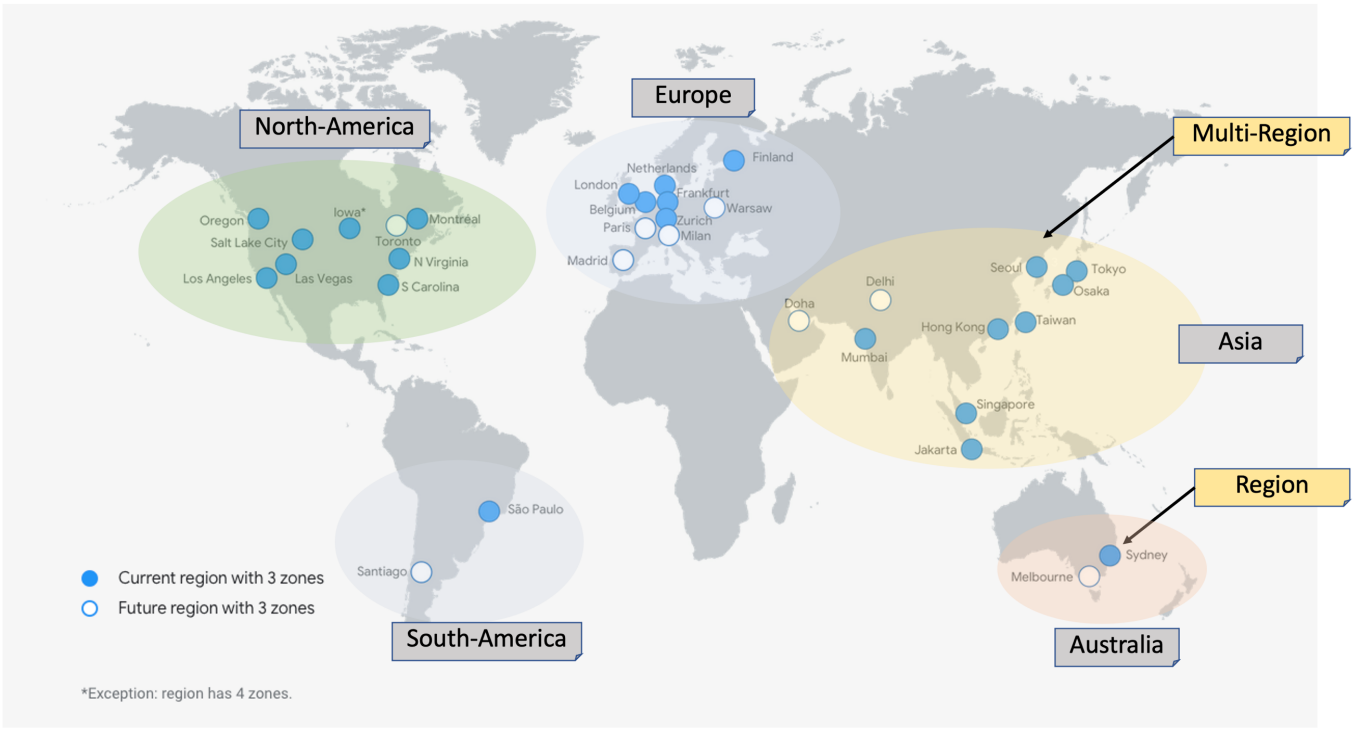
Google Cloud Platform – Products
2021

Table of Contents

1 GCP Structure and Design

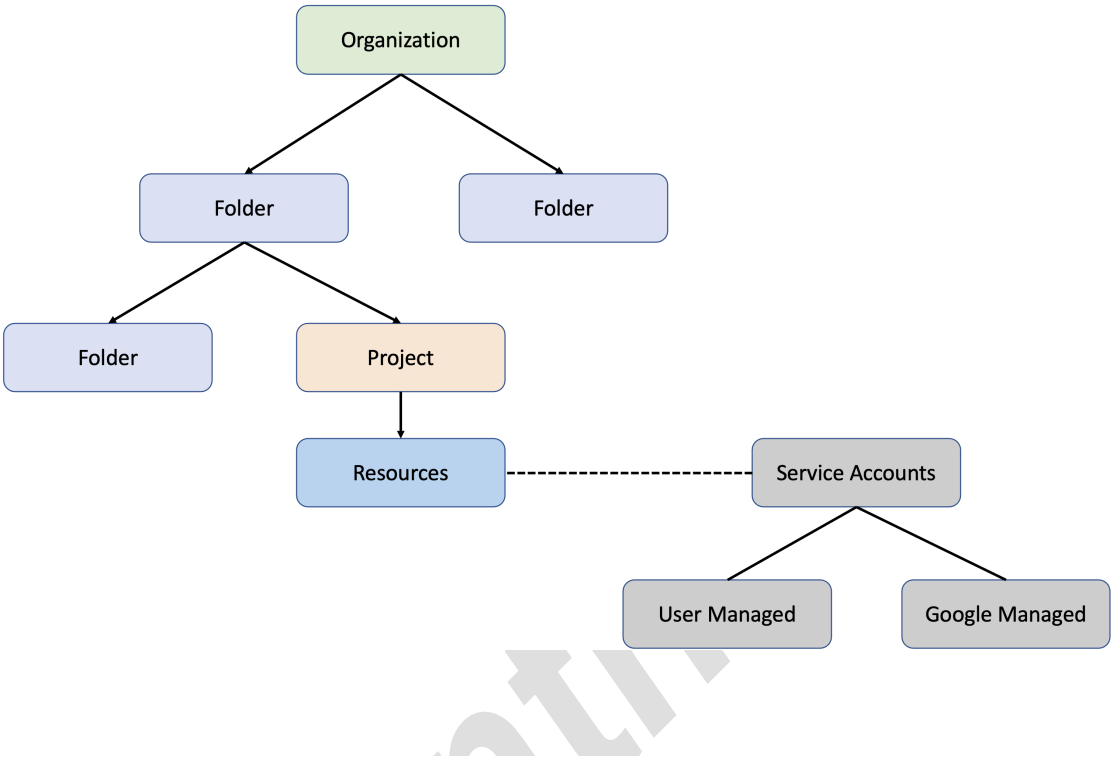
- GCP is intrinsically global
  - Easier to handle latency and failures in a global way
- AWS is region-scoped
  - Simplifies data sovereignty

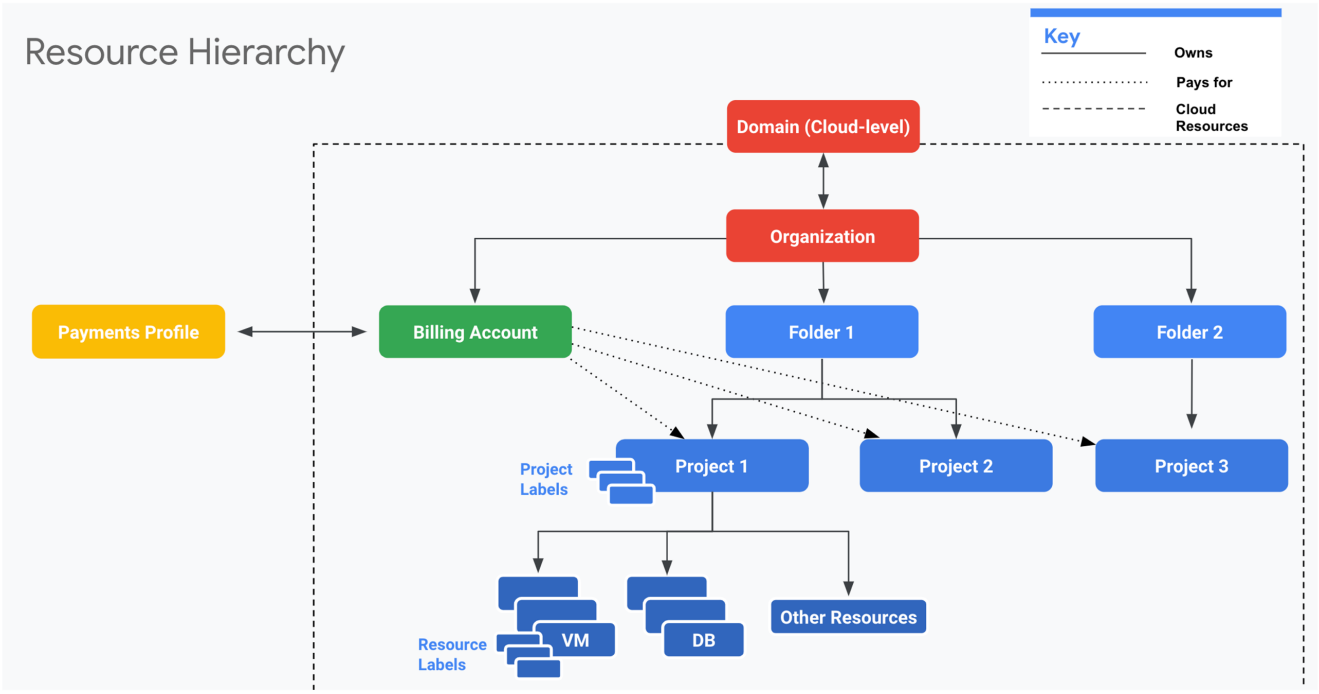
- **GCP Physical Infrastructure**
  - vCPU
  - Physical server
  - Rack
  - Data Center(building)
  - Zone
    - one or more data centers
    - Independent of each other
  - Region
    - One or more zones grouped together
    - Comms between zones in a region is very fast
  - Multi-Region
    - Regions grouped together
    - Each region is at least 100 miles away from another within a multi-region.
  - Private global network
    - All google regions and zones are connected by a private global network.
    - No public internet access required
  - Points of Presence (PoPs)
    - Network edges and CDN locations.
    - This is where GCP interacts with public internet.

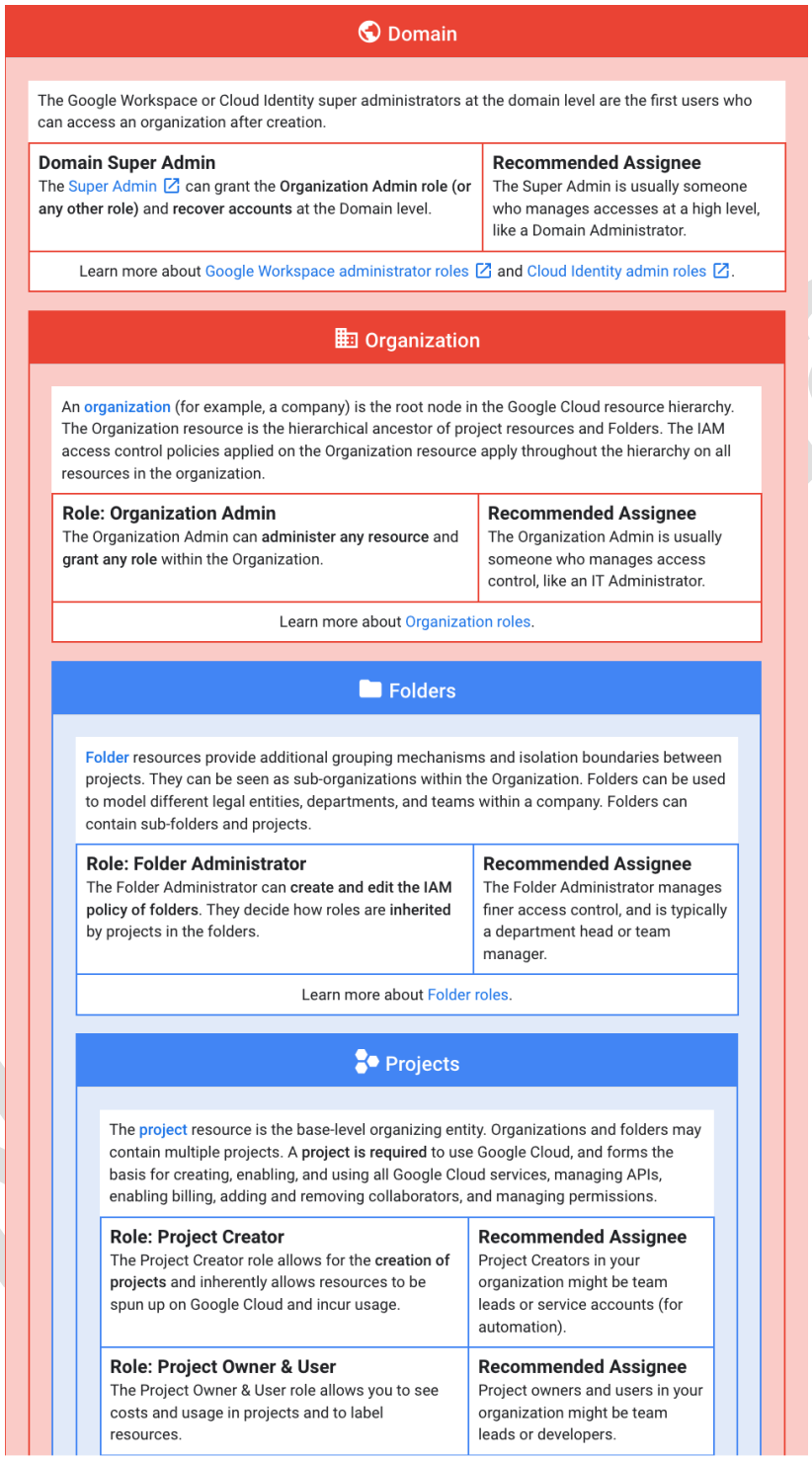

North-America
Europe
Multi-Region
Asia
Region
South-America
Australia
Oregon
Salt Lake City
Los Angeles
Las Vegas
Iowa*
Montréal
Toronto
N Virginia
S Carolina
Netherlands
Finland
London
Frankfurt
Belgium
Warsaw
Paris
Zurich
Milan
Madrid
Seoul
Tokyo
Osaka
Delhi
Doha
Hong Kong
Taiwan
Mumbai
Singapore
Jakarta
São Paulo
Santiago
Sydney
Melbourne
Current region with 3 zones
Future region with 3 zones
*Exception: region has 4 zones.


Americas
us-central1
'us-central1-a'
'us-central1-b'
'us-central1-c'
'us-central1-f'
us-west1
'us-west1-a'
'us-west1-b'
'us-west1-c'
us-east1
'us-east1-b'
'us-east1-c'
'us-east1-d'
us-east4
'us-east4-a'
'us-east4-b'
'us-east4-c'
Europe
europe-west1
'europe-west1-b'
'europe-west1-c'
'europe-west1-d'
europe-west3
'europe-west3-a'
'europe-west3-b'
'europe-west3-c'
europe-west2
'europe-west2-a'
'europe-west2-b'
'europe-west2-c'
Asia
asia-southeast1
'asia-southeast1-a'
'asia-southeast1-b'
asia-east1
'asia-east1-a'
'asia-east1-b'
'asia-east1-c'
asia-northeast1
'asia-northeast1-a'
'asia-northeast1-b'
'asia-northeast1-c'
Australia
australia-southeast1
'australia-southeast1-a'
'australia-southeast1-b'
'australia-southeast1-c'

1.1 GCP Organization






Domain
The Google Workspace or Cloud Identity super administrators at the domain level are the first users who can access an organization after creation.
Domain Super Admin
The Super Admin can grant the Organization Admin role (or any other role) and recover accounts at the Domain level.
Recommended Assignee
The Super Admin is usually someone who manages accesses at a high level, like a Domain Administrator.
Learn more about Google Workspace administrator roles and Cloud Identity admin roles.
Organization
An organization (for example, a company) is the root node in the Google Cloud resource hierarchy. The Organization resource is the hierarchical ancestor of project resources and Folders. The IAM access control policies applied on the Organization resource apply throughout the hierarchy on all resources in the organization.
Role: Organization Admin
The Organization Admin can administer any resource and grant any role within the Organization.
Recommended Assignee
The Organization Admin is usually someone who manages access control, like an IT Administrator.
Learn more about Organization roles.
Folders
Folder resources provide additional grouping mechanisms and isolation boundaries between projects. They can be seen as sub-organizations within the Organization. Folders can be used to model different legal entities, departments, and teams within a company. Folders can contain sub-folders and projects.
Role: Folder Administrator
The Folder Administrator can create and edit the IAM policy of folders. They decide how roles are inherited by projects in the folders.
Recommended Assignee
The Folder Administrator manages finer access control, and is typically a department head or team manager.
Learn more about Folder roles.
Projects
The project resource is the base-level organizing entity. Organizations and folders may contain multiple projects. A project is required to use Google Cloud, and forms the basis for creating, enabling, and using all Google Cloud services, managing APIs, enabling billing, adding and removing collaborators, and managing permissions.
Role: Project Creator
The Project Creator role allows for the creation of projects and inherently allows resources to be spun up on Google Cloud and incur usage.
Recommended Assignee
Project Creators in your organization might be team leads or service accounts (for automation).
Role: Project Owner & User
The Project Owner & User role allows you to see costs and usage in projects and to label resources.
Recommended Assignee
Project owners and users in your organization might be team leads or developers.

Cloud Billing account

Cloud Billing accounts are linked to and pay for **projects**. Cloud Billing accounts are connected to a Google Payments Profile.

Role: Billing Account Admin The Billing Account Admin can enable Billing Export, view cost/spend, set budgets and alerts, and link/unlink projects.	**Recommended Assignee** The Billing Admins in your organization may be someone more finance-minded.
Role: Billing User Billing Users can **link projects to Cloud Billing accounts**, but cannot unlink them. It is usually issued broadly in concert with the **Project Creator** role.	**Recommended Assignee** Trusted Project Creators in your Organization typically need this role.

Learn more about Billing roles.

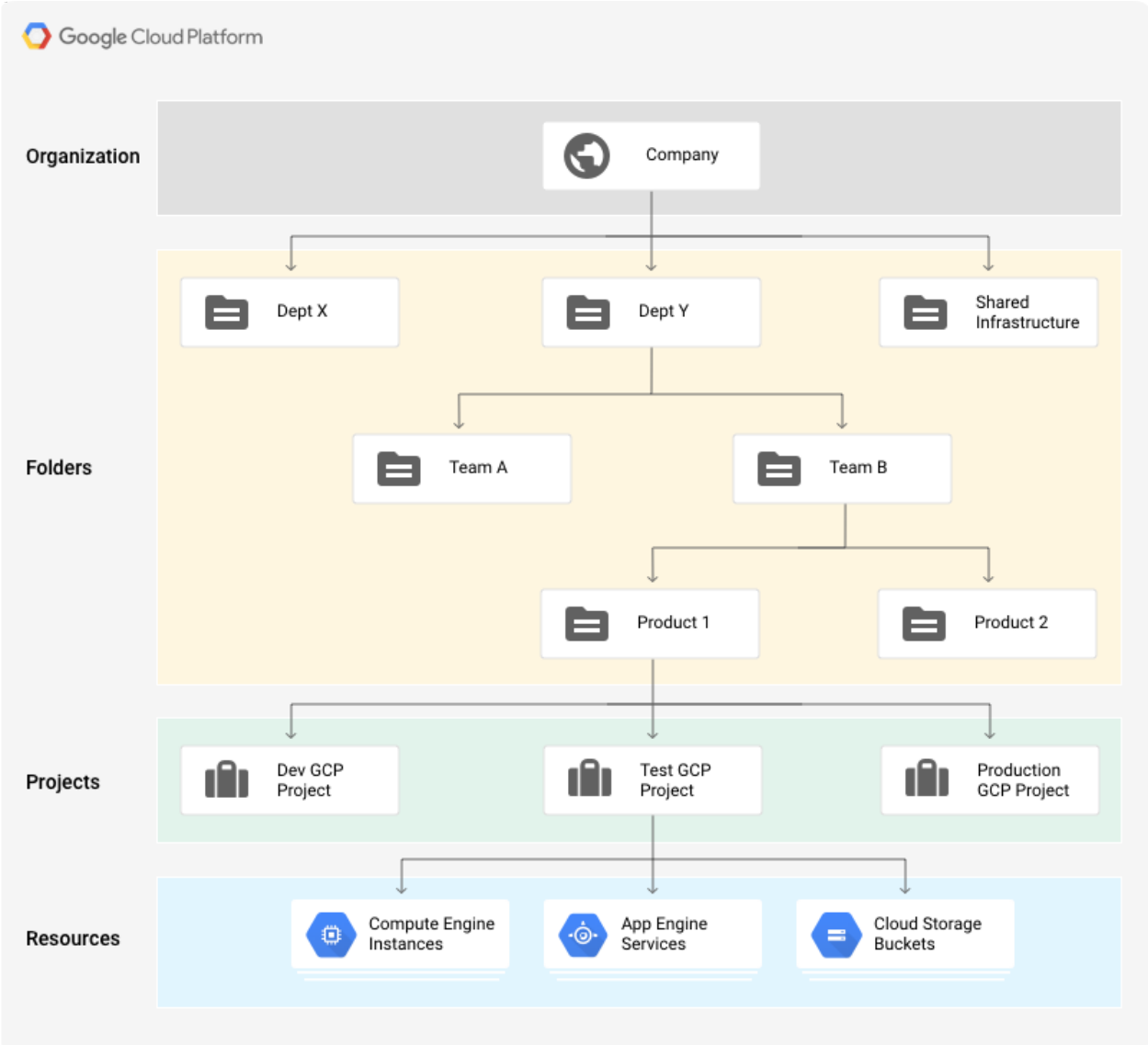
Payments Profile

Payments Profiles are **managed outside of your Cloud Organization**, in the Google Payments Center, a single location where you can manage the ways you pay for all Google products and services, such as Google Ads, Google Cloud, and Fi phone service. Payments Profiles are connected to **Cloud Billing accounts**.

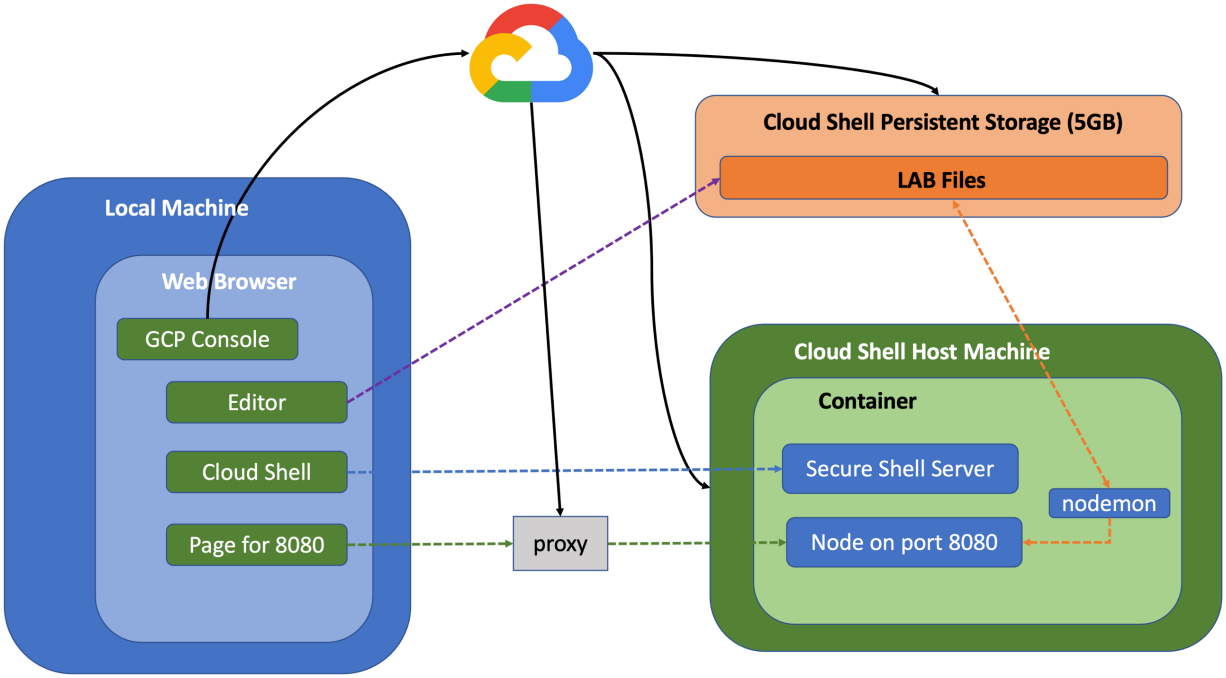
Payments Profile Admin The Payments Profile Admin can view and manage payment methods, make payments, view invoices, and see Payments Accounts.	**Recommended Assignee** The Payments Profile Admins in your organization are typically part of your Finance or Accounting teams.

Learn more about Payments Profile user permissions.

1.1.1 Example resource hierarchy



1.2 Cloud Shell Overview



1.3 Key security products and features (AAA)

- **Authentication**
  - Identity
    - Humans use G-suite, cloud identity
    - Applications and services use service accounts
  - Identity hierarchy
    - Uses google groups
  - If customer manages identity
    - Use Google Cloud Directory Sync (GCDS) to pull data from LDAP.

- **Authorization**
  - Identity hierarchy
    - Uses google groups
  - Resource hierarchy
    - Organization, folders, projects
  - Identity and access management (IAM)
    - Permissions, roles, bindings
  - GCS has Access Control Lists (ACLs)
  - Billing management
  - Network structure and restrictions

- **Accounting**
  - Audit\Activity log
    - Provided by Stackdriver
  - Billing export
    - To BigQuery
    - To a file in a GCS bucket
      - In JSON or CSV format
  - GCS Object Lifecycle Management

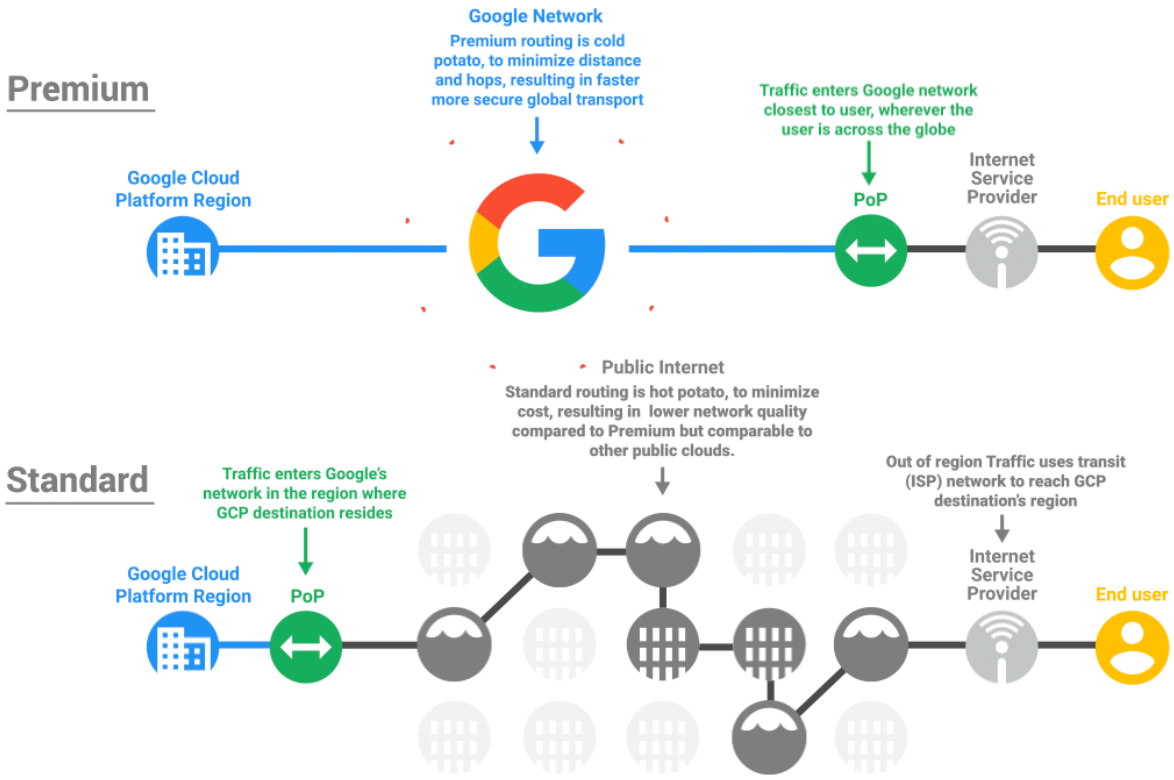
1.4 GCP Networking

- Routing is about making local decisions about where data should go next.
- Routing decisions involve data flow as follows:
    - Client system to google network
    - Google network to the right resource
    - Resource to resource

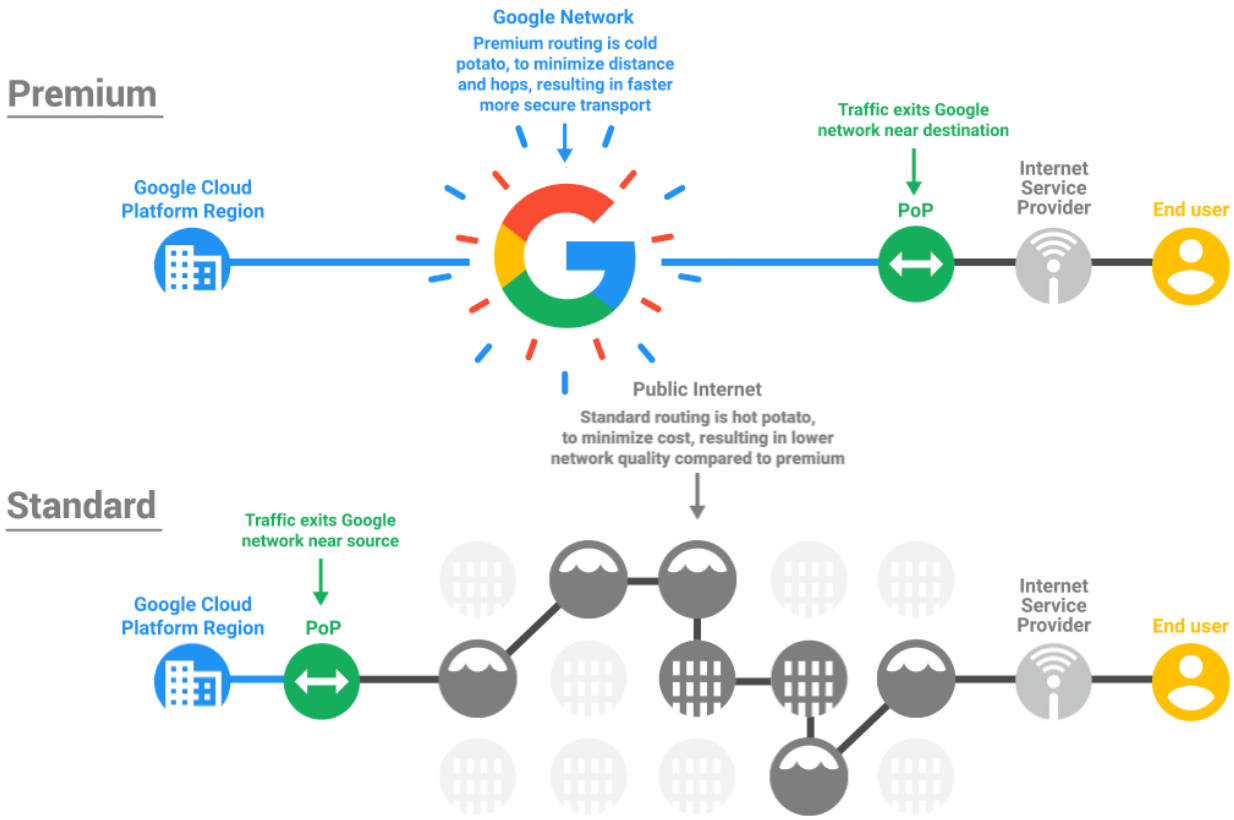
Hot Potato and cold potato routing

- Hot potato routing
    - Always tries to make it someone else's problem
    - Used in standard routing
- Cold Potato routing
    - Opposite of hot potato
    - Always takes up task of routing to closest destination to target
    - Used in premium routing

1.4.1 Client system to Google network



1.4.2 Google network back to client system

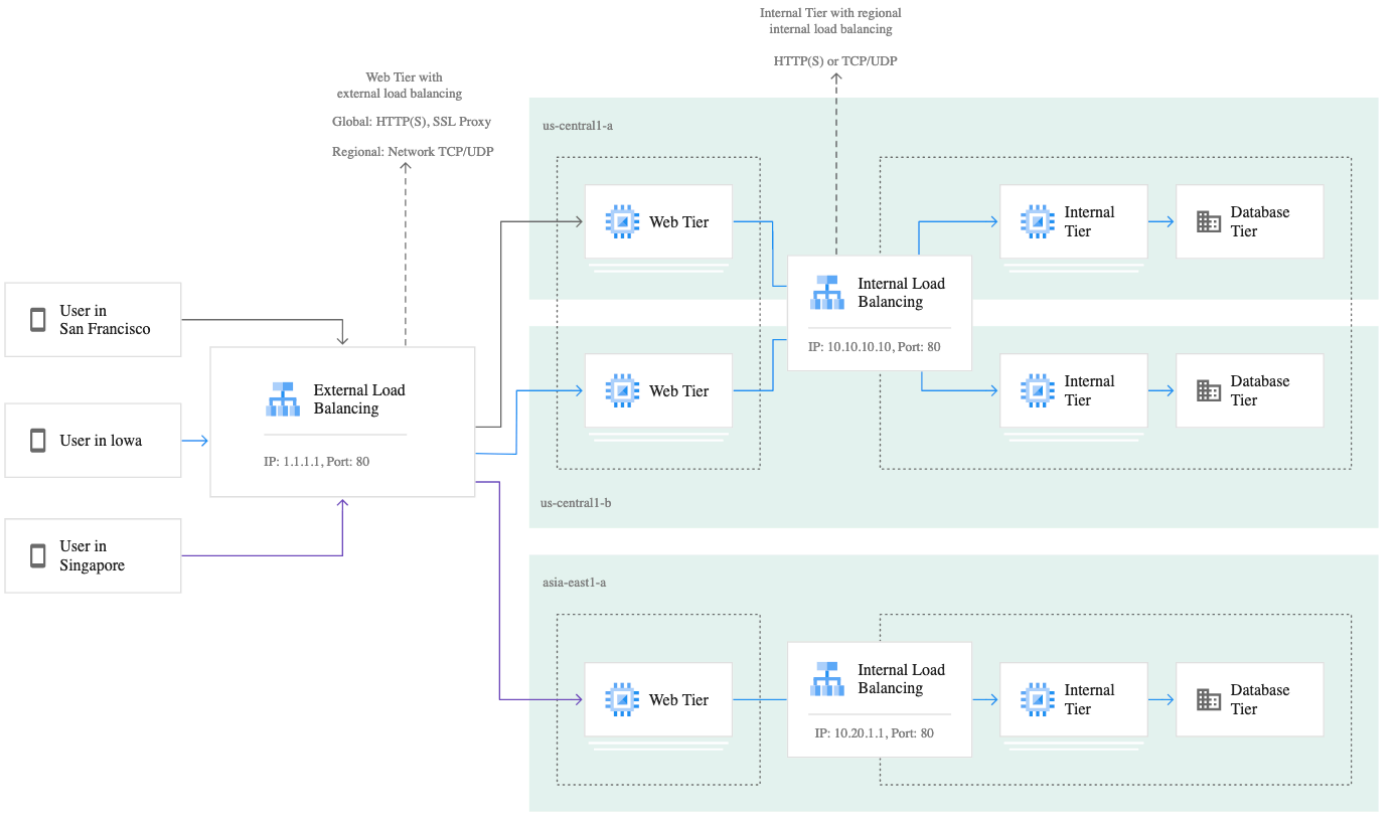


PoP: Point of presence

1.4.3 Routing to the right resource

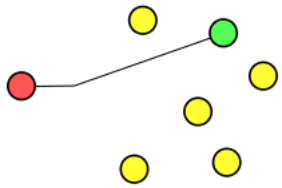
Reasons to route to different resources

- Latency reduction
  - Use servers physically closest to user
  - Achieved using **cross-region load balancing with global anycast IPs**.
    - In premium tier
- Load balancing
  - Achieved using cloud load balancer
    - Internal \ external
    - Layer 4 and layer 7
- System design
  - Micro-services
  - Achieved using HTTP(s) load balancer with a URL map

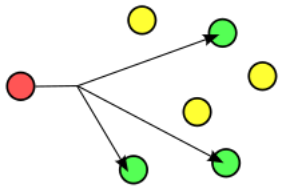


1.4.4IPV6 Address types

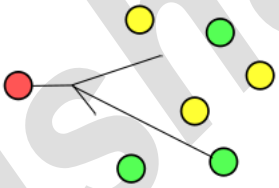
- **Unicast**
  - Refers to a single host
  - Meant to send data to a single destination



- **Multicast**
  - Used to deliver packet to a group of destinations.
  - Any packet sent to a multicast address will be delivered to every host that has joined that particular group.



- **Anycast**
  - Similar to multicast, but packet is delivered to only one single host.
  - In GCP, it is delivered to the host closest to the user.



1.4.4.1 Layer 4 vs Layer 7

- TCP (of TCP\IP) is usually called Layer 4(L4)
    - This layer works solely with IP addresses.
- HTTP and HTTPs work at layer 7 (L7)
    - Application layer
    - These know about URLs and paths
- To route based on URL paths (example.com/cart), routing to needs to understand L7
- L4 cannot route based on URL paths defined in L7.
- Name resolution using DNS (domain name system) can be the first step in routing.
    - This converts the name to an IP address.
    - This is a L4 IP address. So cannot route based on L7 URL paths.
    - Problem:
        - Often cached and reused for huge client sets.
        - Sticky – Has a TTL. Locks on to a target.
        - Not robust. Some clients don't honor TTL.

- ***GCPs premium tier "cold potato" routing with global anycast IPs avoid these problems.***

1.4.5 Routing among resources

- **Virtual Private Cloud (VPC)**
    - Global resource
    - Private SDN space in GCP
    - Manages not only resource to resource interaction, also manages peer networks and external to internal interactions.
    - Defines overall network.

- **Subnets**
    - Regional
    - Divides up space within a VPC to create logical spaces to contain resources.
    - All subnets can reach all others globally without the need for VPNs.

- **Routes**
    - Global
    - Route traffic within a VPC
        - Define next hop for traffic based on destination IP
    - Even though instance level network tags can be defined, routes are not scoped down to a subnet, but available to entire VPC

- **Firewall Rules**
  - Global and scoped at VPC level
  - They can be defined by instance level network tags or at service account level.
  - By **default**, they block all incoming data (ingress) and allow all outgoing data (egress)

1.4.6 Decimal to binary

2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
256	128	64	32	16	8	4	2	1

Example:
12 → 8 + 4 → 1100

2^3	2^2	2^1	2^0
8	4	2	1
1	1	0	0

124 → 124 – 64 = 60
60 – 32 = 28
28 – 16 = 12
12 - 8 = 4 → 64 + 32 + 16 + 8 + 4 → 1111100

255 → 11111111

1.4.7 IP Address and CIDR Blocks (Classless Inter-Domain Routing)

- IP Address is a dotted quad: ***abc.def.ghi.jkl***
- Each part ranges between 0-255 (8 bits)
- If we convert IP to binary, we get a 32-bit binary number
- CIDR block is a group of IP addresses specified in <IP>/xy format
    - xy ranges from 0 to 32
    - It locks the highest (leftmost) bits in the IP address.

- **Example 1**: abc.def.ghi.jkl/32
    - This represents a single IP address because all 32 bits are blocked.

- **Example 2**: abc.def.ghi.jkl/24
    - Here, abc.def.ghi are locked i.e. 24 bits
    - Jkl can vary i.e. from 0-255
    - This represents a group of 256 IP addresses.

- **Example 3**: 0.0.0.0/0
    - Represent “any IP Address”
    - /0 implies no bits are locked

- **RFC1918** defines private address ranges you can use to be:
    - 10.0.0.0/8 (approx. 16 Million) – Single Class A network
    - 172.16.0.0/12 (approx. 1 Million) – 16 contiguous Class B networks
    - 192.168.0.0/16 (approx. 65K) – 256 contiguous Class C networks

1.5 Initial Setup

- Setup cloud CLI
- Setup VM instances
- SSH to VM and setup the following
    - Git
    - Download git repo
    - Install missing libraries
    - Download data
- Create cloud storage bucket
- Copy files from VM to cloud storage

1.6 Choosing the right service

- Compute Engine
    - Individual VMs
    - IaaS
- GKE
    - Containerized clusters
- App Engine
    - PaaS framework
    - Managed
    - Long lived web apps
- Cloud functions
    - Functions as a service
    - Serverless
    - Event based
    - Short lived events
- Database
    - User\custom
        - Spin up VM using compute engine
        - Install and manage DB
    - Managed
        - Bigtable
        - Cloud storage
        - Cloud SQL
        - Spanner
        - Datastore
- Managed big data services
    - Bigquery
    - Cloud Pub\sub
    - Cloud dataflow
    - Cloud Dataproc
    - Cloud Datalab

1.7 Key Roles in a ML\DS team

2 Storage Systems

GCP has several storage services

- **Cache**
  - Used to store small amounts of data and very quick access.
  - Managed **Redis** Service called Memory store
    - 1GB to 300 GB
    - Used with compute engine, App Engine, Kubernetes Engine

- **Persistent Disk\storage**
  - Provides durable block storage
  - Can be attached to VMs in GCE or GKE
  - Up to 64 TB. Data is automatically encrypted.
  - Provides hundreds of IOPS for read and write.
  - Two types:

    - **Solid State Drive (SSD)**
      - High throughput and consistent performance for both random access and sequential access

      - **Network Accessible**
        - Not directly attached to physical server hosting VMs
        - Data continues to exist after VMs are shut down.

      - **Locally mounted**
        - VMs can have locally attached SSDs.
        - Data is lost once VM is shut down.

    - **Hard Disk Drive (HDD)**
      - Longer latency, but cost less
      - Good for large amounts of data

      - **Network Accessible**
        - Not directly attached to physical server hosting VMs
        - Data continues to exist after VMs are shut down.

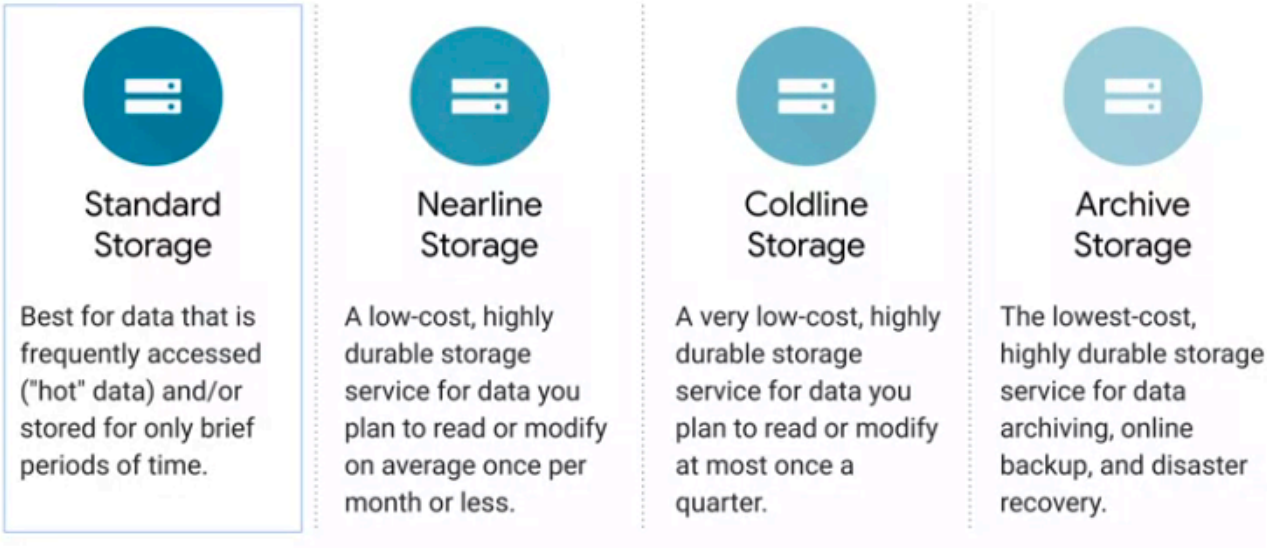
      - **Locally mounted**
        - Not supported.

- Storage options
  - **Zonal**
    - Data stored on multiple physical drives in a single zone.
    - If zone is not accessible, lose access to disks.
  - **Regional**
    - Replicate data across 2 zones within a region.
    - More expensive than zonal.

- **Object Storage and Archival**
  - Used to store very large volumes of data. In exabytes.

  - **Cloud Storage**
    - Google offers cloud storage as the object store.
    - Bucket is a logical unit of organization in cloud storage.
    - Buckets are resources within a project
    - Bucket name should be globally unique.
    - **Four classes of object storage**:

	Regional	**Multi-Region**	**Nearline**	**Coldline**
Features	• Object store replicated across multiple zones • 99.99% available • Can be changed to near line or Coldline	• Replicated across multiple regions • 99.99% available • Data is geo-redundant i.e stored more than 100 miles apart. • Can be changed to nearline or Coldline.	• Storage for access less than once a month • Min 30-day storage duration • Can be changed to only Coldline.	• Storage for access less than once a year • Min 90-day storage duration • Cannot be changed.
Storage cost	$0.20/GB/Month	$0.26/GB/Month	$0.10/GB/Month	$0.07/GB/Month
Access cost			$0.01/GB	$0.05/GB
Use case	Storage shared across applications	Global access to shared objects	Backup data, data lakes.	Compliance data, document retention.



2.1 Storage based on data model

- **Object Storage**
  - Used for large volumes of data
  - Files are atomic and cannot be broken down further.
  - Example
    - **Cloud storage**

- **Relational Database**
  - Used when consistency is critical.
  - Examples
    - **Cloud SQL**
      - Managed database service
      - Supports MySQL and PostgreSQL.
      - Supports transactions.
      - No horizontal scaling. (i.e cannot add more servers)
      - Only vertical scaling. (i.e can add additional memory and CPU)
      - Used for **web apps**, business intelligence and eCommerce apps.
    - **Cloud Spanner**
      - Managed database service
      - Supports transactions.
      - Can scale horizontally and vertically
      - Used for very large volumes of relational data that needs to be globally distributed.

        - Used by large **enterprises** for global supply chains, finance etc.
    - **Big Query**
        - Managed analytics service
        - Not suitable for transaction-oriented apps.
        - Can scale horizontally and vertically
        - Used for very large volumes of data. (In petabytes)
        - Designed for data warehouse and **analytic** applications.

- **No SQL**
    - Examples
        - **Cloud datastore**
            - Managed database service
            - Document database (i.e key value pairs)
            - Used for non-analytical, non-relational storage
            - Supports transactions and indexing
            - Does not support relational operations like joins or aggregates like sum and counts.
            - Terminology:
                - Namespace ➔Schema Name
                - Kind ➔ Table name
                - Entity ➔ Key value pair in document
        - **Cloud Firestore**
            - Managed database service
            - Document database (i.e key value pairs)
            - Designed to store, synchronize and query data across distributed applications like **mobile** apps.
            - Supports transactions and multi-region replication.
            - No need to configure instances.
            - Choose one of two modes
                - Datastore mode
                - Native mode
        - **Big Table**
            - Wide column database
                - Use large number of columns
                - Each row can have different number of columns
            - Designed for **petabyte** scale data
            - Provides consistent low millisecond latency
            - Scales horizontally and across regions.
            - Designed for applications with high volume, high velocity data
                - Time series, IoT and financial apps
                - Data from transactional systems are extracted, transformed and loaded into big table for analytics.

2.2 How to choose storage Solution

	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	Data warehouse
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

Cloud Storage

- Use as a global file system

Cloud SQL

- Use as a RDBMS system
- For transactional, relational data accessed through SQL

Datastore

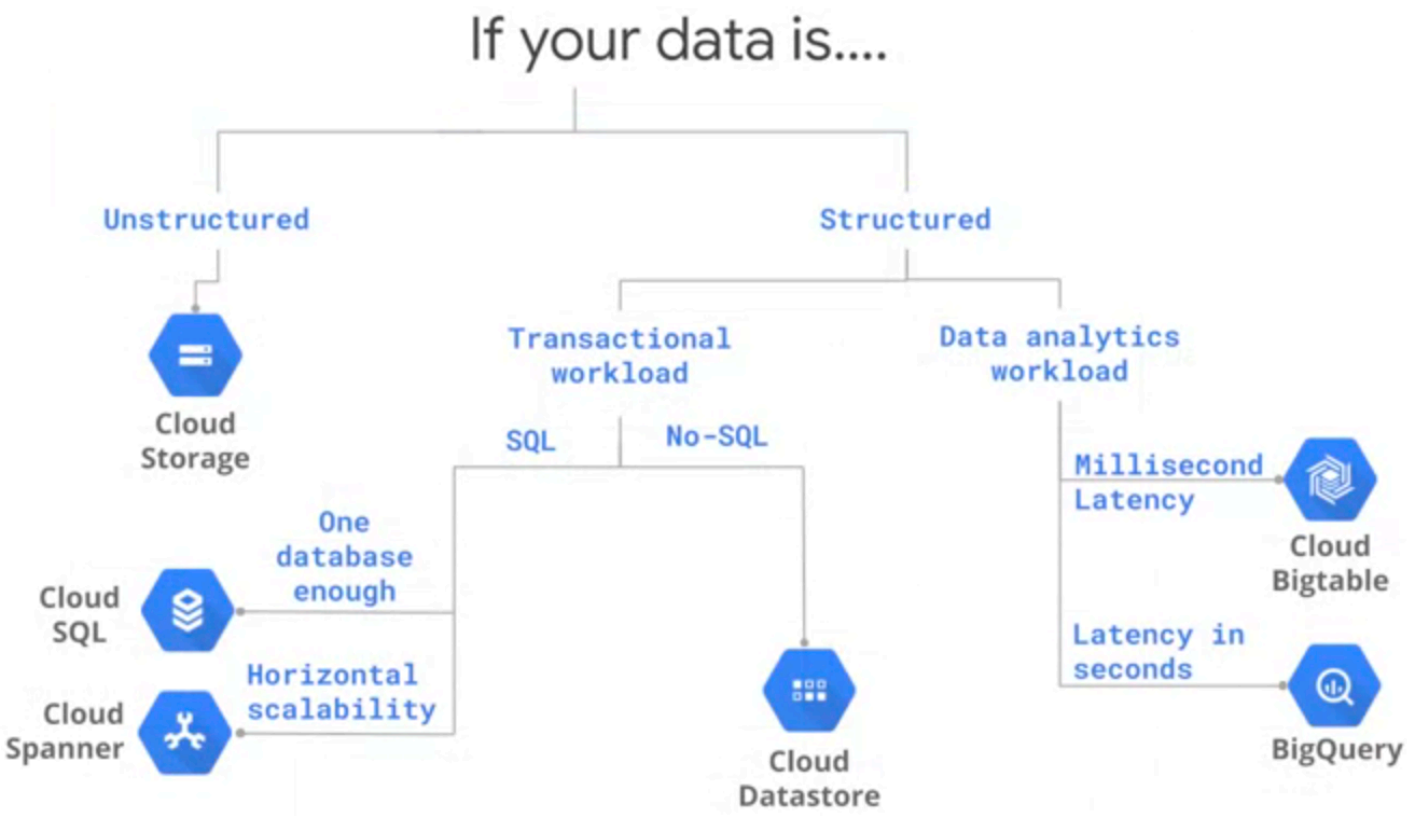
- Transactional, NoSQL, object-oriented database

BigTable

- High throughput, NoSQL, append only
- Eg: sensor data

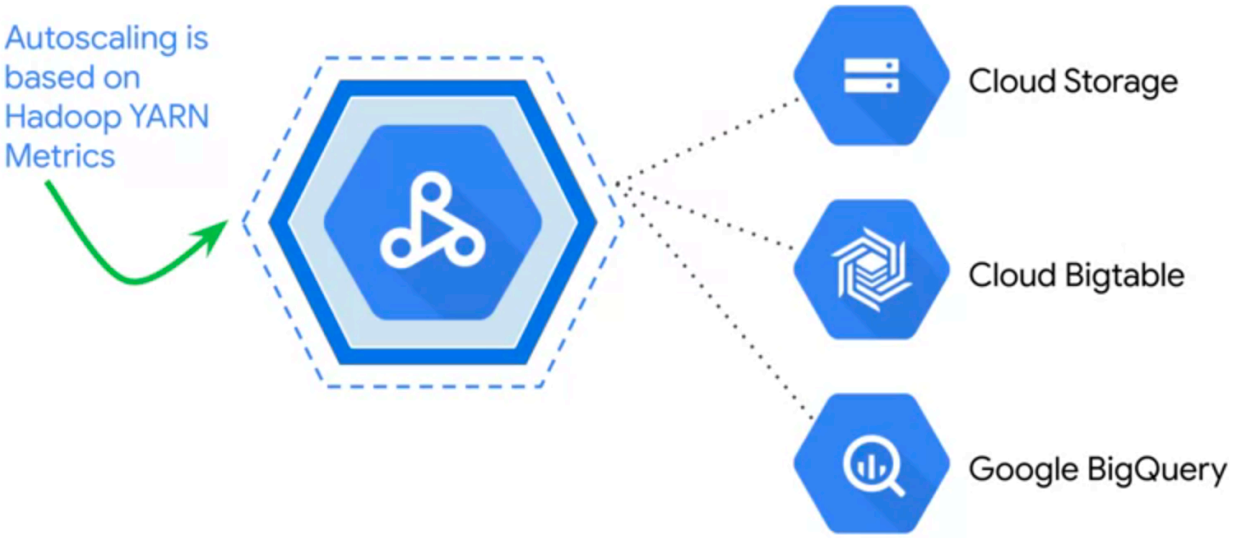
BigQuery

- SQL data warehouse
- Analytics

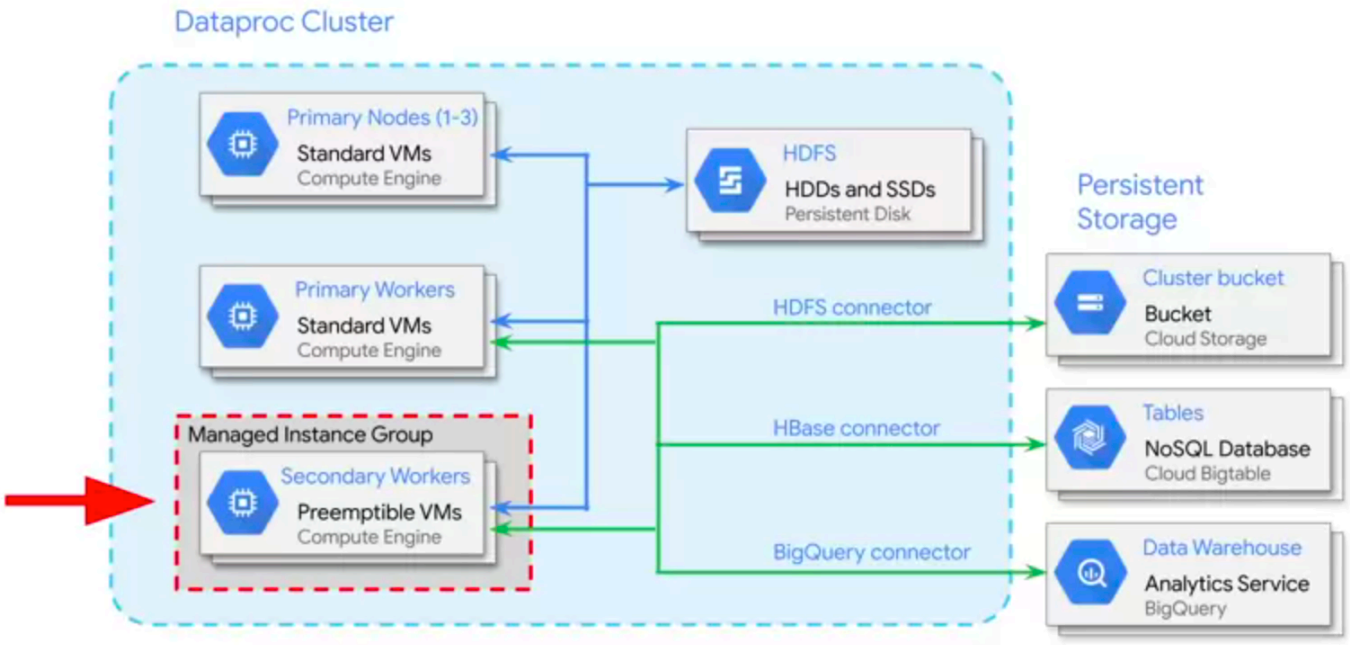

If your data is....
Unstructured
Structured
Cloud Storage
Transactional workload
Data analytics workload
SQL
No-SQL
Millisecond Latency
Cloud Bigtable
One database enough
Cloud SQL
Horizontal scalability
Cloud Spanner
Cloud Datastore
Latency in seconds
BigQuery

3 Cloud Dataproc

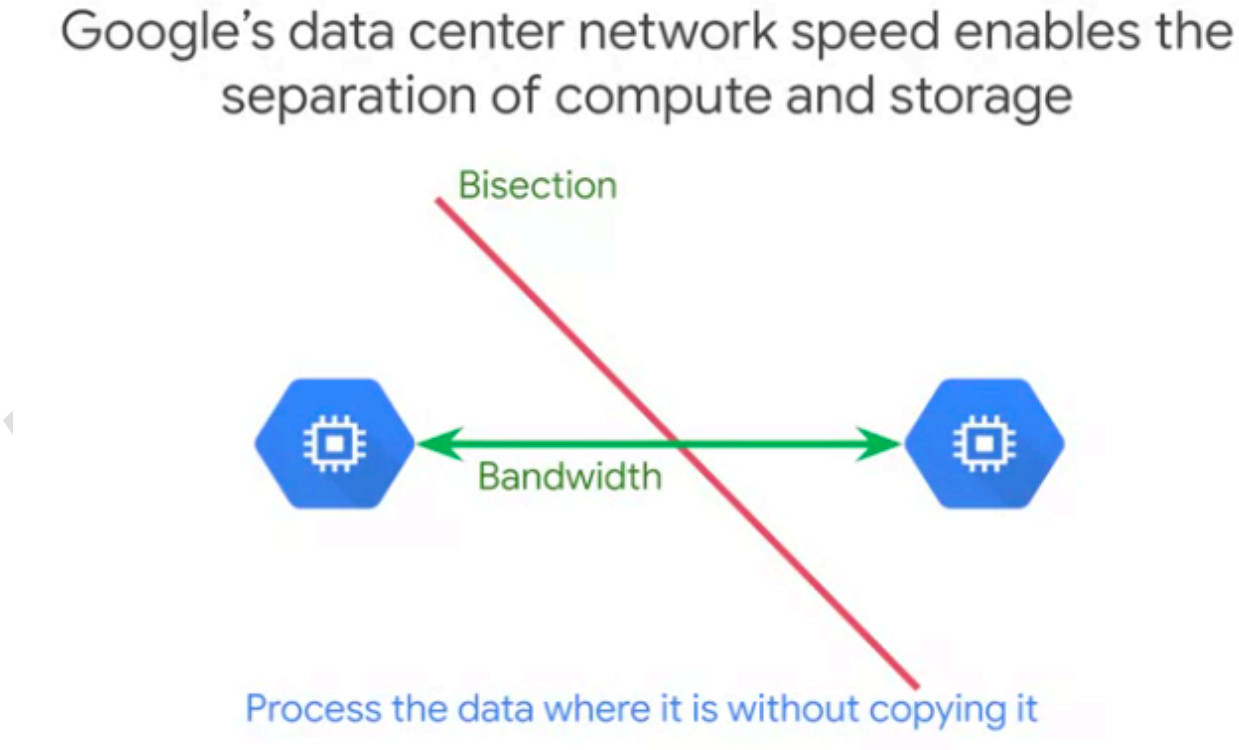
- Dataproc is a managed Spark and Hadoop service that lets you take advantage of open-source data tools for batch processing, querying, streaming, and machine learning. Dataproc automation helps you create clusters quickly, manage them easily, and save money by turning clusters off when you don't need them.
- Create clusters as needed
- Cloud dataproc autoscaling provides flexible capacity



- Autoscaling works as long as data is not stored in HDFS (i.e in the cluster)
    - Recommended not to store data in cluster.
    - Instead use external storage like bigtable, bigquery or cloud storage.
- Another advantage of dataproc is it works well with preemptible VMs significantly reducing costs. (upto 80% lower than regular VMs)



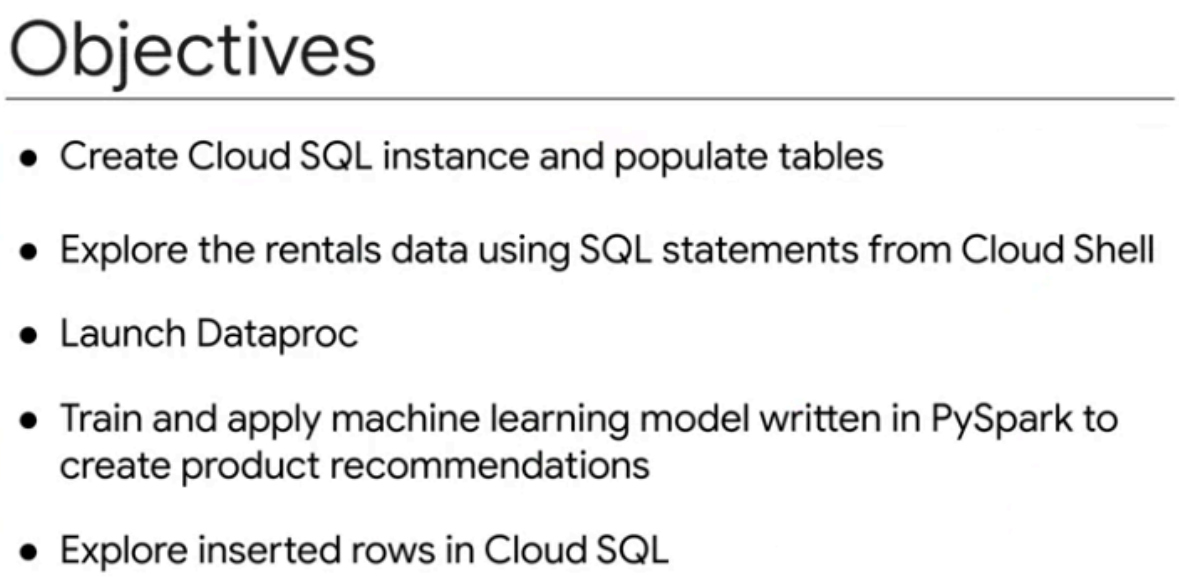
- **Off cluster data storage**
  - Google has petabyte network speed. So it can process data where it is instead of copying over to the cluster.



    - So, instead of storing data in HDFS, data can be stored in cloud storage or any other big data storage.
    - Use HDFS only for the data working data or data being currently processed.

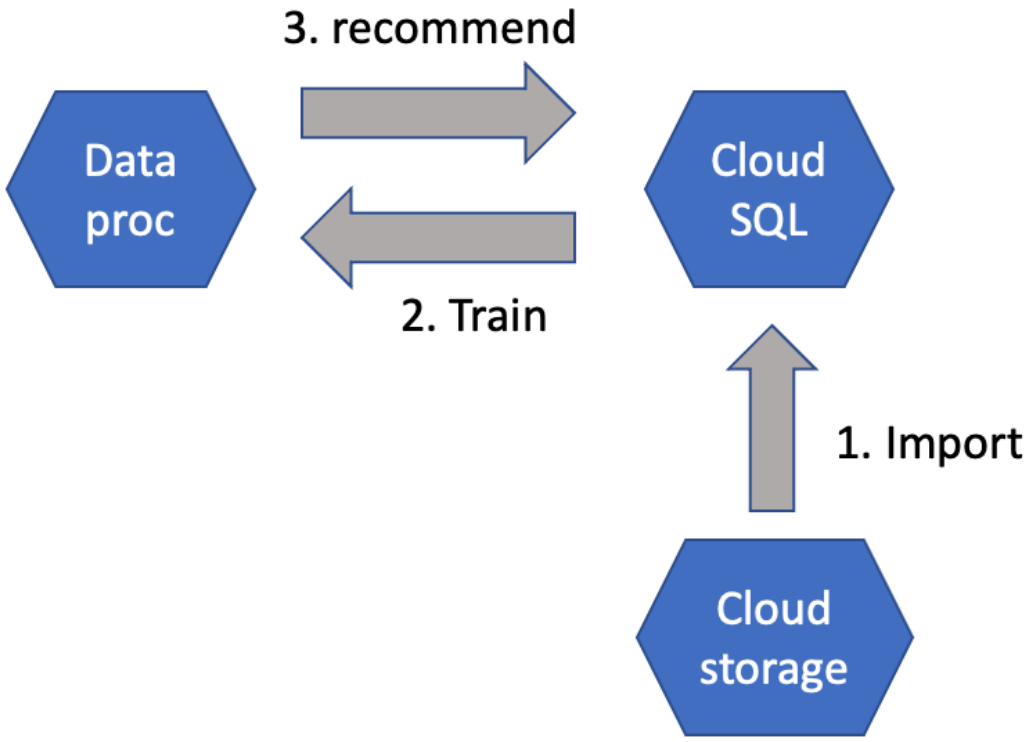
- **Dataproc summary**
    - Use hadoop without cluster management
    - Lift and shift existing hadoop workloads
    - Connect with cloud storage to separate compute and storage
    - Resize clusters effortlessly
    - Use preemptible VMs for cost savings

3.1 LAB: Recommending Products Using Cloud SQL and Sparks



Setup

1. Setup cloud storage and cloud SQL
2. Import data from cloud storage to cloud SQL
3. Run ML job in cloud dataproc that reads data from cloud storage and trains an ML model
4. Run the ML model in dataproc to create recommendations
5. Save top 5 recommendations per user back to cloud SQL

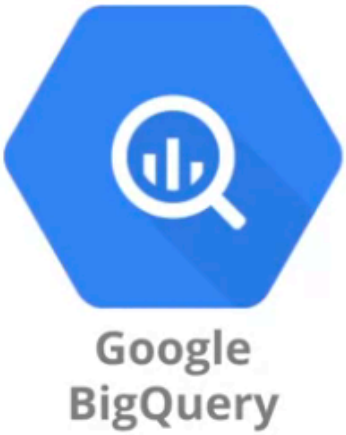


Steps

1. Assign "editor" role to project
2. Cloud SQL setup
    a. Create cloud SQL Instance
        i. user: root
        ii. pwd: password
    b. Connect to cloudSQL instance
        i. gcloud sql connect <instance name> --user=root --quiet
    c. Create database and tables
3. Copy data to cloud storage
    a. gsutil cp gs://cloud-training/bdml/v2.0/data/accommodation.csv gs://$DEVSHELL_PROJECT_ID
4. Load data from cloud storage to cloud SQL
    a. Use Console UI
5. Launch Data proc
    a. Create cluster (name, region, zone, master node, worker nodes)
    b. Authorize dataproc to connect with cloudSQL
        i. gcloud sql instances patch $CLOUDSQL --authorized-networks $ips
6. Run ML Model
    a. Create job pointing to ML python file
    b. Submit job
7. View recommendations in cloudSQL

4 BigQuery

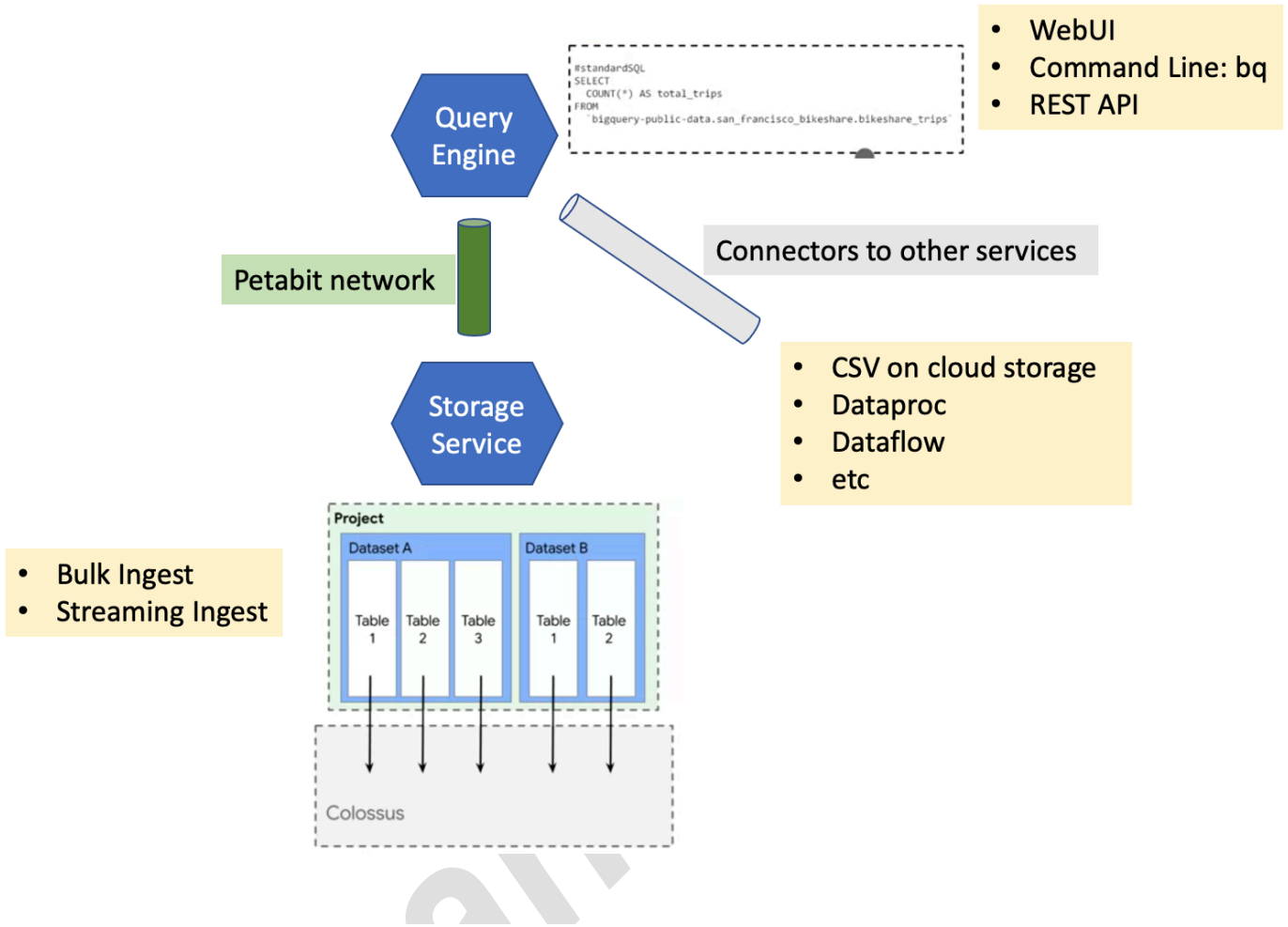
BigQuery is a petabyte-scale fully-managed data warehouse

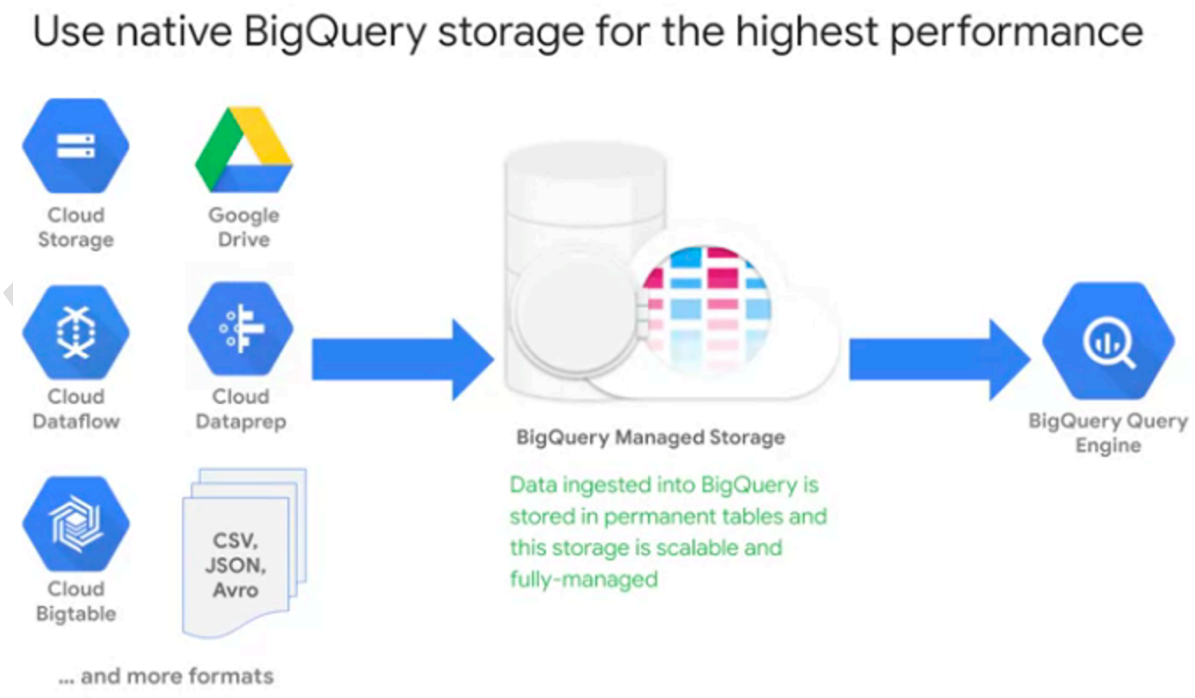


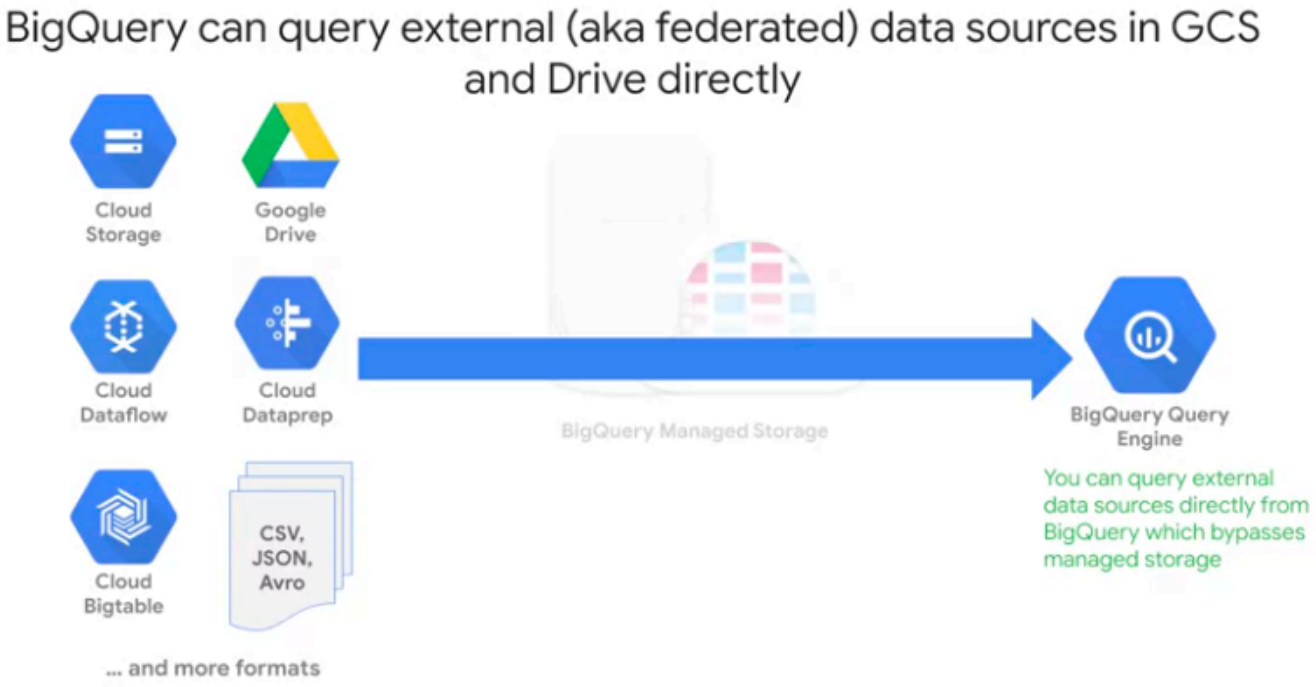
1. It's serverless
2. Flexible pricing model
3. Data encryption and security
4. Geospatial data types & functions
5. Foundation for BI and AI

BigQuery is 2 services in one:

1. **Query Engine**: Fast SQL query engine
2. **Storage service**: Managed storage for datasets


Query Engine
#standardSQL
SELECT
COUNT(*) AS total_trips
FROM
`bigquery-public-data.san_francisco_bikeshare.bikeshare_trips`
WebUI
Command Line: bq
REST API
Petabit network
Connectors to other services
Storage Service
CSV on cloud storage
Dataproc
Dataflow
etc
Bulk Ingest
Streaming Ingest
Project
Dataset A
Dataset B
Table 1
Table 2
Table 3
Table 1
Table 2
Colossus


Use native BigQuery storage for the highest performance
Cloud Storage
Google Drive
Cloud Dataflow
Cloud Dataprep
Cloud Bigtable
CSV, JSON, Avro
... and more formats
BigQuery Managed Storage
Data ingested into BigQuery is stored in permanent tables and this storage is scalable and fully-managed
BigQuery Query Engine



- Cloud **DataFlow** recommended for streaming data with transformations
    - Cost considerations
- Federated data sources not recommended
    - Consistency and reliability considerations

Use STRUCTs and ARRAYS for consolidated reporting

Job information | Results | JSON | Execution details

Row	order_id	service_type	payment_method	event.status	event.time	pickup.latitude	pickup.longitude	destination.latitude	destination.longitude	total_distance_km
1	CR-6098	GO_CAR	CASH	DRIVER_FOUND	2018-12-31 01:45:15.667 UTC	-6.9228116	107.5930599	-6.912966	107.609604	2.9749999046325684
				COMPLETED	2018-12-31 02:06:56.894 UTC					
				PICKED_UP	2018-12-31 02:05:40.075 UTC					
				CREATED	2018-12-31 01:44:59.239 UTC					

Schema | Details | Preview

Field name	Type	Mode
order_id	STRING	NULLABLE
service_type	STRING	NULLABLE
payment_method	STRING	NULLABLE
event	RECORD	REPEATED
event. **status**	STRING	NULLABLE
event. **time**	TIMESTAMP	NULLABLE
pickup	RECORD	NULLABLE
pickup. **latitude**	FLOAT	NULLABLE
pickup. **longitude**	FLOAT	NULLABLE

BigQuery natively supports ARRAYs as data types (and STRUCTs too)

Unlock your geographic insights with BigQuery GIS functions

```
#standardSQL
SELECT
  ST_GeogPoint(longitude, latitude) AS point,
  name,
  iso_time,
  dist2land,
  usa_wind,
  usa_pressure,
  usa_sshs,
  (usa_r34_ne + usa_r34_nw + usa_r34_se + usa_r34_sw)/4 AS radius_34kt,
  (usa_r50_ne + usa_r50_nw + usa_r50_se + usa_r50_sw)/4 AS radius_50kt
FROM
  `bigquery-public-data.noaa_hurricanes.hurricanes`
WHERE
  name LIKE '%MARIA%'
  AND season = '2017'
  AND ST_DWithin(ST_GeogFromText('POLYGON((-179 26, -179 48, -10 48, -10 26,
-100 -10.1, -179 26))'),
    ST_GeogPoint(longitude, latitude), 10)
ORDER BY
  iso_time ASC
```

Visualize using GeoViz

4.1 Data security in BigQuery

When a project is created, BigQuery grants the `Owner` role to the user who created the project.

Primitive role	Capabilities
`Viewer`	• Can start a job in the project. Additional dataset roles are required depending on the job type. • Can list and get all jobs, and update jobs that they started for the project • If you create a dataset in a project that contains any viewers, BigQuery grants those users the bigquery.dataViewer predefined role for the new dataset.

`Editor`	• Same as **`Viewer`**, plus:
	  • Can create a new dataset in the project
	  • If you create a dataset in a project that contains any editors, BigQuery grants those users the bigquery.dataEditor predefined role for the new dataset.

`Owner`	• Same as **`Editor`**, plus:
	  • Can list all datasets in the project
	  • Can delete any dataset in the project
	  • Can list and get all jobs run on the project, including jobs run by other project users
	  • If you create a dataset, BigQuery grants all project owners the bigquery.dataOwner predefined role for the new dataset.

- Dataset users should have the minimum permissions needed for their role
- Use separate projects or datasets for different environments (e.g. DEV, QA, PRD)
- Audit roles periodically

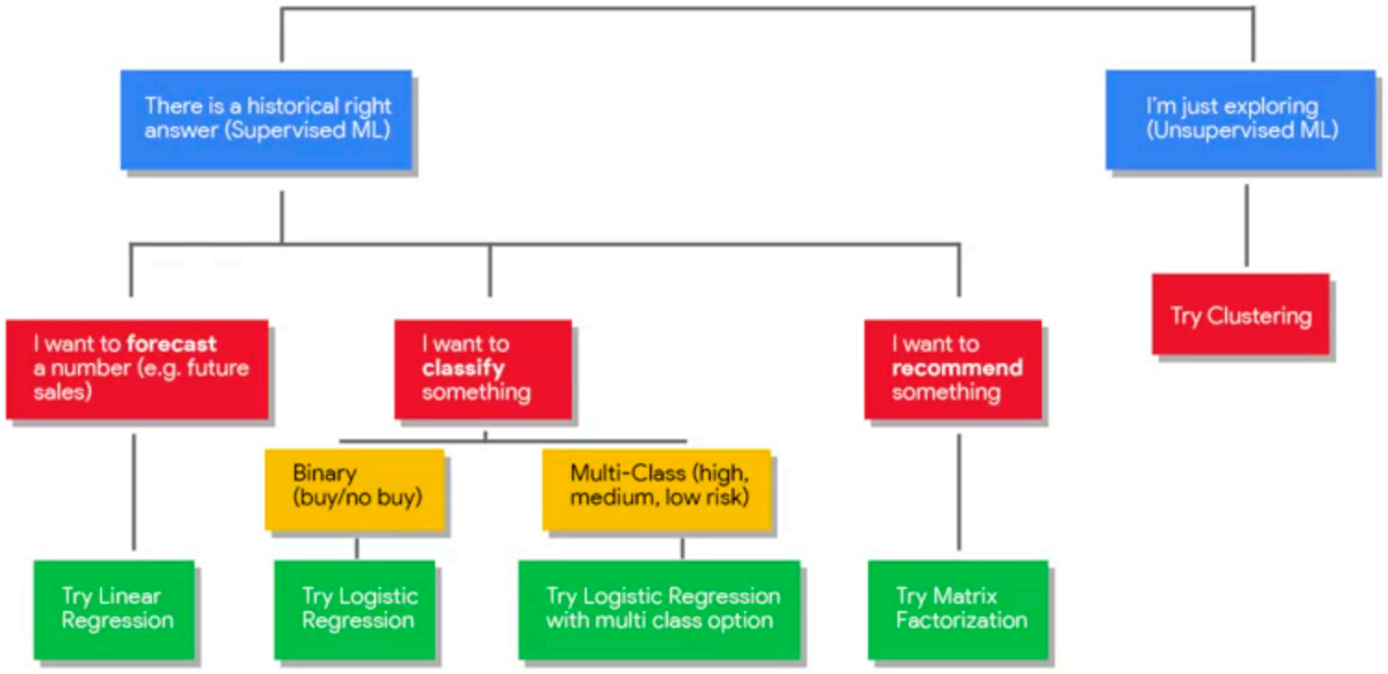
4.2 BigQuery ML

The most common ML models at Google are those that operate on structured data

Source (2017): https://cloud.google.com/blog/big-data/2017/05/an-in-depth-look-at-googles-first-tensor-processing-unit-tpu

Type of network	# of network layers	# of weights	% of deployed models
MLP0	5	20M	61%
MLP1	4	5M	
LSTM0	58	52M	29%
LSTM1	56	34M	
CNN0	16	8M	5%
CNN1	89	100M	

Choose the right model type for your structured data use case



The End-to-End BQML Process

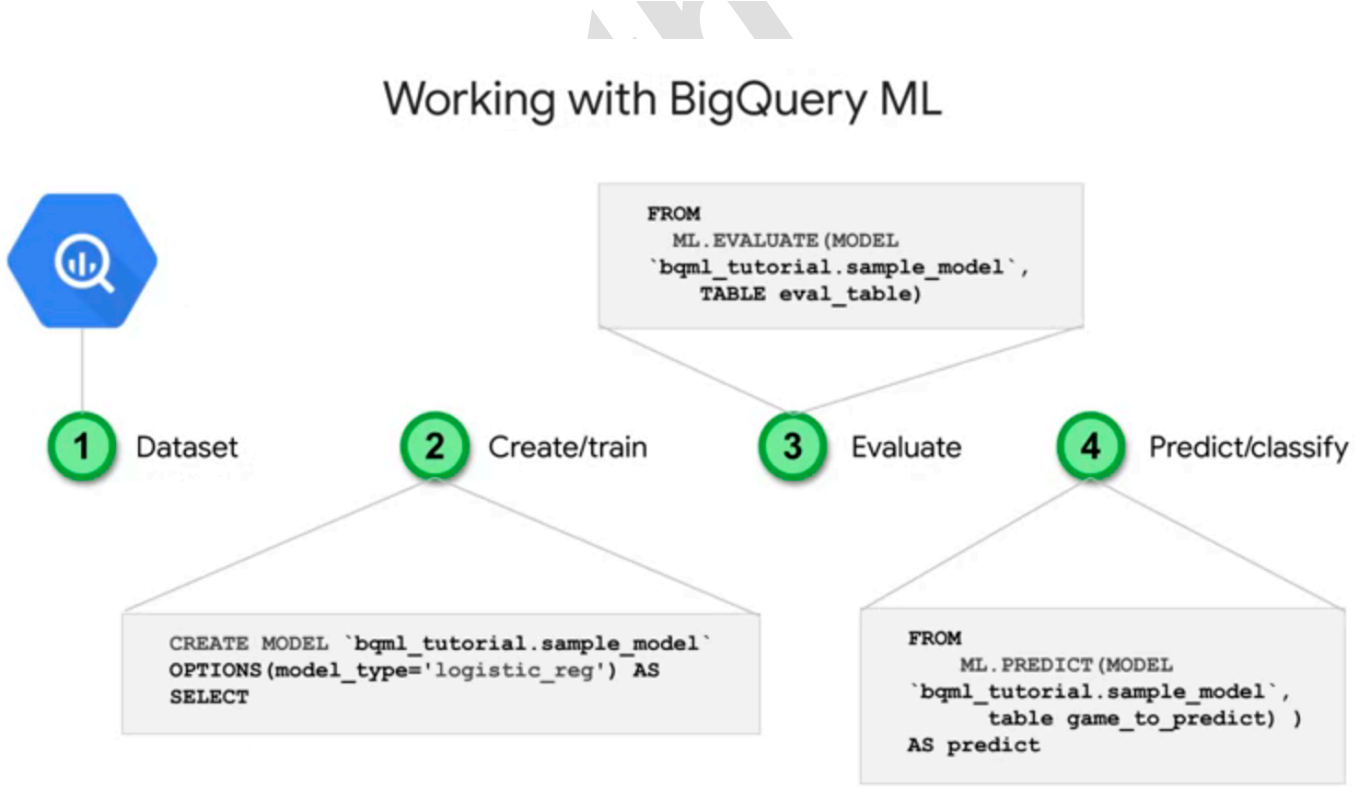
- Create a model

```
CREATE OR REPLACE MODEL
  `mydataset.mymodel`
OPTIONS
  ( model_type='linear_reg',
  input_label_cols= 'sales'
  ls_init_learn_rate=.15,
  l1_reg=1,
  max_iterations=5 ) AS
```

- **Label** = alias a column as 'label' or specify column in OPTIONS using input_label_cols
- **Feature** = passed through to the model as part of your SQL SELECT statement
  `SELECT * FROM ML.FEATURE_INFO(MODEL `mydataset.mymodel`)`
- **Model** = an object created in BigQuery that resides in your BigQuery dataset
- **Model Types** = Linear Regression, Logistic Regression

```
CREATE OR REPLACE MODEL <dataset>.<name>
OPTIONS(model_type='<type>') AS
<training dataset>
```

- **Training Progress** = `SELECT * FROM ML.TRAINING_INFO(MODEL `mydataset.mymodel`)`
- **Inspect Weights** = `SELECT * FROM ML.WEIGHTS(MODEL `mydataset.mymodel`, (<query>))`
- **Evaluation** = `SELECT * FROM ML.EVALUATE(MODEL `mydataset.mymodel`)`
- **Prediction** = `SELECT * FROM ML.PREDICT(MODEL `mydataset.mymodel`, (<query>))`



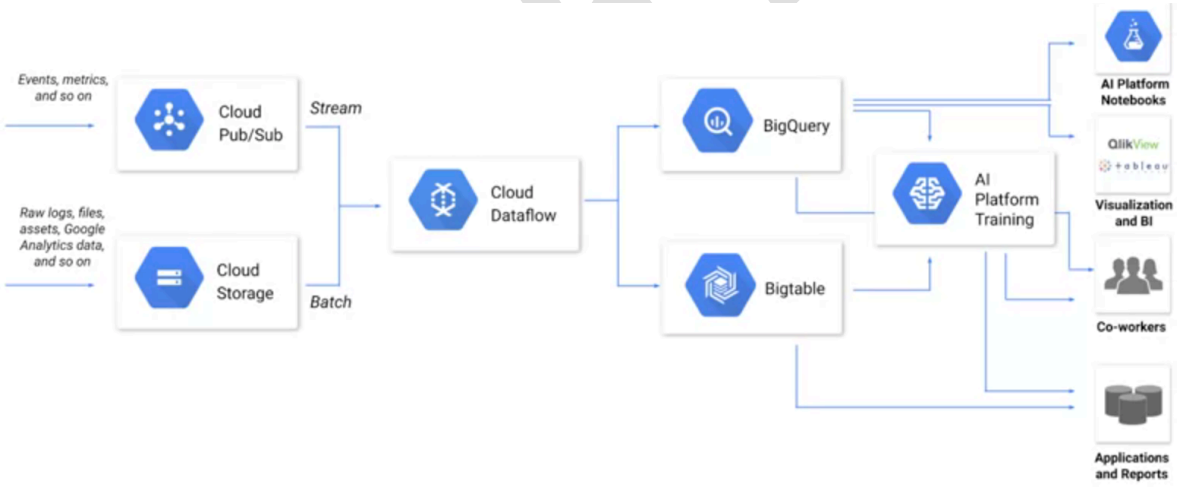
5 Cloud Dataflow & Apache Beam

5.1 Training and serving skew

- Unless a model sees the exact same data in serving (prediction) as it was used to seeing during training, the model predictions are going to be off. This is referred to as ***training serving skew***.
    - For example, training could be on batch data and serving(predictions) could be on stream data.
    - To prevent training serving skew, your data pipelines have to process both batch and Stream.

- This is a key insight behind **dataflow**, a way to author data pipelines in Python, Java or even visually with cloud data prep.

- It's opensource version is ***Apache BEAM***, where B stands for batch and the EAM stands for stream. So, a single system to do both batch and stream.



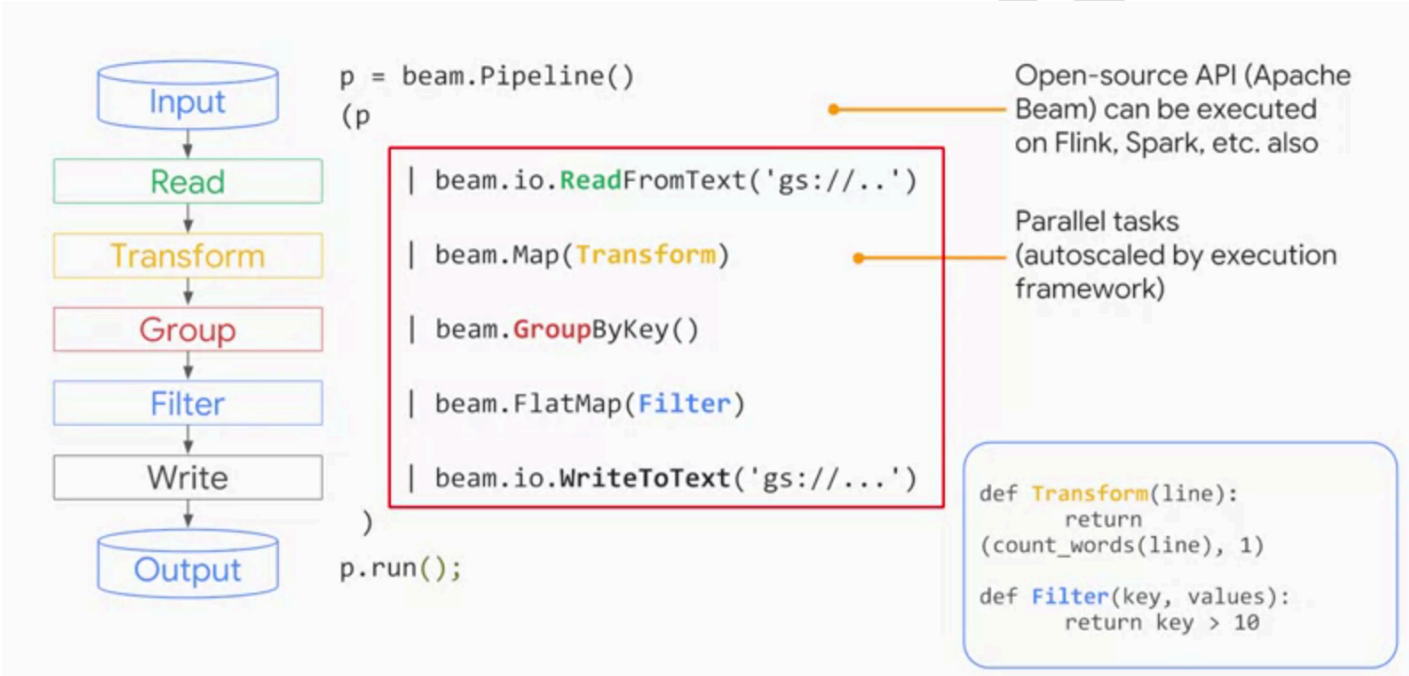
- One of the ways to think about feature pre-processing, or any data transformation, is to think in terms of pipelines. A pipeline is a sequence of steps that change data from one format into another.

- ***Cloud Dataflow*** is a platform that allows you to run these kinds of data processing pipelines.
    - Dataflow can run pipelines written in Python and Java programming languages.
    - Dataflow sets itself apart as a platform for data transformations because it is a serverless, fully managed offering from Google that allows you to execute data processing pipelines at scale.

- Dataflow can change the amount of compute resources, the number of servers that will run your pipeline, and do that elastically depending on the amount of data that your pipeline needs to process.

- Code for Dataflow is written using an opensource library called ***Apache Beam***.
  - To implement a data processing pipeline, you write your code using the Apache Beam APIs, and then deploy the code to Cloud Dataflow.

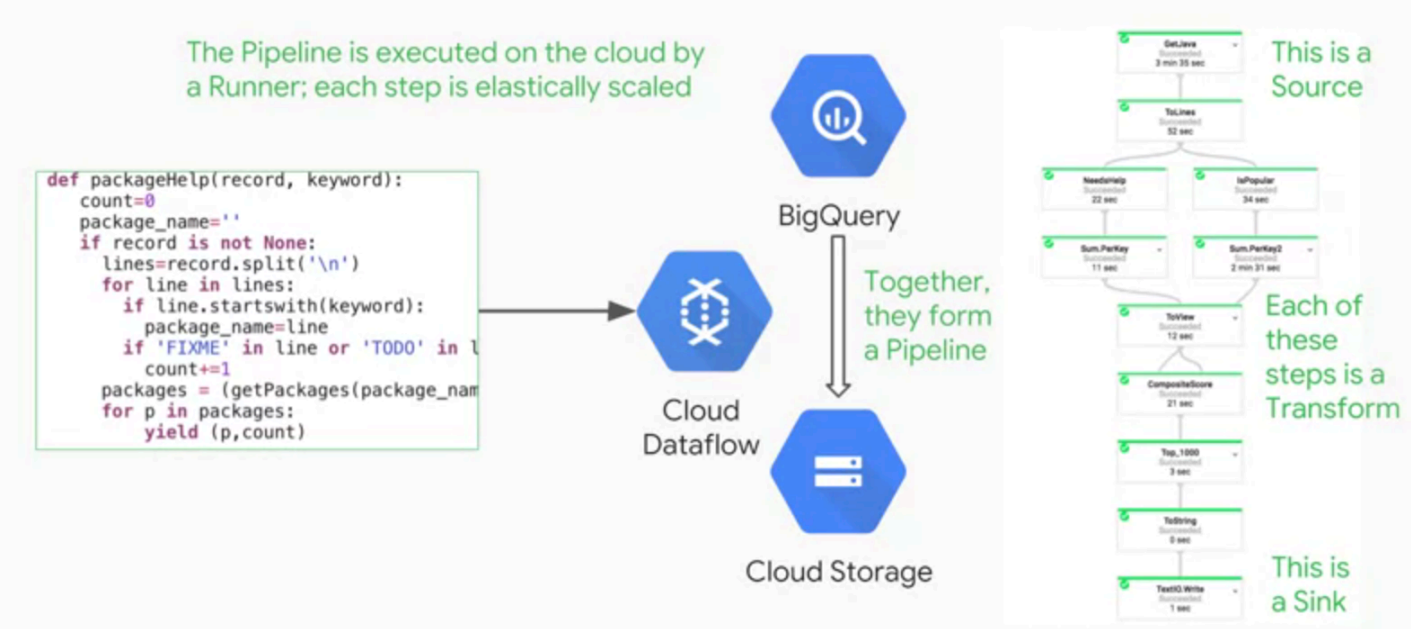
- Example Pipeline



  - The above is an example in python to count the number of words.
  - The pipeline definition (p) is separated from the pipeline execution (p.run()) in the above implementation.
  - Regardless of whether your data is coming from a batch data source, like Google Cloud Storage, or even from a streaming data source, like Pub/Sub, you can reuse the same pipeline logic.

- You can also output data to both batch and streaming data destinations.

5.2 Apache Beam terminology



- The input to the pipeline is called a ***source***.
- Each step in the pipeline is called a ***Transform***.
- Each transform works on a data structure called the ***PCollection***.
  - Every transform gets a PCollection as the input and outputs a PCollection.
- The result of the last transform in a pipeline goes to a ***sink*** which is the output of the pipeline.
- To run a pipeline, you need a ***runner***.
  - A runner takes pipeline code and executes it.
  - Runners are platform specific.
    - You need a specific runner to execute on Cloud dataflow.
    - There is another runner to execute on Apache Spark.
    - There is direct runner to execute pipelines on local computer.
    - Custom runners can be created for custom platforms too.

5.3 Implementing a Pipeline

- ***beam.pipeline*** creates a pipeline instance.
- Once it is created, every transform is implemented as an argument to the ***apply*** method of the pipeline.
- In the Python version of the Apache Beam library, the ***pipe operator*** is overloaded to call the apply method.

```
import apache_beam as beam
if __name__ == '__main__':

   # create a pipeline parameterized by commandline flags
   p = beam.Pipeline(argv=sys.argv)

   (p
      | 'Read' >> beam.io.ReadFromText('gs://...') # read input
      | 'CountWords' >> beam.FlatMap(lambda line: count_words(line))
      | 'Write' >> beam.io.WriteToText('gs://...') # write output
   )

   p.run() # run the pipeline
```

- The strings, like read, countwords, and write are just the human readable names that you can specify for each transform in the pipeline.
- When you need your pipeline to process some data, you need to call the run method on the pipeline instance to execute it.
- Every time you use the pipe operator, you provide a PCollection data structure as input and return a PCollection as output.
  - A PCollection does not store all of its data in memory.
  - A PCollection is like a data structure with pointers to where the data flow cluster stores your data.
  - That's how Dataflow can provide ***elastic scaling of the pipeline***

- Apache Beam SDK comes with a variety of connectors that enable Dataflow to read from many data sources.
  - It's possible to read even from real time streaming data sources like Google Cloud Pub/Sub, or Kafka.
  - When using the BigQuery connector, you need to specify the SQL statement that BigQuery will evaluate to return back a table with rows of

results. The table rows are then passed to the pipeline in a PCollection to export out the result of a pipeline.

BigQuery returns a TableRow

```
rows = beam.io.Read(beam.io.BigQuerySource(query='SELECT x, y, z ' \
                'FROM [project:dataset.tablename]', project='PROJECT'))
```

- When writing to a file system, remember that data flow can distribute execution of your pipeline across a cluster of servers.
    - This means that there can be multiple servers trying to write results to the file system.
    - To avoid contention issues where multiple servers are trying to get a file lock to the same file concurrently, by default, the text I/O connector will shard the output, writing the results across multiple files in the file system.

Can prevent sharding of output (do only if it is small)

```
beam.io.WriteToText(file_path_prefix='/data/output',
file_name_suffix='.txt', num_shards = 1)
```

The output must be a PCollection of Strings before writing out

- To execute a pipeline in python, running main() is sufficient

Simply running main() runs pipeline locally

```
python ./grep.py
```

- To submit the pipeline ***as a job*** to execute in Dataflow on GCP, you need to provide some additional information.

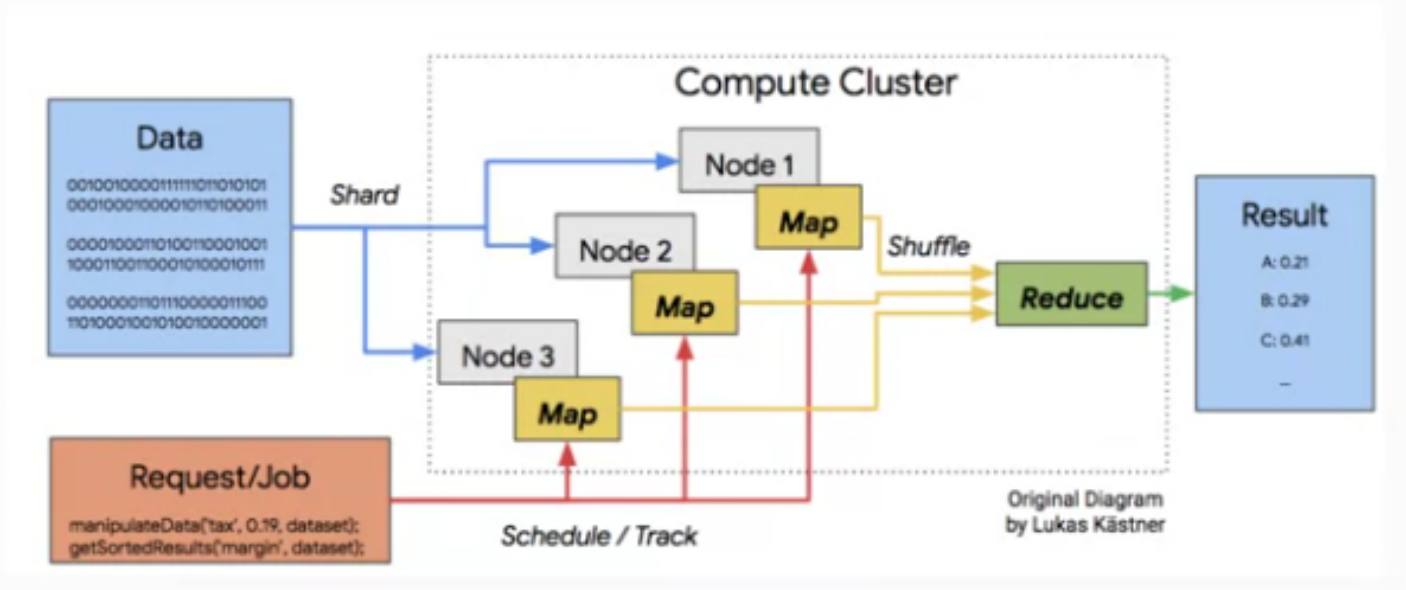
To run on cloud, specify cloud parameters, and submit the job to Dataflow

```
python ./grep.py \
      --project=$PROJECT \
      --job_name=myjob \
      --staging_location=gs://$BUCKET/staging/ \
      --temp_location=gs://$BUCKET/staging/ \
      --runner=DataflowRunner
```

- Include arguments with the name of the GCP project, location in Google Cloud Storage Bucket where Dataflow will keep some staging and temporary data.
- You also need to specify the name of the runner, which in this case is the DataFlowRunner.

5.4 Pipelines that scale

5.4.1 Using map reduce



- Map-Reduce is a distributed fault tolerant data processing framework that was described by Google in an influential research paper published in 2004.
- To run a data processing job in mapReduce, you write code for Map and Reduce functions.
- A map should be a stateless function, so that it can be scheduled to run in parallel across the nodes in the cluster.
- Each Map reads the data from the storage on the node where this running, processes the data and generates an output.
- The output of the map operations are shuffled from the different nodes in the cluster to the next stage of processing called Reduce. You can think of reductions as an aggregation operation over data.

5.4.2 Using ParDo (Parallel Do)

Use the Apache beams ***ParDo*** class if you want to take a transformation in your data processing pipeline and let data flow run it at scale with automatic distribution across many nodes in a cluster.

- ParDo is short for parallel do.
- The transformation steps created using ParDo are similar to the maps in MapReduce.
- They have to be stateless so they can be run in parallel.
- In Python, there are helper methods that run ParDo

5.4.2.1Map Function

- **Python Map**
  - Used for 1:1 relation between input and output.
  - For example, if we need length of each word, one word can have 1 length and so we use Map
  - Dataflow will automatically handle running this transformation in multiple nodes on the cluster.

```
'WordLengths' >> beam.Map( lambda word: (word, len(word)) )
```

- **Python FlatMap**
  - beam.FlatMap supports transformations that can generate any number of outputs for an input including zero outputs.
  - For example, for every word you would like to output the list of vowels for that word. You can have zero, one or more vowels per word.
  - The transformations in beam.FlatMap can also be run in parallel by dataflow

```
def vowels(word):
  for ch in word:
    if ch in ['a','e','i','o','u']:
      yield ch

'WordVowels' >> beam.FlatMap( lambda word: vowels(word) )
```

5.4.2.2 Shuffle Function

- In Dataflow, shuffle explicitly using GroupByKey()
- For example:
  - You have a pipeline that processes postal addresses and tries to find all the zip codes for every city.
  - In the map step, your pipeline has a P collection of key value pairs with each pair containing the city as the key and the zip code as the value.
  - The output created by beam.GroupByKey() will produce a P collection of pairs, where every pair has the city as a key and the list of the city's zip codes as the value.

```
cityAndZipcodes = p

     | beam.Map(lambda address: (address[1], address[3]) )
                                CITY        ZIPCODE
   | beam.GroupByKey()
```

5.4.2.3 Reduce Function

- While groupByKey() is similar to the shuffle step in MapReduce, the ***combine*** operation is more general and includes both shuffle and reduce steps to help you implement aggregations like sum, count.

- **Combine.globally()**
  - You can use combine.globally() method to compute over your entire dataset.
  - For example:
    - If you're processing financial transaction data, so that every row in your collection is a transactions of sales amounts, then to compute the total sales over all transactions, you can use the combine.globally() with the ***sum operation*** as the argument.

```
totalAmount = salesAmounts | Combine.globally(sum)
```

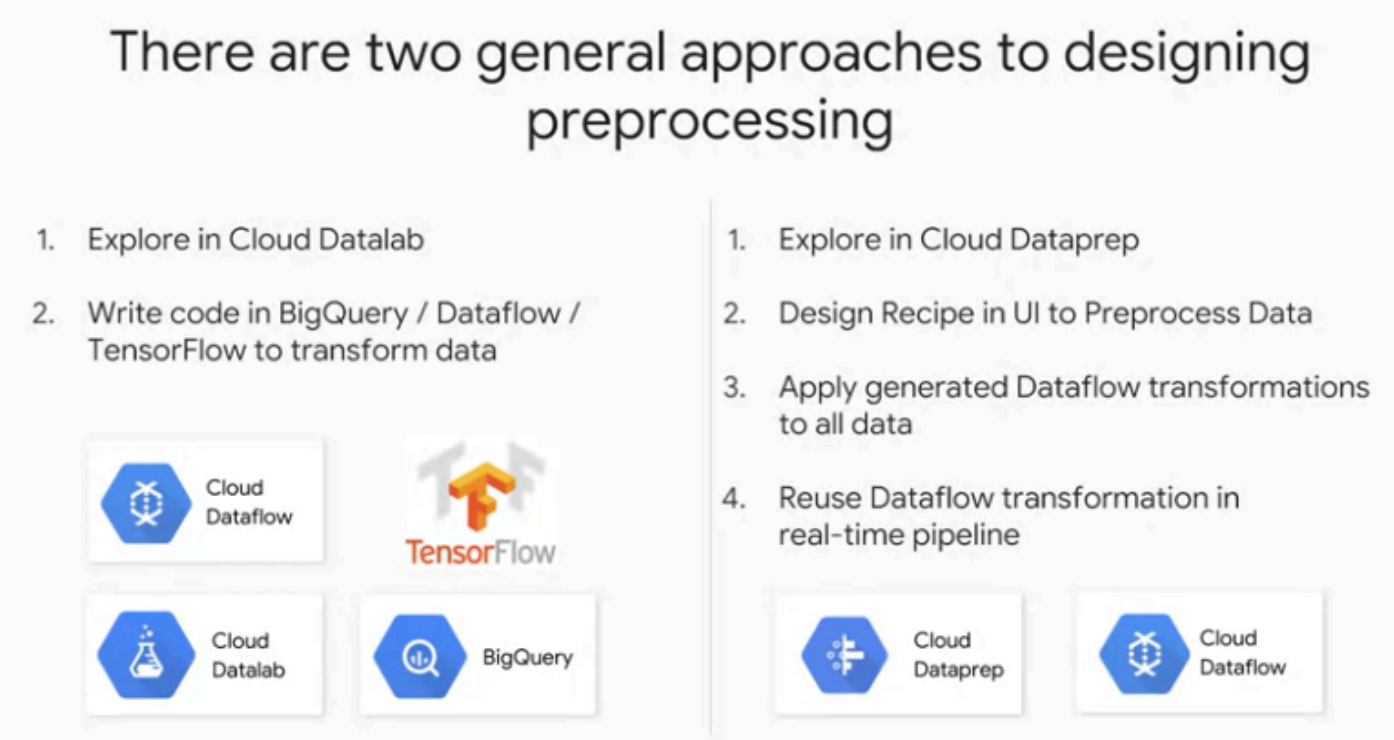
- **Combine.perKey()**
  - Combine, also supports more fine-grained aggregations.
  - For example
    - If your financial transaction records include the name of the salesperson in addition to the sales amount, you can pass the ***sum*** operation to the combine.PerKey() and use it to combine the total sales per salesperson.

```
totalSalesPerPerson = salesRecords | Combine.perKey(sum)
```

- Many built-in aggregate functions like ***sum, mean, count*** etc. can be used with combine.

6 Cloud Dataprep

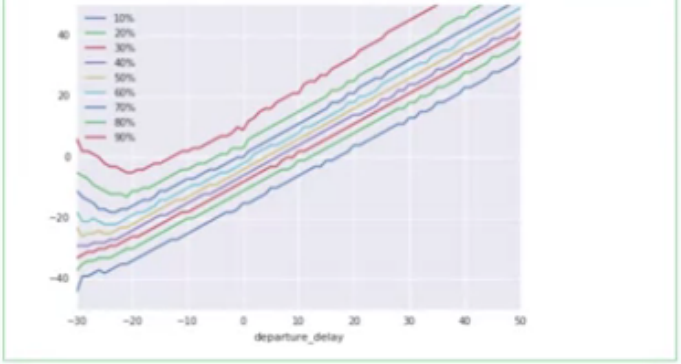
6.1 Data Preprocessing



- **Approach 1**: use tools like BigQuery, Cloud Dataflow and Tensorflow.
    - You can use a notebook in Datalab to explore and visualize your data set
    - However, it will be impractical or too expensive to plot and analyze millions\billions of records using just a single node datalab environment.
    - Alternatively, you can use SQL and calculate summary statistics using BigQuery.

```
query="""
SELECT
  departure_delay,
  COUNT(1) AS num_flights,
  APPROX_QUANTILES(arrival_delay, 10) AS arrival_delay_deciles
FROM
  `bigquery-samples.airline_ontime_data.flights`
GROUP BY
  departure_delay
HAVING
  num_flights > 100
ORDER BY
  departure_delay ASC
"""

import google.datalab.bigquery as bq
df = bq.Query(query).execute().result().to_dataframe()
df.head()
```

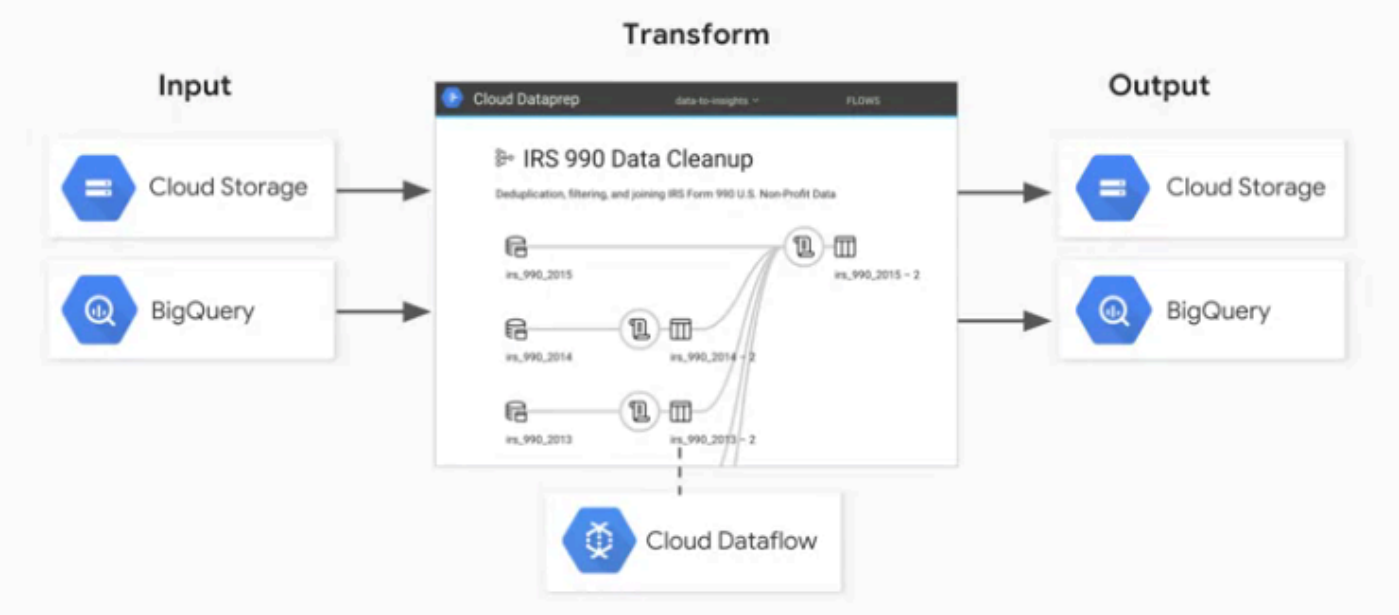
```
without_extremes = df.drop(['0%', '100%'], 1)
without_extremes.plot(x='departure_delay', xlim=(-30,50), ylim=(-50,50));
```



- As shown in the above diagram,you can still use datalab to write you SQL code,once the code is ready,you submit the SQL statement to BigQuery via the APIs and get back the result.
- Since the summary statistics are just a few rows of data,you can easily plot them in datalab using Seaborne,or other Python visualization libraries.
- We can also use cloud DataFlow pipelines to calculate summary statistics and run other data preprocessing jobs.
- We can also use java\python \tensorflow to write data processing pipelines.

- **Approach 2**: Use cloud DataPrep.
  - You can do feature engineering using an interactive visual interface, instead of writing low level code
  - A tool that lets you use an interactive, graphical user interface to better understand, visualize and preprocess your data.
  - Provides a UI to view and transform datasets.
  - Schedule pipeline as needed
  - It can help with both exploratory analysis and data processing.

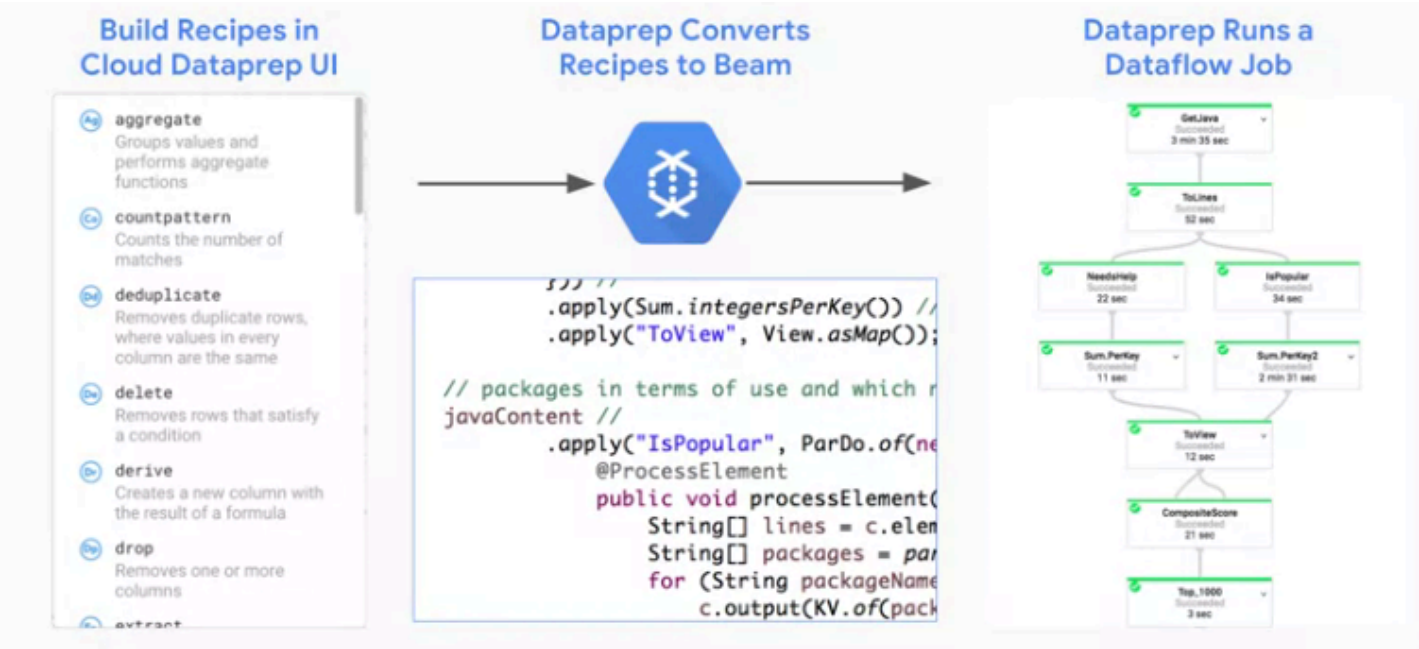
6.2 Overview



- DataPrep is a fully managed service available from GCP, and it lets you explore and transform your data interactively using a web browser with a minimal amount of code,
- Dataprep can get data from a variety of sources including Google Cloud storage, and BigQuery.
- You can also upload your own data to Dataprep.
- Once Dataprep knows where to get your data, you can use this graphical UI to explore your data, and create data visualizations.

6.2.1 Data Transformations

- You can use Dataprep to compute flows of data transformations.
- The flows are similar to the pipelines that you have seen in dataflow.
- In fact, the flows are compatible with dataflow.
- You can take a Dataprep flow, and run it as a pipeline on the data flow platform.
- In Dataprep, the flows are implemented as a sequence of **recipes**.
- The recipes are data processing steps built from a library of so called ***wranglers***.

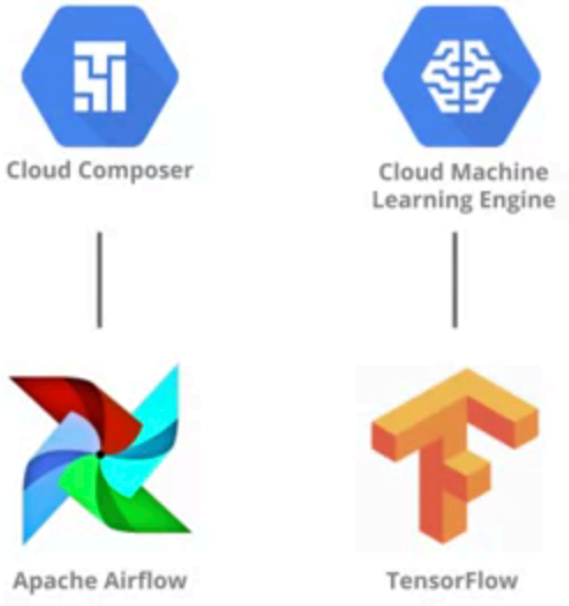


- Dataprep has Wranglers for many common data processing tasks (like aggregate etc..)
  - Use wranglers instead of writing your own code.
  - You can clean up data using de-duplication or filter out missing an outlier values,
  - or you can do common aggregations like counting or summing up values,
  - or you can join a union different data tables together,
  - and you can transform data into different types like strings or integers.
- Dataprep can take your flow and its recipes and convert them to a dataflow pipeline.
- Then, using the same Dataprep interface, you can take the flow, run it as a job on Dataflow and monitor the progress of the job.
- While the flow is executing, you can use the Dataflow interface to monitor the details of the jobs progress, and once the job is done, you can get a summary of the job status in Dataprep.

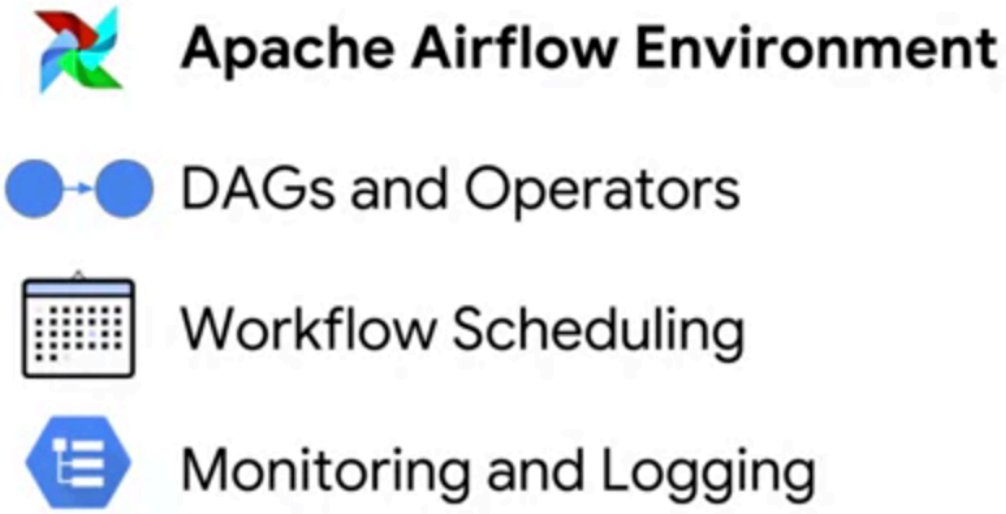
7 Cloud Composer & Apache Airflow

7.1 Overview

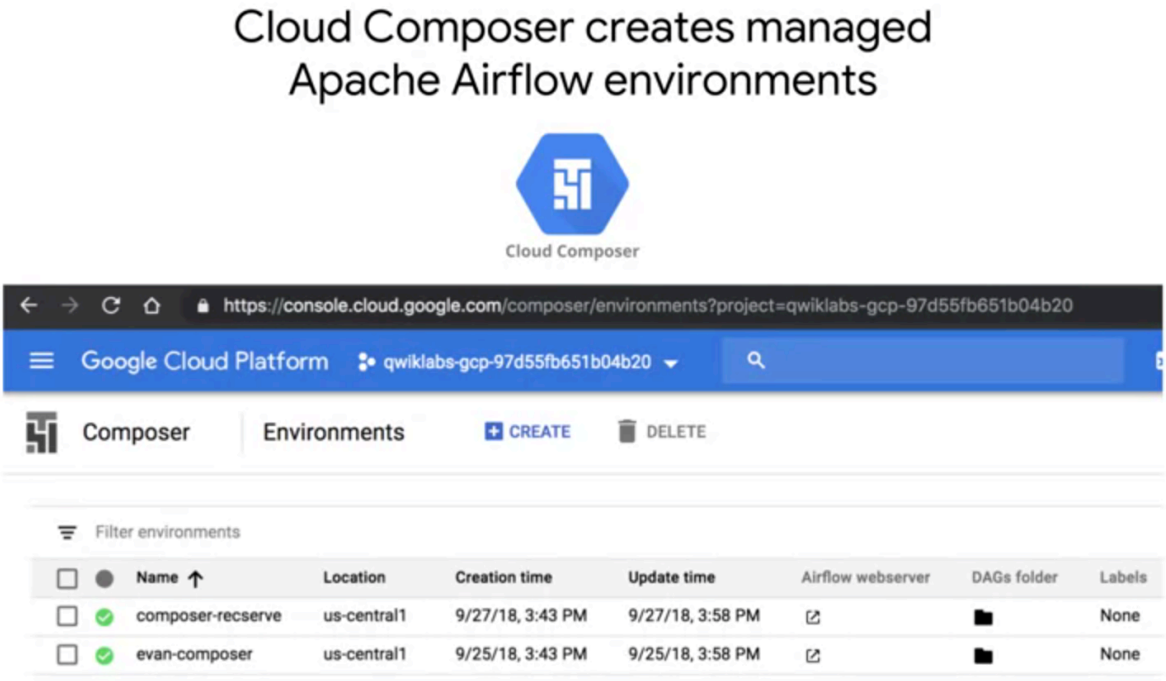
- Cloud Composer is an orchestration service that commands the GCP services we need to run.
    - An orchestrator is responsible for gluing the system components together.
    - They are used in ML pipelines to ensure consistency with execution order, component logging, retries and failure recovery, and intelligent parallelization of component data processing.
    - Some common orchestrators include:
        - ***Apache Airflow***:
            - Used by cloud composer
            - Used to **author** workflows as directed acyclic graphs (DAGs) of tasks.
        - ***Apache Beam***
            - Used for distributed data processing.
            - Also used to orchestrate and execute a pipeline DAG.
- Cloud Composer is a serverless environment on which Airflow runs.
- Airflow works as follows:
    - Write up an ordered set of tasks as part of what's called a DAG. (in python)
        - A DAG, or directed acyclic graph, is a set of tasks and special operators that allow you to programmatically and periodically schedule tasks to happen.
        - Directed means that there's a specified dependency flow.
        - Acyclic means it can't be a feedback loop into itself like a circle. i.e non circular.
    - Airflow executes the tasks
- Cloud Composer is to Apache Airflow as Cloud Machine Learning Engine(now AI Platform or Vertex AI) is to TensorFlow.



- Building any workflow in Cloud Composer consists of these four steps

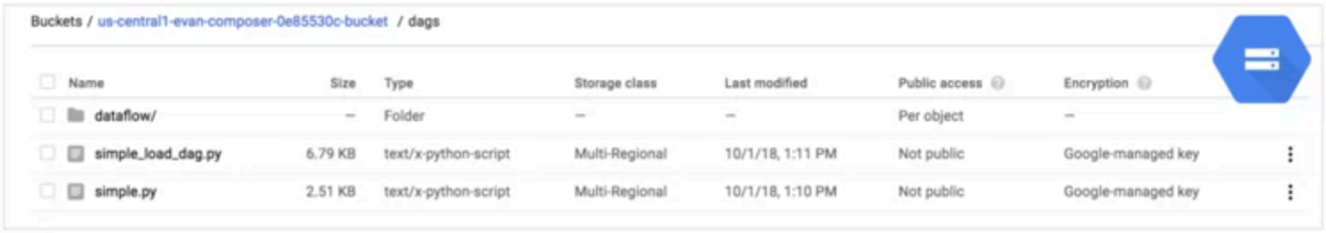


7.2 Airflow: Environment

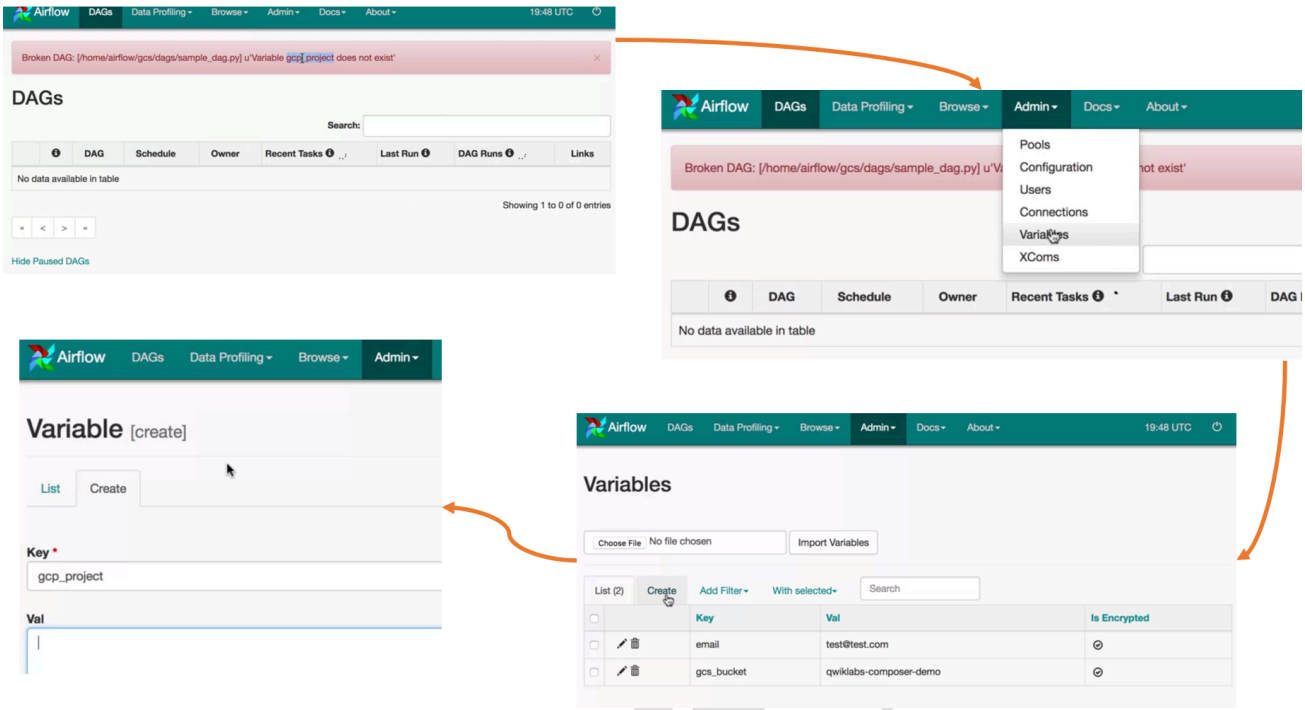


- You can use the command line or GCP web UI to launch a Cloud Composer instance
  - You can have multiple Cloud Composer environments and within each environment, you can have a separate Apache Airflow instance that can have zero to many DAGs inside of it.
- Generally, use the Cloud Composer page only to create new environments
  - Sometimes you'll be required to edit environmental variables for your workflows, like specifying your specific GCP project account.
  - Do not do that at the cloud composer level, but instead, do it at the actual Apache Airflow level
- To access the Airflow admin UI where you can monitor and interact with your workflows, you'll click on the link underneath the Airflow webserver.

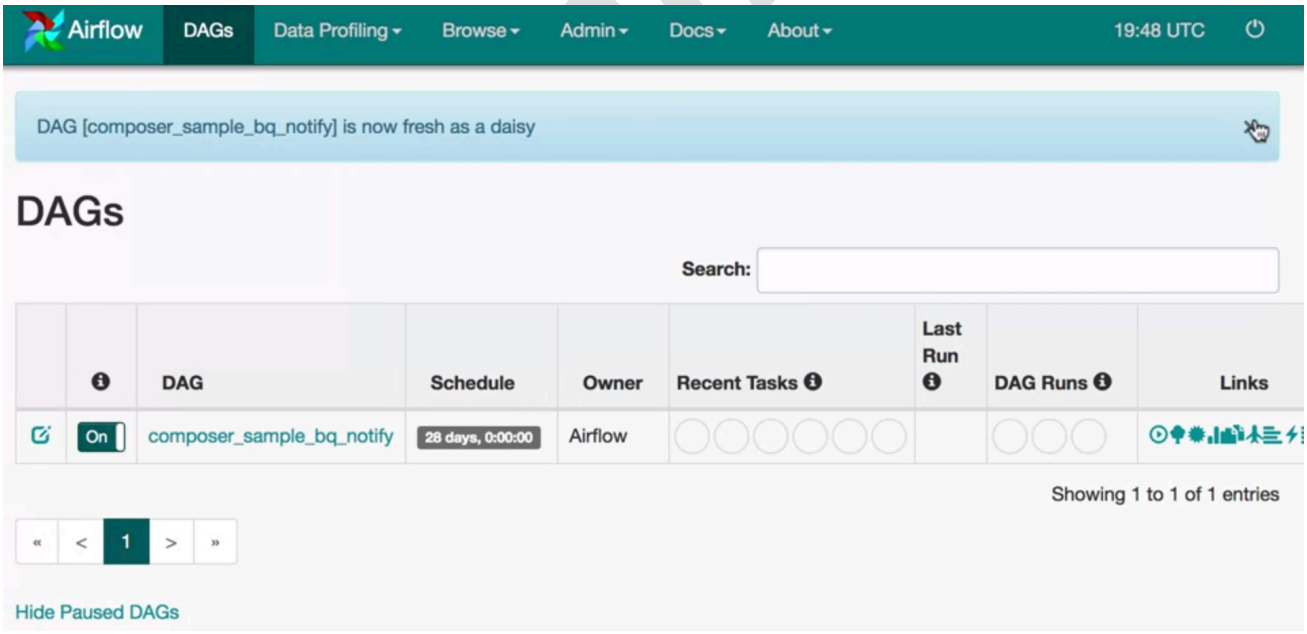
- ***Uploading DAGs***



  - The DAGs folder is where the code of your actual workflows will be stored.
  - The DAGs folder for each airflow instance is simply a GCS bucket that's automatically created for you when you launch your cloud composer instance.
  - To go here, click the link under the DAGs folder on cloud composer web UI
  - DAGs are written in python
  - You could have multiple workflows or multiple DAGs in a single airflow environment.
    - You only have one Python file for each DAG.
  - Once uploaded, you can view the DAG in graph\tree\code form using the airflow admin UI
    - First define any missing environment variables



- View the DAG on Airflow admin UI

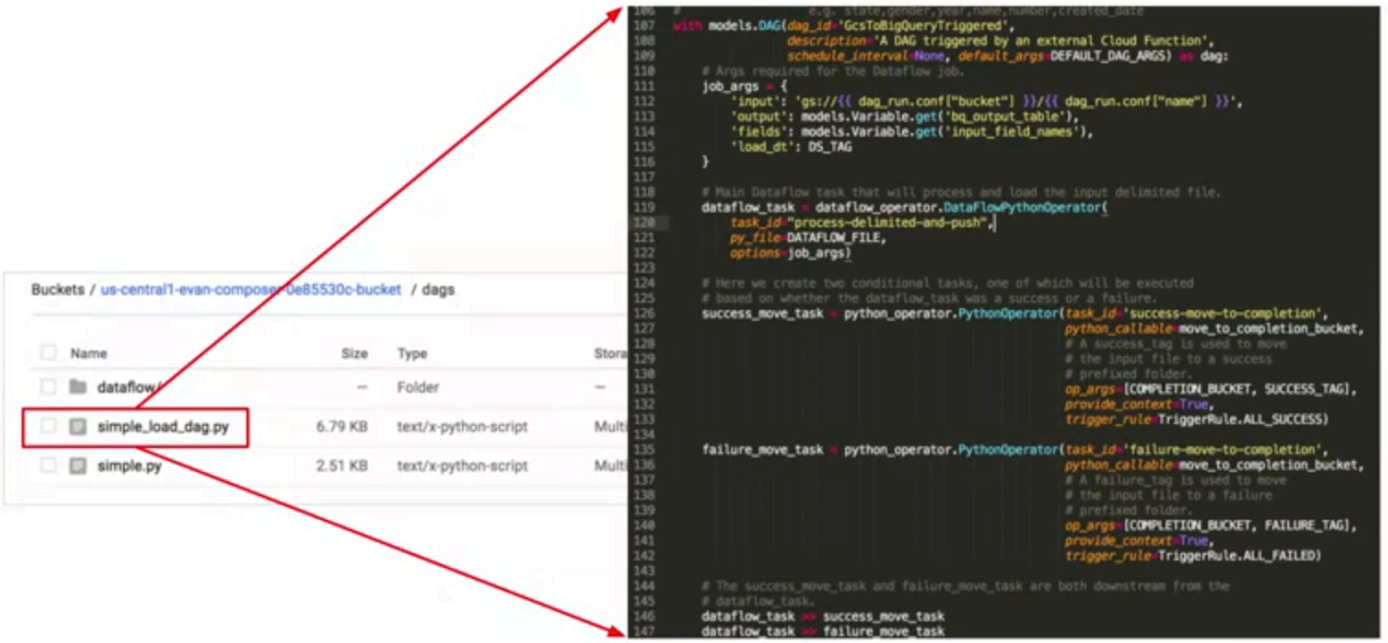


- View DAG in various formats.

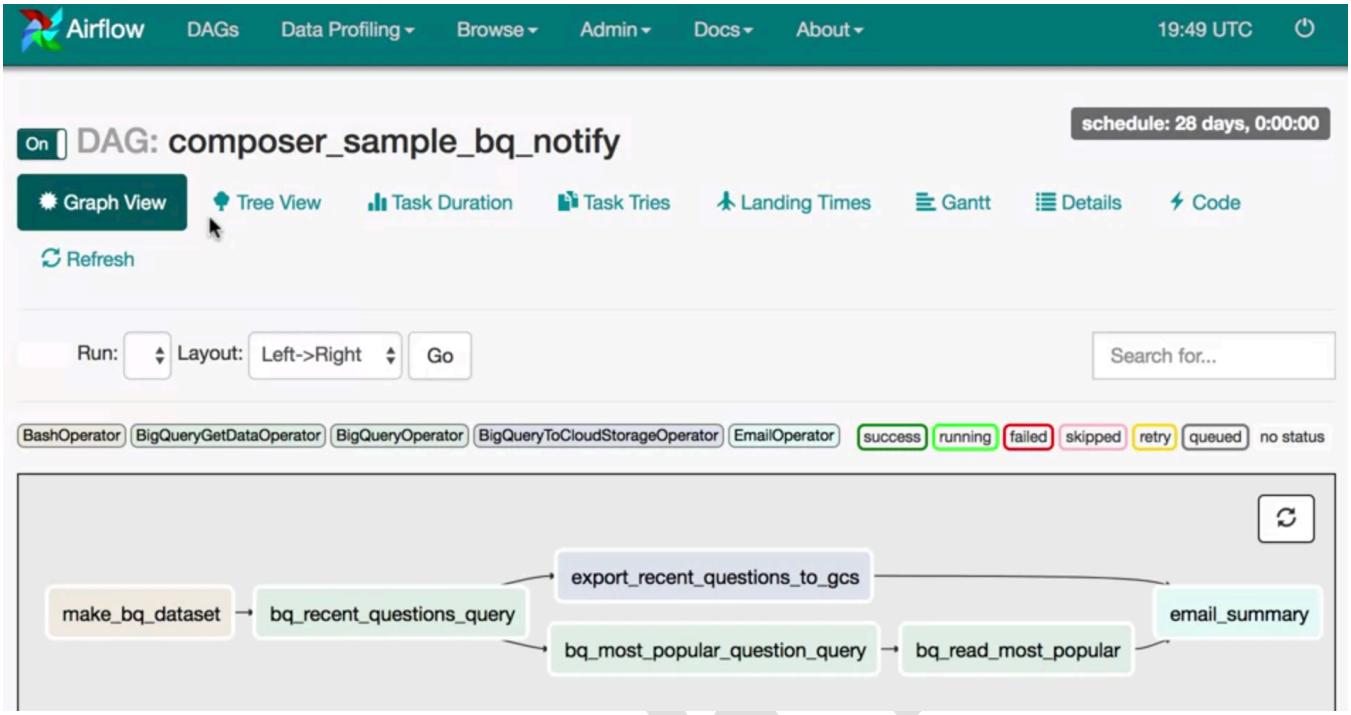
- You can use the Airflow Admin page to manage the entire execution
    - You can schedule DAGs
    - Trigger execution
    - Monitor Logs
    - Monitor status of DAG
    - Monitor status of each task within a DAG

7.3 Airflow: DAGs & Operators

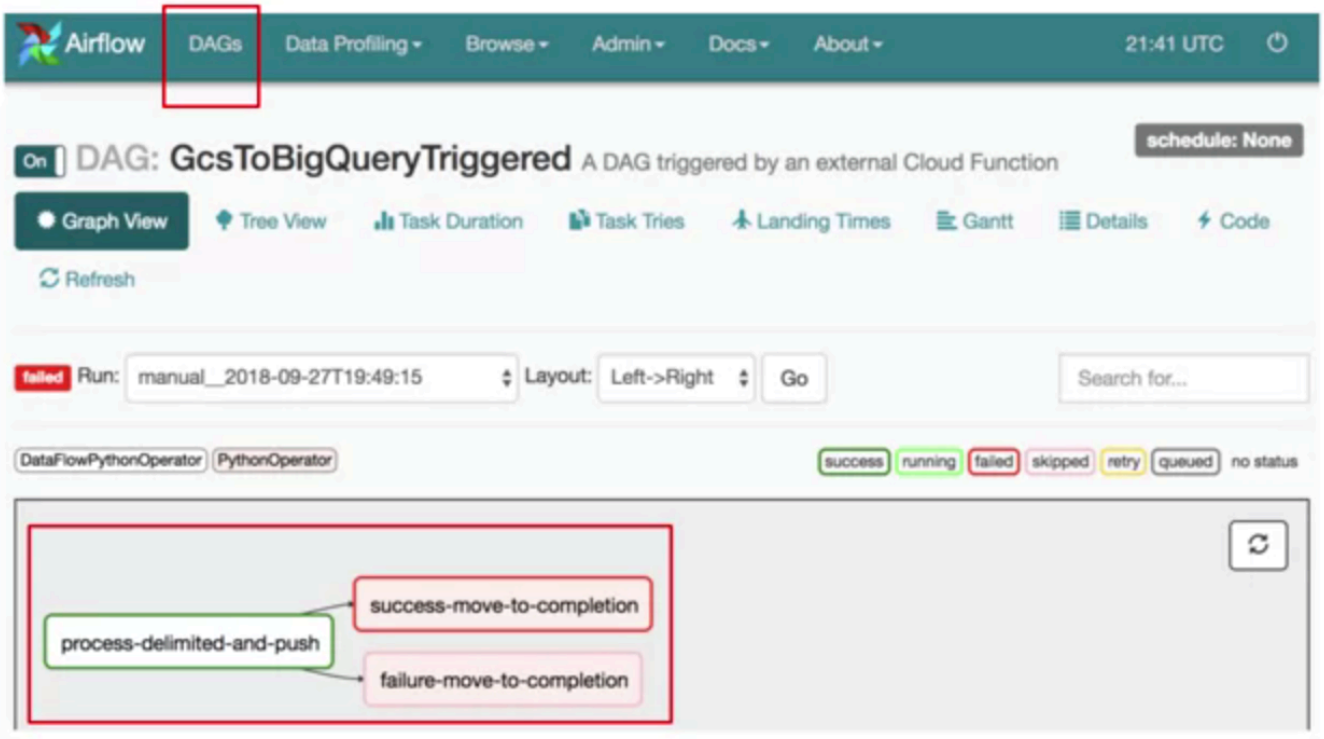
- Airflow workflows are written in Python.
- You only have one Python file for each DAG.



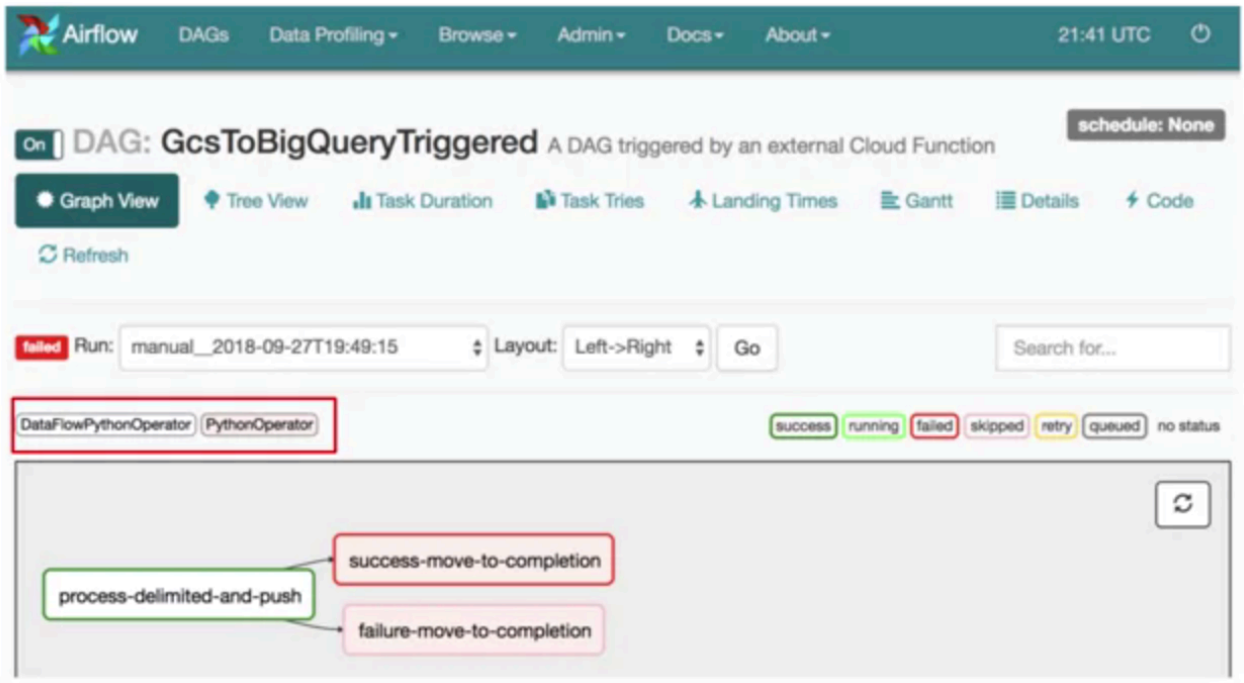
- There are a series of user created tasks in each DAG file that invoke predefined operators.
- Once you've uploaded the python file to the DAGs folder you can then navigate back to the Airflow web server.
- Under DAGs you'll see the DAG you just created with code visually represented as a directed graph, nodes and edges.



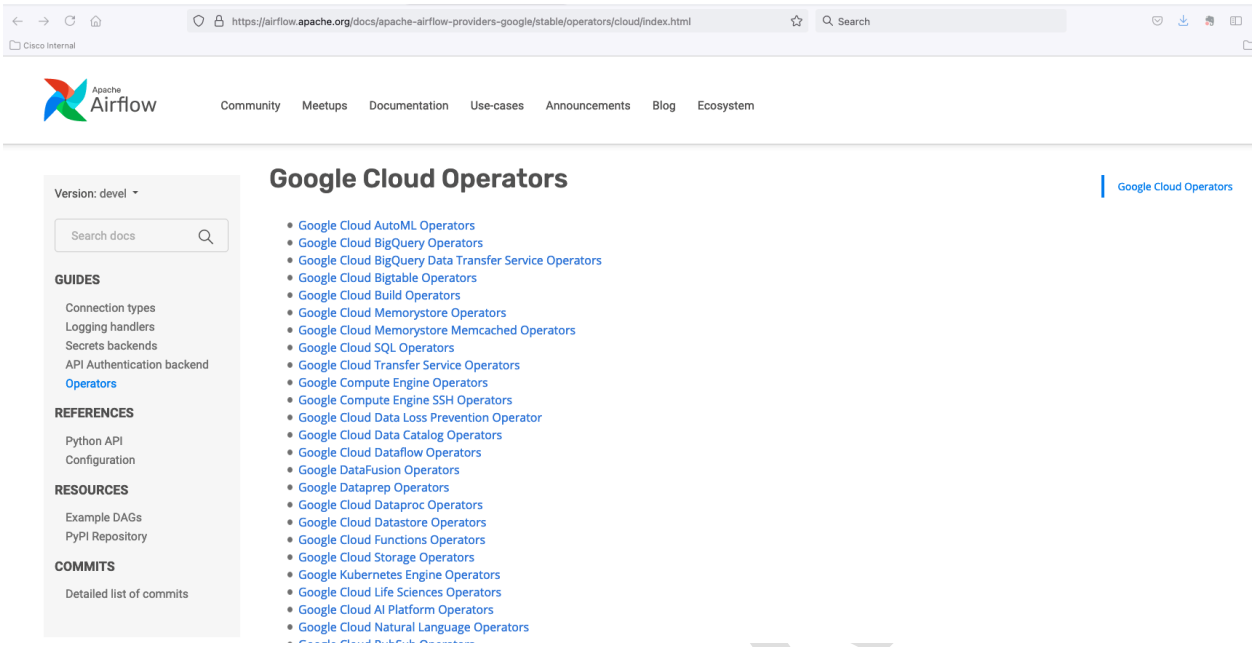
- Below is an example of a DAG that isn't strictly sequential. There's a decision made to run one node or a different node, based on the outcome of the parent node.



- Regardless of the size and shape of your workflow DAG, one common thread for all workflows is the common ***operators*** that are used.
- If the DAG itself is how to run the code in the workflow, the ***operator*** is specifying the what that actually gets done as part of the task.

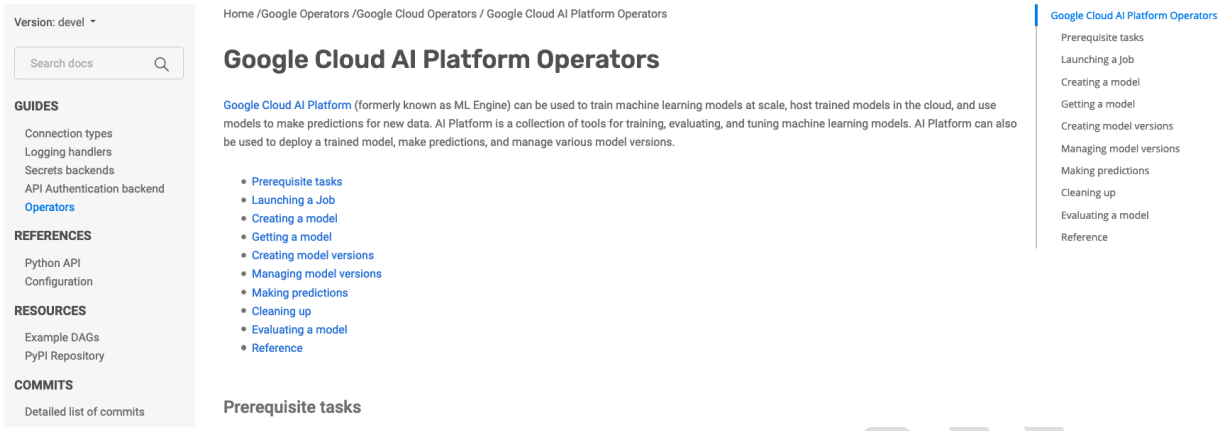


- In this simple example, we're calling on the dataflow Python operator and the general Python operator.
- Airflow has many operators which you can invoke to use the tasks that you want to complete.
- Operators are usually atomic in a task, which means that generally,
- You only see one operator per one task.
- Below is a list directly from the Apache Airflow docs of all the services that Airflow can orchestrate to.



- Below is a page that details operators that we can invoke in a task to call on the BigQuery service for querying and other data related tasks.

- In the same DAG file, after the BigQuery operator is complete, we can then make a service call to a cloud AI Platform operator to kick off a new training job and manage our model like incrementing the version.



- Airflow DAG can have operators that send out tasks to other cloud providers.

Providers packages

Providers packages include integrations with third party integrations. They are updated independently of the Apache Airflow core.

- Airbyte
- Amazon
- Apache Beam
- Apache Cassandra
- Apache Druid
- Apache HDFS
- Apache Hive
- Apache Kylin
- Apache Livy
- Apache Pig
- Apache Pinot
- Apache Spark
- Apache Sqoop
- Celery
- IBM Cloudant
- Kubernetes
- Databricks
- Datadog
- Dingding
- Discord
- Docker
- Elasticsearch
- Exasol
- Facebook
- File Transfer Protocol (FTP)
- Google
- gRPC
- Hashicorp
- Hypertext Transfer Protocol (HTTP)
- Internet Message Access Protocol (IMAP)
- Java Database Connectivity (JDBC)
- Jenkins
- Jira
- Microsoft Azure
- Microsoft SQL Server (MSSQL)
- Windows Remote Management (WinRM)
- MongoDB
- MySQL
- Neo4J
- ODBC
- OpenFaaS
- Opsgenie
- Oracle
- Pagerduty
- Papermill
- Plexus
- PostgreSQL
- Presto
- Qubole
- Redis
- Salesforce
- Samba
- Segment
- Sendgrid
- SFTP
- Singularity
- Slack
- Snowflake
- SQLite
- SSH
- Tableau
- Telegram
- Trino
- Vertica
- Yandex
- Zendesk

- This is great for hybrid workflows where you have components across multiple cloud platforms or even a combination of cloud and on-premise.

7.3.1 Airflow: Operators for ML example

- Example DAG

```
# update training data
t1 = BigQueryOperator(
)

# BigQuery training data export to GCS
t2 = BigQueryToCloudStorageOperator(
)


# ML Engine training job
t3 = MLEngineTrainingOperator(
)

# App Engine deploy new version
t4 = AppEngineVersionOperator(
)

# DAG dependencies
t2.set_upstream(t1)
t3.set_upstream(t2)
t4.set_upstream(t3)
```

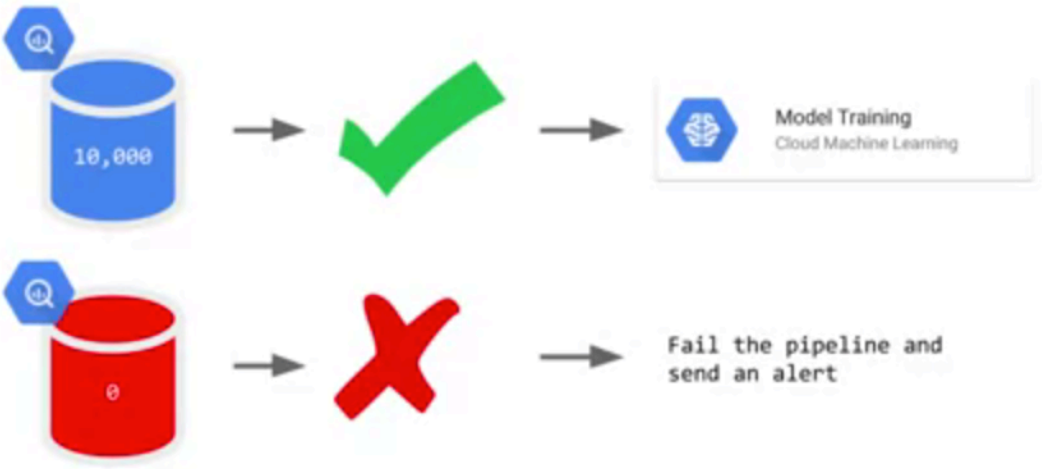
- Here, you see four tasks, t1, t2, t3, and t4, and the four operators corresponding to four Google Cloud Platform services.
- The first two are concerned with getting fresh model data from the BigQuery dataset and into GCS for consumption by our ML model later on in the workflow.
    - The **BigQueryOperator** allows you to specify a SQL query to run against a BigQuery data set.

```
from airflow.contrib.operators import bigquery_operator

# constants or can be dynamic based on Airflow macros
max_query_date = '2018-02-01' # {{ macros.ds_add(ds, -7) }}
min_query_date = '2018-01-01' # {{ macros.ds_add(ds, -1) }}

    # Query recent StackOverflow questions.
    bq_recent_questions_query = bigquery_operator.BigQueryOperator(

        task_id='bq_recent_questions_query',

        bql="""
        SELECT owner_display_name, title, view_count
        FROM `bigquery-public-data.stackoverflow.posts_questions`
        WHERE creation_date < CAST('{max_date}' AS TIMESTAMP)
            AND creation_date >= CAST('{min_date}' AS TIMESTAMP)
        ORDER BY view_count DESC
        LIMIT 100
        """.format(max_date=max_query_date, min_date=min_query_date),

        use_legacy_sql=False,

        destination_dataset_table=bq_recent_questions_table_id)
```

- Here, we pass in a query which returns the top 100 most popular Stack Overflow posts from the BigQuery public data set for a specified date range that you see there in the WHERE clause.
- You can parameterize pieces of this SQL statement, like what we did here.
- You can make the parameters themselves dynamic and have them be based on the DAG schedule date.
    - For example, macros.ds_add(ds, -7) means this is a week before the DAG is scheduled to run.

- What can you do to guard our workflows against upstream data issues?
  - Use ***BigQueryCheckOperator*** to setup automatic SQL-based health checks.

```
# this should never return 0. If it does, error.
SELECT COUNT(*) AS num_training_records
FROM upstream_data_source;
```



https://airflow.apache.org/integration.html#bigquerycheckoperator

- What about the case where something in the data values themselves changed?
  - For example, below, the scale switched from Fahrenheit to Celsius for a few records.

Date	Temperature
2018-10-01	37.77
2018-09-30	98
2018-09-29	100
2018-09-28	99

```
# compare previous values
SELECT {metrics_threshold_dict_key} FROM {table}
    WHERE {date_filter_column}=<date>
```

        - We can use ***BigQueryIntervalCheckOperator*** which can notice huge changes or swings in the data distribution, and then fail the workflow until further investigation.
            - This operator checks the values of metrics, given as SQL expressions, whether or not they're within a certain distance or tolerance of the ones from our parameter value that you sent.
- The next two operators handle retraining the model by submitting a job to Cloud Machine Learning Engine, and then deploying the updated model to App Engine via an API endpoint.
    - The ***ML Engine training operator*** is how Cloud Composer interacts with Cloud Machine Learning Engine, and thus gives it the ability to periodically schedule new jobs to be sent to CMLE as part of your automated workflow.

```
t3 = MLEngineTrainingOperator(
    task_id='ml_engine_training_op',
    project_id=PROJECT_ID,
    job_id=job_id,
    package_uris=[PACKAGE_URI],
    training_python_module='trainer.task',
    training_args=training_args,
    region=REGION,
    scale_tier='CUSTOM',
    master_type='complex_model_m_gpu',
    dag=dag
)
```

    - Finally, once your model is retrained and ready to be delivered as an API endpoint for serving, we need to redeploy our App Engine project with the latest model.

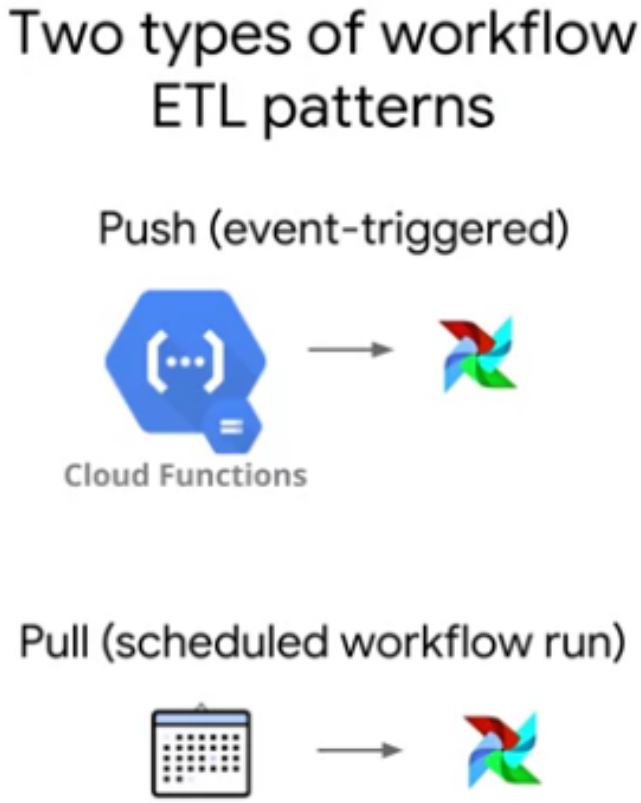
```
t4 = AppEngineVersionOperator(
    task_id='app_engine_deploy_version',
    project_id=PROJECT_ID,
    service_id='default',
    region=REGION,
    service_spec=None,
    dag=dag
)
```

- At the end of most DAG files, you'll find is the actual order in which we want these tasks and operators to be run.

```
# DAG dependencies
t2.set_upstream(t1)
t3.set_upstream(t2)
t4.set_upstream(t3)
```

  - For example, t2, won't run until t1 is completed.

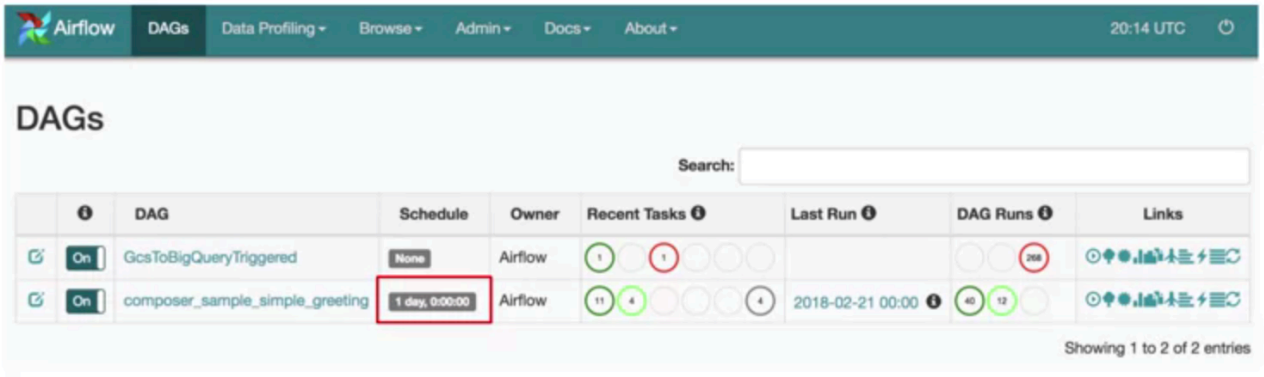
7.4 Cloud Composer: Scheduling & Triggers



- There are two general patterns for ETL workflows:
  - ***Event triggered, aka push***:
    - Push a new file to GCS and then the workflow kicks off.
    - You can use cloud functions, pub-sub etc.
  - ***Pull or Scheduled***
    - Airflow at a set time could look in your GCS folder and take all the contents that are found there for its scheduled workflow run.

7.4.1 Scheduled events

- Here, we set a schedule or a periodic run of the workflow.
- To view the schedules for your DAGs, you first launch the airflow web server UI from cloud composer.

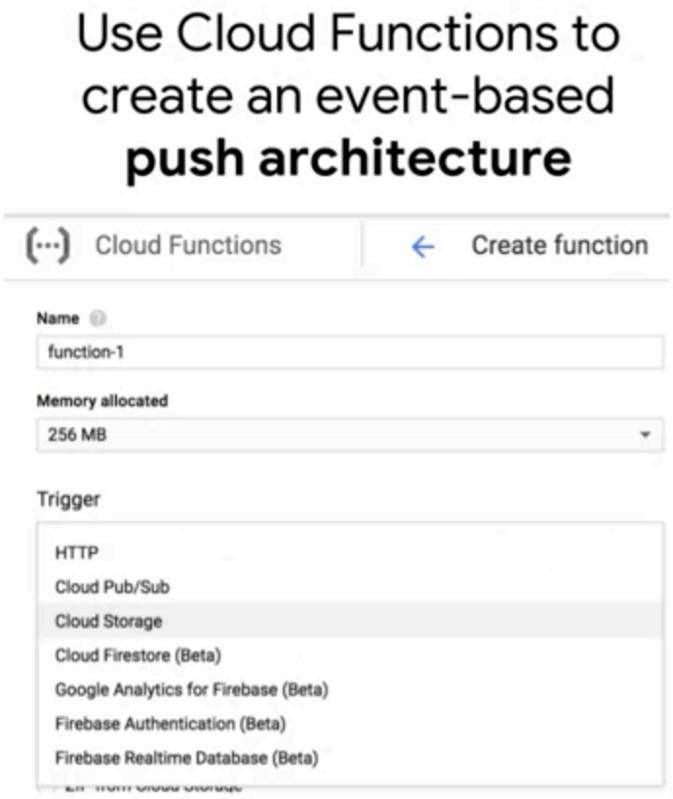



```
with models.DAG(
        'composer_sample_simple_greeting',
        schedule_interval=datetime.timedelta(days=1),
        default_args=default_dag_args) as dag:
```

- Clicking on this schedule of one day here in the UI won't allow you to edit the schedule but instead takes you to the history of all the runs for that particular workflow.
- To set the schedule, you specify the schedule interval in your DAG code as shown above.

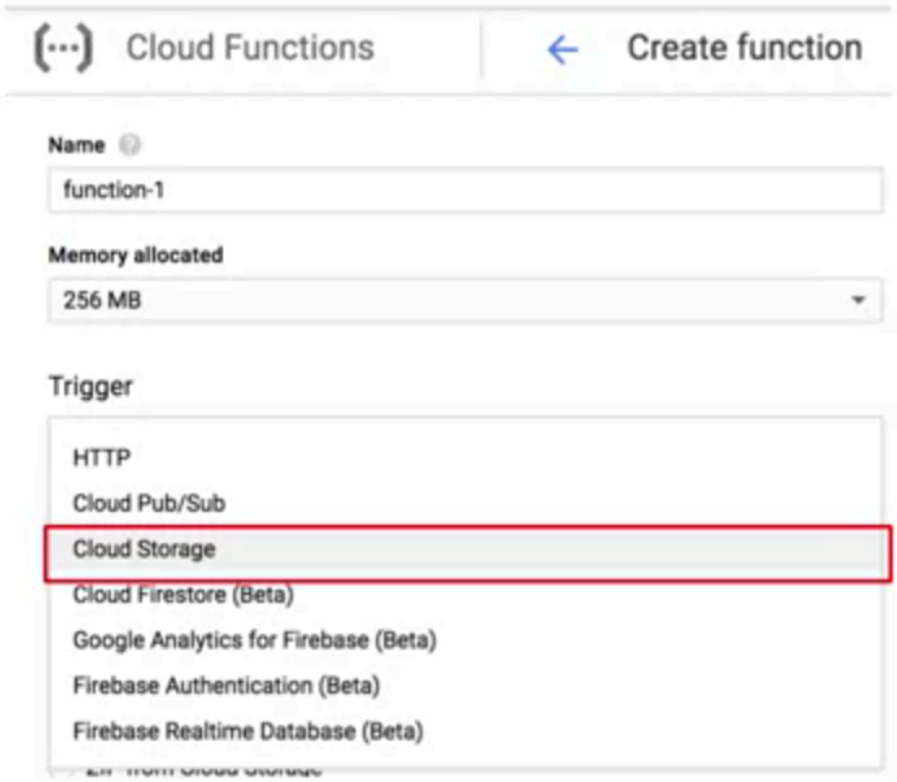
7.4.2 Event Driven (using cloud functions)

- We can use Cloud Functions to create our event-driven or push architecture workflow.

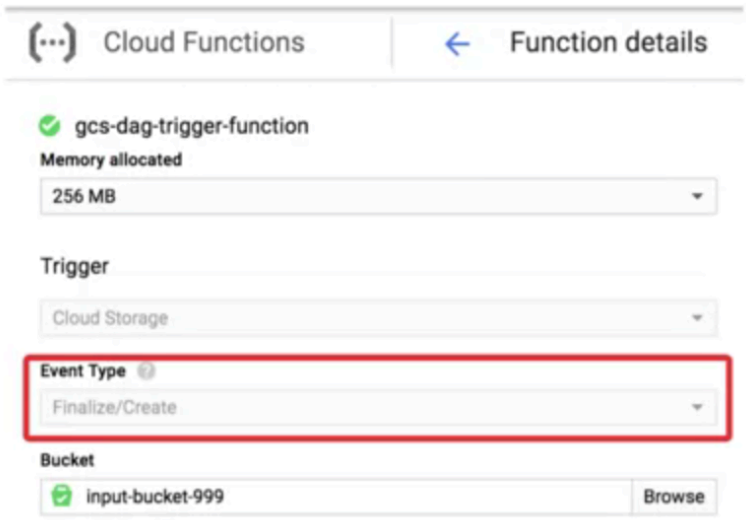


- In addition to using to GCS, you can also trigger events based on HTTP requests, pub/sub topics, Firestore, Firebase etc as shown above.

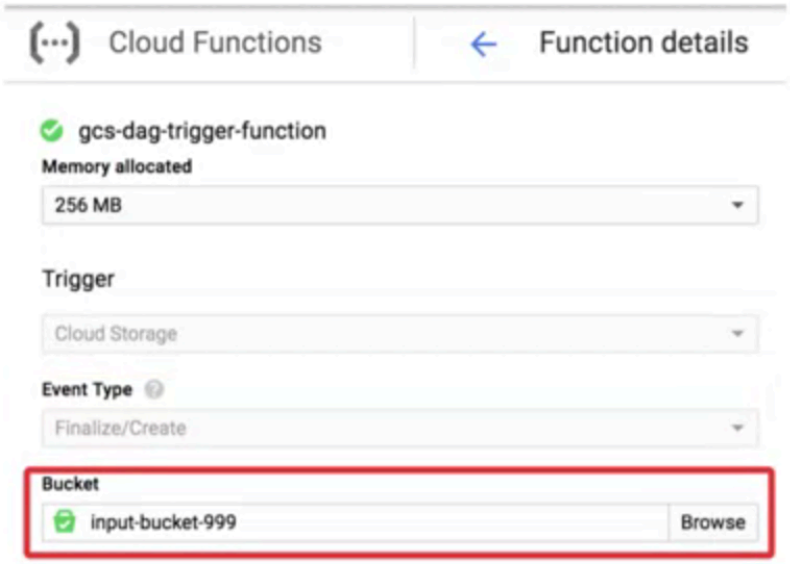
- Below is an example for the setup
    - Assume we have a CSV file or a set of files loaded to GCS,so we'll choose a Cloud Storage trigger for our function.



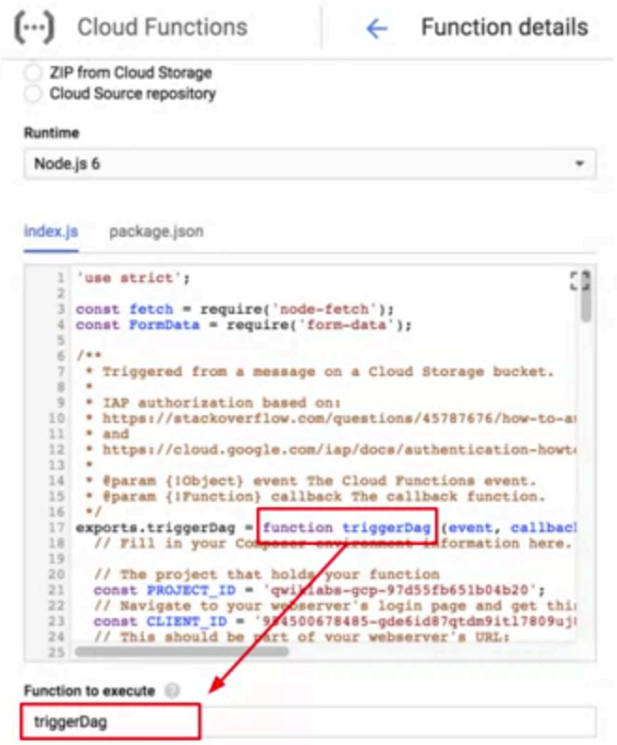
    - Specify an event type. Here it is when we finalize or create new files.



- Specify a bucket to watch.



- Create an actual function, which is written in JavaScript and specify that as the function to execute. NOTE: The name is case sensitive.



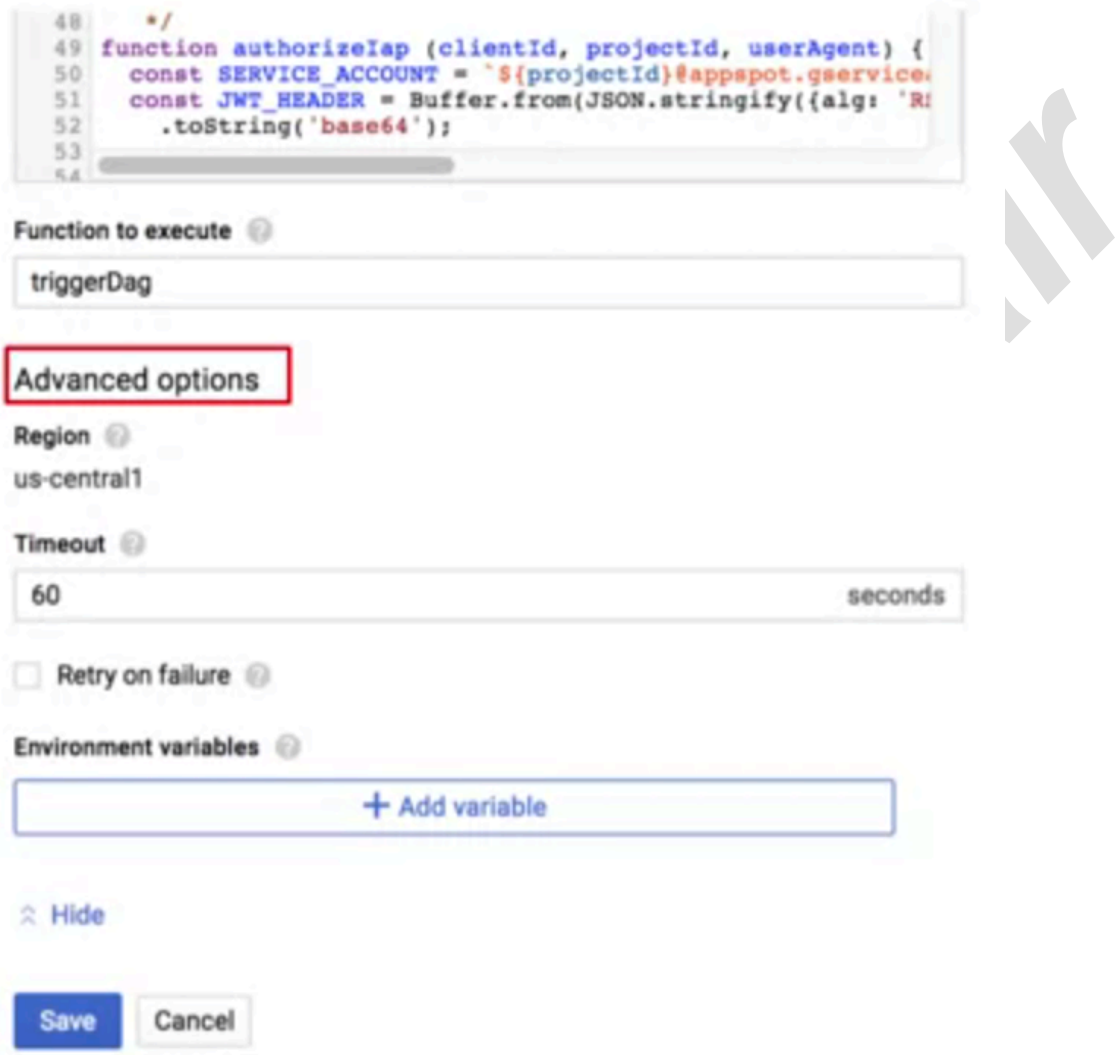
- Below is sample code for the function

index.js package.json

```
 * @param {!Object} event The Cloud Functions event.
 * @param {!Function} callback The callback function.
 */
exports.triggerDag = function triggerDag (event, callback) {
  // Fill in your Composer environment information here.

  // The project that holds your function
  const PROJECT_ID = 'qwiklabs-gcp-                    ';
  // Navigate to your webserver's login page and get this from the URL
  const CLIENT_ID = '                                                    .apps.googleusercontent.com';
  // This should be part of your webserver's URL:
  // {tenant-project-id}.appspot.com
  const WEBSERVER_ID = 'b9:                    -tp';
  // The name of the DAG you wish to trigger
  const DAG_NAME = 'GcsToBigQueryTriggered';

  // Other constants
  const WEBSERVER_URL = `https://${WEBSERVER_ID}.appspot.com/api/experimental/dags/${DAG_NAME}/dag
  const USER_AGENT = 'gcf-event-trigger';
  const BODY = {'conf': JSON.stringify(event.data)};

  // Make the request
  authorizeIap(CLIENT_ID, PROJECT_ID, USER_AGENT)
    .then(function iapAuthorizationCallback (iap) {
      makeIapPostRequest(WEBSERVER_URL, BODY, iap.idToken, USER_AGENT, iap.jwt);
    })
    .then(_ => callback(null))
    .catch(callback);
};
```

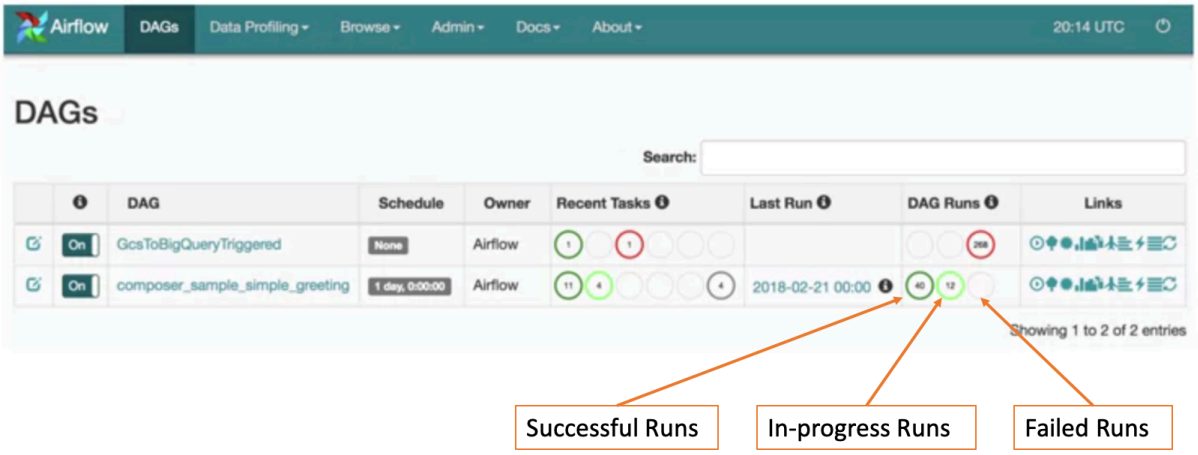
- Specify a name for our function: Here, it is triggerDag.
- Specify airflow environment details to be triggered, and which dag in that airflow environment.
- In this case, is looking for a dag called GcsToBigQueryTriggered.
- Specify a few constants that are provided which construct the airflow URL and how to make a call to the environment. (a POST request)

- There are several advanced options that you can specify for your cloud function like adjusting the default timeout for the function trigger to be made.If you want your function to automatically retry in code failure.

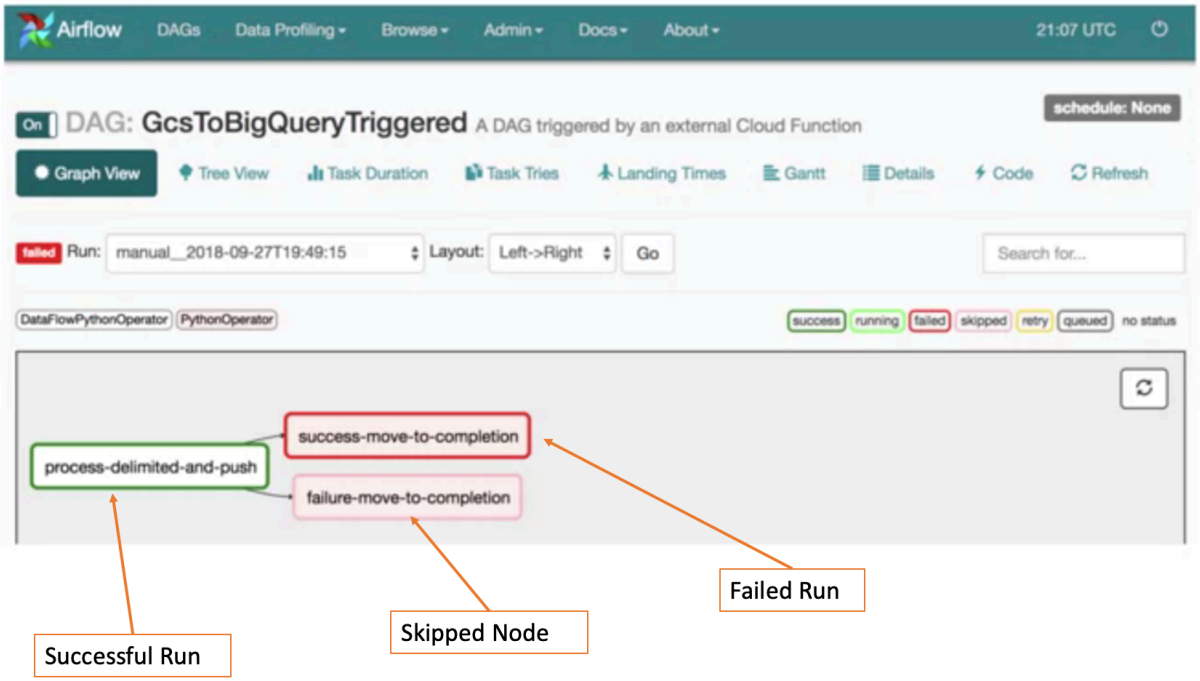


7.5 Cloud Composer: Monitoring and Logging

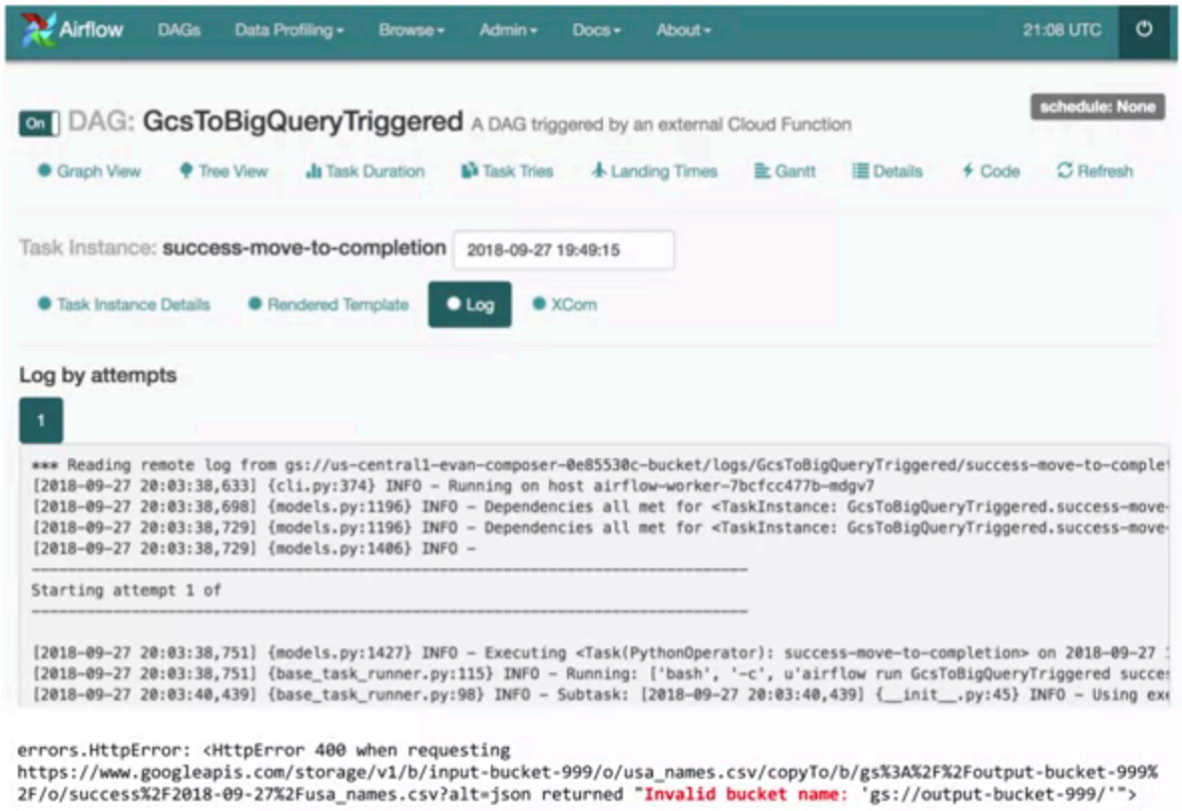
- Monitor status on Admin page



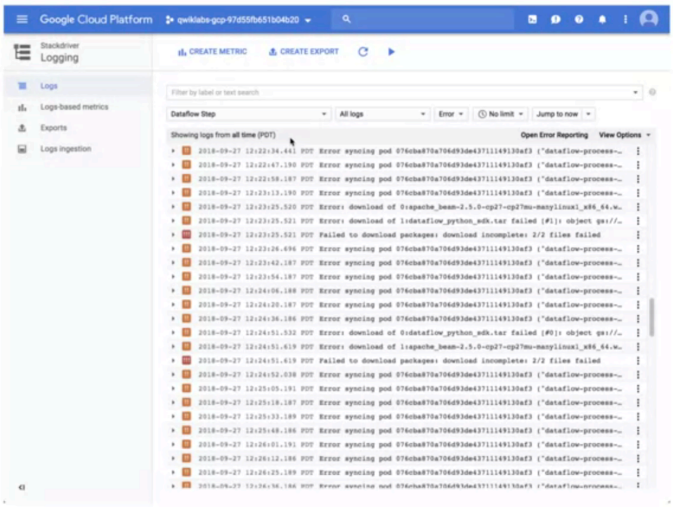
- Monitor status of each DAG



- View logs for each node in a DAG



- Another tool in your toolkit for diagnosing Airflow failures is your general GCP logging.
    - Since Airflow launches other GCP services through tasks, you can see and filter for errors for services in ***Stackdriver***, as you would debugging any other normal application.

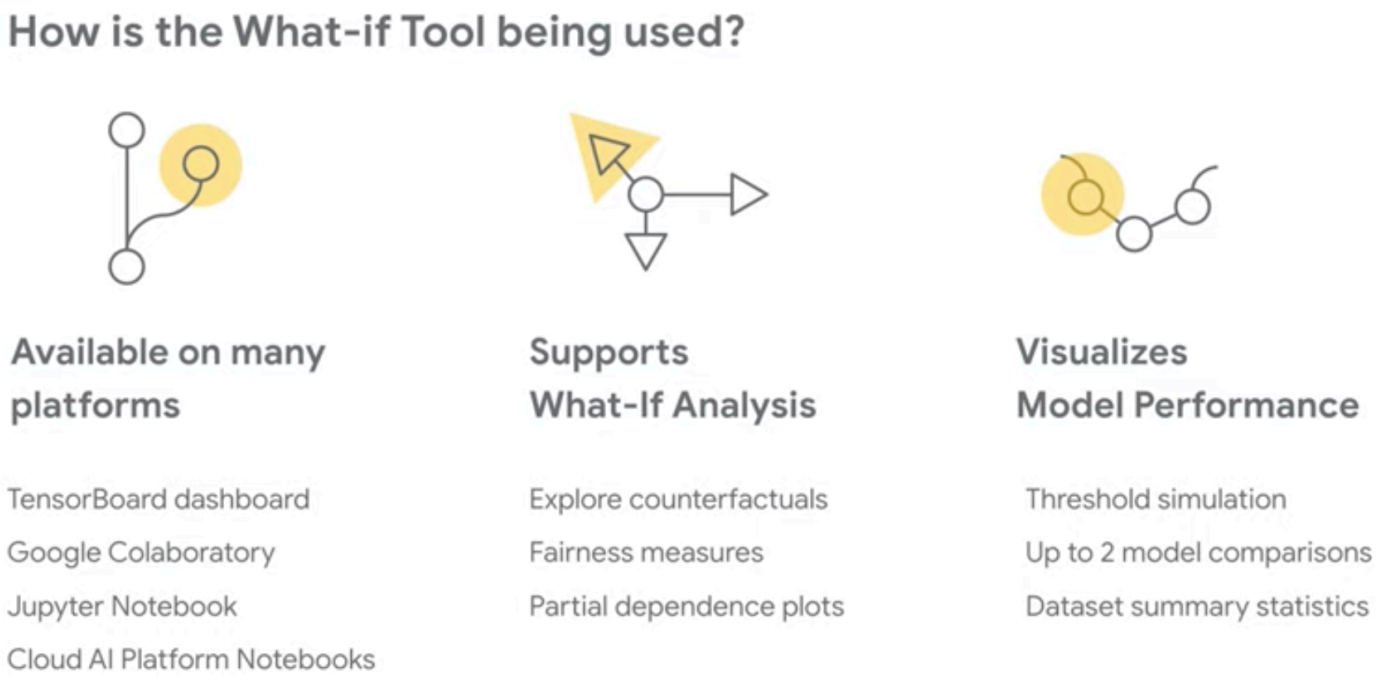


- Before really scheduling or triggering your workflow, try running it manually yourself first. And ensuring that you can get it to complete successfully before tying it to a more aggressive or automated schedule.

8 Explainable AI: What-If Tool (WIT)

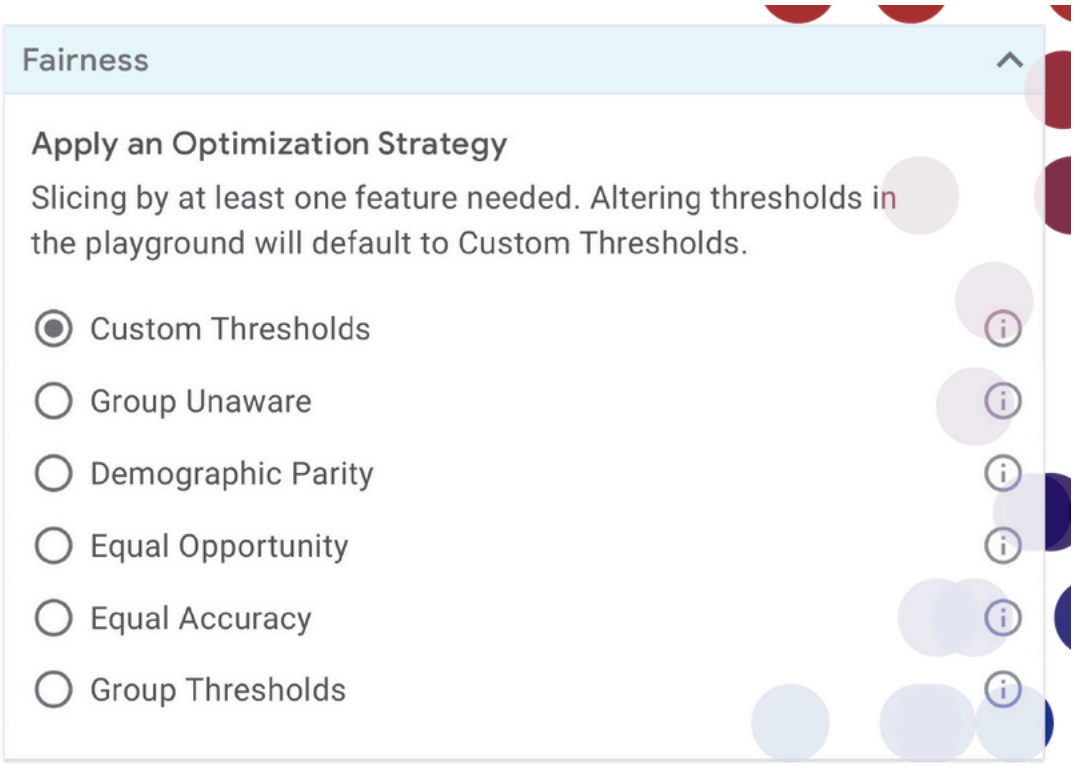
8.1 Overview

- The What-If Tool (WIT) provides an easy-to-use interface for expanding understanding of black-box classification and regression ML models.
- With the plugin, you can perform inference on a large set of examples and immediately visualize the results in a variety of ways.
- Additionally, examples can be edited manually or programmatically and re-run through the model in order to see the results of the changes.
- It contains tooling for investigating model performance and fairness over subsets of a dataset.



8.2 AI Fairness

- What-If presents five buttons, each of which sorts the data according to a different type of fairness, based on mathematical measures.



- Example Scenario:
  - let's say you're a mortgage lender using a machine learning system to sort through loan applications, only 30 percent of which come from women.
  - You need to figure out what gender mix of approved and denied applications would be fair.

- Interpretation of various optimization strategies:

  - **Group Unaware**
    - Totally disregard the gender mix of the applicants who are given loans.
    - People should be chosen purely on their merits – what's sometimes called “group unaware” fairness.
    - If you are group aware, you'd have to kick out a qualified man to make room for a less qualified woman, because there are a limited number of approvals possible.

        - **What this means for our model:**
            - If the system has to be 60 percent certain that a man will pay back a loan in order to recommend approving his loan, it ought to be 60 percent certain about women as well, even if that were to mean, in the very worst case, that the approved pile contains only applications from men.

- **Group Thresholds**
    - Historical biases in the data used to create the system's model can make women look less loan-worthy than men.
    - For example, women's work histories are more likely to be interrupted by multi-month gaps – maternity leave, for example – and that might count against them in the machine learning system's model.
    - We should be able to adjust the confidence thresholds for men and women independently.
    - **What this means for our model**:
        - If we have to tell the model that to put a man's application into the Approved pile requires a 0.6 confidence level that he will pay back a loan, but only a 0.3 confidence level that a woman will, so be it.

- **Demographic parity**
    - The composition of the set of people who are granted loans should reflect the percentage of applicants
    - If 30 percent of the applicants are women, then 30 percent of the pool of approved applicants ought to be women.

- **Equal Opportunity**
    - The same percentage of men and women who are likely to succeed at loans are given loans.
    - You don't want 90 percent of the male good risks to make it into the acceptance pile, but only 40 percent of the female good risks to do so.
    - This meets the lender's objective of identifying loan-worthy applicants but avoids favoring one gender over another in terms of risk.

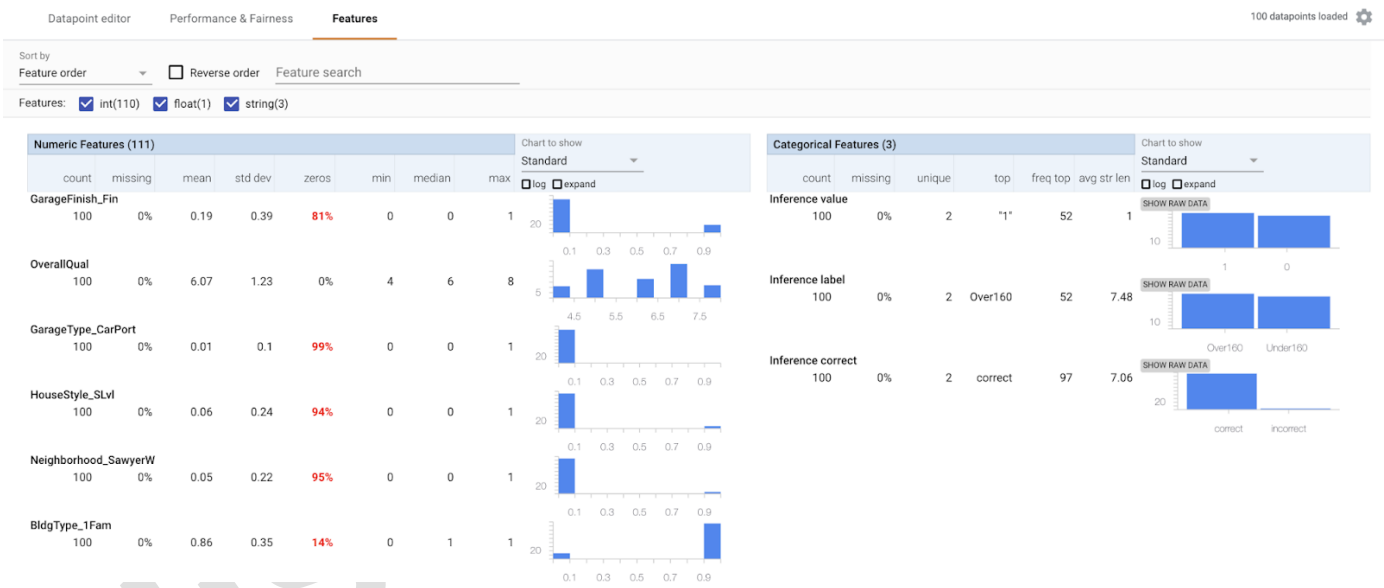
- **Equal Accuracy**
    - Instead of making sure that the genders end up with the same percentage of successes (equal opportunity), each gender should instead have the same percentage of people who were rightly judged as loan-worthy or who were rightly judged as not loan-worthy.
    - That is, the percentage of right classifications (as loan-worthy or as not) should be the same for both genders.

8.3 Analyzing a dataset using What If Tool

- We'll be using a housing dataset from Kaggle to perform fairness analysis in the What-if Tool.
- The dataset includes many pieces of data on a house (square feet, number of bedrooms, kitchen quality, etc.) along with its sale price.
- We will be predicting whether a house will sell for more or less than $160k.
- First Load the data and visualize using WIT

```
config_builder = (WitConfigBuilder(data.tolist(),
                                   data.columns.tolist())
WitWidget(config_builder, height=800)
```

- The data looks as follows:

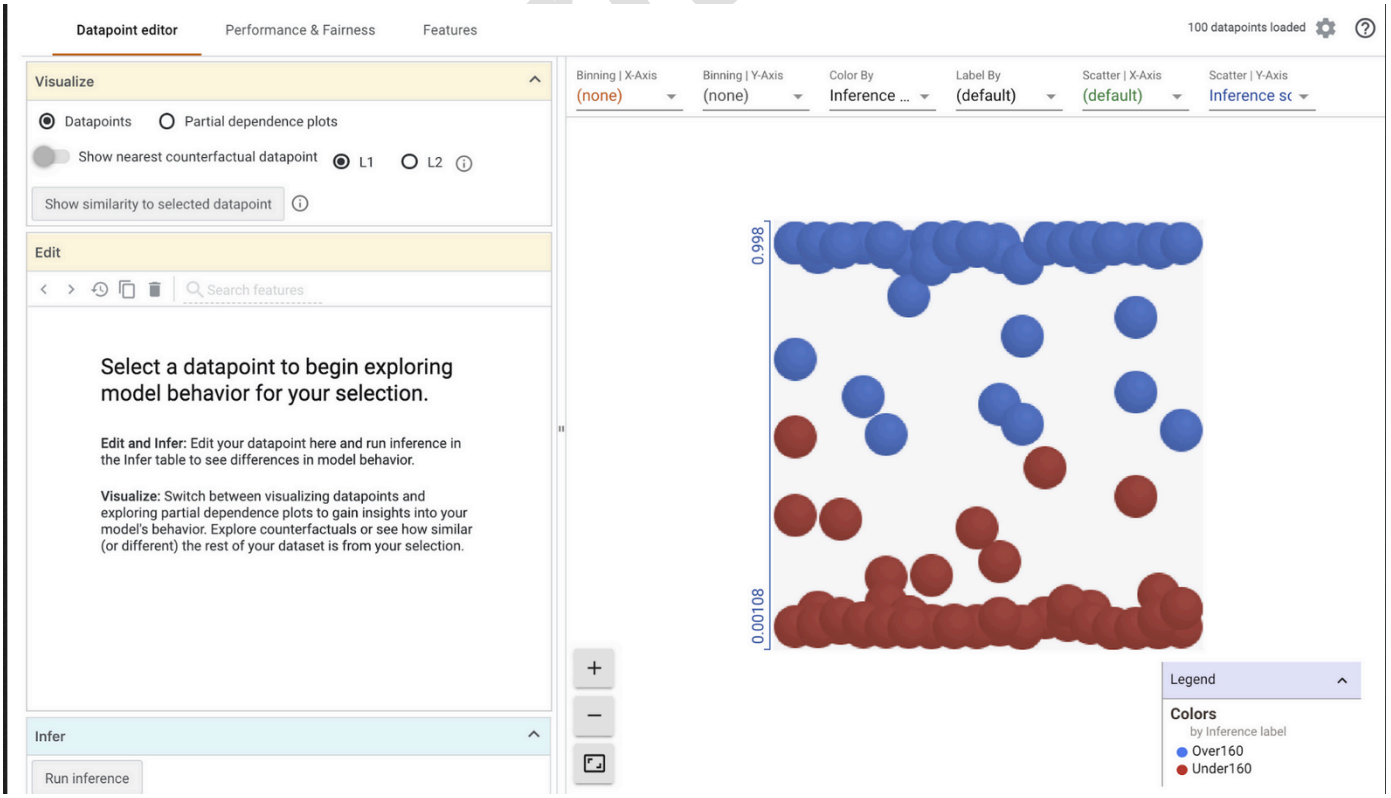


- It was originally intended as a regression problem, but nearly every regression problem can be converted to classification.
- To highlight the What-if Tool features, we turned this problem into a classification problem to predict whether a house is worth more or less than $160k.
    - We purposely chose the $160k threshold to make the label classes as balanced as possible
    - There are 715 houses less than $160k and 745 worth more than $160k.
- Next, train and deploy the model.

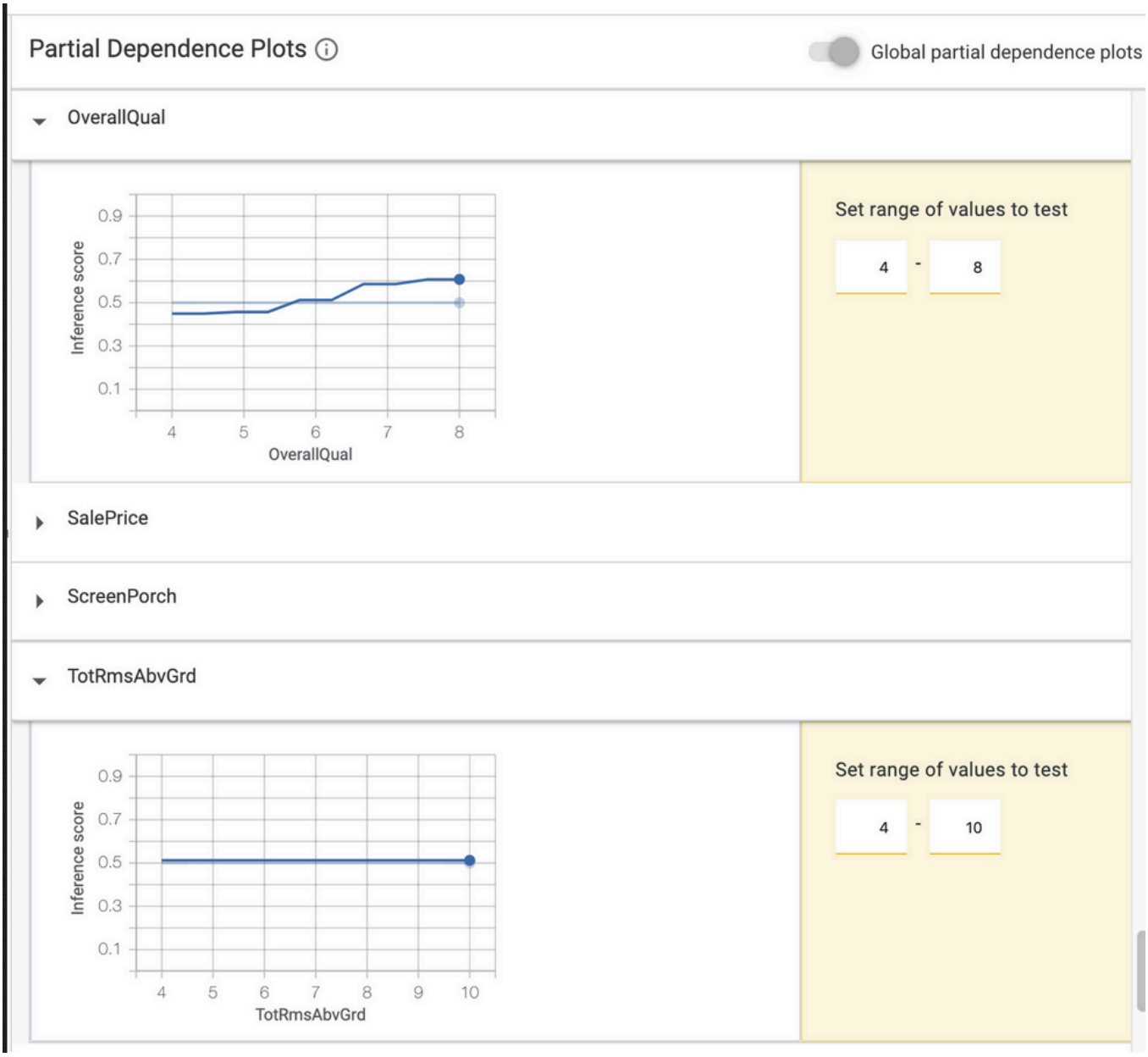
- We then connect the model to the what if tool as follows:

```
# This prediction adjustment function is needed since the
What-If Tool expects
# a model's prediction output as a list of scores for each
class.

def adjust_prediction(pred):
  return [1 - pred, pred]

config_builder = (WitConfigBuilder(test_examples.tolist(),
                  data.columns.tolist() + ['SalePrice'])
              .set_ai_platform_model('your-gcp-project',
                                     'housing', 'v1',
                                      djust_prediction =
                                       adjust_prediction)
              .set_target_feature('SalePrice')
              .set_label_vocab(['Under160', 'Over160']))
WitWidget(config_builder)
```

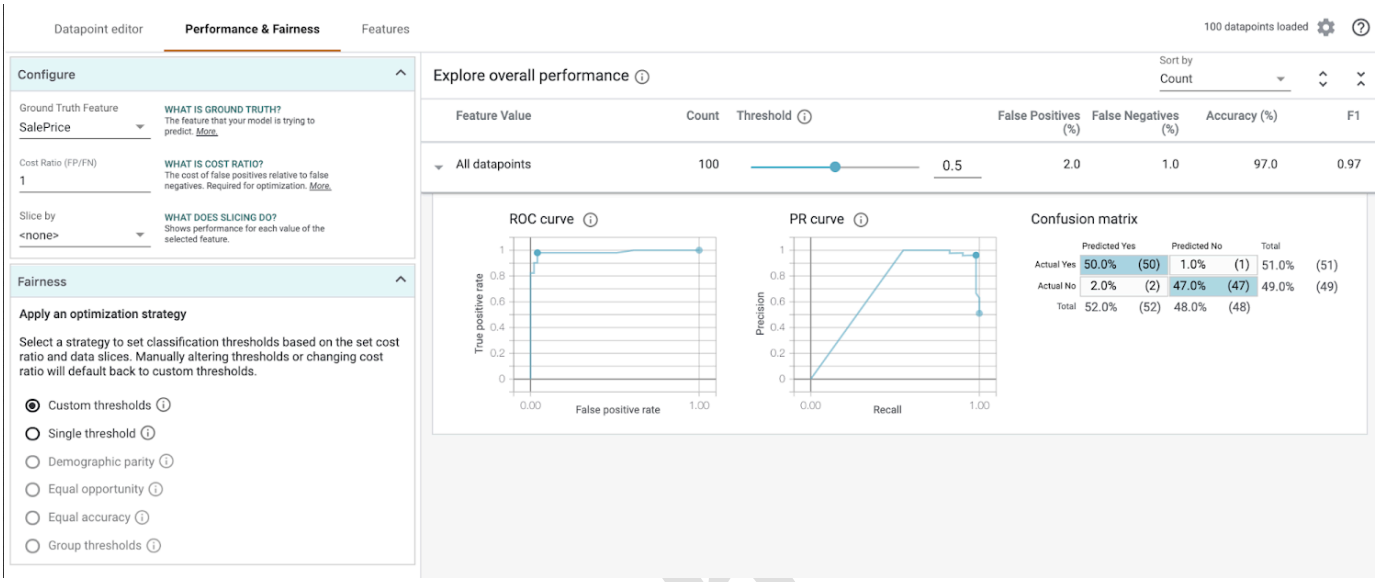
- We get the following visualization



- If you select Partial dependence plots, you can see how individual features impact the model's prediction
    - for an individual data point (if you have one selected), or
    - globally across all data points.

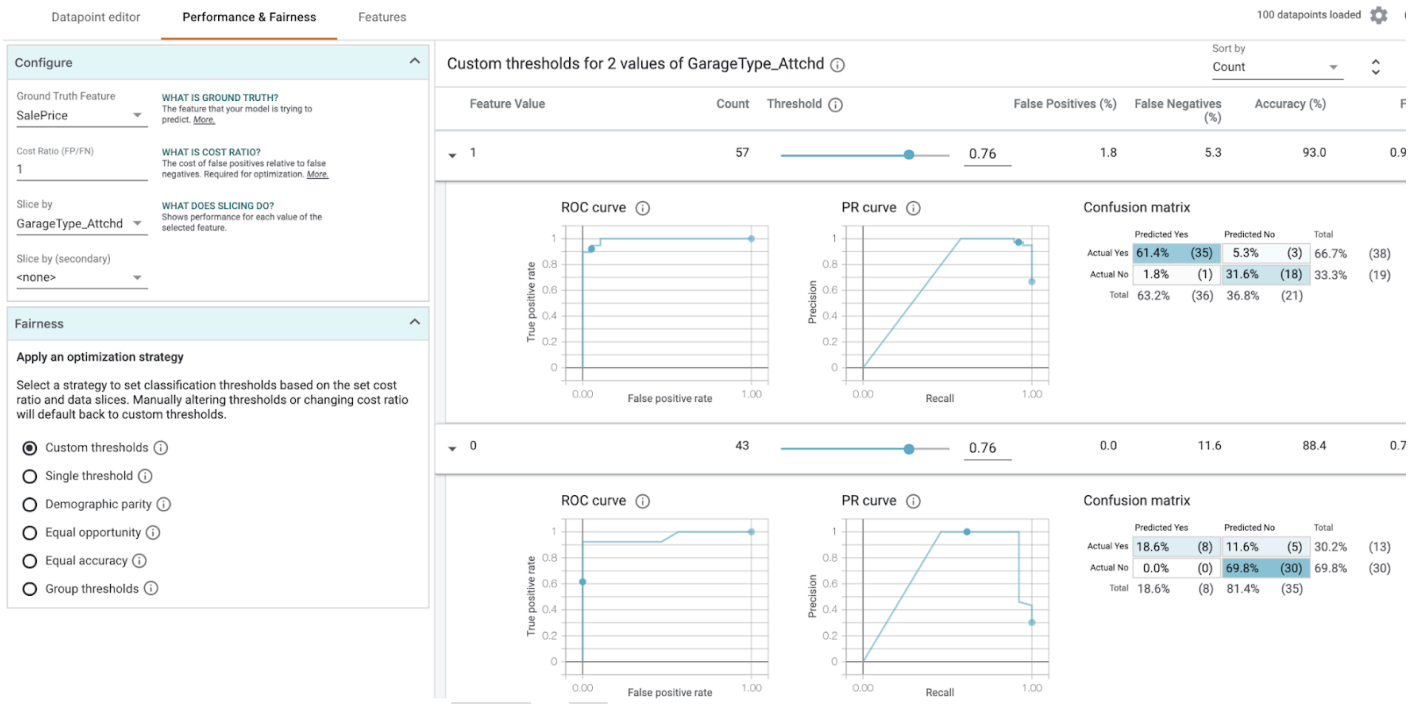


- We now explore the performance & Fairness Tab

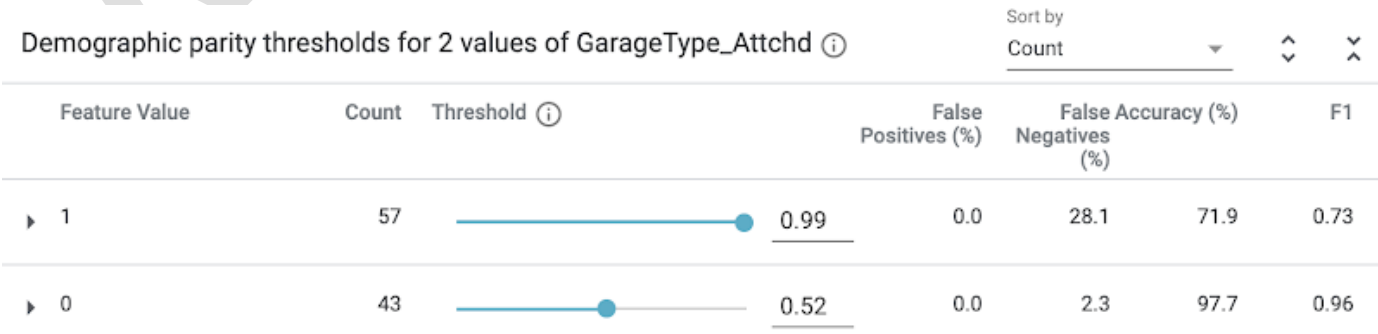


- In the "***Explore overall performance***" section, we can see various metrics related to our model's accuracy.
  - By default the Threshold slider starts at 0.5.
    - This means that our model will classify any prediction value above 0.5 as over $160k, and anything less than 0.5 will be classified as less than $160k.
  - The threshold is something you need to determine after you've trained your model, and the What-if Tool can help you determine the best threshold value based on what you want to optimize for.
  - The confusion matrix tells us the percentage of correct predictions for each class.
  - ROC and Precision / Recall (PR) are also common metrics for model accuracy.
  - We'll get the best insights from this tab once we start slicing our data.

- In the Configure section in the top left of the What-if Tool, select a feature from the Slice by dropdown.
    - First, let's look at "GarageType_Attchd",
        - This indicates whether the garage is attached to the house (0 for no, 1 for yes):



    - Notice that houses with an attached garage have a higher likelihood that our model will value them at more than $160k
    - Say we want our model to price houses with attached and unattached garages in the same way.
    - we care most about having the same percentage of positive classifications across classes, while still achieving the highest possible accuracy within that constraint.
    - For this we should select ***Demographic parity*** from the Fairness section on the bottom left:



- If we don't want the garage placement to influence our model's price, we need to use different thresholds for houses depending on whether their garage is attached.
  - The model will predict the house to be worth over $160k when the prediction score is .99 or higher.
  - Alternatively, a house without an attached garage should be classified as over $160k if the model predicts 0.52 or higher.

- If we instead use the "***Equal opportunity***" strategy,

Equal opportunity thresholds for 2 values of GarageType_Attchd

Sort by: Count

Feature Value	Count	Threshold	False Positives (%)	False Negatives (%)	Accuracy (%)	F1
1	57	0.8	1.8	5.3	93.0	0.95
0	43	0.52	0.0	2.3	97.7	0.96

  - This will optimize for high accuracy predictions within the positive class and ensure an equal true positive rate across data slices.
  - This will choose the thresholds that ensure houses that are likely worth over $160k are given a fair chance of being classified for that outcome by our model.
- If we choose the "***Equal accuracy***" strategy

Equal accuracy thresholds for 2 values of GarageType_Attchd

Sort by: Count

Feature Value	Count	Threshold	False Positives (%)	False Negatives (%)	Accuracy (%)	F1
1	57	0.5	3.5	0.0	96.5	0.97
0	43	0.52	0.0	2.3	97.7	0.96

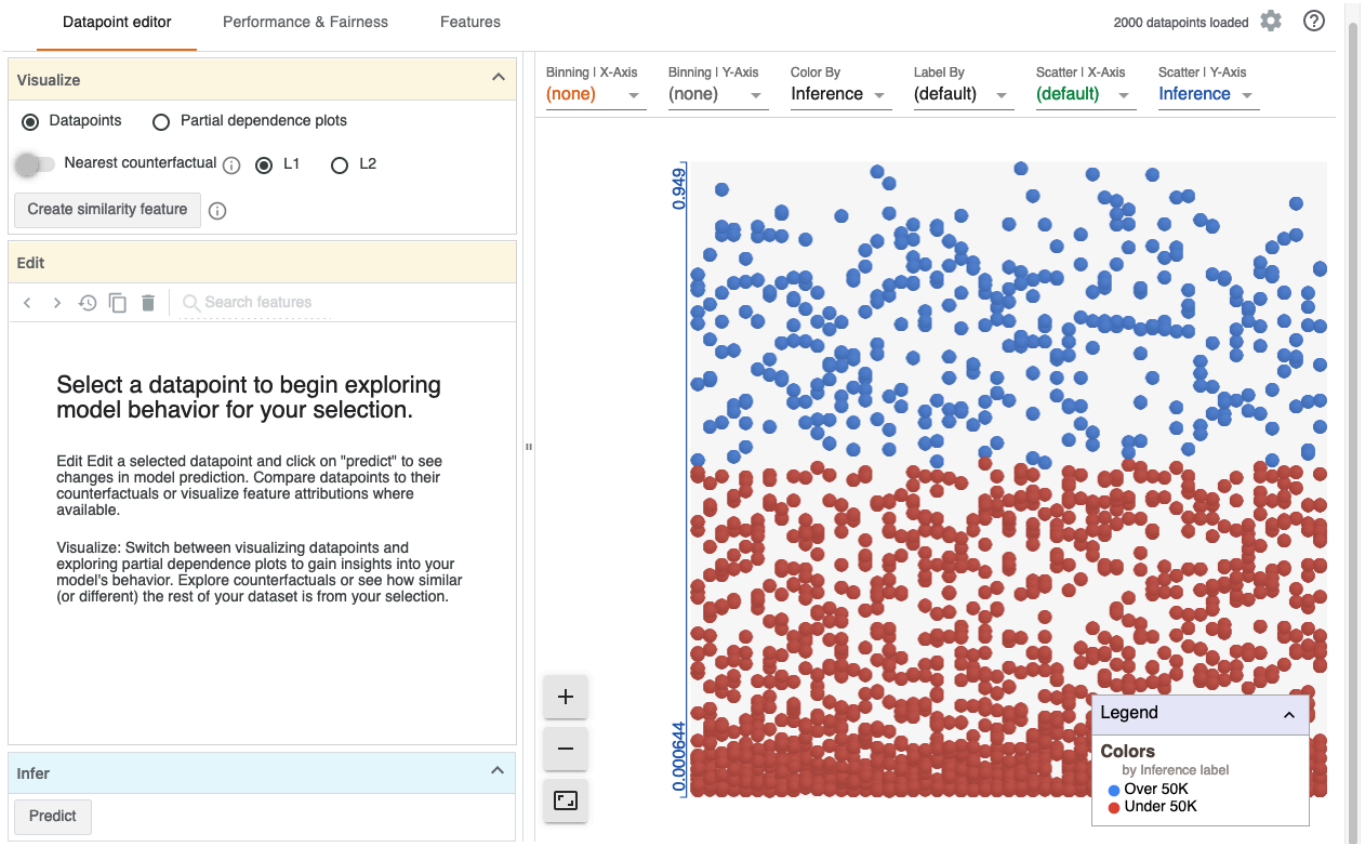
  - This will optimize for accuracy across both classes (positive and negative).

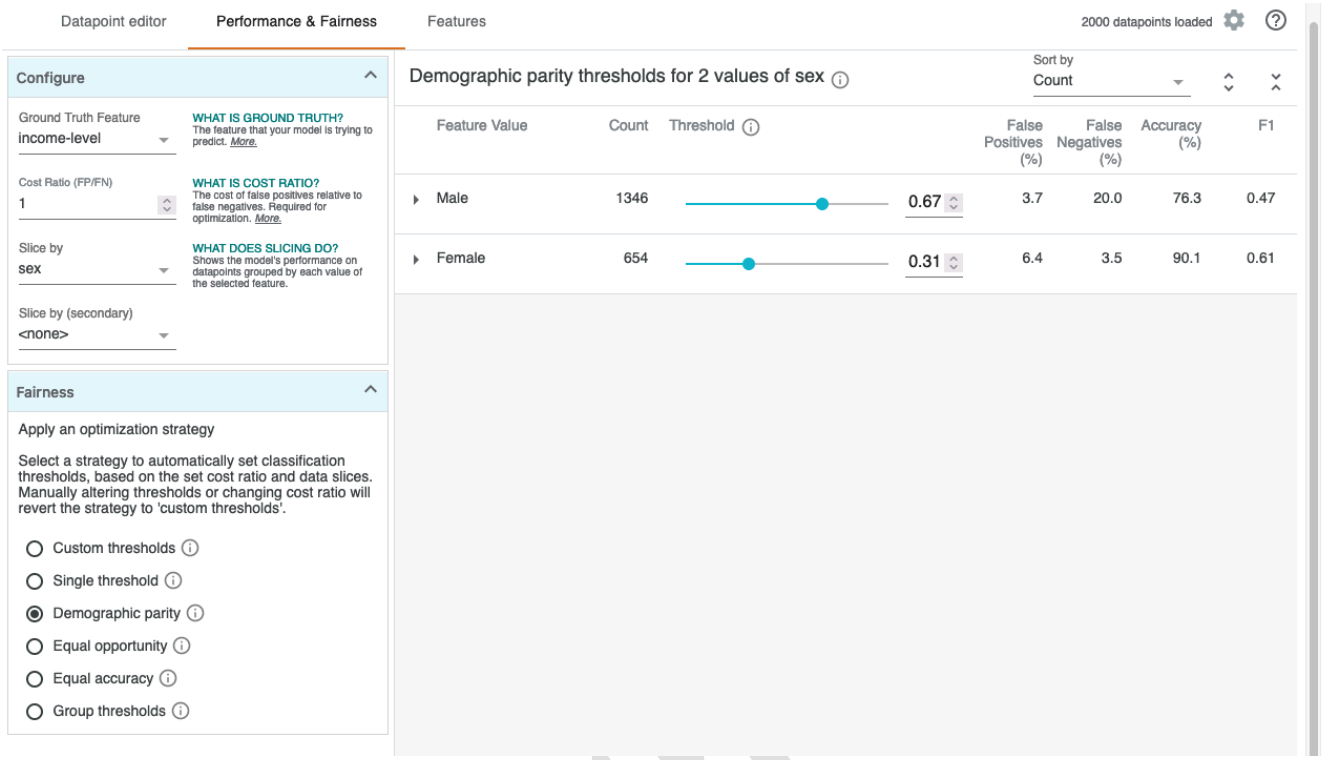
- We can do a similar slice analysis for other features, like neighborhood and house type, or we can do an intersectional analysis by slicing by two features at the same time.

8.4 Comparing models using What-If-Tool

- Analyze the first model using WIT

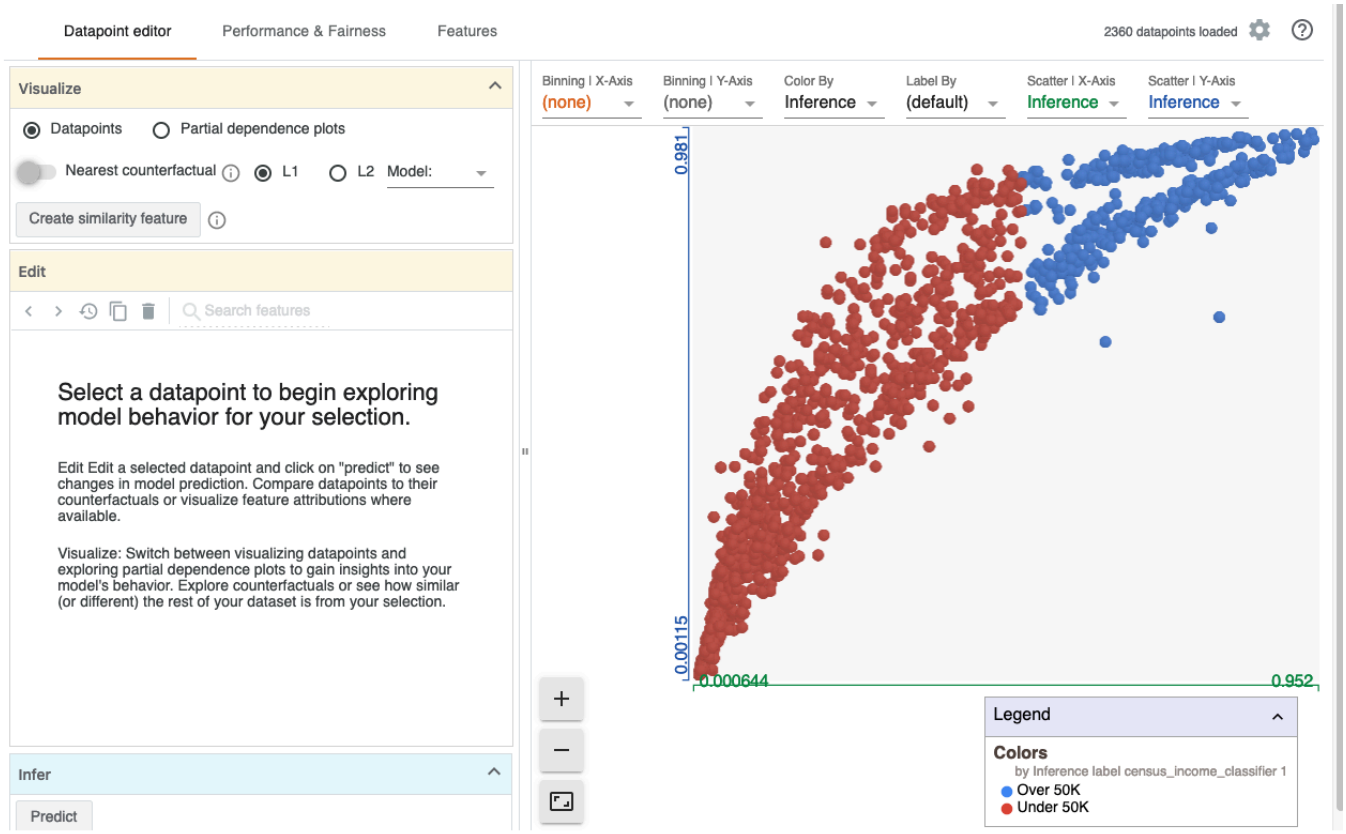
```
config_builder = (
    WitConfigBuilder(test_examples.tolist(), COLUMNS)
    .set_ai_platform_model(os.environ['QWIKLABS_PROJECT_ID'], 'census_income_classifier', 'original')
    .set_target_feature('income-level')
    .set_model_type('classification')
    .set_label_vocab(['Under 50K', 'Over 50K'])
)

WitWidget(config_builder, height=800)
```

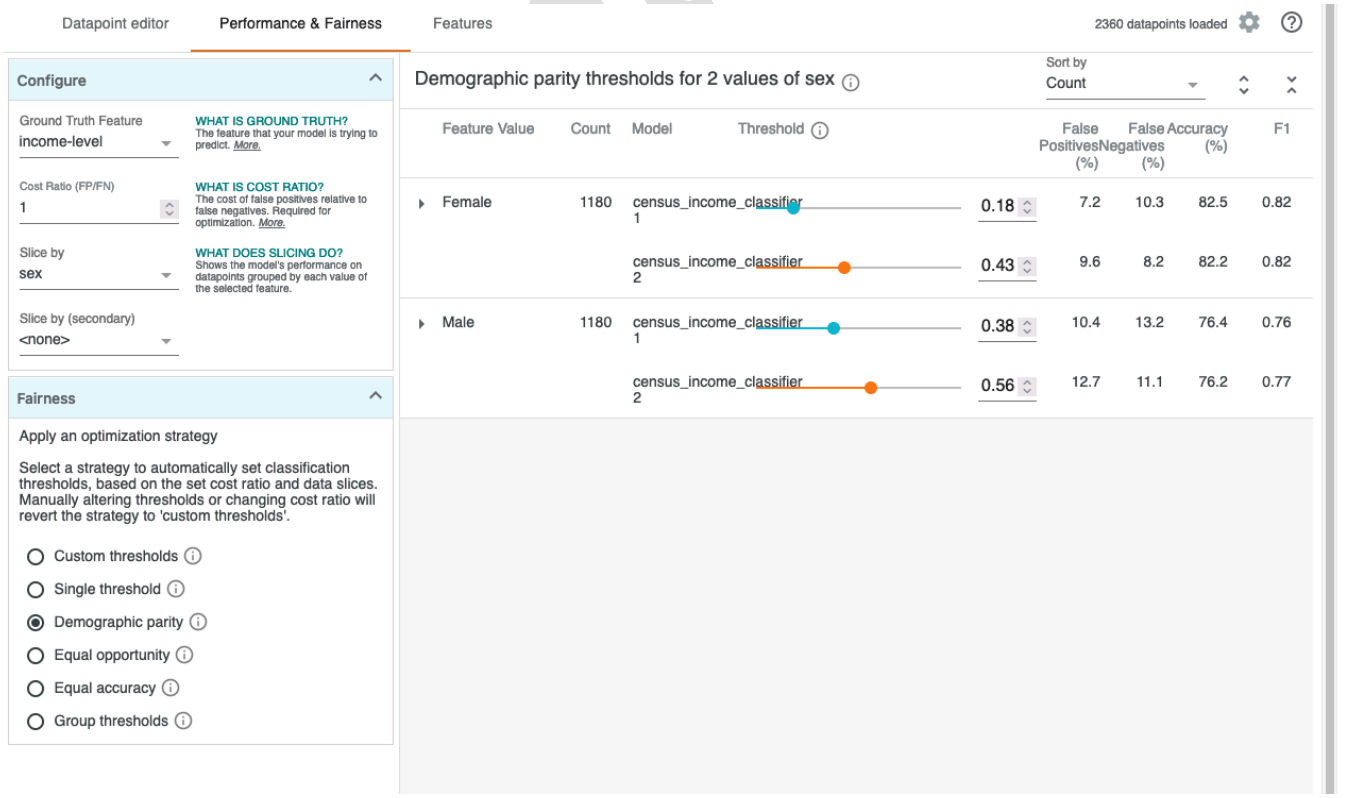




- Create\Get a balanced dataset based on analysis
- Create a second model using the new balanced dataset
- Compare the two models

```
config_builder = (
    WitConfigBuilder(bal_test_data.to_numpy()[1:].tolist(), COLUMNS)
    .set_ai_platform_model(os.environ['QWIKLABS_PROJECT_ID'], 'census_income_classifier', 'original')
    .set_compare_ai_platform_model(os.environ['QWIKLABS_PROJECT_ID'], 'census_income_classifier', 'balanced')
    .set_target_feature('income-level')
    .set_model_type('classification')
    .set_label_vocab(['Under 50K', 'Over 50K'])
)

WitWidget(config_builder, height=800)
```





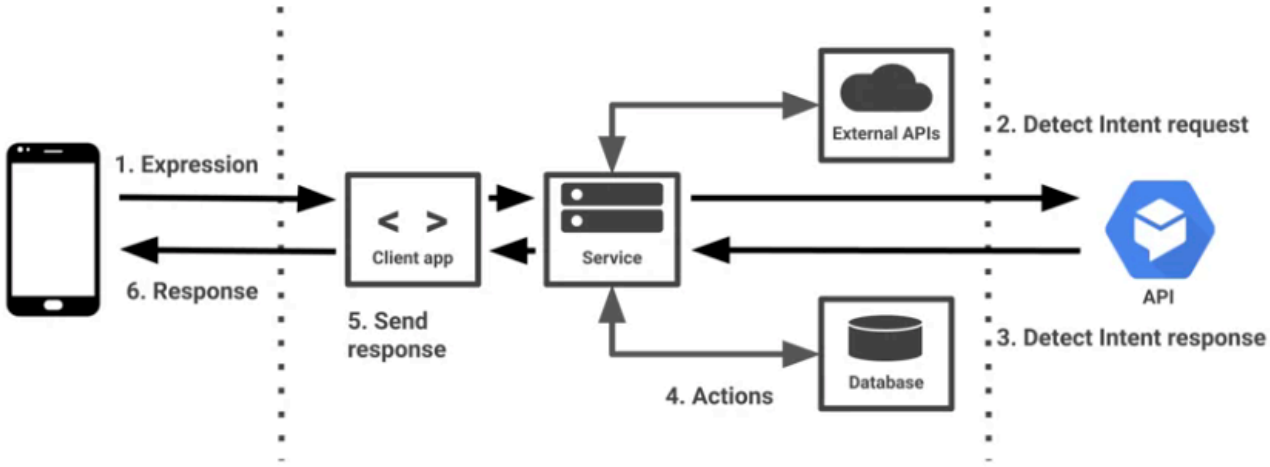
- We see the second model is much more balanced.

9 Contact Center AI

9.1 Some basic APIs

9.1.1 DialogFlow API

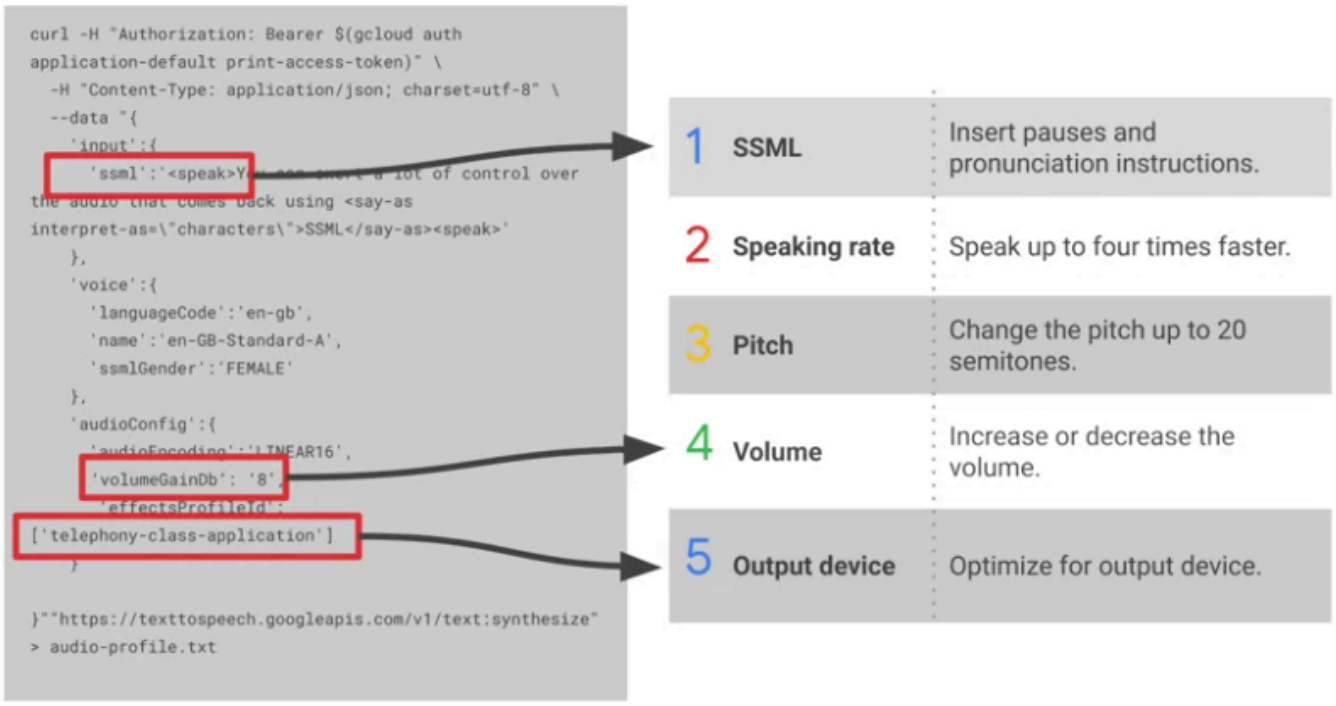
- Used to build conversational interfaces
- It can analyze text or audio and respond to a human in a natural chatty way
- The building blocks for any chatbot are:
    - **Intent**: Meaning behind what user types
    - **Entities**: Objects that your application acts on.
- A DialogFlow conversation takes place in a **session**.
    - A session has a **context** that is used to pass parameters from one intent to another.
- Integration using API



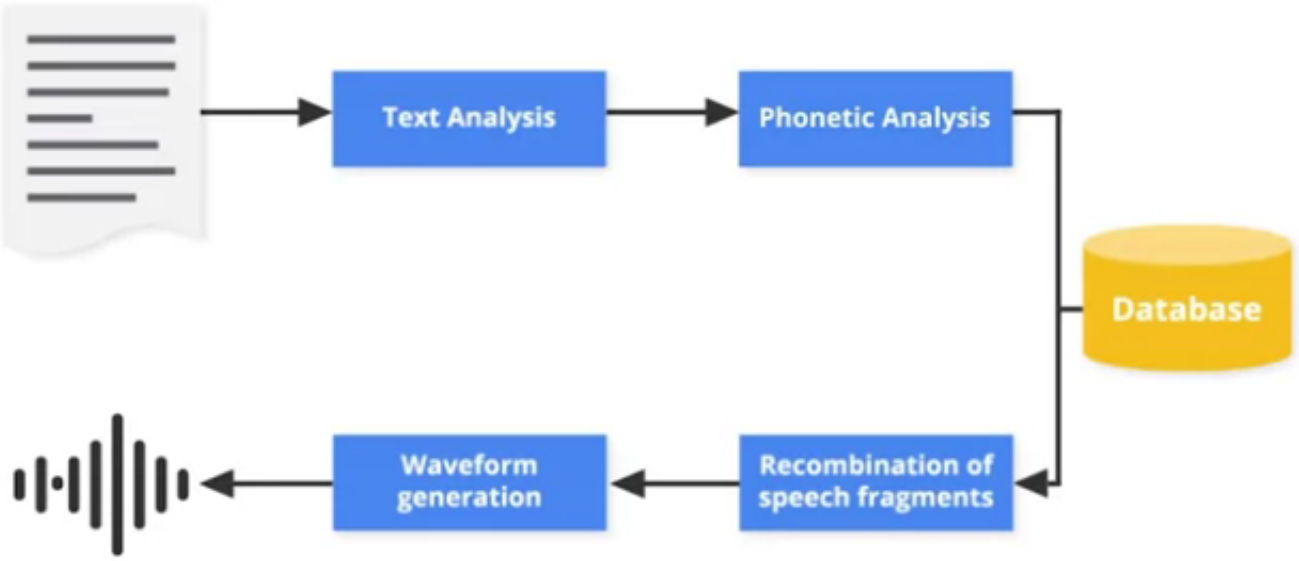
- Can also take audio as input.

9.1.2 Google Text to Speech (Cloud speech) API

- Given text as input, API provides the raw audio waveform as output.
  - This output is in base64 encoded format.
  - Client will need to base64 decode prior to feeding to an audio device
- API uses ***Speech Synthesis Markup Language*** (SSML)
  - Provides a lot of control on how output should be generated.



- Converting text to raw audio is called ***synthesis***. The output of synthesis is called ***synthetic speech***.



  - ***Concatenative text to speech***
    - Database has large number of short speech fragments recorded from a single speaker.
    - Fragments are broken into tiny chunks and recombined to form complete words and sentences.
    - This is called ***concatenative text to speech***.
  - **Parametric Text to Speech**
    - Uses a series of rules and parameters about grammar and mouth movements to guide a generated voice.
    - Cheaper and faster than concatenative TTS, but less natural sounding.

- **WaveNet Model**
  - Google uses a very different approach for synthesis.
  - Uses a waveNet model.
  - It generates raw audio waveforms from the scratch.
  - Uses a neural net similar to one used for images.
  - Produces a very natural voice

9.1.3 Cloud Natural Language API

- Cloud Natural Language API lets you extract information about people, places, events, (and more) mentioned in text documents, news articles, or blog posts.
- You can use it to understand sentiment about your product on social media, or parse intent from customer conversations happening in a call center or a messaging app.
- You can even upload text documents for analysis.
- Some of the features include:
    - **Syntax Analysis**
        - Extract tokens and sentences, identify parts of speech (PoS) and create dependency parse trees for each sentence.
    - **Entity Recognition**
        - Identify entities and label by types such as person, organization, location, events, products and media.
    - **Sentiment Analysis**
        - Understand the overall sentiment expressed in a block of text.
    - **Content Classification**
        - Classify documents in predefined 700+ categories.
    - **Multi-Language**
        - Enables you to easily analyze text in multiple languages including English, Spanish, Japanese, Chinese (Simplified and Traditional), French, German, Italian, Korean and Portuguese.
    - **Integrated REST API**
        - Access via REST API. Text can be uploaded in the request or integrated with Cloud Storage.