Product Recommendations using Cloud SQL and Spark

Lak Lakshmanan
Tech Lead, Big Data & ML

Hi, I'm Lak. Welcome to the third module of our Big Data and ML Fundamentals course. This module is on performing Product Recommendations using Cloud SQL and Spark.

We'll cover product recommendations which is probably the most common ML problem in businesses.
Along the way, we'll also get to look at two Google Cloud products: Cloud SQL which is a managed relational database and Cloud Dataproc which is a managed cluster on which you can run Spark.

Managed? What's managed?
Stick around!

Agenda

Recommendation systems

- Business applications
- Scenario: ML for housing rentals

Choosing the right solution approach

- On-premise to Google Cloud Platform
- Challenge: Utilizing and tuning on-premise clusters
- Off-cluster storage with Google Cloud Storage

We'll start out talking about recommendation systems.

Agenda

Recommendation systems

- Business applications
- Scenario: ML for housing rentals

Choosing the right solution approach

- On-premise to Google Cloud Platform
- Challenge: Utilizing and tuning on-premise clusters
- Off-cluster storage with Google Cloud Storage

What they are, what business applications they power, and consider a typical scenario.

Agenda

Recommendation systems

- Business applications
- Scenario: ML for housing rentals

Choosing the right solution approach

- On-premise to Google Cloud Platform
- Challenge: Utilizing and tuning on-premise clusters
- Off-cluster storage with Google Cloud Storage

In this case, that is product recommendations, using machine learning for recommending rental houses.

Let’s say that our team is familiar with Hadoop and Spark and has already built recommendation system.

Agenda

Recommendation systems

- Business applications
- Scenario: ML for housing rentals

Choosing the right solution approach

- On-premise to Google Cloud Platform
- Challenge: Utilizing and tuning on-premise clusters
- Off-cluster storage with Google Cloud Storage

Our mission, which we have chosen to accept, is to migrate this existing Recommendation system our team has built on-premise to the cloud.

Agenda

Recommendation systems

- Business applications
- Scenario: ML for housing rentals

Choosing the right solution approach

- On-premise to Google Cloud Platform
- Challenge: Utilizing and tuning on-premise clusters
- Off-cluster storage with Google Cloud Storage

We will look at how you would go about migrating on-premises applications to Google Cloud Platform. In particular, why you would want to do that.

Agenda

Recommendation systems

- Business applications
- Scenario: ML for housing rentals

Choosing the right solution approach

- On-premise to Google Cloud Platform
- Challenge: Utilizing and tuning on-premise clusters
- Off-cluster storage with Google Cloud Storage

Maybe to avoid the challenges associated with utilizing and tuning on-premise clusters and

Agenda

Recommendation systems

- Business applications
- Scenario: ML for housing rentals

Choosing the right solution approach

- On-premise to Google Cloud Platform
- Challenge: Utilizing and tuning on-premise clusters
- Off-cluster storage with Google Cloud Storage

why you'd move towards off-cluster storage with Google Cloud Storage.

Agenda

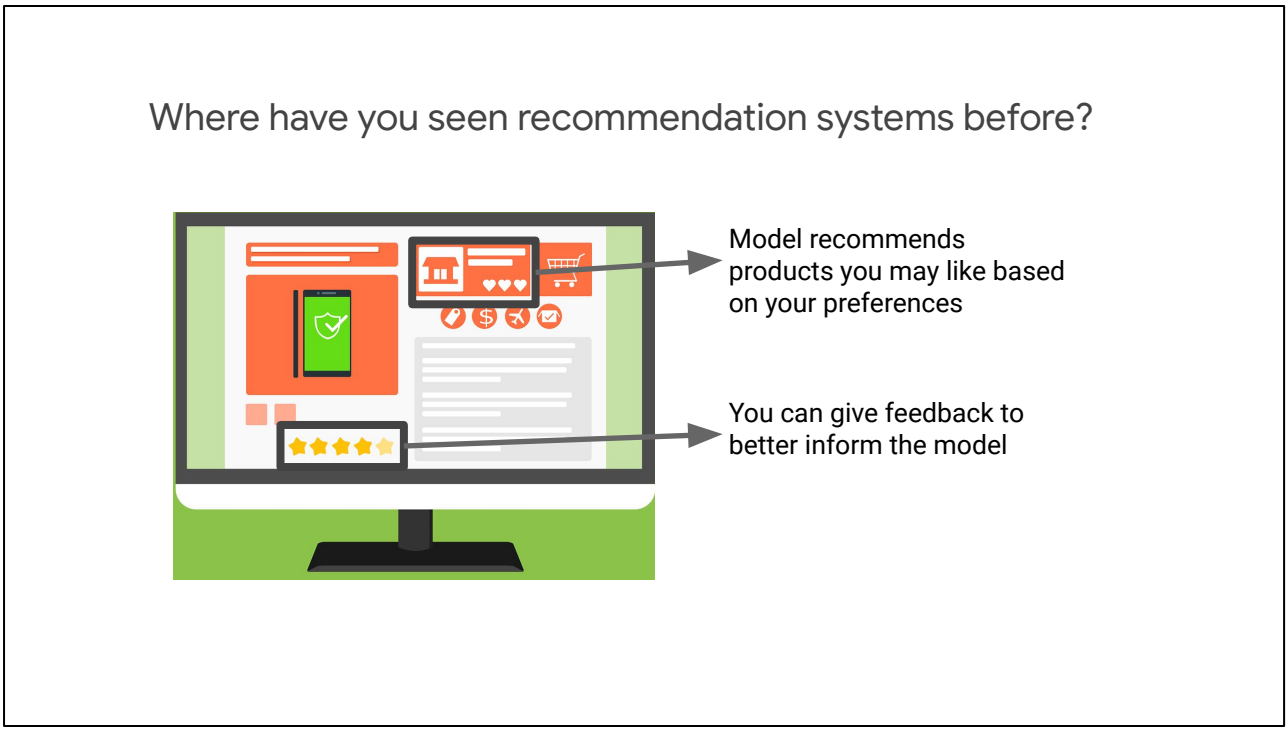
Recommendation systems

- Business applications
- Scenario: ML for housing rentals

Choosing the right solution approach

- On-premise to Google Cloud Platform
- Challenge: Utilizing and tuning on-premise clusters
- Off-cluster storage with Google Cloud Storage

Let’s start with where you find recommendation systems adding value to businesses.



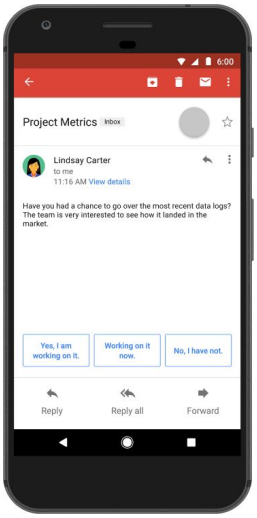
Where have you seen recommendation systems before?

One common place that comes to mind is ecommerce product recommendations. A model learns what you like and don't like and then suggests similar products that you may not have seen before.

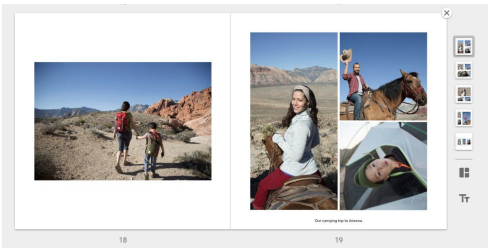
You can then help inform the model explicitly by starring, up voting or down voting, your own personal rating of an item. Or maybe by adding the item to the shopping cart. Notice how I said explicitly -- a model can learn from your preferences implicitly too. Can you think of a few data points that an ecommerce model could learn without you rating items?

Perhaps time spent on the website on particular pages, items you have last viewed, where you navigated from, what device you are on, and -- if you've allowed it -- geographic personalization as well.

Recommendation systems are used in many applications



Smart Reply in Gmail recommends 3 possible answers to your emails



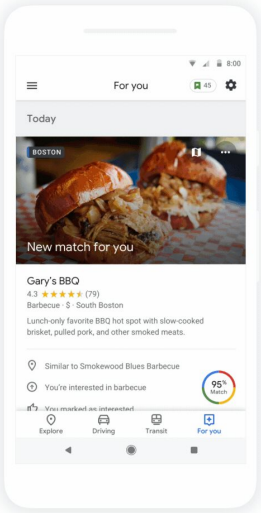
Google Photos recommends similar pictures to include in an album

You have probably seen recommendation engines in in Netflix, Amazon, and so on. It's also what Facebook uses to put posts on your newsfeed and what Google uses to personalize your search results.

Product recommendations are the first machine learning application that the public recognizes, as in "how does Netflix know I will like this movie?" The general population was pretty unaware of machine learning applications like diagnosis systems, airline fares, detecting illegal logging, playing chess, and so on that preceded the "if you like X, you will also like Y".

One cool example of a recommendation system is Smart Reply in Gmail. This is the ML model that recommends three possible answers to your emails. Another is the how Google Photos recommends groups of similar photos into an album that you can then order a printed version of.

Google Maps recommends restaurants based on what you like

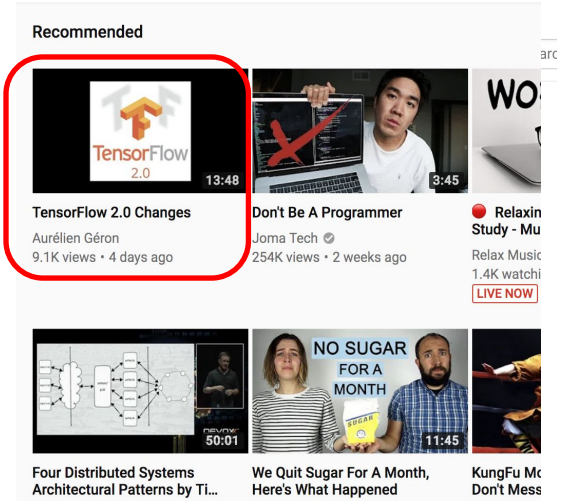


Another example is how Google Maps now serves personalized restaurant recommendations.

When you explore a new area, the app provides users with a “For you” tab which lists restaurants and venues that it thinks you will enjoy based on your history. Fo example, here you can see that good BBQ is a 95% match for me as well as a recommendation for a new bistro that is trending.

https://techcrunch.com/2018/06/26/the-new-google-maps-with-personalized-recommendations-is-now-live/

Recommendation systems must scale to meet demand



A core aspect of a recommendation system is that you need to train and serve it at scale.

Think of the ML system that drives YouTube recommendations for users. It identify things that a user may like, based on what they've watched in the past and serves them like these video recommendations I got when I logged in to YouTube. Most of the recommendations here are about programming, Tensorflow, or machine learning which says a lot about the videos I like.

These recommendations are relevant and useful to me. In the following lessons, you'll implement your own recommendation system for housing rentals.

Agenda

Recommendation systems

- Business applications
- Scenario: ML for housing rentals

Choosing the right solution approach

- On-premise to Google Cloud Platform
- Challenge: Utilizing and tuning on-premise clusters
- Off-cluster storage with Google Cloud Storage

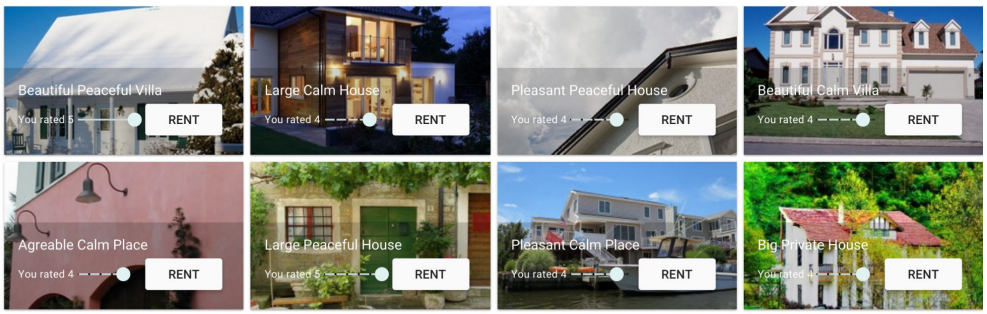
Now that you've seen how recommendation systems are used at Google and in businesses, it's time to look into the specific scenario we'll use in the next lab.

Recommendation systems require **data**, a **model**,
and training/serving **infrastructure**

The core pieces of a recommendation system are data, your model, and infrastructure to train and serve your recommendations to users.

Use case: Recommending housing rental options

Here are some housing rentals you may be interested in:



Our dataset for this scenario will be housing rentals that we want to recommend to our users based on their preferences. We will use a machine learning model to make these recommendations.

Train machine learning models on data not rules

X

```
IF house = 'beach_house'
AND season = 'summer'
AND user_pref = 'cozy'
THEN recommend = house#22
```

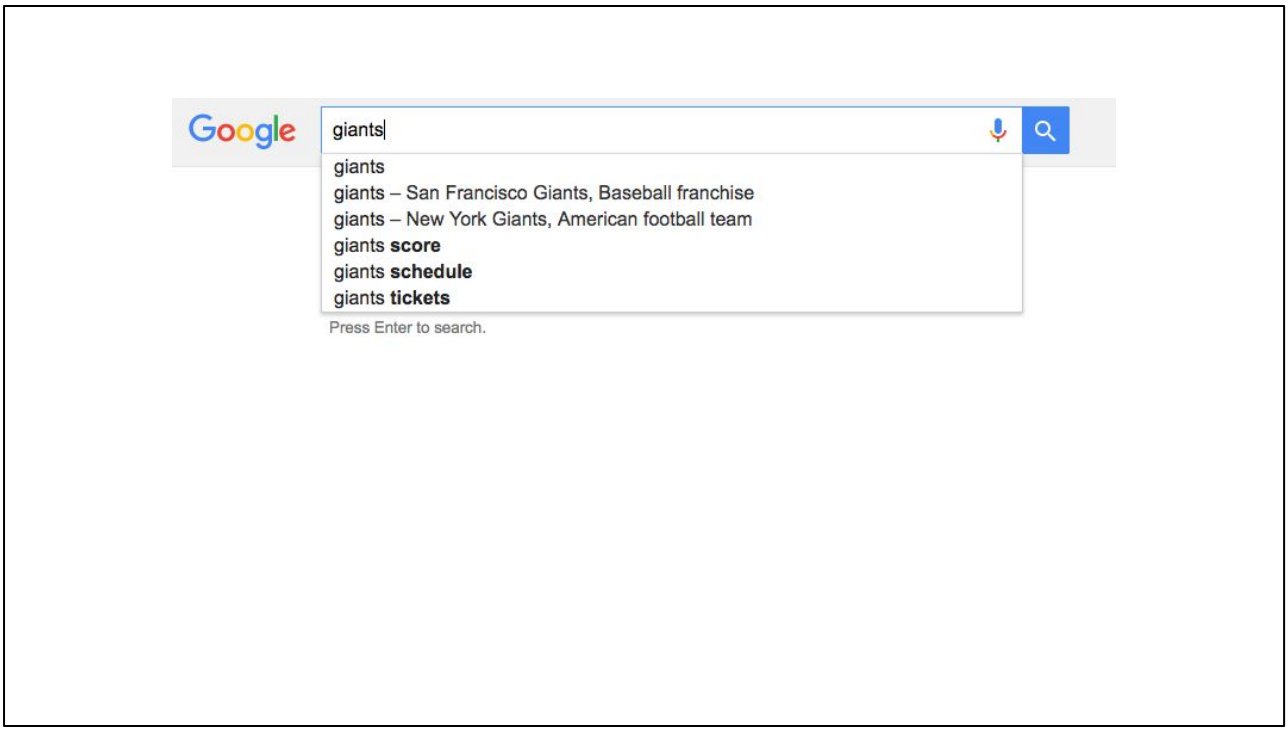
A core tenant of machine learning is to let the model learn for itself what the relationship between the data you have (like user preferences and housing features) and the data you don't have (like a user's rating on a *new* property).

You will not be writing custom logic like IF the house is a beach house and the season is summer and the user preference is for a 'cozy' house then hardcode House #22 as the recommendation.

Can you think of the issues if we used such logic instead of letting the model figure it out?

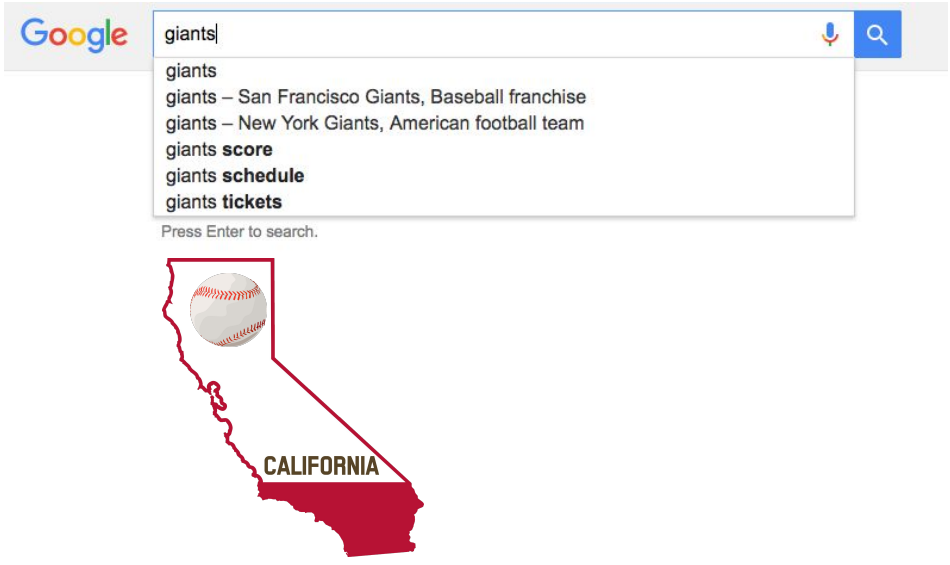
What if there were many different types of beach houses with all kinds of different features like number of bedrooms, location, amenities, you get the picture. Hardcoding logic for each of these features isn't scalable and it assumes that we always know the right answer for every scenario.

https://medium.com/airbnb-engineering/using-machine-learning-to-predict-value-of-homes-on-airbnb-9272d3d4739d
https://medium.com/airbnb-engineering/ai/home



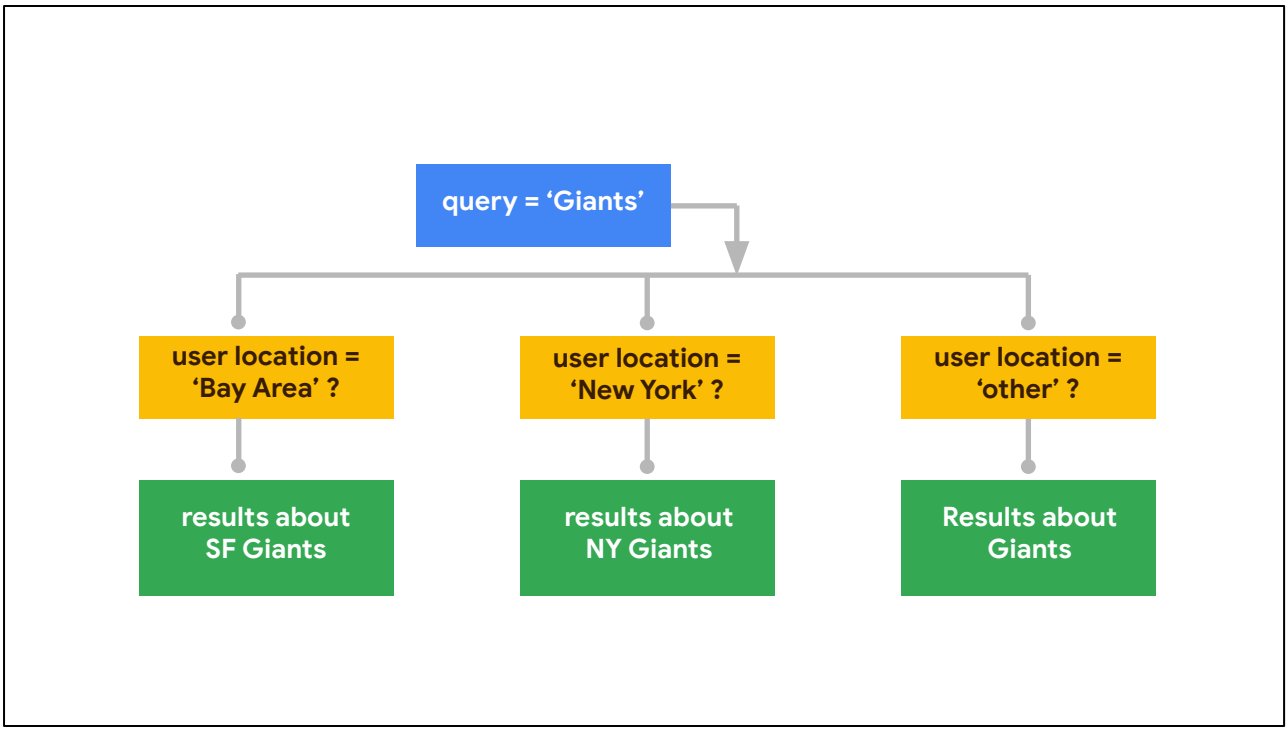
Let’s illustrate this point with an example. Take Google Search. Say you go to Google and search for “giants”

What should we show you as your results to make it most relevant for you?



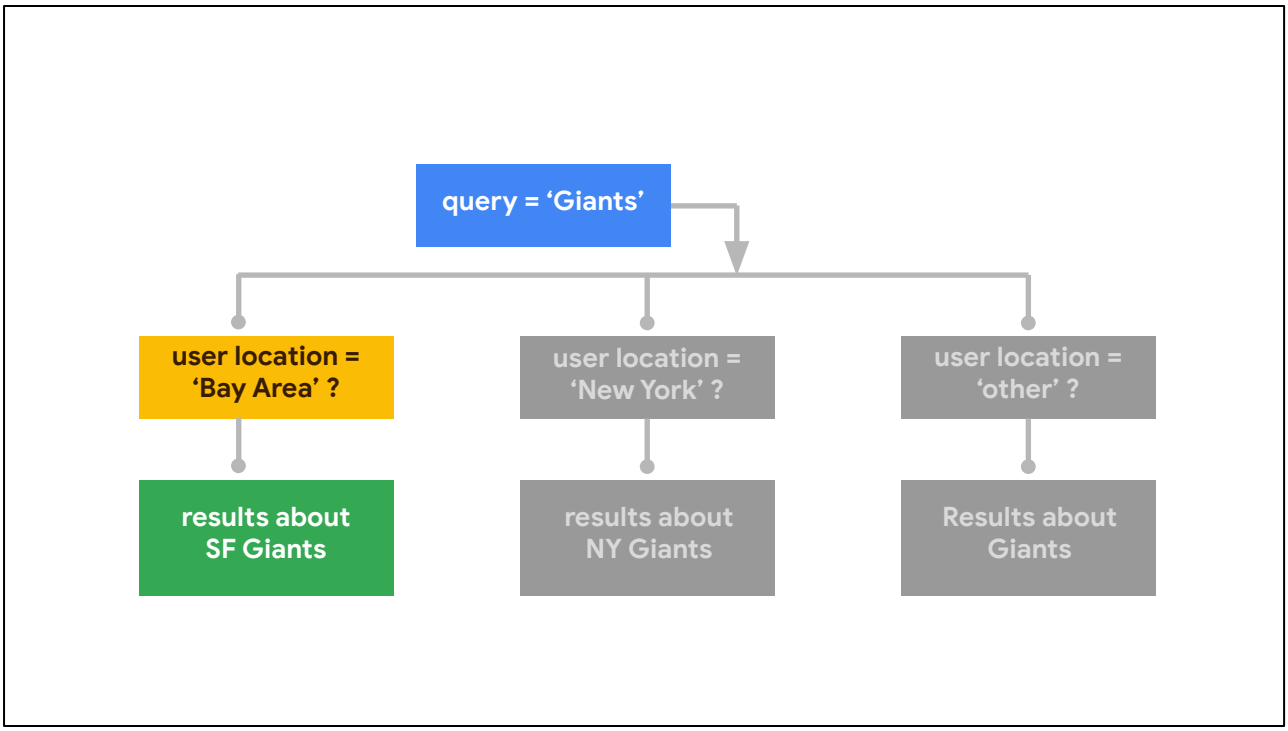
If you're in California, should we show results for the San Francisco Giants baseball team and local games nearby?

What about if you're based in New York, should we tailor the results to show the New York Giants football team instead as a rule?

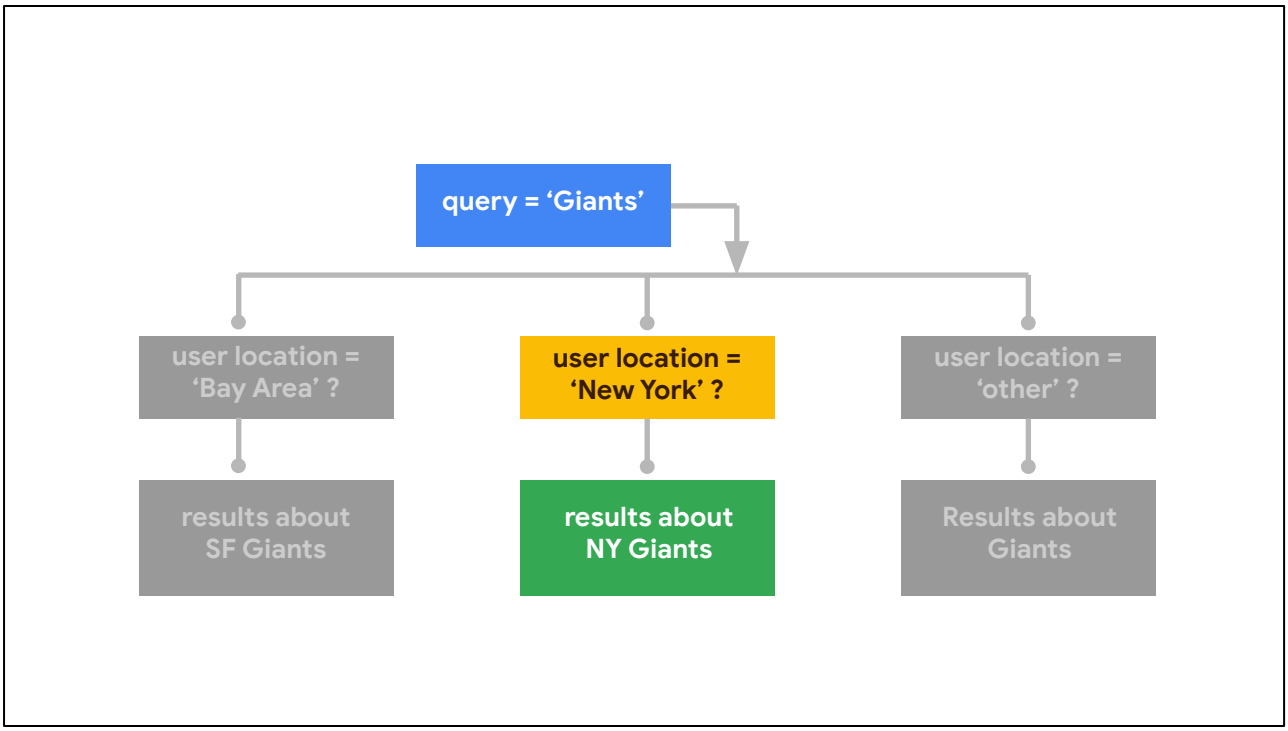


Up until a few years ago, this is how Google search worked.

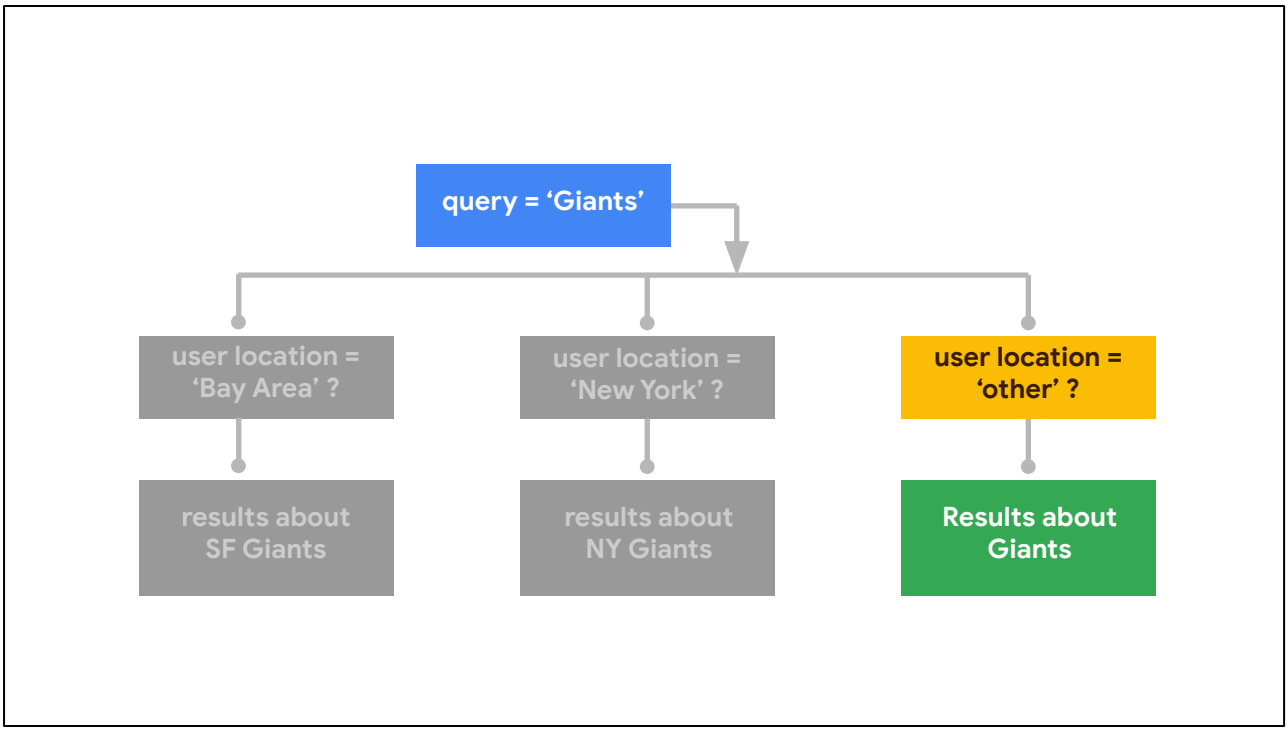
There were a ton of rules that were part of the search engine code base to decide which sports team to show a user.



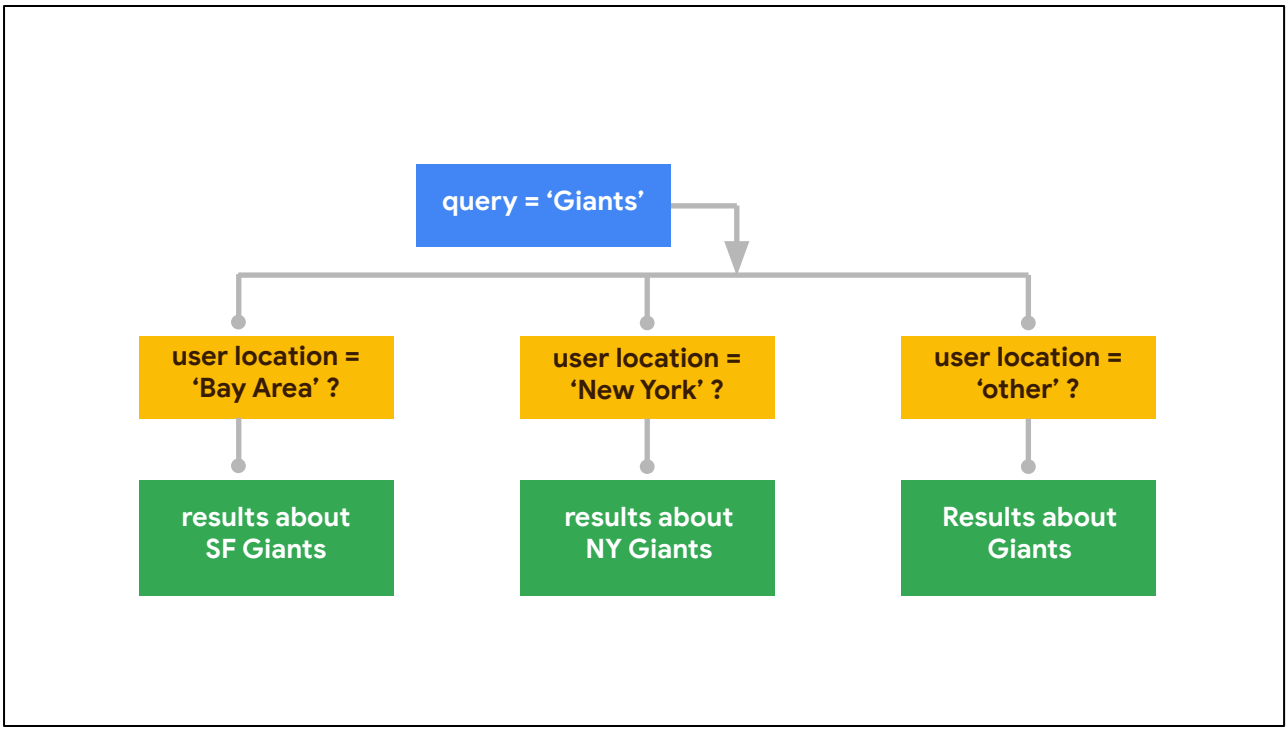
If the query is ‘giants’ AND the user is in the bay area, show them results about San Francisco Giants.



If the user is in the New York area, show them results about NY Giants



If they are anywhere else, show them results about tall people.

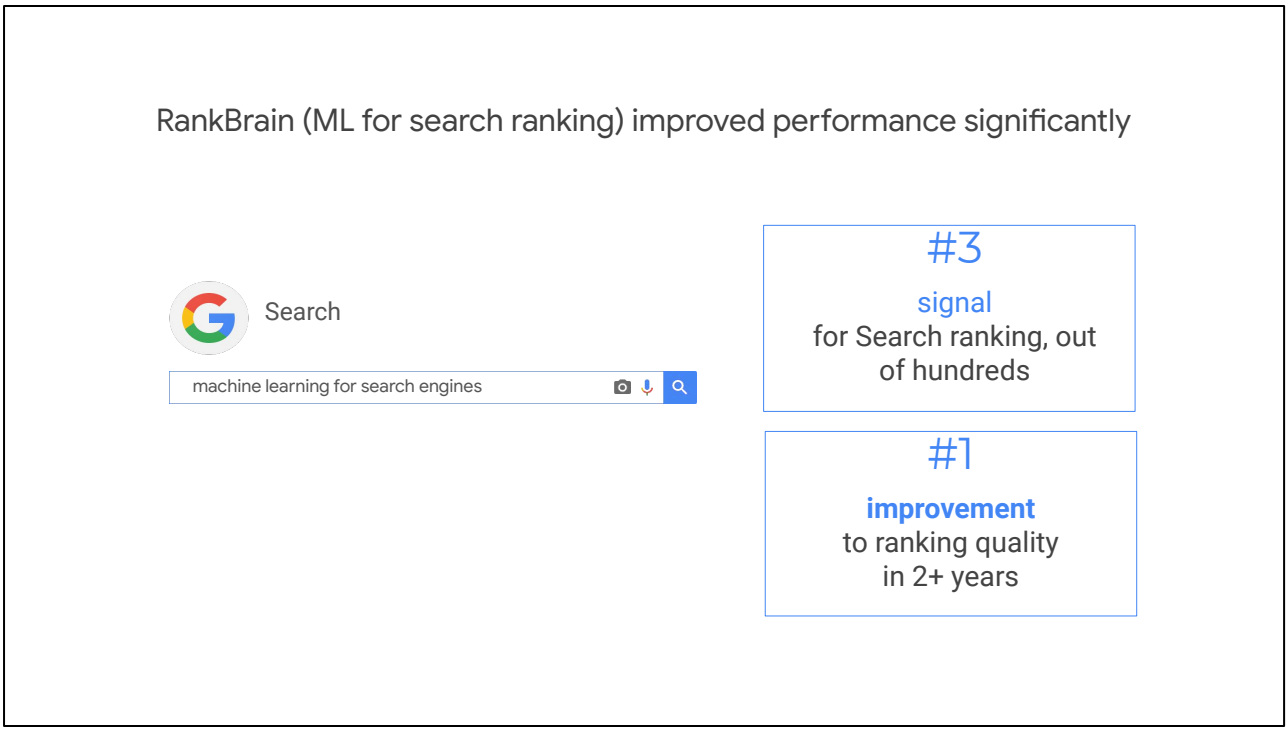


Just imagine how many IF THEN or CASE statements this would be and how hard it would be to maintain. And this is for just one query!

Multiply this by the large variety of queries that people make, where they make them from, what device they’re on, and you can imagine how complex the whole codebase had become. The codebase was getting unwieldy. Hand-coded rules are hard to maintain.

This is where ML comes into play. It scales much better because it requires no hardcoded rules and it’s all automated.

Our dataset in this case is we know historically which people clicked on what links, why couldn’t we train a ML model to provide input in the search ranking?



And that's exactly what Google itself has done internally with a deep learning ML model called RankBrain. After rolling it out, the quality of search ranking results improved dramatically with the signal coming from RankBrain becoming one of the top 3 for influencing how results are ranked

If you're interested, I'll provide a link where you can read more about it.

Machine Learning =

Examples, not rules

We will revisit machine learning in greater depth in each of the modules in this course. For now just remember that we want to teach the computer using examples, not with rules.

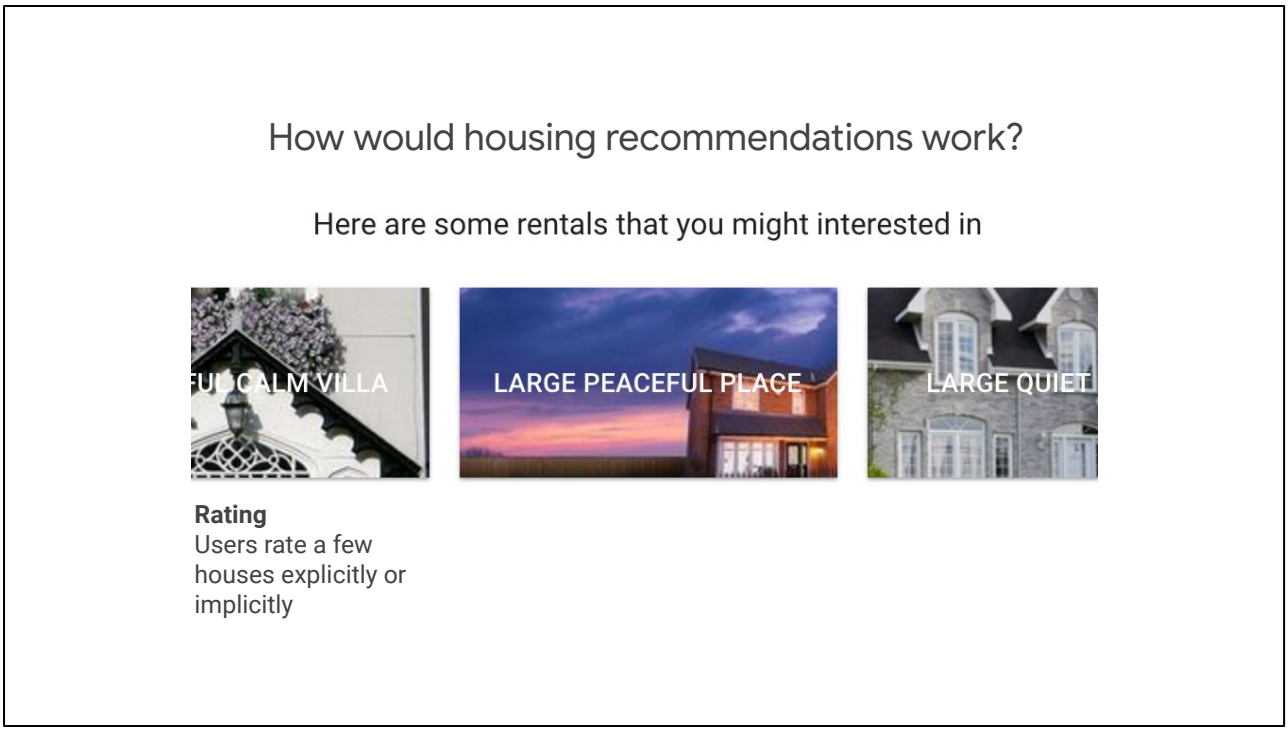
Any business application where you have those long SWITCH or CASE statements or IF THEN logic manually coded AND you have a history of good labeled data is a possible application for Machine Learning.

How would housing recommendations work?

Here are some rentals that you might interested in

So how would housing recommendations work in our system?

First we need to



ingest the ratings of all the houses in our inventory of rentals.

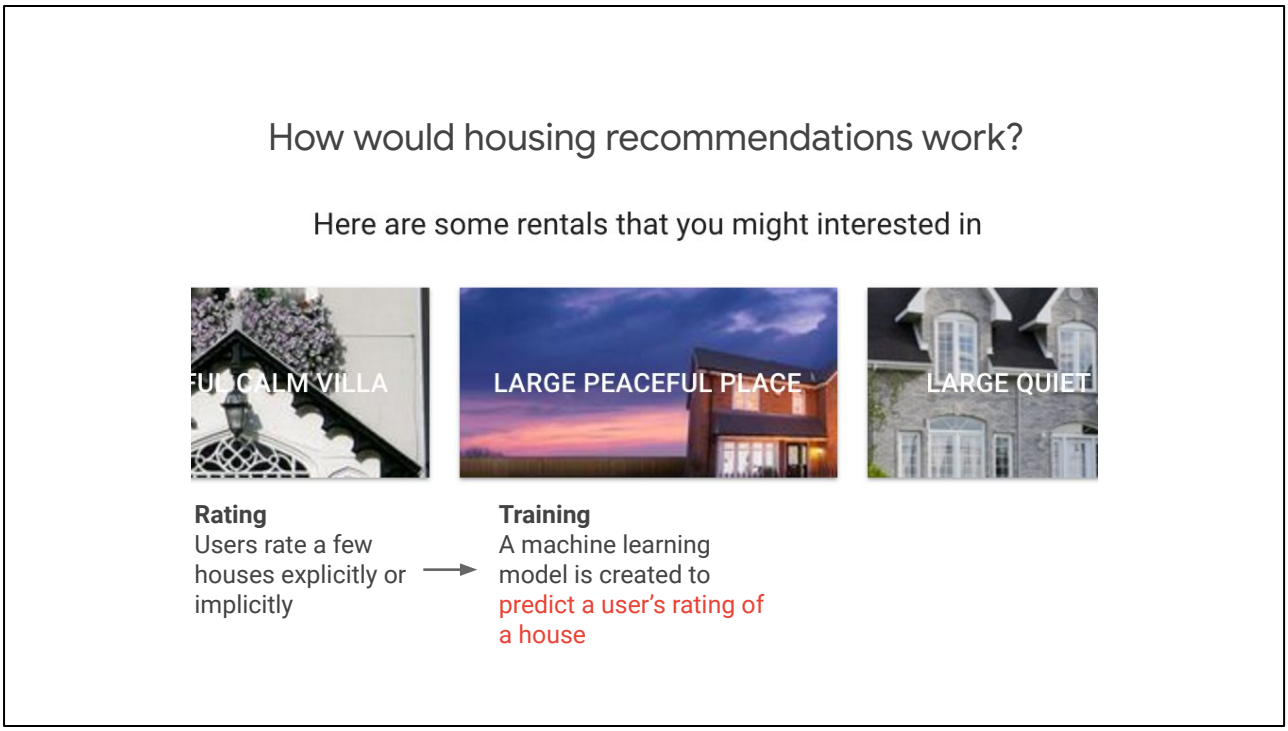
These ratings could come from explicit ratings -- maybe they actually rented the house in the past, or they clicked 4-stars after seeing the house details. Or the ratings could come from implicit ratings -- maybe they spent a lot of time looking at this property.

How would housing recommendations work?

Here are some rentals that you might interested in

Rating
Users rate a few houses explicitly or implicitly →

Then we will



train a machine learning model to predict a user's rating of *every* house. We'll then pick the top 5 best ones for them.

People who are not aware of recommendation engines somehow think of a car "appearing" on their Facebook feed because they happened to read an automobile review article. That is not the case -- reading the article caused the rating of all cars to go up (and the ratings of other items to stay relatively the same) and this caused the highest-ranking car to get into the top 5. The car was always there -- its rating was simply lower before they read the article in question.

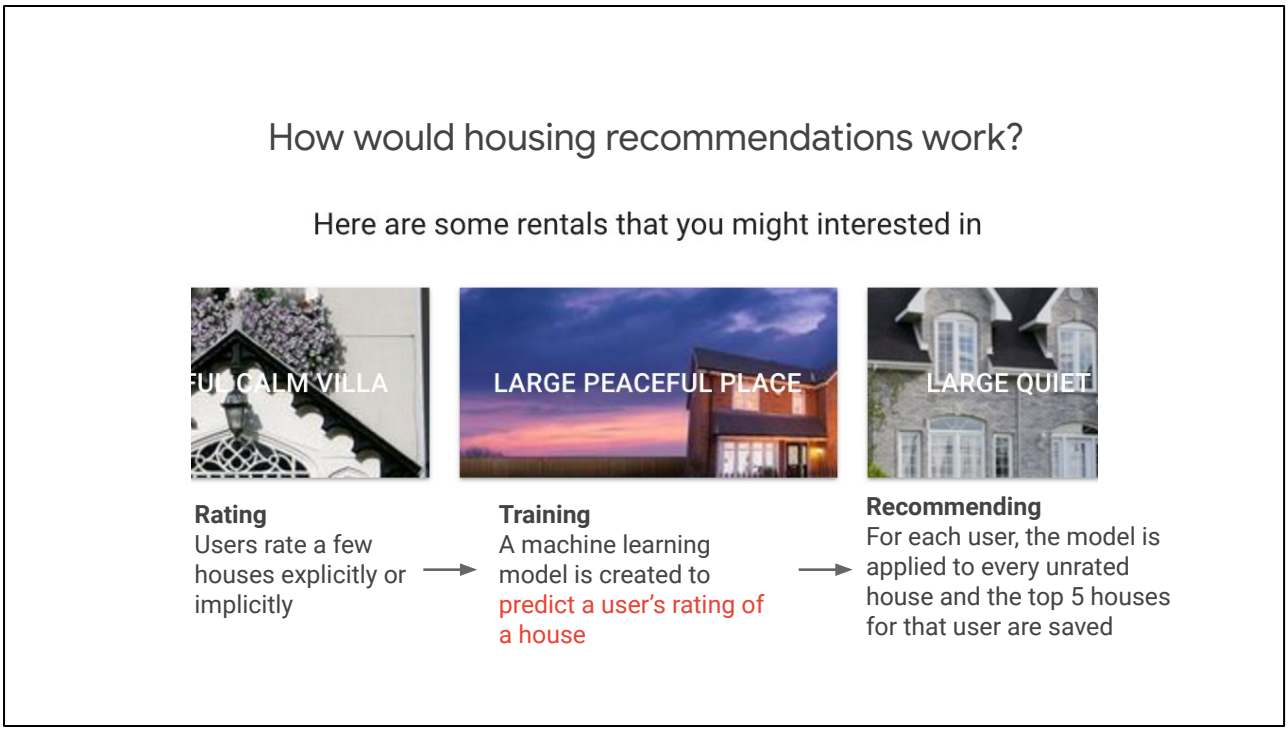
How would housing recommendations work?

Here are some rentals that you might interested in

Rating
Users rate a few houses explicitly or implicitly →

Training
A machine learning model is created to predict a user's rating of a house →

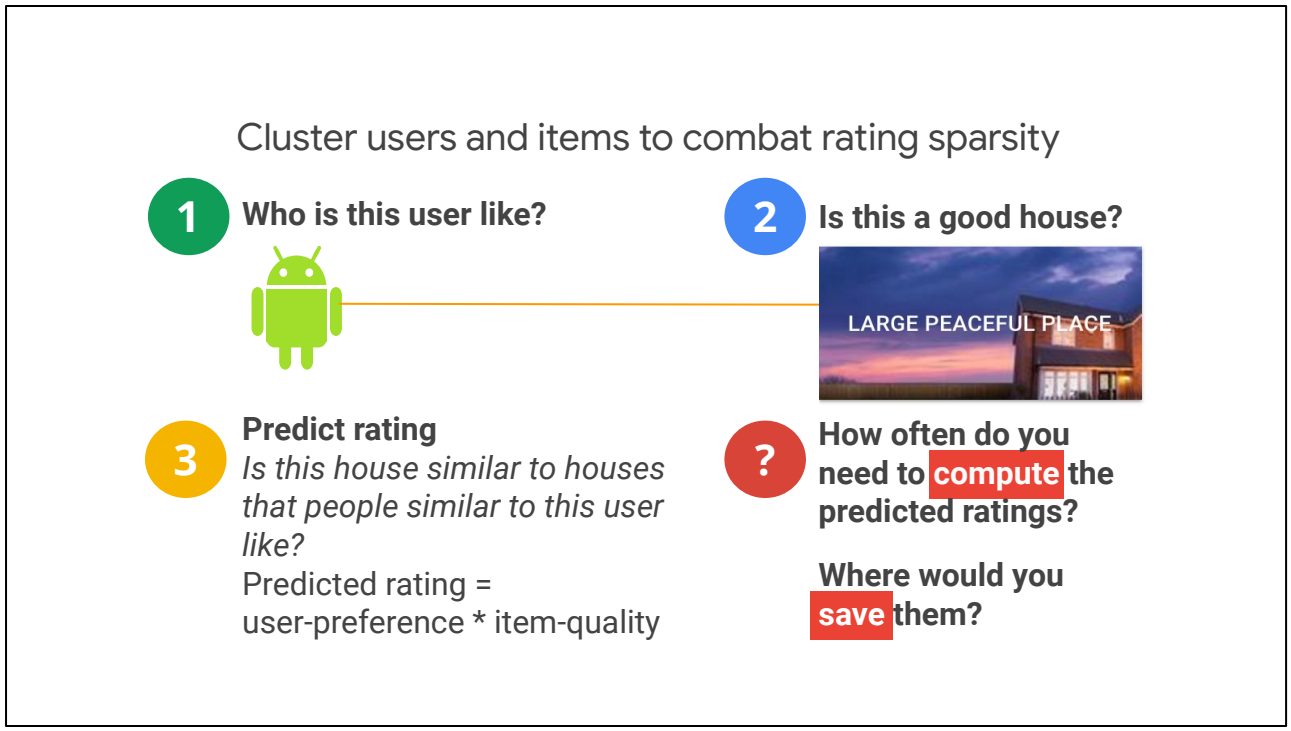
So .. how will we



predict a user's rating of a house -- particularly if they haven't seen it before?

The model is based on your other ratings and others' ratings of that house. A particularly simple model could be to look at all the users who rated that particular house, and find the 3 users in that list who are most like you. Then, the prediction is the average of those 3 users' ratings. This is not a great model of course -- it's too easy to game (think about what happens if three people gang up and rate a house and no one else bothers to review the house) -- but it helps convey the basic premise.

Where is the learning? The model would have to figure out how to find the users who are most like you (how many users to consider, and how to weight different factors such as the overall popularity of the items you have in common, and so on). This can be done by seeing what parameters help predict a few intentionally withheld ratings best.



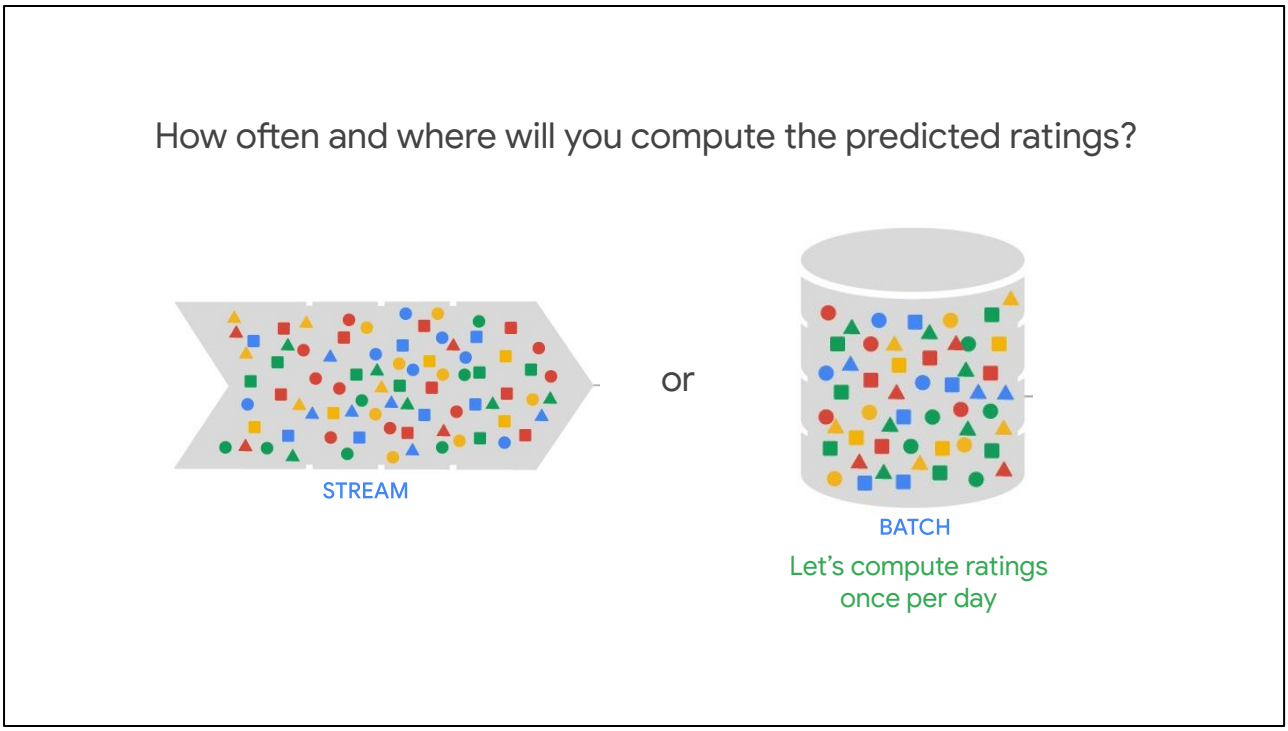
We may have 1000s of items and only 2-3 reviews per item. And chances are that those reviewers have nothing in common with the user we want the rating for. So, we need to cluster items and users together.

To put this in a more immediate form, if all your friends drive SUVs, and you read an article on Porsche, the car that appears on your feed might be a Porsche SUV even if the article was about Porsche cars and all your friends drive Toyota SUVs. The machine learning model is *imputing* a rating for a Porsche SUV that is quite high even though none of your friends rated it.

So, the ML model is essentially asking: who is this user like? Is this, objectively, a house that people tend to rate highly?
The predicted rating is a combination of both these factors. All things considered, the rating of a house for a particular user will be the average of the ratings of users like this user calibrated by the quality of the item itself.

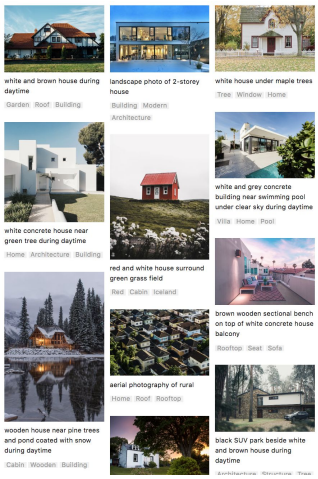
Now that we understand the problem and approach, we need to address the last question. How often and where will you *compute* the predicted ratings. And once you have them, where will you *store* them?

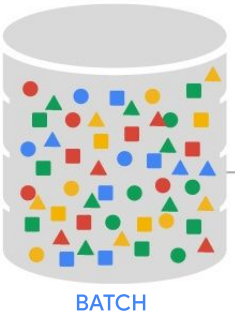
What do you think?



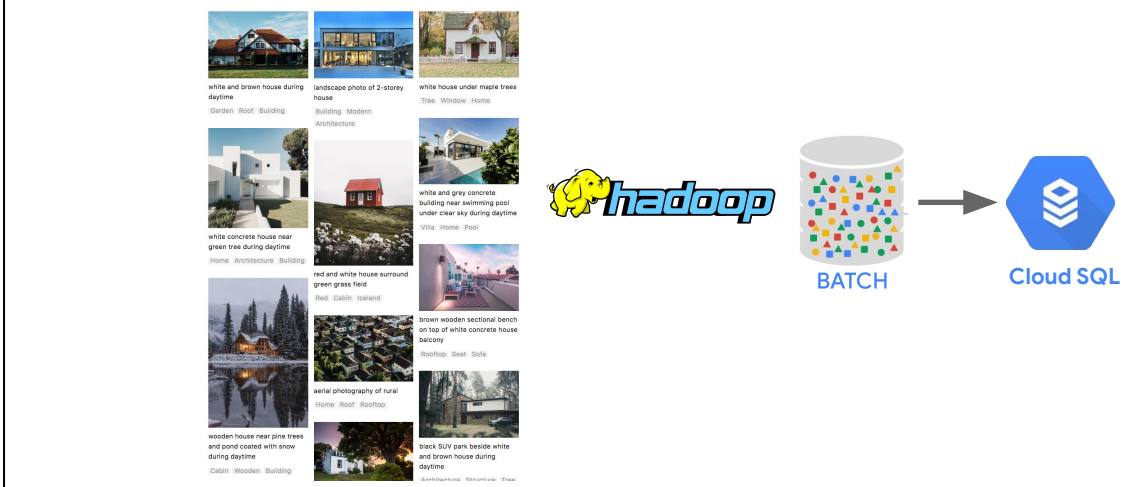
How often and where will you *compute* the predicted ratings? It's not as if rental recommendations have to be updated every time a new rating appears in our system. It is probably sufficient if we update the recommended houses for users once a day. Maybe even once a week. In other words, this does not have to be streaming. It could be batch.

Compute ratings quickly using a Hadoop cluster





On the other hand, we will probably have thousands of houses and millions of users, so it is probably best that we do it in a scalable way. We don't want to do it on a single machine. We want to do it in a fault-tolerant way that can scale to large datasets. A typical solution for this is to do it on a Big Data platform like Hadoop.



Finally, where will you store the results? We probably want to power a web application with these recommendations.

When the user logs on, we want to show that user the recommendations created specifically for them.

So, we need a transactional way to store the predictions. Assuming 5 predictions a user, that's a table of just 5 million rows. It's small enough and compact enough that a typical solution for this would be to store the data in a Relational Database Management System, a RDBMS, like MySQL.

Agenda

Recommendation systems

- Business applications
- Scenario: ML for housing rentals

Choosing the right solution approach

- On-premise to Google Cloud Platform
- Challenge: Utilizing and tuning on-premise clusters
- Off-cluster storage with Google Cloud Storage

So, we decided to use a Big Data platform like Hadoop and a RDBMS like MySQL to solve the housing recommendations problem.

These are both open-source technologies. You probably have Hadoop clusters and MySQL databases running on premises. Let's assume that your team already has this recommendations system working on-prem and see how to migrate it from on-premises to Google Cloud Platform.

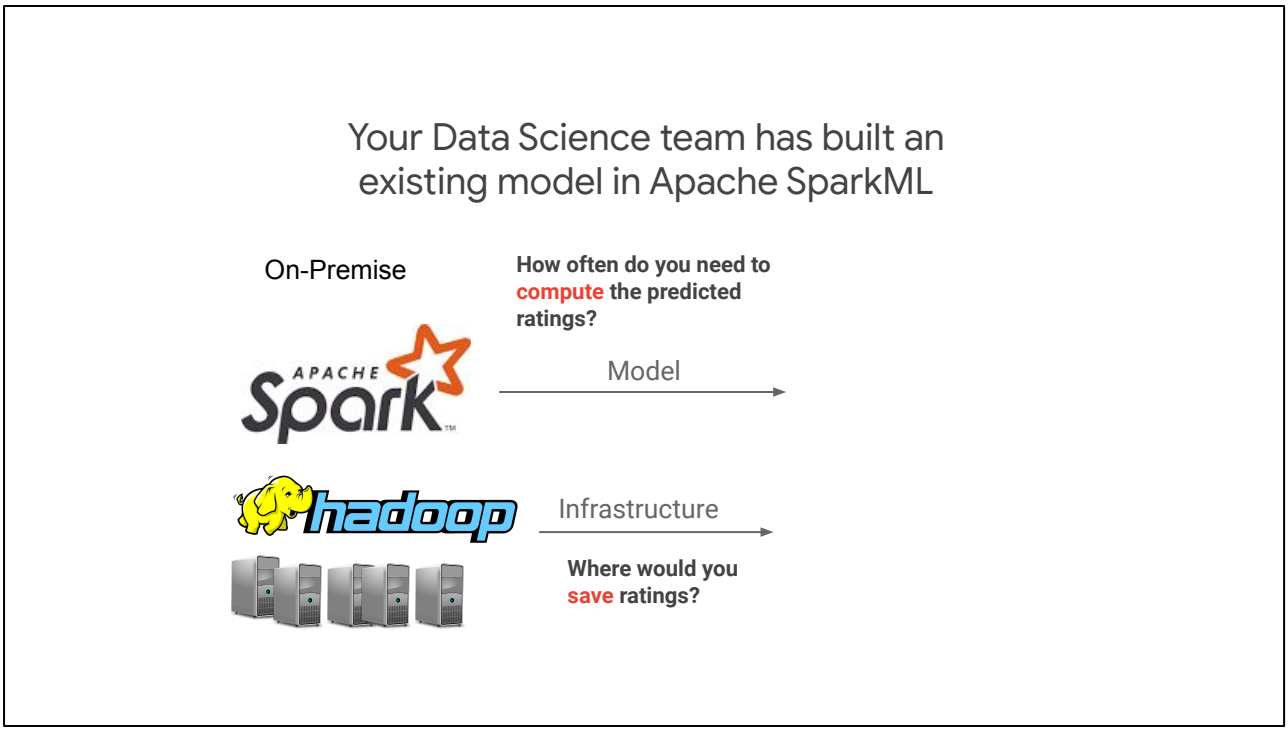
Of course, you want to do that only if the migration can add value, so we will look at that as well.

Your Data Science team has built an existing model in Apache SparkML

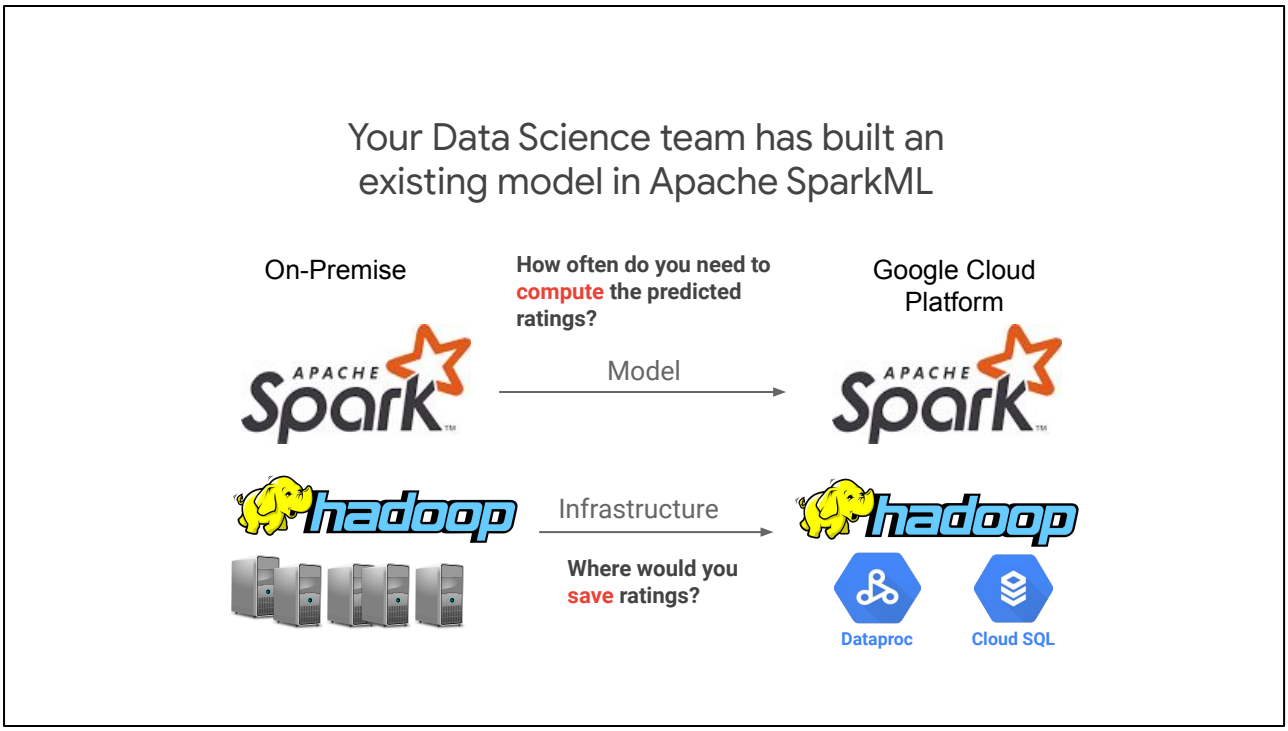
On-Premise

For our housing recommendation model, let's say our data science team already has a working on-premise model using a SparkML job on your Hadoop cluster.

They're interested in the scale and flexibility of Google Cloud Platform and they want to do a like-for-like migration of their existing SparkML jobs on-premise as a pilot project.



As explained in the previous lesson, we can get away with computing predicted ratings once a day. We don't need to get the recommendation in real-time. So Hadoop batch processing is enough.



Here we will use SparkML but instead of doing it on-premise we will run the ML job on Cloud Dataproc and store the ratings in Cloud SQL since it's a relatively small dataset of 5 recommendations for user. This is what you will practice in your first lab.

Incidentally, this is why the "Smart Reply" in Gmail is so amazing -- its recommendations have to be done in real-time when a new email pops up in your mailbox. Needless to say, Gmail is doing something far more sophisticated than what we are talking about here.

Choose your solutions based on access pattern

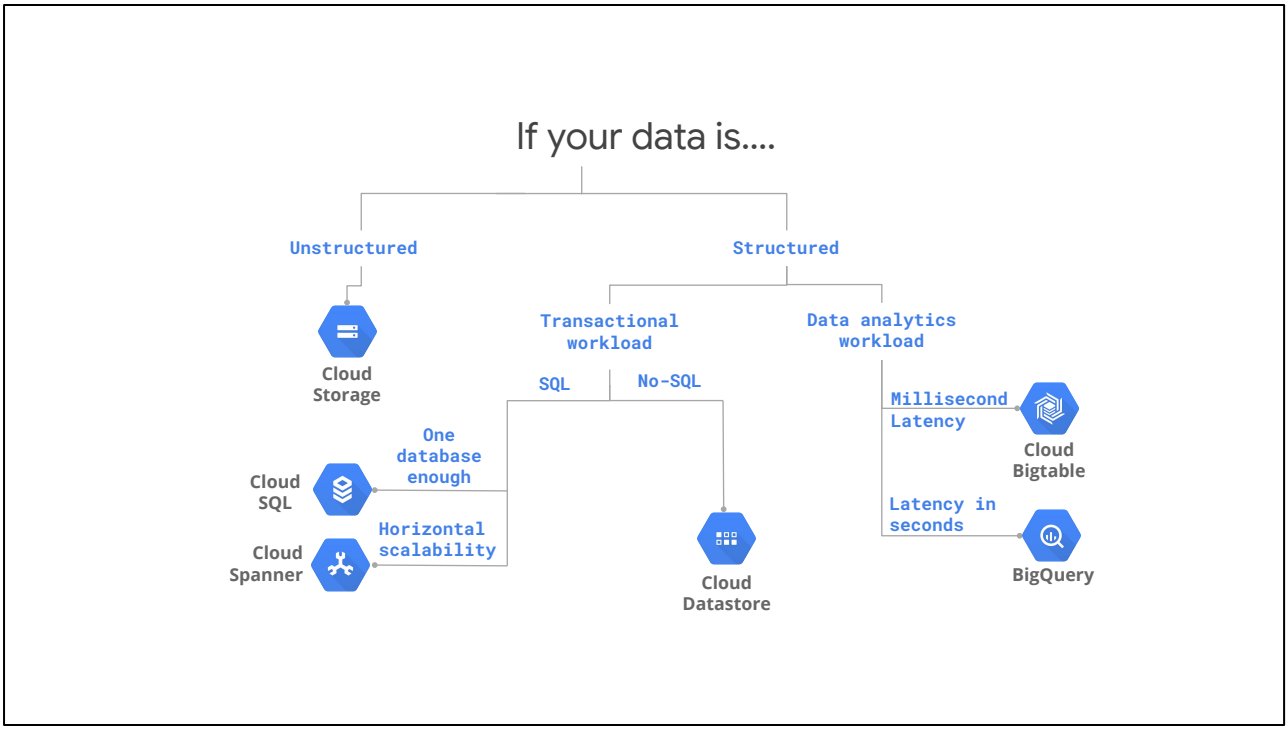
	Cloud Storage	Cloud SQL	Datastore	Bigtable	BigQuery
Capacity	Petabytes +	Gigabytes	Terabytes	Petabytes	Petabytes
Access metaphor	Like files in a file system	Relational database	Persistent Hashmap	Key-value(s), HBase API	Data warehouse
Read	Have to copy to local disk	SELECT rows	filter objects on property	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

Here is how we decided on using Cloud SQL among the other big data products available for storing our ratings. This is a good reference to follow based on your storage access pattern.

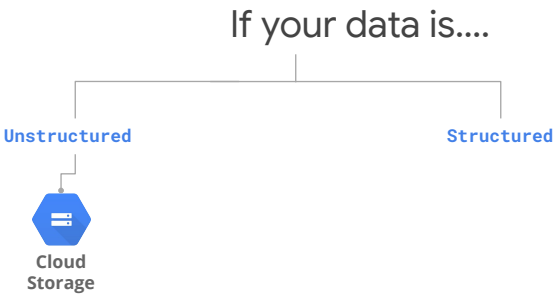
We'll cover the solutions in your other scenarios in this course, but briefly:

- Use Cloud Storage as a global file system.
- Use Cloud SQL as a RDBMS, for transactional, relational data that you access through SQL.
- Use DataStore as a transactional No-SQL, object-oriented database.
- Use Bigtable for high-throughput No-SQL, append-only data. A typical use case is sensor data, from connected devices.
- Use BigQuery as a SQL data warehouse, to power all your analytics needs.

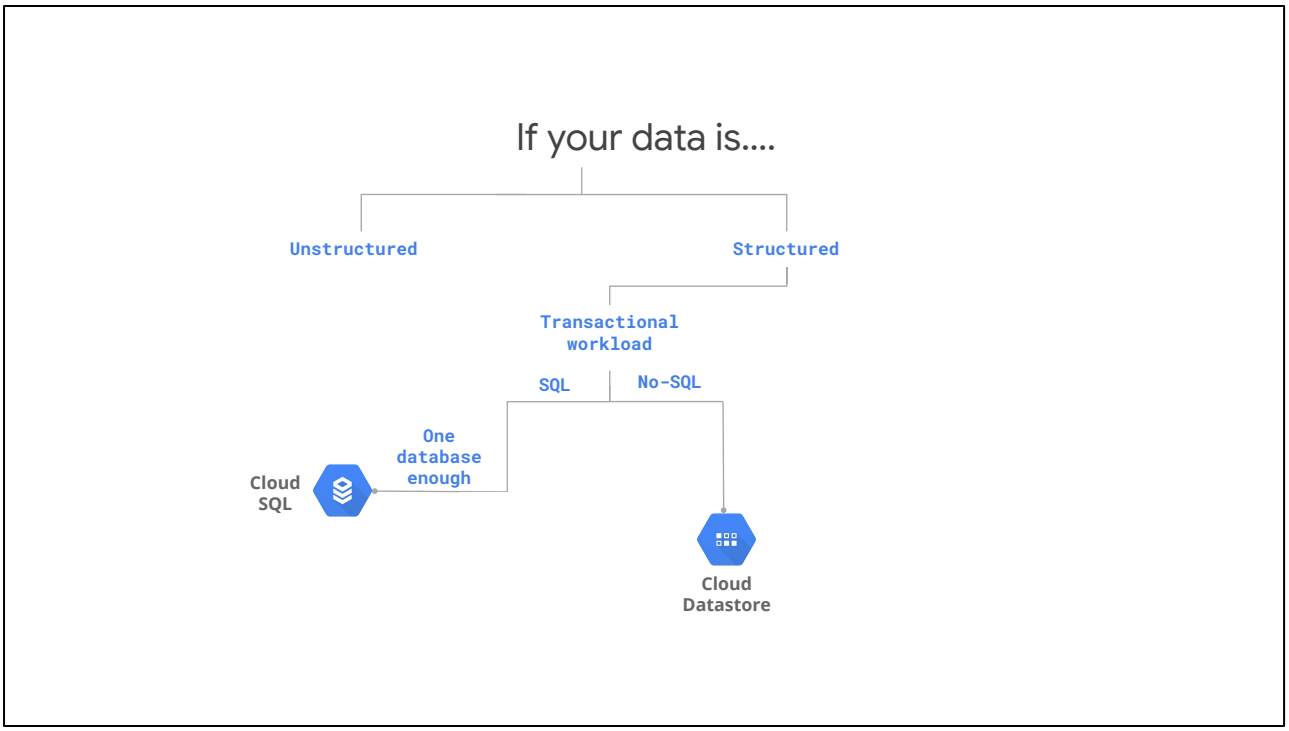
Here, we wanted a transactional database and expect to have data volumes in the gigabytes or less. Hence, Cloud SQL.



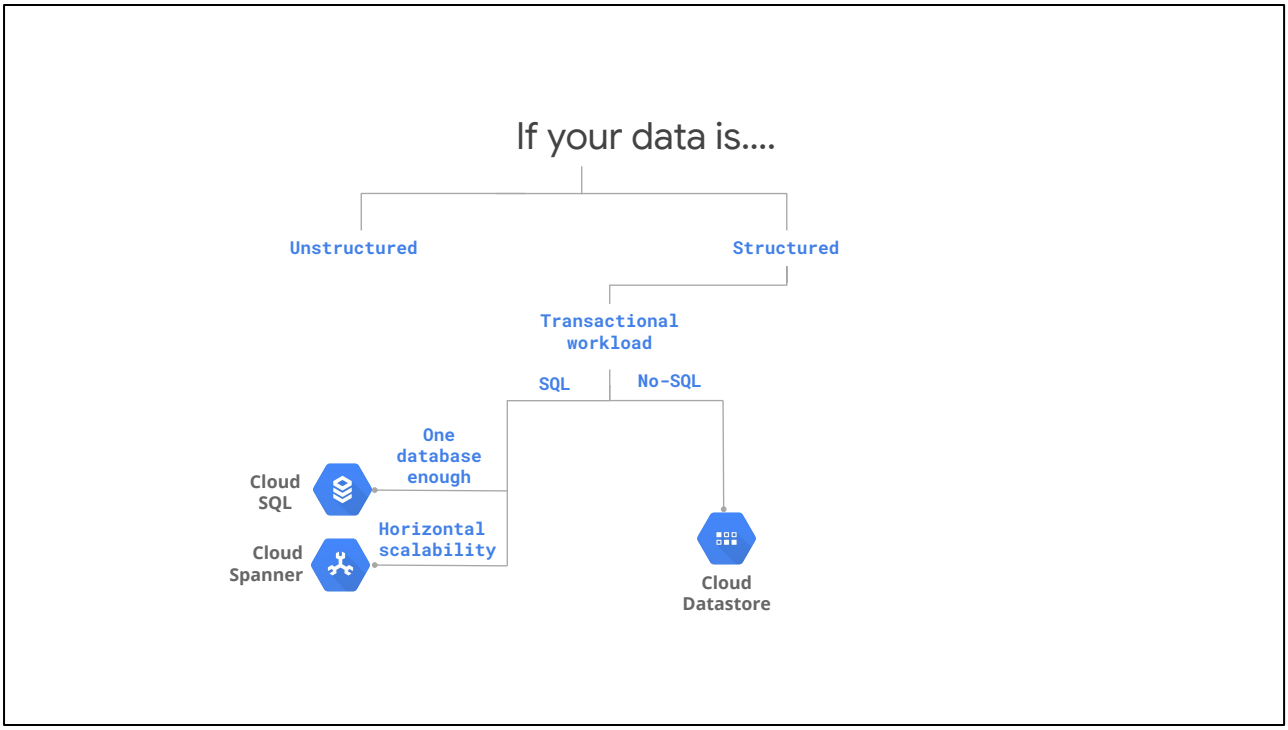
If you are a more visual learner, here is another good map of visualizing where to store your data in Google Cloud Platform.



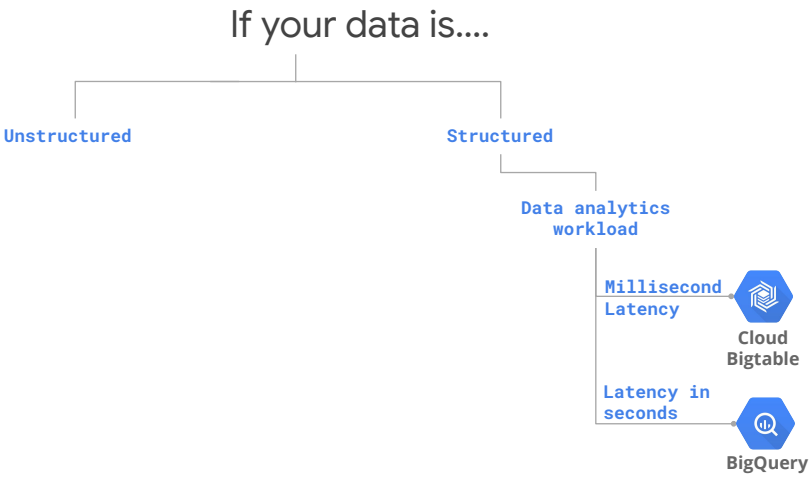
If your data is unstructured, like images and audio, use Cloud Storage.



If your data is structured, and you need transactions, use Cloud SQL or Cloud Datastore depending on whether you want your access pattern to be SQL or no-SQL. By No-SQL, we mean key-value pairs.

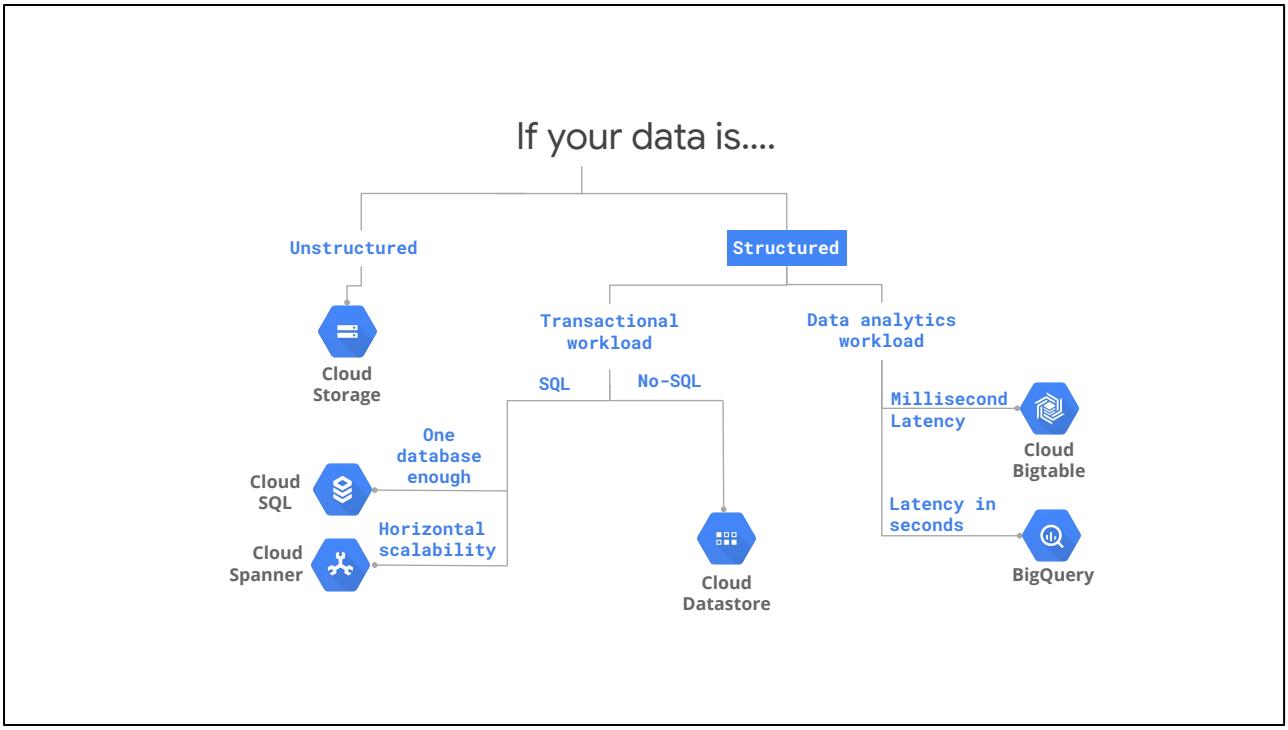


Cloud SQL generally plateaus out at a few gigabytes. For a transactional database with horizontal scalability, to deal with data larger than a few gigabytes or spread globally, use Cloud Spanner. If one database is enough, use Cloud SQL; if you'll need multiple databases either because you have a lot of data or because your application needs to be transactional across different continents, use Cloud Spanner.

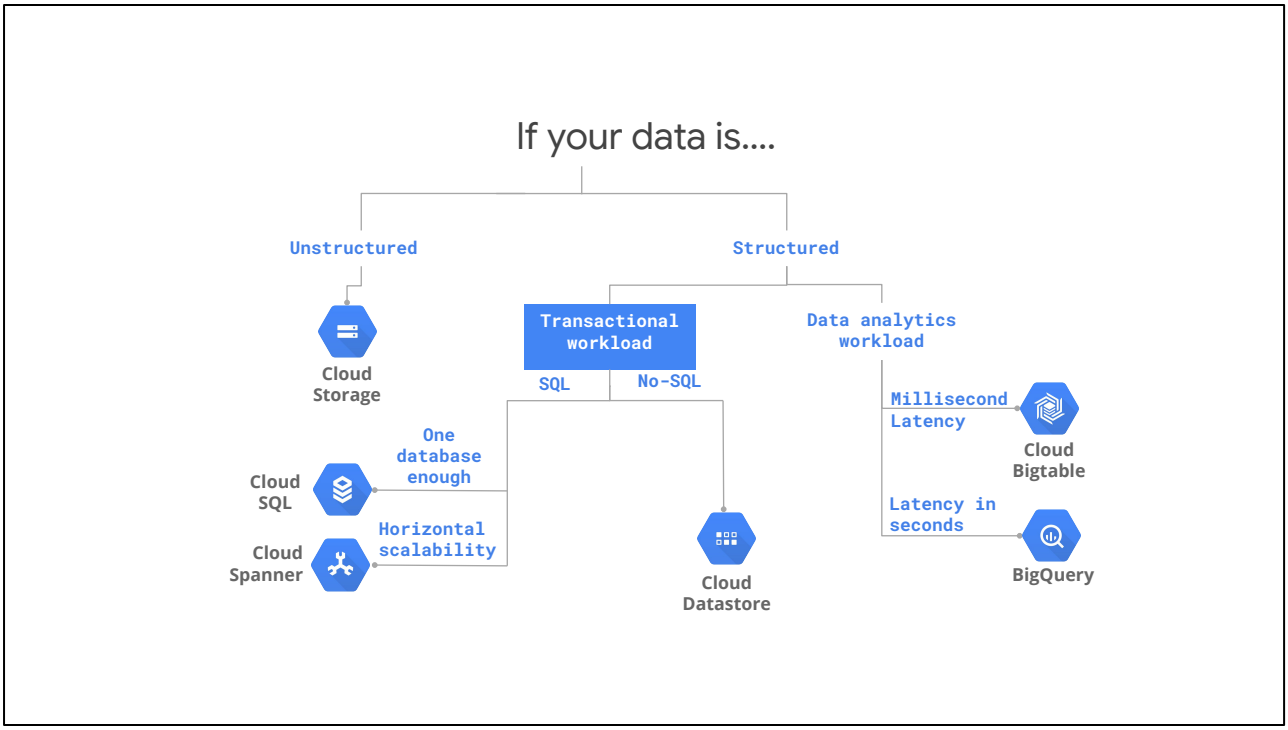


If your data is structured, and you want analytics, consider either Bigtable or BigQuery. Use Bigtable if you need real-time, high throughput applications; use BigQuery if you want analytics over petabyte-scale datasets.

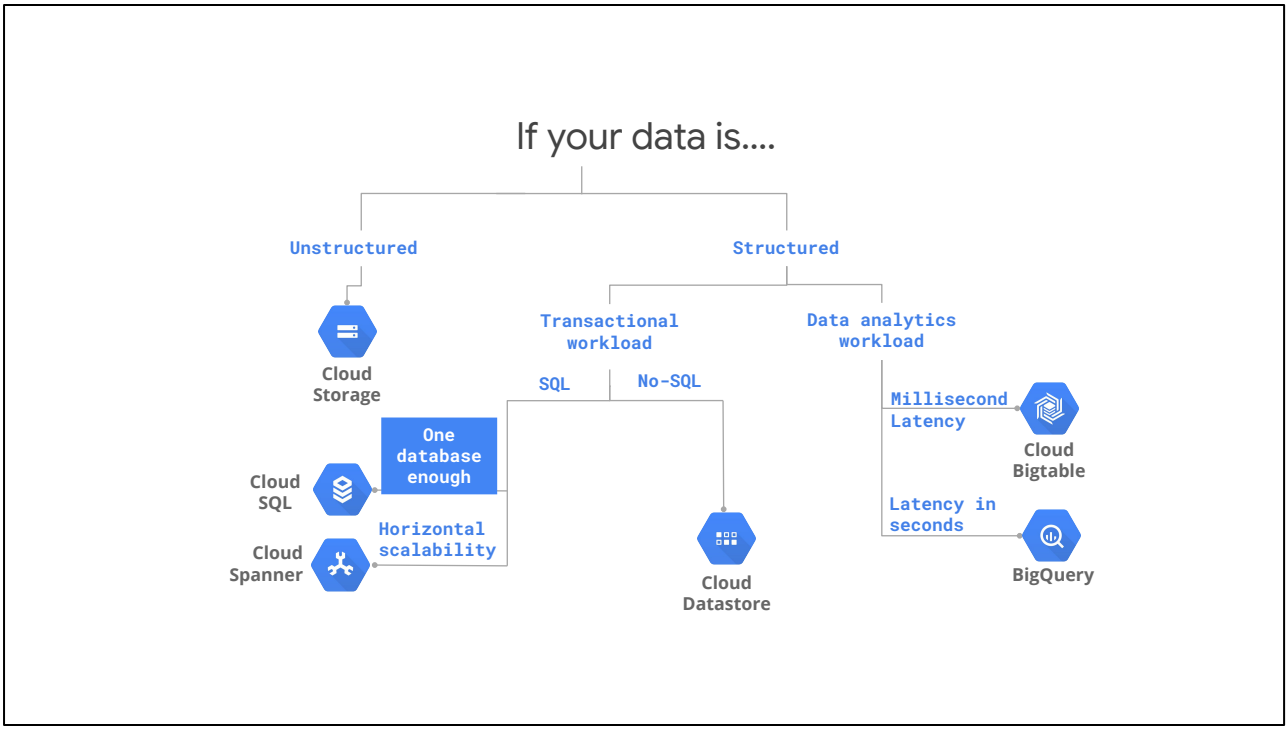
For our housing recommendation use case, we want to store our ratings and predictions somewhere. This is ...



A **structured** dataset of user ratings and houses

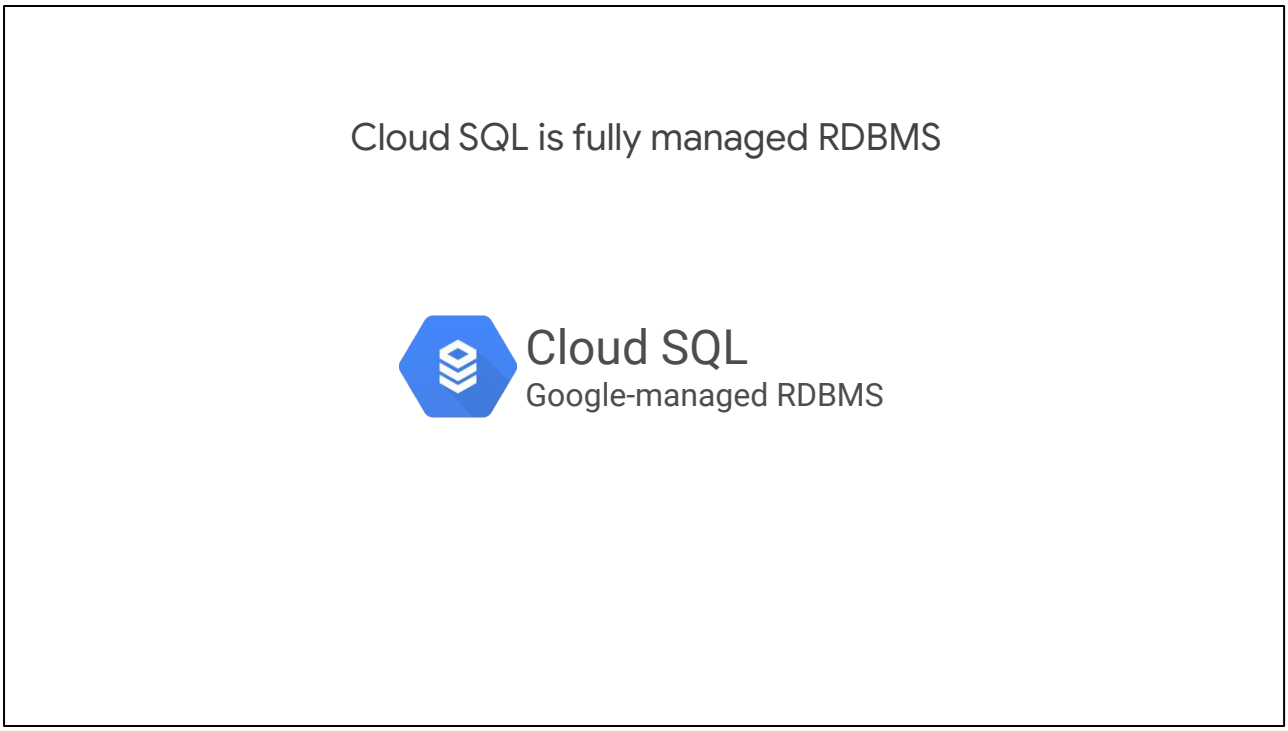


Built for a **transactional** workload (writes and reads)

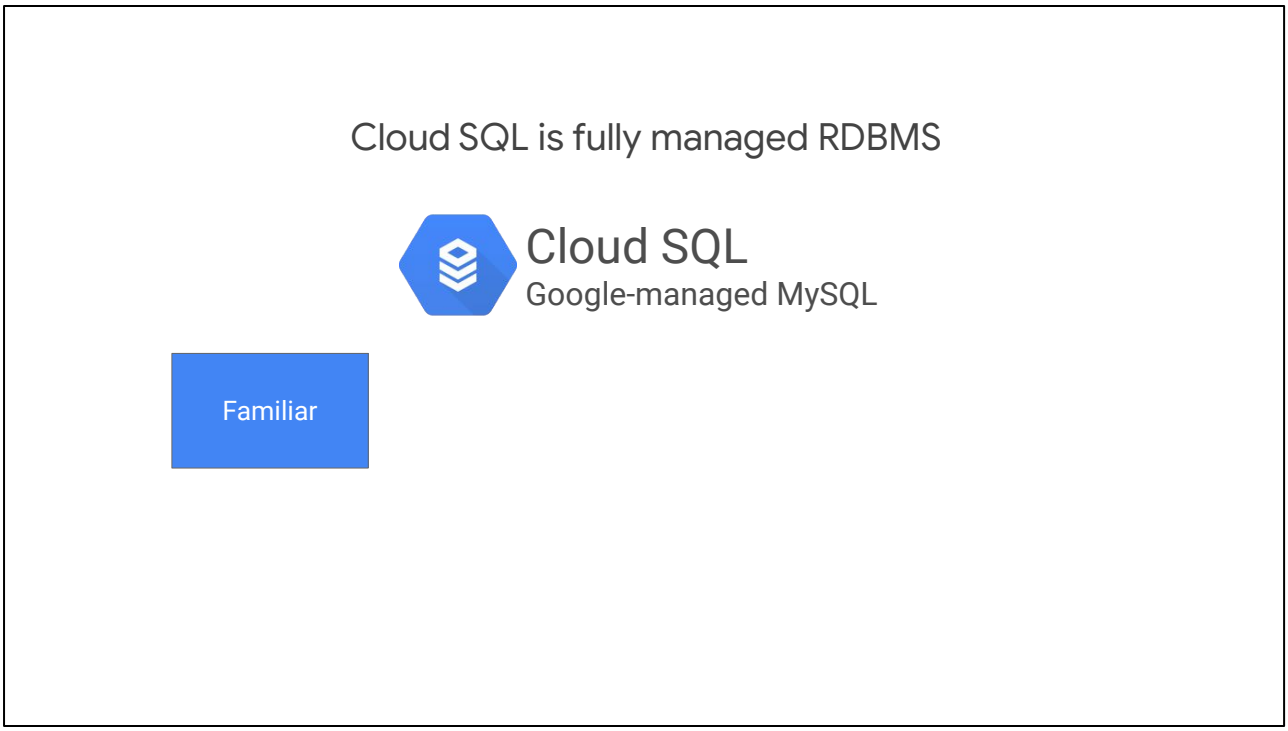


And **one database** is enough for our small dataset

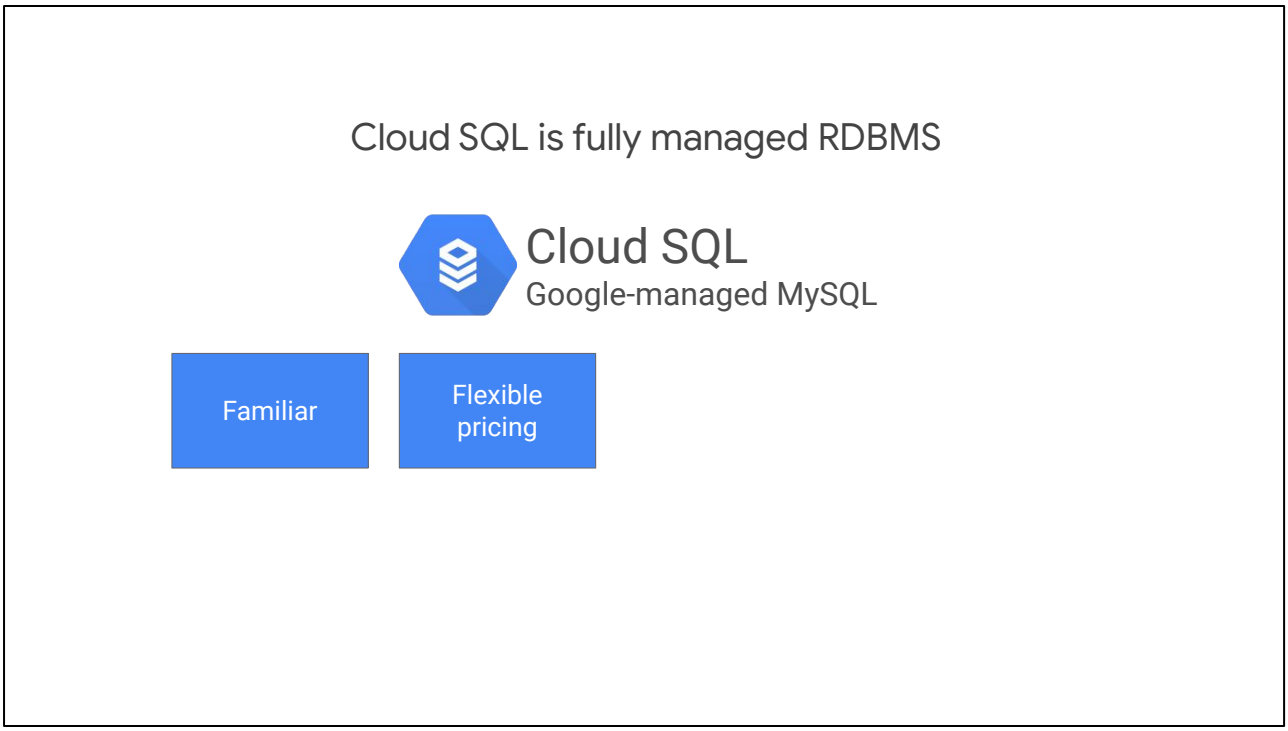
Hence, we pick Cloud SQL.



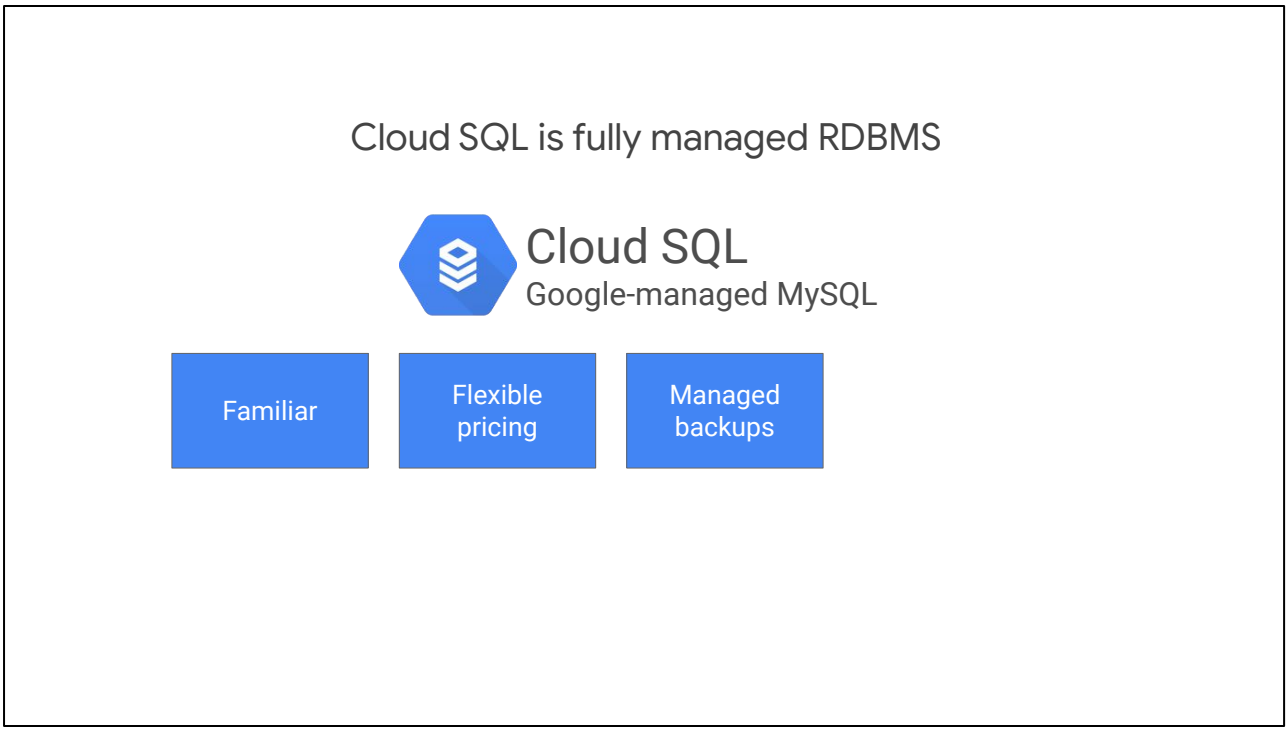
So what is Cloud SQL? It's a Google hosted and managed relational database in the cloud. Cloud SQL supports two open-source databases: MySQL and Postgres and other database solutions. In our case, we will be using MySQL.



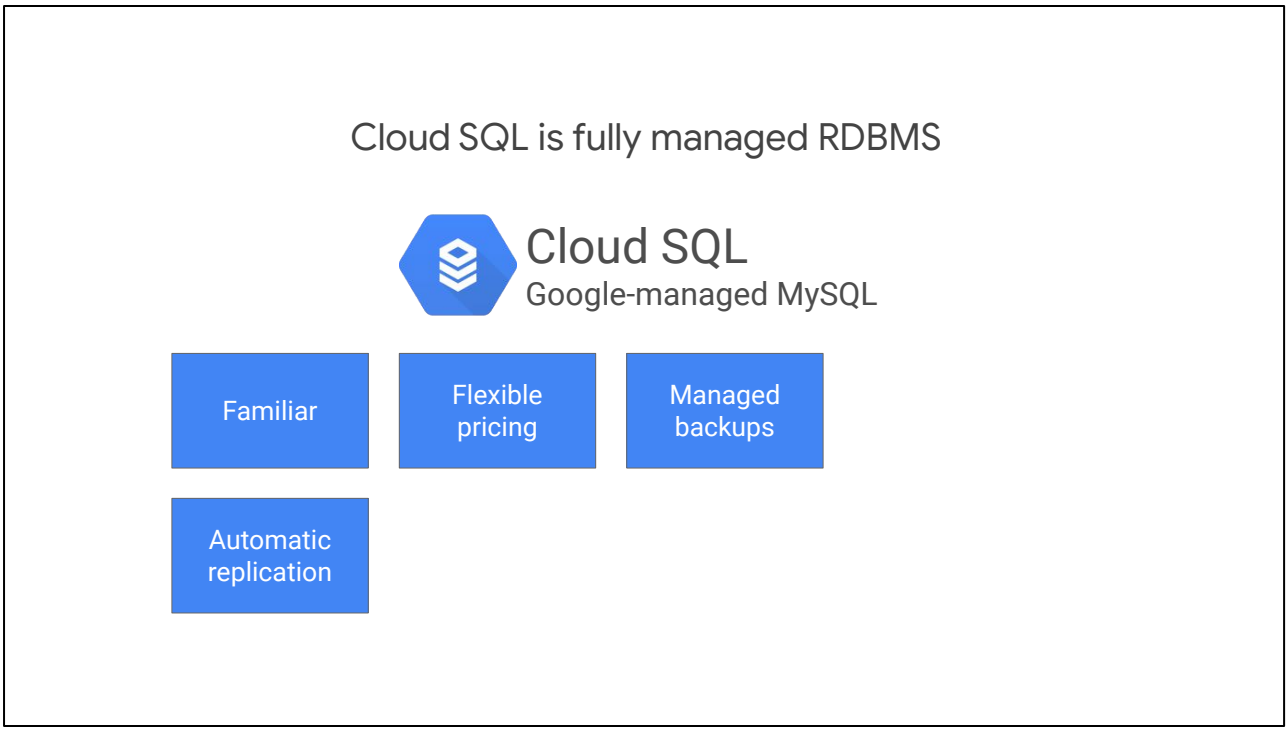
It’s familiar: Cloud SQL supports most MySQL statements and functions, even Stored procedures, Triggers and Views.



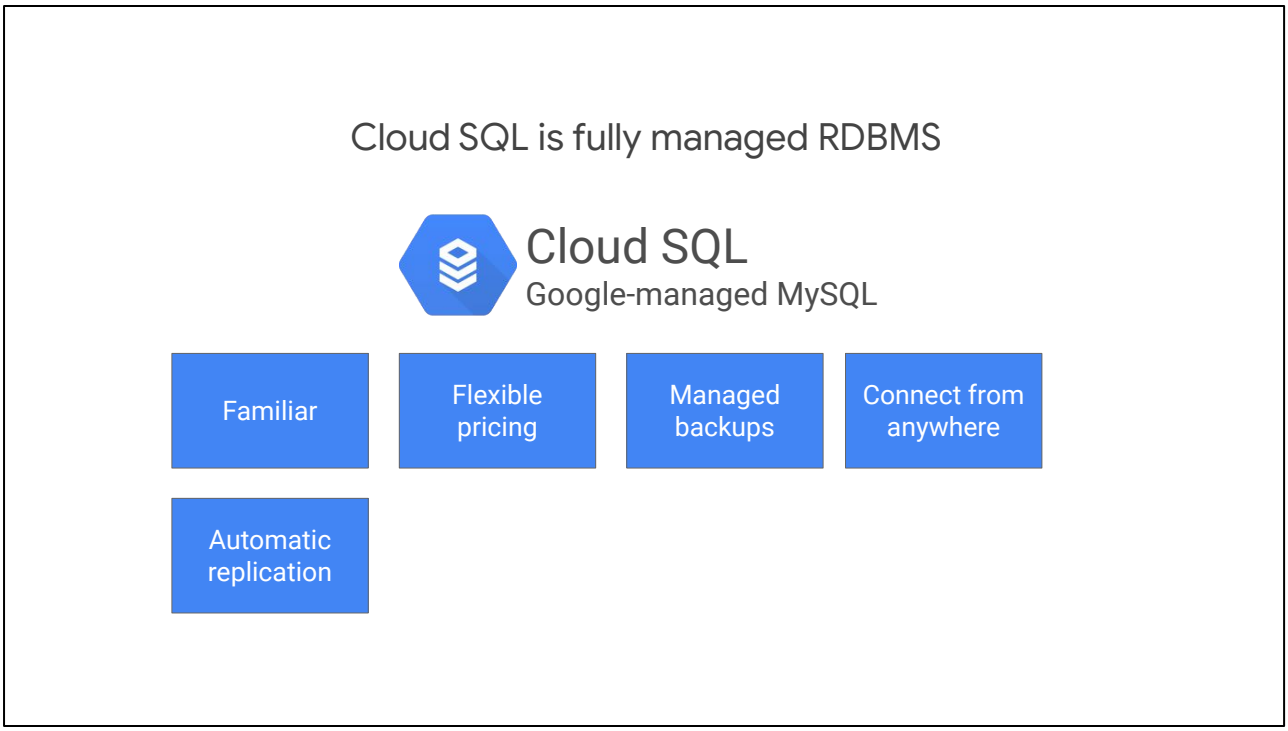
It brings the benefits of cloud economics in the form of Flexible pricing: You can pay for what you use.



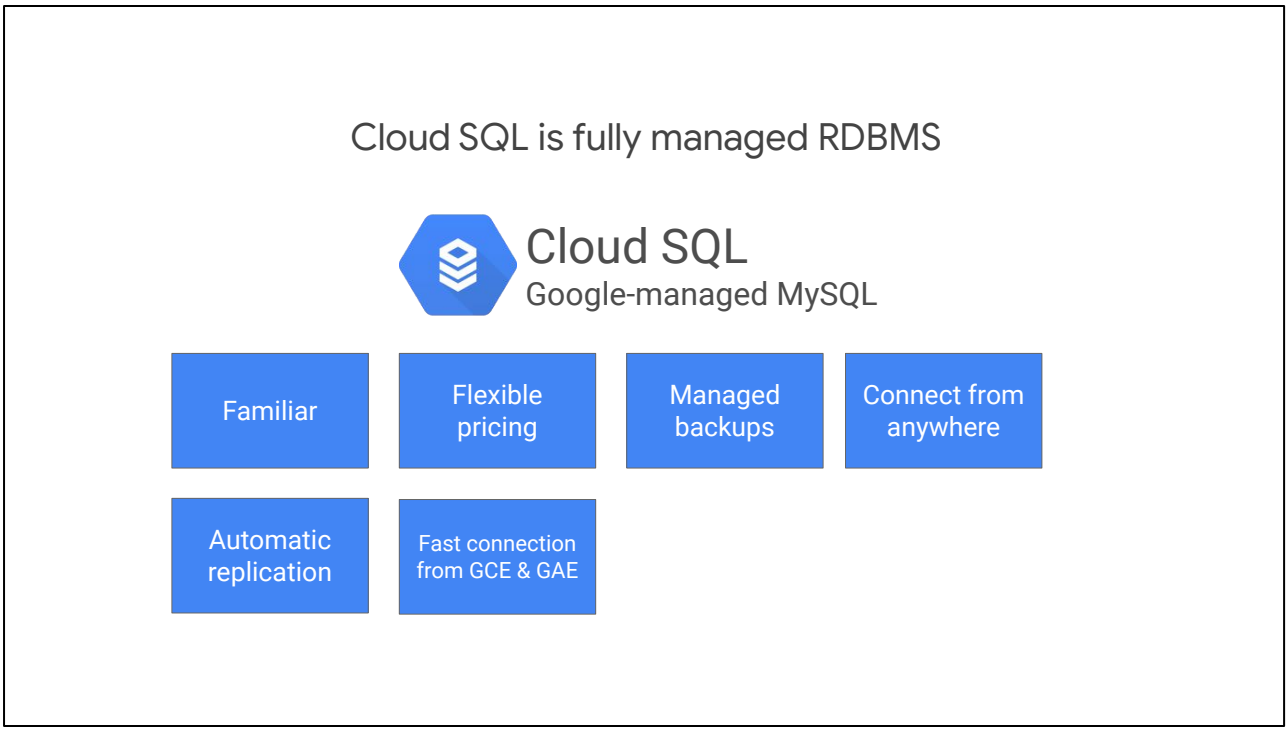
GCP manages the MySQL instance for you. This means things like Backups,



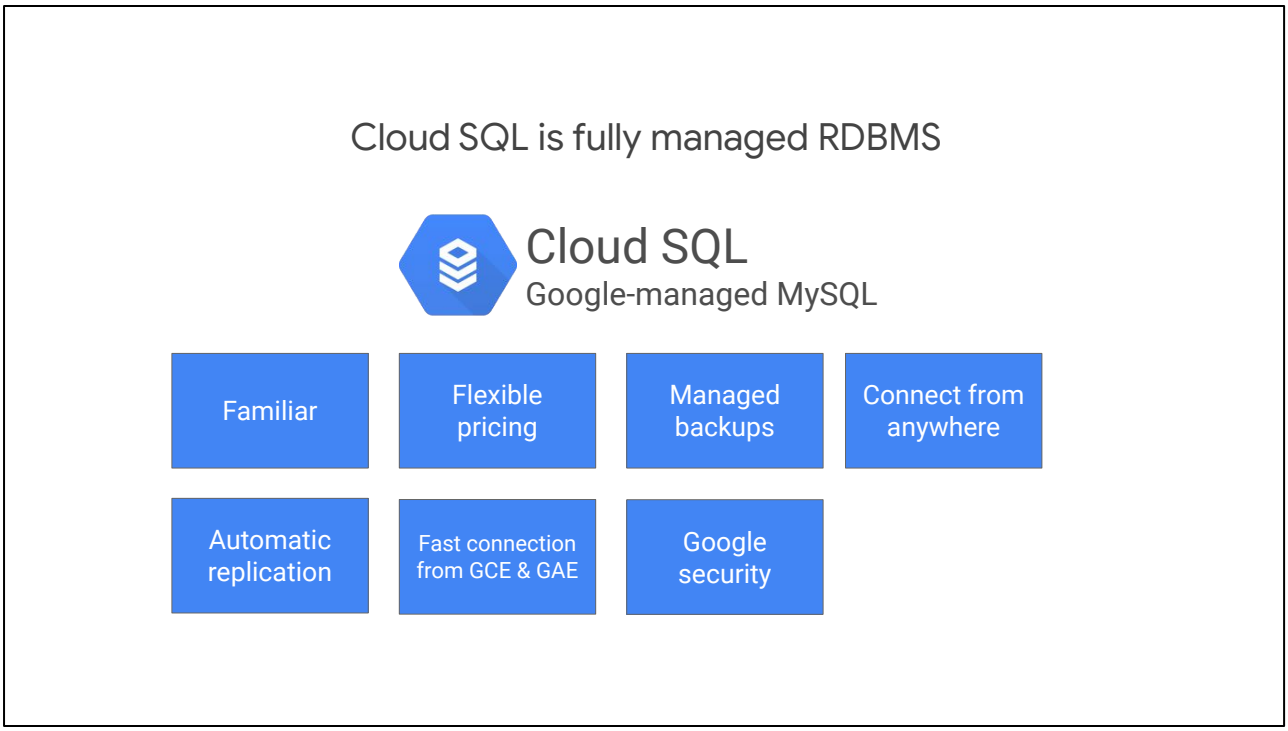
And replication



It's on the cloud, so you can connect to it from anywhere. You can assign it a static IP address, and use typical SQL connector libraries.

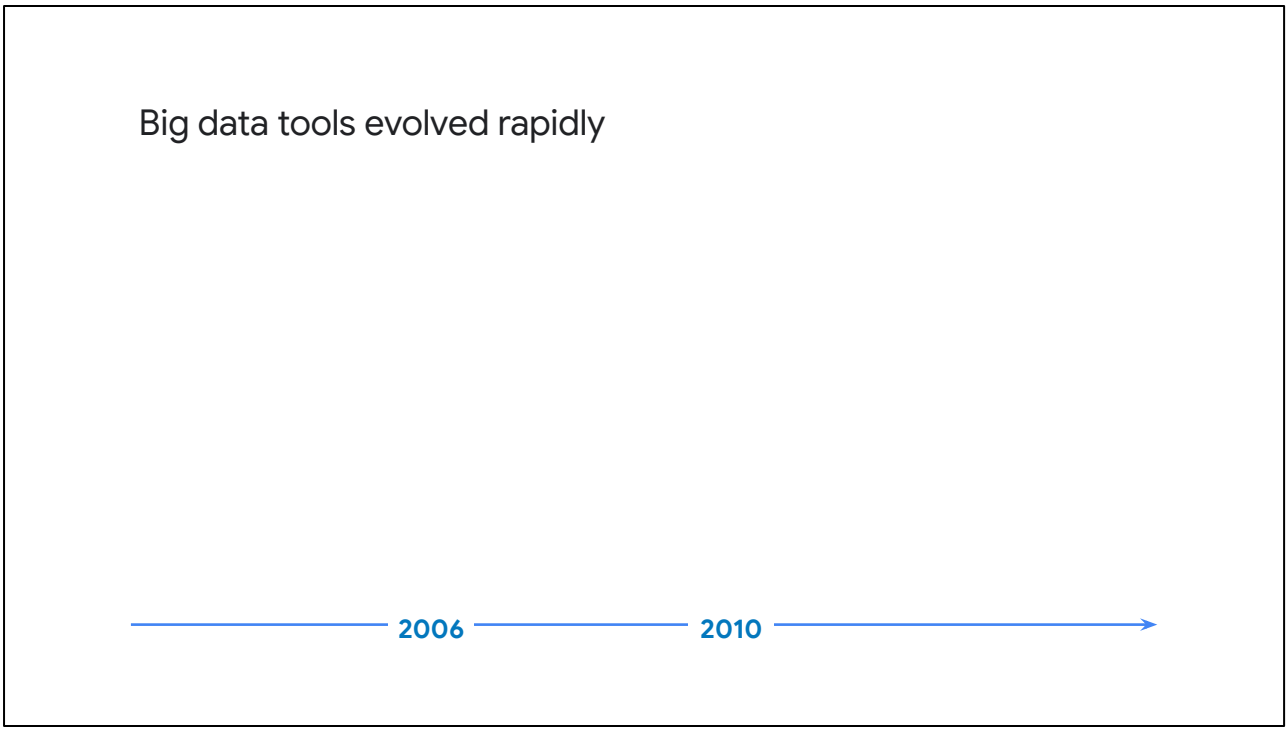


Because it’s behind the Google firewall, it’s fast. you can place your Cloud SQL instance in same region as your App Engine or Compute Engine applications and get great bandwidth.

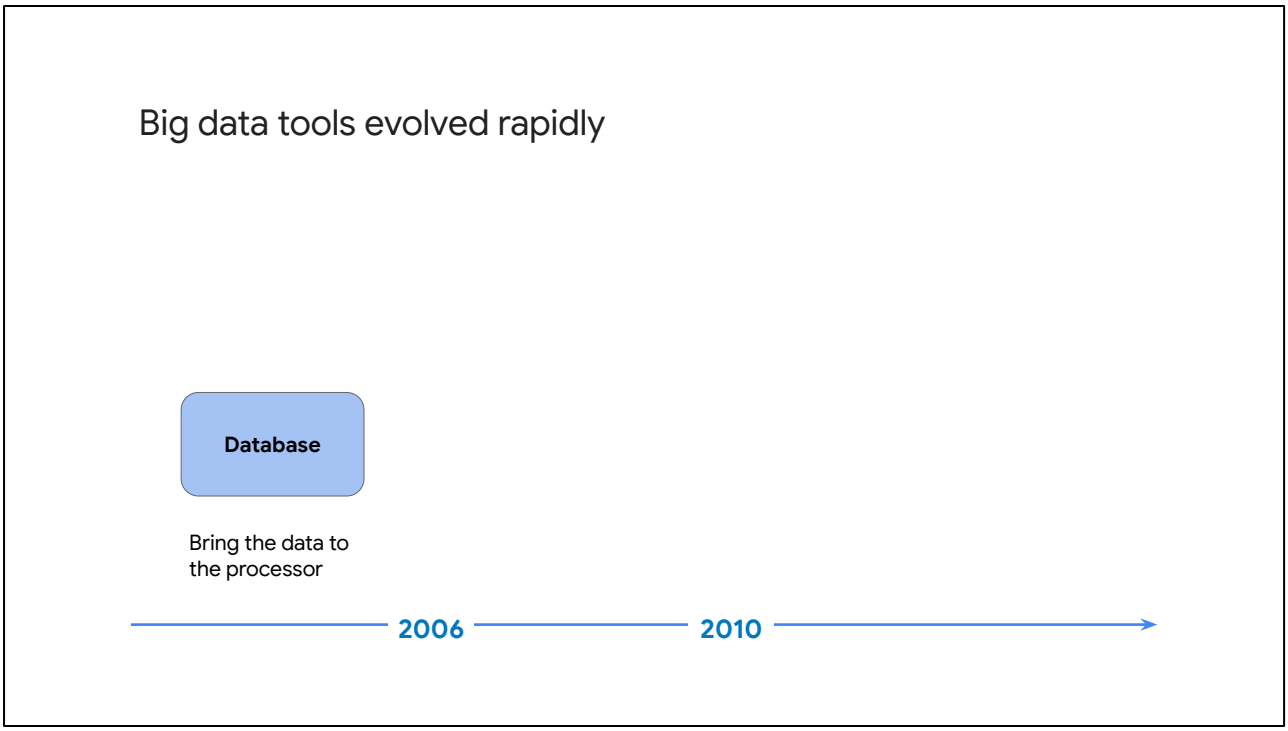


Also, you get Google security: Cloud SQL resides in secure Google datacenters.

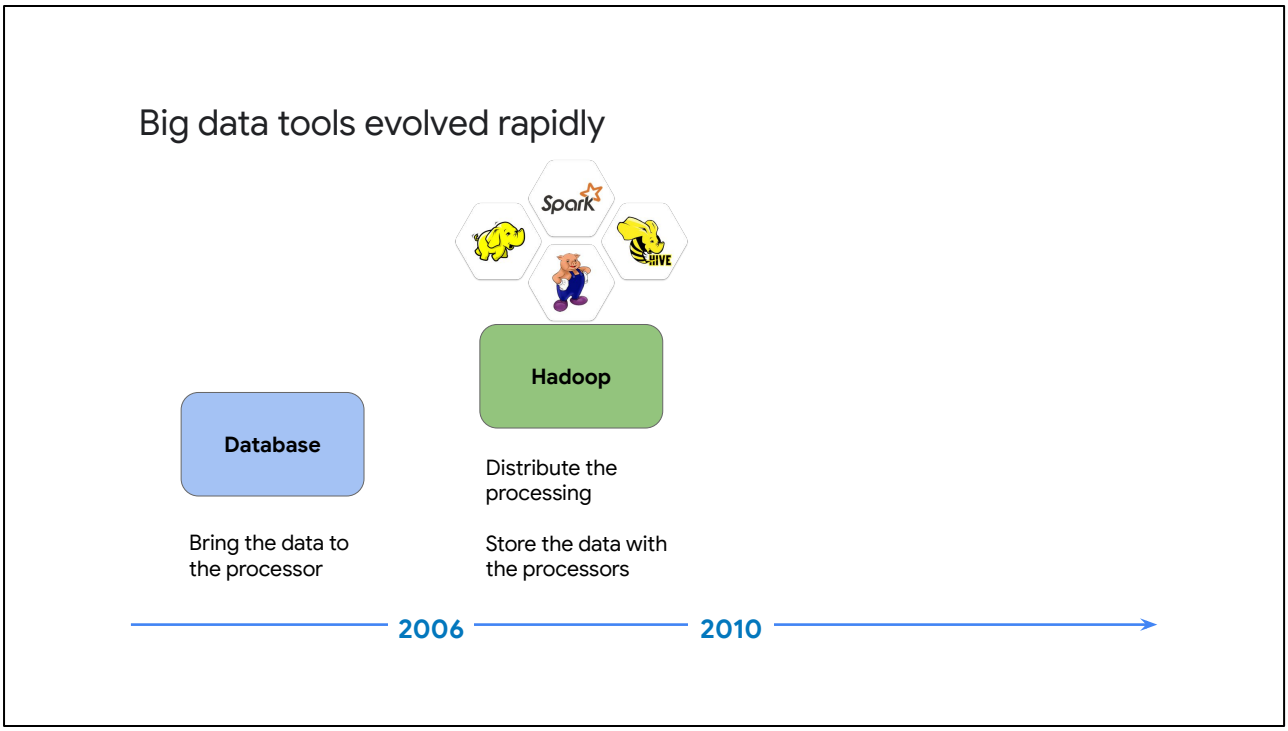
So, that covers where we will be storing the recommendations. What about where we will be doing the compute?



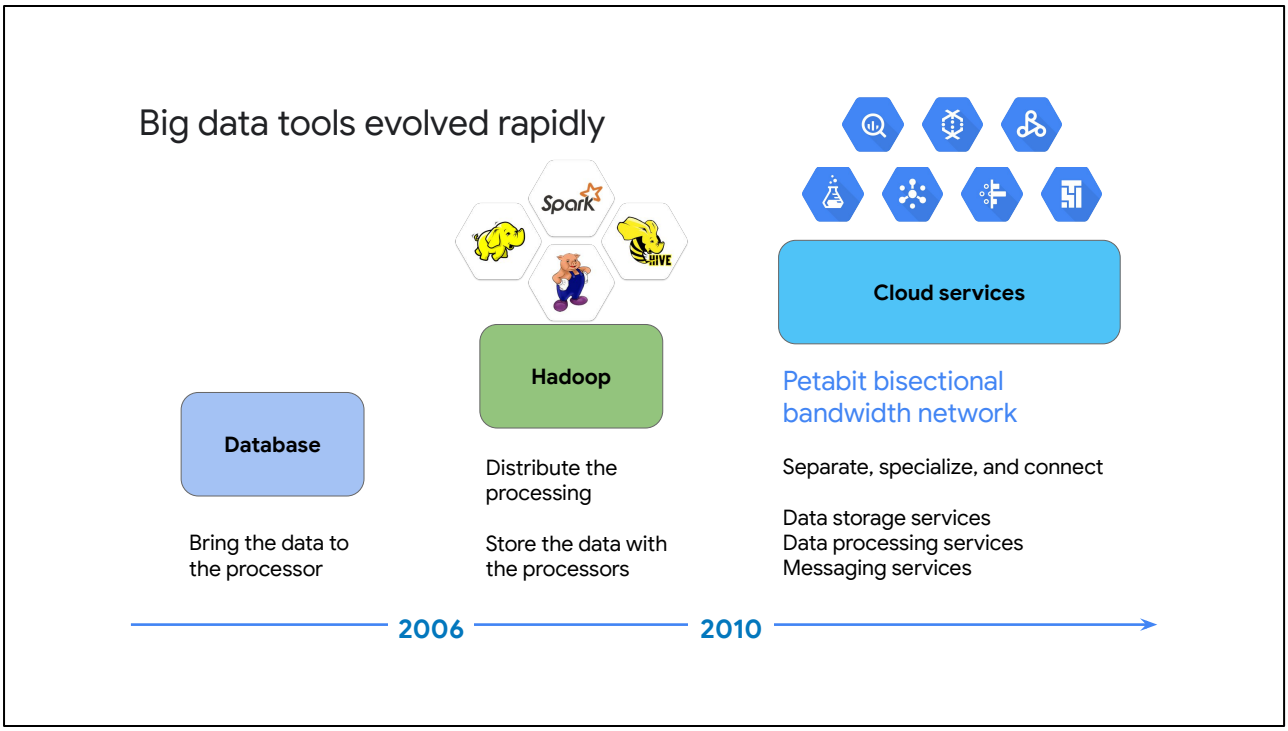
Let’s review where big data computations have been done historically and today.



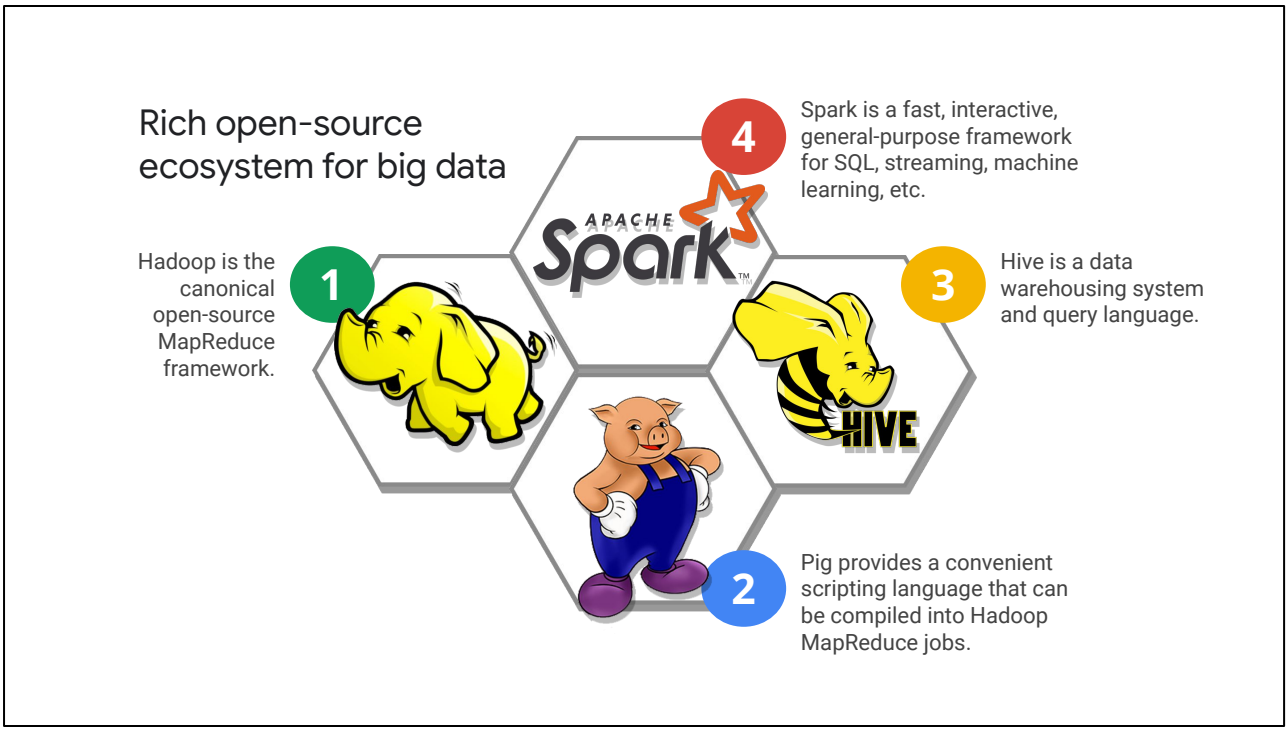
Before 2006 big data meant big databases. Database design came from a time when storage was relatively cheap and processing was expensive, so it made sense to copy the data from its storage location to the processor to perform data processing. Then the result would be copied back to storage.



Around 2006, distributed processing of big data became practical with Hadoop. The idea behind Hadoop is to create a cluster of computers and leverage distributed processing. HDFS -- the Hadoop Distributed File System stored the data on the machines in the cluster, and Map Reduce provided distributed processing of the data. A whole ecosystem of Hadoop-related software grew up around Hadoop, including Hive, Pig and Spark.



Around 2010 BigQuery was released, which was the first of many Big Data services developed by Google. Around 2015 Google launched Cloud Dataproc which provides a managed service for creating Hadoop and Spark clusters and managing data processing workloads.



The other piece of our system is training the machine learning model that our data science team created using SparkML.

Apache Spark is an open source software project that provides a high performance analytics engine for processing batch and streaming data. Spark can be up to 100 times faster than equivalent Hadoop jobs because it leverages in-memory processing. Spark also provides a couple of abstractions for dealing with data, including Resilient Distributed Datasets and Dataframes.

We'll be using our Spark job for the housing rental recommendations but we won't be doing it on-premise anymore.

Agenda

Recommendation systems

- Business applications
- Scenario: ML for housing rentals

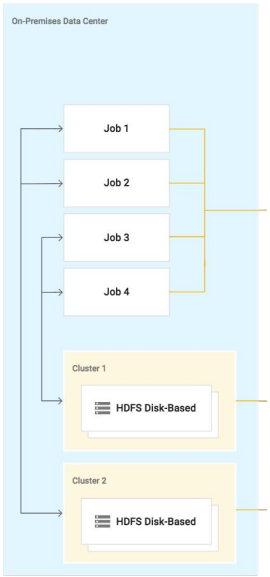
Choosing the right solution approach

- On-premise to Google Cloud Platform
- Challenge: Utilizing and tuning on-premise clusters
- Off-cluster storage with Google Cloud Storage

One of the most common challenges for managing on-premise Hadoop clusters is making sure they are efficiently utilized and tuned properly for the workloads.

Let's check in with our team and see what challenges they are facing running the SparkML job on their on-premise cluster.

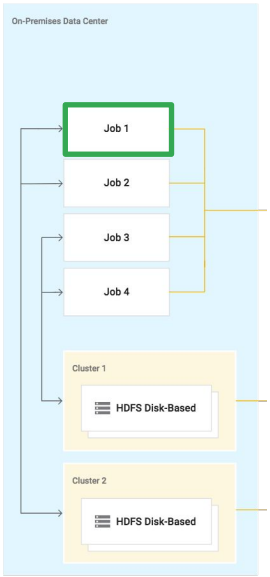
You manage your on-premises cluster and have four jobs



Our team has an on-premises data center running Hadoop represented by the blue box here. On any given week, they manage four different jobs for the organization. Our SparkML recommendation job being one of the four. The data is persisted in traditional HDFS disk-based storage.

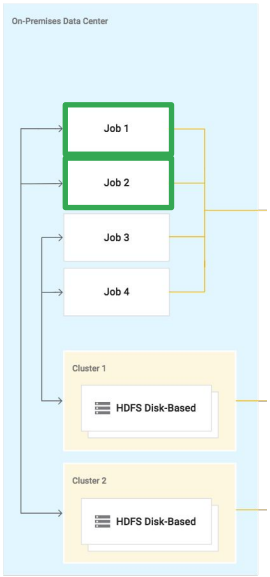
https://cloud.google.com/solutions/migration/hadoop/hadoop-gcp-migration-overview

Scenario 1:
Job #1 starts and consumes 50% of cluster resources



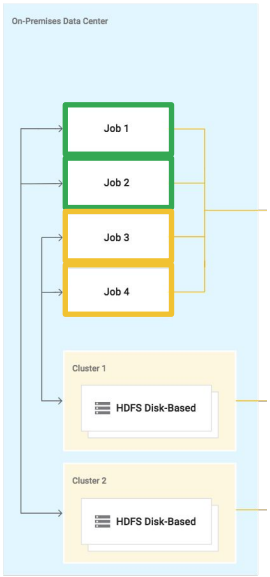
Here's an example scenario that may sound familiar. The team launches job #1 (say for argument it's a big data pipeline job) which ramps up and consumes 50% of the available on-premises cluster resources.

Scenario 1:
Job #2 starts and consumes 50% of cluster resources



Then the team has a special request from marketing to run their prediction job for an upcoming campaign later today. That will be job #2 and it ramps up and consumes the other 50% of available cluster resources. You can probably start to see the problem here.

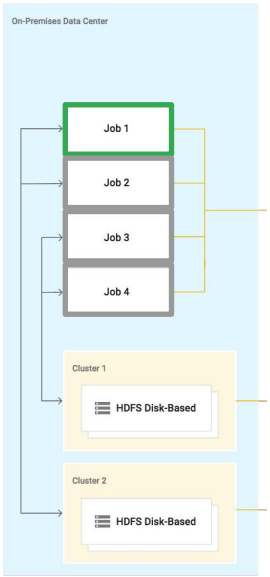
Scenario 1:
Jobs #3 and #4 attempt to start but are starved



Underprovisioned

If our SparkML job was #3 or #4 it would get starved out of resources because the cluster's compute capacity is underprovisioned for the demands of the organization's Hadoop jobs.

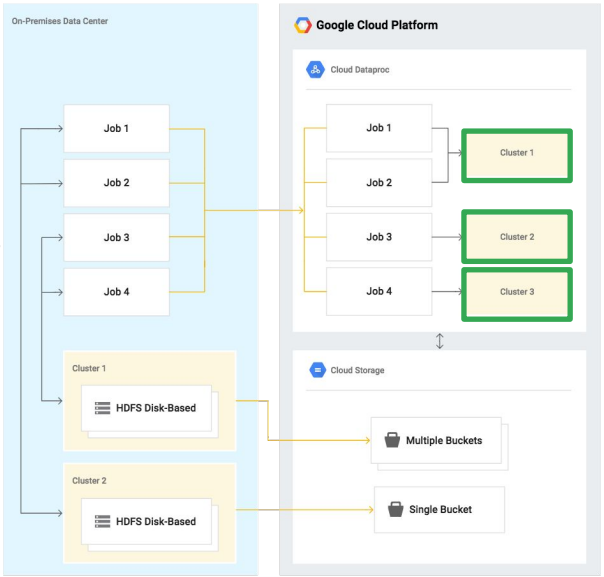
Scenario 2:
Job #1 is consuming 50% of cluster resources and jobs 2, 3, 4 are not needed



Overprovisioned

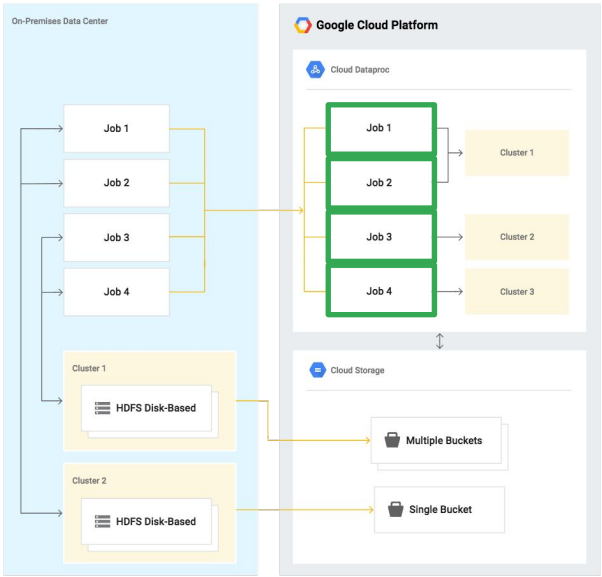
Or, a costly alternative, is where only one job is running and uses 50% of the cluster resources and the other 50% are powered and available but not needed. The problem lies in the static nature of the on-premise cluster capacity. The team needs a better way to optimize the usage of the cluster's resources without the headache of continually tuning and adding and removing servers themselves.

Hadoop clusters are now fungible resources



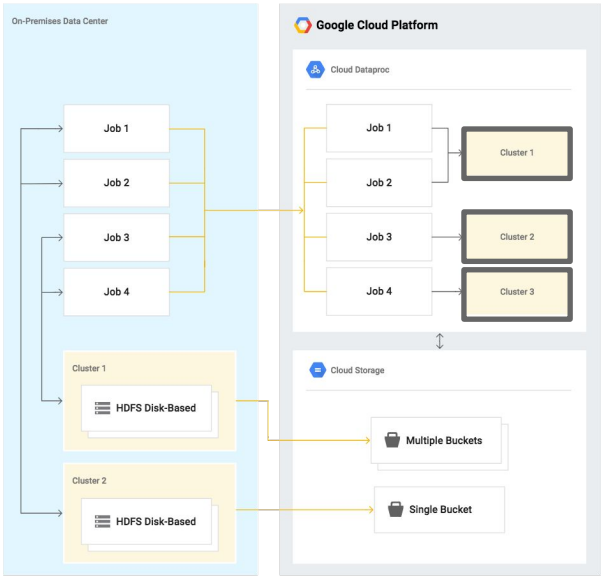
This is where Google Cloud Platform can help. You can now think of clusters as flexible resources. With Cloud Dataproc, GCP's managed Hadoop product, you can spin up as many or as few cluster resources as you need in the cloud. Notice how I said cluster resources -- you use them when you need to run jobs and turn them off when they're not needed anymore. In this scenario, we can have Jobs #1 and #2 run on their customized Cluster #1 and Jobs #3 and #4 can run on their own clusters too.

Jobs can get the resources they need

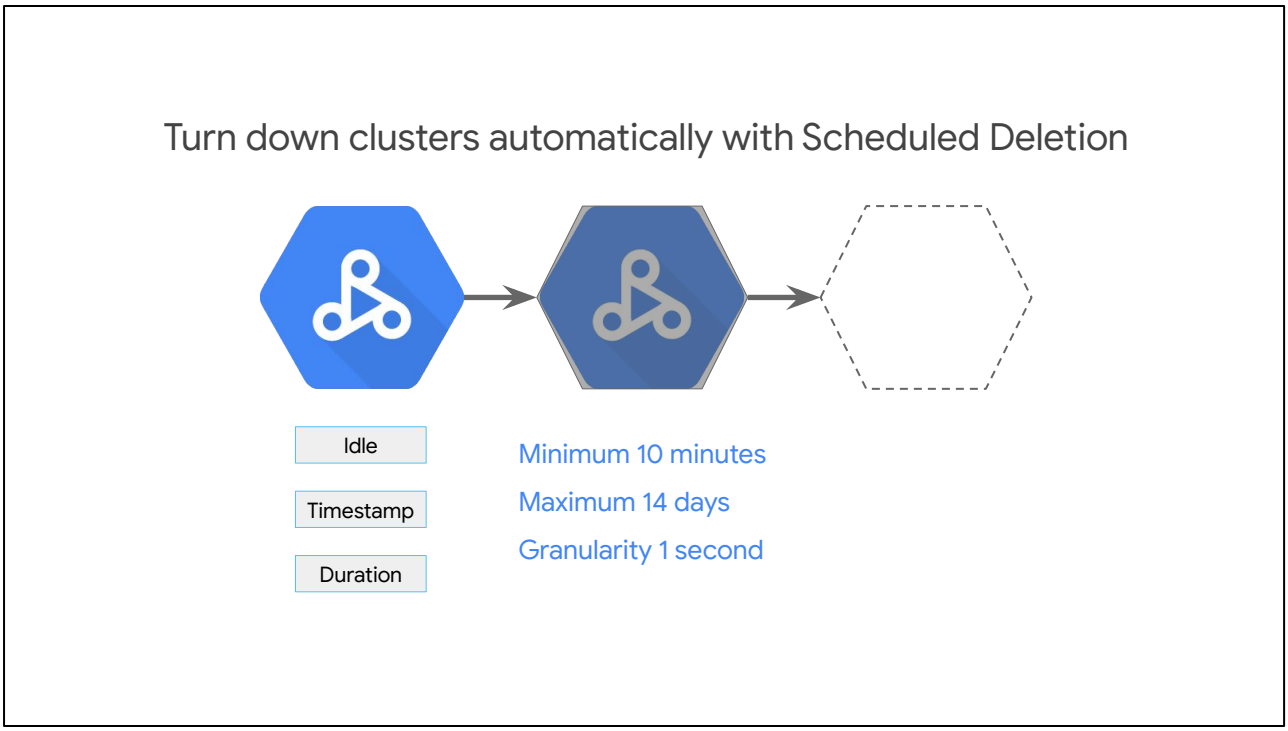


Your jobs get the resources they need.

Clusters can be turned down when no jobs are running



And you can shutdown the clusters when they're not in use. The clusters themselves become fungible resources.



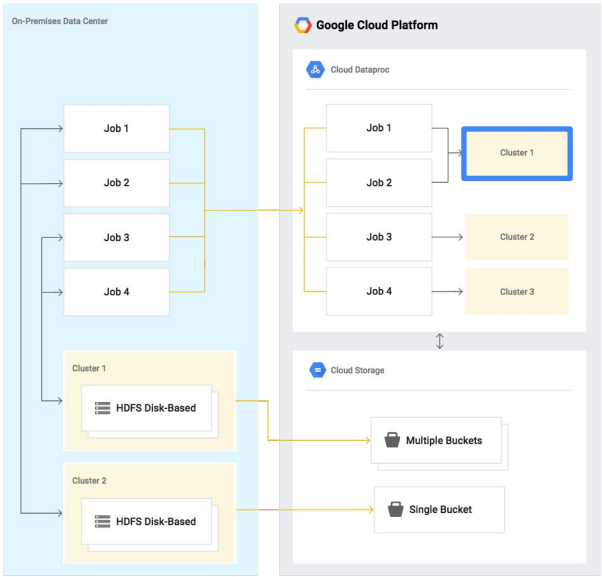
You can even automate when the clusters shutdown so you don't pay for resources that you're not using

You can set shutdown triggers based on

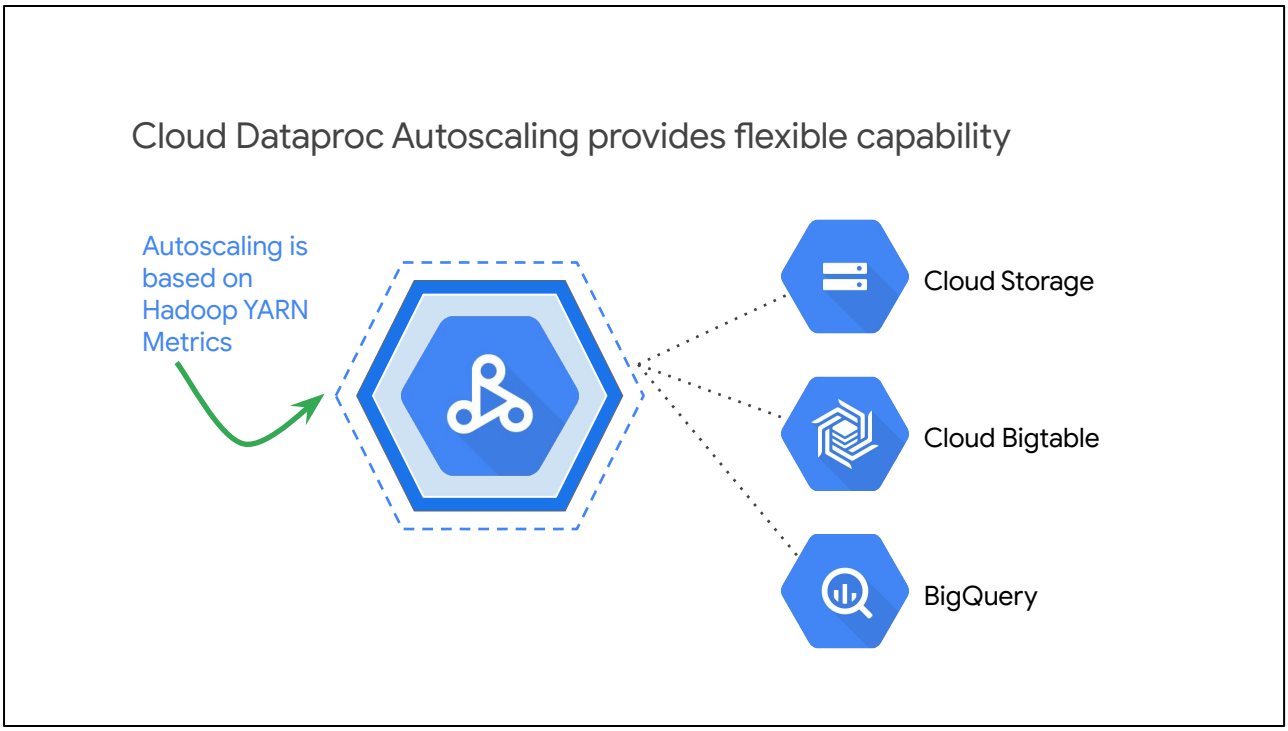
- how long clusters have sat idle
- A specific timestamp
- Or a duration in seconds to wait

https://cloud.google.com/dataproc/docs/concepts/configuring-clusters/scheduled-deletion

Clusters can be resized if job needs change

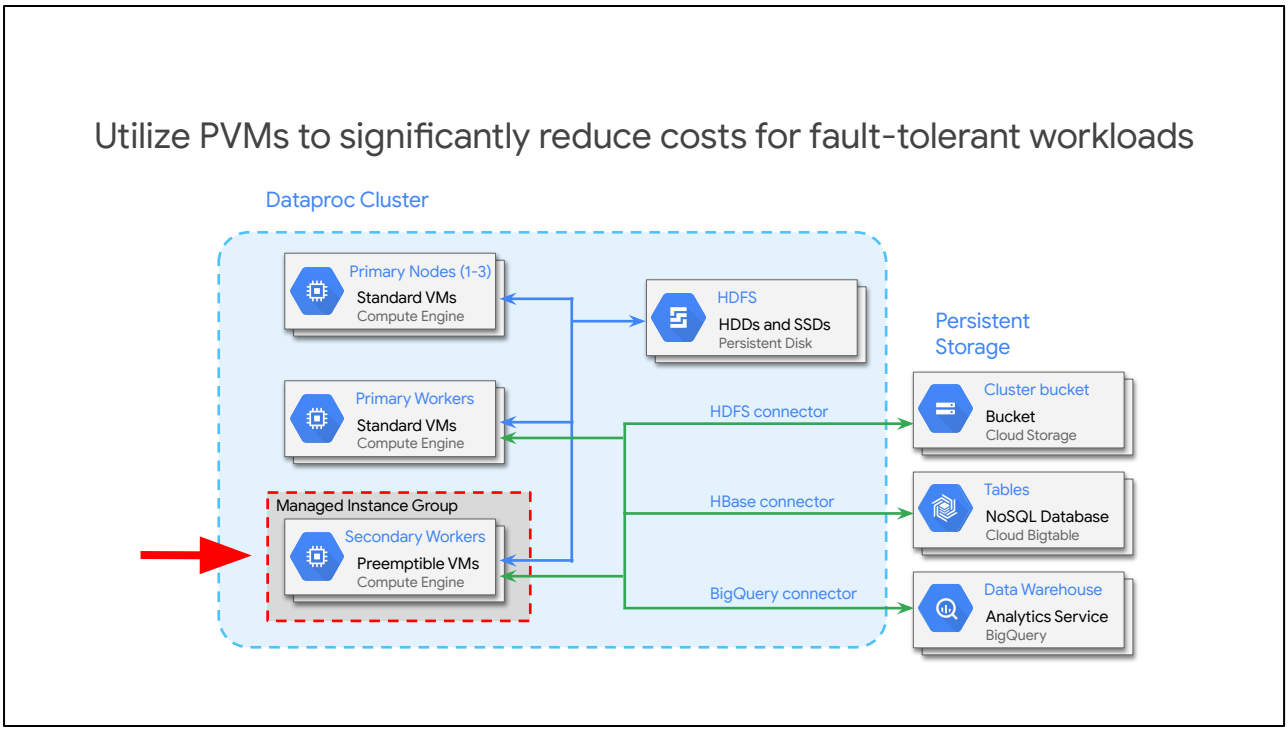


What about if the needs of a job change and we need a bigger or smaller cluster?



Cloud Dataproc Autoscaling provides flexible capacity to help meet that need. It makes scaling decisions based on Hadoop YARN Metrics.

You can use autoscaling as long as shutting down the cluster's nodes will not remove any data. That is the case if you use Cloud Storage, Bigtable, or BigQuery to store your data. But it won't work if you store your data in HDFS.



In addition to autoscaling, another advantage of running Hadoop clusters on GCP is that you can incorporate Preemptible Virtual Machines into your cluster architecture.

Preemptible VMs are highly affordable, short-lived compute instances suitable for batch jobs and fault-tolerant workloads. Preemptible VMs offer the same machine types and options as regular compute instances and last for up to 24 hours. If your applications are fault-tolerant -- and Hadoop applications are -- then preemptible instances can reduce your Google Compute Engine costs significantly.

Preemptible VMs are up to 80% cheaper than regular instances. Pricing is fixed so you will always get low cost and financial predictability, without taking the risk of gambling on variable market pricing.

Like autoscaling, though, Preemptive VMs work best when your workload can function without data being stored on the cluster.
That's what we will look at next.

https://cloud.google.com/preemptible-vms/

Agenda

Recommendation systems

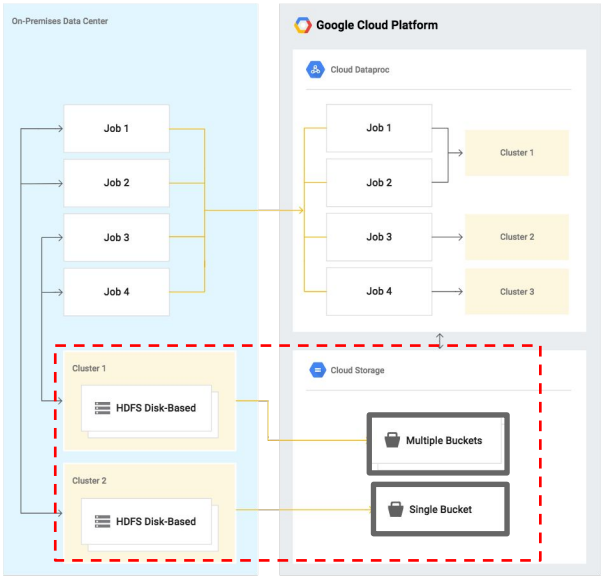
- Business applications
- Scenario: ML for housing rentals

Choosing the right solution approach

- On-premise to Google Cloud Platform
- Challenge: Utilizing and tuning on-premise clusters
- Off-cluster storage with Google Cloud Storage

As you might remember from my demo earlier, separating compute and storage enables cost-effective scale for workloads.

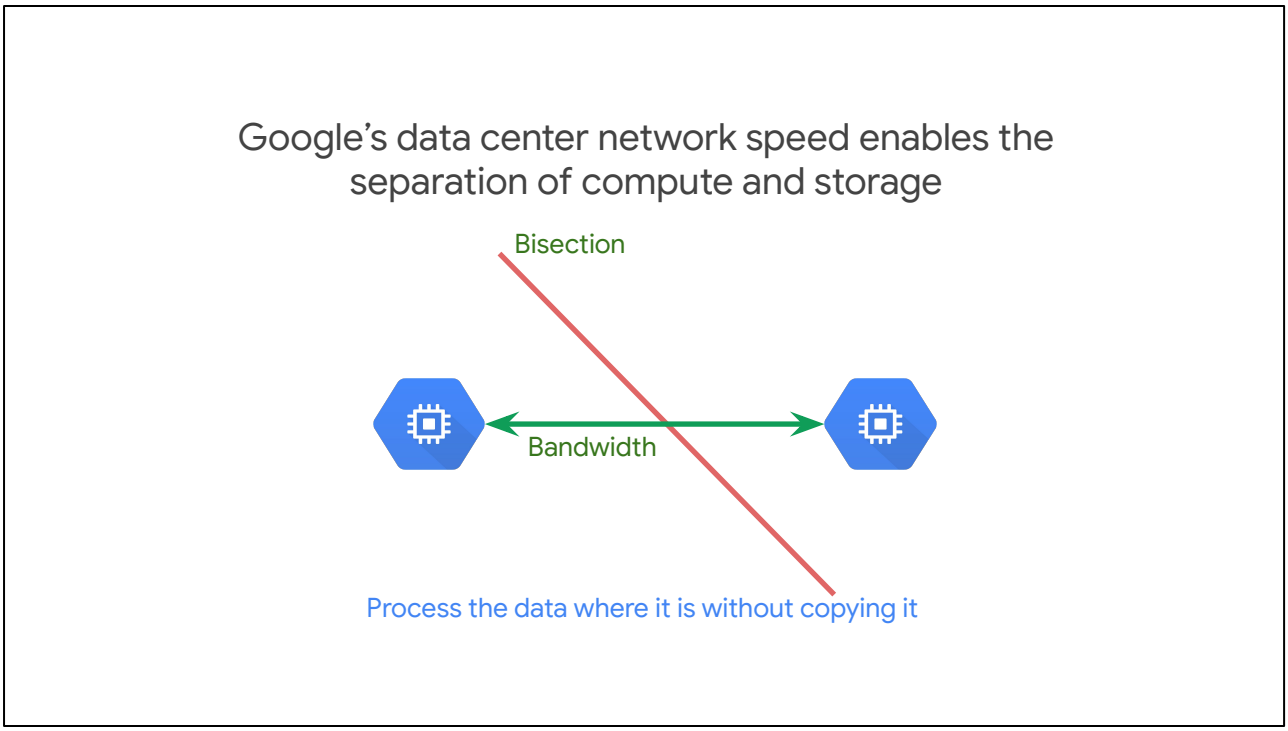
Store data persistently in Google Cloud Storage



With the notion of turning down clusters you may be worried about all the data that is currently stored on disk HDFS on-premise. What happens to that with our new Google Cloud Platform architecture?

Your data is not stored in the cluster!
The data is now stored off-cluster in Google Cloud Storage buckets.

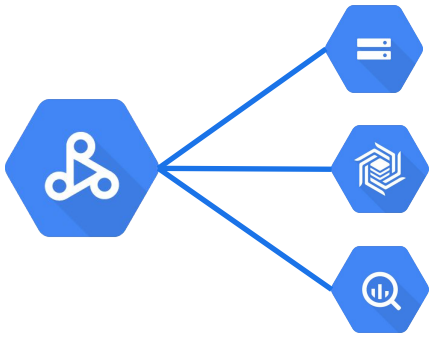
Really? Won't that make things … slow? … to reach out across a network each time the cluster needs some data?



Recall that earlier we mentioned Google's data center bandwidth between compute and storage. If you draw a line somewhere in a network, bisectional bandwidth is the rate of communication at which servers on one side of the line can communicate with servers on the other side. With enough bisectional bandwidth any server can communicate with any other server at full network speeds. With petabit bisectional bandwidth, the communication is so fast that it no longer makes sense to transfer files and store them locally. Instead, it makes sense to use the data from where it is stored.

Off-cluster storage is the gateway to efficiency

More utilization options are available if persistent data is stored off-cluster

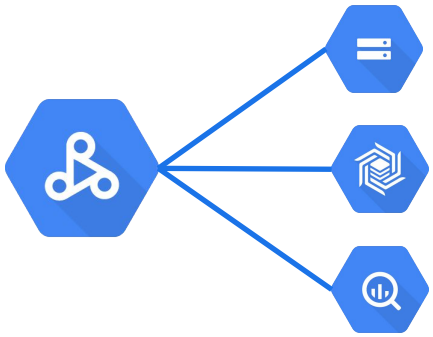


So, here's the plan.

We'll use HDFS on the cluster for *working* storage during processing, but store all actual input and output data on Google Cloud Storage. Because the data is off-cluster, the cluster can be created for a single job or type of workload and can be shut down when not in use.

Off-cluster storage is the gateway to efficiency

More utilization options are available if persistent data is stored off-cluster

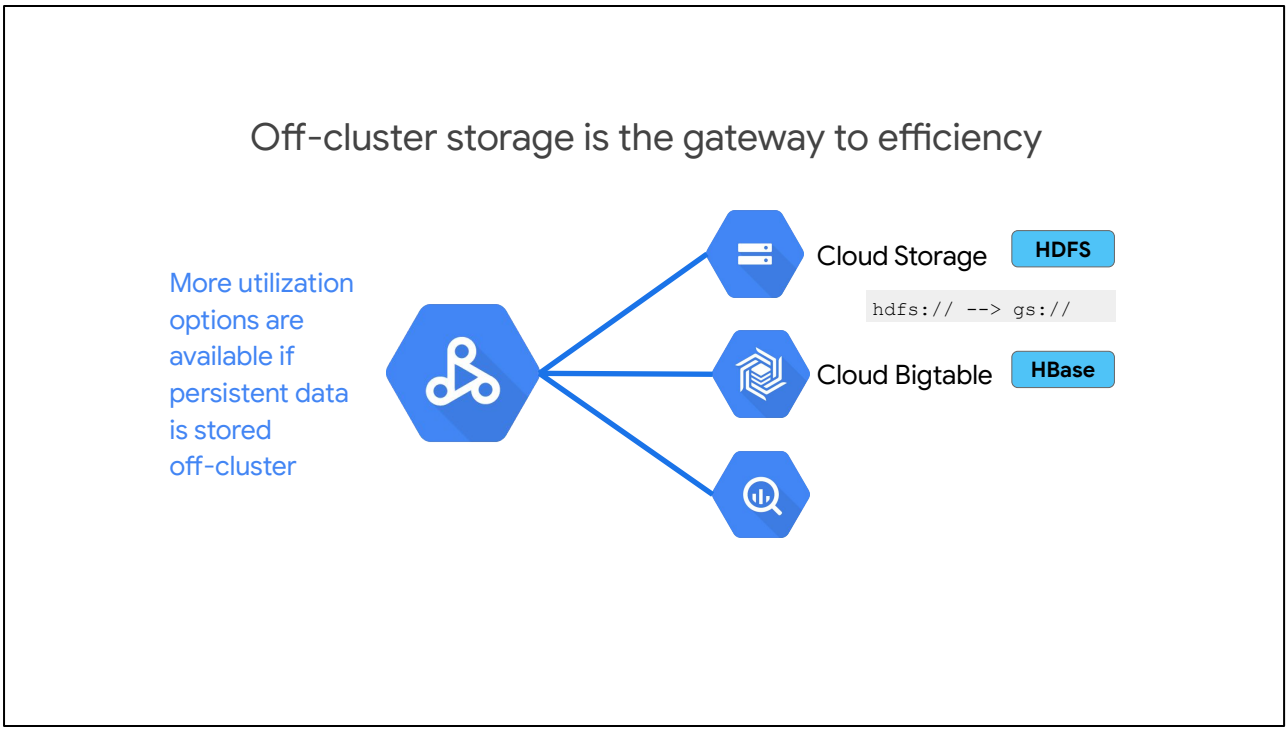


Cloud Storage

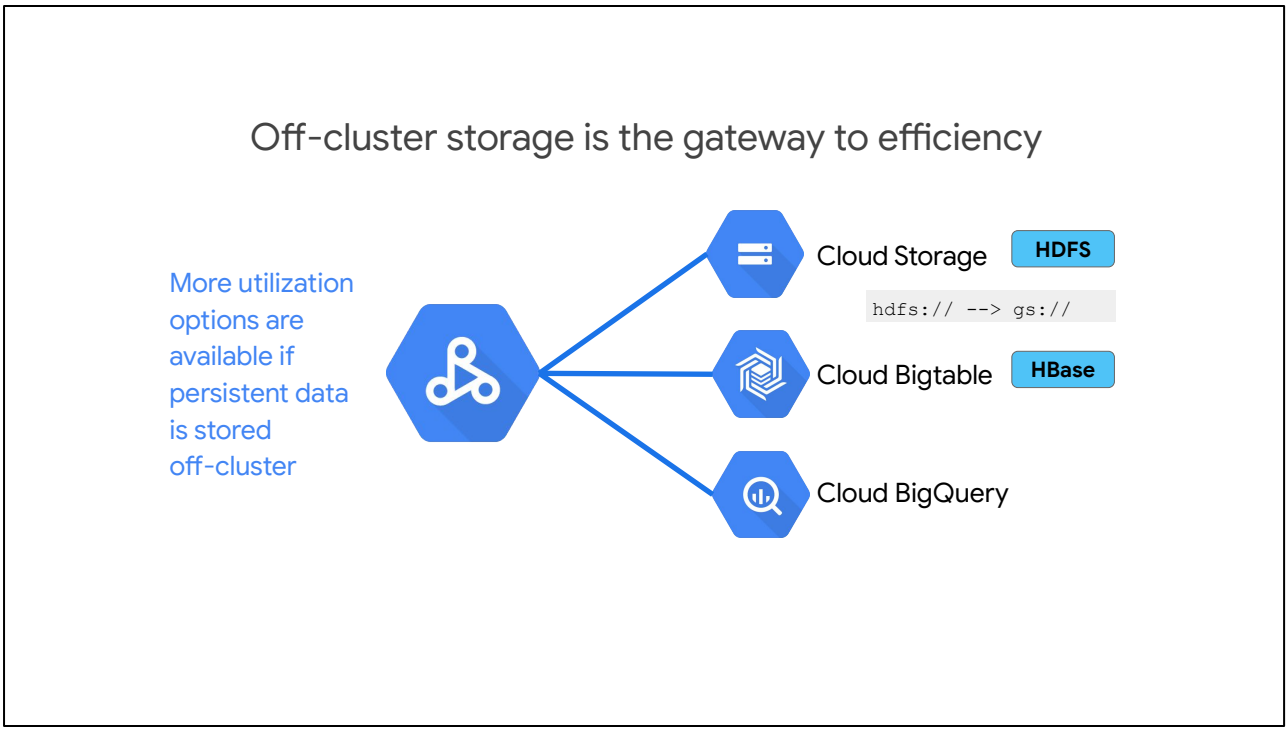
HDFS

```
hdfs:// --> gs://
```

Changing your code that works on-prem to have it work with the data on Cloud Storage is easy. Just replace hdfs:// in your Spark or Pig code with gs://. This will make the Spark or Pig job read or write to Cloud Storage.



There is also an HBase connector for Cloud Bigtable,



and BigQuery connector to work with data in the analytics warehouse.

Dataproc and Google Cloud Storage are tightly coupled with other Google Cloud products. Storing your data off-cluster opens it up to many more utilization options.

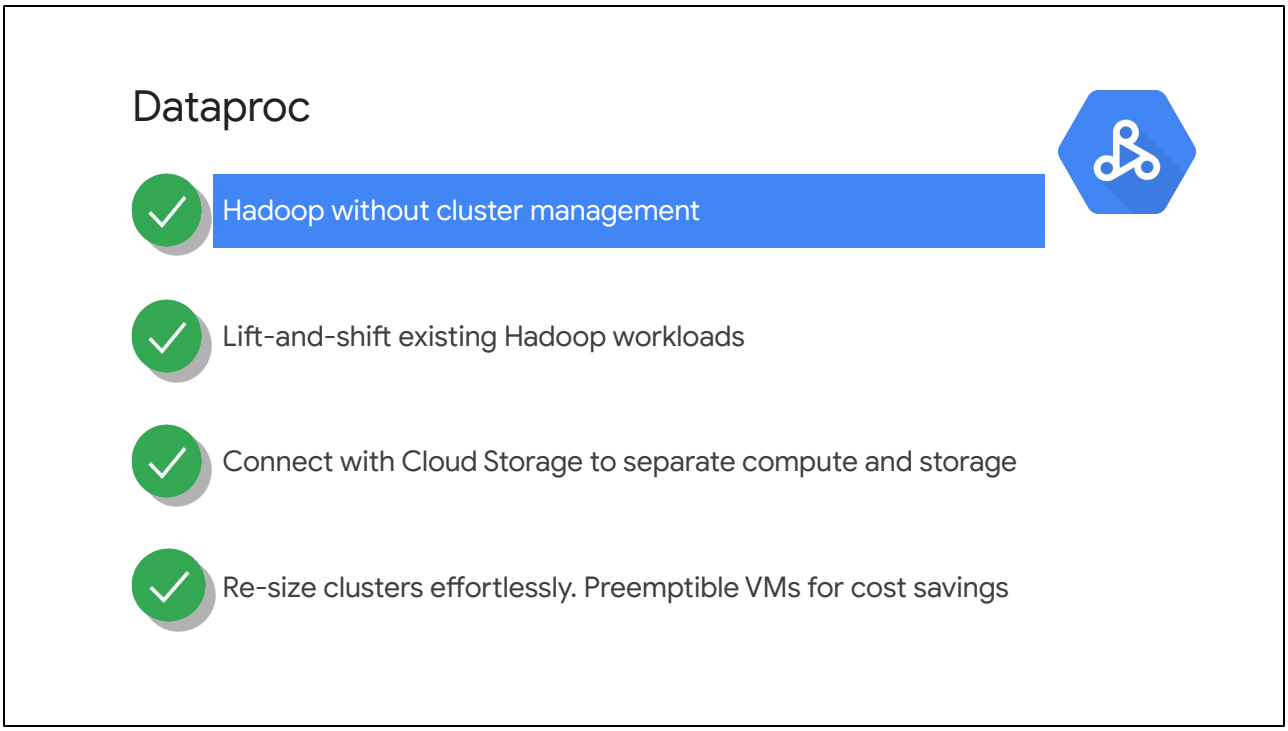
Not to mention storing it off-cluster in Cloud Storage is generally cheaper since (a) disks attached to compute instances are expensive in and of themselves and (b) if your data are off-cluster, you get to shut down the compute nodes when you are not using them.

Bottom line: store your data in Cloud Storage. Friends don't let friends use HDFS.

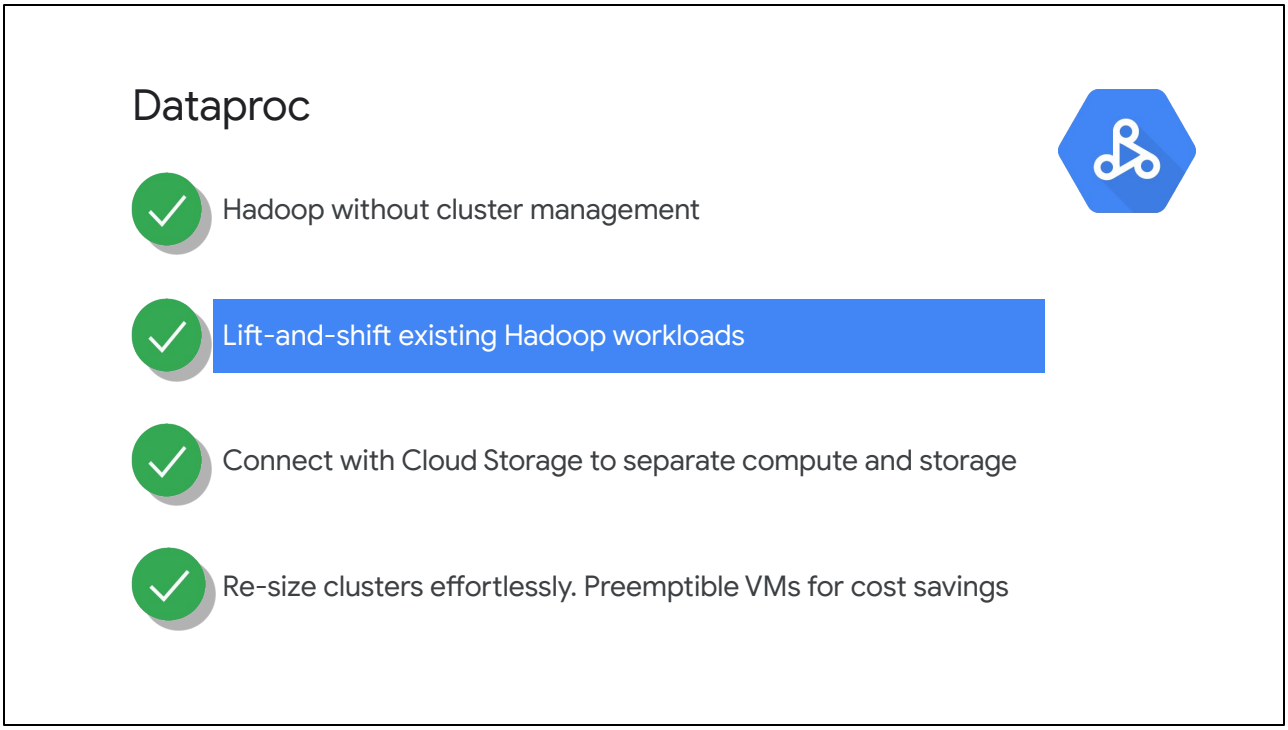
Dataproc

- Hadoop without cluster management
- Lift-and-shift existing Hadoop workloads
- Connect with Cloud Storage to separate compute and storage
- Re-size clusters effortlessly. Preemptible VMs for cost savings

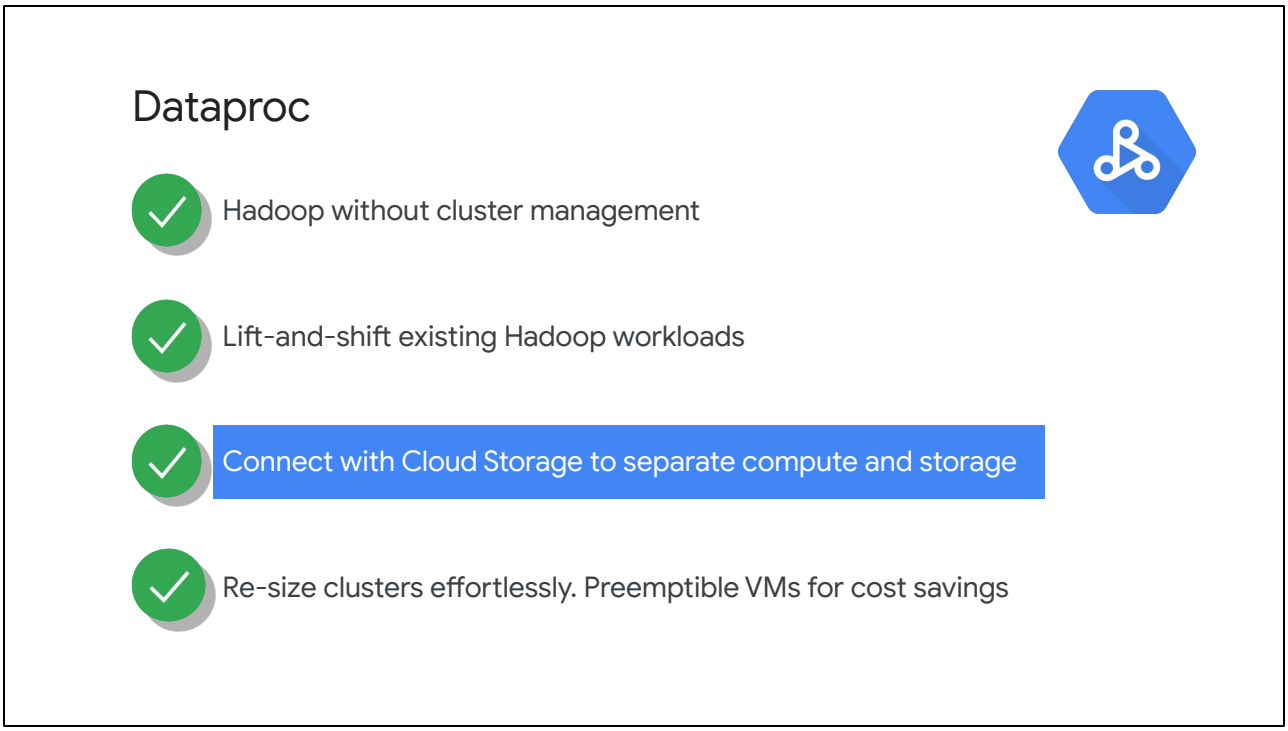
So let's recap what we've covered.



You can get a Cloud Dataproc cluster up and running in about 90 seconds. This gives you all the power of Hadoop without having to manage clusters.

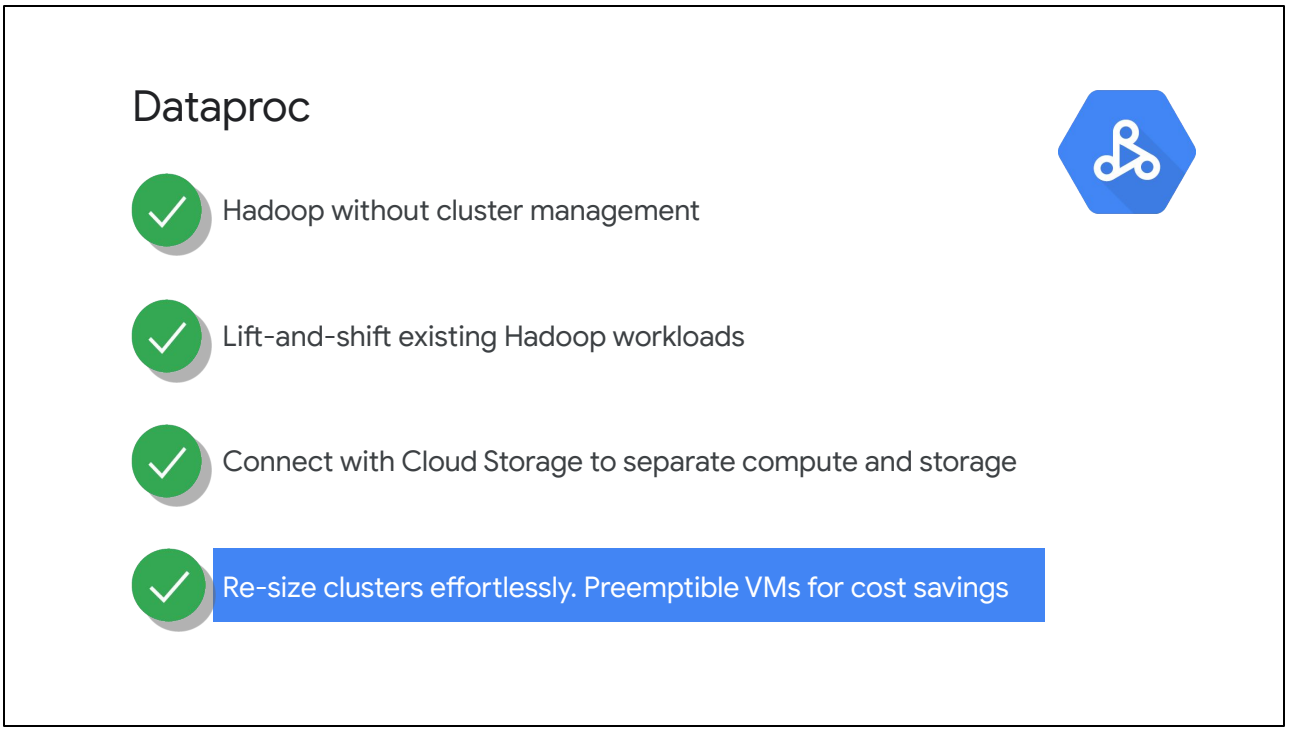


As you saw, you can lift-and-shift your existing Hadoop and Spark workloads by simply replacing hdfs:// urls with gs:// urls.



You can connect Cloud Dataproc to Google Cloud Storage and unlock the benefits of both scale and cloud economics.

You get to provision a cluster per job if you want to, and shut down the cluster when the job is done.



Lastly, your clusters are customizable which could include autoscaling and Preemptible VMs for cost savings.

Lab: Product Recommendations using Cloud SQL and Spark

What housing rentals should I recommend to my customers based on their history?

Cloud SQL

Cloud Dataproc

It's now time for your lab.

You're in charge of migrating your companies existing ML workload for housing recommendations from their on-premises Hadoop cluster to the cloud. Your organization is happy with their current model but the underlying infrastructure on-premise is causing them headaches to tune and utilize efficiently.

Your CTO wants as little friction as possible from your existing Hadoop on-premise infrastructure but has heard of the advantages cloud solutions offer for autoscaling and serverless management

Objectives

- Create Cloud SQL instance and populate tables
- Explore the rentals data using SQL statements from Cloud Shell
- Launch Dataproc
- Train and apply machine learning model written in PySpark to create product recommendations
- Explore inserted rows in Cloud SQL

Here are your lab objectives:

Objectives

- Create Cloud SQL instance and populate tables
- Explore the rentals data using SQL statements from Cloud Shell
- Launch Dataproc
- Train and apply machine learning model written in PySpark to create product recommendations
- Explore inserted rows in Cloud SQL

Create Cloud SQL instance and populate tables

Objectives

- Create Cloud SQL instance and populate tables
- Explore the rentals data using SQL statements from Cloud Shell
- Launch Dataproc
- Train and apply machine learning model written in PySpark to create product recommendations
- Explore inserted rows in Cloud SQL

Explore the rentals data using SQL statements from Cloud Shell

Objectives

- Create Cloud SQL instance and populate tables
- Explore the rentals data using SQL statements from Cloud Shell
- Launch Dataproc
- Train and apply machine learning model written in PySpark to create product recommendations
- Explore inserted rows in Cloud SQL

Launch Dataproc

Objectives

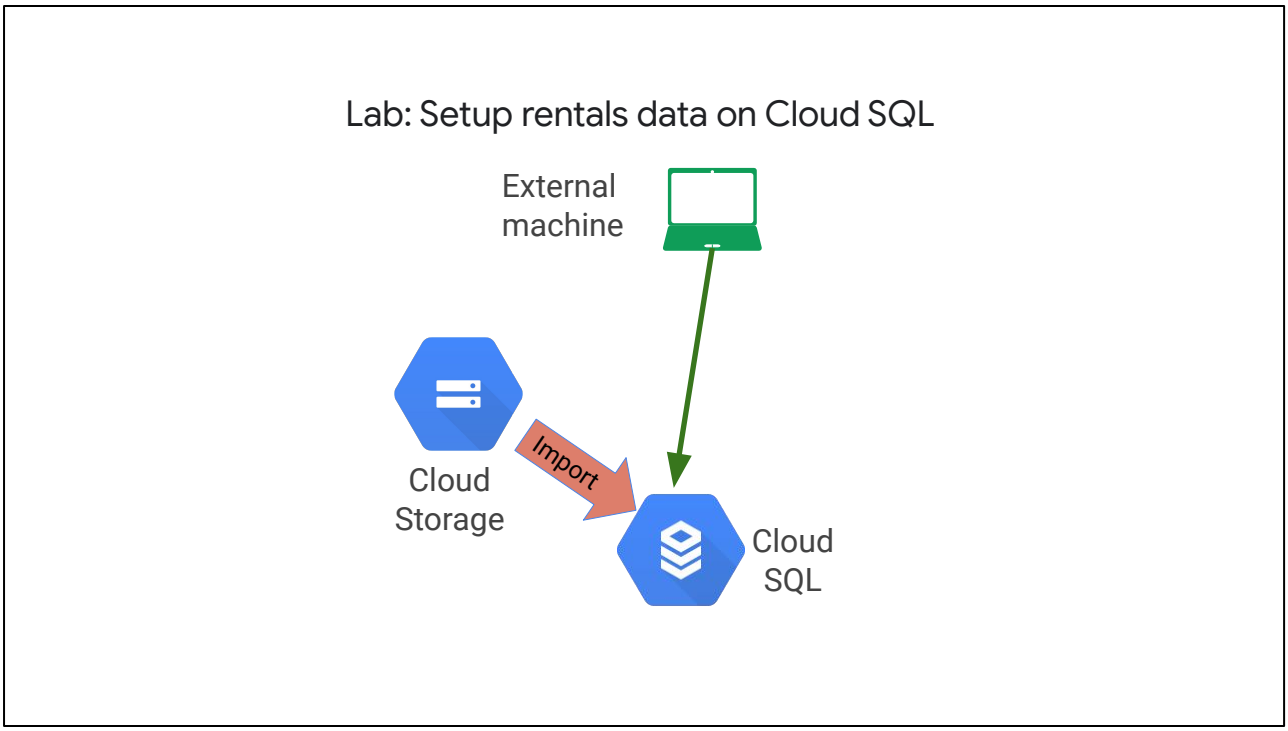
- Create Cloud SQL instance and populate tables
- Explore the rentals data using SQL statements from Cloud Shell
- Launch Dataproc
- Train and apply machine learning model written in PySpark to create product recommendations
- Explore inserted rows in Cloud SQL

Train and apply machine learning model written in PySpark to create product recommendations

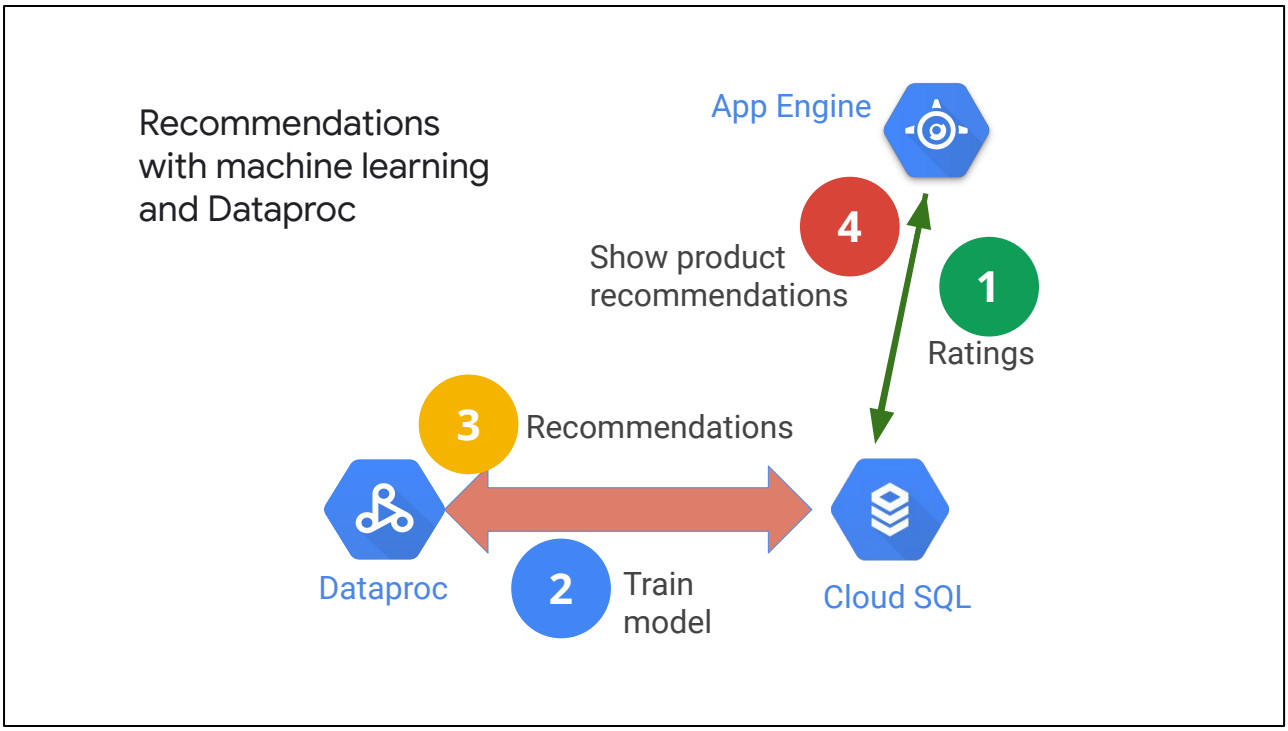
Objectives

- Create Cloud SQL instance and populate tables
- Explore the rentals data using SQL statements from Cloud Shell
- Launch Dataproc
- Train and apply machine learning model written in PySpark to create product recommendations
- Explore inserted rows in Cloud SQL

Explore inserted rows in Cloud SQL

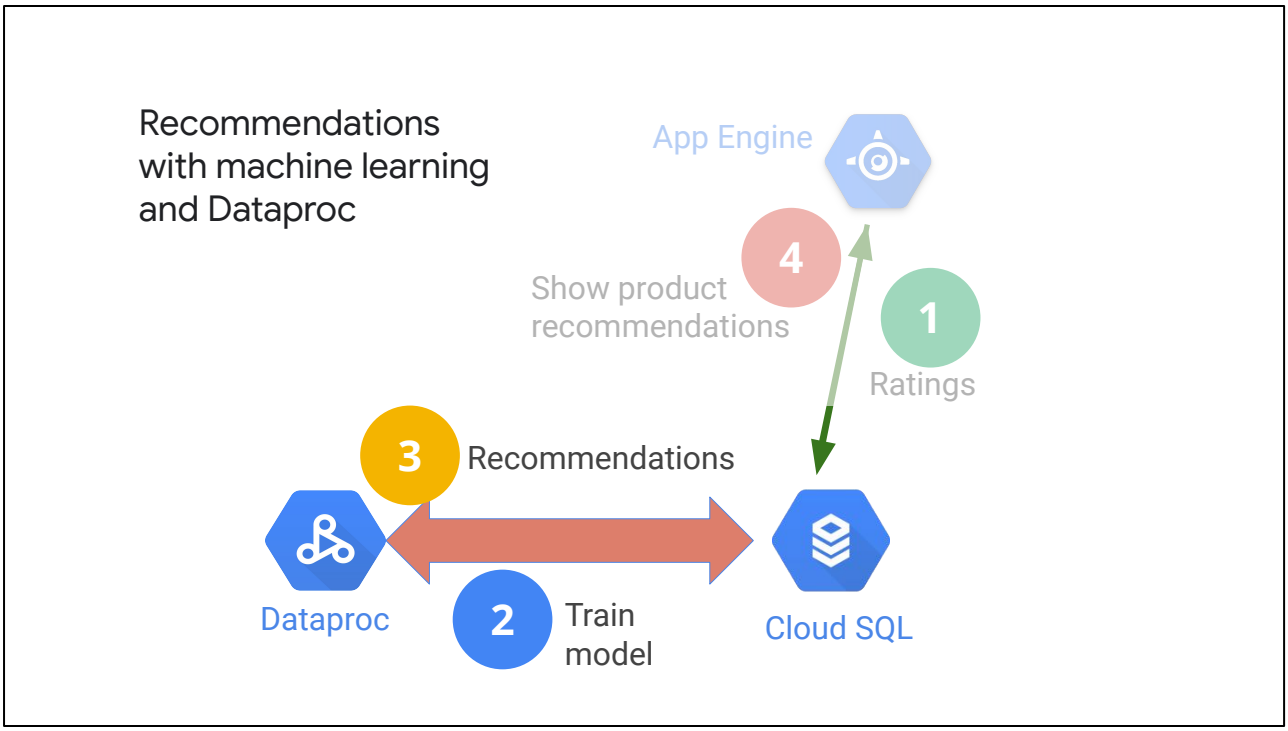


Here's what the setup will look like. We setup GCS and Cloud SQL and then import the records from GCS



Now that your ratings are in Cloud SQL from your application as step 1,
In step 2 you will run a machine learning training job in Cloud Dataproc to build the model
In step 3 you will will run the model in Cloud Dataproc to create recommendations and save the top 5 recommendations for each user into Cloud SQL
Lastly, your ratings can be delivered back to your application running on App Engine.

If you want a fully complete solution with a website that you can demo, go to https://cloud-training-demos.appspot.com/ and see the App Engine code in https://github.com/GoogleCloudPlatform/spark-recommendation-engine/tree/master/appengine



You won't do Step 4 in this lab because those steps deal mostly with web programming. In this lab, you concentrate on doing steps 2 and 3.

Try out the lab and keep in mind you have multiple attempts for each lab so you can always come back and practice more.