# KT Session 1 : (15/11/22)

# Infrastructure as Code

**Terraform Folder Structure**

* Data-science-infra
  + .github
  + Terraform

***a. Live***

Develop

**data.csv** - client data for timestamp table

**Inventory.json**

**item\_sales**.json includes schema for both iv and is tables

**Load.json** schema for mo\_methods\_timestamp

**Main.tf** main.tf will **contain the main set of configuration for our module**.(in this we have commented out resources like gke\_cluster as they are already migrated to prod env)

**Variables.tf-** includes definition of all variables required in the resource creation

**Outputs.tf -** You can use this data to configure other parts of your infrastructure.Outputs are also necessary to share data from a child module to your root module. We have implemented this

**Terragrunt.hcl** the main purpose of using terrgraunt.hcl is it becomes easy to deploy the resources to higher environments. Terragrunt helps us to keep our configuration DRY. In this case, we only need to come and update some values for specific env and then it gets deployed to higher environment

***B. Modules (Custom)***

**prerequisite**

Enabling prerequisite APIs

**Monitoring**

This module enables us to set up the monitoring dashboard for composer and bigquery and also add alerting as well as this alerts will be send to the slack notification channel and we have configured this using custom modules

**Terraform Code**

For terraform code we have used CFT modules( which recommends us the gcp best practices).

We have created few resources like

1. VPC with one subnet
2. Service Account with roles as BigQuery Data Editor, BigQuery Job User and composer worker we also need to provide BigQuery DataViewer role to the composer SA for datalake project
3. Composer - here for composer\_sa we are using the service account created above as outputs.tf which means this will share data from one module to another
4. BQ dataset for MO tables - for\_each is used as for loop here as we use it other programming language, this will execute and create the bq dataset along with the list of clients.
5. BQ Load job dataset - One time creation , Jobs are actions that BigQuery runs on your behalf to load data, export data, query data, or copy data. Once a BigQuery job is created, it cannot be changed or deleted.we have created one bq dataset and two table (mo\_methods\_timestamp and data\_validation\_for\_all\_clients) we have also created a bq storage bucket and bucket object in which we have globally unique name for bucket and then in bq load job we will give the source uri from where we need to load the data that is the storage bucket object path to the destination tables
6. BQ dataset for materialized view here we have created a separate dataset to store this item\_sales and inventory tables

**Deployment to Staging or Production**

For Infrastructure deployment we are using the same github actions workflow implemented by TakeOff, the terragrunt.hcl has the DRY configuration which will require just few changes to deploy from stage to prod.

# CI/CD

-------------------------------------------------

pipeline triggers

-------------------------------------------------

- creating a pull request for both pipelines triggers a dry run

- if the PR is approved and merge to master is made, it is actually deployed

for terraform:

- changes to the terraform/live/ folder for the specific environment

for DAGs:

- changes to the dsm/ folder

- for deployment to higher environments, the config file for that specific environment should be modified

-------------------------------------------------

environment variables being passed:

-------------------------------------------------

- environment name based on trigger

- whether to apply based on event type and branch

- PR details for passing to sonarqube

-------------------------------------------------

stages of DAGs:

-------------------------------------------------

- based on sonarqube workflow shared during discovery

- explain pytest stage

- explain sonar scan stages + quality gate requirements

- getting composer bucket name from state file

- removes old files and upload new files to composer bucket

-------------------------------------------------

rollback:

-------------------------------------------------

- go to PR history and click on "Revert Pull Request"

- this creates a new branch and pull request to revert the changes merged in that particular request

- the same deployment pipelines for tf and dags are used

- if multiple deployments have to be reverted, care needs to be taken to revert them in the same order

# KT Session 2 : (17/11/22)

# Data lake project repository, resources created

For this we have referred the [**Delivering Data to Data Lake**](https://engineering-handbook.takeofftech.org/docs/centers-of-excellence/data-lake/add-data-to-datalake/)documentation. In this we have used **daas-data-lake-sub-dataflow-bigquery repository.**

1. **Adding data source to Data Lake**

This includes creation of source configuration in Data Lake repository, so that Data Lake automation workflows will validate the configuration and create relevant infrastructure on Data Lake side (like, Pub/Sub schema, Pub/Sub topic, Dataflow pipeline, BigQuery table) to accept data for the source and write it to BigQuery.

Lets consider we need to add a new source (mo\_demand\_forecast) with a BigQuery table called mo\_demand\_forecast which does not exist yet for the client **ahold**

Navigate to the folder conf/ahold and then add appropriate configurations in config.yaml file(**open config.yml for ahold**)

| * **client\_id** - this is client name. Only allowed in lowercase letters and underscores. * **enabled** - so this states either to create infra for the client or not * **dataset\_id** - target BigQuery dataset to store tables for specific client. Naming convention is datalake\_<client\_name>. Allows lowercase letters and underscores.   sources: **# This contains list of sources with their schema version.**  mo\_demand\_forecast\_v1:  source\_name: mo\_demand\_forecast **# Source name. Allows lowercase letters and underscores.**  schema\_version: v1 **# Schema version.**  configuration: mo\_demand\_forecast\_v1.json **# Pointer to source schema configuration file within conf/<client\_name>/sources directory.**  owner: quantiphi-ds-team@takeoff.com **# email of data owner team and for now its Quantiphi**  tables: **# This dict contains mapping of source names to specific tables in BigQuery.**  mo\_demand\_forecast: **# Should be the same as sources.source\_name**  table\_id: mo\_demand\_forecast  **# Target table name in BigQuery.** |
| --- |

Within **conf/ahold/sources** directory create a new file named **mo\_demand\_forecast\_v1.json**. The file name should be the same you mentioned in **sources.mo\_demand\_forecast\_v1.configuration.**

**Target BigQuery table schema will be created based on this file**

**(Now show mo\_demand\_forecast\_v1.json)**

* **schema\_name -** should be the same as sources.mo\_demand\_forecast\_v1.source\_name in **config.yaml**
* **schema\_description -** should contain relevant description of the data source. This will be added as a description to BigQuery target table.
* **schema\_version -** should be the same as sources.mo\_demand\_foreacst\_v1.schema\_version in config.yaml
* **labels -**You can add any relevant labels here
* **fields** - Everything under fields will be as schema for BigQuery table.

**Within fields, you must provide several mandatory/required fields - \_\_client\_name, \_\_source\_name, \_\_schema\_version, \_\_event\_timestamp. You will not be able to commit schema without these mandatory fields since PR validation will fail.**

(**now move to protobuff file)**

Next , we will have to generate the protobuff file to validate the JSON schema.If validation failed, then adjust either JSON schema for data (i.e. fix **sources/mo\_demand\_forecast\_v1.json** file) and generate new Protobuf schema out from fixed JSON schema and then repeat validation again.

Once validation succeeded, then we can remove Protobuf schema

Then we are good to commit the changes.

* + **Multiplying configuration to all clients**

Once source configuration is thoroughly tested for one client (in this sample case, Ahold), then we can multiply configuration for all clients.

Multiplying configuration for all clients means duplicating configuration to other client directories under **conf/** directory.

To deploy it to staging env we have used the develop branch as per the msg from **team-mario**

—------------------------------------------------------------------------------------------------------------------

# Access Required , Snowflake Steps, Snowflake Objects

The Agenda for todays meet for Historical Data Migration is to Cover

1. **Access Required**
2. **Migration Steps**
3. **Snowflake Resources**

1.**Access Required:**

To migrate Historical data migration from snowflake to BigQuery. We

Required Read Access to those Databases and Tables and write access to Create

**STORAGE** **INTEGRATION , FILE FORMATS,STAGE OBJETS,** soTAKEOFF provided one Service Account with required Role .

2. **Migration Steps:** To Migrate or Unload data from Snowflake  **to** BigQuery We Required

**1.** **STORAGE INTEGRATION**

**2. FILE FORMAT**

**3.STAGE OBJECTS**

**4. COPY INTO COMMAND**

3. **Snowflake Resources:**

1. **STORAGE INTEGRATION**:

Storage Integration is a Snowflake Object that stores a generated identity and Access Management(IAM) Entity for your External Cloud Storage , along with an optional set of Allowed (or) Blocked Storage Locations.

However Storage Integration We used the existing integration which **TAKEOFF Team** already created and used for earlier Migration.

2. **FILE FORMAT :**

File Format that Describes a Set of staged data to Access (or) load, into Snowflake Tables are Unload from snowflake tables

**Syntax:**

| **CREATE OR REPLACE FILE FORMAT my\_json\_unload\_format**  **TYPE = json**  **compression = gzip;**  **COMMENT = 'FILE FORMAT FOR UNLOADING AS JSON NEWLINE DELIMITED FILES';** |
| --- |

3**.STAGE OBJECT:**

This **INTERNAL (or) EXTERNAL** Stage to Use for Loading data from Files into Snowflake Tables and Unloading data from tables into files.

**INTERNAL STAGE:**

Store data files internal within the Snowflake

**EXTERNAL STAGE:**

External Stage of data files Stored in a Location outside of snowflake . Currently 3 Cloud Storage services are Supported .

1**.AWS**

**2. GCP**

**3. Microsoft Azure**

**Syntax:**

| **CREATE OR REPLACE STAGE my\_gcs\_unload\_stage**  **URL = 'gcs://prj-daas-n-stg-dl-sub-bq-b703-historical/WINGS/'**  **storage\_integration = gcs\_historical**  **FILE\_FORMAT = my\_json\_unload\_format**  **COMMENT = 'GCS Stage for the Snowflake external JSON export';** |
| --- |

3. **COPY INTO COMMAND:**

The Snowflake COPY INTO COMMAND Allows you to Load data from Staged Files on External Locations to an Existing table (or) Vice versa.

**Syntax:**

| **COPY INTO @my\_gcs\_unload\_stage**  **from (**  **select OBJECT\_INSERT(OBJECT\_INSERT(OBJECT\_INSERT(OBJECT\_INSERT(OBJECT\_INSERT(OBJECT\_INSERT(OBJECT\_INSERT(OBJECT\_INSERT(KEYS\_TO\_SNAKE(REMOVE\_NULLS(doc)), 'extra', TO\_VARCHAR(doc:extra), true),**  **'\_\_client\_name', 'ALPHA', true),**  **'\_\_source\_name', 'mo\_methods', true),**  **'\_\_schema\_version', 'snowflake\_historical', true),**  **'\_\_event\_timestamp', TO\_VARCHAR(created), true),**  **'\_\_publish\_timestamp', null, true),**  **'\_\_ingestion\_timestamp', null, true),**  **'method\_selection\_ts', TO\_TIMESTAMP(replace(DOC:method\_selection\_ts::varchar, '+', '.')), true)**    **from MO\_METHODS**  **--where created >= to\_timestamp('2021-10-01 00:00:00') and created < to\_timestamp('2021-12-01 00:00:00')**  **)**  **HEADER = FALSE;** |
| --- |

# KT Session 5 : (29/11/22)

# Pytest

Link : - <https://docs.pytest.org/en/7.2.x/how-to/unittest.html#unittest>

Link : - <https://docs.pytest.org/en/7.2.x/>

The pytest framework makes it easy to write small, readable tests, and can scale to support complex functional testing for applications and libraries

* Pytest’s detailed assertion introspection, only plain assert statements are used

# Sonar property variable

Sonar property variable need to set the parameter based on sonarqube UI

sonar.projectKey=TakeoffTech\_data-science-infra

sonar.projectName=data-science-infra

sonar.projectVersion=1.0

sonar.sources=./dsm/dags

sonar.tests=./tests

sonar.sourceEncoding=UTF-8

sonar.python.coverage.reportPaths=./coverage.xml

# Sonar scan

Sonar does static code analysis, which provides a detailed report of bugs, code smells, vulnerabilities, code duplications.

It supports 25+ major programming languages through built-in rulesets and can also be extended with various plugins.

# Sonar quality gate

Set of conditions against which projects are measured.

For example:

* No new blocker issues
* Code coverage on new code greater than 80%