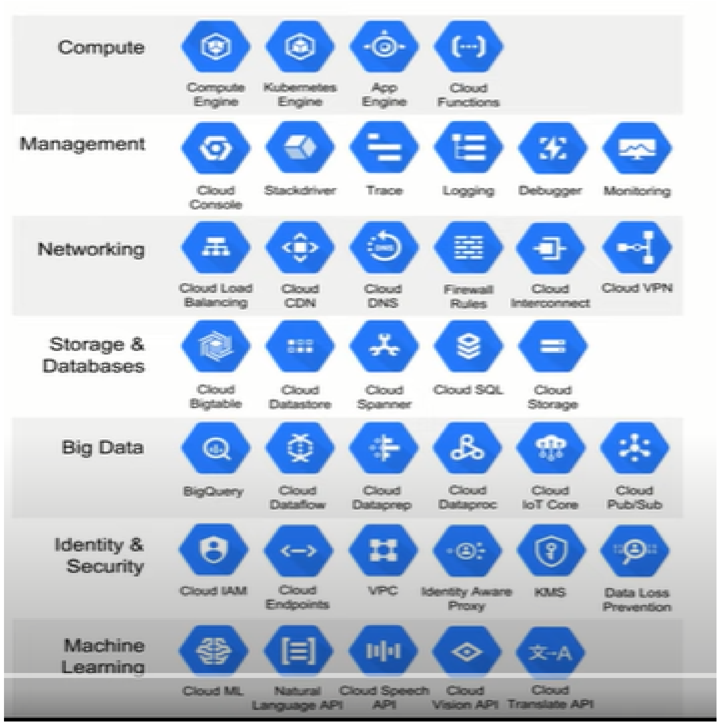
WEEK-1 Basics and Setup

Google Cloud Services



Installing Python on Ubuntu:

https://phoenixnap.com/kb/how-to-install-python-3-ubuntu

Step 1: Update and Refresh Repository Lists

Open a terminal window, and enter the following:

```
sudo apt update
```

Step 2: Install Supporting Software

The software-properties-common package gives you better control over your package manager by letting you add PPA (Personal Package Archive) repositories. Install the supporting software with the command:

```
sudo apt install software-properties-common
```

Step 3: Add Deadsnakes PPA

Deadsnakes is a PPA with newer releases than the default Ubuntu repositories. Add the PPA by entering the following:

```
sudo add-apt-repository ppa:deadsnakes/ppa
```

The system will prompt you to press enter to continue. Do so, and allow it to finish. Refresh the package lists again:

```
sudo apt update
```

Step 4: Install Python 3

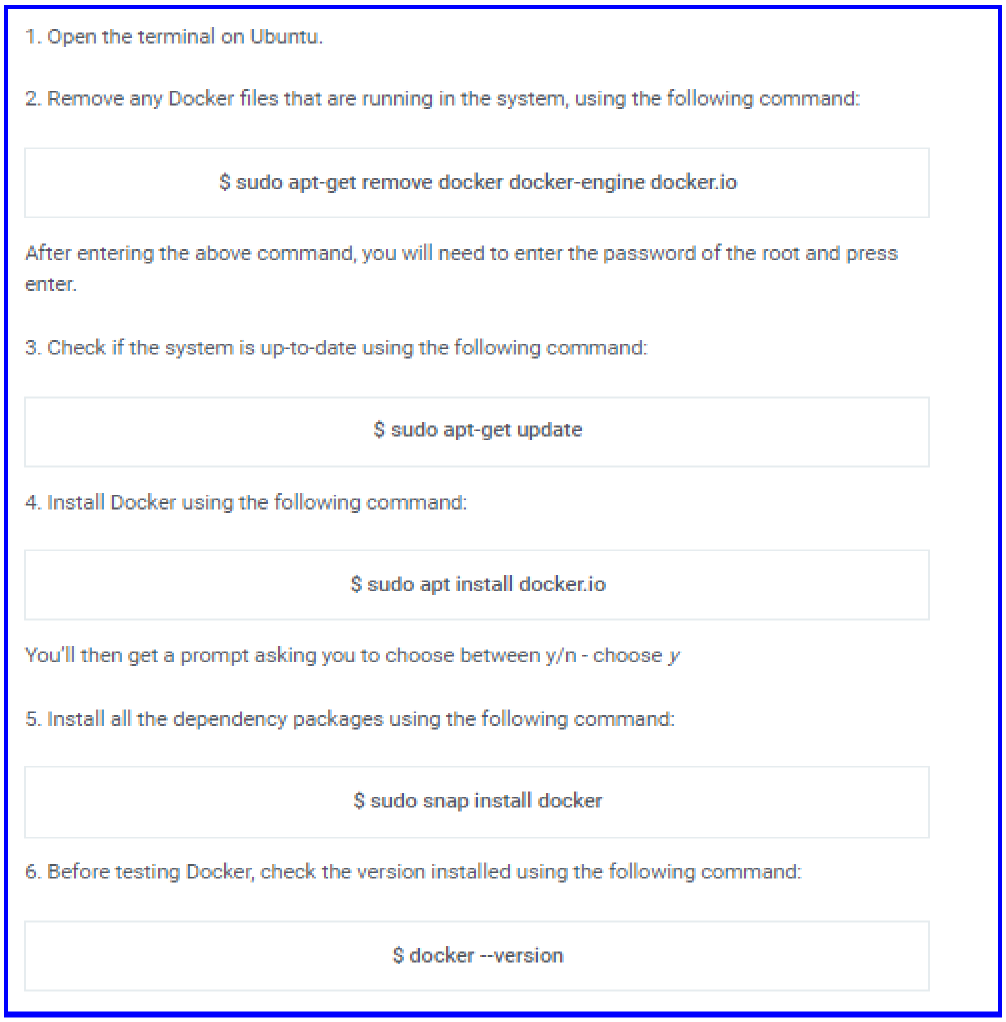
Now you can start the installation of Python 3.8 with the command:

```
sudo apt install python3.8
```

Setting Python3 as default compiler

```
sudo update-alternatives --set python /usr/bin/python3.8
python  - - version
```

Installing Docker on ubuntu



1. Open the terminal on Ubuntu.

2. Remove any Docker files that are running in the system, using the following command:

```
$ sudo apt-get remove docker docker-engine docker.io
```

After entering the above command, you will need to enter the password of the root and press enter.

3. Check if the system is up-to-date using the following command:

```
$ sudo apt-get update
```

4. Install Docker using the following command:

```
$ sudo apt install docker.io
```

You'll then get a prompt asking you to choose between y/n - choose *y*

5. Install all the dependency packages using the following command:

```
$ sudo snap install docker
```

6. Before testing Docker, check the version installed using the following command:

```
$ docker --version
```

Executing Docker Commands as Non-root user

```
sudo usermod -aG docker $USER
```

```
docker ps
```

Install Anaconda on Ubuntu

1- Follow the steps from the blog
https://phoenixnap.com/kb/how-to-install-anaconda-ubuntu-18-04-or-20-04

2- To use gui (anaconda navigator)

```
export PATH=/home/yourUserName/anaconda3/bin:$PATH
```

```
anaconda-navigator
```

Installing Sublime text editor

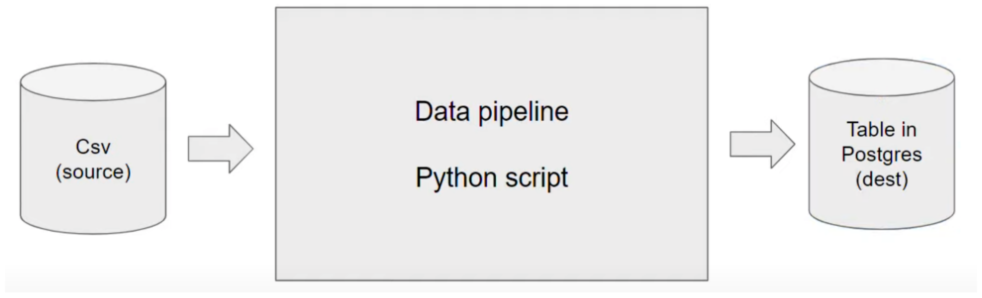
https://linuxize.com/post/how-to-install-sublime-text-3-on-ubuntu-20-04/

Data Engineering Basics

Data engineers are responsible for consuming this data, designing a system that can take this data as input from one or many sources, transform it, and then store it for their customers / end-users to analyze at scale.

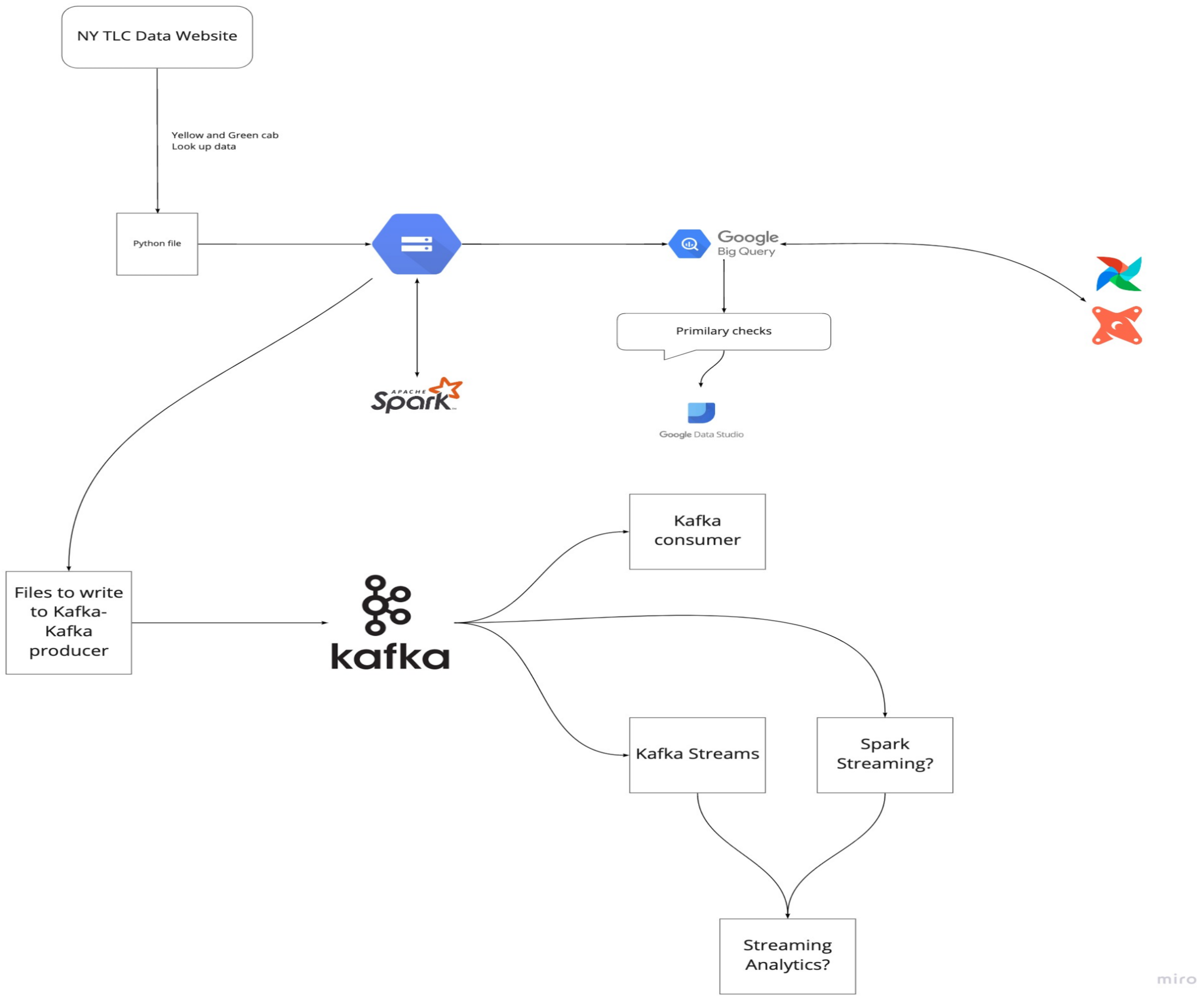
Data Pipeline

A data pipeline is a means of moving data from one place (the source) to a destination (such as a data warehouse). Along the way, data is transformed and optimized, arriving in a state that can be analyzed and used to develop business insights.



Course Outcome:

Designing an end-to-end data pipeline to automate data flows.



Technologies to be used:

- *Google Cloud Platform (GCP)*: Cloud-based auto-scaling platform by Google
  - *Google Cloud Storage (GCS)*: Data Lake
  - *BigQuery*: Data Warehouse
- *Terraform*: Infrastructure-as-Code (IaC)
- *Docker*: Containerization
- *SQL*: Data Analysis & Exploration
- *Airflow*: Pipeline Orchestration
- *DBT*: Data Transformation
- *Spark*: Distributed Processing
- *Kafka*: Streaming

2_Docker_Sql

Docker is a container management service. The keywords of Docker are develop, ship and run anywhere. The whole idea of Docker is for developers to easily develop applications, ship them into containers which can then be deployed anywhere.

- Github link: data-engineering-zoomcamp/week_1_basics_n_setup/2_docker_sql at main · DataTalksClub/data-engineering-zoomcamp (github.com)
- Videos playlist: Data Engineering Zoomcamp - YouTube

Reference tutorial: Docker Tutorial: Get Going From Scratch – Stackify

Running docker images:

```
docker run -it ubuntu bash

docker run -it python:3.8
```

Installing pandas package in python container:

```
docker run -it entrypoint=bash python:3.8

pip install pandas

python
```

Example-1 with dockerfile:

pipeline.py:

```
import pandas as pd

# some transformations

print("job completed successfully")
```

Dockerfile: (Please note that the name of the file has to be "Dockerfile" with "D" as capital)

```
FROM python:3.9

RUN pip install pandas

WORKDIR /app
COPY pipeline.py pipeline.py

ENTRYPOINT [ "bash" ]
```

To run dockerfile:

```
docker build -t test:pandas .
```

Run the image:

```
docker run -it test:pandas
```

Example-2 with dockerfile:

pipeline.py:

```
import sys

import pandas as pd

print(sys.argv)

day = sys.argv[1]

# some fancy stuff with pandas

print(f'job finished successfully for day = {day}')
```

Dockerfile:

```
FROM python:3.9

RUN apt-get install wget
RUN pip install pandas sqlalchemy psycopg2

WORKDIR /app
COPY pipeline.py ingest_data.py

ENTRYPOINT [ "python", "pipeline.py" ]
```

To run dockerfile:

```
docker build -t test:pandas .
```

Run the image:

```
docker run -it test:pandas 2022-01-18
```

Running a PostgreSql on Docker:

```
docker run -it \
  -e POSTGRES_USER="root" \
  -e POSTGRES_PASSWORD="root" \
  -e POSTGRES_DB="ny_taxi" \
  -v $(pwd)/ny_taxi_postgres_data:/var/lib/postgresql/data \
  -p 5432:5432 \
  postgres:13
```

➢ postgres container defined above has following:
  - - e for environment variables
  - "ny_taxi" is the name given to the DB
  - -v for volumes (storage)
  - Format is -v (path in host machine):(path in container)
  - - p for ports in host:container
  - Note: folders in host should be created before running the command

CLI for PosgtreSql:

```
sudo apt-get install libpq-dev python3.8-dev
sudo pip install pgcli
```

Connect to DB

```
pgcli -h localhost -p 5432 -u root -d ny_taxi
```

Working with Jupyter Notebooks

- Cheatsheet: Jupyter_Notebook_CheatSheet (edureka.co)
- Get the source file
  wget https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2021-01.csv
- More info on fields data_dictionary_trip_records_yellow.pdf (nyc.gov)
- We use pandas to import dataset, transform and load to target. More info on pandas can be found at Python-Pandas-Cheat-Sheet.png (1365×768) (intellipaat.com)
- Jupyter Notebook used can be found at git-hub url

Connecting pgAdmin and Postgres with Docker networks

- **web-based GUI tool used to interact with the Postgres DB sessions**
- Also used for database administration.

Download pgAdmin docker container at Download (pgadmin.org) or follow instructions to pull the container from docker hub dpage/pgadmin4 - Docker Image | Docker Hub.

To create a docker virtual network:

```
docker network create pg-network
```

To run the pgAdmin:

```
docker run -it \
  -e PGADMIN_DEFAULT_EMAIL="admin@admin.com" \
  -e PGADMIN_DEFAULT_PASSWORD="root" \
  -p 8080:80 \
  --network=pg-network \
  --name pgadmin \
  dpage/pgadmin4
```

To login:

Go to localhost:8080 >> enter email and password

Note:

PostgresSql DB and pgAdmin are both running in different docker containers. So we need a virtual docker network to connect these client-server apps.

To run the container in virtual network:

```
docker run -it \
  -e POSTGRES_USER="root" \
  -e POSTGRES_PASSWORD="root" \
  -e POSTGRES_DB="ny_taxi" \
  -v $(pwd)/ny_taxi_postgres_data:/var/lib/postgresql/data \
  -p 5432:5432 \
  --network=pg-network \
  --name pg-database \
  postgres:13
```

Now to login and connect to DB:

Go to localhost:8080 >> enter email and password >> Give name to this connection >> Enter Server Credentials (Host name / Address), (Username, Password, Port) >> Use query editor to run queries.

Dockerizing the Ingestion script:

To convert the .ipynb notebook to .py file

```
jupyter nbconvert --to=script upload-data-prac.ipynb
```

Note: drop table yellow_taxi_data; before running ingestion script as we recreate it in script.

Few key-points in ingest_data-prac.py file:

- The standard Python library **argparse** is used to incorporate the parsing of command line arguments. More on this library here

Run the script:

- If there is no “pandas” library installed locally, run the command with the prefix ‘!’ on Anaconda Jupyter.

```
python ingest_data-prac.py \
   --user=root \
   --password=root \
   --host=localhost \
   --port=5432 \
   --db=ny_taxi \
   --table_name=yellow_taxi_trips \
   --url="https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2021-01.csv"
```

To Dockerize the script (modify Dockerfile):

```
FROM python:3.9.1

# We need to install wget to download the csv file from S3 bucket
RUN apt-get install wget
# psycopg2 is a postgres db adapter used by sqlalchemy for python
RUN pip install pandas sqlalchemy psycopg2

WORKDIR /app
COPY ingest_data-prac.py ingest_data.py

ENTRYPOINT [ "python", "ingest_data.py" ]
```

Build the image and Run it:

```
docker build -t taxi_ingest:v001 .

docker run -it \
   --network=pg-network \
   taxi_ingest:v001 \
   --user=root \
   --password=root \
   --host=pg-database \
   --port=5432 \
   --db=ny_taxi \
   --table_name=yellow_taxi_trips \
   --url="https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2021-01.csv"
```

Note:

- Dropping the table before running the container as the script is optional (replace option in code).
- Before the name of the image, we specify the name of the network.
- As we have pgAdmin already on same network, we can connect to DB from container
- In the arguments here, we are pointing to
  (hostname, db_name, tbl_name = pg_database, ny_taxi, yellow_taxi_trips)

Useful docker commands:

- To stop all running containers: `docker kill $(docker ps -q)`
- To remove all containers: `docker rm $(docker ps -a -q)`
- To remove all images of containers: `docker rmi $(docker images -q)`

Running Postgres and pgAdmin with Docker Compose:

- Instead of running pgAdmin and Postgresql engine from separate terminals, commands;; we use "docker compose" to have them running in one command.
- **Docker Compose** is a tool for running multi-container applications on Docker defined using the Compose file format.
- Tutorial Docker Compose Tutorial: advanced Docker made simple (educative.io)
- In the yaml file, we don't specify network because all services/containers run by docker-compose can communicate with each other by "name of service" specified.

docker-compose.yml

```
services:
  pgdatabase:
    image: postgres:13
    environment:
      - POSTGRES_USER=root
      - POSTGRES_PASSWORD=root
      - POSTGRES_DB=ny_taxi
    volumes:
      - "./ny_taxi_postgres_data:/var/lib/postgresql/data:rw"
    ports:
      - "5432:5432"

  pgadmin:
    image: dpage/pgadmin4
    environment:
      - PGADMIN_DEFAULT_EMAIL=admin@admin.com
      - PGADMIN_DEFAULT_PASSWORD=root
    ports:
      - "8080:80"
```

Useful docker-compose commands:

- To stop: `docker compose down`
- To run in detached mode: `docker compose up -d)`

Note:

If there are any errors (eg: "docker-compose : Unsupported config option for services service") in running the docker-compose, it means the version used is not supporting the way we defined yml code. To resolve it, Just add `version: "3"` to the start of your docker-compose.yaml file.

SQL Refresher

- Code file at link
- Video at link
- Tutorial SQL Tutorial - An Ultimate Guide for Beginners (tutorialrepublic.com)

1_TERRAFORM_GCP

Course Notes: data-engineering-zoomcamp/week_1_basics_n_setup/1_terraform_gcp at main · DataTalksClub/data-engineering-zoomcamp (github.com)

Download Terraform:

```
curl -fsSL https://apt.releases.hashicorp.com/gpg | sudo apt-key add -

sudo apt-add-repository "deb [arch=amd64] https://apt.releases.hashicorp.com $(lsb_release -cs) main"

sudo apt-get update && sudo apt-get install terraform
```

Create Project in GCP:

A project organizes all your Google Cloud resources. A project consists of a set of users; a set of APIs; and billing, authentication, and monitoring settings for those APIs. So, for example, all of your Cloud Storage buckets and objects, along with user permissions for accessing them, reside in a project.

Note: project Id should be unique in entire gcp

Create Service Account in GCP:

- A service account is a special type of Google account intended to represent a non-human user that needs to authenticate and be authorized to access data in Google APIs.
- More tutorial link

Service accounts are similar to user accounts in that they are identifiable by a unique email address and they are used for making authorized API calls, but there are some differences b/w the two:

- User accounts authorize people, service accounts authorize applications.
- Service accounts do not have passwords. Therefore, it is not possible for someone to log in using a service account via a browser.
- Service accounts are not members of your Google Workspace domain. This means that assets shared with your Google Workspace domain are not shared with service accounts.

Steps to create Service Account:

.

1. IAM > Service accounts > Create
2. Provide a service account name. Grant Basic > Viewer role for now > Skip adding users
3. When the service account is created, click on it > Keys tab > Add Key.
4. Add key > Create new key > Select JSON and click Create. The Private key will be downloaded to your computer.

Download GCP SDK:

Google Cloud SDK is a flexible application to access Google Cloud via terminal. To download the SDK, follow steps mentioned at Install Google Cloud SDK on Ubuntu 20.04 – VITUX

After installation >> Open new Terminal window >> Enter command "`gcloud -v`" to verify the installation!

Lastly, set environment variable to point to your downloaded GCP keys >> Refresh the token >> Verify the authentication with the GCP SDK with commands below:

```
export GOOGLE_APPLICATION_CREDENTIALS="<path/to/your/service-account-authkeys>.json"
```

```
gcloud auth application-default login
```

GCP Setup for Access:

We will use Google Cloud Storage as Data Lake and BigQuery as DWH.

- ➢ IAM Roles for Service account:
  - ➢ Go to the IAM section of IAM & Admin https://console.cloud.google.com/iam-admin/iam
  - ➢ Click the Edit principal icon for your service account.
  - ➢ Add these roles in addition to Viewer : Storage Admin (buckets) + Storage Object Admin (objects in buckets) + BigQuery Admin (BigQuery)

- ➢ For Terraform to interact with GCP, we need to enable these APIs for your project:
  - ➢ https://console.cloud.google.com/apis/library/iam.googleapis.com
  - ➢ https://console.cloud.google.com/apis/library/iamcredentials.googleapis.com

- ➢ Please ensure GOOGLE_APPLICATION_CREDENTIALS env-var is set.

```
export GOOGLE_APPLICATION_CREDENTIALS="<path/to/your/service-account-authkeys>.json"
```

Creating GCP infrastructure with Terraform

Tutorial for Terraform: Introduction to Terraform | Baeldung

Key points from tutorial:

- A Terraform project is just a set of files in a directory containing resource definitions. Those files, which by convention end in .tf, use Terraform's configuration language to define the resources we want to create.
- The main.tf file contains two blocks: a provider declaration and a resource definition.
- init, plan, and apply:
  - In **init** step, Terraform scans our project files and downloads any required provider
  - **plan** command to verify what actions Terraform will perform to create our resources
  - we proceed to actual resource creation using the **apply** command

Basics of Terraform: dataeng-zoomcamp/1_intro.md at main · ziritrion/dataeng-zoomcamp (github.com)

Files in Terraform:

- main.tf   - variables.tf   - Optional: resources.tf, output.tf   -.tfstate

Components in Terraform:

- Code files   - Terraform commands

Declarations:

- terraform: configure basic Terraform settings to provision your infrastructure
  - required_version: minimum Terraform version to apply to your configuration
  - backend: stores Terraform's "state" snapshots, to map real-world resources to your configuration.
    - local: stores state file locally as terraform.tfstate
  - required_providers: specifies the providers required by the current module
- provider:
  - adds a set of resource types and/or data sources that Terraform can manage
  - The Terraform Registry is the main directory of publicly available providers from most major infrastructure platforms.
- resource
  - blocks to define components of your infrastructure
  - Project modules/resources: google_storage_bucket, google_bigquery_dataset, google_bigquery_table
- variable & locals
  - runtime arguments and constants

Execution steps:

1. terraform init:
   - Initializes & configures the backend, installs plugins/providers, & checks out an existing configuration from a version control
2. terraform plan:
   - Matches/preview local changes against a remote state, and proposes an Execution Plan.
3. terraform apply:
   - Asks for approval to the proposed plan, and applies changes to cloud
4. terraform destroy
   - Removes your stack from the Cloud

Exercise:

- We are creating GCP resources like Storage Bucket and BigQuery as part of the course project.
- Terraform will read the **GOOGLE_APPLICATION_CREDENTIALS** from system environment variables, added in the GCP SDK set up. So skip specify credentials in the provider block.
- Navigate to the folder in which we have main.tf and variables.tf and enter following commands:

```
terraform init
terraform plan
```

– – here it'll ask for project id that's not specified in code

```
terraform apply
```

– – here it'll ask whether or not to apply changes ??? enter yes