**Environment Manager**

**Installation and Usage Guide**

**Author: Ian Fickling**

Table of Contents

[1 Introduction 3](#_Toc85435953)

[1.1 Installation 3](#_Toc85435954)

[1.2 Installation Pre-Requisites 4](#_Toc85435955)

[1.2.1 Snowflake Python Connector 4](#_Toc85435956)

[1.2.2 Pandas 4](#_Toc85435957)

[2 Environment python scripts 5](#_Toc85435958)

[2.1 Configuration 6](#_Toc85435959)

[3 Spreadsheet – Config Tab 7](#_Toc85435960)

[3.1 Config: TYPE 7](#_Toc85435961)

[4 Script Details and Runtime parameters 9](#_Toc85435962)

[4.1 Build Environment 9](#_Toc85435963)

[4.1.1 Restrictive Mode 9](#_Toc85435964)

[4.1.2 Permissive mode 11](#_Toc85435965)

[4.1.3 Script Parameters 12](#_Toc85435966)

[4.2 Synchronize Environment 13](#_Toc85435967)

[4.3 Clone Environment 14](#_Toc85435968)

[4.4 Destroy Environment 15](#_Toc85435969)

# Introduction

Creating and managing security within a snowflake environment is not a simple process and is subject to the cause of several issues. It’s a fundamental that environments are configured correctly. To help make like easy ‘Role Base Access Control Manager’ has been developed.

RBAC Manager is a python utility and is used in conjunction with a spread sheet. The spread sheet defines environments and the scripts can be used to generate the DDL that and be executed to create, modify, clone, sync or destroy a given environment.

The purpose is to provide Snowflake customers with a simple way to define environments and to provide confidence they are created using best practice methods.

The spreadsheet utilises Snowflake’s reference architecture and is used to define the following objects:

* Databases
* Schemas
* Access Roles
* Functional Roles
* Privileges for Access Roles. (ie. Schema to Access Role)
* Grants of Access to Functional Roles

## Installation

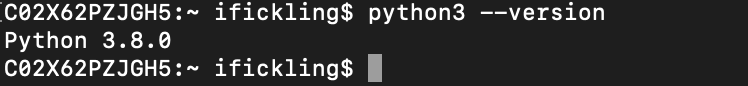
Installation of the tool consists of 2 steps

* Pre-Requisites
* Download DDL migration scripts.

## Installation Pre-Requisites

Ensure python 3.8.0 or higher is installed on your workstation. Details for installing and downloading python can be found at <https://www.python.org/downloads/>

Once python is installed, verify the version. E.g.



### Snowflake Python Connector

<https://docs.snowflake.net/manuals/user-guide/python-connector-install.html>

Note: OpenSSL must also be installed when using the python connector. Snowflake doc describes openSSL install for mac and linux. If using windows see <https://www.xolphin.com/support/OpenSSL/OpenSSL_-_Installation_under_Windows>

### Pandas

The pandas python library is required for RBAC manager. The following command is show is this library is present.

Version 1.2.5 or higher is required

Text

Description automatically generated

If the library is not present issue the command



This will install pandas and any pre-requisites.

# Environment python scripts

The scripts can be located at <https://drive.google.com/drive/u/0/folders/1SmlzypxUe_cce3ygwMQRpQoaXfDgOtLg>

Included with the scripts is a sample configuration file. The next section describes the config file.

The python scripts are:

* sfBuildSecutity.py Used to build the security layer for one or more environments.

This requires the snowflake user be granted the SECURITYADMIN and SYSADMIN roles.

The following scripts do not require access to SECURITYADMIN or SYSADMIN.

* sfBuild.py Used for building an environment for the first time.
* sfSync.py Used to validate and synchronize an environemnt
* sfReplicate.py Used to replicate from one environment into another.
* sfDestroy.py Used to remove an environment. USE CAUTION with this - it is destructive.
* sfUsers.py Used to create users in snowflake.

The following python files are classes used throughout the application.

* SFConfig.py Helper class for reading the snowflake config file.
* SFDBCalls.py Helper class containing for database calls
* SFParams.py Helper class for process parameters
* SFPrompt.py Helper class for providing prompts to user

Sample configuration files:

* RBACManager.xlsx The configuration spreadsheet
* sfConf.txt Snowflake config file. Contains connection details and is optional.

## Configuration

An optional Snowflake configuration file which holds connection details for snowflake. This file is only required if the -r option is “True” – meaning, execute the DDL.

The following is an example of the configuration options.

Note: If password is not present, a prompt to enter the password will be given.

# Snowflake

account=aws\_cas1

user=ifickling

password=xxxxxx

role=securityAdmin

warehouse=if\_wh\_small\_0

# Spreadsheet – Config Tab

The spreadsheet contains a configuration tab. This tab is used to define the keys used in the database tabs and provides some addition options

The config tab provides four columns

* TYPE
* KEY
* VALUE
* Comment

## Config: TYPE

Valid types are

* COMMENT

This plays no role in the configuration itself, it’s a comment that describes the page

* DB\_PRIVS

This type is only valid when ‘permissive’ mode is used. It defines database level access settings and can differ to that in the SCHEMA\_PRIVS if required.

This is followed by KEY and VALUE.

* + KEY – This is the abbreviation used in the database TABS. E.g. SR = Read
  + VALUE – For each object type, the valid snowflake privilages.

E.g. Schema(Usage):Tables(Select):Stages(Usage,Read):File Formats(Usage):Streams(Select):Functions(Usage):Procedures(Usage)

* SCHEMA\_PRIVS

This type is only valid for permissive override and ‘RESRTICTIVE’ mode.. It defines database level access settings and can differ to that in the SCHEMA\_PRIVS if required.

This is followed by KEY and VALUE.

* + KEY – This is the abbreviation used in the database TABS. E.g. SR = Read
  + VALUE – For each object type, the valid snowflake privilages.

E.g. Schema(Usage):Tables(Select):Stages(Usage,Read):File Formats(Usage):Streams(Select):Functions(Usage):Procedures(Usage)

* WAREHOUSE\_PRIVS

This is valid only for snowflake virtual data warehouses.

Similar in behaviour to SCHEMA\_PRIVS and DB\_PRIVS, only that the privileges defined to relevant to virtual warehouse permissions.

E.g.

Key=WFULL

VALUE= Warehouse(Usage, Operate,Modify)

* ENVIRONMENT\_ROLE
  + With KEY=SECURITY\_ADMIN, the value is used to define the role admin for the given environment
  + With KEY=SYSTEM\_ADMIN, the value is used to define the system admin for the given environment
* DB\_OPTION
  + Allows for DB Level options
* GRANT\_OPTION
  + These are reserved. Do not modify
* ACCESS\_ROLE\_PREFIX
  + Defines a prefix for access roles. A value of ‘NONE’ implies no prefix

E.g. ‘\_’

* ACCESS\_ROLE\_SUFFIX
  + Defines a suffix for access roles. A value of ‘NONE’ implies no prefix

E.g. ‘\_’

* ROLE\_TYPES

There are three role types

* + ACCOUNT\_ONLY
    - This will generate an RBAC script based on traditional account level roles.

E.g. Account Level Access Roles -> Account Level Functional Roles

* + DBR\_FUNC\_ACC
    - This will generate an RBAC script based on having database access roles and database functional roles as well as account level functional roles.

E.g. Database Level Access Roles -> Database Level Functional Roles -> Account level functional roles

* + DBR\_ACC\_ONLY
    - This will generate an RBAC script based on having database access roles and account level functional roles.

E.g. Database Level Access Roles -> Account level functional roles

* INCLUDE\_ENV\_IN\_DB\_ROLE

There are two possible values, TRUE or FLASE. This setting is only used when generating database level roles.

* + TRUE: The environment name is included as part of the database role names
  + FALSE: The environment name is not included in the database role names.
* INCLUDE\_ENV\_IN\_SDR\_ROLE

There are two possible values, TRUE or FLASE. This setting is only used when generating secure data roles. SDR roles are generated as account level roles

* + TRUE: The environment name is included as part of the SDR role names
  + FALSE: The environment name is not included in the SDR role names.

# Script Details and Runtime parameters

This section provides details of the run time parameters for each python script.

As a general rule, when first becoming familiar with the scripts, do **not** use the option ‘-r TRUE as this connects to Snowflake and immediately executes the script. Instead, execute the generated script manually once you are confident of the result

## Build Environment

**Script:** sfBuild.py

This python script is used to build a specific environment from the spread sheet.

sfBuild.py -s <spreadsheet.xlsx> -e < EnvTabName > -o <security\_script.sql> [ -c <config\_file> ] -r TURE|FALSE

There are two modes for the build script

* RESTRICTIVE
* PERMISSIVE

This value is specified on the spread sheet.

### Restrictive Mode

This provides security access at schema level and both current grants and future grants are generated. In restrictive mode, there must be a spreadsheet tab for each database created

This allows permissions to be set at schema level for each functional role. Thus, a given functional role can have different permissions against different schemas.

The following shows what is built for restrictive

Table

Description automatically generated

The following diagram shows the grants in place for the function role DM\_DEVELOPER for schemas DW and DM. The purpose of omitting some schemas from the diagram is to ensure it does not become cluttered.

Also, note that the grants are for both future and current.

Chart

Description automatically generated

### Permissive mode

This provides security access at database level, only future grants are required. The database is created and then ‘Grant on future <object type> in database’ is issued. This ensures all objects created fall under the ‘future’ permissions.

In permissive mode, the spreadsheet tab behaves as a template and the database name is passed in as a parameter in the build script.

Table

Description automatically generated

The difference here is that there is only one developer role and all schema’s have ‘N’ in their cell. In this example, the DEVELOPER functional role will have read/write permissions in all schema’s.

The exception to this rule is the schema type ‘Schema\_O’, this implies ‘schema override’. Schema’s defined under schema override behave in the exact same way as restrictive mode.

The following diagram shows what is built for permissive

Chart

Description automatically generated with medium confidence

As can seen above, the access roles no longer contain the schema name.

Every object becomes a ‘future’ object so there is no need to provide permissions for current grants.

In the case for schema ‘WRK’, the behaviour for this schema follows the same pattern as defined in ‘RESTRICTIVE’

### Script Parameters

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Mandatory** | **Description** |
| -s | --spreadsheet= | Yes | The path of the spreadsheet. This can be an absolute path or a relative path |
| -e | -- EnvTabName = | Yes | The target environment. This must be the same as the label(s) on the tab in the spreadsheet. Only one environment can be specified here. |
| -o | --outscript= |  | Is the script output. If specified a script containing the DDL for the security layer is generated. If -r False is used, the -o must be specified. |
| -r | --run= |  | TRUE/FALSE:  TRUE – Connect to Snowflake and immediately execute the script. FALSE - Otherwise.  Default - FALSE. |
| -c | --config= |  | Identifies the configuration file for parameters.  If -r=TRUE then the configuration file must be provided. |
| -d | database= |  | This parameter if only for restrictive mode only. It specifies the name of database to be built. |

For Example:

* **Spreadsheet:** env\_manager\_control.xlsx
* **Spreadsheet Tab:** TEMPL\_R
* **Database / Environment Name:** DEV
* **Output Script:** /users/jryan/myPython/020\_build\_env.sql
* **Action:** Creates the Database, Schemas, Access Roles, Functional Roles and all Grants

Python3 ./code/sfBuild.py -s ./env\_manager\_control.xlsx -e TEMPL\_R -o ./020\_build\_prod\_env.sql -d DEV -r FALSE

## Synchronize Environment

**Script:** sfSync.py

This python script will begin by comparing the spread sheet to what has been defined in the database and compare what is in the database to what is defined in the spreadsheet.

The next stage is to build the DDL to ensure the database matches the definition of the spreadsheet.

It is assumed that the spreadsheet is the source of truth.

sfSync.py -s <spreadsheet.xlsx> -e < EnvTabName > [ -o <security\_script.sql> ] [ -c <config\_file> ] -r TURE|FALSE -d <database\_name>

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Mandatory** | **Description** |
| -s | --spreadsheet= | Yes | The path of the spreadsheet. This can be an absolute path or a relative path |
| -e | -- EnvTabName = | Yes | The target environment. This must be the same as the label(s) on the tab in the spreadsheet. Only one environment can be specified here. |
| -o | --outscript= |  | Is the script output. If specified a script containing the DDL for the security layer is generated. If -r False is used, the -o must be specified. |
| -r | --run= |  | TRUE/FALSE:  TRUE – Connect to Snowflake and immediately execute the script. FALSE - Otherwise.  Default - FALSE. |
| -c | --config= |  | Identifies the configuration file for parameters.  If -r=TRUE then the configuration file must be provided. |
| -d | --database= | Yes | The database name |

For Example:

* **Spreadsheet:** Env\_manager\_control.xlsx
* **Database:** DEV
* **Tab Name:** TEMPL\_R
* **Output Script:** ./030\_sync\_prod\_env.sql
* **Action:** Generates a script to refresh the database based upon the spreadsheet

Python3 ./code/sfSync.py -s Env\_manager\_control.xlsx -e TEMPL\_R -o ./030\_sync\_prod\_env.sql

## Clone Environment

**Script:** sfClone.py

This python script is used to clone an environment. If target environment already exists, this script will fail to execute.

For cloning a ‘from’ environment is required and a ‘to’ environment is required. The ‘from’ environment will be cloned and the permissions specified in the ‘to’ environment will be configured.

All schema’s in the from environment will be cloned into the ‘to’ environment, even if the ‘to’ environment does not specify the schema. In this case, the ‘to’ environment should be synchronized using sfSync.py.

sfClone.py -s <spreadsheet.xlsx> -e <target\_environment> -f <from\_environment> [ -o <security\_script.sql> ] [ -c <config\_file> ] -r TURE|FALSE

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Mandatory** | **Description** |
| -s | --spreadsheet= | Yes | The path of the spreadsheet. This can be an absolute path or a relative path |
| -e | -- ToEnvTabName = | Yes | The target environment. This must be the same as the label(s) on the tab in the spreadsheet. Only one environment can be specified here. |
| -d | --database | Yes | The name of the target database |
| -f | --envFrom | Yes | The environment that is being cloned |
| -o | --outscript= |  | Is the script output. If specified a script containing the DDL for the security layer is generated. If -r False is used, the -o must be specified. |
| -r | --run= |  | TRUE/FALSE:  TRUE – Connect to Snowflake and immediately execute the script. FALSE - Otherwise.  Default - FALSE. |
| -c | --config= |  | Identifies the configuration file for parameters.  If -r=TRUE then the configuration file must be provided. |

For Example:

* **Spreadsheet:** env\_manager\_control.xlsx
* **From Environment Name:** EDW\_PROD
* **To Environment Name:** EDW\_TEST
* **To Database Name:** TESTDB
* **Output Script:** ./040\_replicate\_env.sql
* **Action:** Generates a script to clone the database plus the roles

Python3 ./code/sfClone.py -s env\_manager\_control.xlsx -f EDW\_PROD -e EDW\_TEST -d TESTDB -o ./040\_replicate\_prod\_test.sql

## Destroy Environment

**Script:** sfDestroy.py

**Warning:** The script produced is destructive and will completely destroy an environment including the Database, Schemas, Access and Functional roles. As a result, although this script may connect to the database, it will not execute the script immediately. This must be run manually.

sfDestroy.py -s <spreadsheet.xlsx> -e <target\_environment> -o <security\_script.sql>

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Mandatory** | **Description** |
| -s | --spreadsheet= | Yes | The path of the spreadsheet. This can be an absolute path or a relative path |
| -e | --envTo= | Yes | The target environment. This must be the same as the label(s) on the tab in the spreadsheet. Only one environment can be specified here. |
| -o | --outscript= | Yes | Is the script output. If specified a script containing the DDL for the security layer is generated. If -r False is used, the -o must be specified. |
| -c | --config= | Yes | Database to connect to. |

For Example:

* **Spreadsheet:** env\_manager\_control.xlsx
* **From Environment Name:** EDW\_PROD
* **Output Script:** ./060\_destroy\_env.sql
* **Config File:** ./env\_manager\_config.cfg
* **Action:** Generates a script to drop the environment.

Python3 ./code/sfDestroy.py -s env\_manager\_control.xlsx -e EDW\_PROD -o ./060\_destroy\_test\_env.sql -c ./env\_manager\_config.cfg