GCP Data Engineer[1]

v20190802

1. Exam overview

The exam consists of 50 questions that must be answered in 2 hours.

The content including:
- Storage (20% of questions),
- Big Data Processing (35%),
- Machine Learning (18%),
- case studies (15%) and
- others (Hadoop and security about 12%).

All the questions are scenario simulations where you have to choose which option would be the best way to deal with the situation.

2. Big Data Ecosystem[2]

particularly focusing on Apache Pig, Hive, Spark, and Beam

- Hadoop
    - open source **MapReduce** framework
    - the underlying technology for **Dataproc**
- HDFS
    - Hadoop File System
- Pig
    - scripting language that compiles into MapReduce jobs
    - Procedural Data Flow Language: **PigLatin**
    - Less development effort & code efficiency
    - does not have any notion for **partitions**
    - supports **Avro**
- Hive
    - data warehousing system and query language
    - SQL-like
- Spark
    - fast, interactive, general-purpose framework for SQL, streaming, machine learning, etc.
    - solves similar problems as Hadoop MapReduce but with a **fast in-memory** approach.
- Sqoop
    - transfer data between Hadoop and structured datastores (relational)
    - Sqoop imports data from a relational database system or a mainframe into HDFS.
    - Running Sqoop on a Dataproc Hadoop cluster gives you access to the built-in Google Cloud Storage connector
    - The two previous points mean you can use **Sqoop to import data directly into Cloud Storage**
- Oozie
    - workflow scheduler system to manage Apache Hadoop jobs
    - Oozie Workflow jobs are Directed Acyclical Graphs (DAGs) of actions.
- Cassandra
    - Wide-column store based on ideas of BigTable and DynamoDB(**Datastore**)
    - Wide column store
    - solution for problems where one of your requirements is to have a very **heavy write** system and you want to have a quite responsive reporting system on top of that stored data.
    - **does not provide ACID** and relational data properties
    - an available, partition-tolerant system that supports eventual consistency
- MongoDB
    - is fit for use cases where your system demands a schema-less document store.
- HBase
    - might be fit for search engines, analyzing log data, or any place where scanning huge, two-dimensional join-less tables is a requirement.
- Redis
    - is built to provide In-Memory search for varieties of data structures like trees, queues, linked lists, etc and can be a good fit for making real-time leaderboards, pub-sub kind of system.
- MYSQL
    - You can easily add nodes to MySql Cluster Data Nodes and build cube to do OLAP. (answer for a mock question)

2. Storage

Cloud Storage/Cloud SQL/DataStore/BigTable/ BigQuery.

---

1 by ***github.com/xg1990*** modified based on 'James' GCP Dotpoints' with various online sources which are acknowledged at the end, please share under the Creative Commons Attribution 3.0 Australia License. Please feel free to submit any issues on https://github.com/xg1990/GCP-Data-Engineer-Study-Guide/issues if you find any mistakes

2 refer to https://hadoopecosystemtable.github.io

2.1.Cloud Storage (GCS)

- Blob storage. Upload any bytes to a location. The content **isn't indexed** at all just stored. (Amazon S3)
- Virtually **unlimited** storage
- Nearline and Coldline: for ~1 sec lookup for access, charged for volume of data accessed
    - **Nearline** for once per month
    - **Coldline** for once per year
- buckets to segregate storage items
- geographical separation:
    - persistant, durable, replicated
    - spread data across zones to minimise impact of service disruptions
    - spread data across regions to provide global access to data
- Ideal for storing but **not for high volume of read/write** (e.g. sensor data)
- A way to store the data that can be **commonly used by Dataproc and BigQuery**

Encryption:
- Google Cloud Platform encrypts customer data stored **at rest** by default
- Encryption Options:
    - Server-side encryption:
        - **Customer-supplied encryption keys**: You can create and manage your own encryption keys for server-side encryption
        - **Customer-managed encryption keys**: You can generate and manage your encryption keys using **Cloud Key Management Service**.
    - Client-side encryption: encryption that occurs before data is sent to Cloud Storage.

2.2.Cloud SQL & Spanner

- Managed/No ops relational database (MySQL and PostgreSQL) like **Amazon RDS**.
- best for **gigabytes** of data with **transactional** nature
    - Low latency
    - Doesn't scale well beyond GB's
    - data structures and underlying infrastructure is required.
- **Spanner is a distributed and scalable solution** for RDBMS however also **more expensive**.
- Management:
    - Managed backups & automatic replication
    - fast connection with GCE/GAE
    - uses Google security
    - Flexible pricing, pay for when you use it

2.3.BigTable

Feature:
- Stored on Google's internal store **Colossus**
- **no transactional** support (so can handle **petabytes** of data)
- **not relational** (No SQL or joins), ACID only at row level.
    - Avoid schema designs that require atomicity across rows
- **high throughput**: Throughput has linear growth with node count if correctly balanced.
- work with it using **Hbase API**
- **no-ops**, auto-balanced, replicated, compacted,

Query:
- **Single key lookup**. No property search.
- Stored **lexicographically** in big endian format so keys can be anything.
- **quick range lookup**

Performance:
- Fast to **petabyte** scale, **not a good** solution for storing **less than 1 TB of data**.
- **Low-latency** read/write access
- High-throughput analytics
- **Native time series** support
- for large **analytical and operational** workloads
- designed for **sparse tables**

Key Design:
- Design your keys **how you intend to query**.
- If your most common query is the **most recent data**, use a **reverse date stamp** at the end of the key.
- Ensure your **keys are evenly distributed** to void **hot spotting**. This is why date stamps as a key or starting a key is bad practise as all of the most recent data is being written at the same time.
    - For **historical data analytic, hotspot issue** may not a biggest concern.
- For **time-series** data, use **tall/narrow** tables. Denormalize- prefer multiple tall and narrow tables
- **avoid hotspotting**
    - **Field promotion (preferred):** Move fields from the column data into the row key to make writes non- contiguous.
    - **Salting**: (only where field promotion does not resolve) Add an additional calculated element to the row key to artificially make writes non- contiguous.

Performance Test
- learns about your **access patterns** and will adjust the metadata stored in nodes in order to try balance your workloads.
- This takes minutes to hours and requires to use at least **300GB of data**
- use a **production** instance
- Stay **below the recommended storage utilization** per node.

- Before you test, run **a heavy pre-test** for several minutes
- Run your test for at least **10 minutes**

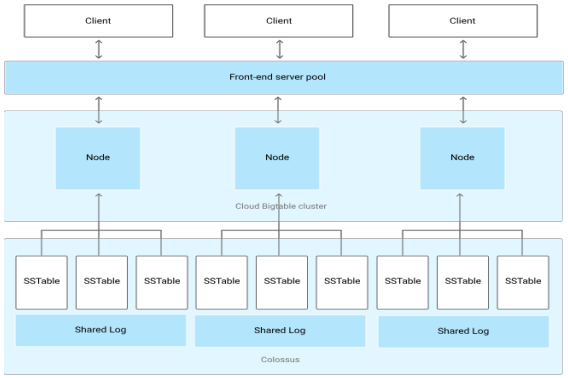
Data update
- When querying BigTable selects **the most recent value that matches the key**.
    - This means that when **deleting/updating we actually write a new row with the desired data** and compaction can remove the deleted row later. This means that deleting data will temporarily increase the disk usage
- append only (cannot update a single field like in CSQL/CDS)
- tables should be **tall and narrow** (store changes by appending new rows- tall, collapse flags into a single column - narrow)

Group columns
- **Group columns** of data you are likely to query together (for example address fields, first/last name and contact details).
- use **short column names**, organise into column families (groups e.g. MD:symbol, MD:price)

Periodically compacts
- BigTable periodically compacts the data for you. This means reorganising and removing deleted records once they're no longer needed.

Sorted String Tables
- BigTable relies on **Sorted String Tables** to organise the data.
- are immutable pre sorted **key,value pairs** of strings or **protobufs**.

Architecture



Size limits
- A single row key: 4 KB (soft limit?)
- A single value in a table cell: 100 MB
- All values in a single row: 256 MB

Access Control
- you can configure access control at the **project** level and the **instance** level(lowest IAM resources control)
    - A Cloud Bigtable **instance** is mostly just a container for your clusters and nodes, which do all of the real work.
    - **Tables belong to instances**, not to clusters or nodes. So if you have an instance with up to 2 clusters, you can't assign tables to individual clusters

Production & Development
- **Production**: A standard instance with **either 1 or 2 clusters**, as well as **3 or more nodes** in each cluster.
    - use **replication** to provide high availability
- **Development**: A low-cost instance for development and testing, with performance limited to the equivalent of a 1-node cluster.
- It is recommended to create your **Compute Engine instance** in the **same zone** as your **Cloud Bigtable** instance for the best possible performance (when use Cloud Bigtable with a Compute Engine-based application)

TOOLS
- **cbt** is a tool for doing basic interactions with Cloud Bigtable.
- **HBase shell** is a command-line tool that performs administrative tasks, such as creating and deleting tables.
- you can update any of the following settings without any downtime:
    - number of clusters / replication settings
    - upgrade a development instance to a production (permanent)
- Impossible to Switching between SSD and HDD
    - export the data from the existing instance and import the data into a new instance.
    - OR write a Cloud Dataflow or Hadoop MapReduce job that copies the data from one instance to another.

2.4.Datastore

- Built **on top of BigTable**.
    - non-consistent for every row
    - document DB for **non-relational** data
- Suitable:
    - **Atomic transactions**: can execute a set of operations where either all succeed, or none occur.
    - Supports **ACID transactions, SQL-like queries**.
    -
    - for structured data.
    - for **hierarchical** document storage such as HTML
- Query
    - can search **by keys or properties** (if indexed)
    - Key lookups somewhat like Amazon DynamoDB

    - Allows for **SQL-like querying** down to property level
    - does **not support complex joins** with multiple inequality filters
- Performance:
    - Fast to **Terabyte** scale, Low latency
    - Quick read, **slow write** as it relies on indexing every property (by default) and must **update indexes as updates/writes occur**

RDBMS/CloudSQL/ Spanner	Datastore
Row	Entity
Tables	Kind
Fields	Property
Column values must be consistent	Properties can vary between entities
Structured relational data	Structured hierarchical data (html, xml)

Errors and Error Handling
- UNAVAILABLE, DEADLINE_EXCEEDED
    - Retry using **exponential backoff**.
- INTERNAL
    - Do not retry this request more than **once**.
- Other
    - Do not retry without **fixing the problem**

2.5.BigQuery

Feature
- **Fully managed** data warehouse
- Has connectors for BigTable, GCS, Google Drive and can import from Datastore backups, CSV, JSON and ARVO
- for **analytics**. **serverless.**
- alternative to Hadoop with Hive

Performance
- **Petabyte** scale
- **High latency** used more for analytics than for low latency rapid lookups like a RDBMS like CloudSQL or Spanner

Query
- Standard SQL (preferred) or Legacy SQL (old)
- Cannot use both Legacy and SQL2011 in the same query
- Table partitioning
- Distributed writing to file for output. Eg: `file-0001-of-0002`
- user defined functions in JS (**UDFJS**)
- Query jobs are actions executed asynchronously to load, export, query, or copy data.
- If you use the **LIMIT** clause, BigQuery will still process the **entire table**.
- **Avoid SELECT *** (full scan), select only columns needed (SELECT * EXCEPT)
- benefits of using **denormalized data**
    - Increases query speed
    - makes queries simpler
    - BUT: Normalize is the way make dataset better organized but **less performance optimized**

types of queries:
- **Interactive**: query is executed immediately, counts toward daily/concurrent usage (default)
- **Batch**: batches of queries are queued and the query starts when idle resources are available, only counts for daily and switches to interactive if idle for 24 hours

Data Import
- **batch (free)**
    - web console (local files), GCS, GDS
- **stream (costly)**
    - data with CDF, Cloud logging or POST calls
- **Raw files:**
    - federated data source, CSV/JSON/Avro on GCS, Google sheets
- **Google Drive**
    - Loading data into BigQuery from Google Drive is not currently supported,
    - but can query data in Google Drive using an external table.
- By default, the BigQuery service expects all source data to be **UTF-8 encoded**
    - JSON files must always be encoded in UTF-8
- to support (occasionally) **schema changing** you can use '**Automatically detect**' for schema changes. Automatically detect is not default selected
- You cannot use the Web UI to:
    - Upload a file greater than 10 MB in size
    - Upload multiple files at the same time
    - Upload a file in SQL format

Partitions
- which improves query performance and reduces costs
- You **cannot change an existing table into a partitioned table**. You must create a partitioned table from scratch.
- Two **types** of partitioned tables:
    - **Ingestion Time**: Tables partitioned based on the data's ingestion (load) date or arrival date. Each partitioned table will have pseudocolumn_PARTITIONTIME, or time data was loaded into table. Pseudocolumns are reserved for the table and cannot be used by the user.
    - **Partitioned Tables**: Tables that are partitioned based on a TIMESTAMP or DATE column.

- Wildcard tables
    - Used if you want to union all similar tables with similar names. '*' (e.g. project.dataset.Table*)
- Partitioned tables include a pseudo column named _**PARTITIONTIME** that contains a date-based timestamp for data loaded into the table
    - It can be used to query specific partitions in the WHERE clause

Windowing:
- window functions increase the efficiency and reduce the complexity of queries that analyze partitions (windows) of a dataset by providing complex operations without the need for many intermediate calculations.
- They reduce the need for intermediate tables to store temporary data

Bucketing
- Like partitioning, but each split/partition should be the same size and is based on the hash function of a column. Each bucket is a separate file, which makes for more efficient sampling and joining data.

legacy vs. standard SQL[3]
- `project.dataset.tablename*`
- It is set **each time you run a query**
- default query language is
    - Legacy SQL for classic UI
    - Standard SQL for Beta UI

Anti-patterns

- Avoid self-joins
- Partition/Skew: **avoid unequally sized partitions**, or when a value occurs more often than any other value -
- **Cross-Join**: avoid joins that generate more outputs than inputs
- Update/Insert Single Row/Column: avoid point-specific DML, instead batch updates and inserts
- anti-patterns and schema design: https://cloud.google.com/bigtable/docs/schema-design

Access Control
- Security can be applied at **the project and dataset level**, but not table or view level
- three types of resources in BigQuery are organizations, projects, and datasets
- **Authorized views** allow you to share query results with particular users/groups without giving them access to underlying data
    - Can be used to restrict access to **particular columns or rows**
    - Create a separate dataset to store the view

Billing
- based on **storage** (amount of data stored), **querying** (amount of data/number of bytes processed by query), and **streaming** inserts
- **Storage options** are active and long-term (modified or not past 90 days).
- **Query options** are on-demand and flat-rate.

Table types:
- **Native tables:** tables backed by native BigQuery storage.
- **External tables:** tables backed by storage external to BigQuery(also known as a **federated data source**). For more information, see Querying External Data Sources.
- **Views:** Virtual tables defined by a SQL query. For more information, see Using views.

Caching:
- There is no charge for a query that retrieves its results from cache.
- BigQuery caches query results for 24 hours.
- By default, a query's results are cached unless:
    - When a destination table is specified
    - If any of the referenced tables or logical views have changed since the results were previously cached
    - When any of the tables referenced by the query have recently received streaming inserts (a streaming buffer is attached to the table) even if no new rows have arrived
    - If the query uses non-deterministic functions such as CURRENT_TIMESTAMP() and NOW(), CURRENT_USER()
    - If you are querying multiple tables using a **wildcard**
    - If the query runs against an external data source

Export:
- Data can only be exported in JSON / CSV / Avro
- The only compression option available is GZIP.
    - GZIP compression is not supported for Avro exports.
- To export more than 1 GB of data, you need to put a wildcard in the destination filename. (up to 1 GB of table data to a single file)

More refer to: https://cloud.google.com/bigquery/docs/
https://github.com/jorwalk/data-engineering-gcp/blob/master/know/bigquery.md

[3] https://cloud.google.com/bigquery/docs/reference/standard-sql/migrating-from-legacy-sql

3. Big Data Processing

Covering knowledge about BigQuery, Cloud Dataflow, Cloud Dataproc, Cloud Datalab and Cloud Pub/Sub.

3.1.App Engine

- run code on managed instances of machines with automated scaling and deployment
- Handle sudden and extreme spikes of traffic which require immediate scaling

3.2.GCP Compute

- VM
- **Pre-emptible** instances (up to 80% discount but can be taken away)
- **Allocated on-demand** and only pay for the time they are up

3.3.Dataflow

Feature

- Executes **Apache Beam** Pipelines(no-ops, could use **Spark, Flink**)
- Can be used for **batch or stream** data
- **scaleable**, **fault-tolerant**, multi-step processing of data
- Often used for data preparation/ETL for data sets
- filter, group, transform
- Pipelines and how it works for ETL

DataSource

- The **Cloud Dataflow connector** for Cloud Bigtable makes it possible to use Cloud Bigtable in a Cloud Dataflow pipeline.
- Can read data from **multiple sources** and can kick off multiple cloud functions in parallel writing to multiple sinks in a distributed fashion (eg Bigtable for low latency use and bigquery for data exploration).

Windowing

- Can apply **windowing** to streams for rolling average for the window, max in a window etc.
- **window types**
    - Fixed Time Windows
    - Sliding Time Windows (overlapped)
    - Session Windows
    - Single Global Window
- **Default** windowing behavior is to assign all elements of a PCollection to a **single, global window**, even for unbounded PCollections

Triggers

- IT determines when a Window's contents should be output based on certain criteria being met
    - Allows specifying a trigger to control when (in processing time) results for the given window can be produced.
    - If unspecified, the default behavior is to trigger first when the watermark passes the end of the window, and then trigger again every time there is late arriving data.
- **Time-Based Triggers**
    - **Event time triggers**. These triggers operate on the event time, as indicated by the timestamp on each data element. Beam's default trigger is event time-based.
    - **Processing time triggers**. These triggers operate on the processing time – the time when the data element is processed at any given stage in the pipeline.
- **Data-driven triggers**. These triggers operate by examining the data as it arrives in each window, and firing when that data meets a certain property.
    - Currently, data-driven triggers only support firing after a certain **number of data elements.**
- **Composite triggers**. These triggers combine multiple triggers in various ways.

Tech:

- **PCollections:** abstraction that represents a potentially distributed, multi-element data set, that acts as the pipeline's data
- A **transform** is a data processing operation, or a step, in your pipeline. A transform takes one or more PCollections as input, performs a processing function that you provide on the elements of that PCollection, and produces an output PCollection.
- **DirectPipelineRunner** allows you to execute operations in the pipeline directly & **locally**
- Create a cron job with **Google App Engine Cron Service** to run the Cloud Dataflow job

IAM

- **dataflow.developer** role enable the developer interacting with the Cloud Dataflow job , with data privacy.
- **dataflow.worker** role provides the permissions necessary for a **Compute Engine service account** to execute work units for a Dataflow pipeline

Pipeline Update

- With Update, to replace an existing pipeline in-place with the new one and preserve Dataflow's exactly-once processing guarantee
- When update pipeline manually, use **DRAIN** instead of CANCEL to maintain in-flight data.
    - The Drain command is supported for **streaming pipelines only**.
- pipelines cannot share data or transforms

Key things to focus on are:
- Event vs. Processing time
- Configuring ETL pipelines

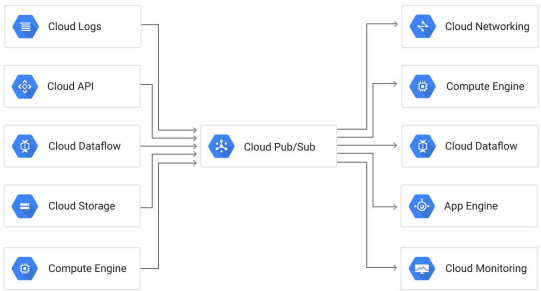
How to integrate with BigQuery

constraints you might have.

why you would use JSON or Java related to Pipelines.

3.4.Cloud Pub/Sub

Feature
- **server-less** messaging
- **decouples** producers and consumers of data in large organisations / complex systems
- the glue that connects all the components.
- **order not guaranteed**



Asynchronous processing
- **availability** (buffer during outages)
- **change management**
- **throughput** (balance load among workers)
- unification (cross-organisational)
- **latency** (accept requests at edge of network)
- consistency

basic concepts
- topics
- subscriptions
- push and pull
  - pull is a more efficiency message deliver/ consume mechanism

Message Flow
- Publisher **creates a topic** in the Cloud Pub/ Sub service and sends messages to the topic.
- Messages are **persisted** in a message store until they are **delivered and acknowledged** by subscribers.
- The Pub/Sub service forwards messages from a topic to all of its subscriptions, individually. Each subscription receives messages either by pushing/pulling.
- The subscriber receives pending messages from its subscription and acknowledges message.
- When a message is acknowledged by the subscriber, it is removed from the subscription's message queue.

Deduplicate
- Maintain a database table to store the hash value and other metadata for each data entry.
- Cloud Pub/Sub assigns a unique `message_id` to each message, which can be used to **detect duplicate messages** received by the subscriber.
- Lots duplicate messages may happen when: endpoint is not acknowledging messages within the **acknowledgement deadline**

3.5.Dataproc

Feature
- Dataproc is managed (**No Ops**) Hadoop cluster on GCP (i.e. managed Hadoop, Pig, Hive, Spark programs)
- automated cluster management. resizing
- Code/Query only
- Job management screen in the console.
- think in terms of a 'job-specific resource', for each job, create a cluster then delete it
- Used **if migrating existing on-premise Hadoop or Spark infrastructure** to Google Cloud Platform without redevelopment effort.

Storage
- Can store data on disk (**HDFS**) or can use **GCS**
- **GCS** allows for the use of preemptible machines that can reduce costs significantly
- - store in HDFS (split up on the cluster, but requires cluster to be up) or in GCS (separate cluster and storage)

Customize the software
- Set initialization actions
- Modify configuration files using cluster properties
- Log into the master node and make changes from there
- ~~NO Cloud Deployment Manager~~

Tech:
- creating a new Cloud Dataproc cluster with the projects.regions.clusters.create operation, these four values are required: **project, region, name, and zone**.
- you can access the YARN web interface by configuring a browser to connect through a **SOCKS** proxy
- You can **SSH directly** to the cluster nodes
- you can use a **SOCKS** proxy to connect your browser through an **SSH tunnel**.
- **YARN** ResourceManager and the **HDFS** NameNode interfaces are available on **master node**

Billing:
- is billed by the second. All Cloud Dataproc clusters are billed in **one-second clock-time increments**, subject to a **1-minute minimum billing**

IAM

- Service accounts used with Cloud Dataproc must have **Dataproc/Dataproc Worker** role (or have all the permissions granted by Dataproc Worker role).
    - need permissions to **read and write** to **Google Cloud Storage**, and to **write to Google Cloud Logging**.

3.6.Dataprep

- Managed Trifacta for **preparing and analysing quality and transforming** the input data
- service for visually **exploring, cleaning**, and preparing data for analysis. can transform data of any size stored in CSV, JSON, or relational-table formats

3.7.Cloud Functions:

- NodeJS functions as a service.
- No ops, no server just code entry point and response, autoscaled by GCP
- Can be triggered by dataflow, GCS bucket events, Pub/Sub Messages and HTTP Calls

4. Machine Learning[4]

Covering knowledge on GCP API (Vision API, Speech API, Natural Language API and Translate API) and Tensorflow.

- Embeddings
- Deploying Models

[4] This part is not well organised yet

- TensorFlow CheatSheet and Terminology

Understand the different ML services available: ML Engine, ML APIs, and TensorFlow, as well as the relevance of Cloud DataLab.

mostly ML domain (and not TensorFlow specific) basically about training. Nothing about the Cloud ML service

4.1.ML Terms:

- Label: The correct classification/value
- Input: Predictor Variables
- Example: Input + Label sample to train your model
- Model: Mathematical function, Some work is done on inputs for an output
- Training: Adjusting Variable weights in a model to minimise error
- Prediction: Using the model to guess the label for an input
- Supervised Learning: Training your model using examples data to predict future data
- Unsupervised Learning: Data is analysed without labels for patterns or clusters.
- Neuron: A way to combine inputs and weighting them to make a decision (it is one unit of input combination)
- Gradient Descent: The process of testing error in order to minimise its value iteratively decreases towards a minimum. (This can be global or local maximum and starting points and learning rates are important to ensure the process doesn't stop at local minima. Too low a learning rate and your model will tgrain slowly, too high and it may miss the minimum)
- Hidden Layer: A set of neurons that act on the same input data.
- Features: The data values/fields you choose to model, these can be transformed (x^2, y^2, etc)
- Feature Engineering: The process of building a set of feature combinations to act on inputs.
- Precision: The positive predictive value how many times it correctly predicted a thing as its classification (eg cat)
- Recall: The true positive rate, How many times a think IS in the class (the actual number of cats)
- Only recognised 1/10 cats, but was right. 100% precision, 10% recall

Recommendation Engine

- cluster similar users: User A and User B both rate House Z as a 4
- cluster similar items (products, houses, etc): Most users rate House Y as a 2
- combine these two to product a rating

4.1.ML Basics

- There are two main stages of ML, Training and Inference. Inference is often predictive in nature
- Common models tend to include regression (what value?) and classification (what category?)
- Converting inputs to vectors for analysis a well documented problem that can actually be improved with the use of ML itself.Some public models already exist incluing https://en.wikipedia.org/wiki/Word2vec
- Initial weight selection is hard to get right, human intuition can often help with this starting point for gradient descent
- Weights are iteractively tweaked. Initial weights -> Calculate Erorr -> adjust weight ->

recalculate error -> repeat until error is minimised.
- In Image ML each pixel is represented by a number, to vectorising images is actually much easier than text where you have to recognise that Man is to Woman as Boy is to Girl so they need to have similar magnitude differences.

Neural Network
- Goal is to minimize cost
- Cost depends on problem - usually the sum of square errors for regression problems or Cross_entropy on classification problems
- Feature Engineering less needed than linear models, but still useful
- Understand how to reduce noise

Wide & Deep Learning model
- The **wide** model is used for **memorization**, while the **deep** model is used for **generalization**
- Use for recommender system, search, and ranking problems.

Online training and/or continue learning
- build pipeline to continue train you model based on both new and old data,

Effective ML:
- Data Collection
- Data Organisation
- Model Creation using human insight and domain knowledge
- Use machines to flesh out the model from the input data
- Your Dataset should cover all cases both positive and negative and should have at least 5 cases of each otherwise the model cannot correctly classify that case. Near misses are also important for this, Explore your data, find causes of problems and try fix the issue. If it can't be fixed try find more `bad` cases to try train your model for it, Otherwise remove these from the data set.
- Overfitting and how to correct
- Neural network basics (nodes / layers)
- Use the Tensorflow playground to understand neural networks

Machine Learning on GCP
- Tensorflow - for ML researcher, use SDK
- CloudML - for Data Scientist, use custom model, scaleable, no-ops (where have enough data to train a model)
- ML APIs - for App Developer, use pre-built models, e.g. vision, speech, language (where would need more data to train a model)

4.2.ML Engine
- Train and predict ML models
- Can use multiple ML platforms such as **TensorFlow , scikit-learn and XGBoost**

Training cluster:
- The training service allocates the resources for the machine types you specify
- **Replica**: Your running job on a given node is called a **replica**.
    - each replica in the training cluster is given a single role or task in distributed training:

- **master**: Exactly one replica is designated the master.
    - This task manages the others and reports status for the job as a whole.
    - If you are running a single-process job, the sole replica is the master for the job.
- **workers**: One or more replicas may be designated as workers. These replicas do their portion of the work as you designate in your job configuration.
- **parameter servers:** One or more replicas may be designated as **parameter servers**. These replicas coordinate shared model state between the workers.
- **CUSTOM tier** for Cloud Machine Learning Engine allows you to specify the number of Workers and parameter servers

Online versus Batch Prediction
- **Online**
    - Optimized to **minimize the latency** of serving predictions.
    - Predictions returned in the response message.
    - Returns as soon as possible.
- **Batch**
    - Optimized to **handle a high volume** of instances in a job and to run more complex models.
    - Predictions written to output files in a **Cloud Storage location** that you specify.
    - Asynchronous request.

Exception
- The training service runs until your job succeeds or encounters an unrecoverable error.
- In the distributed case, it is the status of the master replica that signals the overall job status.
- **run a Cloud ML Engine training job locally** (gcloud ml-engine local train) is especially useful in the case of testing distributed models

4.3.TensorFlow
- OS Machine learning/Deep Learning platform
- **Lazy evaluate** during build, full

evaluate during execution

Tensorflow
- Machine learning library
- - Underpins many of Google's products
- - C++ engine and API (so can run on GPUs)
- - Python API (so can easily write code)
- To use Tensorflow:
- - Collect predictors and target data
- - discard info that identifies a row (need at least 5-10 examples of a particular value - to avoid overfitting)
- - predictor columns must be numerical (not categorial / codes)
- - Create model
- - how many nodes and layers do we need?
- - Train the model based on input data
- - Regression model predicts a number
- - Classification model predicts a category
- Use the model on new data

Feature Engineering
- **a sparse vector**
  - very long, with many zeros, contains only a single 1
- If you don't know the set of possible values in advance, you can use categorical_column_with_hash_bucket instead.
- An **embedding** is a mapping from discrete objects, such as words, to vectors of real numbers.

4.4.Datalab

- Datalab: Managed **Jupyter notebooks** great for use with a dataproc cluster to write pyspark jobs
- How to run Datalab

- - open source notebook built on Jupyter
- - use existing Python packages
- - also can insert SQL & JS for BigQuery, HTML for web content, charts, tables
- - analyse data in BQ, GCE, GCS
- - free, just pay for resources
- Three ways to run Datalab:
- - locally (good, if only one person using)
- - Docker on GCE (better, use by multiple people through SSH;CloudShell, uses resources on GCE)
- Docker + Gateway (best, uses a gateway and proxy, runs locally)

- a powerful interactive tool created to explore, analyze, transform and visualize data and build machine learning models on Google cloud platform.

5. Management

5.1.**IAM & Billing**

- 3 Member types, **Service account**, **google account** and **google group**.
  - Service Accounts are for non human users such as applications
  - Google Accounts are for single users
  - Google groups are for multi users
- **Project transfer fees** are associated with the instigator.
- Billing access can be provided to a project or set of projects without granting access to the content. This is useful for separation of duties between finance/devs etc.
- How billing works across projects

5.2.Stackdriver

- For store, search, analyse, monitor, and alert on log data and events.
- Be sure to know the sub-products of Stackdriver (Debugger, Error Reporting, Alerting, Trace, Logging), what they do and when they should be used.
- Hybrid monitoring service.
- how you can debug, monitor and log using Stackdriver.
- how to use Stackdriver to help debug source code
- **Audit Logs** to review data access (e.g. BigQuery)
- **Stackdriver Monitoring**
  - can see the usage of BigQuery query slots.
- **Stackdriver Trace**
  - is a distributed tracing system for Google Cloud Platform that collects latency data from Google App Engine, Google HTTP(S) load balancers, and applications instrumented with the Stackdriver Trace SDKs, and displays it in near real time in the Google Cloud Platform Console

5.3.Data Studio

- Data Dashboard
  - can use the existing **YouTube data source**
- The **prefetch cache** is only active for data sources that **use owner's credentials** to access the underlying data.
- Disabling the **query cache** could result in **higher data usage costs** for paid data sources, such as BigQuery
- you can turn the **prefetch cache off** for a given report. You might want to do this if:

    - your data changes frequently and you want to prioritize freshness over performance.
- you can turn the **prefetch cache on**, if
    - you are using a data source that incurs usage costs (e.g., BigQuery) and want to minimize those costs.

5.4. Cloudshell

- temporary VM
- recycled every 60 minutes (approx.)

5.5. Cloud Deployment Manager

- Allows you to specify all the resources needed for your application in a declarative format using yaml
- Repeatable Deployment Process

5.6. Data Transfer

- Storage Transfer Service
    - import **online data** into Cloud Storage.
    - repeating schedule
    - transfer data within Cloud Storage, from one bucket to another.
    - Source: **GCS, S3, URL**
- Transfer Appliance
    - one-time
    - Rack, capture and then ship your **offline** data to Google Cloud
- If you have a large number of files to transfer you might want to use the **gsutil -m** option, to perform a parallel (multi-threaded/multi-processing) copy

- **Compressing and combining smaller files** info fewer larger files is also a best practice for speeding up transfer speeds

Avro Data Format[5]:
- **Is faster to load**. The data can be read in parallel, even if the data blocks are compressed.
- Doesn't require typing or serialization.
- Is easier to parse because there are no encoding issues found in other formats such as ASCII.
- Compressed Avro files are not supported, but compressed data blocks are. BigQuery supports the DEFLATE and Snappy codecs.

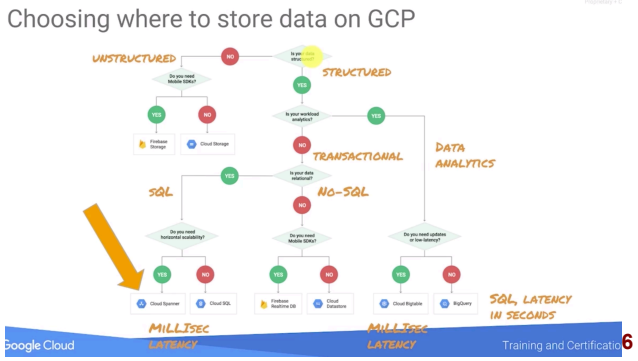
6. Product Selection

- if there is a requirement to search **terrabytes -> petabytes** of data relatively quickly it will make more sense to simply store in **BigQuery** (comparable to AWS Redshift).
-
- For **DataStore**, there is a possibility that this could work as a replacement for **Cassandra**.
- It is most likely that **BigTable** will be a better solution
- if the data set is relatively small < 10TB then **DataStore** will be preferred.
- If the data set is > 10TB and/or there is no requirement for **multiple indexes** then **BigTable** will be better.
-
- Be aware of any limitations regarding indexes and partitioning,
-

- Searching for objects by attribute value - **Datastore** (bigtable only by single row key)
- **High throughput writes** of wide column data: **Bigtable**
- Warehousing structured data - **BigQuery**

Google Platform Product	Service Function
Storage	Unified object storage
CloudSQL	Fully-managed MySQL database
BigTable	NoSQL massive data big data service
BigQuery	Petabyte scale data warehouse
Pub/Sub	Asynchronous messaging service
Cloud Dataflow	Data Processing (Pipelines)
Cloud DataProc	Managed Hadoop and Spark
TensorFlow	Machine learning language
Cloud Datastore	NoSQL database (think adhoc storage)

	Cloud Storage	Cloud SQL	Datastore	Bigtable	BigQuery
Capacity	Petabytes +	Gigabytes	Terabytes	Petabytes	Petabytes
Access metaphor	Like files in a file system	Relational database	Persistent Hashmap	Key-value(s), HBase API	Relational
Read	Have to copy to local disk	SELECT rows	filter objects on property	scan rows	SELECT rows
Write	One file	INSERT row	put object	put row	Batch/stream
Update granularity	An object (a "file")	Field	Attribute	Row	Field
Usage	Store blobs	No-ops SQL database on the cloud	Structured data from AppEngine apps	No-ops, high throughput, scalable, flattened data	Interactive SQL* querying fully managed warehouse



[5] https://cloud.google.com/bigquery/docs/loading-data-cloud-storage-avro

[6] https://cloud.google.com/storage-options/

7. Case Study

There are 2 case studies which are as same as in the GCP website: a logistic Flowlogistic company and a communications hardware MJTelco company. Each case study includes about 4 questions which ask how to transform current technologies of that company to use GCP technologies. We can learn details about these case studies in LinuxAcademy.

the correct answer is not necessary the best technical solution but a solution that achieves the right outcome for the company based on their current limitations.[7]

flowlogistic: https://cloud.google.com/certification/guides/data-engineer/casestudy-flowlogistic

- to find way to store the data that can be commonly used by Dataproc and BigQuery
    - Second both Dataproc and BigQuery integrating with **Cloud Storage** well
    -

mjtelco: https://cloud.google.com/certification/guides/data-engineer/casestudy-mjtelco

Deconstructing a Customer Case: Data Engineer Exam: https://www.youtube.com/watch?v=r_yYDysfB-k

Other helpful case:
Spotify's Event Delivery – The Road to the Cloud: https://labs.spotify.com/2016/02/25/spotifys-event-delivery-the-road-to-the-cloud-part-i/

8. Resources and references

Resources:

- Google Data Engineering Cheatsheet: https://github.com/ml874/Data-Engineering-on-GCP-Cheatsheet
- Data Engineering Roadmap: https://github.com/hasbrain/data-engineer-roadmap
- Whizlab Mock Exam
- Let me know any source is missing

Youtube Video to Watch:

Auto-awesome: advanced data science on Google Cloud Platform (Google Cloud Next '17): https://www.youtube.com/watch?v=Jp-qJFF9jww&list=PLIivdWyY5sqLq-eM4W2bIgbrpAsP5aLtZ

Introduction to Google Cloud Machine Learning (Google Cloud Next '17): https://www.youtube.com/watch?v=COSXg5HKaO4
Introduction to big data: tools that deliver deep insights (Google Cloud Next '17): https://www.youtube.com/watch?v=dlrP2HJMlZg
Easily prepare data for analysis with Google Cloud (Google Cloud Next '17): https://www.youtube.com/watch?v=Q5GuTIgmt98
Serverless data processing with Google Cloud Dataflow (Google Cloud Next '17): https://www.youtube.com/watch?v=3BrcmUqWNm0

Data Modeling for BigQuery (Google Cloud Next '17) https://www.youtube.com/watch?v=Vj6ksosHdhw
Migrating your data warehouse to Google BigQuery: Lessons Learned (Google Cloud Next '17): https://www.youtube.com/watch?v=TLpfGaYWshw

Webinar: Building a real-time analytics pipeline with BigQuery and Cloud Dataflow (EMEA): https://www.youtube.com/watch?v=kdmAiQeYGgE

[7] https://www.linkedin.com/pulse/google-cloud-certified-professional-data-engineer-writeup-rix/