# **.MERAKI\_HISTORICAL\_LOAD\_PROC**

use role fr\_dev\_dataengineer;

CREATE OR REPLACE PROCEDURE RAW\_DEV.DEV\_RAW\_MERAKI.MERAKI\_HISTORICAL\_LOAD\_PROC()

RETURNS VARCHAR(500)

LANGUAGE SQL

EXECUTE AS OWNER

AS '

DECLARE RecordCount INT;

BEGIN

-- Author: VAMSI

-- Created: 17/02/2025

-- Description: Historical load Procedure

-- Modifications: 17/02/2025: Created the procedure for M Historical Load

COPY INTO RAW\_DEV.DEV\_RAW\_MERAKI.MERAKI\_HISTORICAL FROM ''@"RAW\_DEV"."DEV\_RAW\_MERAKI"."MERAKI\_STAGE"/meraki\_data.csv'' FILE\_FORMAT = (TYPE = ''CSV'' FIELD\_OPTIONALLY\_ENCLOSED\_BY = ''"'' SKIP\_HEADER = 1) ON\_ERROR = ''SKIP\_FILE'';

SELECT "rows\_loaded" INTO RecordCount FROM TABLE(RESULT\_SCAN(LAST\_QUERY\_ID()));

INSERT INTO HARMONIZED\_DEV.AUDIT\_LOGS.AUDIT (

SOURCE,

DATABASE\_NAME,

SCHEMA\_NAME,

TABLE\_NAME,

ROW\_COUNT,

LOAD\_TIMESTAMP,

PROCEDURE\_NAME,

STATUS,

ERROR\_MESSAGE

)

VALUES(

''MERAKI\_API'',

''RAW\_DEV'',

''DEV\_RAW\_MERAKI'',

''MERAKI\_HISTORICAL'',

:RecordCount,

CURRENT\_TIMESTAMP(),

''MERAKI\_HISTORICAL\_LOAD\_PROC'',

''SUCCESS'',

NULL

);

EXCEPTION

WHEN OTHER THEN

INSERT INTO HARMONIZED\_DEV.AUDIT\_LOGS.AUDIT (

SOURCE,

DATABASE\_NAME,

SCHEMA\_NAME,

TABLE\_NAME,

ROW\_COUNT,

LOAD\_TIMESTAMP,

PROCEDURE\_NAME,

STATUS,

ERROR\_MESSAGE

)

VALUES(

''MERAKI\_API'',

''RAW\_DEV'',

''DEV\_RAW\_MERAKI'',

''MERAKI\_HISTORICAL'',

0,

CURRENT\_TIMESTAMP(),

''MERAKI\_HISTORICAL\_LOAD\_PROC'',

''ERROR'',

:sqlerrm

);

END;

';

CALL MERAKI\_HISTORICAL\_LOAD\_PROC();

—---------

# **MERAKI\_HISTORICAL\_HAR**

CREATE OR REPLACE PROCEDURE HARMONIZED\_DEV.DEV\_HAR\_MERAKI.MERAKI\_HISTORICAL\_HAR()

RETURNS VARCHAR(500)

LANGUAGE SQL

EXECUTE AS OWNER

AS '

DECLARE RecordCount\_Meraki NUMBER;

TableName STRING;

BEGIN

-- Author: Vamsi

-- Created: 02/17/2025

-- Description: Historical load Procedure

-- Modifications: 02/17/2025: Created the procedure for Meraki Historical Load to Harmonized DB

CREATE TABLE IF NOT EXISTS HARMONIZED\_DEV.DEV\_HAR\_MERAKI.MERAKI

(

adaptivePolicyGroup VARCHAR(50),

description VARCHAR(100),

deviceTypePrediction VARCHAR(50),

firstSeen TIMESTAMP\_NTZ(9),

groupPolicy8021x VARCHAR(50),

id VARCHAR(30),

ip VARCHAR(40),

ip6 VARCHAR(50),

ip6Local VARCHAR(50),

lastSeen TIMESTAMP\_NTZ(9),

mac VARCHAR(20),

manufacturer VARCHAR(30),

namedVlan VARCHAR(20),

notes VARCHAR(100),

os VARCHAR(30),

pskGroup VARCHAR(30),

recentDeviceConnection VARCHAR(50),

recentDeviceMac VARCHAR(30),

recentDeviceName VARCHAR(20),

recentDeviceSerial VARCHAR(30),

smInstalled VARCHAR(20),

ssid VARCHAR(30),

status VARCHAR(20),

switchport VARCHAR(20),

usage\_RECV NUMBER,

usage\_SENT NUMBER,

usage\_TOTAL NUMBER,

user VARCHAR(30),

vlan NUMBER,

wirelessCapabilities VARCHAR(50),

LOCATION VARCHAR(20),

ATTENDANCEDATE TIMESTAMP\_NTZ(9)

);

CREATE TABLE IF NOT EXISTS HARMONIZED\_DEV.AUDIT\_LOGS.AUDIT (

SOURCE VARCHAR(16777216),

DATABASE\_NAME VARCHAR(16777216),

SCHEMA\_NAME VARCHAR(16777216),

TABLE\_NAME VARCHAR(16777216),

ROW\_COUNT NUMBER(38,0),

LOAD\_TIMESTAMP TIMESTAMP\_NTZ(9),

PROCEDURE\_NAME VARCHAR(16777216),

STATUS VARCHAR(16777216),

ERROR\_MESSAGE VARCHAR(16777216)

);

INSERT INTO HARMONIZED\_DEV.DEV\_HAR\_MERAKI.MERAKI (

adaptivePolicyGroup,

description,

deviceTypePrediction,

firstSeen,

groupPolicy8021x,

id,

ip,

ip6,

ip6Local,

lastSeen,

mac,

manufacturer,

namedVlan,

notes,

os,

pskGroup,

recentDeviceConnection,

recentDeviceMac,

recentDeviceName,

recentDeviceSerial,

smInstalled,

ssid,

status,

switchport,

usage\_RECV ,

usage\_SENT ,

usage\_TOTAL,

user,

vlan,

wirelessCapabilities,

LOCATION,

ATTENDANCEDATE

)

SELECT

NULL,

DESCRIPTION,

NULL,

NULL,

NULL,

NULL,

NULL,

NULL,

NULL,

LASTSEEN,

MACADDRESS,

NULL,

NULL,

NULL,

NULL,

NULL,

RECENTDEVICECONNECTION,

NULL,

NULL,

NULL,

NULL,

NULL,

NULL,

NULL,

NULL,

NULL,

NULL,

NULL,

NULL,

NULL,

LOCATION,

ATTENDANCEDATE

FROM RAW\_DEV.DEV\_RAW\_MERAKI.MERAKI\_HISTORICAL;

TableName := ''MERAKI'';

RecordCount\_Meraki := SQLROWCOUNT;

INSERT INTO HARMONIZED\_DEV.AUDIT\_LOGS.AUDIT (

SOURCE,

DATABASE\_NAME,

SCHEMA\_NAME,

TABLE\_NAME,

ROW\_COUNT,

LOAD\_TIMESTAMP,

PROCEDURE\_NAME,

STATUS,

ERROR\_MESSAGE

)

VALUES(

''MERAKI\_RAW\_DB'',

''HARMONIZED\_DEV'',

''DEV\_HAR\_MERAKI'',

''MERAKI'',

:RecordCount\_Meraki,

CURRENT\_TIMESTAMP(),

''MERAKI\_HISTORICAL\_HAR'',

''SUCCESS'',

NULL

);

EXCEPTION

WHEN OTHER THEN

INSERT INTO HARMONIZED\_DEV.AUDIT\_LOGS.AUDIT (

SOURCE,

DATABASE\_NAME,

SCHEMA\_NAME,

TABLE\_NAME,

ROW\_COUNT,

LOAD\_TIMESTAMP,

PROCEDURE\_NAME,

STATUS,

ERROR\_MESSAGE

)

VALUES(

''MERAKI\_RAW\_DB'',

''HARMONIZED\_DEV'',

''DEV\_HAR\_MERAKI'',

:TableName,

0,

CURRENT\_TIMESTAMP(),

''MERAKI\_HISTORICAL\_HAR'',

''FAILURE'',

:sqlerrm);

END;

';

call MERAKI\_HISTORICAL\_HAR();

—-

# **Meraki\_API\_ContinuousLoad**

use role accountadmin;

CREATE OR REPLACE NETWORK RULE Meraki\_API\_Rule

MODE = EGRESS

TYPE = HOST\_PORT

VALUE\_LIST = ('api.meraki.com');

CREATE OR REPLACE SECRET Meraki\_Secret

TYPE = GENERIC\_STRING

SECRET\_STRING = '021d19ae0ee48a09b0c6a5c5635eaf03372f4948';

GRANT READ ON SECRET Meraki\_Secret TO ROLE FR\_DEV\_DATA\_ENGINEER;

CREATE OR REPLACE EXTERNAL ACCESS INTEGRATION Meraki\_External\_Network\_Access

ALLOWED\_NETWORK\_RULES = (Meraki\_API\_Rule)

ALLOWED\_AUTHENTICATION\_SECRETS = (Meraki\_Secret)

ENABLED = TRUE;

GRANT USAGE ON INTEGRATION Meraki\_External\_Network\_Access TO ROLE fr\_dev\_data\_engineer;

use role fr\_dev\_dataengineer;

use role accountadmin;

show integrations;

CREATE OR REPLACE PROCEDURE RAW\_DEV.DEV\_RAW\_MERAKI.Meraki\_API\_ContinuousLoad()

RETURNS VARCHAR

LANGUAGE PYTHON

RUNTIME\_VERSION = '3.10'

PACKAGES = ('snowflake-snowpark-python','requests')

HANDLER = 'fetch\_posts'

EXTERNAL\_ACCESS\_INTEGRATIONS = (MERAKI\_EXTERNAL\_NETWORK\_ACCESS)

SECRETS = ('cred'=MERAKI\_SECRET)

EXECUTE AS OWNER

AS '

import \_snowflake

import requests

import json

from snowflake.snowpark import Session

def fetch\_posts(session: Session):

token = \_snowflake.get\_generic\_secret\_string(''cred'')

api\_urls = {

"San\_Francisco": "https://api.meraki.com/api/v1/networks/L\_638385247179776905/clients?perPage=1000&vlan=108",

"San\_Juan": "https://api.meraki.com/api/v1/networks/L\_638385247179774525/clients?perPage=1000&vlan=88",

"New\_York\_City": "https://api.meraki.com/api/v1/networks/L\_638385247179774894/clients?perPage=1000&vlan=78"

}

headers = {"Authorization": f"Bearer {token}"}

# Ensure raw table exists

session.sql("CREATE TABLE IF NOT EXISTS RAW\_DEV.DEV\_RAW\_MERAKI.MERAKI\_RAW (data VARIANT)").collect()

# Ensure audit table exists

session.sql("""

CREATE TABLE IF NOT EXISTS HARMONIZED\_DEV.AUDIT\_LOGS.AUDIT (

SOURCE VARCHAR(16777216),

DATABASE\_NAME VARCHAR(16777216),

SCHEMA\_NAME VARCHAR(16777216),

TABLE\_NAME VARCHAR(16777216),

ROW\_COUNT NUMBER(38,0),

LOAD\_TIMESTAMP TIMESTAMP\_NTZ(9),

PROCEDURE\_NAME VARCHAR(16777216),

STATUS VARCHAR(16777216),

ERROR\_MESSAGE VARCHAR(16777216)

)

""").collect()

success\_flag = True

error\_messages = []

for location, url in api\_urls.items():

try:

response = requests.get(url, headers=headers)

if response.status\_code == 200:

json\_data = response.json()

row\_count = len(json\_data) if isinstance(json\_data, list) else 1

# Insert data into raw table

session.sql(

"INSERT INTO RAW\_DEV.DEV\_RAW\_MERAKI.MERAKI\_RAW (data) SELECT PARSE\_JSON(?)",

[json.dumps(json\_data)]

).collect()

# Insert audit log (Success)

session.sql("""

INSERT INTO HARMONIZED\_DEV.AUDIT\_LOGS.AUDIT

(SOURCE, DATABASE\_NAME, SCHEMA\_NAME, TABLE\_NAME, ROW\_COUNT, LOAD\_TIMESTAMP, PROCEDURE\_NAME, STATUS, ERROR\_MESSAGE)

VALUES (?, ''RAW\_DEV'', ''DEV\_RAW\_MERAKI'', ''MERAKI\_RAW'', ?, CURRENT\_TIMESTAMP(), ''Meraki\_API\_ContinuousLoad'', ''SUCCESS'', NULL)

""", [location + "\_MERAKI\_API", row\_count]).collect()

else:

raise Exception(f"Error fetching {location} data: {response.status\_code}. Response: {response.text}")

except Exception as e:

success\_flag = False

error\_message = str(e)

error\_messages.append(f"{location}: {error\_message}")

# Insert audit log (Failure)

session.sql("""

INSERT INTO HARMONIZED\_DEV.AUDIT\_LOGS.AUDIT

(SOURCE, DATABASE\_NAME, SCHEMA\_NAME, TABLE\_NAME, ROW\_COUNT, LOAD\_TIMESTAMP, PROCEDURE\_NAME, STATUS, ERROR\_MESSAGE)

VALUES (?, ''RAW\_DEV'', ''DEV\_RAW\_MERAKI'', ''MERAKI\_RAW'', NULL, CURRENT\_TIMESTAMP(), ''Meraki\_API\_ContinuousLoad'', ''FAILURE'', ?)

""", [location + "\_MERAKI\_API", error\_message]).collect()

# Get email recipients

email\_query = "SELECT EMAIL\_ID FROM RAW\_DEV.DEV\_RAW\_MERAKI.EMAIL WHERE FLAG=TRUE;"

email\_result = session.sql(email\_query).collect()

email\_list = [row["EMAIL\_ID"] for row in email\_result]

if email\_list:

email\_addresses = ",".join(email\_list)

# Send email notification

session.sql("""

CALL SYSTEM$SEND\_EMAIL(

''email\_integration'',

?,

?,

?

)

""", [email\_addresses, f"{location} Meraki API Failed", error\_message]).collect()

# Return final status based on all API calls

if success\_flag:

return "Procedure executed successfully"

else:

return f"Procedure failed. Errors: {'' | ''.join(error\_messages)}"

'

;

–

# **MERAKI\_TRANSFORMATION**

CREATE OR REPLACE PROCEDURE HARMONIZED\_DEV.DEV\_HAR\_MERAKI.MERAKI\_TRANSFORMATION()

RETURNS VARCHAR(500)

LANGUAGE SQL

EXECUTE AS OWNER

AS '

DECLARE RecordCount\_Meraki NUMBER;

TableName VARCHAR;

BEGIN

-- Author: Vamsi

-- Created: 02/14/2025

-- Description: Continuous load Procedure

-- Modifications: 02/14/2025: Created the procedure for Microsoft Continuous Load from RAW to HARMONIZED database

INSERT INTO HARMONIZED\_DEV.DEV\_HAR\_MERAKI.MERAKI(

adaptivePolicyGroup,

description,

deviceTypePrediction,

firstSeen,

groupPolicy8021x,

id,

ip,

ip6,

ip6Local,

lastSeen,

mac,

manufacturer,

namedVlan,

notes,

os,

pskGroup,

recentDeviceConnection,

recentDeviceMac,

recentDeviceName,

recentDeviceSerial,

smInstalled,

ssid,

status,

switchport,

usage\_RECV ,

usage\_SENT ,

usage\_TOTAL,

user,

vlan,

wirelessCapabilities,

location,

attendancedate

)

SELECT

flattened\_data.value:adaptivePolicyGroup::STRING AS adaptivePolicyGroup,

flattened\_data.value:description::STRING AS description,

flattened\_data.value:deviceTypePrediction::STRING AS deviceTypePrediction,

flattened\_data.value:firstSeen::STRING AS firstSeen,

flattened\_data.value:groupPolicy8021x::STRING AS groupPolicy8021x,

flattened\_data.value:id::STRING AS id,

flattened\_data.value:ip::STRING AS ip,

flattened\_data.value:ip6::STRING AS ip6,

flattened\_data.value:ip6Local::STRING AS ip6Local,

flattened\_data.value:lastSeen::STRING AS lastSeen,

replace(flattened\_data.value:mac::STRING, '':'','''') AS mac,

flattened\_data.value:manufacturer::STRING AS manufacturer,

flattened\_data.value:namedVlan::STRING AS namedVlan,

flattened\_data.value:notes::STRING AS notes,

flattened\_data.value:os::STRING AS os,

flattened\_data.value:pskGroup::STRING AS pskGroup,

flattened\_data.value:recentDeviceConnection::STRING AS recentDeviceConnection,

flattened\_data.value:recentDeviceMac::STRING AS recentDeviceMac,

flattened\_data.value:recentDeviceName::STRING AS recentDeviceName,

flattened\_data.value:recentDeviceSerial::STRING AS recentDeviceSerial,

flattened\_data.value:smInstalled::STRING AS smInstalled,

flattened\_data.value:ssid::STRING AS ssid,

flattened\_data.value:status::STRING AS status,

flattened\_data.value:switchport::STRING AS switchport,

flattened\_data.value:usage.recv::NUMBER AS recv,

flattened\_data.value:usage.sent::NUMBER AS sent,

flattened\_data.value:usage.total::NUMBER AS total,

flattened\_data.value:user::STRING AS user,

flattened\_data.value:vlan::STRING AS vlan,

flattened\_data.value:wirelessCapabilities::STRING AS wirelessCapabilities,

case

when flattened\_data.value:vlan::NUMBER = 108 THEN ''San Francisco''

when flattened\_data.value:vlan::NUMBER = 88 THEN ''San Juan''

when flattened\_data.value:vlan::NUMBER = 78 THEN ''New York City''

ELSE NULL

END AS location,

CURRENT\_TIMESTAMP as attendancedate

FROM

RAW\_DEV.DEV\_RAW\_MERAKI.MERAKI\_RAW,

LATERAL FLATTEN(input => MERAKI\_RAW.DATA) AS flattened\_data;

TableName := ''MERAKI'';

RecordCount\_Meraki := SQLROWCOUNT;

INSERT INTO HARMONIZED\_DEV.AUDIT\_LOGS.AUDIT (

SOURCE,

DATABASE\_NAME,

SCHEMA\_NAME,

TABLE\_NAME,

ROW\_COUNT,

LOAD\_TIMESTAMP,

PROCEDURE\_NAME,

STATUS,

ERROR\_MESSAGE

)

VALUES(

''MERAKI\_RAW\_DB'',

''HARMONIZED\_DEV'',

''DEV\_HAR\_MERAKI'',

''MERAKI'',

:RecordCount\_Meraki,

CURRENT\_TIMESTAMP(),

''MERAKI\_TRANSFORMATION'',

''SUCCESS'',

NULL

);

EXCEPTION

WHEN OTHER THEN

INSERT INTO HARMONIZED\_DEV.AUDIT\_LOGS.AUDIT (

SOURCE,

DATABASE\_NAME,

SCHEMA\_NAME,

TABLE\_NAME,

ROW\_COUNT,

LOAD\_TIMESTAMP,

PROCEDURE\_NAME,

STATUS,

ERROR\_MESSAGE

)

VALUES(

''MERAKI\_RAW\_DB'',

''HARMONIZED\_DEV'',

''DEV\_HAR\_MERAKI'',

:TableName,

0,

CURRENT\_TIMESTAMP(),

''MERAKI\_TRANSFORMATION'',

''FAILURE'',

:sqlerrm

);

END;

';

call MERAKI\_TRANSFORMATION();