

## ❖ **Monitor User Modification**

Monitor QUERY\_HISTORY for ALTER user activity, for example to flag non-SSO authentication method grants.

**select** end\_time, query\_type, query\_text, user\_name, role\_name

from ORG\_PROD.METRICS.query\_history

where execution\_status = **'SUCCESS'**

and query\_type = **'ALTER\_USER'**

and query\_text ilike **'%alter user%set rsa\_public\_key%'**

AND TIMESTAMPDIFF(minute,END\_TIME,current\_timestamp()) <= 60

order by end\_time desc ;

## ❖ **Monitor How Users Are Authenticated**

The following sample statement shows the number of times each user authenticated and the

authentication method they used.

--Each User and their most frequently used authentication methods

**select** user\_name, first\_authentication\_factor, second\_authentication\_factor, count(\*)

from ORG\_PROD.METRICS.login\_history

where is\_success = **'YES'**

AND TIMESTAMPDIFF(minute,EVENT\_TIMESTAMP,current\_timestamp()) <= 60

group by user\_name, first\_authentication\_factor, second\_authentication\_factor

order by user\_name, count(\*) desc ;

## ❖ Monitor Users Who Used SSO Before And Now Using Other Authentication Methods

After users successfully authenticate using SSO, they should not be using other methods.

--login events for users who have sso but are not using it

**select** sso.\*, event\_timestamp, user\_name, first\_authentication\_factor, second\_authentication\_factor,

client\_ip, reported\_client\_type, reported\_client\_version

from ORG\_PROD.METRICS.login\_history l

join (**select** user\_name user\_has\_used\_sso, min(event\_timestamp) firstsso

from ORG\_PROD.METRICS.login\_history

where first\_authentication\_factor in ( **'SAML2\_ASSERTION' , 'OAUTH\_ACCESS\_TOKEN'** )

group by user\_name) sso

on sso.user\_has\_used\_sso = l.user\_name

where first\_authentication\_factor not in ( **'SAML2\_ASSERTION' , 'OAUTH\_ACCESS\_TOKEN'**)

AND TIMESTAMPDIFF(minute,EVENT\_TIMESTAMP,current\_timestamp()) <= 60

and l.event\_timestamp > firstsso

order by l.event\_timestamp desc ;

## ❖ **Monitor For Key Pair Authentication**

Monitor the use of key pair authentication by querying login attempts.

**select** event\_timestamp, user\_name, client\_ip, reported\_client\_type, reported\_client\_version,

first\_authentication\_factor, Second\_authentication\_factor

from ORG\_PROD.METRICS.login\_history

where first\_authentication\_factor = **'RSA\_KEYPAIR'**

AND TIMESTAMPDIFF(minute,EVENT\_TIMESTAMP,current\_timestamp()) <= 60

order by event\_timestamp desc ;

## ❖ Monitor If Key Pair Authentication Users Also Configured For Other Methods

Users who have key pair authentication should be using it exclusively.

--Key Pair Users who also have passwords

**select** \*

from ORG\_PROD.METRICS.users

where has\_rsa\_public\_key = 'true'

and has\_password = **'true'** ;

--Key Pair Users who authenticated in a different way, and how many times

**select** u.name, first\_authentication\_factor, second\_authentication\_factor, count(\*)

from ORG\_PROD.METRICS.login\_history l

join ORG\_PROD.METRICS.users u

on l.user\_name = u.name

and has\_rsa\_public\_key = **'true'**

where is\_success = **'YES'**

AND TIMESTAMPDIFF(minute,EVENT\_TIMESTAMP,current\_timestamp()) <= 60

and first\_authentication\_factor != **'RSA\_KEYPAIR'**

group by name, first\_authentication\_factor, second\_authentication\_factor

order by count(\*) desc ;

## ❖ **Monitor For SCIM Access Token Creation**

SCIM access tokens have a six-month lifespan so it is important to track how many were generated.

**select** \*

from ORG\_PROD.METRICS.query\_history

where execution\_status = 'SUCCESS'

AND TIMESTAMPDIFF(minute,END\_TIME,current\_timestamp()) <= 60

and query\_text ilike **'%system$generate\_scim\_access\_token%'** ;

## ❖ **Monitor Failed Login Attempts**

The following approach returns results based on either the FAILED\_LOGINS count or the login failure rate ( AVERAGE\_SECONDS\_BETWEEN\_LOGIN\_ATTEMPTS ). This approach helps distinguish a brute force attack from a human who is struggling to remember their password. There are inline comments on how to adjust the query to limit results.

**select** user\_name, count(\*) as failed\_logins,

avg(seconds\_between\_login\_attempts) as average\_seconds\_between\_login\_attempts

from ( select user\_name, timediff(seconds, event\_timestamp, lead(event\_timestamp)

over ( partition by user\_name order by event\_timestamp)) as seconds\_between\_login\_attempts

from ORG\_PROD.METRICS.login\_history

where event\_timestamp > date\_trunc( month , current\_date )

and is\_success = **'NO'**

)

group by 1

having count(\*) > **3** //number of failures - adjust as needed

and avg(seconds\_between\_login\_attempts) < **5** //average seconds between failures - adjust as needed

order by avg(seconds\_between\_login\_attempts) desc ;

## ❖ **Monitor Snowflake MFA**

Monitor the SECOND\_AUTHENTICATION\_FACTOR field in LOGIN\_HISTORY to get per-user

Snowflake MFA login information.

**select** first\_authentication\_factor, second\_authentication\_factor, count(\*)

from ORG\_PROD.METRICS.login\_history

where is\_success = **'YES'**

AND TIMESTAMPDIFF(minute,EVENT\_TIMESTAMP,current\_timestamp()) <= 60

group by first\_authentication\_factor,second\_authentication\_factor

order by count(\*) desc ;

-- Most recent logins without MFA

**select** event\_timestamp, user\_name, client\_ip, reported\_client\_type, reported\_client\_version,

first\_authentication\_factor, Second\_authentication\_factor

from ORG\_PROD.METRICS.login\_history

where first\_authentication\_factor = **'PASSWORD'**

AND TIMESTAMPDIFF(minute,EVENT\_TIMESTAMP,current\_timestamp()) <= 60

and second\_authentication\_factor is null

order by event\_timestamp desc ;

## ❖ **Monitor For Re-Enabling Disabled Users**

Re-enabling a user is a rare event that could be a threat. Monitor QUERY\_HISTORY for ALTER USER activity.

**select** end\_time, query\_type, query\_text, user\_name, role\_name

from ORG\_PROD.METRICS.query\_history

where execution\_status = **'SUCCESS'**

and query\_type = **'ALTER\_USER'**

and query\_text ilike **'%alter user%set disabled = false%'**

AND TIMESTAMPDIFF(minute,END\_TIME,current\_timestamp()) <= 60

order by end\_time desc ;

## ❖ **Monitor for Enabled Plaintext User Passwords**

Monitor QUERY\_HISTORY for password changes.

**select** end\_time, query\_type, query\_text, user\_name, role\_name

from ORG\_PROD.METRICS.query\_history

where execution\_status = **'SUCCESS'**

and query\_type = **'ALTER\_USER'**

and query\_text ilike **'%alter user%set password%'**

AND TIMESTAMPDIFF(minute,END\_TIME,current\_timestamp()) <= 60

order by end\_time desc ;

## ❖ **Monitor SCIM user-provisioning API calls**

Applicable if SCIM user-provisioning via the REST API is configured. Monitor SCIM API calls to ensure API requests comply with policy.

**select** \*

from table (snowflake.information\_schema.rest\_event\_history (

**'scim'**,dateadd(**'minutes',- 5**,current\_timestamp ()),current\_timestamp (),**200** )

)

order by event\_timestamp desc ;

## ❖ Monitor **Highly Privileged Database Objects**

Monitor the following global privileges in QUERY\_HISTORY because they involve elevated privileges in your Snowflake Account:

* create user
* create role
* manage grants
* create integration
* create share
* create account
* monitor usage
* OWNERSHIP

**select** user\_name, role\_name, query\_text

from ORG\_PROD.METRICS.query\_history

where execution\_status = **'SUCCESS'**

and query\_type = **'GRANT'**

AND TIMESTAMPDIFF(minute,END\_TIME,current\_timestamp()) <= 60

and (

query\_text ilike **'%create user%'** or

query\_text ilike **'%create role%'** or

query\_text ilike **'%manage grants%'** or

query\_text ilike **'%create integration%'** or

query\_text ilike **'%create share%'** or

query\_text ilike **'%create account%'** or

query\_text ilike **'%monitor usage%'** or

query\_text ilike **'%ownership%'**

)

order by end\_time desc ;

## ❖ **Monitor ACCOUNTADMIN role grants**

The Snowflake role ACCOUNTADMIN should be closely monitored. Monitor QUERY\_HISTORY for GRANT ROLE .

**select** user\_name, role\_name, query\_text

from ORG\_PROD.METRICS.query\_history

where execution\_status = 'SUCCESS'

and query\_type = **'GRANT'**

and query\_text ilike **'%grant%accountadmin%to%'**

AND TIMESTAMPDIFF(minute,END\_TIME,current\_timestamp()) <= 60

order by end\_time desc ;

## ❖ **Monitor grants to the public role**

The *public* role should have the fewest possible grants. Every user in a Snowflake account has public. Monitor QUERY\_HISTORY for alterations or grants to the public role.

**select** user\_name, role\_name, query\_text, end\_time

from ORG\_PROD.METRICS.query\_history

where execution\_status = **'SUCCESS'**

and query\_type = **'GRANT'**

and query\_text ilike **'%to%public%'**

AND TIMESTAMPDIFF(minute,END\_TIME,current\_timestamp()) <= 60

order by end\_time desc ;

# Operational Monitoring

## ❖ **Monitor Network Policies**

Monitor changes to Network Policies and associated objects.

**select** end\_time, query\_type, query\_text, user\_name, role\_name

from ORG\_PROD.METRICS.query\_history

where execution\_status = **'SUCCESS'**

**AND** query\_type <> **'SELECT'**

AND TIMESTAMPDIFF(minute,END\_TIME,current\_timestamp()) <= 60

and query\_type in ( **'CREATE\_NETWORK\_POLICY' , 'ALTER\_NETWORK\_POLICY' , 'DROP\_NETWORK\_POLICY'** )

or (

query\_text ilike '**% set network\_policy%**'

or query\_text ilike '**% unset network\_policy%**'

)

order by end\_time desc ;

## ❖ **Show Network and User-level Policy Settings**

Use these queries to verify that allow lists and block lists match expected configurations.

Use the SHOW NETWORK POLICIES command to list all network policies defined in the system.

**show network policies**;

**desc network policy** <*name*>;

**show** **parameters** like **'network\_policy'** in account;

**show** **parameters** like **'network\_policy'** in user <*username*>;

**show integrations;**

**desc** **integration** <*integration\_name*>;

## ❖ **Monitor for long sessions**

Monitor for client applications that are keeping sessions open longer than desired by policy. Also look for client applications that have the CLIENT\_SESSION\_KEEP\_ALIVE connection parameter set to true, which allows the session to NEVER expire if the connection is left open.

**select** end\_time, query\_type, query\_text, user\_name, role\_name

from ORG\_PROD.METRICS.query\_history

where execution\_status = **'SUCCESS'**

and query\_type = **'ALTER\_SESSION'**

and query\_text ilike **'%client\_session%'**

AND TIMESTAMPDIFF(minute,END\_TIME,current\_timestamp()) <= 60

order by end\_time desc ;

# [PII Monitoring](https://confluence.hyattdev.com/display/AD/9.4.4.2+PI+Monitoring)

## ❖ **Monitor Use of PII Roles**

Since one of the ways for users to access PI data is by using a PI role, we can monitor when the users either select the role in the UI or run the command “**USE ROLE** <***role\_name***>”.  This will show when the user successfully uses that role and when a user tries to use a role that they are not assigned to.  It will show the failure, saying the role is not assigned to the executing user.

**SELECT** USER\_NAME, SESSION\_ID, START\_TIME, QUERY\_ID, QUERY\_TYPE,

QUERY\_TEXT, EXECUTION\_STATUS, ERROR\_MESSAGE

FROM ORG\_PROD.METRICS.QUERY\_HISTORY

WHERE QUERY\_TYPE = **'USE'**

AND QUERY\_TEXT ILIKE **'%ROLE%'**

AND QUERY\_TEXT ILIKE **'%\_PI%'**

AND TIMESTAMPDIFF(minute,END\_TIME,current\_timestamp()) <= 60

## ❖ **Monitor Tables Not Accessed**

History of access to tables and columns are stored in Snowflake for 365 days.  You can get to this data using the ACCOUNT\_USAGE.ACCESS\_HISTORY view in the Snowflake database.  This view provides JSON that shows which objects were access or modified

**SELECT** \*

FROM ORG\_PROD.METRICS.ACCESS\_HISTORY

WHERE TIMESTAMPDIFF(minute,QUERY\_START\_TIME,current\_timestamp()) <= 60

The JSON in the Access History view can be flattened and put into columns.   This data can then be joined to your tag references to better determine which PI columns are being read.

**SELECT** USER\_NAME, QUERY\_START\_TIME,

F1.VALUE:"objectName"::string AS TABLE\_PATH,

F1.VALUE:"objectID"::int AS TABLE\_ID,

F3.VALUE:"columnName"::string AS COLUMN\_NAME,

F3.VALUE:"columnID"::int AS COLUMN\_ID, QUERY\_ID

FROM ORG\_PROD.METRICS.ACCESS\_HISTORY,

LATERAL FLATTEN(DIRECT\_OBJECTS\_ACCESSED) F1,

LATERAL FLATTEN(F1.VALUE) F2,

LATERAL FLATTEN(F2.VALUE) F3

WHERE 1=1

AND TIMESTAMPDIFF(minute,QUERY\_START\_TIME,current\_timestamp()) <= 60

## ❖ **Monitor Users With PII Access**

The security admin in Snowflake can keep track of all users that have a tag that will give them access to PI data.  This can be pulled by querying the ACCOUNT\_USAGE.TAG\_REFERENCES view in the Snowflake database

**SELECT** TAG\_DATABASE, TAG\_SCHEMA, TAG\_ID, TAG\_NAME, DOMAIN, OBJECT\_NAME

FROM ORG\_PROD.METRICS.TAG\_REFERENCES

WHERE TAG\_NAME LIKE **'PI\_%'**

AND DOMAIN = **'USER'**;

# Encryption and Key Management Monitoring

For Tri-Secret Secure, you can disable access to your KMS (the KMS where your customer-managed key is stored) at any time to verify this mechanism is configured properly. This is at your discretion and does not require any involvement from Snowflake. To validate, verify that any query of Snowflake data in the environment fails.

## ❖ **Verify that annual rekeying is enabled**

Annual rekeying is a best practice.

**show** **parameters** like **'%periodic\_data\_rekeying%'** in account;

## ❖ **Monitor for any changes to the periodic\_data\_rekeying setting**

Changes to this setting are rare and deserving of scrutiny.

**select** end\_time, query\_type, query\_text, user\_name, role\_name

from ORG\_PROD.METRICS.query\_history

where execution\_status = **'SUCCESS'**

and query\_type = **'ALTER\_ACCOUNT'**

and query\_text ilike **'%periodic\_data\_rekeying%'**

AND TIMESTAMPDIFF(minute,END\_TIME,current\_timestamp()) <= 60

order by end\_time desc ;

# Integration Monitoring

Validate your integrations and document how they are configured. Once validated, monitor the QUERY\_HISTORY view for queries like: **'%integration%' ;**

Because integrations can enable a new means of access to Snowflake data, closely monitor for new integrations or the modification of existing integrations.

## ❖ **Show integrations in Snowflake**

**show** **integrations**;

**desc integration** <name>;

## ❖ **Monitor for security integrations that have been created or altered**

Security integrations shouldn't be added or changed on a frequent basis. Monitor QUERY\_HISTORY for alterations.

**select** end\_time, query\_type, query\_text, user\_name, role\_name

from ORG\_PROD.METRICS.query\_history

where execution\_status = **'SUCCESS'**

and query\_type in ( **'CREATE', 'ALTER'** )

and query\_text ilike **'%security integration%'**

AND TIMESTAMPDIFF(minute,END\_TIME,current\_timestamp()) <= 60

order by end\_time desc ;

# Backup and Recovery Monitoring

Snowflake Time Travel enables accessing historical data ( *e.g.* data that has been changed or deleted) at any point within a defined period.

**Monitoring data retention (Time Travel)**

Any changes to retention time should be monitored on any objects containing sensitive objects. Retention time is inherited from parent objects, so altering retention time at the Database level, for example, would also alter the retention time of the objects therein.

## ❖ **Monitor Data Retention**

There is retention at different levels (database, schema, account). Monitor QUERY\_HISTORY for queries matching the following pattern for all sensitive objects for any deviation from the standard determined by your business requirements as they could indicate that persisted data may be used for unintended purposes. For a full audit of the retention policy, the TABLES view in Account Usage provides a field called RETENTION\_TIME .

**show parameters** like **'%data\_retention\_time\_in\_days%'** in account;

**select** schema\_name, catalog\_name, schema\_owner, retention\_time

from ORG\_PROD.METRICS.schemata;

**select** database\_name, database\_owner, retention\_time

from ORG\_PROD.METRICS.databases ;

**select** table\_name, table\_schema, table\_catalog, table\_owner, table\_type, retention\_time

from ORG\_PROD.METRICS.tables ;

# Snowflake Parameter Monitoring

## ❖ **Monitor Account-Level Parameters**

Snowflake recommends monitoring the following account-level parameters for security purposes.

* ALLOW\_ID\_TOKEN
* CLIENT\_ENCRYPTION\_KEY\_SIZE
* INITIAL\_REPLICATION\_SIZE\_LIMIT\_IN\_TB
* NETWORK\_POLICY
* PERIODIC\_DATA\_REKEYING
* PREVENT\_UNLOAD\_TO\_INLINE\_URL
* REQUIRE\_STORAGE\_INTEGRATION\_FOR\_STAGE\_CREATION
* REQUIRE\_STORAGE\_INTEGRATION\_FOR\_STAGE\_OPERATION
* SSO\_LOGIN\_PAGE

**show** **parameters** like **'%allow\_id\_token%'** in account;

**show parameters** like **'%client\_encryption\_key\_size%'** in account;

**show parameters** like **'%initial\_replication\_size\_limit\_in\_tb%'** in account;

**show parameters** like **'%network\_policy%'** in account;

**show parameters** like **'%periodic\_data\_rekeying%'** in account;

**show parameters** like **'%prevent\_unload\_to\_inline\_url%'** in account;

**show parameters** like **'%require\_storage\_integration\_for\_stage\_creation%'** in account;

**show parameters** like **'%require\_storage\_integration\_for\_stage\_operation%'** in account;

**show parameters** like **'%sso\_login\_page%'** in account;

## ❖ **Monitor SAML and SSO Account-Level Parameters**

SAML and SSO parameter changes should be a rare event.

**show parameters** like **'saml%'** in account;

**show parameters** like **'sso%'** in account;

# Query Monitoring

## ❖ **Monitor Query Details**

**SELECT** QUERY\_TYPE, EXECUTION\_STATUS, QUERY\_ID, QUERY\_TEXT, DATABASE\_NAME, USER\_NAME,

WAREHOUSE\_NAME, WAREHOUSE\_SIZE, ERROR\_CODE, ERROR\_MESSAGE, START\_TIME, END\_TIME,

TOTAL\_ELAPSED\_TIME, (TOTAL\_ELAPSED\_TIME / 1000) / 60 AS TOTAL\_ELAPSED\_TIME\_MINUTES,

BYTES\_SCANNED, PERCENTAGE\_SCANNED\_FROM\_CACHE, BYTES\_WRITTEN, ROWS\_PRODUCED, ROWS\_INSERTED,

ROWS\_UPDATED, ROWS\_DELETED, ROWS\_UNLOADED, EXECUTION\_TIME, CREDITS\_USED\_CLOUD\_SERVICES,

QUERY\_LOAD\_PERCENT

FROM ORG\_PROD.METRICS.QUERY\_HISTORY

AND TIMESTAMPDIFF(minute,END\_TIME,current\_timestamp()) <= 60

## ❖ **Monitor Expensive Queries**

**SELECT** QUERY\_TYPE, EXECUTION\_STATUS, QUERY\_ID, QUERY\_TEXT, DATABASE\_NAME, USER\_NAME,

WAREHOUSE\_NAME, WAREHOUSE\_SIZE, ERROR\_CODE, ERROR\_MESSAGE, START\_TIME, END\_TIME,

TOTAL\_ELAPSED\_TIME, TOTAL\_ELAPSED\_TIME / 1000) / 60 AS TOTAL\_ELAPSED\_TIME\_MINUTES,

BYTES\_SCANNED, PERCENTAGE\_SCANNED\_FROM\_CACHE, BYTES\_WRITTEN, ROWS\_PRODUCED, ROWS\_INSERTED,

ROWS\_UPDATED, ROWS\_DELETED, ROWS\_UNLOADED, EXECUTION\_TIME, CREDITS\_USED\_CLOUD\_SERVICES,

QUERY\_LOAD\_PERCENT

FROM ORG\_PROD.METRICS.QUERY\_HISTORY

AND TIMESTAMPDIFF(minute,END\_TIME,current\_timestamp()) <= 60

ORDER BY CREDITS\_USED\_CLOUD\_SERVICES DESC

## ❖ **Monitor Longest Running Queries**

**SELECT** QUERY\_TYPE, EXECUTION\_STATUS, QUERY\_ID, QUERY\_TEXT, DATABASE\_NAME, USER\_NAME,

WAREHOUSE\_NAME, WAREHOUSE\_SIZE, ERROR\_CODE, ERROR\_MESSAGE, START\_TIME, END\_TIME,

TOTAL\_ELAPSED\_TIME, (TOTAL\_ELAPSED\_TIME / 1000) / 60 AS TOTAL\_ELAPSED\_TIME\_MINUTES,

BYTES\_SCANNED, PERCENTAGE\_SCANNED\_FROM\_CACHE, BYTES\_WRITTEN, ROWS\_PRODUCED, ROWS\_INSERTED,

ROWS\_UPDATED, ROWS\_DELETED, ROWS\_UNLOADED, EXECUTION\_TIME, CREDITS\_USED\_CLOUD\_SERVICES,

QUERY\_LOAD\_PERCENT

FROM ORG\_PROD.METRICS.QUERY\_HISTORY

AND TIMESTAMPDIFF(minute,END\_TIME,current\_timestamp()) <= 60

ORDER BY TOTAL\_ELAPSED\_TIME DESC

## **Monitor Queries Queuing**

# Storage Monitoring

## ❖ **Monitor Storage Summary**

SELECT USAGE\_DATE, ROUND((STORAGE\_BYTES+STAGE\_BYTES+FAILSAFE\_BYTES)/1000000000) AS TOTAL\_STORAGE\_GIG,

ROUND(STORAGE\_BYTES/1000000000) AS STORAGE\_IN\_GB,

ROUND(STAGE\_BYTES/1000000000) AS STAGE\_IN\_GB,

ROUND(FAILSAFE\_BYTES/1000000000) AS FAILESAFE\_IN\_GB

FROM ORG\_PROD.METRICS.STORAGE\_USAGE

ORDER BY USAGE\_DATE DESC

LIMIT 90;

## ❖ **Monitor Storage Details**

Sum #Rows By Database