Contents

Notes on Hands-on Workshop of ingest nodes

These notes should be read in conjunction with the session recording.

Agenda

- Review of last session and context setting for this one
  - Previous session was focussed on the "Why", today we will focus on the "How".
- Build your own standard DBT project: DBT-based ingest node "Dummy Jaffle_".
- Addressing the feedback questions
- Demonstrating how to share data with Unity Catalog
- Demo of DBT project with automate_dv DBT macro extension: "Dummy Bridgetown".

We will demonstrate aspects of:

- DBT-based ingest node
  - best suited for OLAP use case

  - can ingest data through `dbt seed`
  - can ingest data from cloud storage through `COPY INTO` statement
  - can model data using DBT models
  - can use macro packages such as automate_dv to simplify data vault building

Definitions

Quick setup instructions forOSX (Mac)

To get set up: * Docker Desktop. Choose Apple Chip/Intel depending on your Mac and install. * `brew install git` if you haven't yet * `xcode-select --install` if you haven't installed Xcode command line tools yet (Might take a while to download/install) * Install cookiecutter: run `export PATH=$HOME/.local/bin:$PATH` and then `brew install cookiecutter` * For pulling docker images from Azure Container Registry we need to authenticate via the Azure CLI, so please install: `brew update && brew install azure-cli` * Generate Databricks token, in User Settings -> Access Tokens

Cookiecutter run

- All of the answers below to the Cookiecutter questions are obtained from ADO and Databricks.
- The ADO config file is something you have to setup in your home directory.
- The ADO config file is only required for the ADO Pipeline automation, at the time when you want to setup the build, release and schedule pipelines.

```
cookiecutter git@ssh.dev.azure.com:v3/exploreai/CORE.Utilities/CORE.Meshnodes.BaseTemplates/
ingest_node_name [Ingest Node Template]: dummy_jaffle_kr
az_devops_repo [https://dev.azure.com/exploreai/CORE.Utilities/_git/CORE-POC1]: https://dev.
Select data_substrate:
1 - databricks
2 - postgres
Choose from 1, 2 [1]: 1
databricks_workspace [https://adb-891777510264692.12.azuredatabricks.net/]: https://adb-4472
database_host [adb-891777510264692.12.azuredatabricks.net]: adb-4472170994427587.7.azuredata
databricks_sql_warehouse_path [/sql/1.0/endpoints/a077556ed384ed67]: /sql/1.0/warehouses/22a
databricks_catalog [null]: dummy_jaffle
database_user [psqladmin]:
database_port [5432]: 443
database_threads [4]: 5
profile_name [data_vault]:
azure_storage_container_url [wasbs://adf-pipeline-demo@datavalidation.blob.core.windows.net/
ado_pipelines_build_agent_pool_name [CORE Pipelines]: Default
ado_pipelines_folder_name [CORE]: dummy_jaffle
ado_profile_name [CORE]: DEFUALT
scheduled_release_cron [0 6 * * *]:
```

```
----------
Template clone successful
----------

--> Attempting to move pipeline definitions to relevant location...
         --> Moving files...
         --> File move successful.
--> Searching for ADO config file in home dir...
? ADO config file detected. Would you like to alter/extend it? No
User declined to modify ADO config file at /home/kerneels/.adocfg. Skipping file generation

--> Generating `.env` file from supplied definition file...
         --> Generation successful.
         --> Please inspect/edit the generated file at `dummy_jaffle/.env` to ensure that th
```

Update your `.env` file

- Set your PAT for `ACCESS_TOKEN`.
- Set a SAS token as `AZURE_SAS_TOKEN` if you intend to read from blob storage. Ensure the token has read and list privileges.

Copy an existing DBT project to use for now

```
cp -fr dummy_jaffle/src/ dummy_jaffle_<initials>/
```

Adjust the project and model name to match your dummy_jaffle_ name in the `dbt_project.yml` file

- Find the `dbt_project.yml` file in `dummy_jaffle_<initials>/src/` folder.
- Edit it and rename `dummy_jaffle` to `dummy_jaffle_kr` to match your new project.

Source aliases and log into ACR, pull images

```
cd dummy_jaffle_<initials>/
source .bash_aliases
ingest-acr-login
ingest-pull
ingest-build
```

Check that you are able to connect to the Databricks SQL Warehouse

```
ingest-dbsqlcli
```

Check that `dbt debug` succeeds

```
ingest-debug
```

Optionally change the log file format to TEXT since we are not pushing logs already and this is more readable

This is just editing the `.env` file in place with `sed`:

```
sed -i 's/DBT_LOG_FORMAT=JSON/DBT_LOG_FORMAT=TEXT/g' .env
```

From here you can try out any of the modalities:

- `ingest-seed` - to load any seed data via `dbt seed`
- `ingest-debug` - see if everything is setup properly via `dbt debug`
- `ingest-dbsqlcli` - connect to the Databricks Serverless SQL Warehouse or Standard Cluster where you can query using SQL (exit using the keyword `exit`)
- `ingest-exec` - run the entire DBT project (all models) via `dbt run`
- `ingest-test` - run all the tests via `dbt test`
- `ingest-docs` - generate documentation and serve it on http://localhost:8080/
- `ingest-cli` - enter into the running docker container from where you can issue ANY DBT command
- `ingest-other` - specify variable `MODE_SWITCH_OTHER` with ANY DBT command you would like to issue

Feedback questions

How do we specify the source and sink tables for a given transformation?

- The smallest unit of a transformation is a DBT model (we will just call it model onwards).
- A model is essentially something that takes data from one or more inputs and produce one output.
- The input data for a model is one of these:
  - any existing data already ingested/transformed perhaps by other systems - we call these sources and they are referenced via the `{{ source() }}` function.
    - note that we have to define them in YML files so DBT knows about them
  - any data ingested via seeds referenced via the `{{ ref('model_name') }}` function.
  - any other model in the DAG referenced via the `{{ ref('model_name') }}` function.
- The output of a model is determined by the type of materialisation configured, and it can be a table or a view.

    - We do not explicitly “save” into a table, but let DBT decide when it should persist or not

AR: What are some of the best resources for us to learn about DBT?

- Given these already but will list them no problem!

AR: Can we see a simple example of a simple data transformation from end to end?

AR: Can we expand on the simple example, by chaining two simple transformations together using pipelines.

Observability

- We can emit logs to Azure Log Analytics Workspace (LAW).
- Ensure all the LAW-related variables are set.
- If some of the LAW-related variables are not set, the system will inform you of this and no logs can be sent.
- The log data goes here.
- Take a look at this dashboard built off of the data.
- you can alter it and query the log data using KQL.

These notes should be read in conjunction with the session recording.