Until now

- Abstractions for writing and deploying large-scale web applications
  - Managing infrastructure (PaaS, IaaS, Infrastructure-as-Code, FaaS, etc.)
  - Constructing applications (ML APIs, Backend-as-a-Service)
But, cloud is not all front-facing apps

- "Big Computation"
  - Particle physics simulations
  - Genomic searching/matching
- "Big Data"
  - Turning data into actionable knowledge
  - User, application analytics for targeted advertising and usage prediction
  - Business analytics for supply-chain and market price prediction
  - Medical informatics for research
- Sometimes both…
  - Machine learning applications (e.g. prior ML APIs)
Data Science

- Computing, managing and analyzing large-scale data
  - Requires new programming models, algorithms, data structures, and storage/processing systems
  - e.g. new abstractions!
- Some selected topics…
  - Data Warehouses, Data Notebooks
  - Data Processing, Machine Learning
Data Warehouses

Google BigQuery
AWS Redshift
Azure Data Lake
Motivation

- What if you want unlimited capacity while supporting fast querying?
  - Small-ish transactional in-memory databases support fast queries, but do not scale (SQL, MySQL etc.)
  - Large file systems support large size, but can not (natively) support querying (GCS, S3)
  - NoSQL data store massive datasets via distributed hash-table, but also difficult to query efficiently (i.e. puts and gets)
Data warehouses

• Storage for large datasets organized for write once, read/query many access
• Does not require transactional properties of On-line Transaction Processing (OLTP)
  • e.g. No need for ACID as SQL/Spanner support
• Good for On-line Analytical Processing (OLAP) apps
  • e.g. Log processing for site/app analytics
• Can be implemented via cheap disks and slower CPUs
BigQuery
From last weekend...

"Google’s differentiation factor lies in its deep investments in analytics and ML. Many customers who choose Google for strategic adoption have applications that are anchored by BigQuery."

- Gartner's Magic Quadrant report on public cloud services

- CS 410/510: Cloud and Cluster Management
BigQuery

- Fully managed, no-ops data warehouse
  - Developed by Google when MapReduce on 24 hours of logs took 24 hours to execute
- Fast, streaming data storage
  - 100k rows per second, hundreds of TB
- High-performance querying via SQL-like query interface
  - Near real-time analysis of massive datasets via replication and parallelism
- Allows one to bring code to where data is (in the cloud)
  - Key in broadband-limited places

- How?
Column-oriented storage

- Previously, logs stored in a flat file (row-based storage)
  - Recall TCP lab
    - Parsing libpcap trace file to obtain \texttt{cwnd} value over time
    - Entire pcap file file loaded and parsed to generate result
    - All data touched to access \texttt{cwnd} column in line
- Split columns into separate contiguously stored files for performance
  - Reduces data accesses for column-oriented queries
  - Common access pattern for data analytics
- Achieve better compression
  - Grouping of similar data types in columns
- Parallelizable via fast replication
  - Only common columns needed in queries replicated
Serverless querying

- Queries spawn off computing and storage resources to execute
  - Up to 2,000 nodes/shards if available
  - Done over a petabit network in backend data center
- Pay per query with minimal cost to store data
  - < $0.02 per GB stored per month (first TB free)
  - But, $5 per TB processed
    - Do NOT do a “SELECT *”
    - Do a dry run or preview first!
Architecture

- Columnar data replicated automatically (via Colossus, successor to Google Filesystem)
- Computation scaled automatically (via Borg)
- Horizontal scaling via cheap CPUs and disks
  - Allows system to approach performance of in-memory datastores

![Diagram showing Borg - Cluster management system and Dremel query engine]

- **Dremel query engine**
  - Dynamic, shared serving tree
    - Number of shards and levels based on query needs
  - High speed, petabit network
  - Distributed, columnar storage
BigQuery demo

- Run a query after doing a preview showing how much data will be accessed

```sql
SELECT name, sum(number) as name_count
FROM [bigquery-public-data:usa_names.usa_1910_2013]
WHERE gender='F'
GROUP BY name
ORDER BY name_count DESC
LIMIT 10
```

```sql
SELECT language, SUM(views) as views
FROM [bigquery-samples:wikipedia_benchmark.Wiki10B] // 10 b rows
WHERE regexp_match(title,"Goog.*")
GROUP BY language
ORDER BY views DESC
```

- Cached results are free
- Check timing
BigQuery demo

- Larger query (Preview only. DO NOT RUN)

```
SELECT language, SUM(views) as views
  FROM [bigquery-samples:wikipedia_benchmark.Wiki100B] // 100 b rows
  WHERE regexp_match(title,"G.*o.*o.*g")
  GROUP BY language
  ORDER BY views DESC
```
Public datasets on BigQuery

- QuickDraw with Google
  - 50 million drawings
  - [https://quickdraw.withgoogle.com/data](https://quickdraw.withgoogle.com/data)
- Github
  - Find out whether programmers prefer tabs or spaces
- NYC public data
  - Find out which neighborhoods have the most car thefts
  - Find out which neighborhoods have issues with rat infestation (311 calls on rats)
- NOAA ICODE ship data from 1662
  - Find ships nearby when Titanic sank
Data Notebooks

iPython, Jupyter
Google Cloud Datalab
Data notebooks

• Interactive authoring tool
  • Helps document data exploration, transformation, analysis, and visualization tasks
  • Combine program code (Python) with rich document elements (text, figures, equations, links)
  • e.g. Like a Google Doc that can execute code
• Data products and artifacts along with code that generated them
  • Disseminate results in a reproducible manner!
Data notebooks

- Initially iPython (interactive Python)
- Now Jupyter
  - Server-based
    - Interpreter runs on server, wrapped in HTML
    - Contains all packages and data for producing artifacts within code
  - Implements GUI for adding elements (e.g. Markdown) and code (e.g. Python)
  - Supports other languages other than Python (e.g. Javascript, Ruby)
Installing Jupyter locally

virtualenv -p python3 env
source env/bin/activate
pip install jupyter
jupyter notebook

• Launches a web server that hosts the interactive notebook as a web app
• Visit URL in browser
Google Cloud Datalab

- Hosted Jupyter instance
  - For analyzing data in the cloud
  - Avoid downloading data
  - Avoid installing all of GCP libraries
- Service automatically spins up a Jupyter instance on a Compute Engine VM
  - Access to BigQuery or Cloud Storage
  - Access to services such as Machine Learning Engine
Labs
BigQuery Lab #1

- Create datasets and run queries on BigQuery (25 min)
- Launch Cloud Shell
- List the APIs to see the range of services available
  ```sh
gcloud services list --available
  ```
  - To enable a service like the Cloud Datastore API, the command would be
  ```sh
gcloud services enable datastore.googleapis.com
  ```
  - From the list, enable the BigQuery API
- Go to console, and menu of services
- **BigQuery**
  - Click on drop-down next to project name and create dataset
- For Dataset ID, type cp100
Copy file from bucket into Cloud Shell and take a look

```
gsutil cp gs://cloud-training/CP100/Lab12/yob2014.txt .
head -3 yob2014.txt
wc -l yob2014.txt
```

```
wuchangfeng@lateral-array-175417:~$ gsutil cp gs://cloud-training/CP100/Lab12/yob2014.txt .
Copying gs://cloud-training/CP100/Lab12/yob2014.txt...
/ [1 files][415.5 KiB/415.5 KiB]
Operation completed over 1 objects/415.5 KiB.
wuchangfeng@lateral-array-175417:~$ head -3 yob2014.txt
Emma,F,20799
Olivia,F,19674
Sophia,F,18490
wuchangfeng@lateral-array-175417:~$ wc -l yob2014.txt
33044 yob2014.txt
wuchangfeng@lateral-array-175417:~$
```
- Create table from file in bucket
  - Specify input file location and format (CSV)
  - Specify table name (namedata), table type (native) and schema columns and types
  - Edit schema to add fields for name and gender as STRING, count as INTEGER
  - Field delimiter as a Comma, then Create Table
- Click table and **Preview**, show the number of rows in **Details**
3 ways to query

- Via UI
  - Click on "Query Table"
  - Run a query that lists the 20 most popular female names in 2014
    - Click on Validator to see how much data you will hit before running

```
SELECT name, COUNT
FROM [lateral-array-175417:cp100.namedata]
WHERE gender = 'F'
ORDER BY COUNT DESC
LIMIT 5
```

Valid: This query will process ... when run.
• Via command-line in Cloud Shell
  • Run query to get the 20 least popular boys names in 2014

```
wuchangfeng@lateral-array-175417:~$ bq query "SELECT name, count FROM [lateral-array-175417:cp100.namedata] WHERE gender = 'F' ORDER BY count DESC LIMIT 5"
Waiting on bqjob_r44c73adf90e286eb_0000015e59995e17_1 ... (0s) Current status: DONE
+-----------------+--------+
| name            | count  |
+-----------------+--------+
| Emma            | 20799  |
| Olivia          | 19674  |
| Sophia          | 18490  |
| Isabella        | 16950  |
| Ava             | 15586  |
+-----------------+--------+
wuchangfeng@lateral-array-175417:~$  
```
- Via BigQuery shell (`bq shell`)
- Run a query to find 20 most popular male names in 2014
BigQuery Lab #1

- Keep project
- Create datasets and run queries on BigQuery
  - [https://codelabs.developers.google.com/codelabs/cp100-big-query/](https://codelabs.developers.google.com/codelabs/cp100-big-query/) (25 min)
BigQuery Lab #2

- Query Github Data Using BigQuery (8 min)
- (Extra: not in Codelab) Find public dataset containing all of the blocks and transactions on the Bitcoin blockchain
  - Click on Preview to find the number of blocks that are currently being stored on a full node.
  - Click on Details to find the size of the block-chain in BigQuery (uncompressed).
• Visit dataset containing all github commits
  • https://bigquery.cloud.google.com/table/bigquery-public-data:github_repos.commits
  • Click on Preview and examine the columns associated with commits
  • Click on Details to find the size of the commits table
- Go to console, and open a BigQuery window
- Click on "Compose Query"

- Click Show Options

- Unclick Legacy SQL (to use standard SQL)
- Enter a query to find commits with duplicate subject lines (commit messages)

```sql
#standardSQL
SELECT subject AS subject, COUNT(*) AS num_duplicates
FROM `bigquery-public-data.github_repos.commits`
GROUP BY subject
ORDER BY num_duplicates DESC
LIMIT 100
```

- Open the validator and show the amount of data that will be processed in query if executed

- Run the query to find commits with duplicate subject lines (commit messages)
  - What is the most common subject message used?
  - Show quickly the query runs
Run query to find projects with the most contributors

- Extract name of repo from `repo_name` path

```sql
#standardSQL
SELECT
  COUNT(DISTINCT author.email) AS num_authors,
  REGEXP_EXTRACT(repo_name[ORDINAL(1)], r"([^/]+)$") AS repo
FROM `bigquery-public-data.github_repos.commits`
GROUP BY repo
ORDER BY num_authors DESC
LIMIT 1000
```

Run query to find most popular languages used in commits

```sql
#standardSQL
SELECT
  COUNT(*) pr_count, JSON_EXTRACT_SCALAR(payload, '$.pull_request.base.repo.language') lang
FROM `githubarchive.month.201801`
WHERE JSON_EXTRACT_SCALAR(payload, '$.pull_request.base.repo.language') IS NOT NULL
GROUP BY lang
ORDER BY pr_count DESC
LIMIT 10
```
BigQuery Lab #2

• Query Github Data Using BigQuery (8 min)
  • [https://codelabs.developers.google.com/codelabs/bigquery-github](https://codelabs.developers.google.com/codelabs/bigquery-github)
BigQuery Lab #3

- Looking at campaign finance with BigQuery (14 min)
  - First 8 steps
- Skip step 2 (should already be done)
- Create a dataset via `bq` command-line interface

```
DATASET=campaign_funding
bq mk -d ${DATASET}
```

- Source of campaign finance data and its format are at
- Copy uncompressed version from a GCS bucket and examine the last several entries with `tail`

```
gsutil cp gs://campaign-funding/indiv16.txt .
tail indiv16.txt
```
BigQuery Lab #3

• Use `du` and `wc` to find out how large the file is and how many individual contributions were made
• Contribution data definitions by individuals (`indiv16.txt`), by committees, and by candidates available at
• We will be linking a BigQuery table with these definitions to the downloaded files stored in GCS
Create a BigQuery definition specifying CSV data from the bucket location via command-line and obtain data definition JSON output

```
bq mkdef --source_format=CSV gs://campaign-funding/indiv*.txt "CMTE_ID, AMNDT_IND, RPT_TP, TRANSACTION_PGI, IMAGE_NUM, TRANSACTION_TP, ENTITY_TP, NAME, CITY, STATE, ZIP_CODE, EMPLOYER, OCCUPATION, TRANSACTION_DT, TRANSACTION_AMT:FLOAT, OTHER_ID, TRAN_ID, FILE_NUM, MEMO_CD, MEMO_TEXT, SUB_ID" > indiv_def.json
```

Note that file is not actually in CSV format
- Data separated by pipe character '|
- In Line #6, change `fieldDelimiter` to indicate this
- Or run…

```
sed -i 's/"fieldDelimiter": ","/"fieldDelimiter": "|"/g; s/"quote": "\\\""/"quote":"\"/g' indiv_def.json
```
• Copy similarly modified definition files for committee and candidate data

```sh
gsutil cp gs://campaign-funding/candidate_def.json .
gsutil cp gs://campaign-funding/committee_def.json .
```

• Create BigQuery tables with the definitions

```sh
bq mk --external_table_definition=indiv_def.json -t ${DATASET}.transactions
bq mk --external_table_definition=committee_def.json -t ${DATASET}.committees
bq mk --external_table_definition=candidate_def.json -t ${DATASET}.candidates
```

• Note that because BigQuery tables are linked to flat files, queries will not perform well for large data
• Goto BigQuery UI and run a simple query

```sql
SELECT * FROM [campaign_funding.transactions]
WHERE EMPLOYER contains "GOOGLE"
ORDER BY TRANSACTION_DT DESC
LIMIT 100
```

• Note that because we pointed BigQuery to files in a storage bucket, the validator will not be able to estimate the amount of data that will be processed for the query
Then run the following query to obtain party-based contributions for those with an engineering occupation.

```sql
SELECT affiliation, SUM(amount) AS amount
FROM ( SELECT * 
    FROM ( SELECT t.amt AS amount,
                  t.occupation AS occupation,
                  c.affiliation AS affiliation,
        FROM ( SELECT trans.TRANSACTION_AMT AS amt,
                  trans.OCCUPATION AS occupation,
                  cmte.CAND_ID AS CAND_ID 
        FROM [campaign_funding.transactions] trans
        RIGHT OUTER JOIN EACH ( 
            SELECT CMTE_ID, FIRST(CAND_ID) AS CAND_ID
            FROM [campaign_funding.committees] 
            GROUP EACH BY CMTE_ID) cmte
        ON trans.CMTE_ID = cmte.CMTE_ID) AS t
    RIGHT OUTER JOIN EACH ( 
        SELECT CAND_ID, 
            FIRST(CAND_PTY_AFFILIATION) AS affiliation,
        FROM [campaign_funding.candidates] 
        GROUP EACH BY CAND_ID) c
        ON t.CAND_ID = c.CAND_ID)
WHERE occupation CONTAINS "ENGINEER"
GROUP BY affiliation
ORDER BY amount DESC
```
• Query needs to join with committees table (Republican/Democratic) and candidates table to associate candidate to party for individual contribution
• Repeat previous query on any other profession besides Engineer to find
  • A profession that has more Republican contributions than Democratic
  • A profession that has more Democratic contributions than Republican
BigQuery Lab #3

- Looking at campaign finance with BigQuery (14 min)
  - First 8 steps
  - https://codelabs.developers.google.com/codelabs/cloud-bq-campaign-finance
Cloud Datalab Lab #1

- Analyzing data using Datalab and BigQuery (11 min)
- Launch Cloud Datalab docker container onto a VM instance nearby
  - Go to next step while waiting (takes > 5 min)

```bash
datalab create mydatalabvm --zone us-west1-b
```
• Run standard SQL query to list delayed departures

```sql
SELECT
    departure_delay, COUNT(1) AS num_flights,
    APPROX_QUANTILES(arrival_delay, 4) AS arrival_delay_quantiles
FROM
    'bigquery-samples.airline_onetime_data.flights`
GROUP BY
    departure_delay
HAVING
    num_flights > 100
ORDER BY
    departure_delay ASC
```

• Run query to find 20 most popular flights

```sql
SELECT
    departure_airport, arrival_airport, COUNT(1) AS num_flights
FROM
    'bigquery-samples.airline_onetime_data.flights`
GROUP BY
    departure_airport, arrival_airport
ORDER BY
    num_flights DESC
LIMIT 20
```
• Go back to Cloud Shell that launched Cloud Datalab
• Go to Web Preview of shell, change port to 8081, and preview to pull up Cloud Datalab UI

• Start a new notebook called 'flights'
• Paste Python code into notebook cell and run it
• Note that \texttt{df} is a pandas data-frame
• Get count of flight departure delays and their associated arrival delays, then run

```
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()
```
• Append a new code cell to notebook

```python
import pandas as pd
percentiles = df['arrival_delay_deciles'].apply(pd.Series)
percentiles = percentiles.rename(columns = lambda x : str(x*10) + ' %')
df = pd.concat([df['departure_delay'], percentiles], axis=1)
df.head()
```

• Paste Python code to create deciles on arrivals in next notebook cell and run it

• Paste Python code to plot delays into next notebook cell and run it

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

• Show the plot
Cloud Datalab Lab #1

- Skip Step #5
- Analyzing data using Datalab and BigQuery (11 min)
- Link
Cloud Datalab #2

- Image Classification Using Cloud ML Engine & Datalab (30 min)
- Steps through work flow of an ML data-scientist
  - Other notebooks included in samples directory
  - Codelabs for other scientific computing notebooks in next lecture
- Start at Step #4 (use previous Datalab instance)
- Plot the initial graph and render some Markdown

```python
import matplotlib.pyplot as pl
pl.plot([1,2,3,4,5], [1,4,9,16,25])
pl.axis([0,6,0,30])
pl.show()
```

```
# Hello
This is some markdown
```

```
Hello
This is some markdown
```
In Cloud Datalab, click on Home icon, then navigate to:

datalab/docs/samples/ML Toolbox/Image Classification/Flower

Click on:

Local End to End.ipynb

In notebook, clear all cells:

The notebook will take a pre-trained model, then allow you to apply transfer learning to modify the model with your own flower images.

- Performs typical steps in an ML workflow (preprocessing data, training, prediction, and evaluation)
Individually select code cells and click Run
- Download and store image information in CSV files and the images themselves from GCS to Datalab VM
- Go back to Cloud Datalab to see files
  - `/content` places you at the root of the notebook
- Run the Cloud Dataflow pipeline to prepare the images and run it through pre-trained model in TensorFlow
- Then evaluate the new model, put results into BigQuery and analyze
- Stop TensorBoard and delete BigQuery tables at the end
Taffy
Extra
Cloud Dataprep
Cloud Dataprep

• Problem
  • Data in the real-world often "dirty"
    • Incomplete
    • Error-ridden
    • Malformatted
  • Estimated 60-70% of time in data science tasks spent on cleaning data
  • Attempt to automate and to apply machine learning to clean data