Google Cloud

Running Dataproc jobs

Data Engineering on Google Cloud Platform

Notes:

42 slides + 1 lab

Cloud Dataproc provides compelling reasons to run open-source tools on GCP

- Stateless clusters in <90 seconds MODULE 1
- Supports Hadoop, Spark, Pig, Hive, etc.
- High-level APIs for job submission
- Connectors to Bigtable, BigQuery, Cloud Storage

Notes:

We have already looked at #1.

Let's look at #2 and #3 here. Starting with #2.

Agenda

Running jobs + Lab

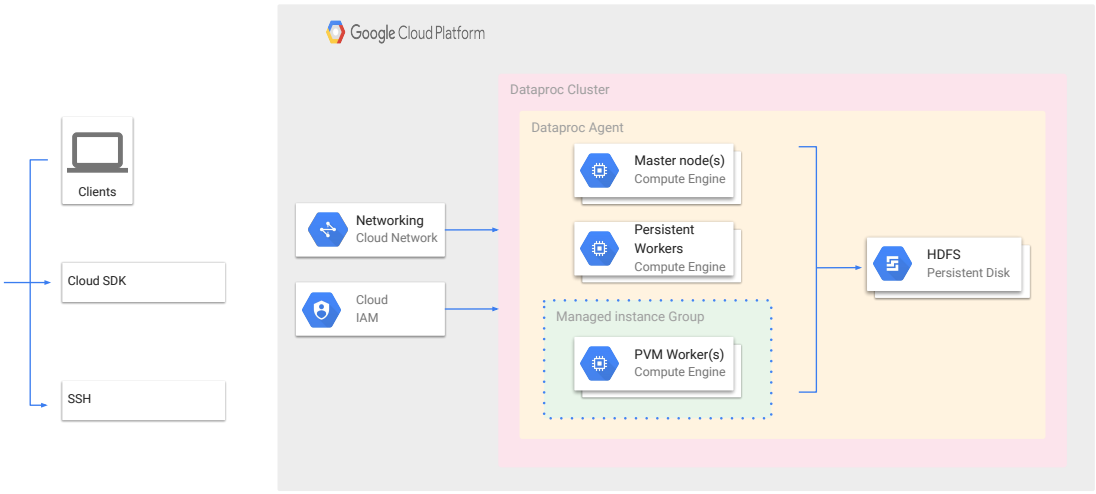
Separation of storage and compute

Submitting jobs

Spark RDDs, Transformations, and Actions + Lab

Cluster management and monitoring

Can SSH to cluster and run Pig/Spark



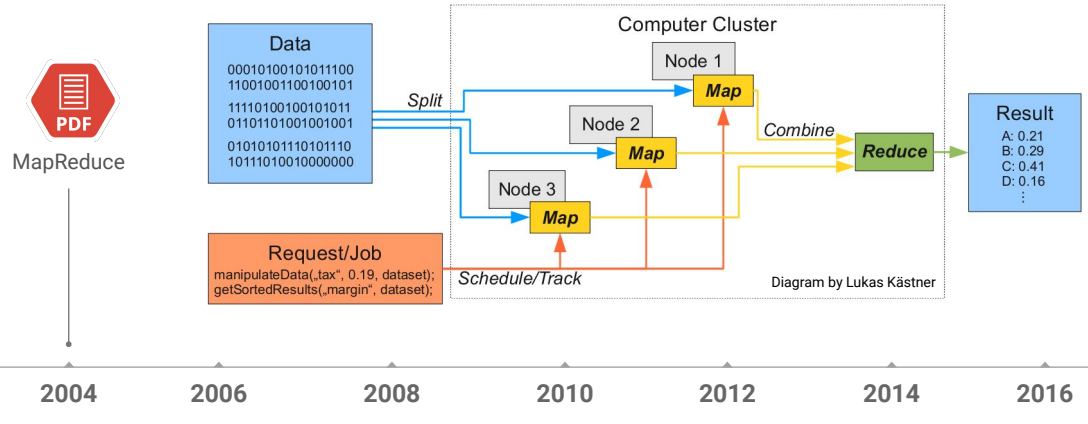
Lab 2: Work with structured and semi-structured data

- Learn about tools for working with structured and semi-structured data
- Use the Hive CLI
- Hive is used for structured data, similar to SQL
- Run a Pig job
- Pig is used for semi-structured data, similar to SQL + scripting

Agenda

Separation of storage and compute

MapReduce approach splits Big Data so that each compute node processes data local to it

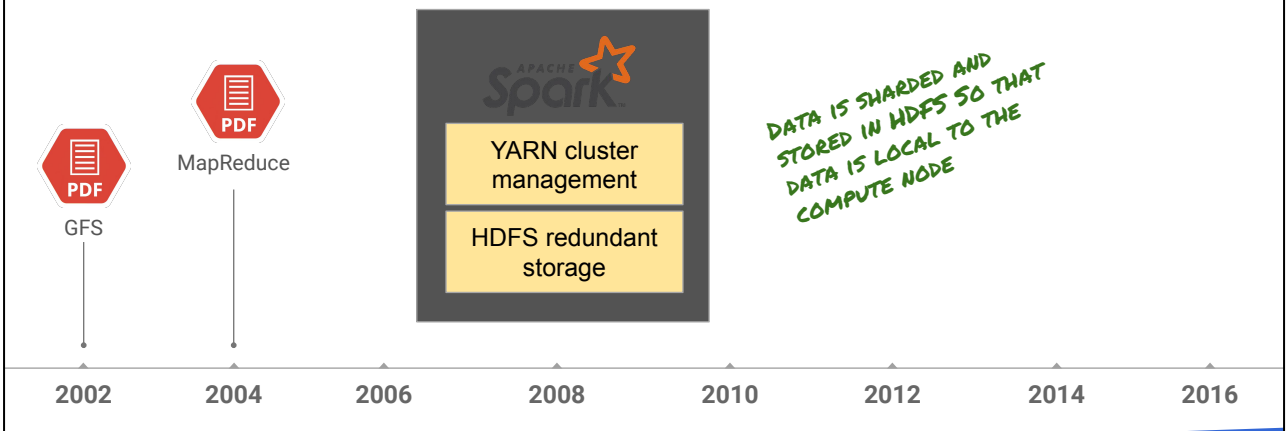


Notes:

This slide was also in Chapter 1, so just mention that they've already seen this.

Diagram source: https://www.flickr.com/photos/lkaestner/4861146813
cc-by-sa Lukas Kastner

To get data local to the machine, you pre-shard the data onto Hadoop Distributed File System



Notes:

HDFS is based on the 2002 paper from Google on Google File System.

Compute and Storage are closely tied in traditional MapReduce architecture

Scenario	What needs to happen?
Compute node needs to be replaced	?
Append new year of data	?
?	?
?	?

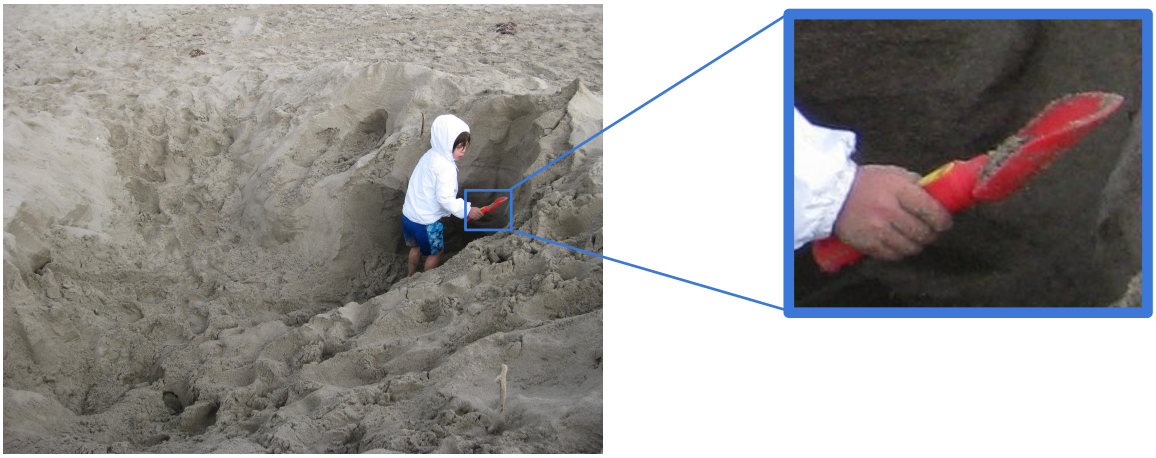
Notes:

Ask the class what problems this leads to. Have them think about a scenario.

Example scenario: You split your data into 10 nodes. One node goes down. You bring in a new replacement. What has to happen? At least some part of the data needs to be copied onto the new nodes before jobs have to be partitioned.

Example scenario: The data changes (maybe you need to change formats or append a new year of data). What needs to happen?

Need to provision increased resources quickly

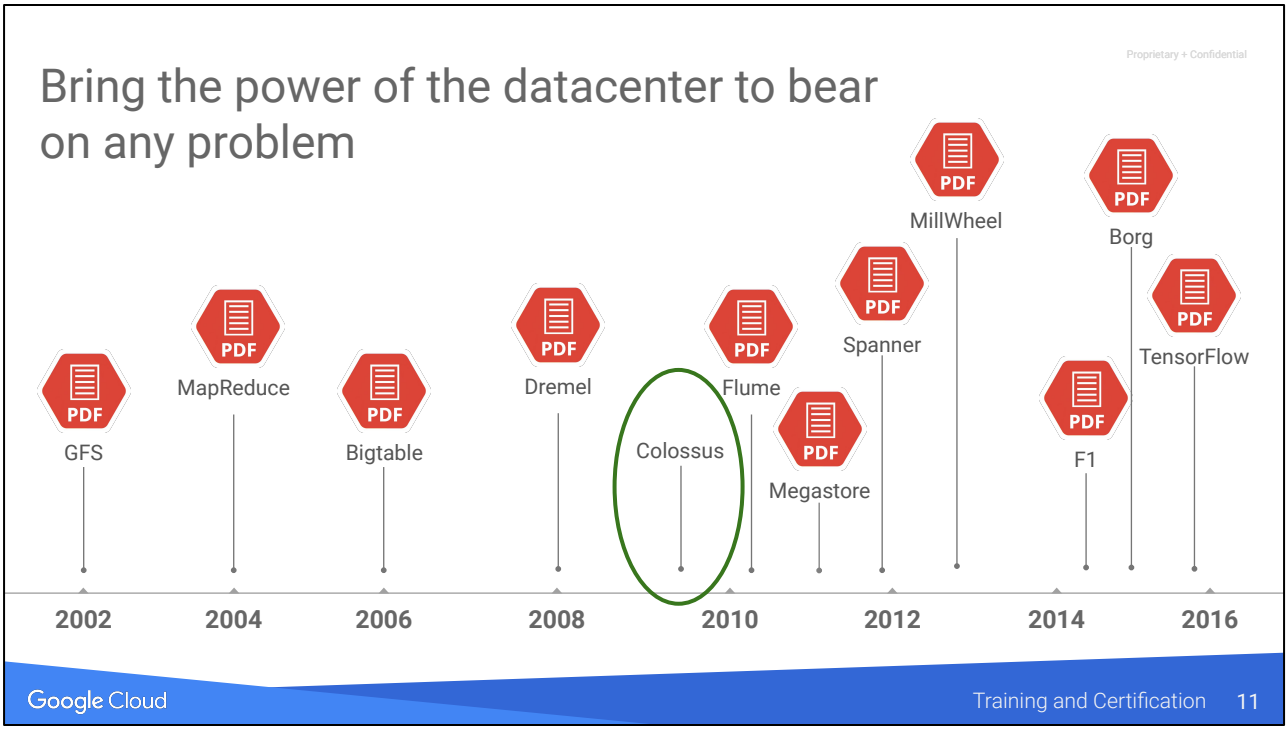


Notes:

Beyond the obvious question of deciding how much to provision, the larger issue is that of the risk/cost of experimentation.

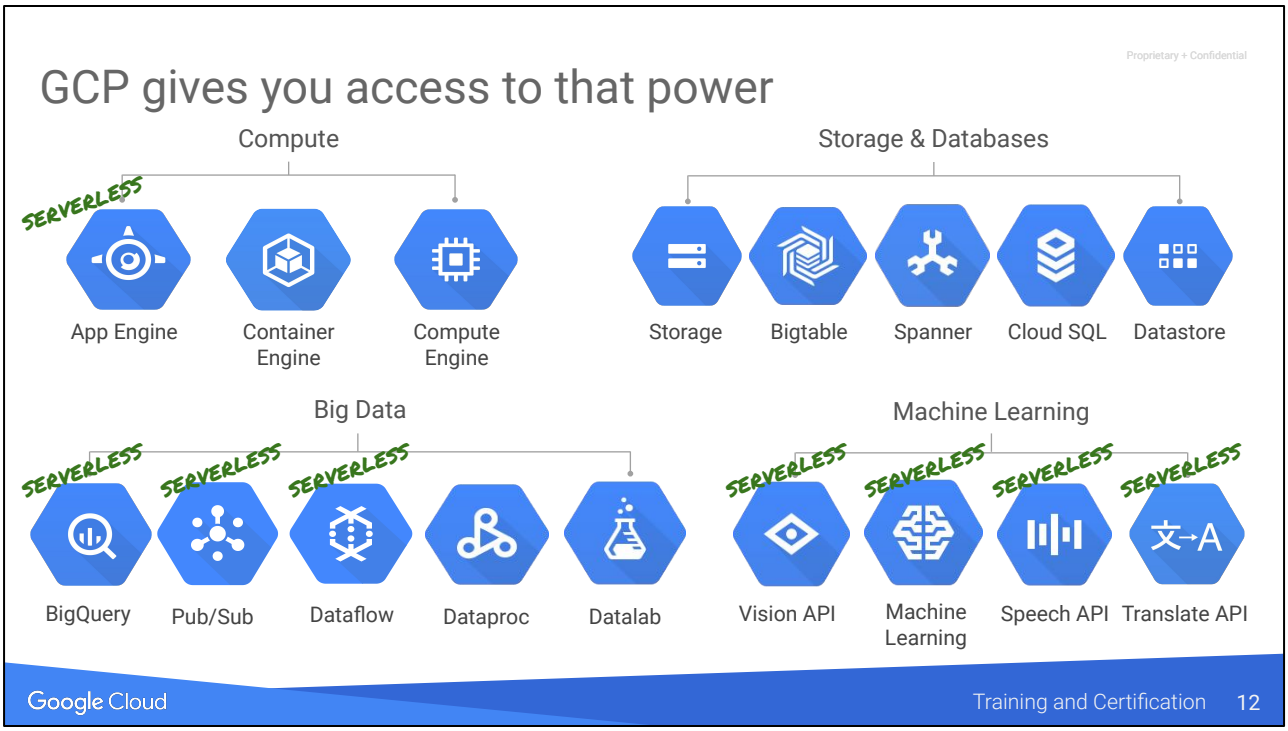
If they knew for sure what they have to do, how to do it, and what resources they need for it, enterprises would be able to provision large amounts of resources. But the reality is that you have to experiment. It's not about how much you can invest, it's how quickly you can iterate.

Image source: I (vbp@) took it myself.



Notes:

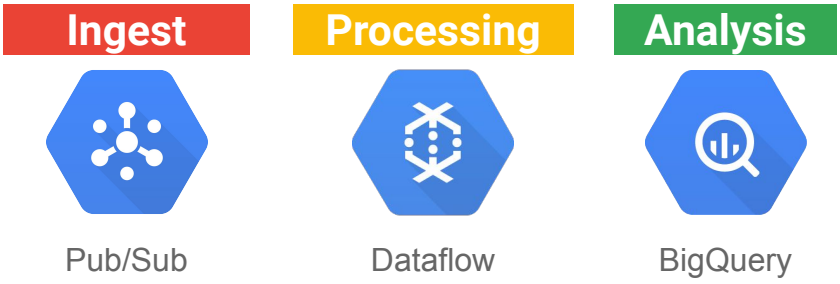
Colossus, the replacement for GFS, is the key innovation and led to a bunch of serverless offerings. It's in our datacenter and enables you to not have to shard data. Instead, you have a global filesystem that offers petabit/second bisection bandwidth. Yes, Colossus has not been published … public information on it is scarce and consists of an unofficial copy of a slide deck by Andrew Fikes:
https://www.systutorials.com/3306/storage-architecture-and-challenges/



Notes:

The Big Data and ML offerings are serverless (except for Dataproc & Datalab because they are based on OSS -- Hadoop ecosystem and Jupyter respectively). The APIs are of course serverless although you tend not to think of them as serverless offerings.

GCP gives you serverless platform for all stages of the analytics data lifecycle



Notes:

In particular, the three Big Data serverless offerings are ...

Serverless data processing is about speed, low cost, and freedom

Speed to insights

Focus on insights

Not administration

Low cost

Practically infinite scale, exactly when you need it

Pay only for what you use

Freedom to experiment

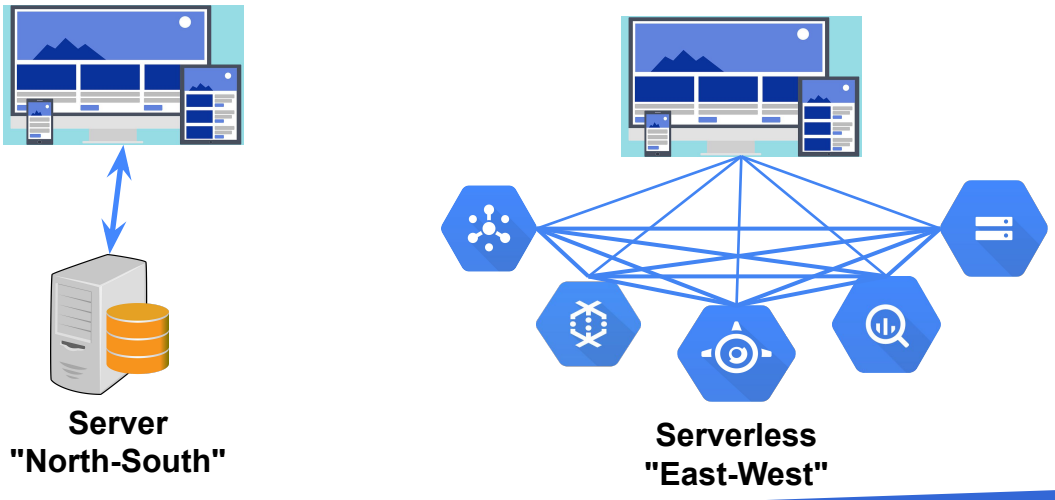
Experiment, fail quickly, and iterate

Successful experiments are ready to go live right away

Notes:

Summary slide for this section: Not just about low-cost, but also about speed and freedom.

Networking to support new software and data methods



Networking within a data center used to be focused on transactions between a client and a server, what is called "North-South" communications. For example, a user might request a web page, and the server generates the web page. In that paradigm, the network needs to support low latency transactions between the user and the server.

Networking for cloud applications must account for distributed data and "serverless" services. (Serverless, of course, meaning that the servers, if they exist, are not exposed to the client/user. The client only has visibility to an API, not to the VMs.) In the web page example, the single web page might be built from data drawn from many services. Beneath those services might be communications between thousands of servers. So very fast server-to-server communication inside the network becomes a most important network design criteria. This is called "East-West" communications.

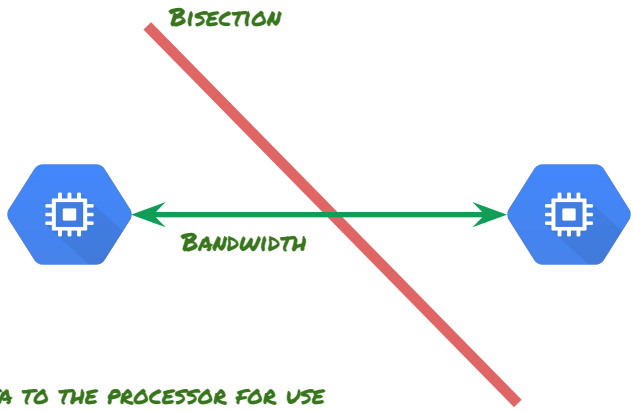
One way of measuring this or describing "East-West" communications is "bisectional bandwidth".

Google Cloud's internal network supports "petabit bisection bandwidth".

More on this subject:
http://highscalability.com/blog/2015/8/10/how-google-invented-an-amazing-datacenter-network-only-they.html

https://pixabay.com/en/server-web-network-data-computer-567944/
https://pixabay.com/en/responsive-web-pages-websites-1622825/

Why high bisection bandwidth is a game-changer

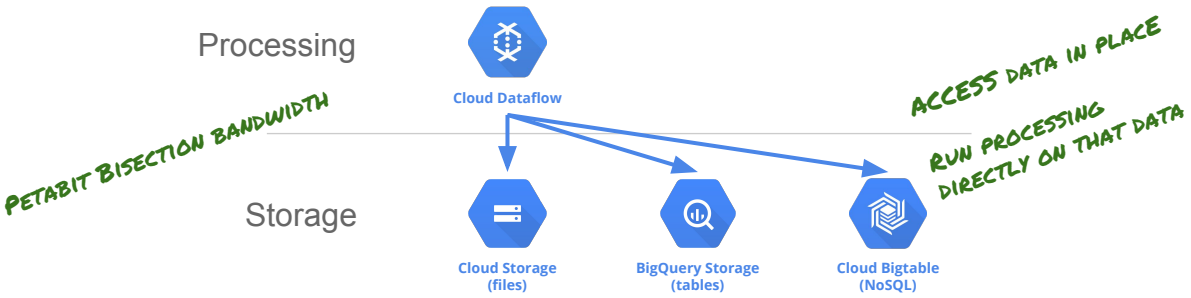


First wave: Copy the data to the processor for use
Second wave: (Hadoop) Distribute the data and process it in place
Third Wave: (Google Cloud) Use the data where it is without copying it

Bisectional bandwidth

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.

Separation of Storage and Compute is what enables Serverless to work



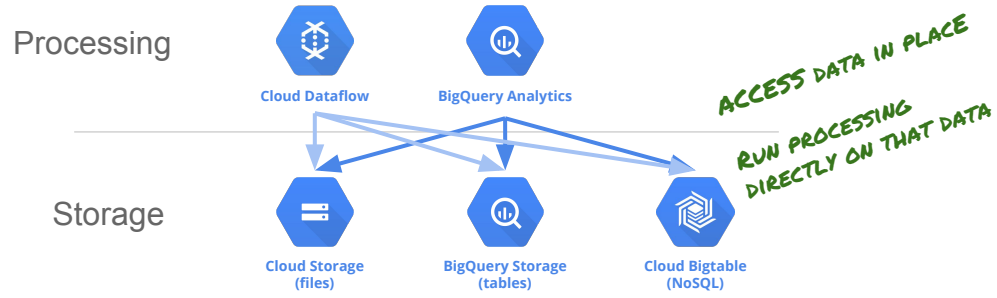
Notes:

Separation of storage and compute is what enables “serverless to work” -- Dataflow can read use any of these as source/sink. This sort of direct read is efficient because of very high sustained read speed from Cloud Storage -- any two computers in data center are connected by very fast network.

Keep as much data as you want, economically.

Share data in place, no more FTP and copying.

BigQuery also separates compute and storage



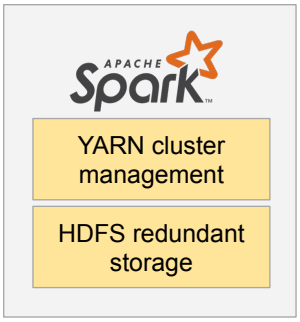
Notes:

Access any storage system from any processing tool.

Compare and contrast bq's data separation w/ typical DB storage mgmt system.

BigQuery is "just" a query engine. It can query csv files on cloud storage also, for example.

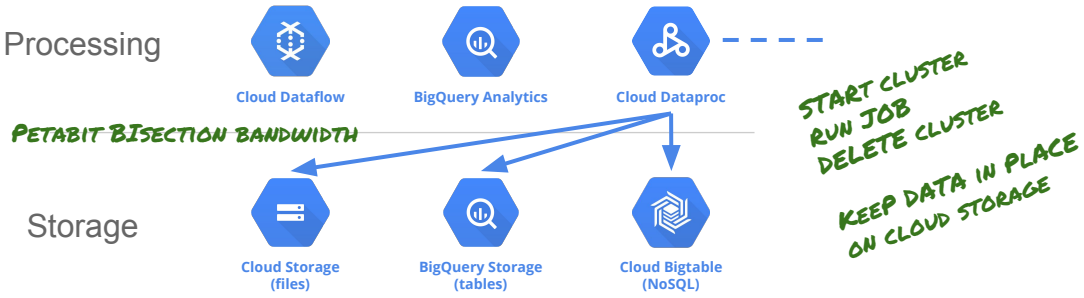
Can I run Spark and still get separation of storage and compute?



Notes:

But what if I want to run Spark programs? How can I get separation of compute & storage. Spark runs on Hadoop … and Hadoop is a cluster-aware piece of software … we need to take our data and split it into pieces and store them on cluster so that data is local to compute … but then we are limited by the number of processing nodes or the number of storage nodes. These are not independent

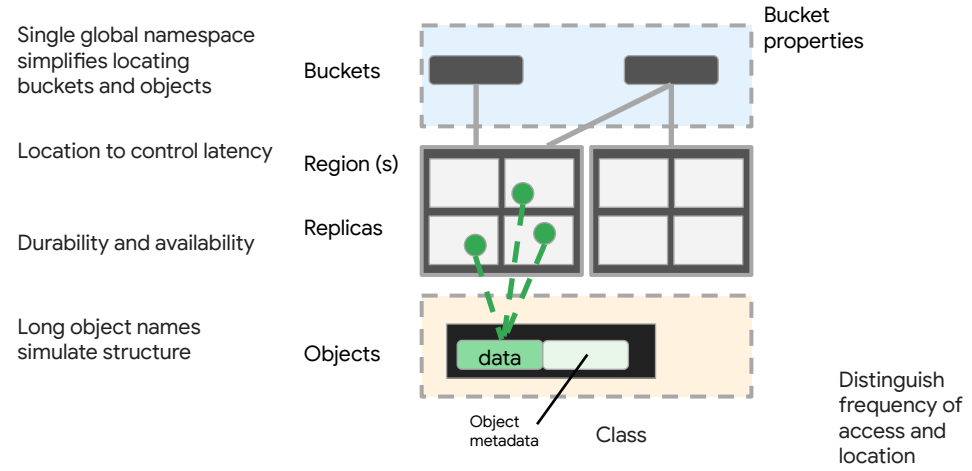
Cloud Dataproc provides the ability for Spark programs to also separate compute & storage



Notes:

Just change all your input urls from hdfs:// to gs:// …
The reason you can do this is the speed of the inter-networking ….
Cloud Dataproc is Spark/Hadoop “the Cloud way”
Deploy cluster in ~90 seconds
Pay by the minute

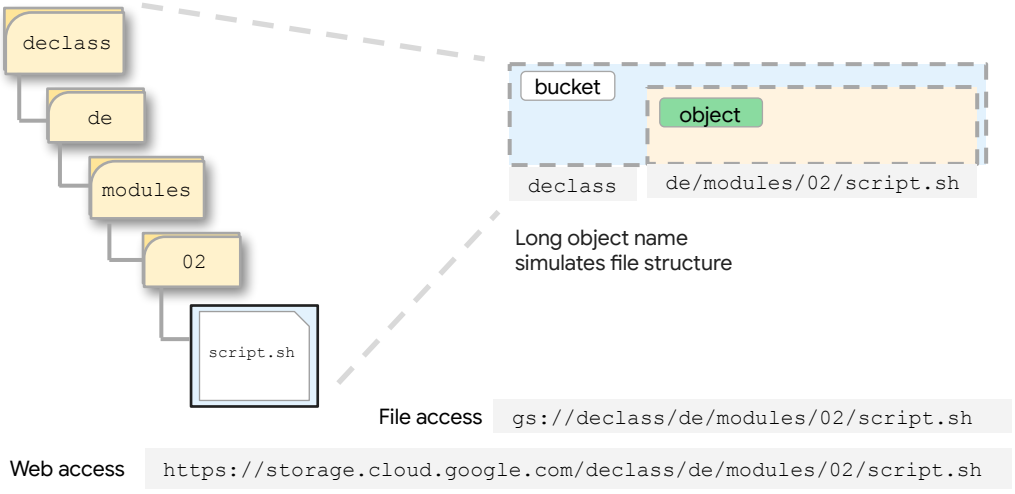
How does Cloud Storage work?



Cloud Storage is an object store. A lot of its amazing properties have to do with the fact of what's missing -- that it is not a file system.

A global namespace for buckets simplifies locating a specific bucket. Buckets are either multi-regional or regional. Objects within a bucket are replicated across zones. That gives the objects durability and availability that are not possible in a physical storage device like a disk. Objects have several properties, among them is "class" which declares the frequency of anticipated access of the object. This allows the Cloud Storage service to distinguish between types of objects and internally manage them to service levels.

Cloud Storage simulates a file system



Cloud Storage simulates a file system by enabling forward slashes to be used as part of the object name. In the example above, the object is named "de/modules/02/script.sh" And because of the consistent use of slashes in the name, it is possible to simulate a file in a directory.

This allows objects to be addressed using file access and web access protocols.

The simulation does not work identically to a file system. For example, when you move a "file" like script.sh from one directory to another, what is really happening is the object is being renamed. This makes Cloud Storage faster for some operations and slower for others when compared with a file system.

https://storage.cloud.google.com uses TLS (HTTPS) to transport your data which protects credentials as well as data in transit. Avoid the use of sensitive information as part of bucket or object names.

Using Cloud Storage with Cloud Dataproc

Directories are simulated, so renaming a directory involves renaming all the objects

Objects do not support "append"

Cloud Storage is a distributed service

Eliminates traditional bottlenecks and single points of failure

Directories are simulated, so renaming a directory involves renaming all the objects.
Objects are not files, so they don't support "append".
Cloud Storage is a distributed service. That means it eliminates traditional file system bottlenecks and Single Points of Failure (SPoFs).

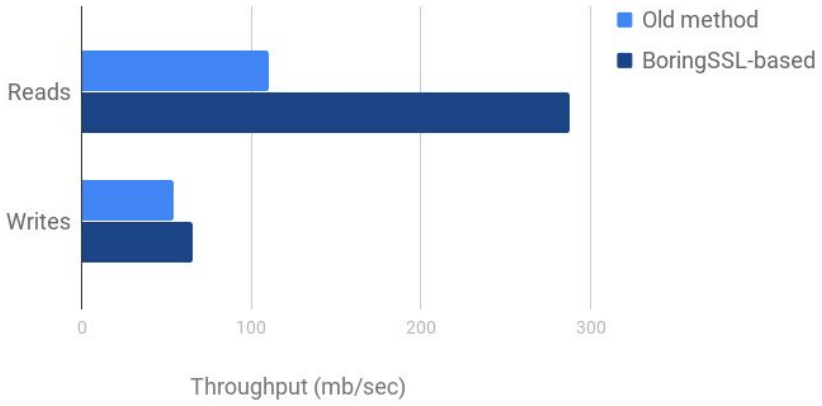
Latency is higher for data in Cloud Storage than HDFS on a Persistent Disk in the cluster.
Throughput for processing data in Cloud storage is higher than throughput for HDFS on Persistent disk in the cluster.

Cloud Storage is very good at bulk and parallel operations on larger objects. So for performance, keep these in mind:

- Avoid small reads
- Use large block/file sizes where possible
- Avoid iterating over many nested directories in a single job

https://pixabay.com/vectors/magic-wand-magician-sorcerer-297332/
https://pixabay.com/photos/puzzle-match-missing-hole-blank-693873/
https://pixabay.com/illustrations/one-way-street-decisions-opportunity-1113973/

Sustained reads from Cloud Storage are even faster than before ...



Notes:

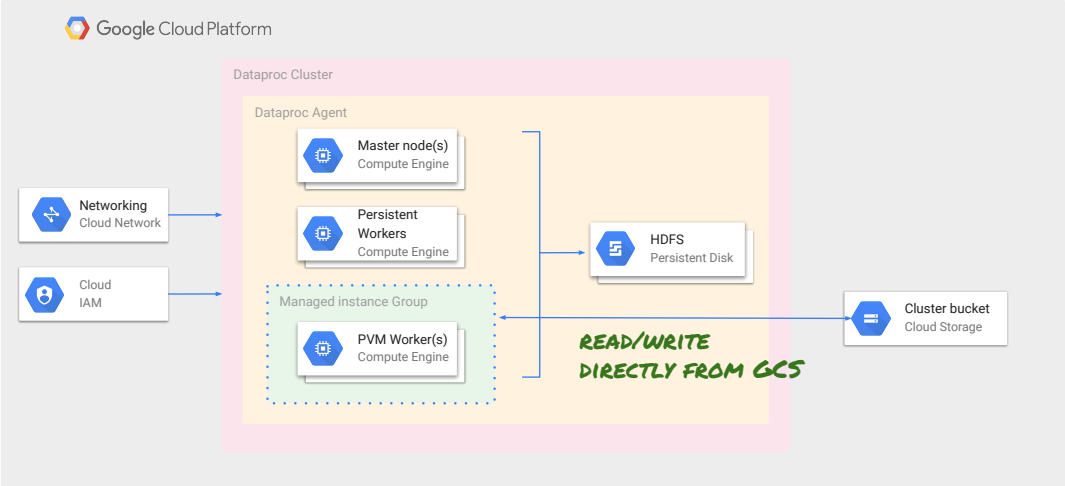
The impact of the PB/s networking and software improvements is that sustained reads are very fast. You can keep things in GCS.

Old: 2016
New: 2017

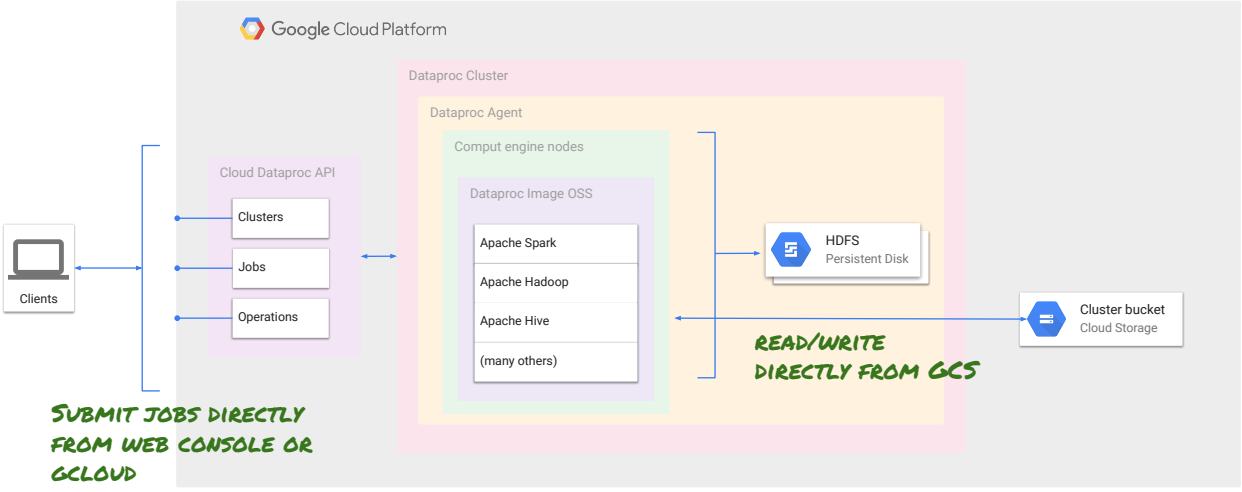
Agenda

Submitting jobs

Cloud Dataproc hardware architecture



Cloud Dataproc software architecture



Notes:

Can directly read/write to GCS from Spark, Pig, etc.
Submit jobs

Lift and shift work to Cloud Dataproc

1

Copy data to GCS

Copy your data to Google Cloud Storage (GCS) by installing the connector or by copying manually

2

Update file prefix

Update the file location prefix in your scripts from `hdfs://` to `gs://` to access your data in GCS

3

Use Cloud Dataproc

Create a Cloud Dataproc cluster and run your job on the cluster against the data you copied to GCS. Done

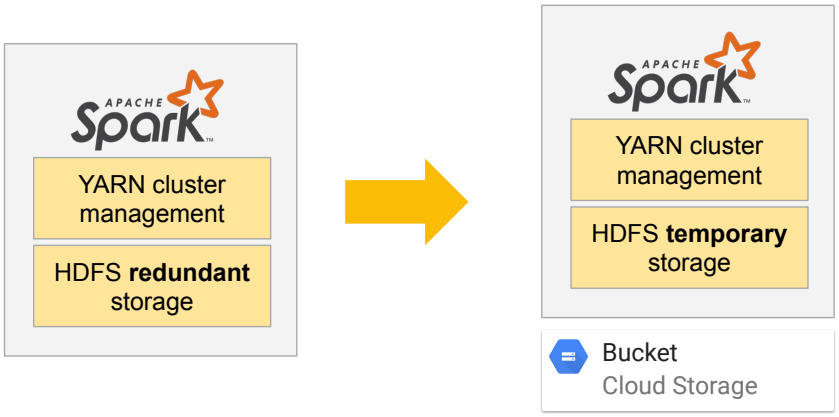
Migrating code

- In most cases, you only need to update jobs so they read from Google Cloud Storage (`gs://`) instead of HDFS

```
textFile = sc.textFile("~~hdfs~~ gs://...") # Read data

# Creates a DataFrame having a single column
df = textFile.map(lambda r: Row(r)).toDF(["line"])
errors = df.filter(col("line").like("%ERROR%"))
# Counts all the errors
errors.count()
# Counts errors mentioning MySQL
errors.filter(col("line").like("%MySQL%")).count()
# Fetches the MySQL errors as an array of strings
errors.filter(col("line").like("%MySQL%")).collect()
```

Why change hdfs:// to gs://?



Notes:

Because the cluster is temporary …. We want to be able to delete the cluster when we are done.

If the first case, HDFS is the durable storage for data. We can't delete the cluster.

In the second case, HDFS is only temporary. We can delete the cluster.

Agenda

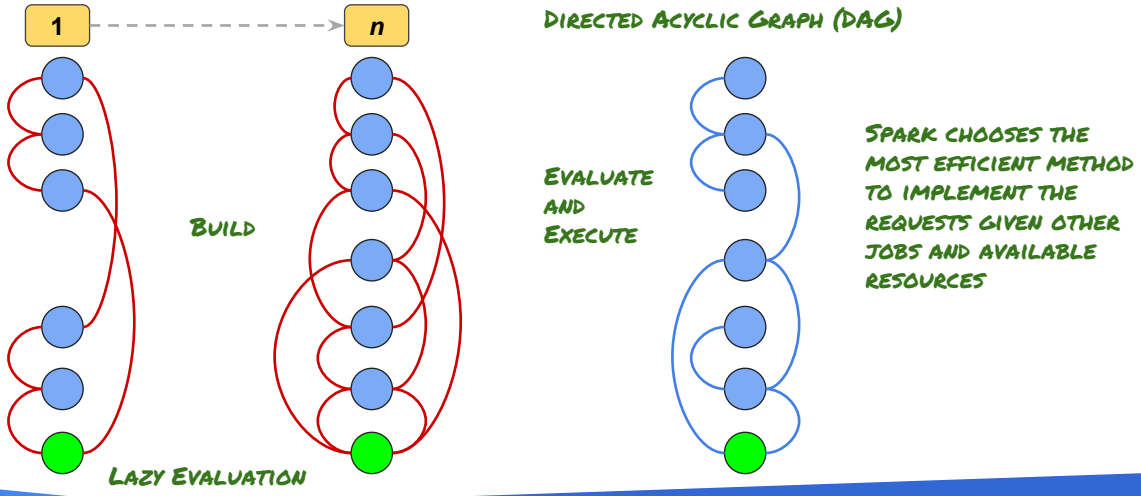
Spark RDDs, Transformations, and Actions + Lab

Spark hides data complexity with an abstraction

RDDs hide the complexity of the location of data within the cluster, and also the complexity of replication.
Spark partitions data in memory across the cluster and knows how to recover through an RDD's lineage, should anything go wrong.
Spark has the ability to direct processing to occur where there is processing resource available.
You treat your data as a single entity, Spark knows the truth.
Data partitioning, Data replication, Data recovery, Pipelining of processing -- all are automated by Spark so you don't have to worry about them.

https://pixabay.com/en/books-door-entrance-italy-colors-1655783/
https://pixabay.com/en/book-red-closed-library-education-34014/

Sometimes being lazy is more efficient than hurrying



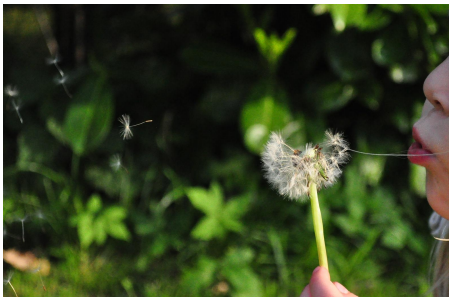
When you program in command languages, you "tell the system what to do". With Spark, you program with "requests". Spark doesn't immediately perform these actions. Instead, it stores them in a graph system called a DAG. Only when a request is submitted that requires output, does Spark actually process the data.

The benefit of this strategy is that Spark can look at all the requests and the intermediate results, and construct parallel "pipelines" based on the resources that are available in the cluster at that time. This hides a lot of complexity from the user of the service. It also allows Spark to mix different kinds of applications -- those that are more processing intensive and those that are more data intensive -- and balance the work flows. Before Spark (with MapReduce) the cluster has to be "tuned" for the kinds of applications being run.

Being lazy with RDD operations

TRANSFORMATIONS ARE "LAZY"

ACTIONS -- "DO IT NOW"



ANONYMOUS FUNCTIONS

Transformation
Input is an RDD and output is an RDD
Registered in DAG awaiting an action (lazy)

Action
Output is a result format, such as a text file
Triggers Spark to process the pipeline

Transformations and Actions are API calls that reference the functions you want them to perform.

Anonymous functions (in Python, Lambda functions) are commonly used for the following reasons:

- A self-contained way to make a request to Spark
- Lambda functions are defined "inline" making it easy to read and understand in sequence.
- They are limited to a single specific purpose.
- They don't clutter the namespace with function names for code that is only used in one place.

https://pixabay.com/en/dandelion-colorful-people-of-color-2817950/

https://pixabay.com/en/blowball-dandelion-girl-blowing-384598/

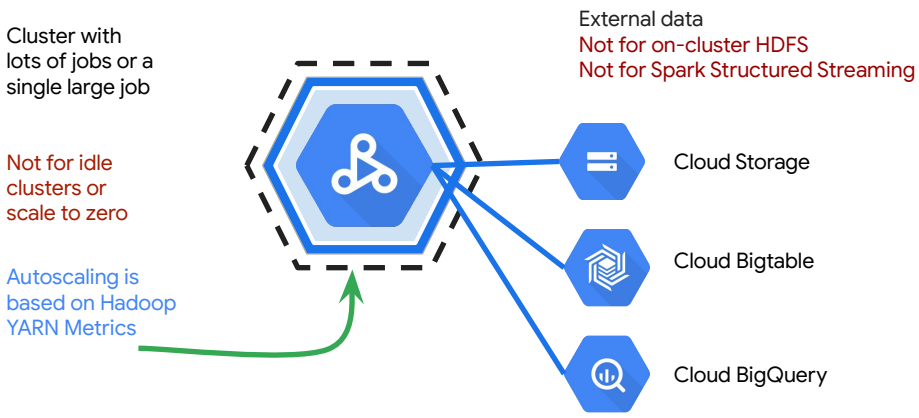
Lab 3: Work with unstructured data; Submit Dataproc Jobs; Explore Spark

- Work with unstructured data
- Submit Dataproc Jobs
- Explore Spark
- Explore HDFS and Cloud Storage
- Use interactive PySpark to learn about RDDs
- Learn about RDD operations (transformation and actions)

Agenda

Cluster management and monitoring

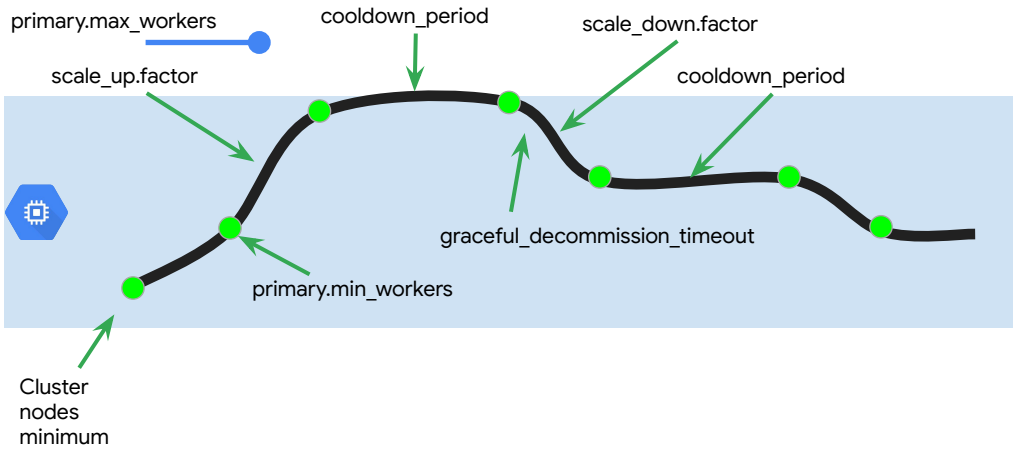
Cloud Dataproc Autoscaling provides flexible capacity



Cloud Dataproc Autoscaling provides flexible capacity for more efficient utilization.
It makes scaling decisions based on Hadoop YARN Metrics.
It is designed to be used only with off-cluster persistent data, not on-cluster HDFS or HBase.
It works best with a cluster that processes a lot of jobs or that processes a single large job.

It does not support Spark Structured Streaming (a streaming service built on top of Spark SQL).
It is not designed to scale to zero. So it is not the best for sparsely utilized or idle clusters.
In these cases it is equally fast to terminate a cluster that is idle and create a new cluster when it is needed.
For that purpose you would look at Cloud Dataproc Workflows or Cloud Composer, and Cluster Scheduled Deletion.

How Cloud Dataproc Autoscaling works



Initial workers -- The number of initial workers is set from Worker Nodes > nodes minimum. Setting this value ensures that the cluster comes up to basic capacity faster than if you let autoscaling handle it. Because autoscaling might require multiple autoscale periods to scale up.

The primary minimum number of workers may be the same as the cluster nodes minimum. There is a maximum that caps the number of worker nodes.

Now there is heavy load on the cluster. And Autoscaling determines it is time to scale up. The scale_up.factor determines how many nodes to launch. This would commonly be one node. But if you knew that a lot of demand would occur at once, maybe you want to scale up faster.

After the action, there is a cooldown period to let things settle before autoscaling evaluation occurs again. The cooldown period reduces the chances that the cluster will start and terminate nodes at the same time.

In this example, the extra capacity isn't needed. And there is a graceful decommission timeout to give running jobs a chance to complete before the node goes out of service. Notice there is a scale down factor. In this case it is scaling down by one node at a time for a more leisurely reduction in capacity.

After the action, there is another cooldown period.

And a second scale down, resulting in a return to the minimum number of workers.

A secondary min_workers and max_workers controls the scale of preemptible workers.

You can read about the scaling algorithm here:
https://cloud.google.com/dataproc/docs/concepts/configuring-clusters/autoscaling

scale_up.factor -- how many nodes to add during a scale-up event.

scale_down.factor -- how many nodes to remove during a scale-down.event.

graceful_decomission_timeout -- how long to wait for a jobs to complete before shutting down the node.

Cluster Scheduled Deletion

Idle

Timestamp

Duration

Minimum 10 minutes
Maximum 14 days
Granularity 1 second

Efficient utilization, don't pay for resources you don't use.

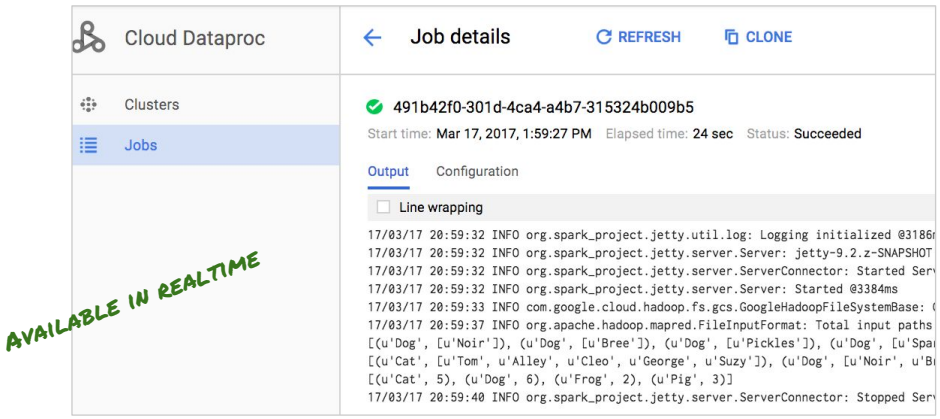
A fixed amount of time after the cluster enters the Idle state.
Set a timer. You give it a timestamp. The count starts immediately once the expiration has been set.
Set a duration. Time in seconds to wait before deleting the cluster.
Range is from 10 minutes minimum to 14 days maximum, with a granularity of 1 second.

Currently available from the command line and REST API, but not through Console.

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

RFC 3339 UTC "Zulu" format, accurate to nanoseconds
Provide a duration in seconds with up to nine fractional digits, terminated by 's'.
Example: "3.5s".

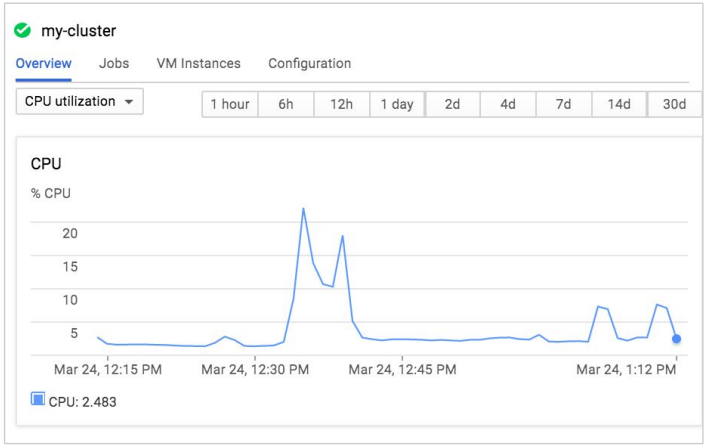
Monitor logs of submitted jobs from web console



Notes:

These logs are available in real-time.

Monitor cluster usage graphs on Web UI, Stackdriver



Notes:

Overall cluster usage from Dataproc page.

Individual VMs from Compute Engine VM.


cloud.google.com
Image by Connie Zhou