# PoCs

## dbt with Airflow

In this PoC I am running dbt code in Airflow. That dbt code is making transformations on data in a MS SQL server. Everything is running in Docker containers on our local machine.

There is a folder ‘dbt with Airflow PoC’ in the same folder as this document containing a PowerPoint presentation and videos exaplaining this PoC.

# dbt setup

Here is a link showing how to install and configure dbt: [www.youtube.com](https://www.youtube.com/watch?v=1fY1A8SRflI&list=PLc2EZr8W2QIBegSYp4dEIMrfLj_cCJgYA&index=3).

What we need to do is (we need to execute all the below commands in the directory with the dbt project):

* create venv: py -m venv venv
* install python libraries in that venv:
  + pip install dbt-core
  + pip install dbt-sqlserver (here we need to chose a porper adapter depending on what database we are using)
* create .dbt folder in the user directory, for example in the C:\Users\<username>: mkdir $home/.dbt
* Initialize dbt using the 'dbt init' command and provide the following values when prompted:
  + database: sqlserver
  + host name: name of the sql server
  + port: 1433
  + user, password, database: For those value we can use the same values which we provided when creating a SQL db using Terraform and sql\_db module from azure\_terraform repository.
  + threads: 4 is recommended, but other value can be used as well.

The dbt init command at the end will create the profiles.yml file in the .dbt folder which specifies our dbt configuration. It will also create a new folder with the name of our specified project name. We need to modify the profiles.yml file manually at the end so it contains the following content:

A screenshot of a computer program

AI-generated content may be incorrect.

Then we can run the 'dbt debug' command in the project folder in order to test if it is set up properly.

# dbt functionalities

## Nodes

Node is a general term describing a single object in dbt. We have different kinds of nodes:

* models – tables which we will be building.
* Snapshots - for creating slowly changing dimensions type 2.
* Macros - functions which we can use in other nodes.
* Tests – data quality tests.

## Snapshots

Snapshots are used for creating slowly changing dimensions type 2.

In the snapshots folder in our dbt project we can define snapshots either as YAML file (a new method) or as SQL files.

## Macros

Macros are functions which we can use in other nodes. They are defined in the macros folder in our dbt project.

## Schemas

We can create files called schema.yml which are describing our models, there are defined data quality test, tables and columns descriptions.

In the models folder we can create a few subfolders, each containing different models, and then in each of those folders we can create a schema.yml file describing models in that folder:

A screenshot of a computer

AI-generated content may be incorrect.

## Sources

We can create the sources.yml file which describes tables which are sources, that is they are not created by dbt. Just like in the schema.yml we are defining there data quality test, tables and columns descriptions.

We can have such a file created in the models folder and then it can be used by all the models, like it is shown on the screenshot in the ‘Schemas’ section above.

## dbt project file

We need to create a dbt\_project.yml file which defines settings which applies to the entire dbt project. For example we are defining there where in the database we will be saving created models. It is created automatically when creating a dbt project.

## Profiles file

We need to create the profiles.yml file which defines how we will be connecting to the SQL server. It is created automatically when creating a dbt project.

## Targets

In the profiles.yml file we can specify different targets, that is different databases where we will be building models. For example here we are specifying two targets called ‘dev’ and ‘prod’:

A screenshot of a computer screen

AI-generated content may be incorrect.

When we are building models using dbt commands we can specify in which target we want build our models:

A screen shot of a computer

AI-generated content may be incorrect.

## Selector patterns/methods

When running dbt we can select which nodes (models, snapshots) will be ran. More info about that: [docs.getdbt.com](https://docs.getdbt.com/reference/node-selection/methods).

When running dbt in a command line we are using it in the following way:

Dbt run –select “pattern”

For example this command will build all the snapshots:

Dbt snapshot –select “snapshot:”

And this will build a specific model:

Dbt run –select “model:model\_name”.

**Excluding**

In the same way we can use the –exclude option to exclude specific nodes from running.

# cosmos functionalities

Cosmos is a tool for running dbt nodes in Airflow. Here are links to the documentation:

* [www.astronomer.io](https://www.astronomer.io/docs/learn/airflow-dbt/)
* [astronomer.github.io](https://astronomer.github.io/astronomer-cosmos/getting_started/open-source.html)

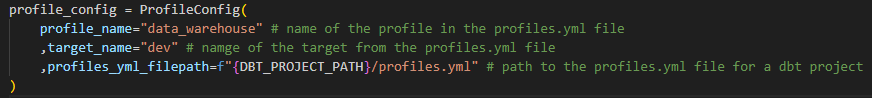
## dbtTaskGroup and dbtDag

The cosmos.dbtDag function creates the entire Airflow DAG for our dbt project.

The cosmos.dbtTaskGroup function create a task group in Airflow DAG for our dbt project.

## ProfileConfig

We need to create the ProfileConfig object where we provide path to the profiles.yml file and specifying the target:



Then we are using this profile config object in the dbtTaskGroup or dbtDag functions:

A black screen with white text

AI-generated content may be incorrect.

## ProjectConfig

We need to create the ProjectConfig object where we provide a path to the dbt project and pass it to the dbtTaskGroup or dbtDag functions:

A screen shot of a computer

AI-generated content may be incorrect.

## ExecutionConfig

We need to create the ExecutionConfig object where we provide a path to the dbt executable.

It is a good practice to create a virtual environment for dbt, install there all the dbt libraries needed (dbt-core, dbt-databricks) and provide the following executable path:

A black screen with white text

AI-generated content may be incorrect.

And pass it to the dbtTaskGroup or dbtDag:

A black background with white text

AI-generated content may be incorrect.

This way dbt libraries will be kept separate to other libraries used by Airflow what is beneficial as those libraries might be in conflict.

## Rendering options for dbtTaskGroup

For every task group created using the dbtTaskGroup we can define which dbt commands will be executed, for example we can only build models, or snapshots, or only perform tests.

We configure that using the render\_config argument and the RenderConfig() function in the following way:

A screenshot of a computer program

AI-generated content may be incorrect.

Here are example arguments for the RenderConfig function:

* **select=["selector"]** - Use a node selector in order to build only a specific nodes (models, snapshots). For example a selector can be "path:" to build all the nodes from a folder at a given path (like 'models' or 'snapshots') or "model\_name" to build a specific model.
* **exclude=["selector"]** - We are using here node selectors like in the select=["selector"] but it excludes specific nodes.
* **should\_detach\_multiple\_parents\_tests=True** - This argument makes sure that when we have models dependent on each other then we will not execute the same tests twice and when tests depends on a multiple models, then they will be performed when all those models have been built. Otherwise we might perform test which includes a model which has not been built yet and it will fail.
* **test\_behavior="after\_all"** - This argument causes that all tests will be performed after all the models have been built (there will be a single task in Airflow for testing).

## Variables for dbt commands

If we want to use variables for running dbt commands, like 'dbt run --vars', then we can do it in the following way:

A screenshot of a computer program

AI-generated content may be incorrect.

## Custom schema macro

The purpose of the custom\_schema.sql macro is to create models in schemas specified in the dbt\_project.yml file. Otherwise when running dbt with cosmos it might use as a schema name a name obtained by contatenating schema names from profiles.yml and dbt\_project.yml.

## Snapshots

### Metadata columns names problem

When we want to change metadata columns names for snapshots there is some problem and it doesn't work. I was trying both snapshots defined as a YAML file and SQL.