Data Engineering Project: Toronto Covid-19

Introduction:

This project was created as part of DataTalks' Data Engineering zoomcamp final assignment. The tools used are:

- Cloud: GCP
- IaC: Terraform
- Workflow Orchestration: Prefect
- Data Warehouse: BigQuery
- Transformation: DBT

I really enjoyed learning about data engineering as it has help me grow my programming skills as well as my knowledge about various tools. Working on this project helped me improve my debugging skills as well as self-learning from docs. I am thankful to the instructors of this zoomcamp as well as the Slack community for helping me whenver I got stuck. This was my third time trying out a zaamcamp by DataTalks and I am glad I was able to work till the end as the previous two times I had to drop out due to academic commitments. Thank you 😃

Problem Description:

I have chosen the COVID-19 cases in Toronto dataset for this project as the pandemic is still ongoing and the results from the analysis is something everyone can learn from to help each other. The dataset is updated on a weekly basis.

The first case of COVID-19 in Toronto was reported in January 2020 and since then the virus is been monitored along with its mutation and the kind of experience the patients of various demographics have gone through.
The focus of this project is to identify which groups had the most number of cases as well as to identify the most common source of infection and other information. The analysis performed can help in protecting the vulnerable groups as well as understand how to restrict the spread of the virus.

Data Description:

The final dataset after the transformation has the following columns:

- Assigned_ID: "A unique ID assigned to cases by Toronto Public Health for the purpose of posting to Open Data, to allow for tracking of specific cases." (Source)
- Age_Group: Age of the person at the time they got infected
- Client_Gender: Gender of the person reported by themselves
- Neighbourhood: Neighbourhoods in Toronto
- Postal_District: The first 3 characters of postal code
- Outbreak_Associated: Outbreaks associated with COVID-19
- Classification: Is the case confirmed to be COVID-19 case or it's just a possibility
  = Source: Source from where COVID-19 was possibly acquired
- Episode_Date: The earliest date the virus was acquired

- Reported Date: The date the case was reported on to Toronto Public Health
- Delay_in_Reporting: Number of days between Episode_Date and Reported_Date
- Outcome: Describes if the patient died, recovered or still has the virus
- Ever_Hospitalized: Cases that were hospitalized due to COVID-19
- Ever_in_ICU: Cases admitted to ICU due to the virus
- Ever_Intubated: Cases that had a tube inserted for ventilation due to COVID-19
  All the columns starting with "Ever_" include cases that are currently hospitalized, deceased or discharged. The Delay_in_Reporting column was not provided in the original dataset; it had to be created.

Replication:

In order to replicate this project, you need a GCS account. You can run this project locally. Assuming you have anaconda installed, create a virtual environment using `conda create -n <envname> python=3.9 anaconda` where the envname can be anything you want. I chose python 3.9 as it is recommended version.

GCS & Terraform

- Assuming you still have the 3-month trial ongoing or a credit card linked to your account, create a new project exclusively for the project (You can continue with an ongoing project if you wish to but to avoid confusion I created a new one).
- Next, go to "IAM & Admin > Service Account". Create a service account (name can be of your choice). While creating, grant the account viewer role as a way to access the project. The user access part can be skipped. Your service account is create.
- Go to Actions and click on "Manage keys > Add Key > Create New Key". Save the key as a JSON file to your project directory.
- Open your project directory in terminal (I recommend Git Bash if you have the option). Run the command `export GOOGLE_APPLICATION_CREDENTIALS="<path/to/your/service-account-authkeys>.json"` (The *<path/...authkeys>* is a placeholder). Then run `gcloud auth application-default login` (if prompted, type Y). We have now our local setup authenticated with the Cloud SDK via OAuth.
- Go to "IAM & Admin > IAM" on GCS. for the service account you created, click on "Edit principal" and grant the roles: Storage Admin + Storage Object Admin + BigQuery Admin.
- Enable the APIs: Identity and Access Managment and IAM Service Account Credentials.
- Assuming you have terraform installed, we proceed to create GCP Infrastructure. Use the files from my repo. You can change the variables "data_lake_bucket" and "BQ_DATASET" to your choice.
- Run the commands `terraform init` to initialize a state, then `terraform plan` and finally `terraform apply` to make and apply changes to the cloud. In `terraform plan` and `terraform apply`, you will have to provide your GCS project ID (de-project-akshar in my case).

We have created a data lake now where we will store our data there.
Note: if you feel stuck anywhere, you can watch the video 1.3.1 Introduction to Terraform Concepts & GCP Pre-Requisites and 1.3.2 Creating GCP Infrastructure with Terraform as the steps are similar.

Prefect + Docker: Workflow Orchestration

In your virtual environment, run the command `pip install -r requirements.txt` to install the necessary packages. In a new terminal, activate the same environment and run `prefect orion start`. In your first

terminal, run `prefect config set PREFECT_API_URL=http://127.0.0.1:4200/api`.
We can now begin orchestration via Prefect. For that, we need to create some blocks to reuse configurations and work with GCP. We can do this either through code or through UI. You can use blocks from prefect_blocks folder to create your own blocks with your own credentials (the key you made with your service account).

- If you are using code to create blocks, fill the service_account_info with your key and then run `refect block register -m prefect_gcp` then `python blocks/make_gcp_blocks.py`.
- If you are using UI to create blocks, the process is similar to 2.2.3 ETL with GCP & Prefect. Creating a block for bigquery is very much same.
- In the `web_to_gcs.py` file, in the `write_gcs` function, write the name of your GCS bucket block in line 30. After that, you can run the file via command `python web_to_gcs.py` to upload data from the site to GCS Bucket.
- In the `gcs_to_bq.py` file, in the all the functions that load a bigQueryWarehouse block, write the name of your bigquery bigquery block in the load method. In the main function call, you can uncomment line 80-81 if you wish to create a partition-only table as well to compare with other tables. Run the file via `python gcs_to_bg.py` to upload data from GCS to BigQuery.

The following steps are for deploying the pipelines and require Docker. If you have the external tables in BigQuery and would like to move on to DBT, you can skip to the next sections.
We will be setting schedules via cron so here is a quick summary of how to read them:

- To deploy the pipeline Web to GCS, run the command `prefect deployment build flows/web_to_gcs.py:etl_parent_flow -n "Web to GCS" --cron "0 1 * * 4"`. The part `web_to_gcs.py:etl_parent_flow` specifies the entrypoint. The name of the deployment will be "Web to GCS". The cron command helps us to set the deployment to run every Thursday 1:00 AM. The data is updated weekly ohence we have setup a weekly schedule.
- In the YAML file, Specify the parameters for months and years `{"months": [1,2,3,4,5,6,7,8,9,10,11,12], "years":[2020, 2021, 2022, 2023]}`. Also, instead of doing this manually, you can add `--params='{"months":[1,2,3,4,5,6,7,8,9,10,11,12], "years": [2020, 2021, 2022, 2023]}'` to the build command.
- To apply the deployment, run the command `prefect deployment apply etl_parent_flow-deployment.yaml` to apply the deployment.
- We will do similar with the GCS to Bigquery pipeline. Run the command `prefect deployment build flows/gcs_to_bq.py:etl_gcs_to_bq -n "GCS to BQ" --params='{"years":[2020, 2021, 2022, 2023]}' --cron "0 2 * * 4" -a --skip-upload`. The pipeline will run an hour after the pipeline for moving data web to GCS. The flag `-a` is to apply the deployment simultaneously with building it.
- Note: you may get a warning regarding no files to be uploaded. You can add the flag `--skip-upload` to avoid the warning.
- (Optional) To trigger running a deployment, we need an agent which can be done by the command `prefect agent start --work-queue "default"`.

We will now run our flows via Docker instead of running it locally.

- Make sure you have the Docker files, i.e., Dockerfile, docker-deploy.py and docker-requirements.
- Store all prefect-related files in a folder like we did in the zoomcamp. Store `docker_deploy.py` in it as well.

- We will start by building the image `docker image build -t <docker-username>/prefect:<tagname> .` The *docker-username* and *tagname* are placeholders which you will have to replace with your own details. The '.' at the end of the command is not a mistake so be careful not to skip that. In case you face any error, check if you have logged in to Docker via `docker login`.
- Run the command `docker image push <username>/prefect:<tagname>` to see your image in Docker.
- We will now create a block for docker using the `make_docker_block.py` file. Replace the placeholder for the image param in line 5 with your docker image name you created. You can give your name for the docker block in the save method. Run `python make_docker_block.py` to make the docker block.
- In the `docker_deploy.py` file, specify your docker-block name when loading in line 7. In the methods build_from_flow, feel free to specify your name ("docker-flow-web2gcs" and "docker-flow-gcs2bq" in my case) so you can find them in deployments. You can also specify your own cron schedule if you would like.
- Run `python docker_deploy.py` to see your docker flow deployments on Orion.
- If you want to run the deployments, start an agent via command `prefect agent start -q default`. Then run the command `prefect deployment run etl-parent-flow/docker-flow-web2-gcs` to run the deployment in a Docker container. You can check the run in the Deployment's Run section of Orion UI.

Note: in case you get stuck, the video from zoomcamp on deployment can help you understand the deployment better.
Additional Note: I really enjoyed learning about Prefect!

DBT

As my DBT trial account got over and I could not build more than one project in Developer plan, I decided to go out of my comfort zone and try dbt locally. In the same virtual environment, run `pip install -U dbt-core dbt-bigquery`. We can now use dbt-core. The docs really helped understand dbt-core better.
If you still would like to replicate but via dbt-cloud, just replace the folders with my folders respective to their names (example: replace models with models). Same goes for the files.
You may have to setup a profile.yml in ~/.dbt folder in case you face an error. You can get .dbt folder via command `dbt init` and then following along the instructions in the docs. Here how your profile.yml may look like:

```
dbt_de_project:
  outputs:
    dev:
      dataset: toronto_covid_data_dbt
      job_execution_timeout_seconds: 300
      job_retries: 1
      keyfile: <path-tp-gcp-keyfile>.json
      location: northamerica-northeast2
      method: service-account
      priority: interactive
      project: <project-name-on-GCP>
      threads: 1
      type: bigquery
  target: dev
```

`dbt_de_project` is the name of my dbt project and `toronto_covid_data_dbt` is the name of the dataset that dbt will create in BigQuery. You may also need a GitHub repo to sotre your dbt project.

- In the `dbt_project.yml`, provide your own profile name and name of the project if you wish to use your own profile and your project name.
- In the `schema.yml`, you will have to provide your own GCP project details in the sources section.
- Run `dbt run` to create dataset `toronto_covid_data_dbt` and table `toronto_covid_dbt` in BigQuery. If you make any changes, run `dbt run --full-refresh` to drop the original table and create a new table.
- Run `dbt test` to test all tests I have written in models/productions/schema.yml.

Initially, my plan was to create a view which had all the transformed columns and using that view, I would create a table via `select * from <table_name>`. In the end, I decided to create the table directly.

Via DBT, I have created a new column called Delay_in_Reporting which calculates the difference between Episode_Date and Reported_Date. I have changed few column names such as FSA to Postal_District to provide more clarity. I have written some unit tests to ensure my columns have the appropriate values. Also, my models retain the partitioning as well as the clustering column from the tables.

We will now dive into deploying this. To deploy this, we will need dbt cloud.

- Create a project on the cloud and provide the github repo you have created for the dbt project. In the account/project settings, hoose BigQuery as your connection and import your gcp-key.json there to fill in your details. Fill "northamerica-northeast2" in Location. In the Artifacts, choose "Production Run" in the Documentation (this can be done after the next step).
- Create an environment called Production. the environment type needs to be deployment Type. Dataset, in my case. would be toronto_covid_data_dbt. We will now create a job.
- Click on Create Job. Choose Production as environment. In Execution settings, make sure to check "Generate docs on run". In the commands, we should have `dbt run --full-refresh` and `dbt test`. For the triggers, we will set schedule via cron: "0 3 * * 4" which is every Thursday at 3 AM.

Now your DBT will create the models in the BigQuery every Thursday 3 AM. Using the models generated, we can use them to answer questions we have about the data.

Data Visualizations

I have used Looker Studio for Data Visualization. You can find the link to the report here.