Executing Spark on Dataproc

Agenda

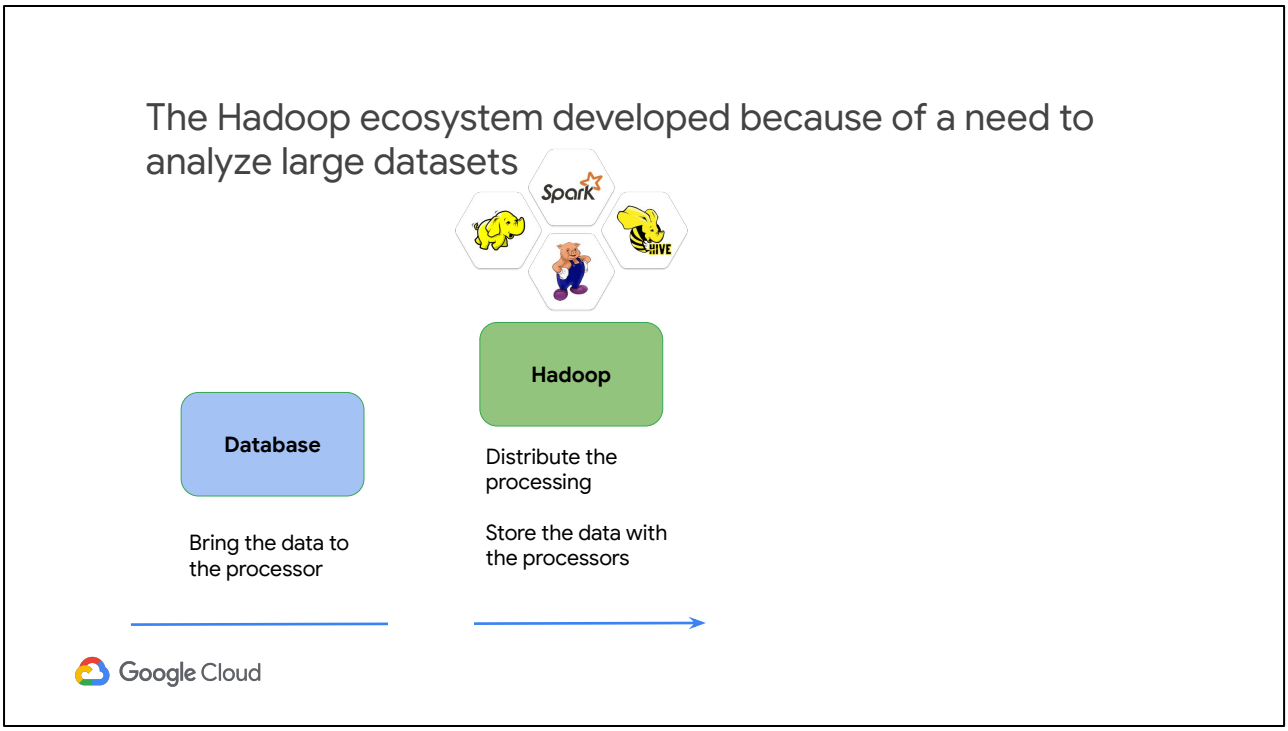
The Hadoop Ecosystem

Running Hadoop on Dataproc

Cloud Storage Instead of HDFS

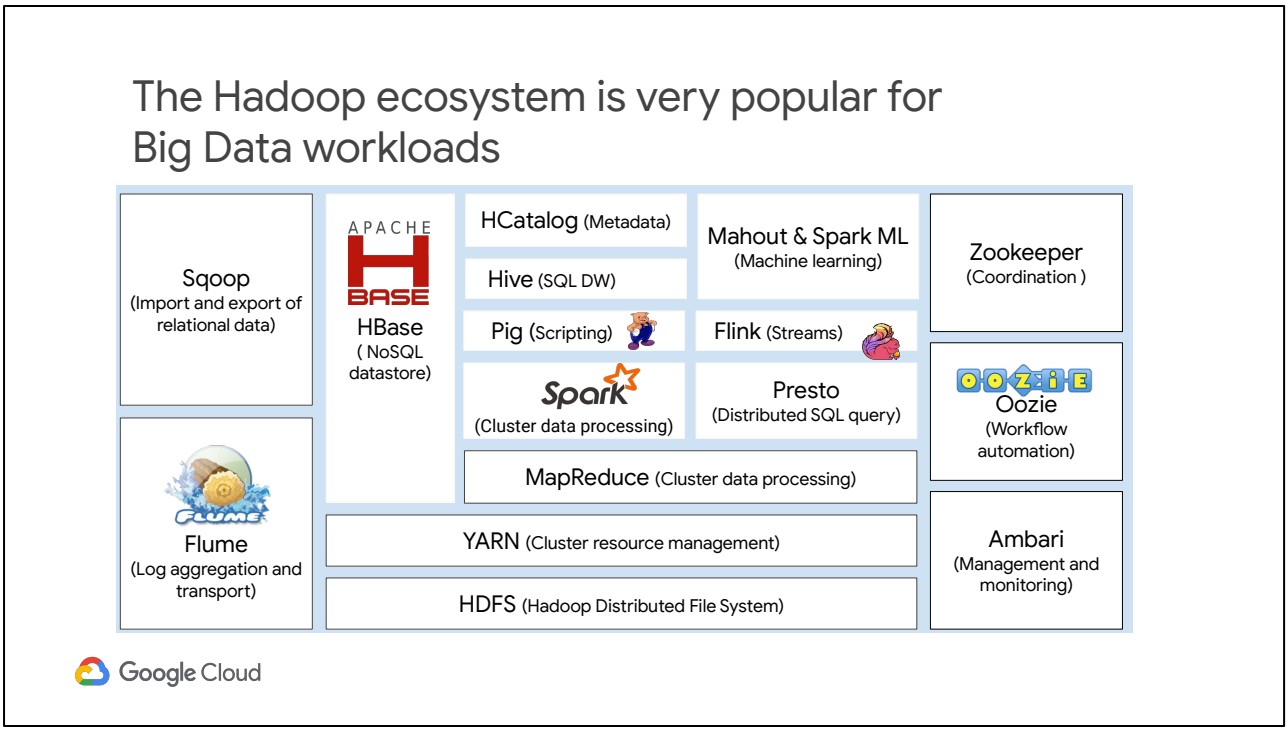
Optimizing Dataproc

Lab: Running Apache Spark jobs on Dataproc



It helps to place the services you will be learning about in historical context. 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.



Organisations use Hadoop for on-premises big data workloads. They make use of a range of applications that run on Hadoop clusters, such as Presto but a lot of customers use Spark..

Apache Hadoop is an open source software project that develops a framework for distributed processing of large data sets across clusters of computers using simple programming models. HDFS is the main file system Hadoop uses for distributing work to nodes on the cluster.

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. Spark, in particular, is very powerful and expressive and used for a lot of workloads.

A lot of the complexity and overhead of OSS Hadoop has to do with assumptions in the design that existed in the datacenter.
Relieved of those limitations, data processing becomes a much richer solution with many more options.

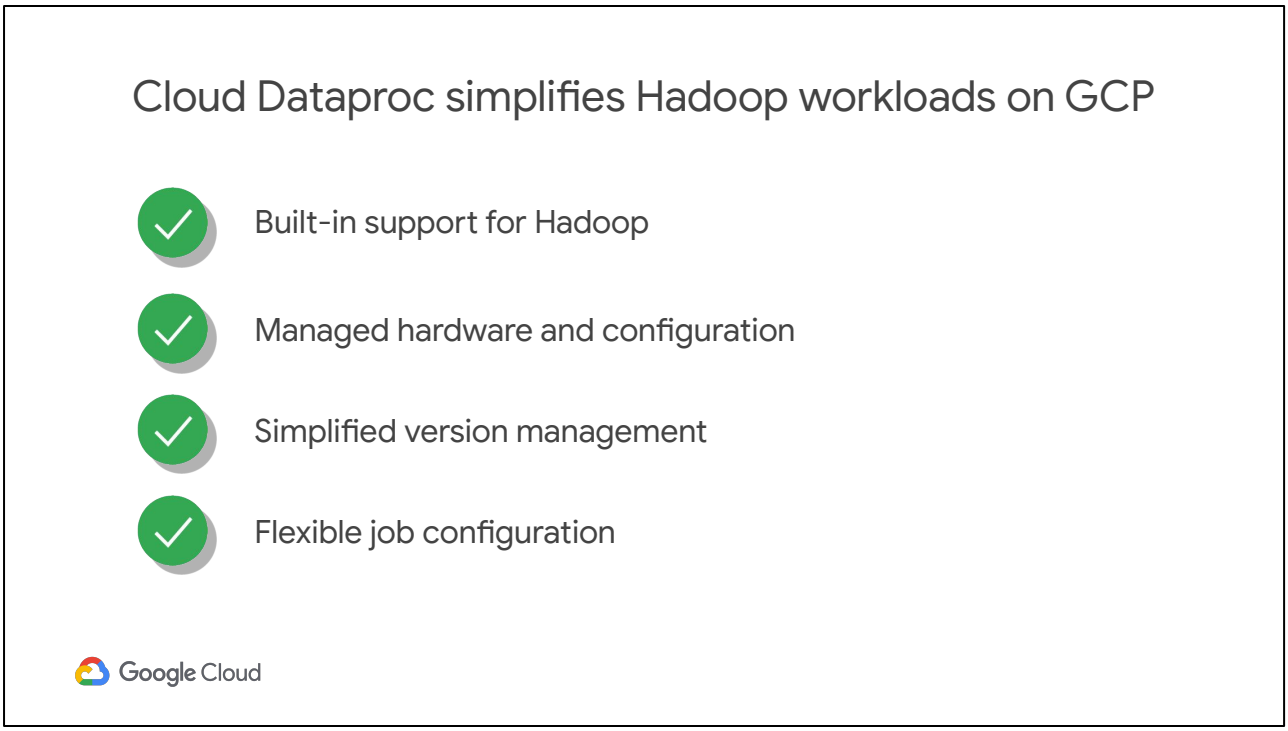
There are two common issues with OSS Hadoop; tuning and utilization. A company will typically have several Hadoop clusters that are shared by several organizations

and run a wide variety of jobs. Hadoop experts have to adjust many configuration settings in the collection of underlying open source project software to optimize the cluster for the varying kinds of work it is tasked to perform. That is the tuning problem. Hadoop clusters tend to have a lot of dedicated hardware, which makes them expensive when they are not being used. That is the utilization problem. Hadoop administrators may find they are searching through organizations to find data processing jobs so they can increase cluster utilization. If they are successful, the capacity will start to be consumed and it will be time to order more hardware. This cycle of tuning, under-utilization, over-utilization, expansion creates a significant overhead for Hadoop.

In many cases, using Cloud Dataproc as designed will overcome these limitations.

On-premises Hadoop clusters have a number of limitations

- X Not elastic
- X Hard to scale fast
- X Have capacity limits
- X Have no separation between storage and compute resources



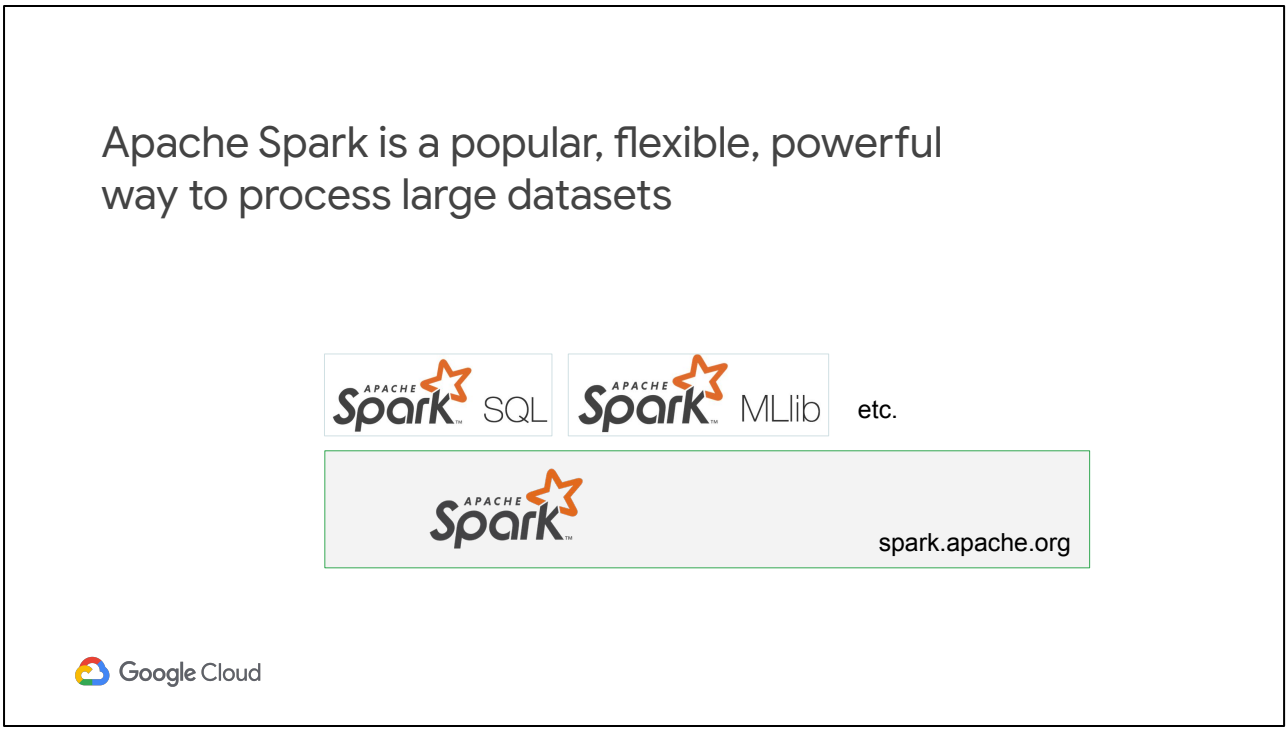
There are many ways in which using GCP can save you time, money, and effort compared to using an on-premises Hadoop solution. In many cases, adopting a cloud-based approach can make your overall solution simpler and easy to manage.

Built-in support for Hadoop
GCP includes Cloud Dataproc, which is a managed Hadoop and Spark environment. You can use Cloud Dataproc to run most of your existing jobs with minimal alteration, so you don't need to move away from all of the Hadoop tools you already know.

Managed hardware and configuration
When you run Hadoop on GCP, you never need to worry about physical hardware. You specify the configuration of your cluster, and Cloud Dataproc allocates resources for you. You can scale your cluster at any time.

Simplified version management
Keeping open source tools up to date and working together is one of the most complex parts of managing a Hadoop cluster. When you use Cloud Dataproc, much of that work is managed for you by Cloud Dataproc versioning.

Flexible job configuration
A typical on-premises Hadoop setup uses a single cluster that serves many purposes. When you move to GCP, you can focus on individual tasks, creating as many clusters as you need. This removes much of the complexity of maintaining a single cluster with growing dependencies and software configuration interactions.



Running MapReduce directly on top of Hadoop is very useful. But it has the complication that the Hadoop system has to be "tuned" for the kind of job being run to make efficient use of the underlying resources. Imagine a job working on millions of pieces of sensor data coming from an Internet of Things application. And Imagine a job working on those huge photos from the previous example. Trying to do both things at the same time efficiently is complicated. One important innovation is Spark. And a simple explanation of Spark is that it is able to mix different kinds of applications and to adjust how it uses the available resources.

You have to learn to program Spark differently from traditional programming. Because you can't tell it how to do things. To give Spark the flexibility it needs to determine how to use the resources that are available, you have to describe what you want and let Spark determine how to make that happen. This is called declarative programming versus imperative programming. In imperative programming you tell the system what to do and how to do it. In declarative programming you tell the system what you want and it figures out how to implement it. You will be learning to work with Spark in the labs in this course.

There is a full SQL implementation on top of Spark. There is a common DataFrame model that works across Scala, Java, Python, SQL and R. And there is a distributed machine learning library called Spark ML-lib.

Student Notes:

You may know of Spark as an open source general large scale data processing tool like Apache Hadoop MapReduce, which it is. But a lot of layers have been built on top of that.

There is a full SQL implementation written on top of it, which provides a common DataFrame data model to Scala, Java, SQL, R, and Python.

And on top that is the Spark MLlib Spark's Distributed Machine Learning library.

Agenda

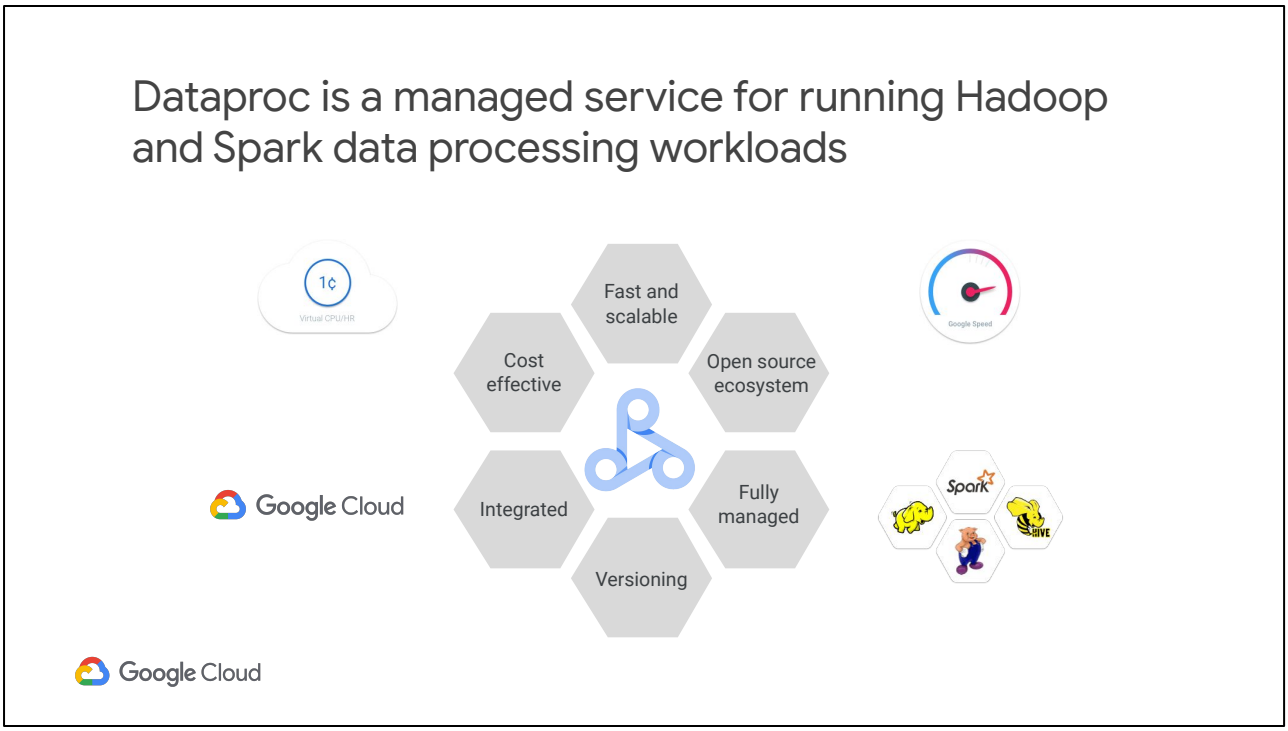
The Hadoop Ecosystem

Running Hadoop on Dataproc

Cloud Storage Instead of HDFS

Optimizing Dataproc

Lab: Running Apache Spark jobs on Dataproc



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

When compared to traditional, on-premises products, and competing cloud services, Dataproc has unique advantages for clusters of three to hundreds of nodes.

There is no need to learn new tools or APIs to use Dataproc, making it easy to move existing projects into Dataproc without redevelopment. Spark, Hadoop, Pig, and Hive are frequently updated.

Here are some of the key features of Dataproc:

- **Low cost:** Dataproc is priced at 1 cent per virtual CPU per cluster per hour, on top of the other Google Cloud resources you use. In addition, Dataproc clusters can include preemptible instances that have lower compute prices. You use and pay for things only when you need them, so Dataproc charges second-by-second billing with a one-minute-minimum billing period.
- **Super-fast:** Dataproc clusters are quick to start, scale, and shutdown, with each of these operations taking 90 seconds or less, on average.
- **Resizable clusters:** Clusters can be created and scaled quickly with a variety of virtual machine types, disk sizes, number of nodes, and networking options.
- **Open source ecosystem:** You can use Spark and Hadoop tools, libraries,

- and documentation with Dataproc. Dataproc provides frequent updates to native versions of Spark, Hadoop, Pig, and Hive, so there is no need to learn new tools or APIs, and it is possible to move existing projects or ETL pipelines without redevelopment.
- **Integrated:** Built-in integration with Cloud Storage, BigQuery, Cloud Bigtable ensures data will not be lost. This, together with Cloud Logging and Cloud Monitoring, provides a complete data platform and not just a Spark or Hadoop cluster. For example, you can use Dataproc to effortlessly ETL terabytes of raw log data directly into BigQuery for business reporting.
- **Managed:** Easily interact with clusters and Spark or Hadoop jobs, without the assistance of an administrator or special software, through the Cloud Console, the Cloud SDK, or the Dataproc REST API. When you're done with a cluster, simply turn it off, so money isn’t spent on an idle cluster.
- **Versioning:** Image versioning allows you to switch between different versions of Apache Spark, Apache Hadoop, and other tools.

In addition:

- **Highly available:** Run clusters with multiple primary nodes and set jobs to restart on failure to ensure your clusters and jobs are highly available.
- **Developer tools:** Multiple ways to manage a cluster, including the Cloud Console, the Cloud SDK, RESTful APIs, and SSH access.
- **Initialization actions:** Run initialization actions to install or customize the settings and libraries you need when your cluster is created.
- **Automatic or manual configuration:** Dataproc automatically configures hardware and software on clusters for you while also allowing for manual control.

There are other OSS options available in Cloud Dataproc

Spark (default)	Hive (default)	HDFS (default)
Pig (default)	Zeppelin	Zookeeper
Kafka	Hue	Tez
Presto	Anaconda	Cloud SQL Proxy
Jupyter	Apache Flink	Cloud Datalab
IPython	Oozie	Sqoop
Much more...		

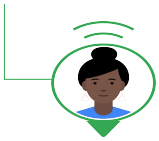
Google Cloud

Cloud Dataproc has two ways to customize clusters; optional components and initialization actions. Pre-configured optional components can be selected when deploying from the console or via the command line and include:Anaconda, Hive WebHCat, Jupyter Notebook, Zeppelin Notebook, Druid, Presto and Zookeeper.

Initialization actions let you customize your cluster by specifying executables or scripts that Cloud Dataproc will run on all nodes in your Cloud Dataproc cluster immediately after the cluster is set up. You can define your own initialization scripts or select from a wide range of frequently used and other sample initialization actions as detailed here : https://cloud.google.com/dataproc/docs/concepts/init-actions#examplewzxhzdk15staging_binaries

Use initialization actions to add other software to cluster at startup

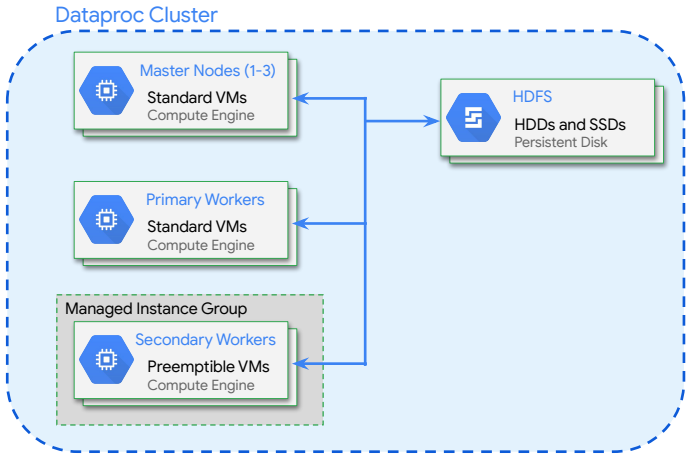
Use initialization actions to install additional components on the cluster.



```
gcloud dataproc clusters create <CLUSTER_NAME> \
    --initialization-actions gs://$MY_BUCKET/hbase/hbase.sh \
       --num-masters 3 --num-workers 2
```

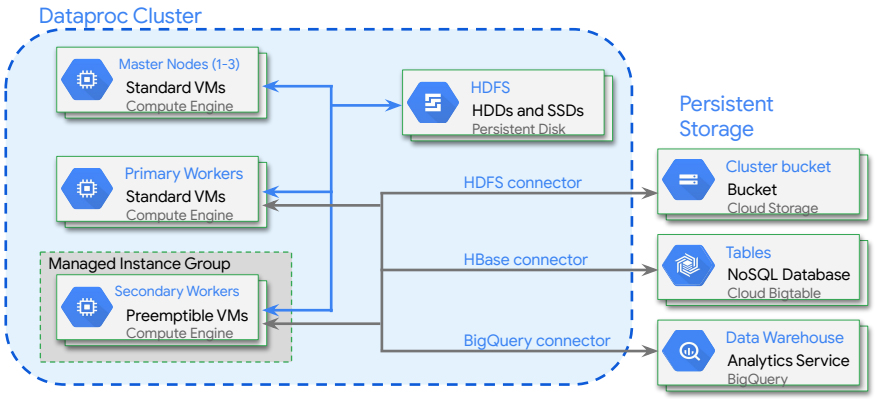
https://github.com/GoogleCloudPlatform/dataproc-initialization-actions (Flink, Jupyter, Oozie, Presto, Tez, HBase, etc.)

A Dataproc cluster has master nodes, workers, and HDFS

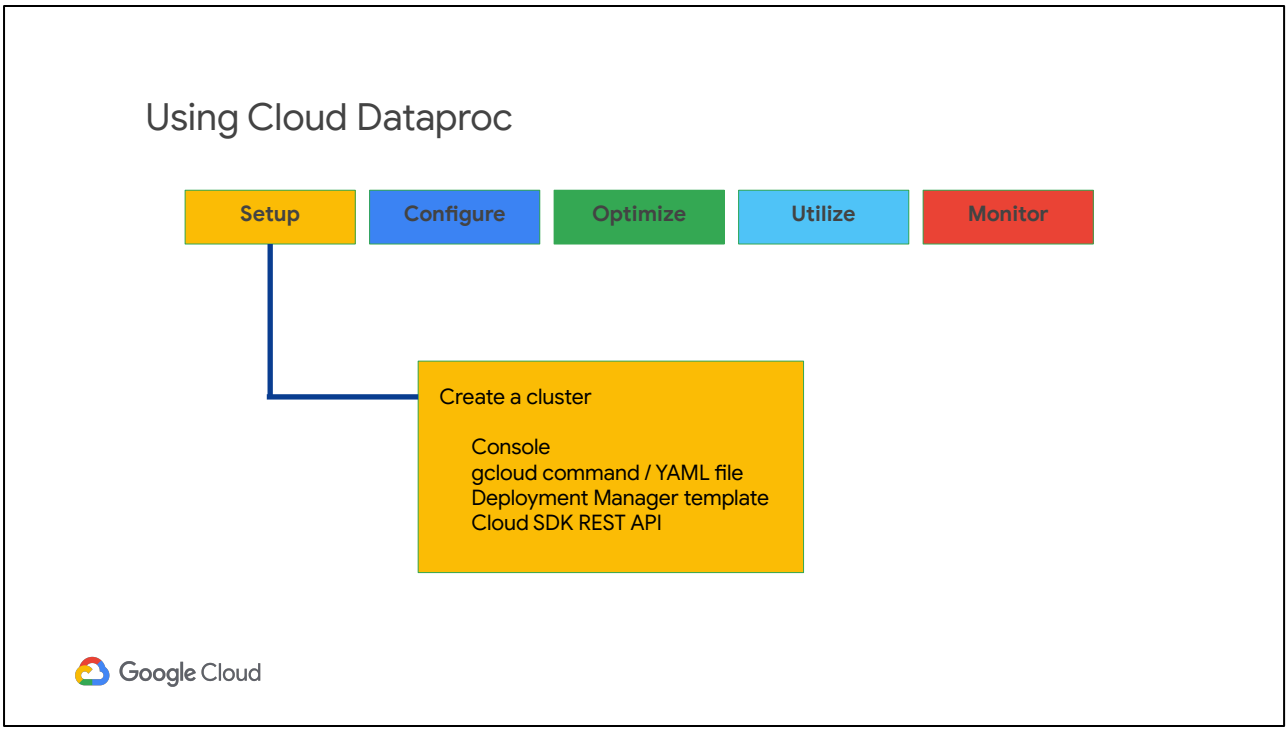


Google Cloud

Dataproc cluster can read/write to GCP storage products



Google Cloud

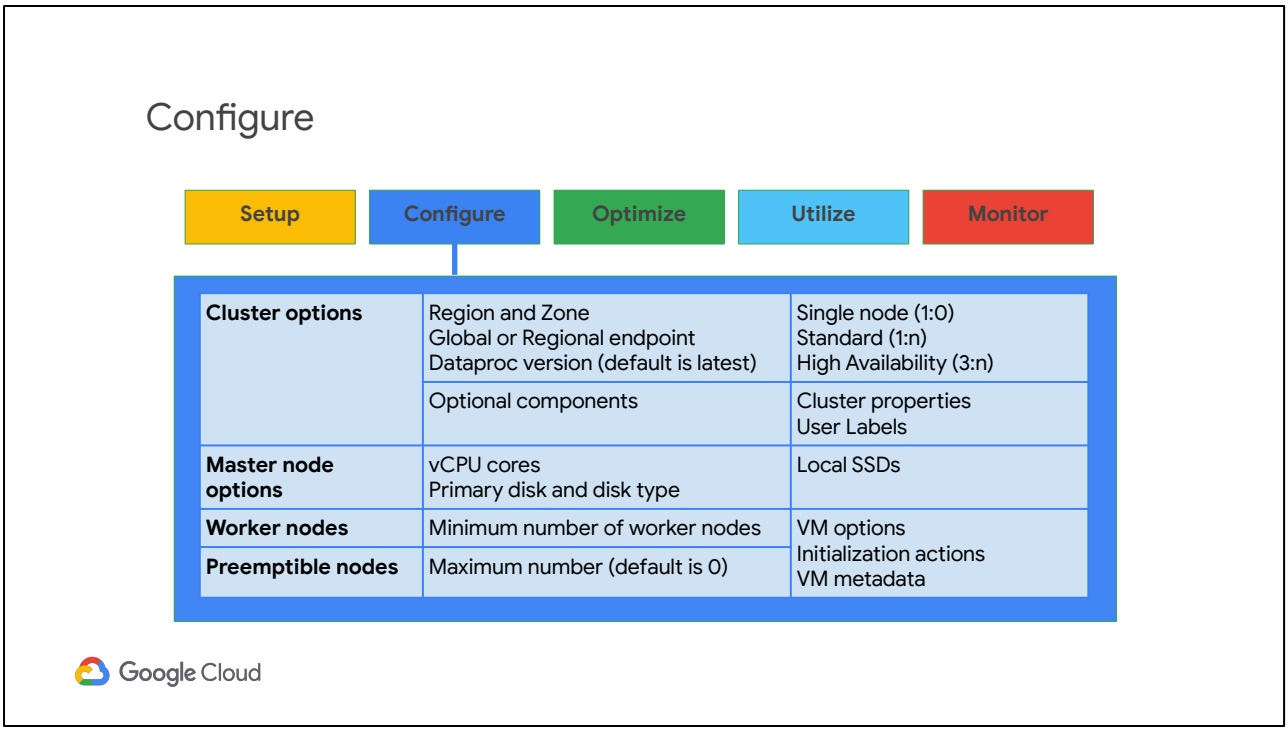


Using Cloud Dataproc involves this sequence of events: Setup, Configuration, Optimization, Utilization, and Monitoring.

Setup means creating a cluster. And you can do that through console, from the command line using the gcloud command. You can also export a YAML file from an existing cluster or create a cluster from a YAML file. You can create a cluster from a Deployment Manager template, or use the REST API.

Create a cluster documentation is here:
https://cloud.google.com/dataproc/docs/guides/create-cluster

Cluster properties can be used to modify common OSS configuration files.
https://cloud.google.com/dataproc/docs/concepts/configuring-clusters/cluster-properties



The cluster can be set as a single VM, which is usually to keep costs down for development and experimentation. Standard is with a single Master Node, and High Availability has three Master Nodes. You can choose a region and zone, or select a "global region" and allow the service to choose the zone for you. The cluster defaults to a Global endpoint, but defining a Regional endpoint may offer increased isolation and in certain cases, lower latency.

The Master Node is where the HDFS Namenode runs, as well as the YARN node and job drivers. HDFS replication defaults to 2 in Cloud Dataproc.

Optional components from the Hadoop-ecosystem include: Anaconda (Python distribution and package manager), Hive Webcat, Jupyter Notebook, Zeppelin Notebook

Cluster properties are run-time values that can be used by configuration files for more dynamic startup options.
And user labels can be used to tag the cluster for your own solutions or reporting purposes.

The Master Node Worker Nodes, and preemptible Worker Nodes, if enabled, have separate VM options, such as vCPU, memory, and storage.

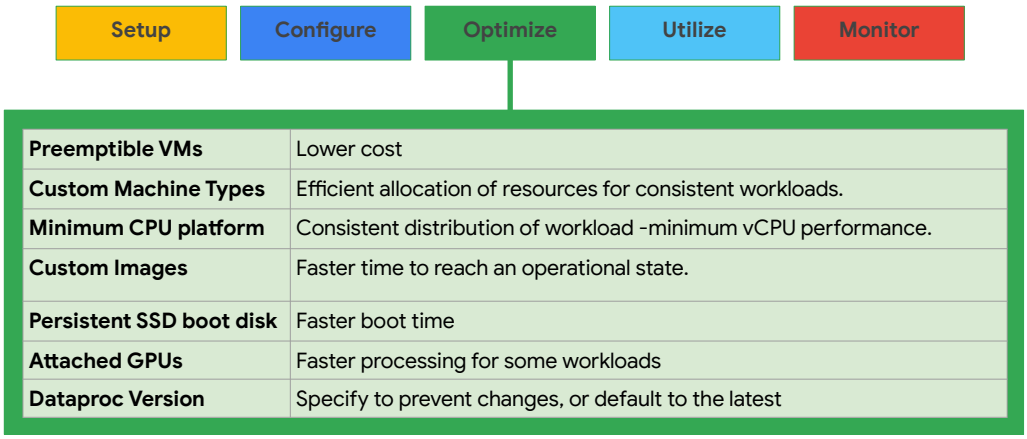
Preemptible nodes include YARN NodeManager but they do not run HDFS.

There are a minimum number of worker nodes, the default is 2. The maximum number of worker nodes is determined by a quota and the number of SSDs attached to each worker.

You can also specify initialization actions, such as initialization scripts that can further customize the worker nodes. And metadata can be defined so that the VMs can share state information.

Cluster properties modify common OSS configuration file values.
https://cloud.google.com/dataproc/docs/concepts/configuring-clusters/cluster-properties

Optimize



Google Cloud

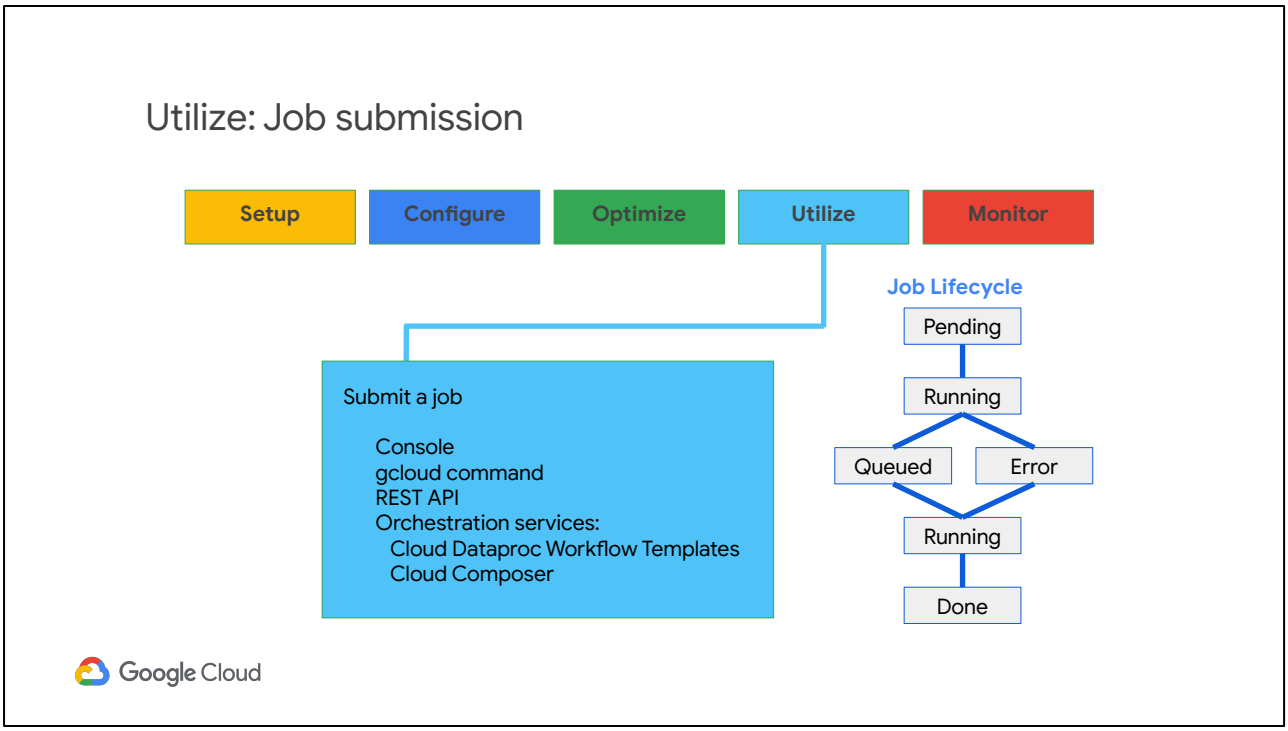
Preemptible VMs can be used to lower costs. Just remember they can be pulled from service at any time and within 24 hours. So your application might need to be designed for resilience to prevent data loss.

Custom Machine Types allow you to specify the balance of Memory and CPU to tune the VM to the load, so you are not wasting resources.

A custom image can be used to pre-install software so that it takes less time for the customized node to become operational than if you installed the software are boot-time using and initialization script.

Custom image creation and use:
https://cloud.google.com/dataproc/docs/guides/dataproc-images

You can get a Persistent SSD boot disk for faster cluster start-up.



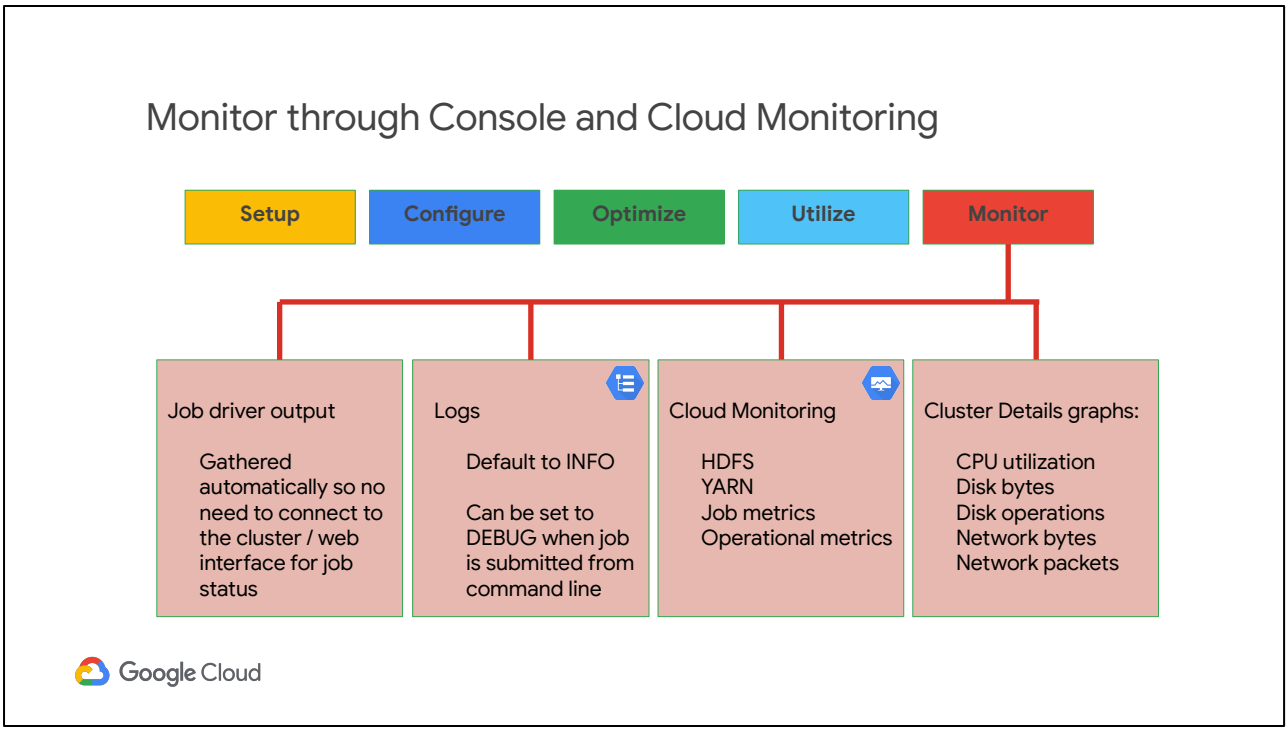
Jobs can be submitted through Console, the gcloud command, or the REST API. They can also be started by orchestration services such as Cloud Dataproc Workflow and Cloud Composer.

Don't use Hadoop's direct interfaces to submit jobs because the metadata will not be available to Cloud Dataproc for job and cluster management, and for security, they are disabled by default.

Job lifecycle:
https://cloud.google.com/dataproc/docs/concepts/jobs/life-of-a-job

By default, jobs are not restartable. However, you can create restartable jobs through the command line or REST API. Restartable jobs must be designed to be idempotent and to detect successorship and restore state.

Restartable jobs:
https://cloud.google.com/dataproc/docs/concepts/jobs/restartable-jobs.



Using Cloud Monitoring, you can build a custom dashboard with graphs and set monitoring alert policy to notify when incidents occur.

https://cloud.google.com/dataproc/docs/guides/monitoring

Cluster page

Status: such as "Running"
Region
Creation date and time
Total number of Worker nodes

Cluster Details page in Console provides graphs of load on the cluster.
Graphs:
CPU utilization
Disk bytes
Disk operations
Network bytes
Network packets

Agenda

The Hadoop Ecosystem

Running Hadoop on Dataproc

Cloud Storage Instead of HDFS

Optimizing Dataproc

Lab: Running Apache Spark jobs on Dataproc

The original MapReduce paper was designed for a world where data was local to the compute machine

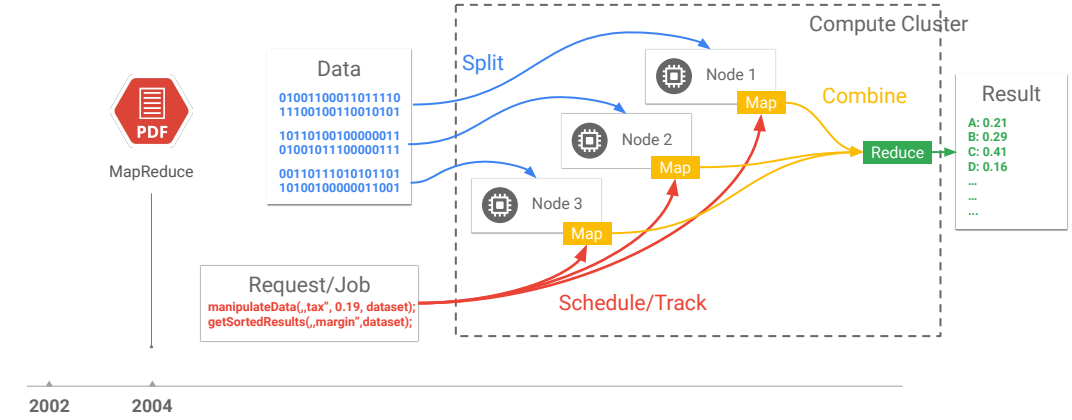
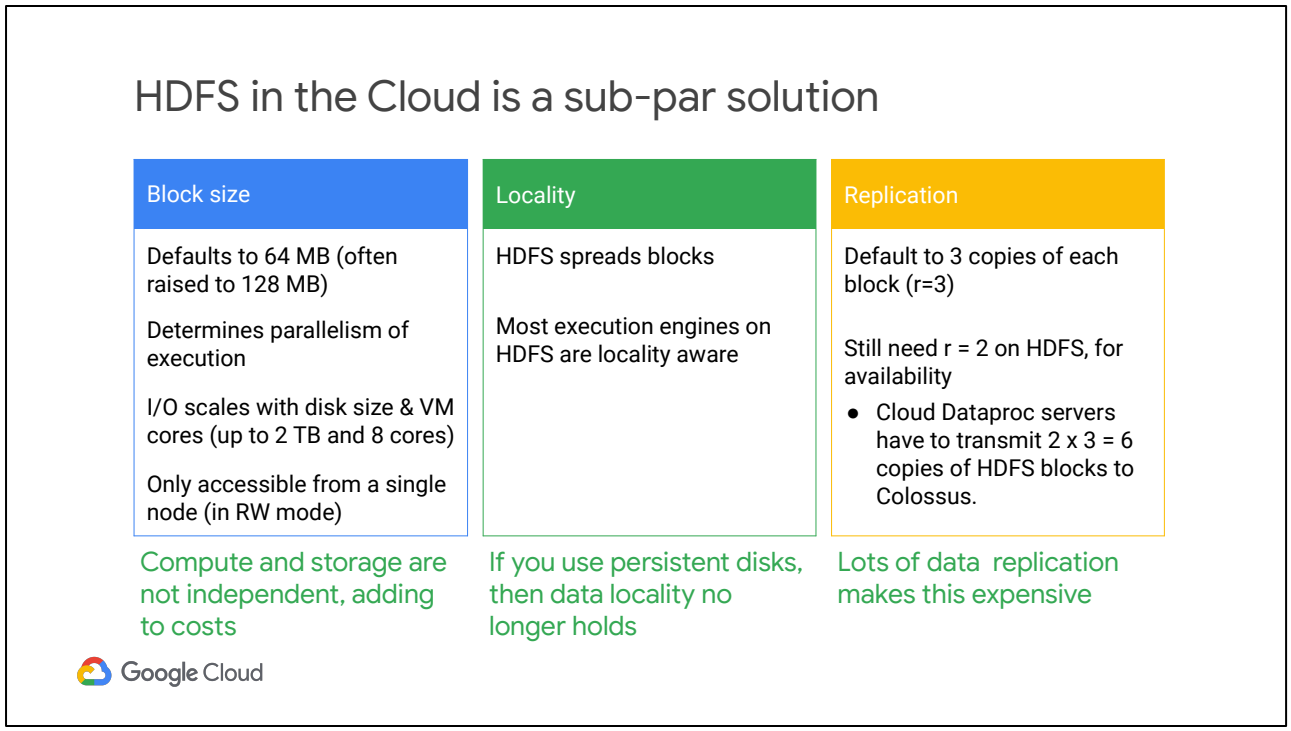


Diagram by Lukas Kästner

Google Cloud

Network speeds were slow originally that's why we kept data as close as possible to the processor. Now, with petabit networking you can treat storage and compute independently and move traffic quickly over the network.

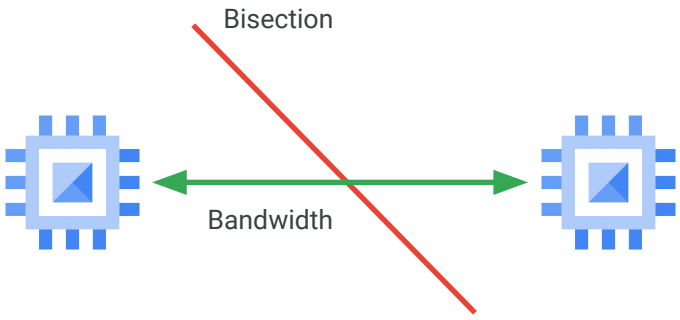


Your on-premise Hadoop clusters need local storage on its disk, since the same server runs, computes, and stores jobs in the Cloud. That's one of the first areas for optimization. You can run HDFS in the Cloud just by lifting and shifting your Hadoop workloads to Cloud Dataproc. This is often the first step to the Cloud, and requires no code changes. It just works, but HDFS on the Cloud is a sub-par solution in the long run.

This is because of how HDFS works on the clusters, with block size, the data locality, and the replication of the data in HDFS. For block size in HDFS, you're tying the performance of input and output to the actual hardware the server is running on. Again, storage is not elastic in this scenario; you're on the cluster. If you run out of persistent disk space on your cluster, you'll need a re-size, even if you don't need the extra compute power.

For data locality, there are similar concerns about storing data on individual persistent disks. This is especially true when it comes to replication. In order for HDFS to be highly available, it replicates three copies of each block out to storage. It would be better to have a storage solution that's separately managed from the constraints of your cluster.

Petabit bandwidth is a game-changer for big data

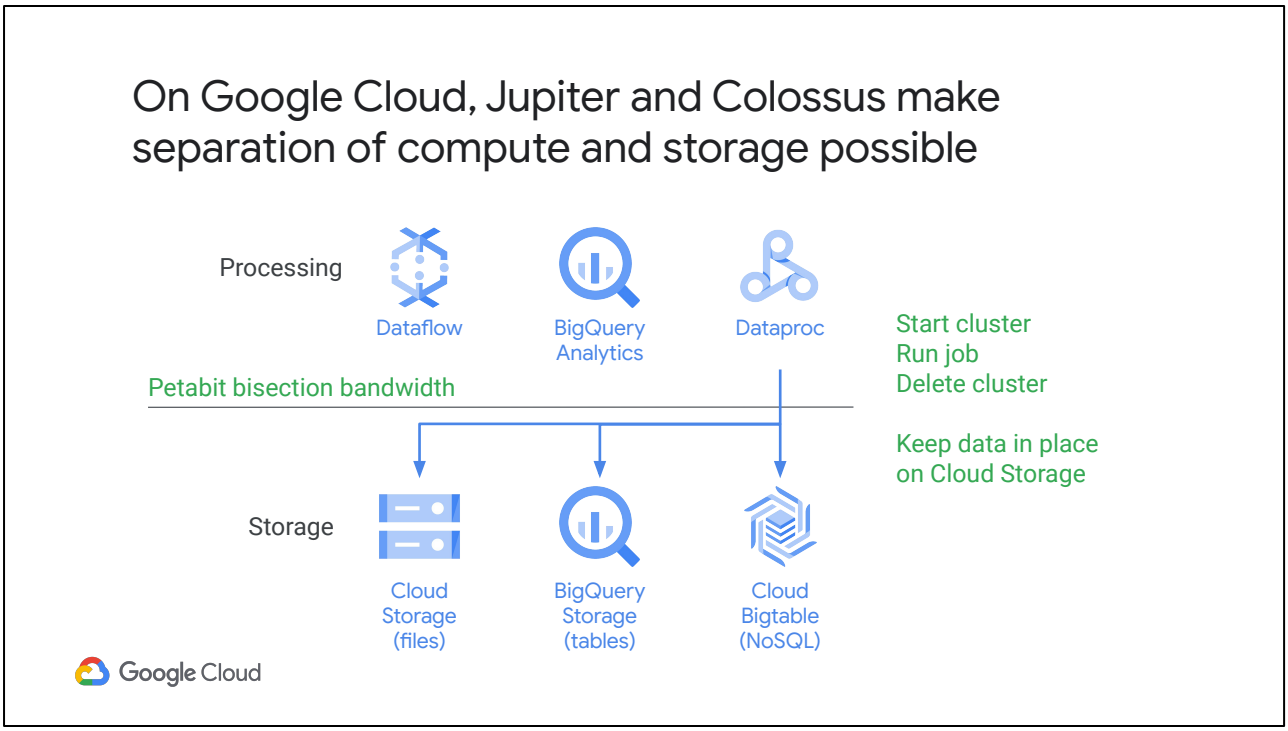


Process the data where it is without copying it

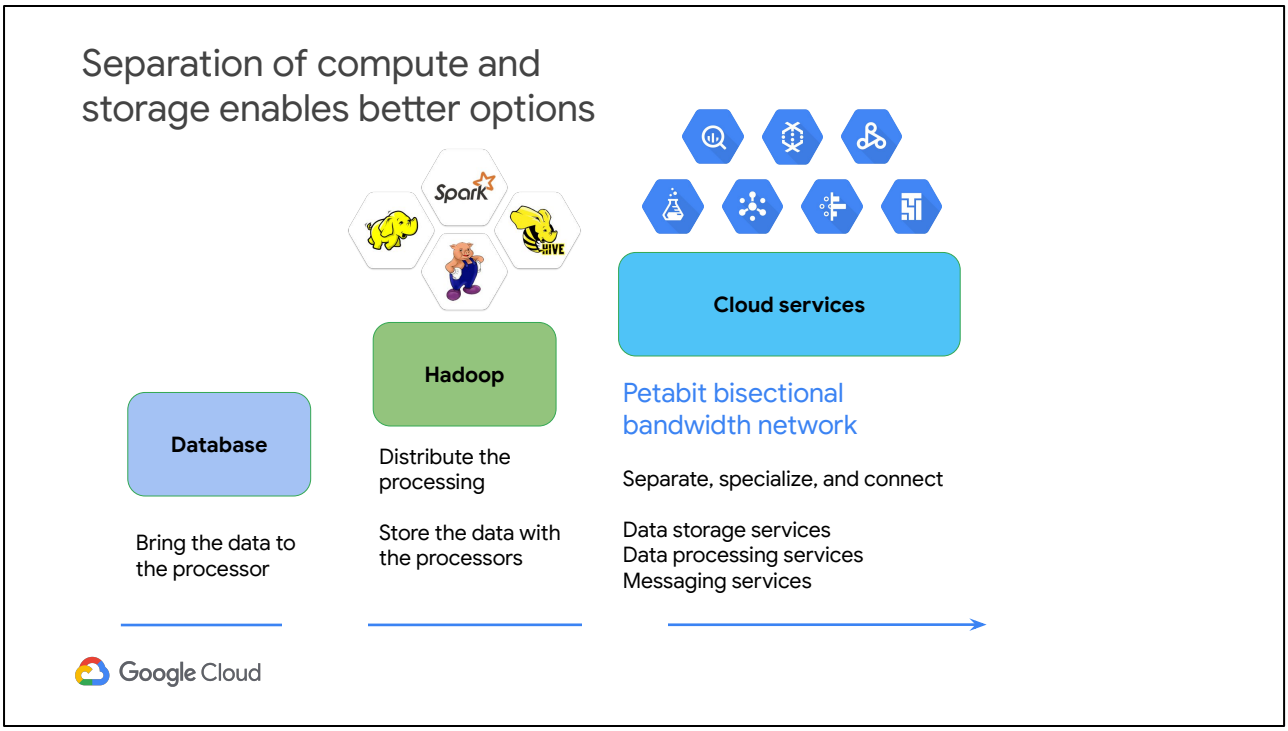
Google Cloud

Google's network enables new solutions for Big Data. The Jupiter networking fabric within a Google data center delivers over 1 PB/s of bandwidth. https://cloudplatform.googleblog.com/2015/06/A-Look-Inside-Googles-Data-Center-Networks.html . To put that into perspective, that's about twice the amount of traffic exchanged on the entire public Internet. (see Cisco's annual estimate of all Internet traffic)

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.



Inside of a Google datacenter, the internal name for the massively distributed storage layer is called Colossus, and the network inside the datacenter is Jupiter. Dataproc clusters get the advantage of scaling up and down VMs that they need to do the compute, while passing off persistent storage needs with the ultra-fast Jupiter network to a storage product like Cloud Storage, which is ran by Colossus behind the scenes.

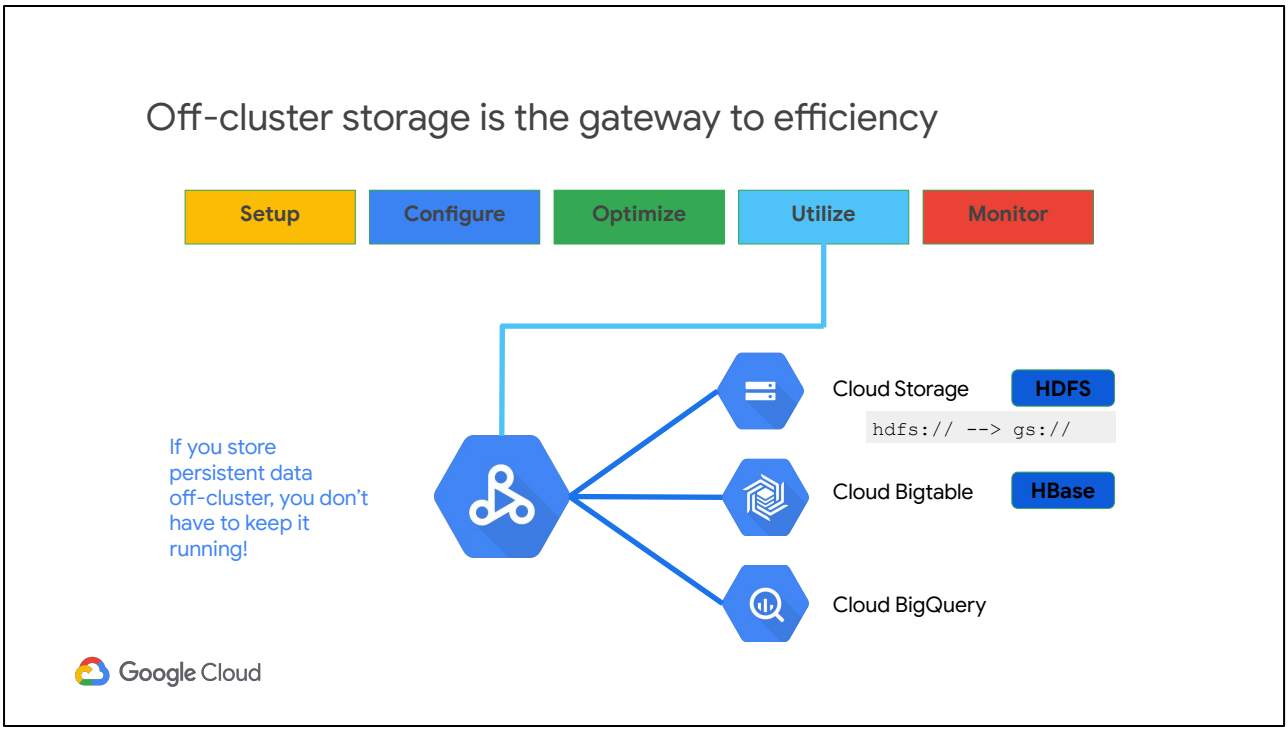


It helps to place the services you will be learning about in historical context. Before 2006 big data meant big databases. Database design came from a time when storage was relatively cheap and processing was expensive, so it made sense to copy the data from its storage location to the processor to perform 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.

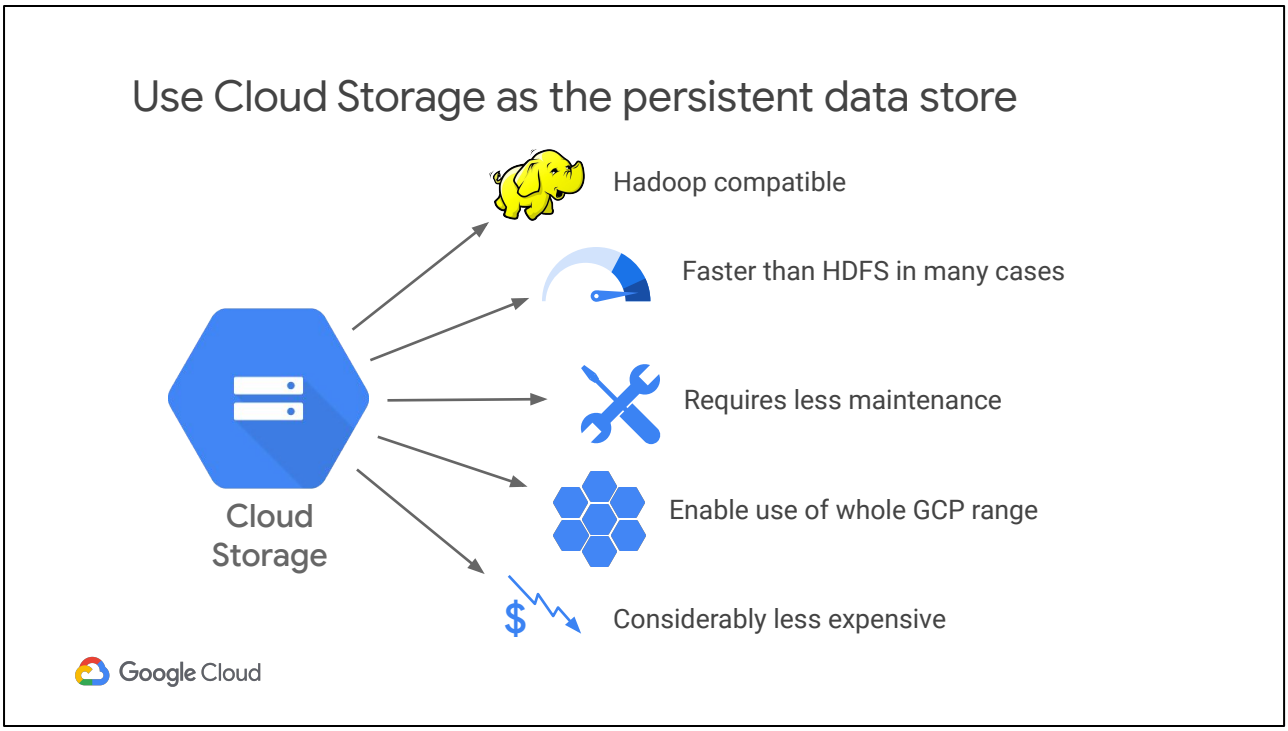
2003, 2004 -- Google File System
2006, 2008 -- Hadoop
2015 -- Cloud Dataproc
BigQuery -- 2010
Cloud Dataflow -- 2014



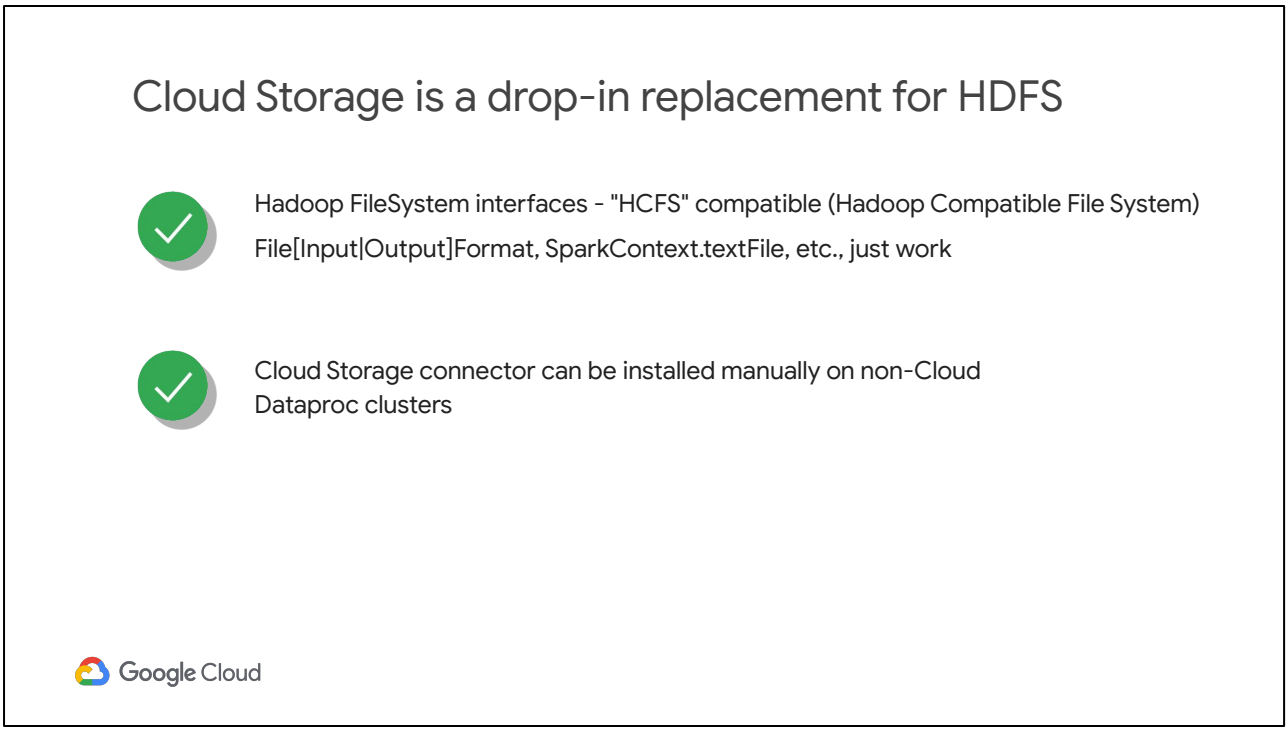
For maximum efficiency, use HDFS on the cluster for working storage just during processing. This allows the cluster to be created for a single job or type of workload and to be shut down when not in use.

There is an HDFS connector for Cloud Storage, and an HBase connector for Cloud Bigtable.

To take advantage of Cloud Storage instead of HDFS, you can simply change your Hadoop job code from hdfs// to gs// as you see here. Additionally, consider using BigQuery for your Data Processing and analytical workloads, instead of performing them on the cluster.



One of the biggest benefits of Hadoop in the cloud is that separation of compute and storage. With Cloud Storage as the backend, you can treat clusters themselves as ephemeral resources, which allows you not to pay for compute capacity when you're not running any jobs. Also, Cloud Storage is its own completely scalable and durable storage service, which is connected to many other GCP projects.



Cloud storage could be a drop-in replacement for your HDFS backend for Hadoop. The rest of your code would just work. Also, you can use the Cloud Storage connector manually on your non-cloud Hadoop clusters if you didn't want to migrate your entire cluster to the Cloud yet.

HDFS - you must overprovision for current data and for data you might have, and you must use persistent disks throughout.
Cloud Storage - pay for exactly what you need, when you use it.

Performance best practices

Optimize for bulk/parallel operations

- Avoid small reads; use large block sizes where possible
- Avoid iterating sequentially over many nested directories in a single job

Google Cloud

Cloud Storage is optimized for large, bulk parallel operations. It has very high throughput, but it has significant latency. If you have large jobs that are running lots of tiny little blocks, you may be better off with HDFS. Additionally, you want to avoid iterating sequentially over many nested directories in a single job.

Use Cloud Storage instead of HDFS with Cloud Dataproc

Setup | Configure | Optimize | Utilize | Monitor

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 do not support "append"

Google Cloud

Using Cloud Storage instead of HDFS provide some key benefits due to the distributed service including eliminating bottlenecks and single points of failure.

However, there are some disadvantages to be aware of, including the challenges presented by renaming objects and the inability to append to objects.

Directory rename in HDFS not the same as in Cloud Storage

Cloud Storage has no concept of directories!

mv gs://foo/bar/ gs://foo/bar2

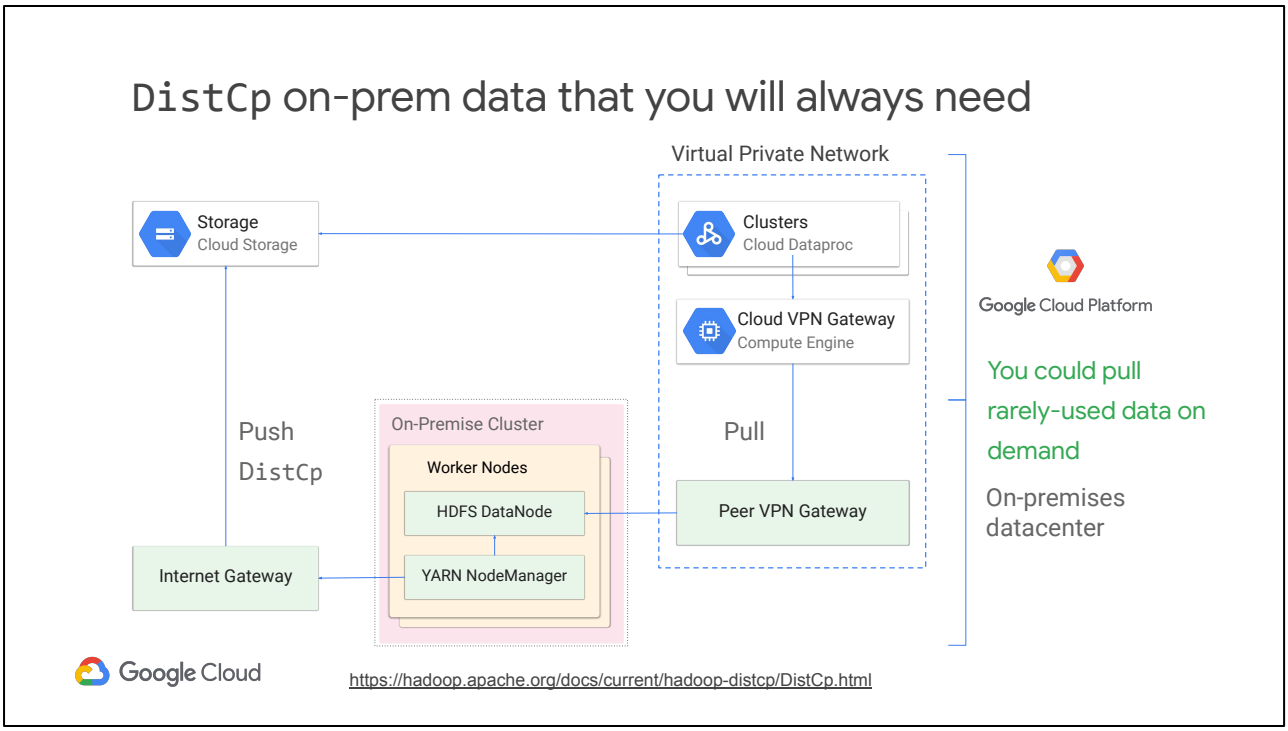
- list(gs://foo/bar/)
- copy({gs://foo/bar/baz1, gs://foo/bar/baz2}, {gs://foo/bar2/baz1, gs://foo/bar2/baz2})
- delete({gs://foo/bar/baz1, gs://foo/bar/baz2})

Migrated code should handle list inconsistency during rename!

- Modern output format committers handle object stores correctly

Google Cloud

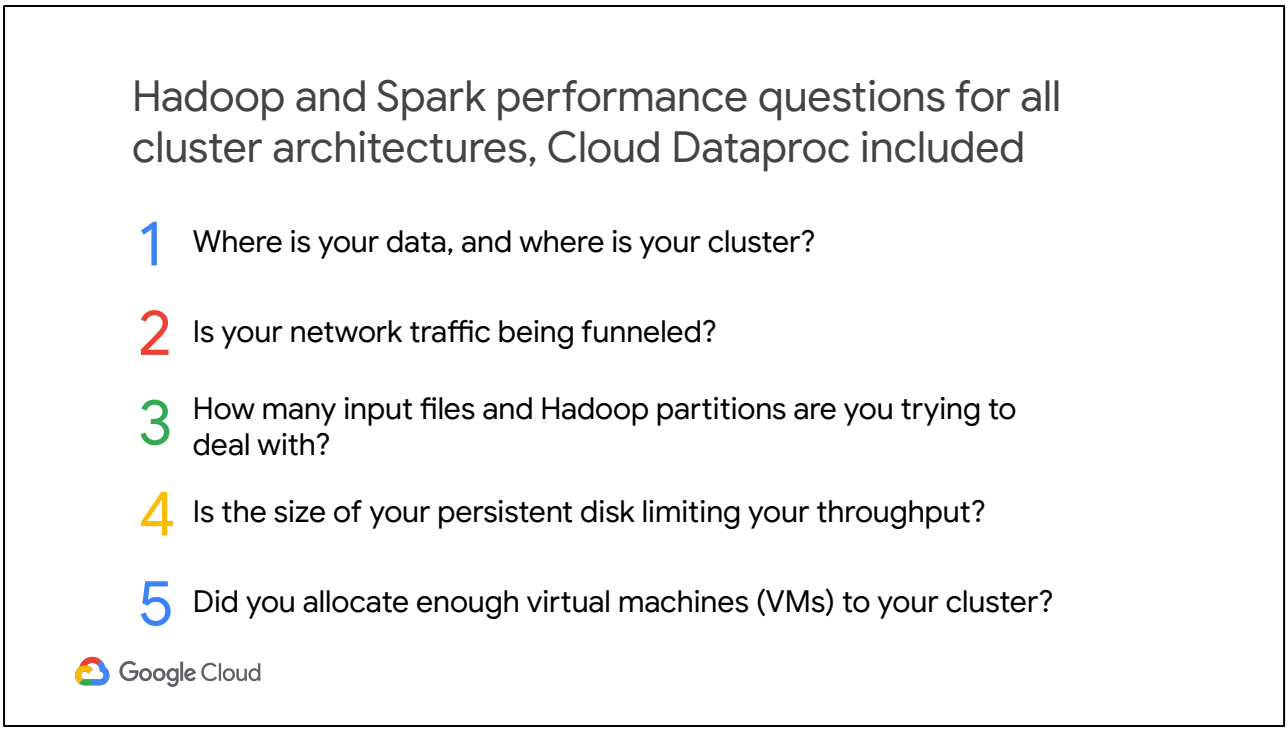
Cloud Storage is at its core an object store. It only simulates a file directory. So directory renames in HDFS are not the same as they are in Cloud Storage, but new objects store oriented output committers mitigate this, as you see here.



To get your data to the cloud, you can use DistCp. In general, you want to use a push-based model for any data that you know you will need, while pull-based may be a useful model if there is a lot of data that you might not ever need to migrate.

Agenda

The Hadoop ecosystem
Running Hadoop on Cloud Dataproc
GCS instead of HDFS
Optimizing Dataproc
Lab



Where is your data, and where is your cluster?
Knowing your data locality can have a major impact on your performance. You want to be sure that your data's region and your cluster's zone are physically close in distance.
When using Cloud Dataproc, you can omit the zone and have the Cloud Dataproc Auto Zone feature select a zone for you in the region you choose. While this handy feature can optimize on where to put your cluster, it does not know how to anticipate the location of the data your cluster will be accessing. Make sure that the Cloud Storage bucket is in the same regional location as your Dataproc region.

Is your network traffic being funneled?
Be sure that you do not have any network rules or routes that funnel Cloud Storage traffic through a small number of VPN gateways before it reaches your cluster. There are large network pipes between Cloud Storage and Compute Engine. You don't want to throttle your bandwidth by sending traffic into a bottleneck in your GCP networking configuration

How many input files and Hadoop partitions are you trying to deal with?
Make sure you are not dealing with more than around 10,000 input files. If you find yourself in this situation, try to combine or "union" the data into larger file sizes. If this file volume reduction means that now you are working with larger datasets (more than ~50k Hadoop partitions), you should consider adjusting the setting fs.gs.block.size to a larger value accordingly.
`fs.gs.block.size` is a configuration parameter that helps jobs perform splits. When

used in conjunction with Cloud Storage data, it becomes a "fake" value since Cloud Storage does not expose actual block sizes. The default value of 64MB is solely for helping Hadoop decide how to perform splits. Therefore, if the files are larger than 512MB, you may find that you can achieve better performance by manually increasing this value up to 1GB or even 2GB. Unlike standard persistent disks, the IOPS performance of SSD persistent disks depends on the number of vCPUs in the instance.

Is the size of your persistent disk limiting your throughput?
Often times, when getting started with Google Cloud, you may have just a small table that you want to benchmark. This is generally a good approach, as long as you do not choose a persistent disk that is sized to such a small quantity of data; it will most likely limit your performance. Standard persistent disks scale linearly with volume size.

Did you allocate enough virtual machines (VMs) to your cluster?
A question that often comes up when migrating from on-premises hardware to Google Cloud is how to accurately size the number of virtual machines needed. Understanding your workloads is key to identifying a cluster size. Running prototypes and benchmarking with real data and real jobs is crucial to informing the actual VM allocation decision. Luckily, the ephemeral nature of the cloud makes it easy to “right-size” clusters for the specific task at hand instead of trying to purchase hardware up front. Thus, you can easily resize your cluster as needed. Employing job-scoped clusters is a common strategy for Dataproc clusters.

Even though we know that clusters can easily scale up or down, it can still be useful to have some back-of-the-napkin calculations as we approach our cluster sizes. For an example calculation, let’s say that we are migrating from 50 physical nodes, each with 12 physical cores and 2 hyperthreads per core.

It’s important to understand that on Compute Engine, each virtual CPU (vCPU) is implemented as a single hardware hyper-thread on one of the available CPU platforms. In our example of 50 nodes each with 12 physical cores, you would have two options you might configure on Compute Engine:

- Option 1: 1,200 4-vCPU VMs (300 n1-standard-4)
- Option 2: 600 8-vCPU VMs (150 n1-standard-8)

When considering which option to choose, it's also important to factor in the storage implications associated with the choice. There are limits to the total amount of persistent disk that you can add to each VM. Most instance types have a 64TB limit, which means that Option 2 would limit the data on your cluster to 225TB. In our example, we should consider if this is enough or if we would prefer to have more VMs and thus, increase our storage size. Since customers typically move the vast majority of their long-term data storage from HDFS into Cloud Storage when they migrate to the cloud, usually the persistent drive limits are more than sufficient. However, you might still want to consider your specific workloads. Some storage requirements worth

investigating include:

- HBase data, including replication
- Un-replicated temporary shuffle spills
- Intra-pipeline datasets produced in the middle of Hive queries, Pig scripts, Mahout jobs, or in other scenarios

Local HDFS is necessary at times

Local HDFS is a good option if:

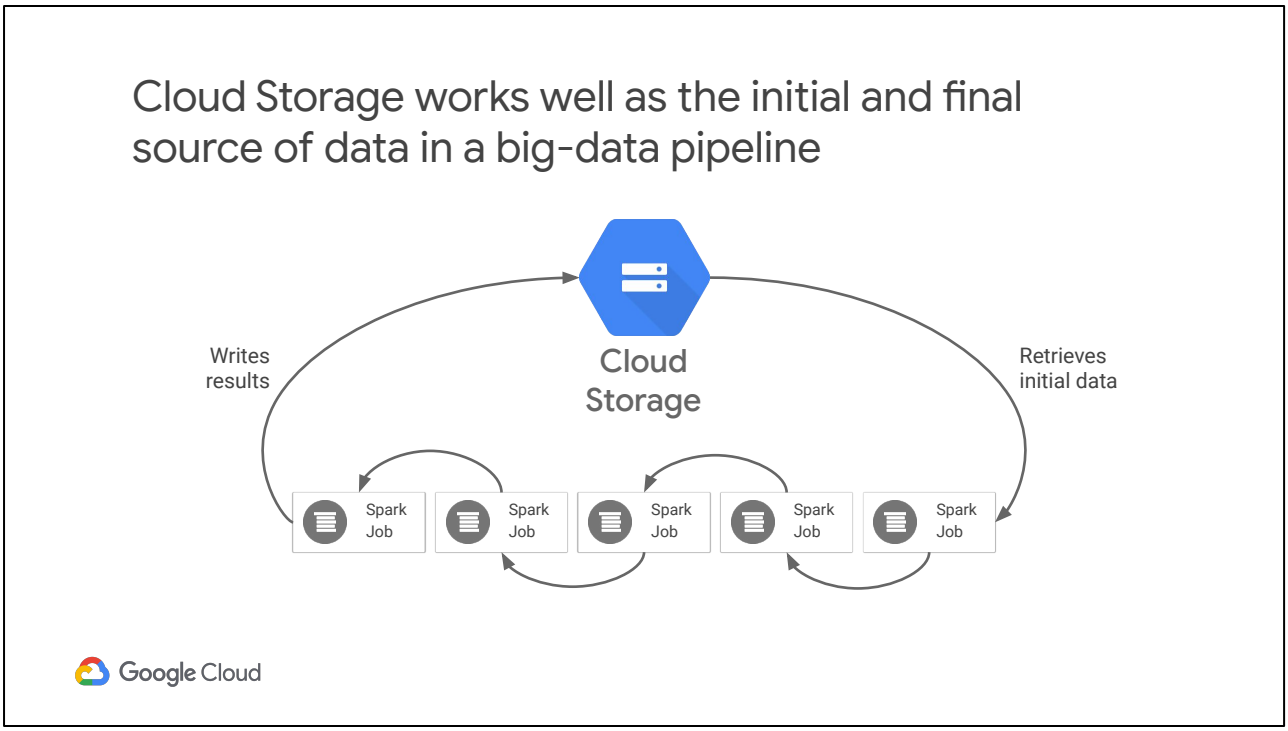
- Your jobs require a lot of metadata operations
- You modify the HDFS data frequently or you rename directories.
- You heavily use the append operation on HDFS files.
- You have workloads that involve heavy I/O.
- You have I/O workloads that are especially sensitive to latency.

Local HDFS is a good option if:

- Your jobs require a lot of metadata operations—for example, you have thousands of partitions and directories, and each file size is relatively small.
- You modify the HDFS data frequently or you rename directories. (Cloud Storage objects are immutable, so renaming a directory is an expensive operation because it consists of copying all objects to a new key and deleting them afterwards.)
- You heavily use the append operation on HDFS files.
- You have workloads that involve heavy I/O. For example, you have a lot of partitioned writes, such as the following:

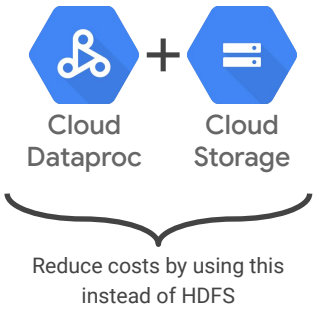
  ```
  spark.read().write.partitionBy(...).parquet("gs://")
  ```

- You have I/O workloads that are especially sensitive to latency. For example, you require single-digit millisecond latency per storage operation.



In general, we recommend using Cloud Storage as the initial and final source of data in a big-data pipeline. For example, if a workflow contains five Spark jobs in series, the first job retrieves the initial data from Cloud Storage and then writes shuffle data and intermediate job output to HDFS. The final Spark job writes its results to Cloud Storage.

Using Cloud Dataproc with Cloud Storage allows you to reduce the disk requirements and save costs



Here are some options to adjust the size of the local HDFS:

- Decrease the total size of the local HDFS by decreasing the size of primary persistent disks for the master and workers.
- Increase the total size of the local HDFS by increasing the size of primary persistent disk for workers.
- Attach up to eight SSDs (375 GB each) to each worker and use these disks for the HDFS.
- Use SSD persistent disks for your master or workers as a primary disk.

Google Cloud

Using Cloud Dataproc with Cloud Storage allows you to reduce the disk requirements and save costs by putting your data there instead of in the HDFS. When you keep your data on Cloud Storage and don't store it on the local HDFS, you can use smaller disks for your cluster. By making your cluster truly on-demand, you're also able to separate storage and compute, as noted earlier, which helps you reduce costs significantly.

Even if you store all of your data in Cloud Storage, your Cloud Dataproc cluster needs HDFS for certain operations such as storing control and recovery files, or aggregating logs. It also needs non-HDFS local disk space for shuffling. You can reduce the disk size per worker if you are not heavily using the local HDFS.

Here are some options to adjust the size of the local HDFS:

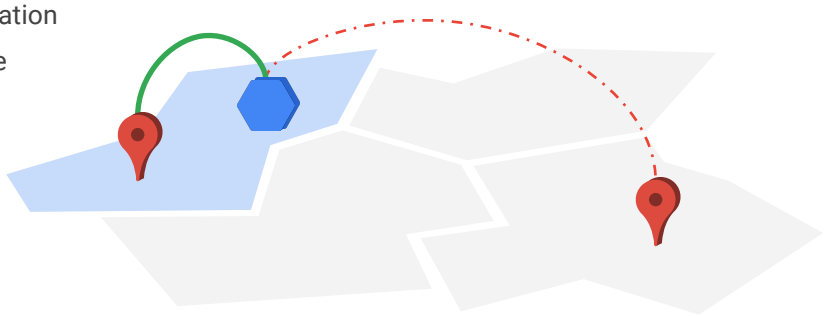
Decrease the total size of the local HDFS by decreasing the size of primary persistent disks for the master and workers. The primary persistent disk also contains the boot volume and system libraries, so allocate at least 100 GB.
Increase the total size of the local HDFS by increasing the size of primary persistent disk for workers. Consider this option carefully— it's rare to have workloads that get better performance by using HDFS with standard persistent disks in comparison to using Cloud Storage or local HDFS with SSD.
Attach up to eight SSDs (375 GB each) to each worker and use these disks for the HDFS. This is a good option if you need to use the HDFS for I/O-intensive workloads and you need single-digit millisecond latency. Make sure that you use a machine type

that has enough CPUs and memory on the worker to support these disks.
Use SSD persistent disks (PD_SSD, currently in beta) for your master or workers as a primary disk.

Geographical regions can impact the efficiency of your solution

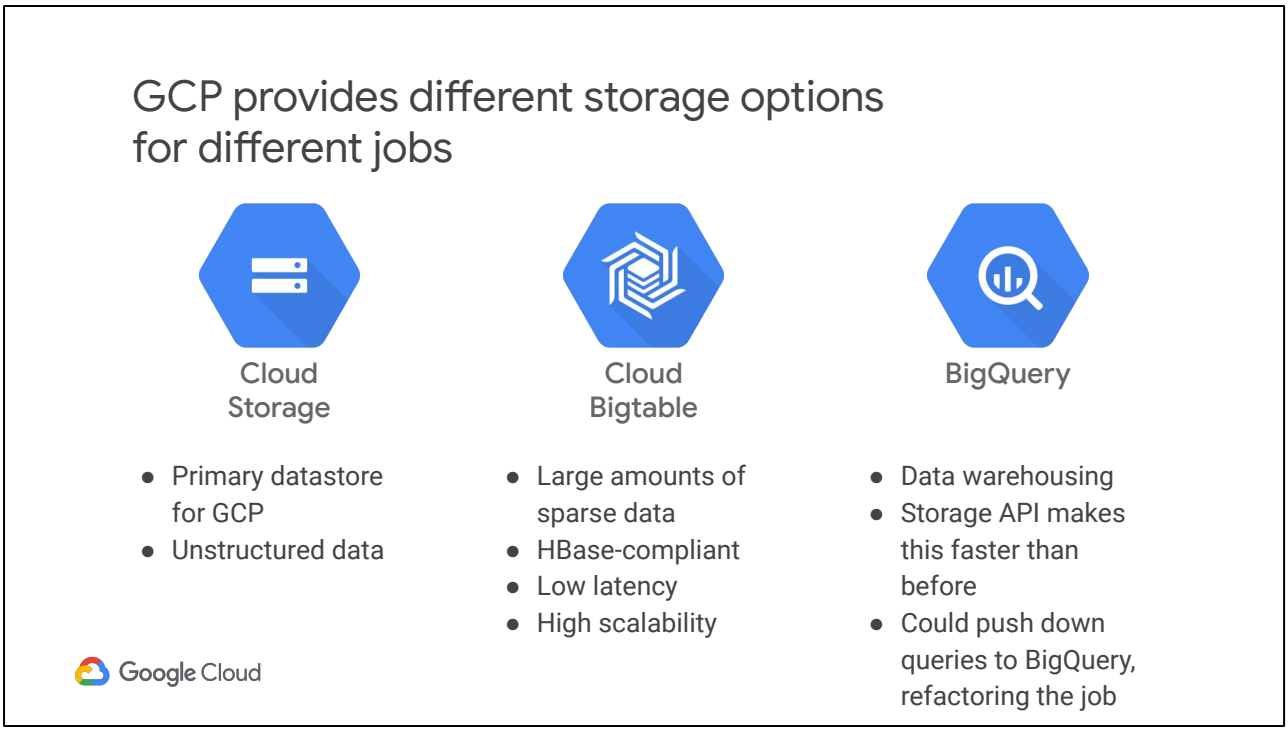
Regions can have repercussions for your jobs, such as:

- Request latency
- Data proliferation
- Performance



Google Cloud

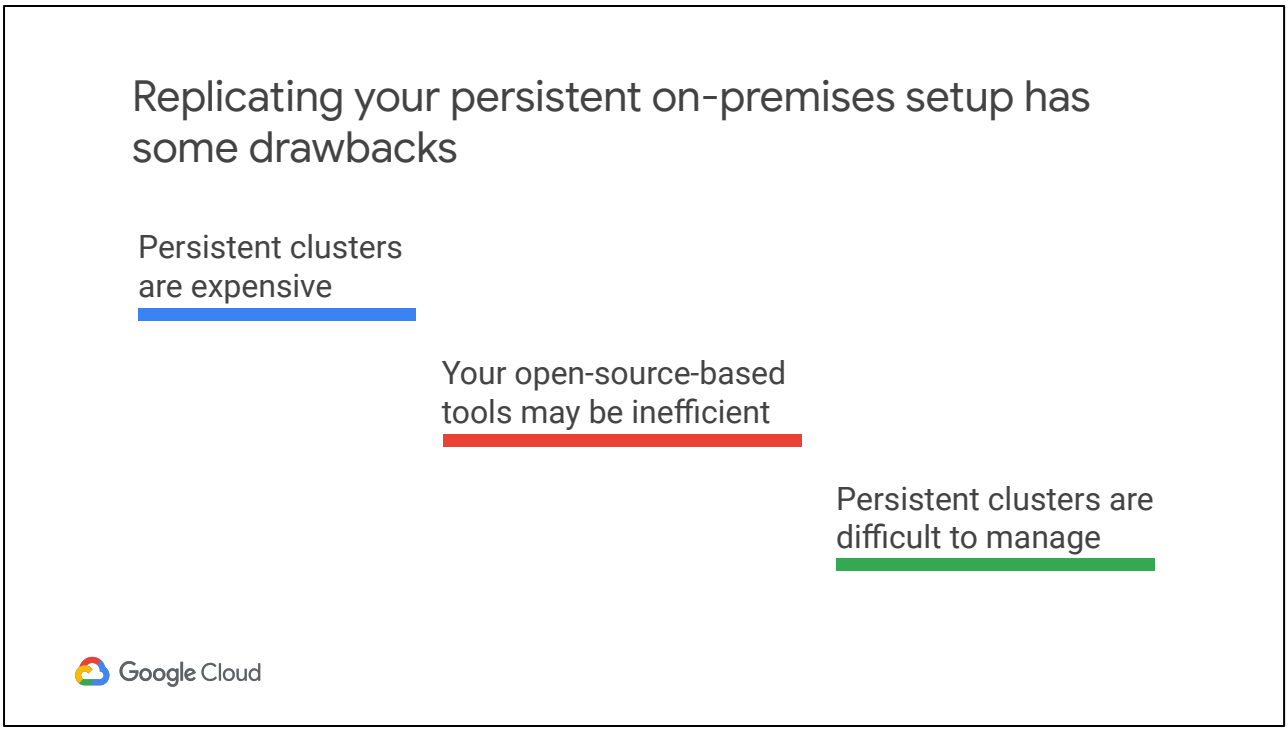
You should understand the repercussions of geography and regions before you configure your data and jobs. Many GCP services require you to specify regions or zones in which to allocate resources. The latency of requests can increase when the requests are made from a different region than the one where the resources are stored. Additionally, if the service's resources and your persistent data are located in different regions, some calls to GCP services might copy all of the required data from one zone to another before processing. This can have a severe impact on performance.



Cloud Storage is the primary way to store unstructured data in GCP, but it isn't the only storage option. Some of your data might be better suited to storage in products designed explicitly for big data.

You can use Cloud Bigtable to store large amounts of sparse data. Cloud Bigtable is an HBase-compliant API that offers low latency and high scalability to adapt to your jobs.

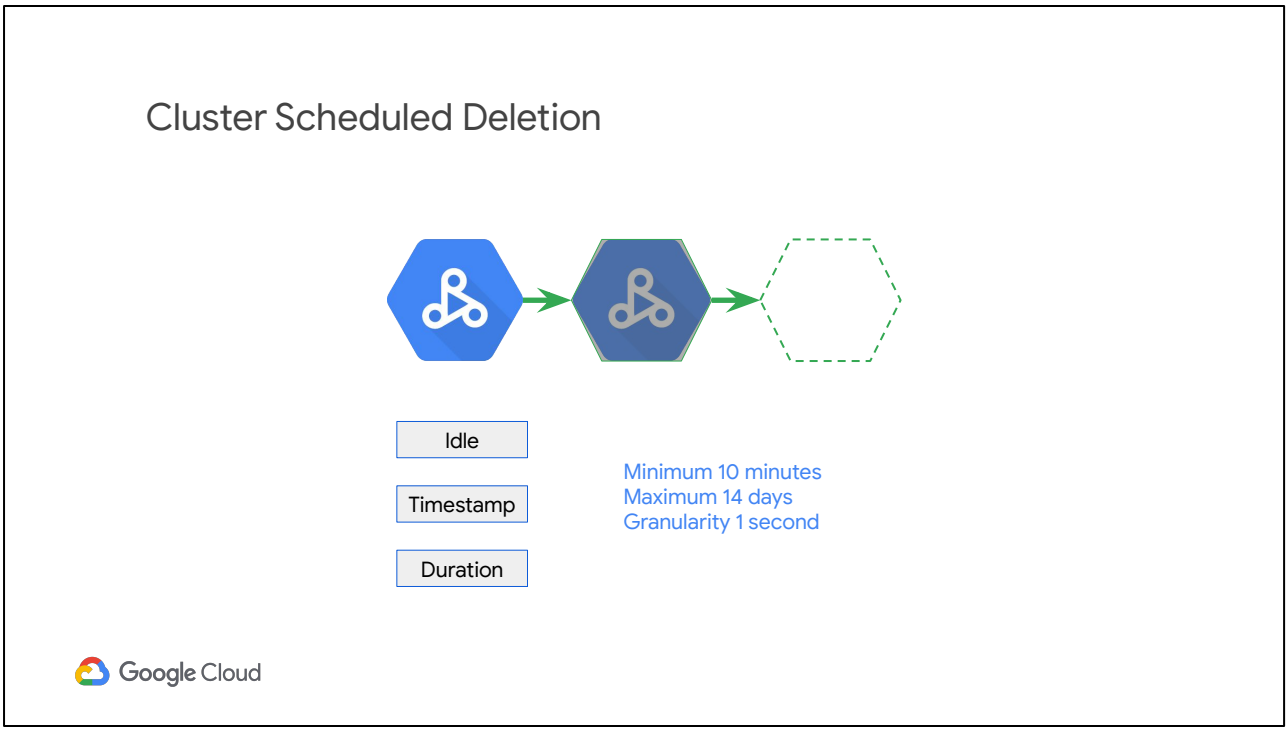
For data warehousing, you can use BigQuery.



Because Cloud Dataproc runs Hadoop on GCP, using a persistent Cloud Dataproc cluster to replicate your on-premises setup might seem like the easiest solution. However, there are some limitations to that approach:

- Keeping your data in a persistent HDFS cluster using Cloud Dataproc is more expensive than storing your data in Cloud Storage, which is what we recommend. Keeping data in an HDFS cluster also limits your ability to use your data with other GCP products.
- Augmenting or replacing some of your open-source-based tools with other related GCP services can be more efficient or economical for particular use cases.
- Using a single, persistent Cloud Dataproc cluster for your jobs is more difficult to manage than shifting to targeted clusters that serve individual jobs or job areas.

The most cost-effective and flexible way to migrate your Hadoop system to GCP is to shift away from thinking in terms of large, multi-purpose, persistent clusters and instead think about small, short-lived clusters that are designed to run specific jobs. You store your data in Cloud Storage to support multiple, temporary processing clusters. This model is often called the ephemeral model, because the clusters you use for processing jobs are allocated as needed and are released as jobs finish.



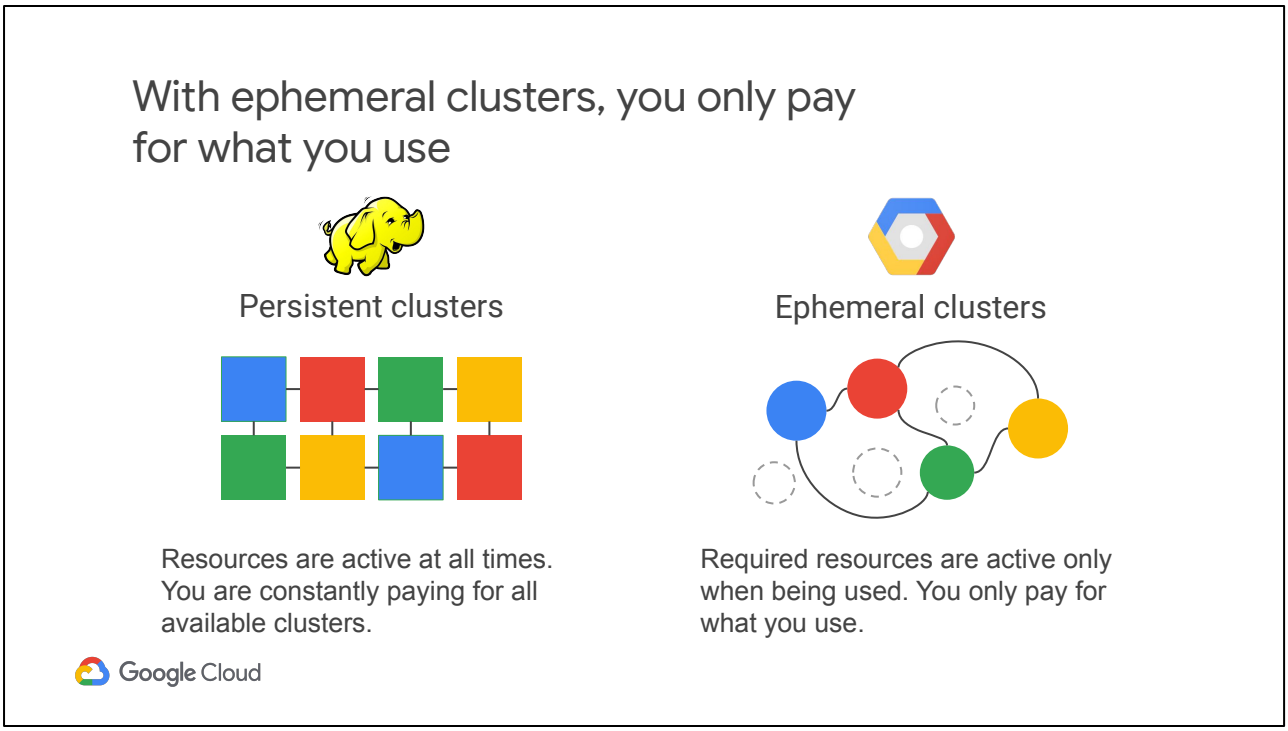
Efficient utilization, don't pay for resources you don't use.

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 an 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".

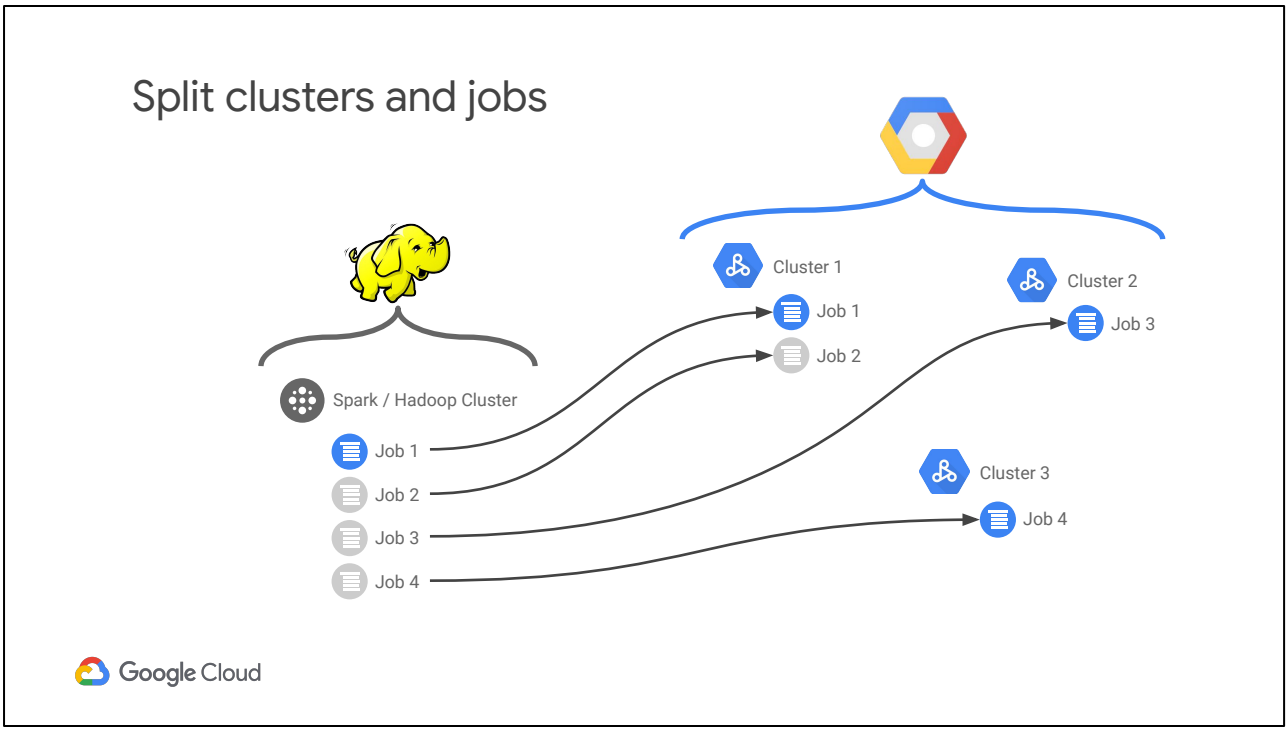


The biggest shift in your approach between running an on-premises Hadoop workflow and running the same workflow on GCP is the shift away from monolithic, persistent clusters to specialized, ephemeral clusters. You spin up a cluster when you need to run a job and then delete it when the job completes. The resources required by your jobs are active only when they're being used, so you only pay for what you use. This approach enables you to tailor cluster configurations for individual jobs. Because you aren't maintaining and configuring a persistent cluster, you reduce the costs of resource use and cluster administration.
This section describes how to move your existing Hadoop infrastructure to an ephemeral model.

To get the most from Cloud Dataproc, customers need to move to an “ephemeral” model of only using clusters when they need them.
This can be scary because a persistent cluster is comfortable.
With GCS data persistence and fast boot of Cloud Dataproc, however, a persistent cluster is a waste of resources.
If a persistent cluster is needed, make it small. Clusters can be resized anytime.

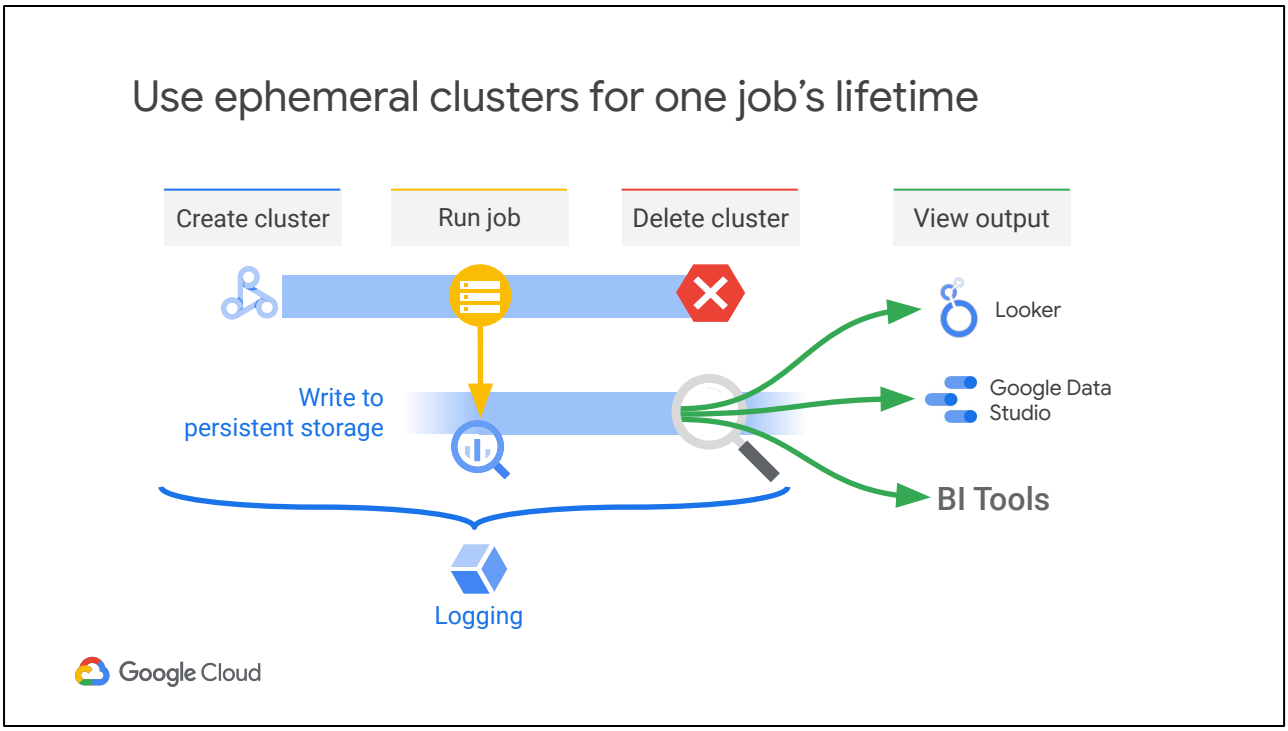
Ephemeral model is the recommended route but it requires storage to be decoupled from compute.

Dunnhumby gave a great talk on moving to ephemeral clusters at
https://youtu.be/2ksD7udWFys



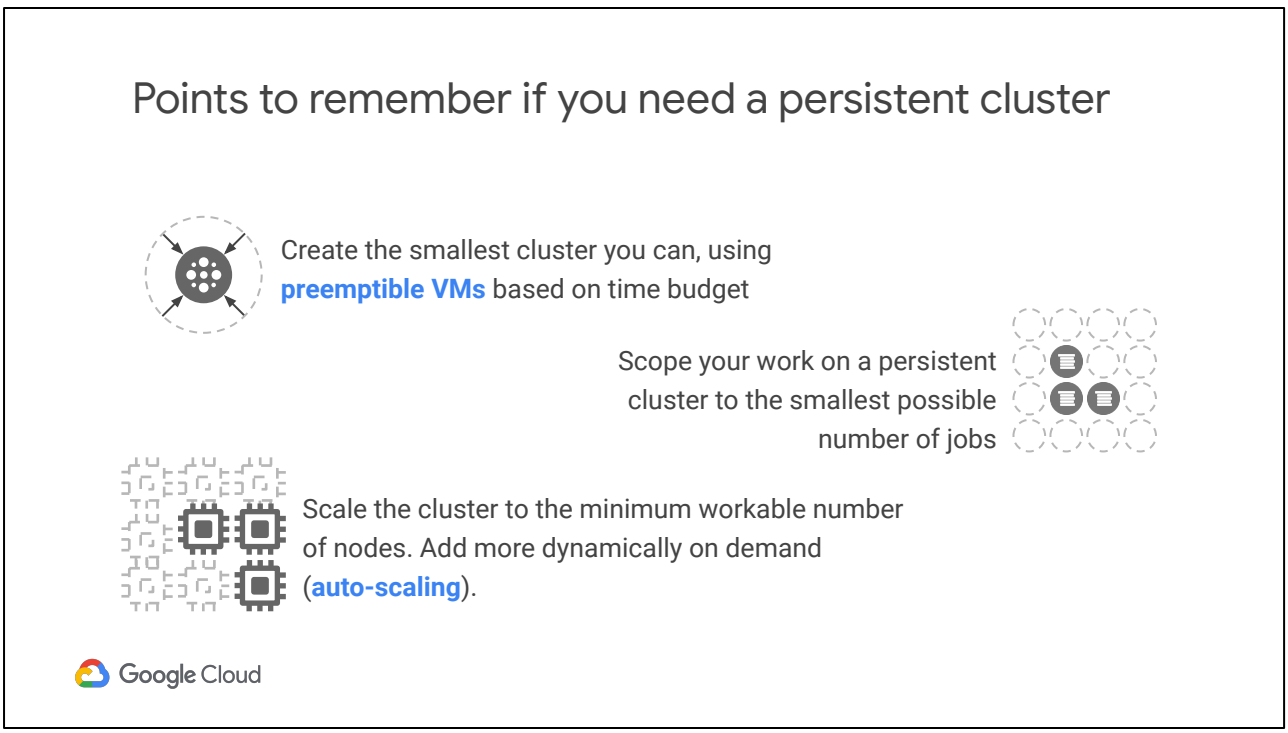
Separate job shapes -> separate clusters. Decompose even further with job-scoped clusters.

Isolate dev, staging, and production environments by running on separate clusters. Read from the same underlying data source on GCS. Add appropriate ACLs to service accounts to protect data.



The point of ephemeral clusters is to use them only for the jobs' lifetime. When it's time to run a job, follow this process:

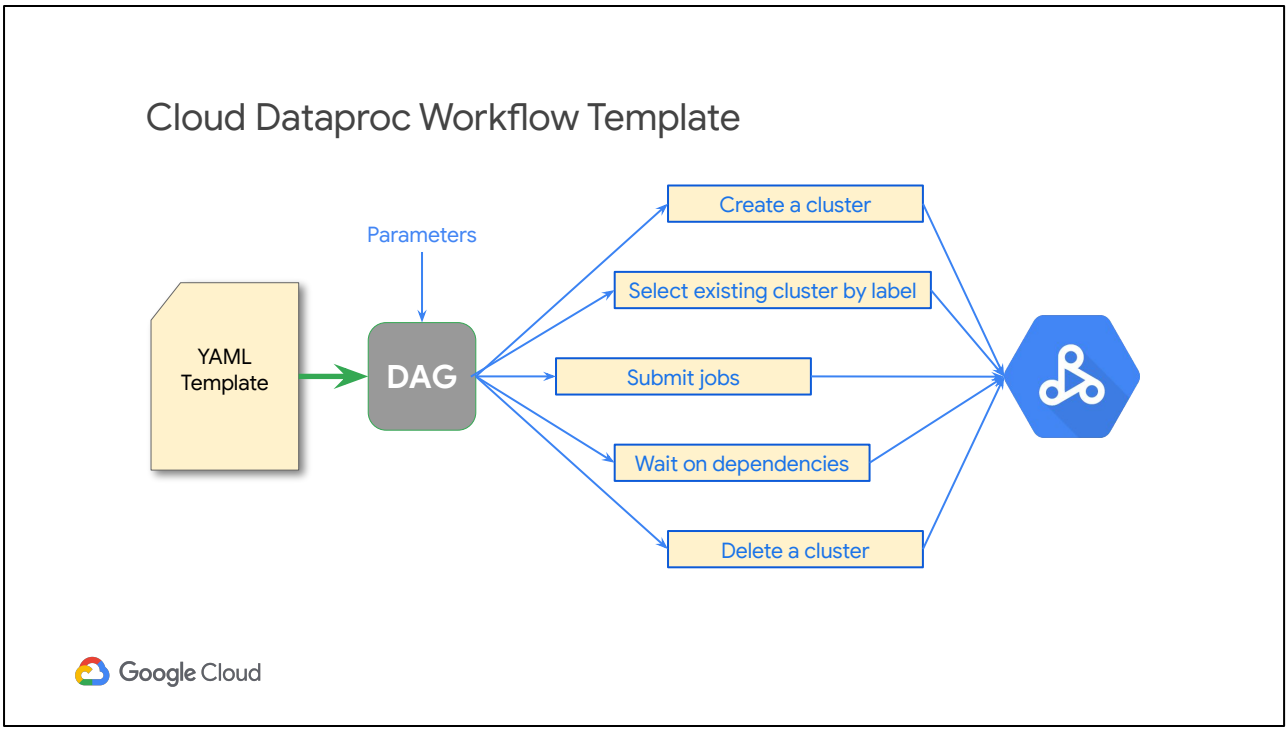
1. Create a properly configured cluster.
2. Run your job, sending output to Cloud Storage or another persistent location.
3. Delete the cluster.
4. Use your job output however you need to.
5. View logs in Cloud Logging or Cloud Storage.



If you can't accomplish your work without a persistent cluster, you can create one. This option may be costly and isn't recommended if there is a way to get your job done on ephemeral clusters.

You can minimize the cost of a persistent cluster by:

- Creating the smallest cluster you can.
- Scoping your work on that cluster to the smallest possible number of jobs.
- Scaling the cluster to the minimum workable number of nodes, adding more dynamically to meet demand.



The Cloud Dataproc Workflow Template is a YAML file that is processed through a Directed Acyclic Graph (DAG). It can create a new cluster, select from an existing cluster, submit jobs, hold jobs for submission until dependencies can complete, and it can delete a cluster when the job is done.

It is currently available through the gcloud command and the REST API, but not through Console.

The Workflow Template becomes active when it is instantiated into the DAG. The Template can be submitted multiple times with different parameter values. You can also write a template inline in the the gcloud command, and you can list workflows and workflow metadata to help diagnose issues.

https://cloud.google.com/dataproc/docs/concepts/workflows/overview
https://cloud.google.com/dataproc/docs/concepts/labels

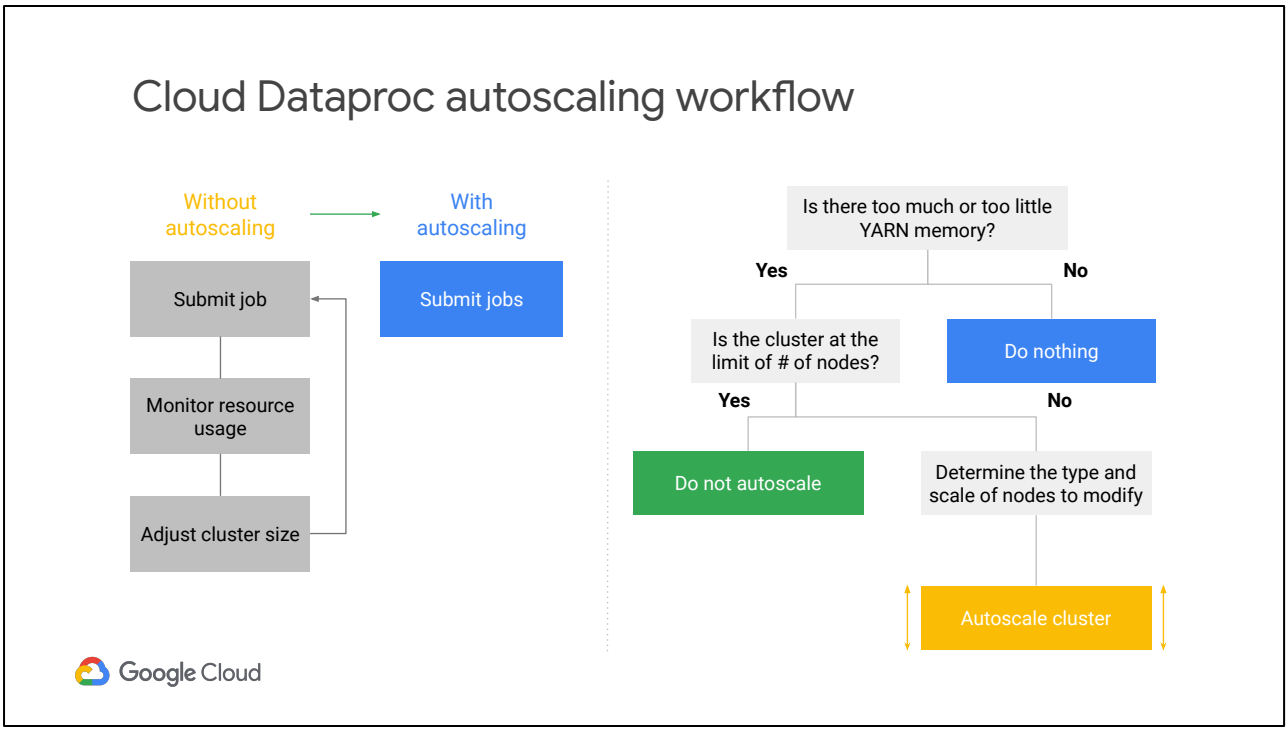
Cloud Dataproc workflow templates

```
# the things we need pip-installed on the cluster
STARTUP_SCRIPT=gs://${BUCKET}/sparktobq/startup_script.sh
echo "pip install --upgrade --quiet google-compute-engine google-cloud-storage matplotlib" >
/tmp/startup_script.sh
gsutil cp /tmp/startup_script.sh $STARTUP_SCRIPT

# create new cluster for job
gcloud dataproc workflow-templates set-managed-cluster $TEMPLATE \
    --master-machine-type $MACHINE_TYPE \
    --worker-machine-type $MACHINE_TYPE \
    --initialization-actions $STARTUP_SCRIPT \
    --num-workers 2 \
    --image-version 1.4 \
    --cluster-name $CLUSTER

# steps in job
gcloud dataproc workflow-templates add-job \
  pyspark gs://$BUCKET/spark_analysis.py \
  --step-id create-report \
  --workflow-template $TEMPLATE \
  -- --bucket=$BUCKET


# submit workflow template
gcloud dataproc workflow-templates instantiate $TEMPLATE
```

Google Cloud

Here's an example of a Cloud Dataproc workflow template. First, we get all the things that need to be installed in the cluster using our startup scripts and manually echoing pip install commands like the one seen here to install matplotlib. You can have multiple startup shell scripts run like you see in this example.

Next, we use the gcloud command for creating a new cluster in advance of running our job. We specify cluster parameters like the template to be used in our desired architecture and what machine types and image versions we want for hardware and software. After that, we need to add a job to the newly created cluster. In this example, we have a Spark job written in Python that exists in a GCS bucket that we control. Lastly, we need to submit this template itself as a new workflow template as you see with the last command.



Cloud Dataproc autoscaling provides clusters that size themselves to the needs of the enterprise. Key features include:

- Jobs are “fire and forget”
- There’s no need to manually intervene when a cluster is over or under capacity
- You can choose between standard and preemptible workers, and
- You can save resources (quota and cost) at any point in time

Autoscaling policies provide fine-grained control. This is based on the difference between YARN pending and available memory. If more memory is needed, then you scale up. If there’s excess memory, you scale down. Obey VM limits and scale based on scale factor.

Autoscaling improvements can be summarized as follows:

Even more fine-grained controls:

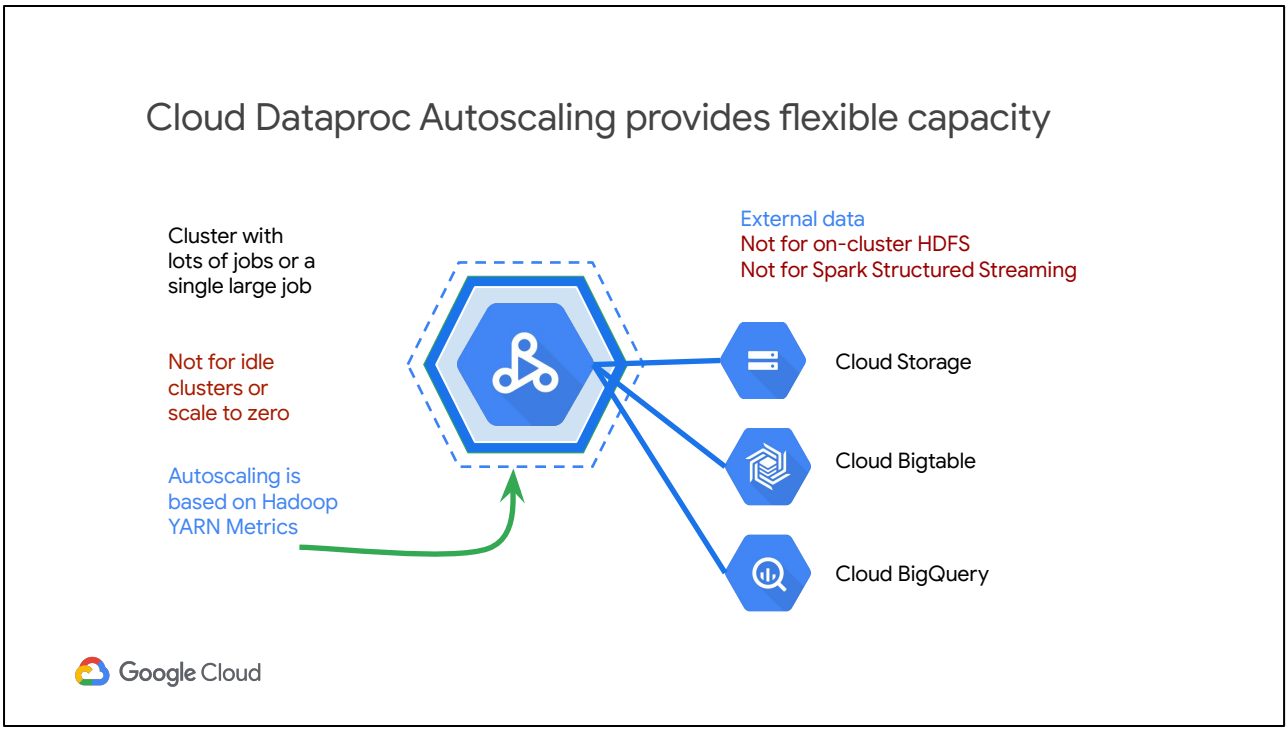
- Autoscaling policies can be updated or removed at any time
- The minimum scaling interval has been reduced from 10 min to 2 min
- Autoscaling policies can be shared between multiple clusters

Easier to understand:

- YARN and HDFS dashboards in cluster page
- Autoscaling decision history is available in Cloud Logging

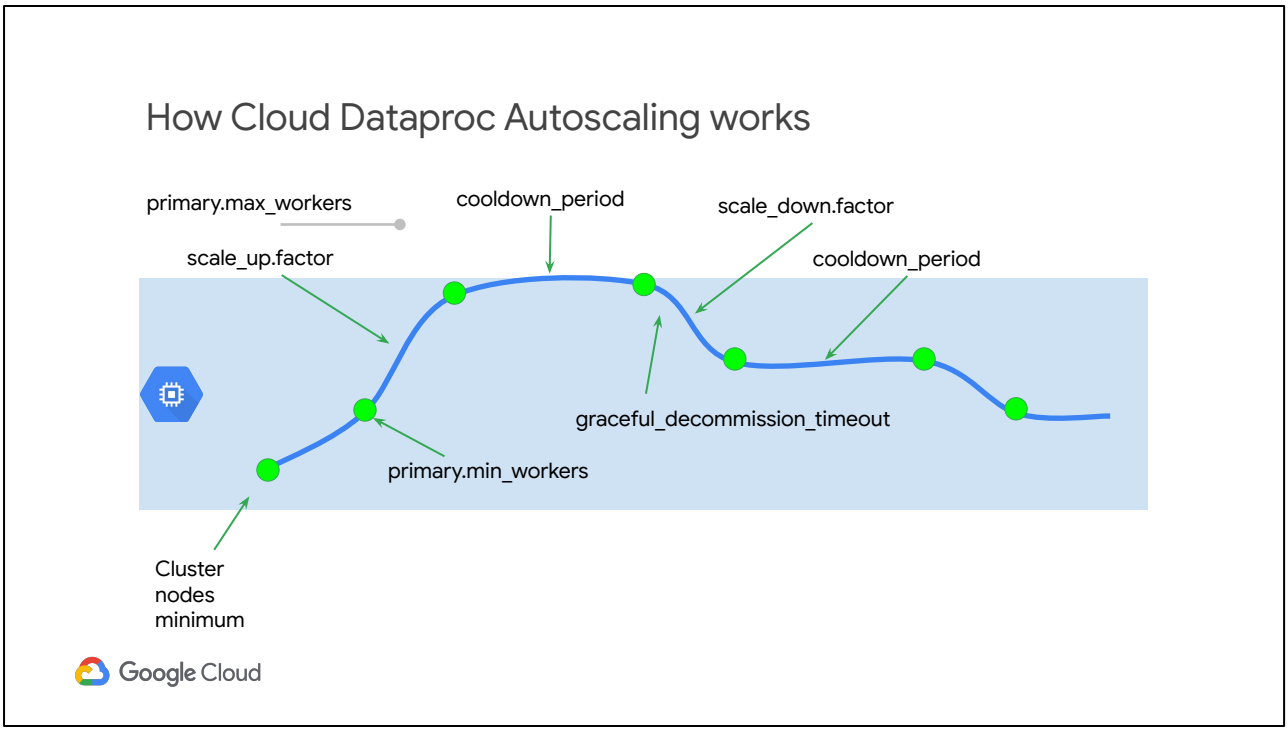
Job stability:

- Scale MapReduce and Spark jobs without losing progress



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.



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.

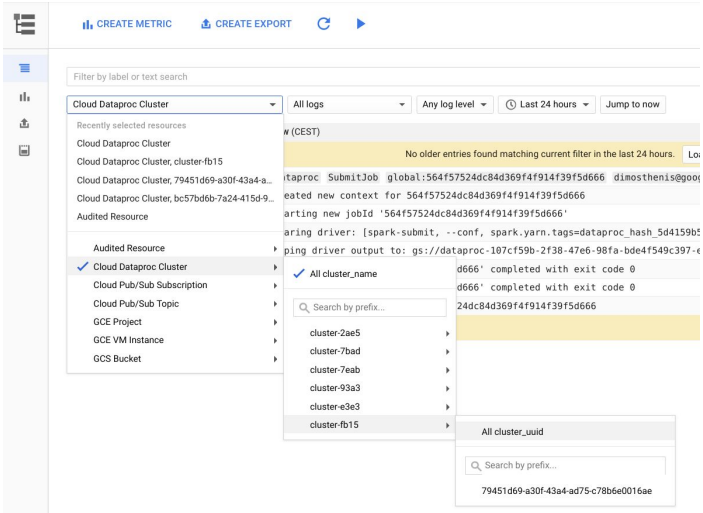
You can read about the scaling algorithm here:
https://cloud.google.com/dataproc/docs/concepts/configuring-clusters/autoscaling

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.

Use Cloud Operations logging and performance monitoring



Google Cloud

In Google Cloud, you can use Cloud Logging and Cloud Monitoring to view and customize logs, and to monitor jobs and resources. The best way to find what error caused a Spark job failure is to look at the driver output and the logs generated by the Spark. Note: if you submit the Spark job by connecting directly to the master node using SSH, it's not possible to get the driver output.

You can retrieve the driver program output by using the Cloud Console or by using a gcloud command. The output is also stored in the Cloud Storage bucket of the Dataproc cluster. Note: when you view the driver program output in the Cloud Console, the progress bar is stored as a long line without a newline at the end. Therefore, the first thing the driver program prints will appear at the end of this line. To view this output more easily, you can click line wrapping.

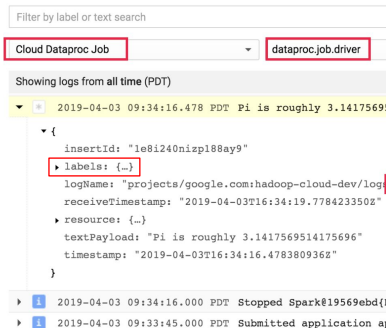
All other logs are located in different files inside the machines of the cluster. It's possible to see the logs for each container from the Spark app web UI (or from the History Server after the program ends) in the executors tab. You need to browse through each Spark container to view each log. If you write logs or print to stdout or stderr in your application code, the logs are saved in the redirection of stdout or stderr.

In a Dataproc cluster, YARN is configured to collect all these logs by default, and they're available in Cloud Logging. Logging provides a consolidated and concise view of all logs so that you don't need to spend time browsing among container logs to find errors.

This screen shows the Logging page in the Cloud Console. You can view all logs from your Dataproc cluster by selecting the cluster's name in the selector menu. Don't forget to expand the time duration in the time-range selector.

You can get logs from a Spark application by filtering for its ID. You can get the application ID from the driver output.

Create labels on clusters and jobs to find logs faster



Google Cloud

To find logs faster, you can create and use your own labels for each cluster or for each Cloud Dataproc job. For example, you can create a label with the key env and the value exploration and use it for your data exploration job. You can then get logs for all exploration job creations by filtering with label:env:exploration in Logging. Note that this filter will not return all logs for this job, only the resource creation logs.

Set the log level

You can set the driver log level using the following gcloud command:

```
gcloud dataproc jobs submit hadoop --driver-log-levels
```

You set the log level for the rest of the application from the Spark context.
For example:

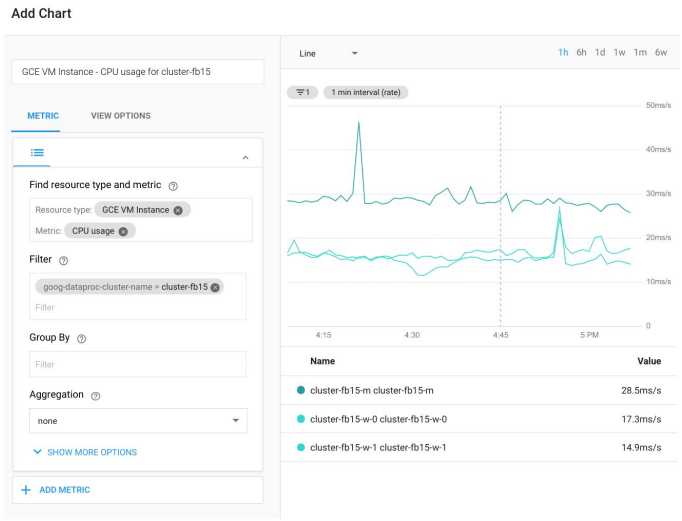
```
spark.sparkContext.setLogLevel("DEBUG")
```

Google Cloud

You can set the driver log level using the following gcloud command:
gcloud dataproc jobs submit hadoop --driver-log-levels

You set the log level for the rest of the application from the Spark context. For example:
spark.sparkContext.setLogLevel("DEBUG")

Monitor your jobs



Google Cloud

Cloud Monitoring can monitor the cluster's CPU, disk, network usage, and YARN resources. You can create a custom dashboard to get up-to-date charts for these and other metrics. Cloud Dataproc runs on top of Compute Engine. If you want to visualize CPU usage, disk I/O, or networking metrics in a chart, you need to select a Compute Engine VM instance as the resource type and then filter by the cluster name. This diagram shows an example of the output.

To view metrics for Spark queries, jobs, stages, or tasks, connect to the Spark application's web UI.

Agenda

The Hadoop ecosystem
Running Hadoop on Cloud Dataproc
GCS instead of HDFS
Optimizing Dataproc
Lab

Running Apache Spark jobs on Cloud Dataproc

Objectives

- Migrate existing Spark jobs to Cloud Dataproc
- Modify Spark jobs to use Cloud Storage instead of HDFS
- Optimize Spark jobs to run on Job specific clusters

Summary

The Hadoop ecosystem
Running Hadoop on Cloud Dataproc
GCS instead of HDFS
Optimizing Dataproc