16/01

① Prerequisites : Python, SQL, Command line

② Dataset : NY Taxi trips dataset

③ Syllabus

→ Week 1 : Intro & Prerequisites

→ Week 2 : Workflow orchestration

→ Week 3 : Data Warehouse

→ Wak 4 : Analytics Engineering

→ Week 5 : Batch processing

→ Week 6 : Stream processing

→ Project : Putting everything we learned in practice

④ Tech Stack

① GCP < Google Cloud Storage : Data Lake
Bigquery : Data Warehouse

② Terraform : Infrastructure as Code

③ Docker : Containerization

④ SQL : Data analysis & exploration

⑤ prefect : Workflow orchestration

⑥ dbt : Data transformation

⑦ Spark : distributed processing

⑧ Kafka : Streaming

22/01

Data Engineering Zoomcamp.

Week 1 : GCP, Docker, SQL, terraform.

→ 1.4.1 : Setting env on GCP

- Compute Engine → VM instances
- Add ssh key to CE metadata (create new key)
- Nearest region, 16gb ram, 30 gb space, Ubuntu
- ssh -i private_key user@external_ip
- wget & bash Anaconda.sh (yes to all)
- Ssh Config (file in ~/.ssh

→ Host de-zoomcamp
  HostName <External ip of VM>
  User <user name>
  IdentityFile <private key path>

→ ssh de-zoomcamp

- Docker installation

→ ~~snap install docker~~ sudo apt-get install docker.io
→ sudo apt-get update &
→ remote ssh extension in VS code
→ google "docker without sudo" & follow. (restart VM)
→ docker run hello-world
→ if external ip not displayed, check columns selected.

Install Docker Compose

→ wget <docker compose link> -O docker-compose
→ Chmod +x docker-compose
→ Add bin folder to path in .bashrc

② apt vs apt-get

③ pip vs conda

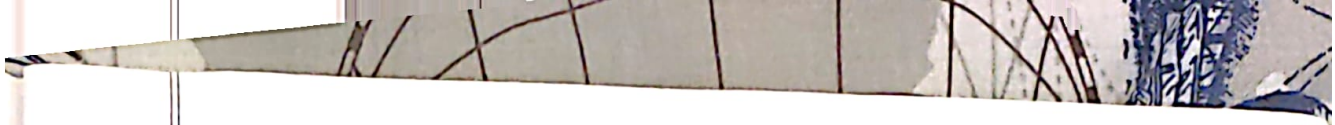
④ default ssh identityfile is id_rsa

⑤ Users added to docker group to use docker without sudo

⑥ Ubuntu VM ARM, AMD, X86_64

⑦ sftp vs scp

⑧ When VM instance is stopped then it is not charged but resources like disks & IP's are charged.



1. Clone dez git repo
2. cd week1 bonus n setup / 2 docker sql
3. docker-compose up -d (detached mode)
   Run multiple container apps at once.
   ↑ run containers in background
4. This is to install pgdatabase & pgadmin services as containers
5. ~~Use Conda to install pgcli & pip install dont~~ for mycli
6. Use pip to install pgcli
7. Forward 8080 (pgadmin port) in VS code and go to localhost:8080 in chrome
8. In jupyter create schema & insert 100 rows of yellow taxi data (download CSV) into pg. Check using pgcli

Terraform

① Dockerfile

```
FROM    python:3.9
RUN     pip install pandas
WORKDIR /app
COPY    pipeline.py  pipeline.py
ENTRYPOINT ["python", "pipeline.py"]
```

② Build Image from Dockerfile

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

If "Can't stat error", add folder to .dockerignore file (Due to permission of folders)

③ Docker run Container

```
docker run -it test:pandas 2023
```

O/P: ['pipeline.py', '2023']
job finished Successfully for day = 2023

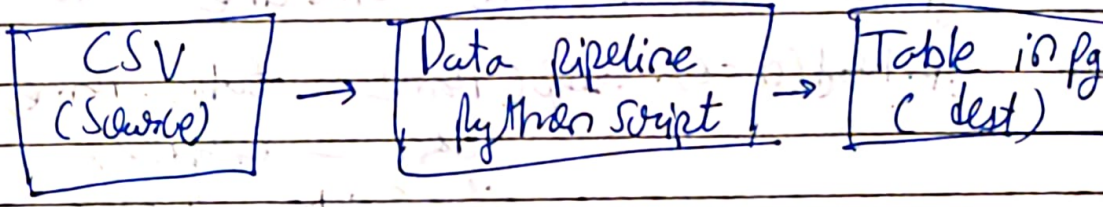
Image:
- Snapshot of filesystem, includes starting cmd
- just diskspace no CPU/memory
- Created using "docker build Dockerfile"
- Includes everything needed to run an app

Container:
- Created from images
- takes up CPU/memory
- always run the same, regardless of where

Jn 23

1.2.1 ⇒ Introduction to Dockers

Data pipeline



Why Docker?

1. Reproducibility
2. Local experiments
3. Integration tests (CI / CD)
4. Running pipelines on the Cloud
5. Spark & Serverless (AWS Lambda)

Docker

- provides ability to package & run an app in a loosely isolated env called a Container
- Dockerfile : text doc with instructions to build an image.

- FROM : Set Base image
- RUN : Execute Cmds
- WORKDIR : Set working dir
- COPY : copy files / dirs
- ENTRY : configure container to

30/26

1.2.4 ⇒ Dockerizing the Ingestion Script

① jupyter notebook to python script

```
jupyter nbconvert --to=script
```

② argparse used to take Cmd Line args.

③ `df.head(n=0).to_sql(name=table_name, con=engine, if_exists='replace')`

drops table before inserting new values.
append : insert new values to existing table

So this command is used to create a fresh new table before adding records to it.

④ Pass passwords using env var/password store instead of passing it in cmdline arg

⑤ Dockerize ingestion script
- add wget (apt-get) & sqlalchemy & psycopg2 (pip install) to Dockerfile
- Build image
- Run container

⑥ Local server (using python -m http.server) can be used for testing

⑦ localhost for container would be container's localhost. So to connect to host machine's network use --network network_name

⑧ If -it flag is absent, interaction is not possible

1.2.2 ⇒ Ingesting NY taxi data to Postgres

① Postgres run as Container

② pgcli used to interact with db

③ wget csv data

④ → sqlAlchemy engine created
→ csv read in chunks by specifying iterator = True, chunksize = 100000
→ In a infinite while loop
- read chunk
- Convert 2 columns to datetime
- load data

→ loop breaks due to StopIteration

⑤ If any issues with pgcli → Use jupyter & pd using pd.read_sql(query, con=engine)

1.2.3 ⇒ Connecting Postgres & pgAdmin

→ forward 8080 (pgadmin port) in VSC

→ Create Server on pgadmin

→ If Postgres & pgadmin are individual Containers started using Command line, then a network has to be created

→ If docker-compose is being used, network is automatically created

→ Run queries.

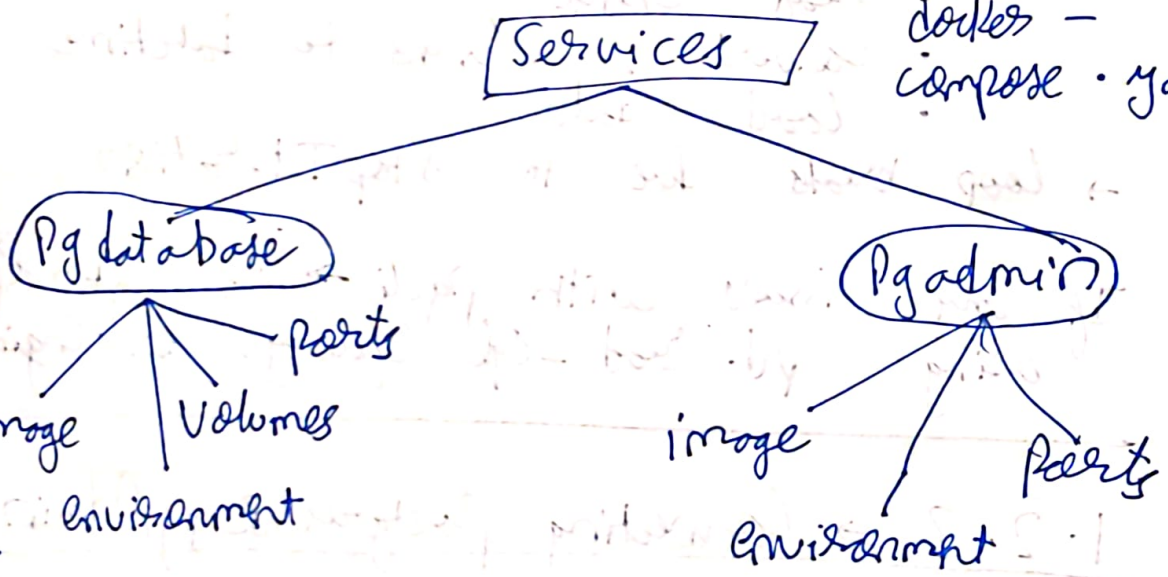
1.2.5 ⇒ Running postgres & PgAdmin with Docker Compose

① Docker-Compose is a tool for defining & running multicontainer Docker applications.

② a YAML file is used to configure the app

③ Then, with a single command, all services from config can be created & started

④ network is automatically created.



⑤ docker-compose up {-d / detached} to start

⑥ docker-compose down } to stop

⑦ docker network ls } to list networks

⑧ By default, "<dir-name>_default" network is created for a docker compose file

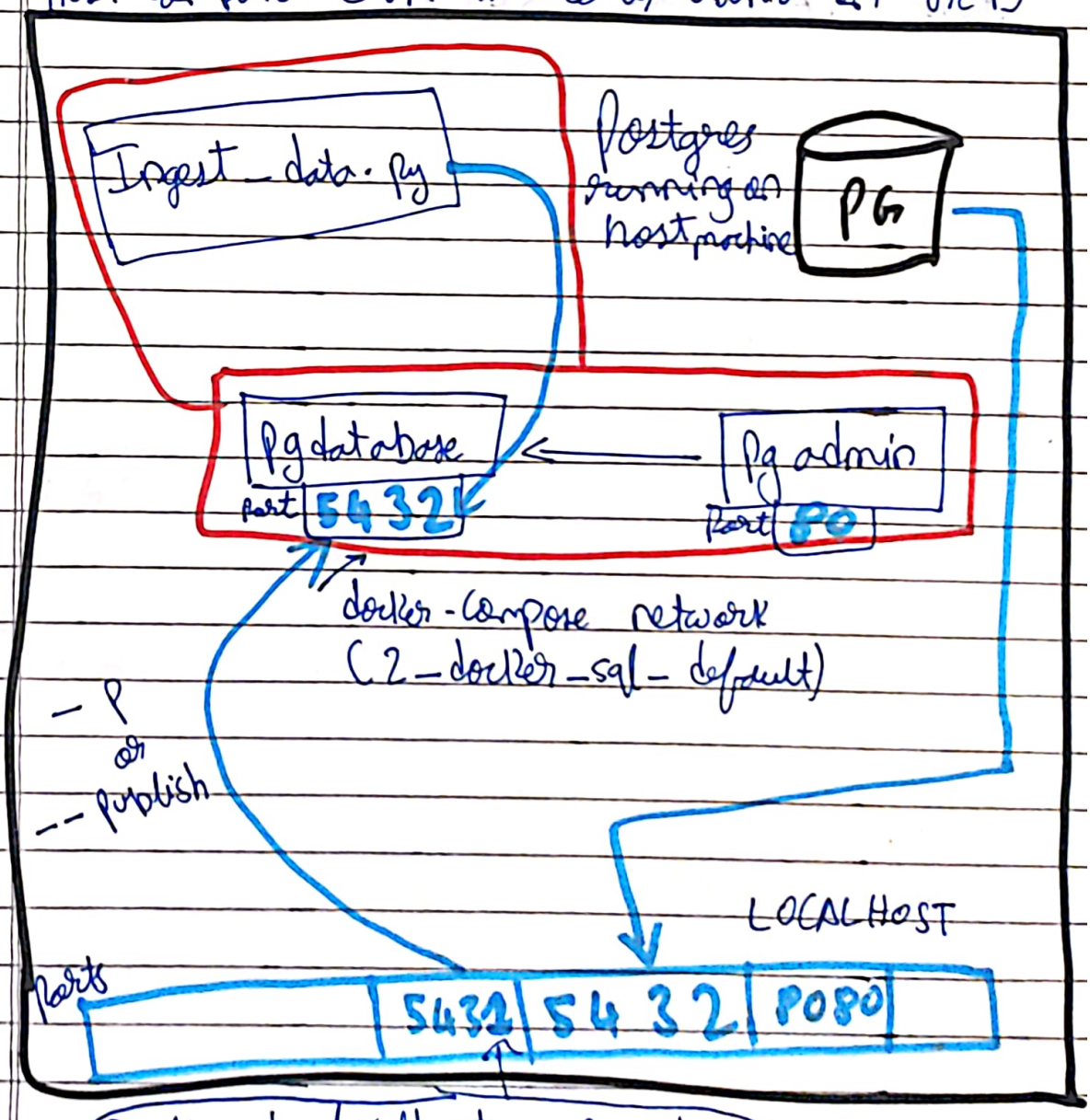
1.4.2 ⇒ Port mapping & Networks in Docker

① error checking context: Can't stat <dir_path>

This is caused as permissions are different dir is owned by root.

to solve it create a data folder, move pg folder inside data and add data folder to .dockerignore.

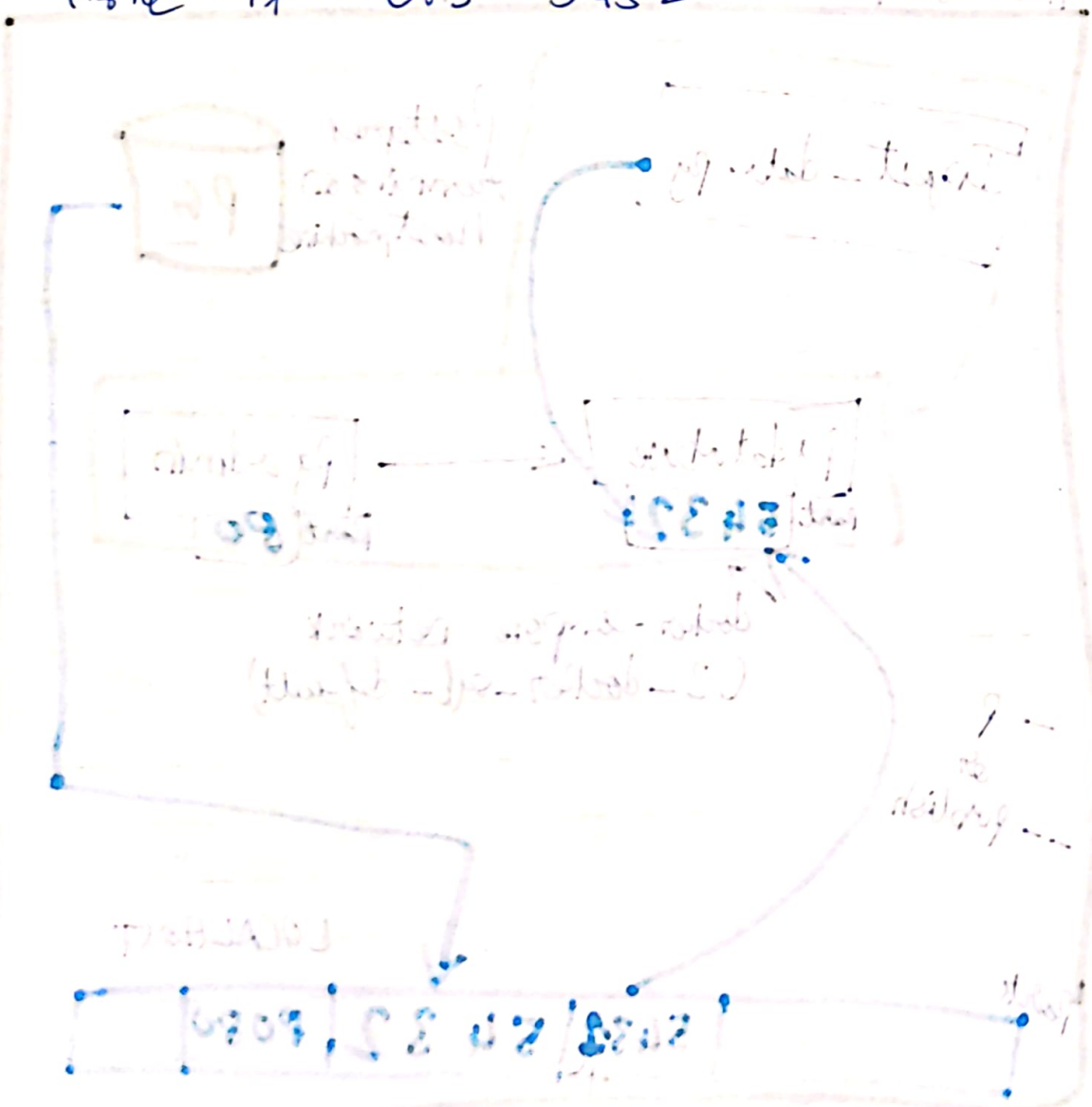
② Docker networks diagram



- On the host computer 5431 is used (Assuming there is a Pg instance running on 5432 on host) by mapping "5431 on host" → "5432 on Contain

So Pgcli would use 5431 to communicate with Pg inside Container

- ingestion Script goes via docker-compose network (as --network is specified) Hence it uses 5432



Jan 29

1.2.6 ⇒ SQL Refresher

① Zones Data in taxi_zone_lookup.csv is downloaded and added to db

② Implicit inner join b/w trips & zones to view zone name instead of zone id

③ Explicit inner join to do the same. This has the JOIN keyword and is preferred over implicit join

④ JOIN clause is used to combine rows from two or more tables based on a related column between them.

⑤ Types of joins

→ (INNER) JOIN: Matching values in both tables

→ LEFT JOIN: All from left, matched from right

→ RIGHT JOIN: All from right and matched from left

→ FULL JOIN: All records when there is a match in either left or right table

⑥ CAST → Convert data from one type to other

⑦ GROUP BY → Group rows with same values.

⑧ ORDER BY → Sorting data in ASC / DESC

1.1.1 ⇒ Intro to GCP

① Google Cloud Platform includes a range of hosted services for Compute, Storage and application development that run on Google hardware.

② Navigate to services using left navbar or top search bar.

1.3.1 ⇒ Intro to terraform

① Open Source tool by Hashicorp

② Used for provisioning infra resources

③ IaC : Infrastructure as Code ⇒

Build, change, manage → your infra in a → Safe, Consistent, Repeatable way

by defining resource Config that you can → Version, reuse, Share

④ Adv :

→ Infra lifecycle management → State based approach to track resource changes

→ Version Control Commits

→ Service account : Identity that an instance / app can use to run API requests on your behalf

→ IAM & Admin → Service Accounts → Create Service Account

→ Add name → Grant viewer → Done access

→ Actions → Manage Keys → Add → Create → JSON Key new key

→ terraform binary & gcloud SDK required

terraform -v

gcloud -v } to view version

→ Add JSON file path to env var & run auth command in gcloud & enter token in terminal

→ IAM → edit permission → Add

→ Usually predefined roles not used in prod Custom roles are created

→ enable IAM & IAM credentials API's

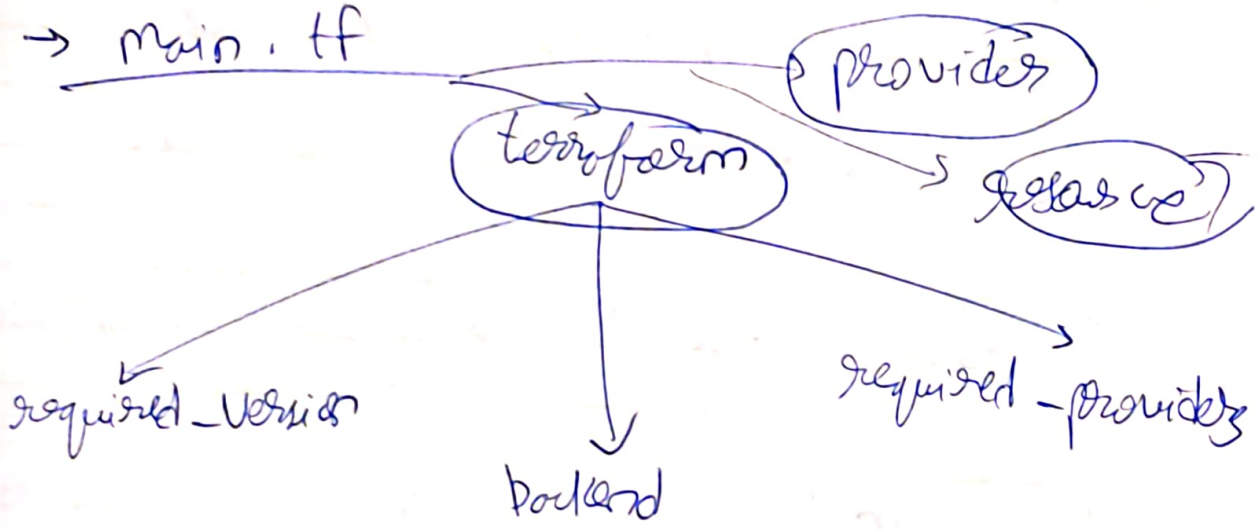
1.3.2 ⇒ Creating GCP infra w/ TF

→ change `.terraform -version` to version being used

→ provider plugins : binaries written in golang, present in terraform registry

→ docs for providers available for reference

→ main.tf



(local ⇒ store as terraform.tfstate file)

→ variables.tf → locals
→ variable

→ change region in variables.tf

→ terraform init : Initialize backend
& install plugins/providers

→ terraform plan : preview local changes against remote state

→ terraform apply : Asks for approval to proposed plan and applies changes to cloud

→ terraform destroy : Remove stack from cloud

Jan 30

Week 2 : Workflow Orchestration

2.1.1 ⇒ Data Lake

- Data Lake holds data from many places.
- Can be Structured, Semi or Unstructured
- Metadata is usually associated for faster access.

Ex: GCS, S3, Azure Blob

	Data Lake	Data Warehouse
*	Contains data with minimal processing	Contains cleaned and pre-processed data
*	Large amount of data	Small amount of data
*	ML, Streaming analytics	Batch processing, Reporting
*	ELT	ETL
*	Schema on read	Schema on write

2.2.1 ⇒ Intro to WO

from docs

If you move data you may need these:

- → Scheduling
- → Retries
- → Logging
- → Caching
- → Notifications
- → Observability

prefect offers this functionality.

2.2.2 ⇒ Introduction to Prefect Concepts

→ Create Conda env ⇒ conda create -n zoomcamp python=3.9

→ Activate ⇒ conda activate zoomcamp

→ Clone Code from datadriven repo

→ pip install -r requirements.txt

Preps

→ Before: ingest_data that takes params for a postgres connection, pull the data & load to db.

→ Has to be manually triggered; no visibility, retries, caching etc.

After:

① A prefect flow is a function with @flow. has workflow logic

② Flow run is a single execution of a flow

③ Task represents unit of work, function with @ task. Not required i.e prefect workflow can contain only flows. tasks are called from a flow

④ extract_data downloads file from github, hence retries & caching is added

⑤ transform_data removes null passenger_count and logs before & after values

⑥ Blocks enable storage of configurations.

⑦ SqlAlchemy Connector block is used to store db config.

Feb 5

2.2.3 ⇒ ETL w/ GCP & prefect

① prefect orion start : Start prefect UI

② etl_web_to_gcs() : ETL flow

tasks

③ Fetch: gets data from web & returns df. retries & waits (Exponential backoff etc) can be added

④ clean : Takes df, fix datatype issues (convert obj to datetime), write locally to Parquet log_prints = True , logs the print statements

parquet is gzipped, using pyarrow for best compression.

⑤ write_gcs : Upload parquet file to GCS.

Create a bucket in GCS
register block in prefect
prefect-gcp

Create service account with BQ Admin & Storage Admin roles.

Create GCS Bucket block, add → bucket name.
run 'Python etl_web_to_gcs.py'

⑥ local parquet file created

⑦ This file is uploaded to bucket in GCS

2.2.4 ⇒ From GCS → BigQuery.

① Flow : etl_gcs_to_bq

② task 1 : extract_from_gcs
Download parquet file from gcs to local

③ task 2 : transform
read parquet file using Pandas, fill NA passenger count with 0, log pre & post count and return df

④ task 3 : write_bq
get creds from prefect block
use df.to_gbq() to load df to BQ in chunks.

⑤ Dataset has to be created in BQ UI
&
table

⑥ change Bucket block name, table & project id

Feb 6

2.2.5 ⇒ Parameterizing flow & deployments with ETL, into GCS flow

1. Pass parameters instead of hardcoding (ex colour, year & month to construct URL for NY taxi dataset).
2. etl_web_to_gcs called inside etl_parent_flow, with different params
3. Cache_key_fn added to avoid fetching pre-fetched data
4. Flow is run manually & observed via UI/API
5. Prefect deployment: trigger & schedule flows from UI instead of terminal. Can be done through CLI / Python
   - prefect deployment build <filename.py>:<flow entrypoint> -n "<name of deployment>"
   - prefect deployment apply <name of generated yaml file>
6. Work Queues organize work that agents pick up. Agent processes are lightweight polling services that get scheduled work from a WQ.
7. Notifications can be sent to different destination.

Office hours 2

① Code formatting

→ Black + VS Code Extension

→ MLOps Zoomcamp → Best practices.

→ typehints

→ docstrings

→ Alex Miller blog.

② prefect recipes Catalog

③ Why Parquet?

Compression w/ Snappy

Columnar format

metadata about Columns

Partioning

Apache Deltalake

Schema definitions has to be strict

④ "prefect register" is used to tell the Server about available prefect blocks.

⑤ prefect vs Airflow

Founder was main Contributor

read why not Airflow blog

prefect Cloud vs open source

→ updates

→ collaborators

→ Actions: trigger something on event

Feb 8

2.2.6 ⇒ Schedules & Docker Storage w/ Infrastructure.

(1) Schedules are used to run flows automatically

(2) Can be added via CLI during deployment build or via UI

(3) Cron, IntervAl, or rrule can be used

(4) Use prefect deployment build --help

(5) Dockerfile with prefect base image, requirements.txt, flows code and empty data folders.

(6) docker image build -t heisenkub/image-name . (should be logged in to dockerhub)

(7) docker image push heisenkub/image-name

(8) Docker block with image created.

(9) deployment of dockerized flow done through python script (can also be done via CLI)

(10) PREFECT_API_URL set to local orion instance

(11) To avoid caching issue in dockerized deployment, remove caching code

(12) Add GCP Credentials block containing service account data, instead of path to JSON file

Feb 12 | Week 2 Homework Solutions

① • Hardcode params or parameterize Script
ideal

• Column names in green & yellow are different
(4prep) (4prep)

• if else based on color to clean these columns

• Code to create data/color dis

② '0 5 1 * *' ⇒ 5 AM on day 1 of the month

③ count sum can be added in code itself instead of adding 2 manually.

④ Before: code locally or code files in docker
after: code in Github stored

GH block created through UI / Python

⑤ Add webhook URL, notified when flow reaches given state

Week 3: Data Warehouse & BigQuery.

3.1.1 ⇒ OLAP vs OLTP

OLTP ↳ Online Transaction Processing

↳ Online Analytical Processing

	OLTP	OLAP
Purpose	Control & run essential business operations in real time	Plan, solve problems, support decisions, discover hidden insights.
Data updates	Short, fast, initiated by user	Data periodically refreshed w/ scheduled batch jobs.
DB design	Normalized for efficiency	Denormalized for Analysis
Space req.	Generally small if historical data is archived	Generally large due to agg large datasets
User cls	Clerks, Online shoppers	Data analysts, BA, Executives

- DWH is a OLAP solution for reporting & data analysis.
- BQ: Serverless DWH - No servers to manage
  No DB software to install
  Software & infra - Scalability
  High availability.

- Built in features like ML, geospatial, BI Analysis

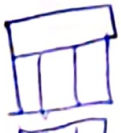
- Maximizes flexibility by seperating Compute engine analyzing data from Storage.

- Hierarchy : Project
  - Schema (dataset)
    - Table

- Cost < On demand / Flatrate

Partitions (Segment)

- Seperate data into smaller Chunks
- If most of our queries filter based on a column value (say date), the dataset can be partitioned on this column.
- That way BQ would go through only the required data. So ↑ perf & ↓ cost
- Partioned table query scans through less data
- Partitions can be viewed by " dataset.INFORMATION_SCHEMA.PARTITIONS
  where table_name = table_name.
- Partitioned table displays table size, No of rows, type & field
- No Partition icon | Partition icon

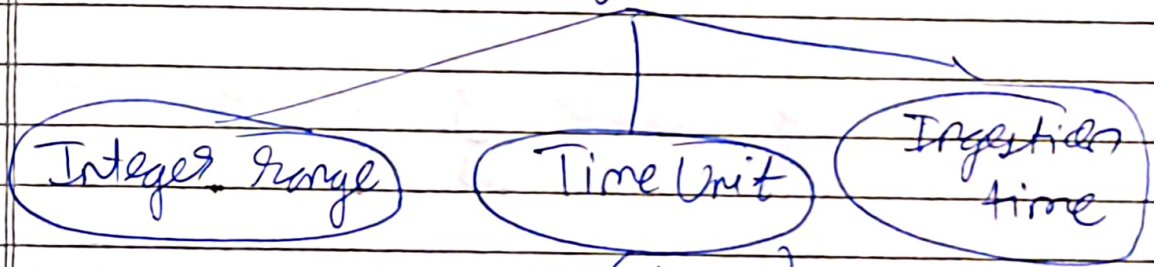


Clustering (Optimize)

- Sort data using 1 or more columns & Store together
- User defined Column Sort Order
- improves query performance & reduces cost

In P : Query Cost known before
In C it is after

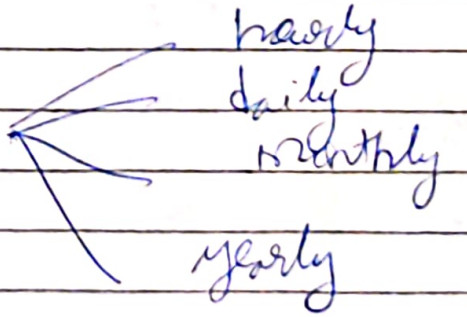
3.1.2 Types of Partitioning



Start, end, interval
0 50 10

0-9 | 10-19 | 20-29 | 30-39 | 40-49

Date
Datetime
Timestamp



- No of partitions can be 4000 max

For Clustering :

→ Columns specified are used to Colocate related data

→ order of columns is important - order of specified columns determine sort order of data

→ Clustering improves filter & Agg queries

→ Table with < 1 GB data size do not show significant improvement with P&C

→ Up to 4 Clustering Can be specified columns

Clustering over partitioning

- If P results in Small amount of data per P less than 1GB
- If P results in large number of P (>4000)
- If P results in modifying majority of partitions frequently

Automatic Reclustering

- When data is added to a C table

→ newly added data can be written to blocks that contain key ranges that overlap with keys ranges in previously written blocks

→ This weakens Sort property of table

- To maintain performance

→ BQ performs Automatic RC in background to restore Sort property

→ For P tables, C is maintained for data within scope of each partition

Feb 19

3.2.1 ⇒ BQ Best Practices

Cost reduction

① Avoid Select * . Use Select <Column names>

② Price queries before running them

③ Clustering & partitioning

④ Use Streaming inserts w/ caution

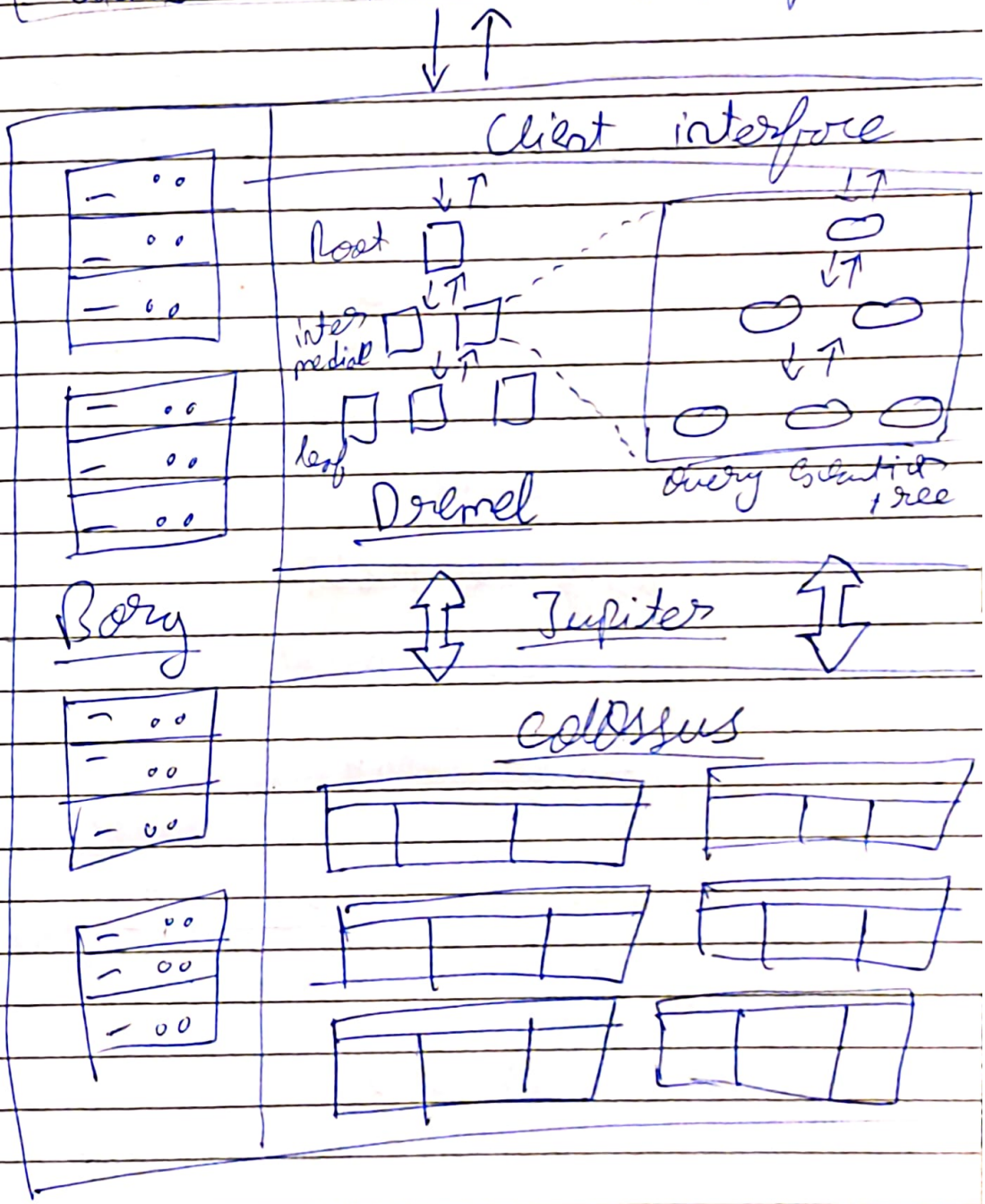
⑤ Materialize query results in stages

Query Performance

1. Filter on partitioned columns
2. Denormalizing data
3. Use nested or repeated columns
4. Use external data sources appropriately
5. Reduce data before using a join
6. Do not treat WITH clauses as prepared statements
7. Avoid oversharding tables
8. Avoid JS User defined functions
9. Use Approx Agg fns (Hyperloglog++)
10. Order last in query
11. optimize join patterns
12. Place table with most rows first then table with the fewest rows then remaining tables by decreasing size

3.2.2 ⇒ Internals of BQ

Clients - Rest API, web UI, bq cmdline.



Dremel	Execution engine : SQL query to Execution tree
Colossus	Distributed storage : Distributed file System
Borg	Compute : Cluster management System
Jupiter	Network : Networking Infra
BQ :	These infra + Dozens of high level tech form BQ

Feb 25

3.3.1 ⇒ BigQuery Machine Learning

- Can build ML model in DWH itself, avoiding extra step of exporting the data.

Steps

① Create a table with appropriate Column data types. Categorical features are considered as string
② Create model with default setting
③ Check features
④ Evaluate the model
⑤ Predict using the model
⑥ Predict & Explain model
⑦ Hyper parameter tuning

3.3.2 ⇒ BQ ML Deployment

① gcloud auth login
② export project to GCS
③ Create dir & copy model to local dir
④ Create Serving dir & Copy model here
⑤ Pull tensorflow serving docker image & run it
⑥ Call API using POST request w/ parameters & get predicted tip amount

Week 3 HW Solution

(1) Modified web to GCS prefect Script to work from FHV data

(2) Created deployment & run with 1-12 months & 2019 parameter. data now uploaded to GCS bucket

(3) In BQ, Create external table from 12 Gz files stored in GCS

(4) Create BQ table from external table.

Q1) Count of records in table.
Sol: 43,244,696

Ext table took 26 Sec
BQ table took 366 ms (faster)

Q2) distinct Count of Column in table
Sol: ext table processes 0 bytes
BQ table processes 317.94 MB

As data is not in BQ, estimation is not possible

Q3) Count where both Column values are null
Sol 717,748

(04) Strategy for optimize query.

filter hints partition
order hints cluster

Sol: P by pdt, C by abn

(05) Performance of optimized table

non optimized ⇒ 647.87 MB

optimized ⇒ 23.05 MB

(06) Sol: Data in External table is stored in GCS bucket

(07) Sol: Not a best practice to always cluster data, ex: when data is less than 1GB, creates overhead instead of optimized perf

2.1 ⇒ office hours ⇒ Prefect, Jeff Hale.

① Better approach than download to local & upload to BQ.

Sol Ingestion tool like Airbyte, fivetran prefect - gcp

② K8's like Argo

Sol: Collections Catalog → Ray, Dask, Kubernetes.

③ Prefect & Spark

Sol: Collections Catalog → Databricks, Fugue

④ prefect agent, deployment, profile

⑤ prefect helm-chart

---

office hours #3 ⇒ Alexey.

① Data cleaning < Soda Core / great expectations.

②

dbt	Spark
• Compute in DWH • data never leaves DWH	needs infra DWH → clean → DWH

Use prefect for orchestration, delegate compute to K8's, Aws batch, spark etc

④ for project

Decide dataset → decide streaming or Batch → Think Criteria Points

---

DATA

- Loading: Fivetran, Stitch, Airbyte
- Storing: Snowflake, BQ, Redshift
- Modelling: dbt, dataform
- presentation: data Studio, Tableau.

Data modelling Concepts.

- ETL: Extract → Transform → Load → Reporting
  - → Slightly more stable & Compliant data analysis
  - → Higher storage & compute costs
- ELT: Extract → Load → Transform → Reporting
  - → Faster and more flexible data analysis
  - → Lower maintenance

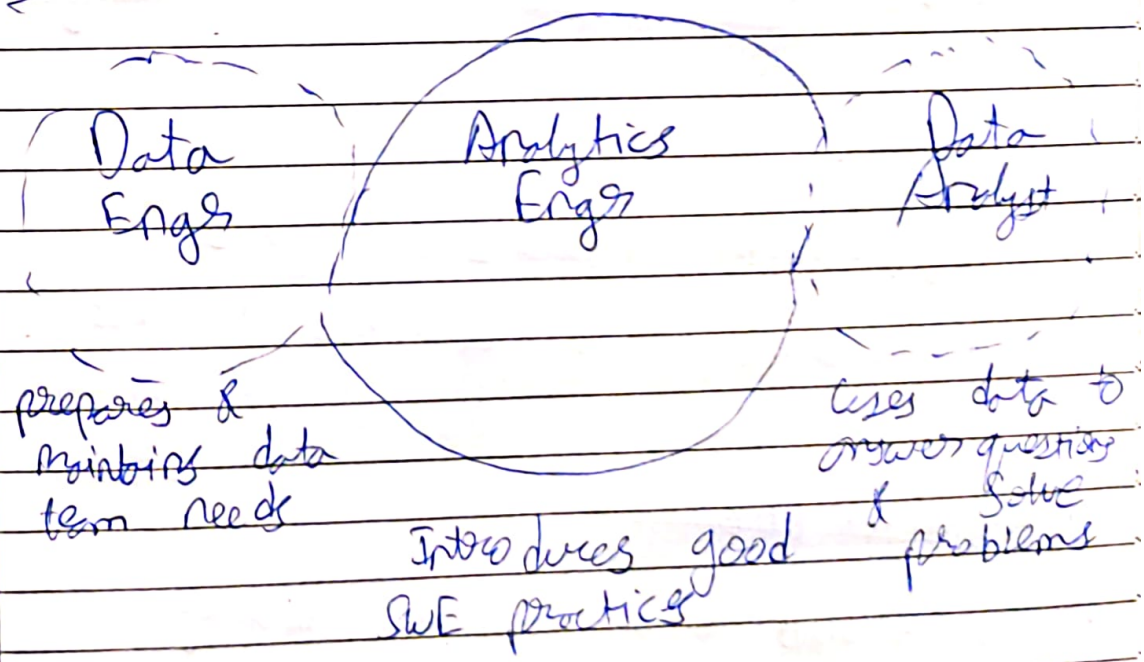
Mar 7

Week 4 : Analytics Engineering.

Setup dbt for BQ

1. Create dbt cloud account
2. Create BQ Service account with BQ Admin role, add & download JSON key
3. Create project in dbt → Select BQ DWH → Add SA key JSON file.
4. Copy SSH git URL of your repo → paste ~~Deploy~~ generated by in github settings under Deploy keys with write access.

4.1.1 ⇒ Analytics Engineering Basics



Kimball's Dimensional Modelling

Objective :
- Deliver Data understandable to the business users.
- Deliver fast query performance

Approach : User understandability & query performance → priority → Non redundant data (3NF)

- **Other approaches** -
  - → Bill Inmon
  - → Data Vault

Elements of Dimensional modelling

Fact table

- Measurements, metrics or facts
- Corresponds to a business process
- Verbs

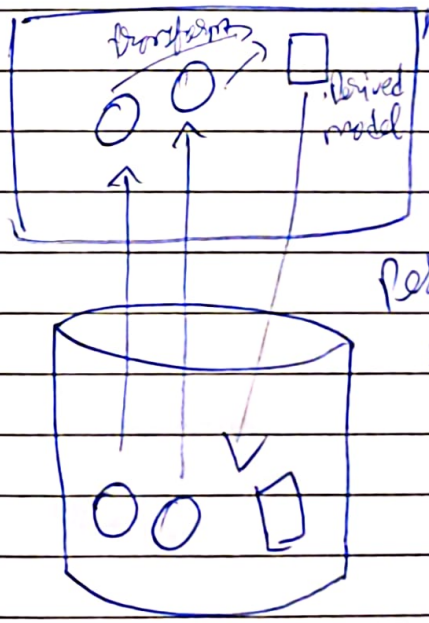
Dimension table

- Corresponds to a business entity
- Provides Context to a business process
- Nouns.

Architecture: Staging → Processing → Presentation

4.1.2 ⇒ What is dbt?

- dbt: data build tool.
- transformation tool, uses SQL to deploy analytics code following SwE best practices like docs, CI/CD, modularity portability



Each model is
→ A SQL file
→ Select statement (no DDL/DML)
→ File that dbt compiles and runs in DWH

dbt Core: Open Source, allows data transformation

dbt Cloud: SaaS app to develop & manage dbt projects

Mar 8

Week 5 : Batch Processing.

5.1.1 ⇒ Intro to BP

Batch processing : Processing & Analysis happens on data already stored over a period of time

Ex: Payroll & Billing systems processed weekly or monthly

Stream processing : Happens as data flows through a system. Analysis & reporting of events as it happens.

Ex: Fraud detection

Tech for Batch Processing.

→ Python Scripts running on → Kubernetes, AWS batch, Airflow/prefect orchestration

→ SQL

→ Spark

→ Flink

Adv

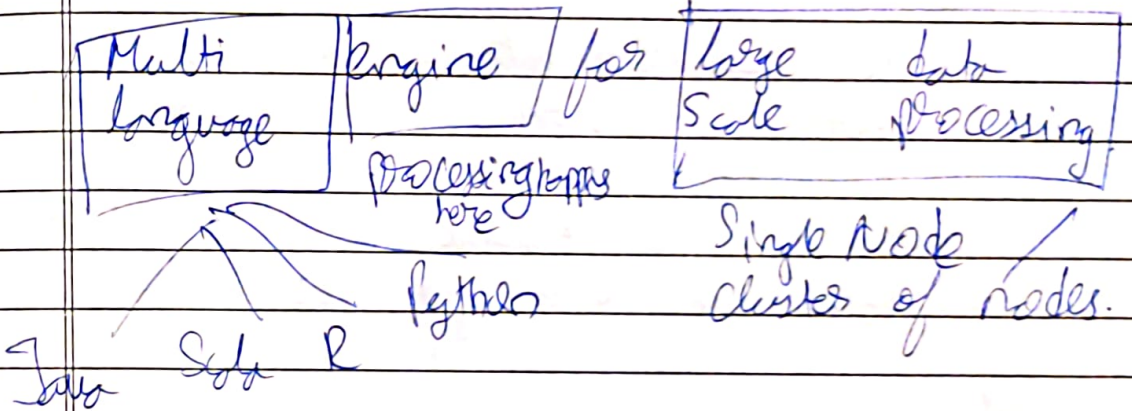
① Easy to manage ③ Scaling is easy.

④ Retry

Dis

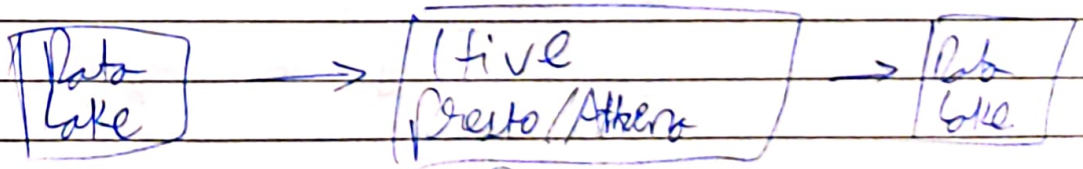
Waiting for data to be processed, ex: to get the data from previous hour we need for it to get processed

5.1.2 ⇒ Intro to Spark.

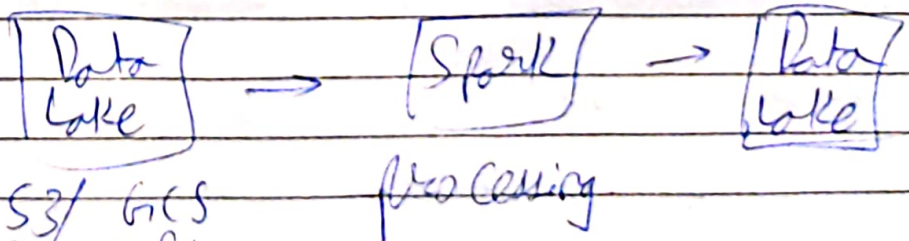


When to use Spark ?

If Batch job can be expressed as SQL then use SQL



Otherwise use Spark.



• Spark can be used for ML work load

5.2.1 ⇒ Installing Spark on Linux (GCP VM)

① Install Java (Open JDK)

wget file → tar xvzf → Set path var → java --version

② Install Spark

wget file → tar xvzf → Set path var → Run Scala Cmd in Spark-shell

③ Add path vars to .bashrc so that when bash is run these vars get set

④ Pyspark

Set path var & add to .bashrc

Download csv file and run python code in Jupyter →

① Create SparkSession
② Read csv file to df
③ Show df
④ Save df to parquet

⑤ Spark UI : When SparkSession is created a UI is hosted on port 4040. Expose port 4040 in VS Code & open localhost:4040 in chrome to view Spark web UI

5.3.1 ⇒ First look at Spark & PySpark

- Create Spark Session
- Download & Extract FHV HV 2021-01 data
- Rd this CSV file using Spark. 11 million rows
- Spark reads all columns as Strings dtype.
- To fix this: ① Read Sample data (1000 rows) in pandas and print df.dtypes
  ② Create df from pd df
  ③ Create Schema with required types
  ④ Pass Schema while reading CSV in Spark
- Partitioning

→ To achieve high parallelism, Spark splits data into smaller chunks called partitions.
→ These are distributed across different nodes in the Spark cluster.
→ Use df.rdd.getNumPartitions() to see no of partitions
→ Use repartition() to change & coalesce() to decrease no of partitions
→ repartition involves shuffling, which is expensive, Coalesce tries not to use shuffling
- df is repartitioned to 24 & Stored as parquet

5.3.2 ⇒ Spark Dataframes

- Parquet knows schema so each column has a dtype.

Actions	Transformations
→ Lazy : Not executed immediately	Eager : Executed immediately
→ Select	→ Show
→ Filter	→ take
→ Join	→ Head
→ GroupBy	→ Write

- Pyspark comes with many in built functions.
  From pyspark.sql import functions as F
- UDF : User defined functions can be used too

5.3.3 ⇒ Preparing Yellow & Green taxi data.

- download_data.sh : to download green|2020 / yellow|2021 data
  - → set -e Used to stop execution if non 0 status code
  - → Takes arg as colour & year
  - → loops through all months
  - → format number as 2 digits 1→01, 10→10 using "%02d"
  - → Create dir using mkdir -p
  - → wget csv.gz file to created path

- Read CSV and store as Parquet.

→ get Schema from CSV file Using pandas.
Create Spark df using pd df

→ Schema created, edited to add types.

→ Convert long to Integer, string to datetime

→ Loop over each month 1-12
Read CSV with Schema
Repartition it into 4 files
Write to parquet

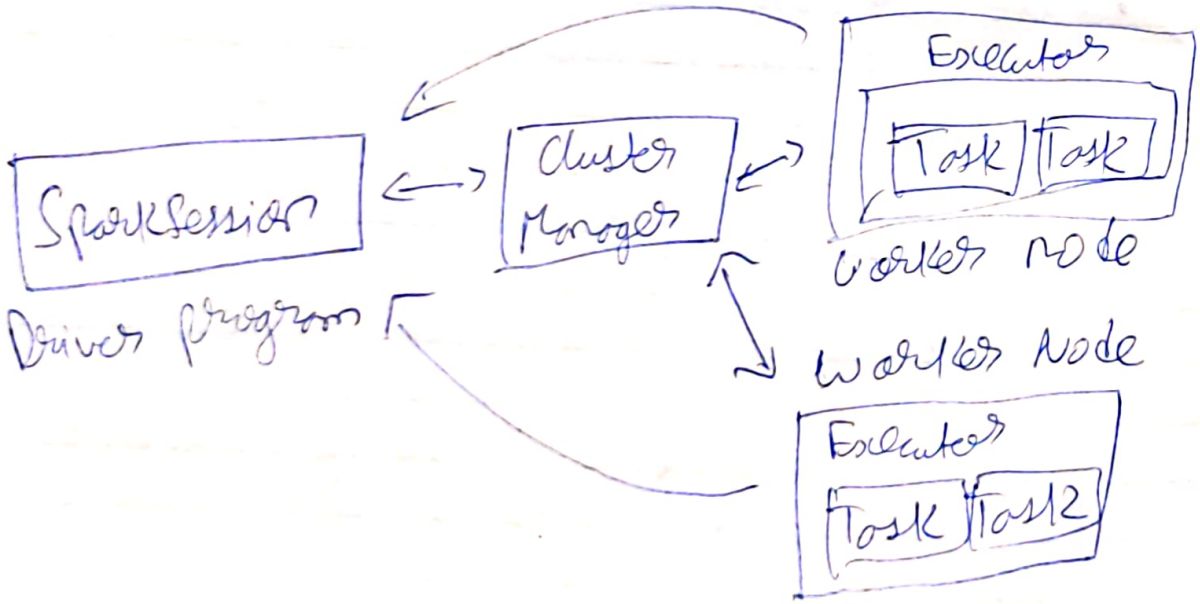
5.3.4 ⇒ SQL with Spark

- We will Use the files created in prev video Using download_data.sh & taxi_schema.ipynb
- The files are 2020 & 2021, Yellow & green taxi data in parquet format
- Create Spark Session, read above data
- Rename datetime columns so green & yellow column names match ('lpep vs tpep)
- Create Common_Columns list. Set intersection not used to preserve order
- Select only common columns & Combine both to Single df. service_type used to Know where Column Came from

- Register df as table using registerTempTable to run SQL queries on table
- Reverse grouping query (from week 4) is run on Spark
- result is saved to Parquet file.
- This creates hundreds of small files.
- Coalesce is used to decrease it to 1
- So the result is stored in a single parquet file

5.4.1 ⇒ Anatomy of a Spark Cluster

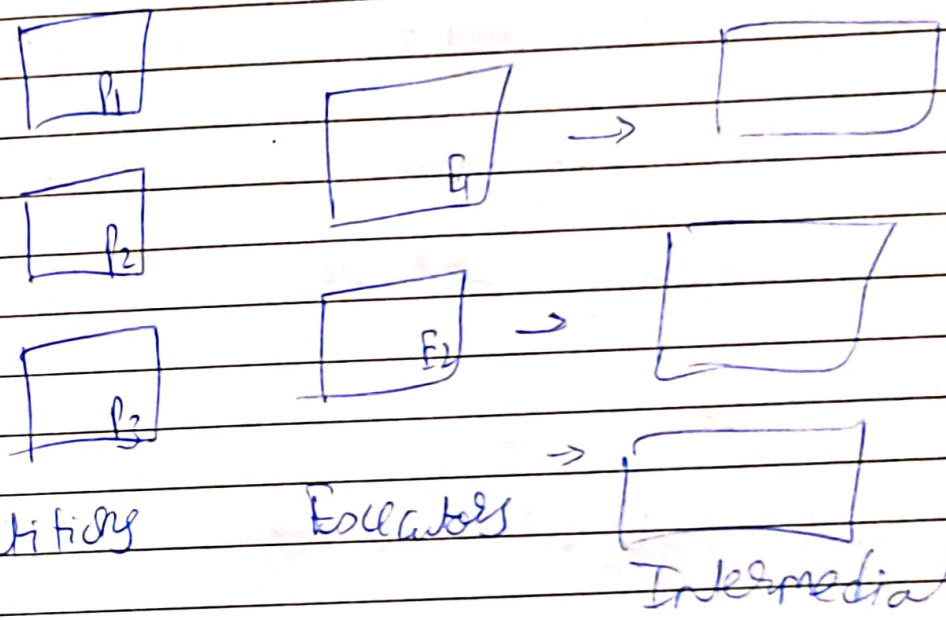
local[*] : Run Spark locally with as many worker threads as logical cores on your machine



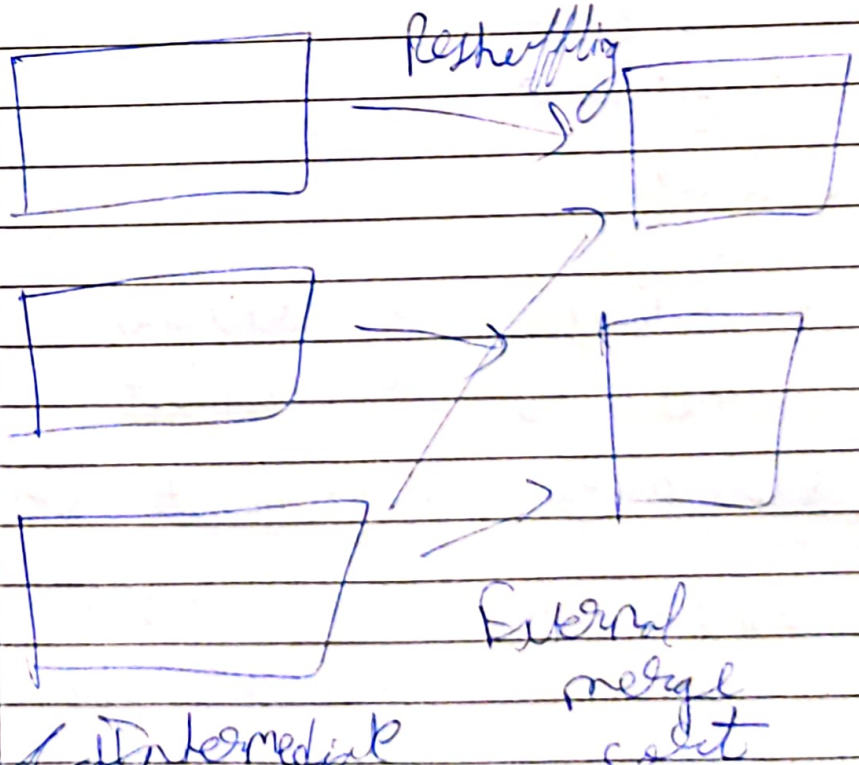
Master Slave

5.4.2 ⇒ Groupby in Spark

Stage 1



Stage 2



Apr 12 2023

5.4.3 ⇒ Joins in Spark

- Joins are used to Combine 2 or more datasets based on a Common key.
- The idea is to match records from multiple datasets and return a new dataset that Contains information from both Sources.
- Join uses reshuffling which uses External merge Sort Algo
- Materialize

→ Convert RDD/Dataframe → Concrete data structure

→ Can be Stored in memory/disk and reused for faster processing

Types of Join
- Inner
- left outer
- right outer
- outer

(Join Strategy)

Based on how data is distributed across nodes
- Partitioned
- Broadcast

Partitioned & Sort merge Join	Broadcast Join
For Large tables	. For 1 large & 1 small table . Smaller table Sent to all nodes
Default Strategy	

5.5.1 Operations on Spark RDD's

→ RDD: Resilient Distributed Dataset

→ Way of representing data in spark that allows you to distribute it across multiple nodes in a cluster so that you can work w/ it in parallel

→ Collection of objects that are partitioned & distributed across different nodes in a cluster

→ DataFrames have become the preferred API for working with structured data.

However RDD's are still important in certain scenarios.

take(n)

returns list of n Rows. Row is a spark object representing a row of data

map()

Applies function to each row & returns new RDD w/ the results

SQL query is expressed as set of steps in Spark RDD query.

① Spark session built

② Green taxi data read from parquet files

③ Only required ~~data~~ columns are selected as RDD

④ A chain of operations are done:

ⓐ filter_outliers: Considers only rows with pickup datetime after 01/01/2020

ⓑ map: prepare_for_grouping:

return (key, value)

(hour, zone) (amount, count)

ⓒ reduceByKey: Calculate revenue

Group elements of a pair RDD by key and apply a reduction function to the values associated with each key.

Ex: [('a', 1), ('b', 2), ('c', 3), ('a', 4), ('b', 5)] —reduceByKey (Sum operation)→ [('a', 5), ('b', 7), ('c', 3)]

it is used to Sum amount and count across grouped by keys

(d) Pop: Unwrap : Single row created from nested structure using tuple unpacking

(e) toDF (schema) : Converts result to Df w/ given schema

Result is also written to Parquet

5.5.2 ⇒ Spark RDD mapPartitions

⇒ map() applies a function to each individual element of an RDD. The function is called once for each element in the RDD.

→ mapPartitions() applies a function to each partition of an RDD, allowing the function to process multiple elements of the partition at once

Eg: To run an ML model on partitions of a dataset.

5.6.1 ⇒ Connecting to GCS

→ gcloud auth login

→ gsutil -m cp -r "pq/" "gs://dez-bucket/pq"

→ All parquet files gets uploaded to GCS bucket

→ GCS Connector: Open Source Java library that can run Spark jobs directly on data in Cloud Storage.

This is downloaded

→ SparkConf modified to include Connectors and credentials file

→ Hadoop Config done

→ now data can be read directly from GCS bucket using "gs://<path>"

→ Sample rows shown & total count is shown

5.6.2 ⇒ Creating a Local Spark Cluster

Before: A local Spark cluster is created. We connect to this.

master is specified as local

Now: Creating Spark cluster outside of jupyter notebook.

AKA: Running Spark in Standalone mode

① Start master & workers

→ cd to SPARK_HOME/sbin dir
→ ./start-master.sh
→ UI is on localhost:8080
→ Instead of master as local[*], the URL of spark master is specified (spark://)

Jobs won't be run yet, as there are no running workers.

So ⇒ ./start-worker.sh <master-spark-URL>

② Convert notebook to ~~Python Script~~

jupyter nbconvert --to=script notebookName

③ Running Script

→ Make sure no other script is running as it might take up all resources
→ Use args instead of hardcoding green, yellow & output path
→ Master URL is hardcoded & no of executors is not specified in script

④ Using Spark - Submit

→ Used to launch applications on a cluster
→ master URL can be specified here instead of in code

→ Run command where .py file is present, Spark-Submit is present in PATH

→ Results are stored after job is run

Stop masters & workers as done

./ stop-worker.sh
./ stop-master.sh

5.6.3 ⇒ Setting up a dataproc cluster.

Dataproc is a Fully managed & highly scalable service for running Spark jobs (among many others)

→ enable the API
→ Set cluster name & region
→ ideally we use Standard (1 master, N workers) but for practice/testing we go with Single node (1 master, 0 workers)
→ enable Jupyter notebook & docker
→ Create cluster

Submitting Jobs

→ Click on Created Cluster → Submit job

→ job type is pyspark, Main python file is the gs:// path of python script

→ Add arguments (change paths from local file system to GCS buckets)

→ Once the job is run & Complete, the output report parquet is stored in GCS bucket

→ In case of permission error: Add IAM roles for dataproc.

Jobs can be Submitted in multiple ways

- Console
- gcloud
- REST
- Python
- Golang/Node/Java

5.6.4 ⇒ Connecting Spark to BQ

→ we saw how to get data from GCS, process using Spark & put result back to GCS

→ We might need to put results data to DWH like BQ, instead of GCS

→ One way: Create External table from files in GCS and then create Normal table

A better way would be to use Spark to write to BQ

→ Set temporary GCS bucket (bucket is automatically created during cluster creation) in spark config

→ Coalesce not required, as data is written to BQ

→ write format is bigquery, option is "table" along with table name

Once the command is run, a Spark job is sent to Dataproc, the job results is stored in BQ