# Enhanced EDM Monitoring Guide

All dataops automation database objects are in the following role, warehouse, schema and database:

* **Role**: EDDM\_DATA\_ENGINEER\_GG
* **Warehouse**: DEV\_ENGINEER\_WH
* **Database**: EDM\_REFINED\_PRD
* **Schema**: SCRATCH

## **Look-up Table Structure**

There are two lookup tables for data ops:

### **AppDTasks**

This table is used to store all tasks and load tables of all bods. It has the following columns/fields:

* **TaskName**: eg: ESED\_ITEMRECODE\_R\_TASK
* **SchemaName**: eg: DW\_R\_PRODUCT
* **Zone**: eg: REFINED
* **CI**: Configuration item to be created by Shauntoria, this field is optional during inserts
* **EffectiveStartDate**: This field is automatically populated during inserts
* **EffectiveEndDate**: This field is initialized to NULL, and only updated whenever something such as taskname changes for existing entry/record
* **DateAdded**: This field automatically populated during inserts
* **DateModified**: This field automatically populated during inserts
* **ModifiedBy**: This field automatically populated during inserts

### **AppDExceptionTables**

This table is used to store all exception tables all bods. It has the following columns/fields:

* **TableName**: eg: ECATALOG\_ITEM\_EXCEPTIONS
* **Kafkaout**: this is the taskname in kafkaout zone for this out table. eg: ECATALOG\_ITEM\_O\_TASK
* **KafkaOutQueue**: this is the taskname in kafkaoutqueue zone for this out table. eg: ECATALOG\_ITEM\_KAFKAOUTQUEUE\_O\_TASK
* **MSGCol**: this is the exception errormessage for this out table. Eg: ERRORMSG
* **DateTimeCol**: this is the datetime field of the exception table indicating the date the exception was logged. Eg: CREATETIME
* **Cols**: this is a comma-separated list of columns from the exception table that will be used by lookup operations to query that exception table
* **CI**: Configuration item to be created by Shauntoria, this field is optional during inserts
* **EffectiveStartDate**: This field is automatically populated during inserts
* **EffectiveEndDate**: This field is initialized to NULL, and only updated whenever something such as taskname changes for existing entry/record
* **DateAdded**: This field automatically populated during inserts
* **DateModified**: This field automatically populated during inserts
* **ModifiedBy**: This field automatically populated during inserts

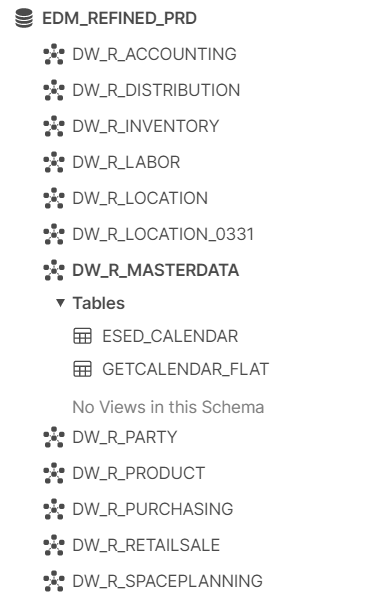
## Loading Look-up Tables

Whenever new tasks are ready for testing (new tasks are ready for testing when their CIs are created in DEV and QA), the lookup task table in Snowflake DEV environment needs to be loaded as well. When inserting into this table, these three columns: TaskName ,SchemaName, Zone are required. For example:

insert into AppDTasks (TaskName ,SchemaName,Zone)

VALUES ('ESED\_CALENDAR','DW\_R\_MASTERDATA','LOAD'), ('ESED\_GETCALENDAR\_R\_TASK','DW\_R\_MASTERDATA','REFINED'), ('SP\_GETCALENDAR\_TO\_BIM\_LOAD\_TASK','DW\_R\_MASTERDATA','CONFIRMED')

After inserting into the task lookup table, next make sure that load table for that BOD also exist in DEV. You can check if it exists by looking in the right corner of the database listing, and expanding the EDM\_REFINED\_DEV database as below:



If it does not exist, then create it as follows:

CREATE OR REPLACE TABLE EDM\_REFINED\_DEV.DW\_R\_MASTERDATA.ESED\_CALENDAR( FILENAME VARCHAR(5000),

SRC\_XML VARIANT,

CREATED\_TS TIMESTAMP\_LTZ(9)

);

Make sure you replace the schema and table names in the above query with the appropriate values.

Whenever new exception out tables are ready for testing, then the exception lookup table in Snowflake DEV environment needs to be loaded as well. When inserting into this table, these columns: Kafkaout,KafkaoutQueue,TableName, MSGCol, DateTimeCol, Cols are required. For example:

insert into AppDExceptionTables(Kafkaout,KafkaoutQueue,TableName , MSGCol, DateTimeCol, Cols)

VALUES ('ECATALOG\_PROMO\_O\_TASK','ECATALOG\_PROMO\_KAFKAOUTQUEUE\_O\_TASK','ECATALOG\_PROMO\_EXCEPTIONS','ERRORMSG','CREATETIME','BANNER, DIVISION, ROGCD, STOREID, UPCID, COUPONAMT, ORIGINALCOUPONFACTOR, ORIGINALCOUPONAMT, COUPONMETHODCD, PROMOTIONSTARTDT, PROMOTIONENDDT, MINIMUMPURCHASEQTY, COUPONLIMITQTY, LINKPLUNBR, BIBTYPECD, BIBTAGNBR, BIBTAGVERBIAGE, VALID, ERRORMSG, to\_varchar(CREATETIME,''MM/dd/yyyy HH:mm:ss'') AS CREATETIME, to\_varchar(MODIFIEDTIME,''MM/dd/yyyy HH:mm:ss'') AS MODIFIEDTIME')

Make sure you convert all timestamp columns to string like the example query above.

## Tracking Exception History

There is a table called EXCEPTIONHISTORY that we use to track historical exceptions. The table’s definition is as follows:

* **TableName**: eg: ECATALOG\_ITEM\_EXCEPTIONS, this column is automatically populated
* **ExceptionReason**: this is the error message, this column is automatically populated
* **ReportedBy**: this is the dataops team member responsible for this exception table, this column is automatically populated
* **IncidentNumber**: this is the incident number to be updated by dataops team member on daily monitoring schedule
* **Severity**: this is the severity level of the exception, to be updated by dataops team member on daily monitoring schedule
* **TotalExceptions**: this is the exception count for a specific day, this column is automatically populated
* **TotalExceptionafter0710**: this is the exception count after 7/10/2020, this column is automatically populated
* **TotalFalseFlag**: this is the total exceptions that were incorrectly reported, this column is automatically populated by taking the difference between **TotalExceptions and TotalFixed**
* **TotalFixed**: this is the total exception count that were fixed, to be updated by the exception auto-reprocessing program/code that was developed by Amit and Manish
* **IsFixed**: this is a Boolean column that indicates if issue is fixed or not, to be updated by dataops team members
* **IssueSrc**: this is source of the exception issue, eg: EDIS, to be updated by dataops team members
* **RootCause**: this is actual cause of the exception, to be updated by dataops team members
* **ActionItem**: this is typically the link to the JIRA ticket or the JIRA ticket number created for this exception, to be updated by dataops team members
* **EmailTrailLink**: this is the link to the PDF document of the email thread of this exception, to be updated by dataops team members
* **DateReported**: This field is automatically populated during inserts
* **DateModified**: This field is automatically populated during inserts
* **ModifiedBy**: This field is automatically populated during inserts

## Updating Exception History table

This table is expected to be updated by dataops team member as more details such as root cause of exceptions and incident numbers become available. Below is a sample query for this:

UPDATE EDM\_REFINED\_PRD.SCRATCH.EXCEPTIONHISTORY

SET Severity='P4',DateUpdated=current\_timestamp

WHERE TABLENAME='ECATALOG\_STOREPRICE\_EXCEPTIONS'

AND TRIM(ExceptionReason)=TRIM('upTblNum is empty. upMeasure is empty. upLabelUnit is empty. upMultFctr is empty.')

In the above query, we are updating the severity of the exception “'upTblNum is empty. upMeasure is empty. upLabelUnit is empty. upMultFctr is empty” that occurred in “ECATALOG\_STOREPRICE\_EXCEPTIONS” table.

Note that the TABLENAME and ExceptionReason columns together serve as primary key for the Exception history table. So each time you are updating this table, ensure that you apply filters on those two columns just like in the sample query above.

Always update the “DateUpdated” column with the current timestamp just like in the above UPDATE query, this way we will keep track of when changes were made to this table.

## Exception History Data Consumption

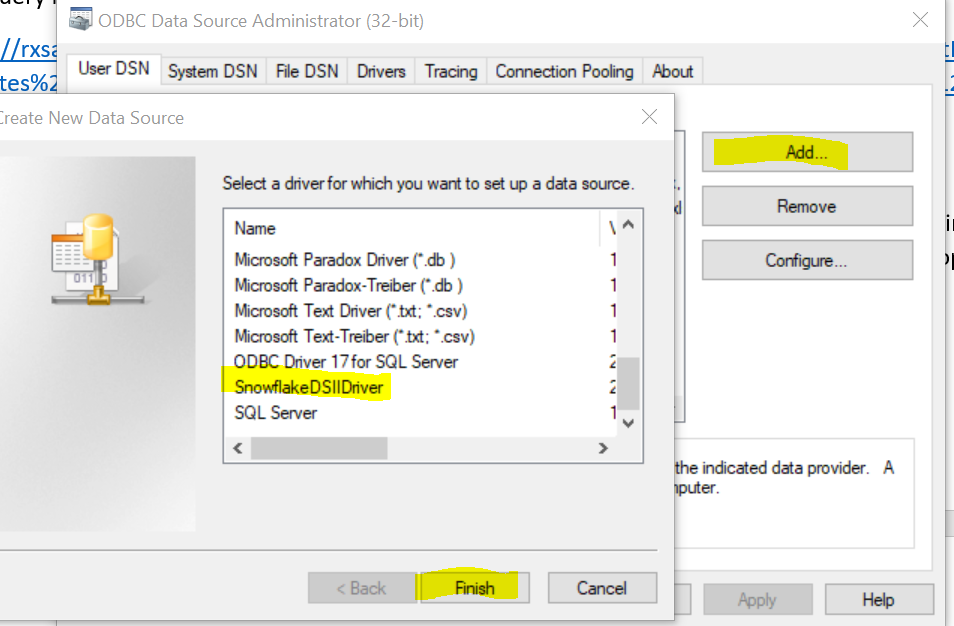
To consume the exception history data especially for non dataops team members, use ODBC in excel to pull latest data from Snowflake. The exception history excel document that has ODBC configuration and SQL query is currently available in our SharePoint shared location at the link below:

<https://rxsafeway.sharepoint.com/sites/EDM/Shared%20Documents/Forms/AllItems.aspx?RootFolder=%2Fsites%2FEDM%2FShared%20Documents%2FDataOps%20And%20Monitoring&CT=1598028124177&OR=OWA-NT&CID=03fd90be-0ac4-3a39-366b-500ee914e1e2>

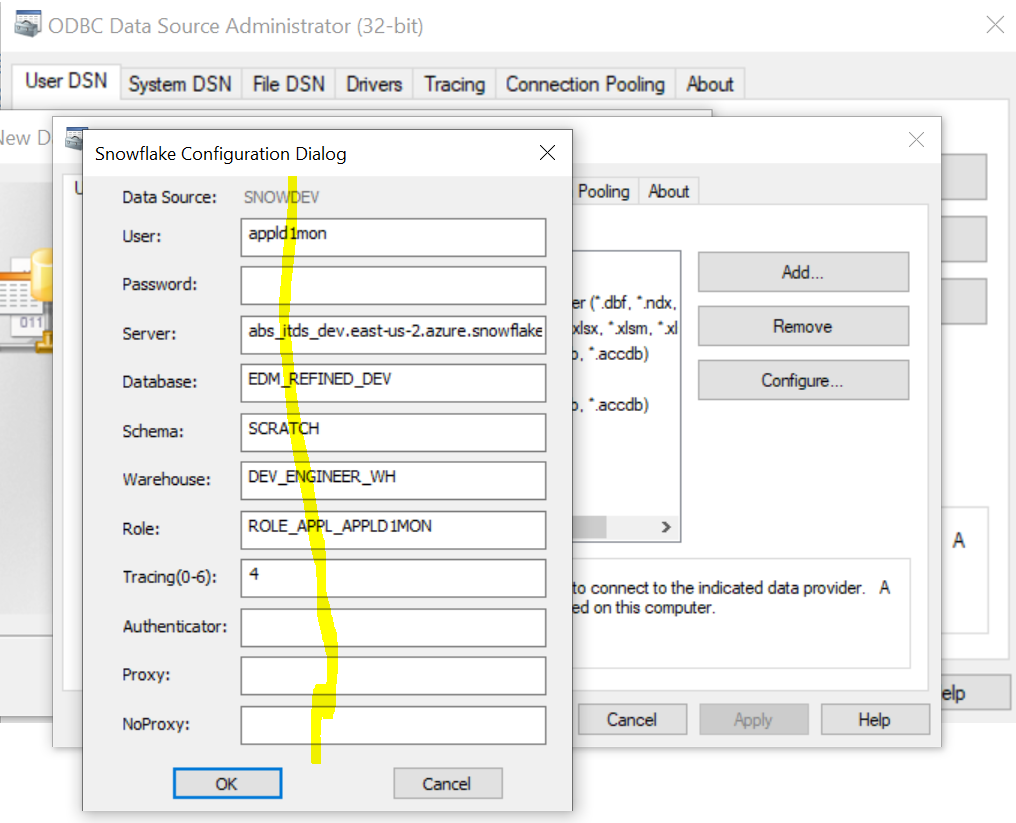
we have also uploaded 32bits and 64bits Snowflake ODBC drivers to this same shared location.

To configure ODBC in your local machine/computer, download the snowflake ODBC drivers and install them. After installation, search for “ODBC Data Source” in your windows start search bar, then open the “ODBC Data Sources (32bit)” app.

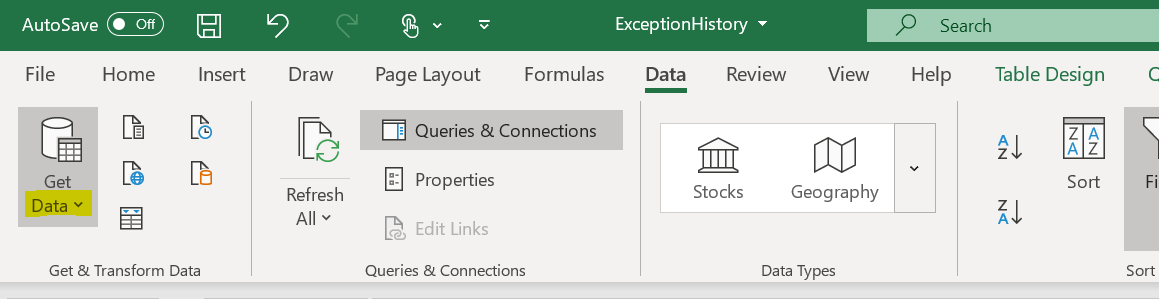
Click on the “Add” button, then select SnowflakeDSIIDriver, and click on finish as highlighted below:



Then enter details as shown below:



Then click OK, and OK again to exit the ODBC DSN configuration window. Once you are done with that, open the exception history excel document you downloaded from SharePoint server, and configure your ODBC DSN as follows. In the excel doc, click on the “data” tab and click on “Get data”,



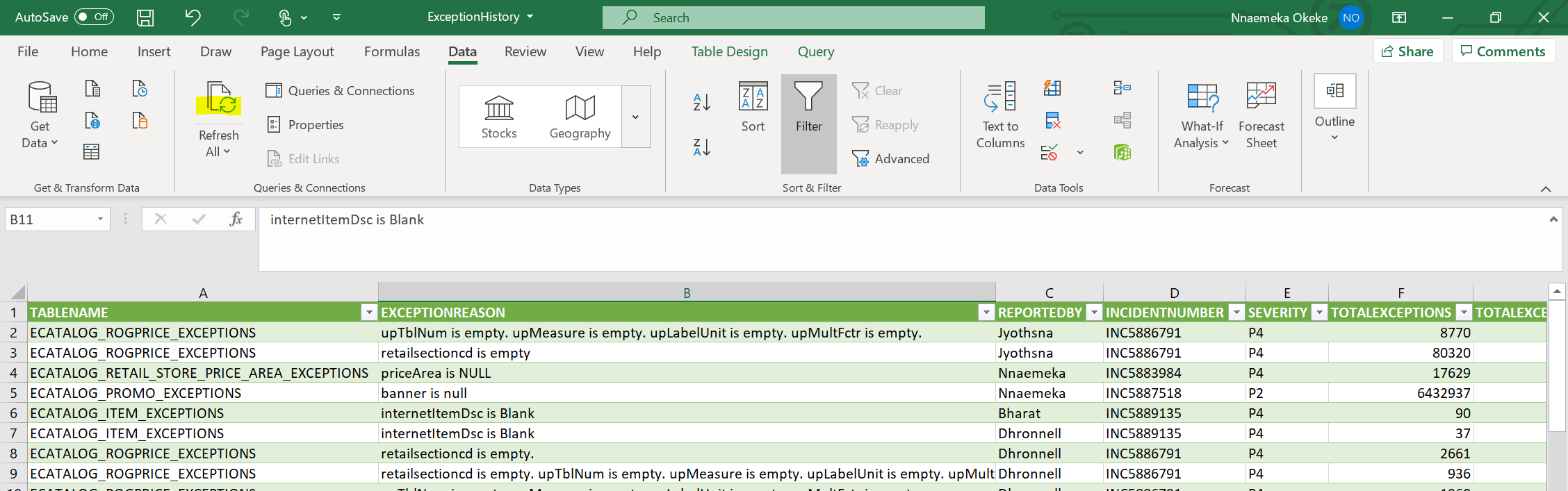
a dropdown will show, then click on “From other sources”, another dropdown will show, then click on “From ODBC”.

Select the “SNOWDEV” DSN we created during the windows ODBC configuration we completed above:



Before you click the “OK” button, make sure that you are connected to the Albertson VPN. Then click “OK” to complete the setup.

Each time you want to get the latest exception history data, just open that exception history excel document you downloaded from SharePoint server, and click on refresh button under the excel “data” tab as below:



## AppD Testing for New Tasks and Query Configuration

AppD queries also use the taskname and exception tablename lookup tables described at the beginning of this document. AppD queries are setup in the yml file as follows:

**Confirmed Zone:**

SELECT P\_NAME,P\_STAGE,NAME,STATE,NO\_JOB,COMPLETED\_TIME,COMPLETE\_RANK FROM TABLE(EDM\_REFINED\_DEV.SCRATCH.GETAPPD\_REFCONFOUT\_DATA('EDM\_CONFIRMED\_DEV','CONFIRMED'))

**Refined Zone:**

SELECT P\_NAME,P\_STAGE,NAME,STATE,NO\_JOB,COMPLETED\_TIME,COMPLETE\_RANK FROM TABLE(EDM\_REFINED\_DEV.SCRATCH.GETAPPD\_REFCONFOUT\_DATA('EDM\_REFINED\_DEV','REFINED'))

**KakfaOut Zone:**

SELECT P\_NAME,P\_STAGE,NAME,STATE,NO\_JOB,COMPLETED\_TIME,COMPLETE\_RANK FROM(

SELECT P\_NAME,P\_STAGE,NAME,STATE,NO\_JOB,COMPLETED\_TIME,COMPLETE\_RANK FROM TABLE(EDM\_REFINED\_DEV.SCRATCH.GETAPPD\_REFCONFOUT\_DATA('EDM\_CONFIRMED\_OUT\_DEV','KAFKAOUT'))

UNION ALL

SELECT P\_NAME,P\_STAGE,NAME,STATE,NO\_JOB,COMPLETED\_TIME,COMPLETE\_RANK FROM TABLE(EDM\_REFINED\_DEV.SCRATCH.GETAPPD\_REFCONFOUT\_DATA('EDM\_CONFIRMED\_OUT\_DEV','KAFKAOUTQUEUE'))

UNION ALL

SELECT P\_NAME,P\_STAGE,NAME,STATE,NO\_JOB,COMPLETED\_TIME,COMPLETE\_RANK FROM TABLE(EDM\_REFINED\_DEV.SCRATCH.GETAPPD\_REFCONFOUT\_DATA('EDM\_ANALYTICS\_DEV','KAFKAOUT')))

**Load Zone (Dev environment):**

SELECT P\_NAME,NAME,STATE FROM TABLE(DEMO\_DB.PUBLIC.GETAPPD\_LOAD());

**Load Zone (Prod environment):**

SELECT P\_NAME,NAME,STATE FROM TABLE(EDM\_REFINED\_PRD.SCRATCH.GETAPPD\_LOAD());

## AppD Query Functions

As you can see from the above queries, we have created a set of user defined functions that AppD application can call to get the data for each specific zone.

The GETAPPD\_REFCONFOUT\_DATA is used to get data for refined, confirmed, and kafkaout zones. It takes two input parameters: the first parameter is the database name, and the second parameter is the zone.

The GETAPPD\_LOAD is used to get data only for the load zone, and as such takes no input parameters.

## Simulating Task Failures and Success

We also created a stored procedure to be called by AppD monitoring team during testing of new tasks and exception tables. Below is a sample call of this SP and the details of its usage:

CALL EDM\_REFINED\_DEV.SCRATCH.SIMULATE\_FALURE('ESED\_GROCERYORDER\_R\_TASK,SP\_GETGROCERYORDER\_TO\_BIM\_LOAD\_TASK,ESED\_OFFEROMS,ESED\_OFFEROMS\_R\_TASK,SP\_OFFEROMS\_TO\_BIM\_LOAD\_TASK,ESED\_STOREGROUP\_R\_TASK',false,false)

The first parameter is the taskname or tablename (if you are simulating failure for exception out tables), the second parameter indicates whether you are simulating failure/success for task or exception out table - false is for task and true is for exception - and the last parameter indicates success or failure - false is for success and true is for failure.

The first parameter can take one or more taskname/tablename. If more than one, they should be separated with comma. Please also note that taskname parameter must be in all capital letter.

## Python Scripts and Automation Job Configuration

Below are the details of all automated processes currently running in production:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Process** | **Python Script** | **Snowflake SP/UDF** | **Linux FilePath** | **Schedule(s)** |
| High Volume Bod Alerts | HighVolumeBodAlert.py | PROC\_DATA\_MONITORING\_BODS | /var/tmp/edm\_dataops/monitoring/HighVolumeBodAlert.py | Every 3 hours |
| Daily Monitoring Alerts | DailyAlert.py | ALERT\_LOADDATA, ALERT\_TASKDATA, GET\_MONITORING\_BODDATA | /var/tmp/edm\_dataops/monitoring/DailyAlert.py | 11AM and 8PM EST Daily |
| Exception History Tracker | ExceptionTracker.py | TRACKEXCEPTIONS | /var/tmp/edm\_dataops/monitoring/ ExceptionTracker.py | 9PM MT Daily |
| Ganesh Alert | GaneshAlert.py | N/A | /var/tmp/edm\_dataops/GaneshAlert.py | 6AM PST Daily |

The AppD Linux server currently hosting these processes is: zduwitdseddw001.albertsons.com and the dataops team members can connect with their LDAP credential.

If you make any changes to any of these python scripts, then you need to redeploy and reschedule the job. To do that, first get the corresponding processID from the processID excel document available in shared location, then kill it in shell terminal. After that redeploy your script, and then reschedule it by calling the below command:

nohup python3 [scriptname] &

That command will return a new processID, take that processID and replace the killed one in the processIDs excel doc.