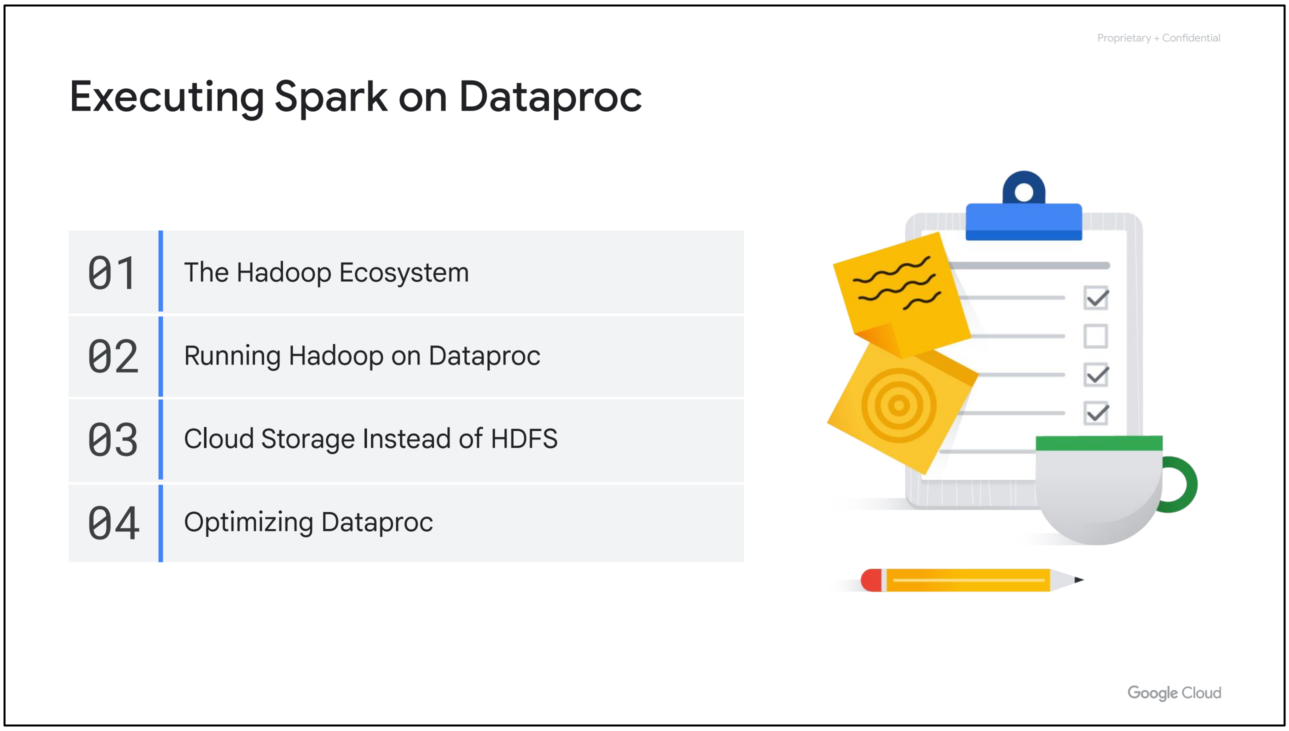
Proprietary + Confidential

02

Executing Spark on Dataproc

Google Cloud

In this module, we will discuss Dataproc, Google Cloud’s managed Hadoop service and in particular Apache Spark.

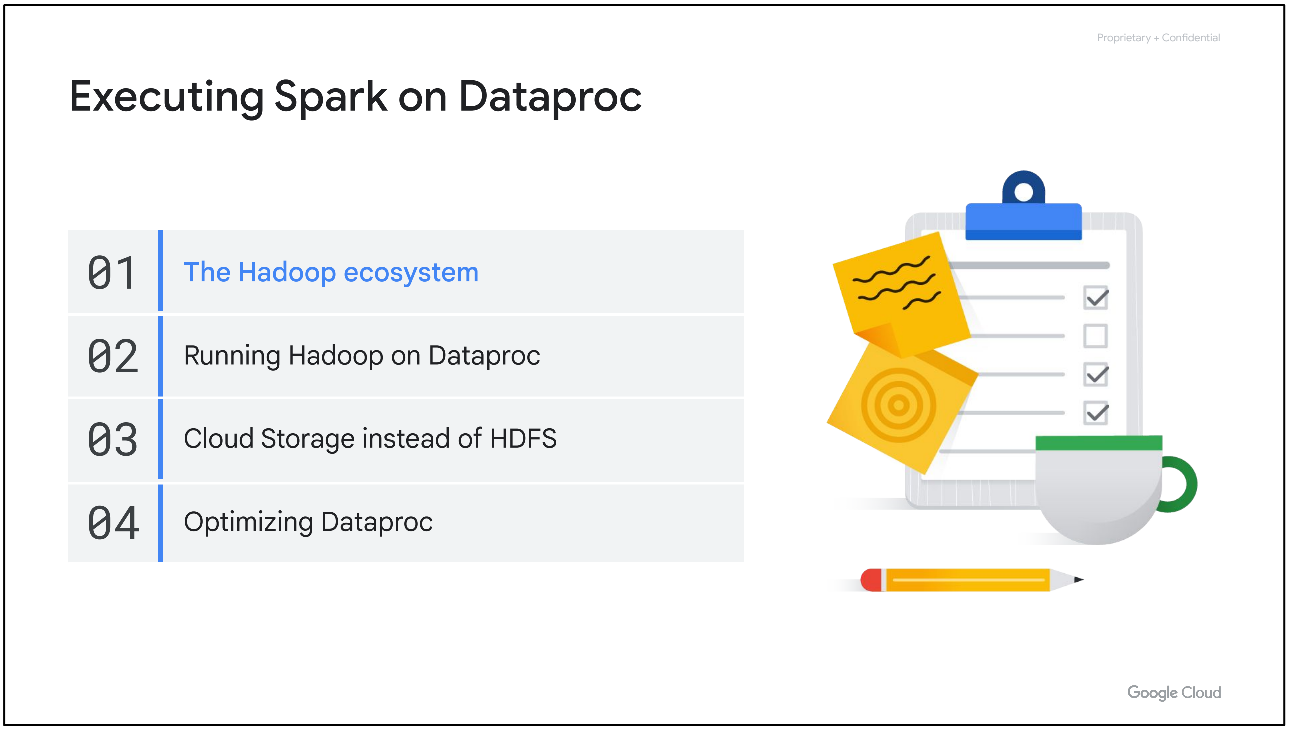


Proprietary + Confidential

Executing Spark on Dataproc

Google Cloud

In this module, we'll cover the Hadoop Ecosystem, learn about running Hadoop on Dataproc, understand the benefits of Cloud Storage instead of HDFS, learn about optimizing Dataproc, and complete a hands-on lab with Apache Spark on Dataproc.

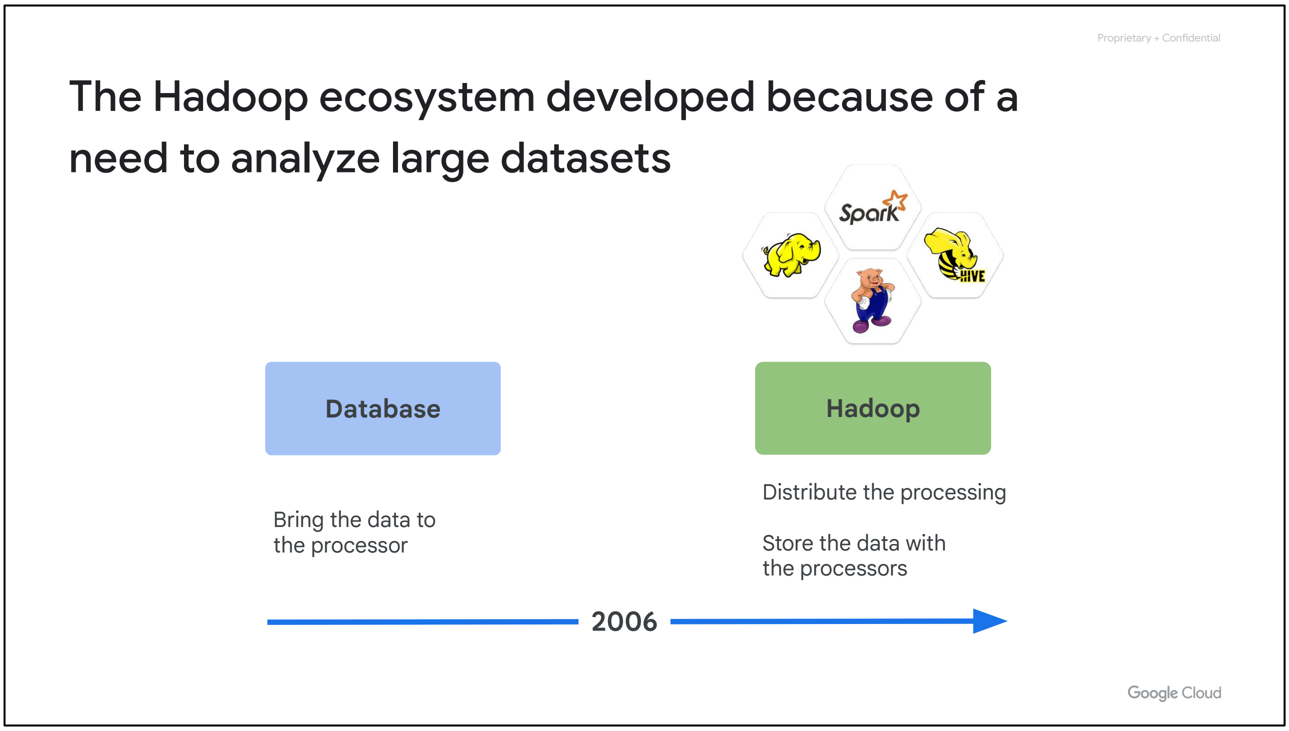


Proprietary + Confidential

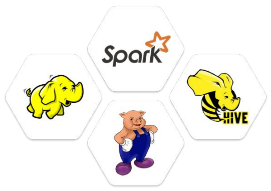
Executing Spark on Dataproc

Google Cloud

Let’s start by looking at the Hadoop ecosystem in a little more detail.



The Hadoop ecosystem developed because of a need to analyze large datasets



Database

Bring the data to the processor

Hadoop

Distribute the processing

Store the data with the processors

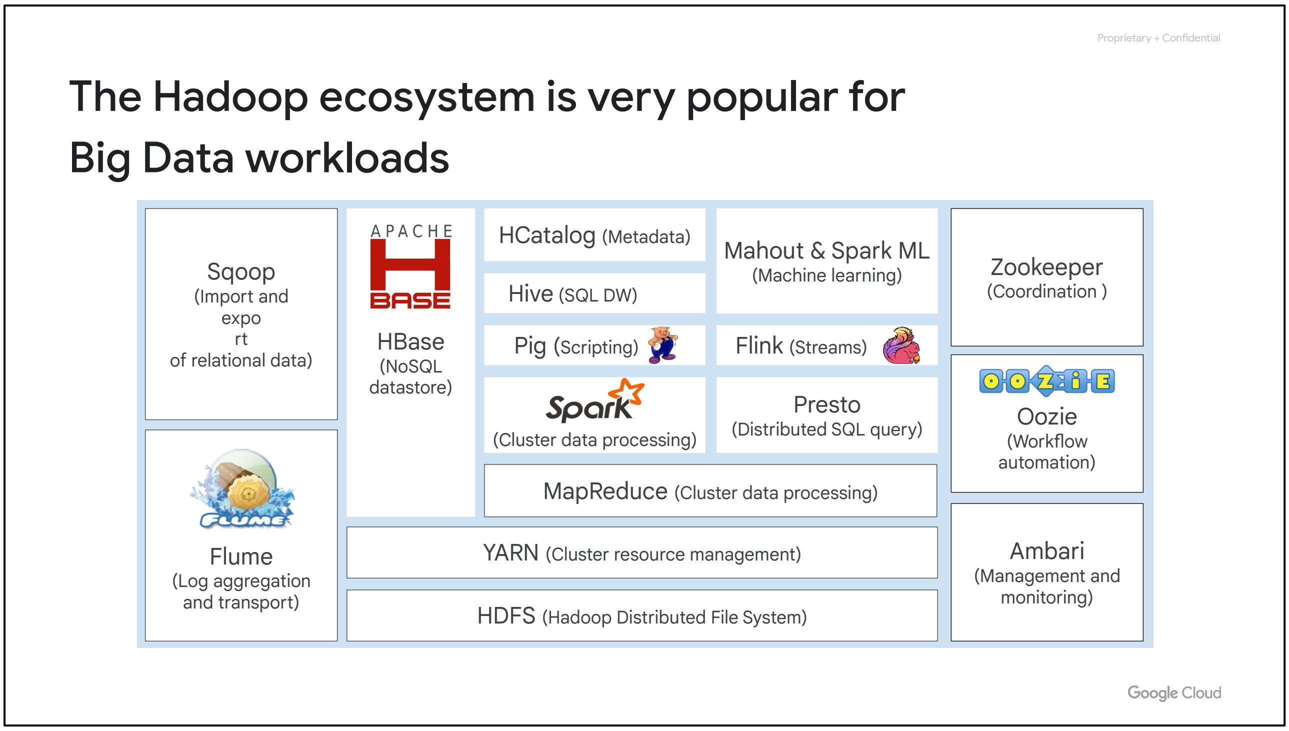
2006

Google Cloud

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 MapReduce provided distributed processing of the data. A whole ecosystem of Hadoop-related software grew up around Hadoop, including Hive, Pig and Spark.



The Hadoop ecosystem is very popular for Big Data workloads

Organizations 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.

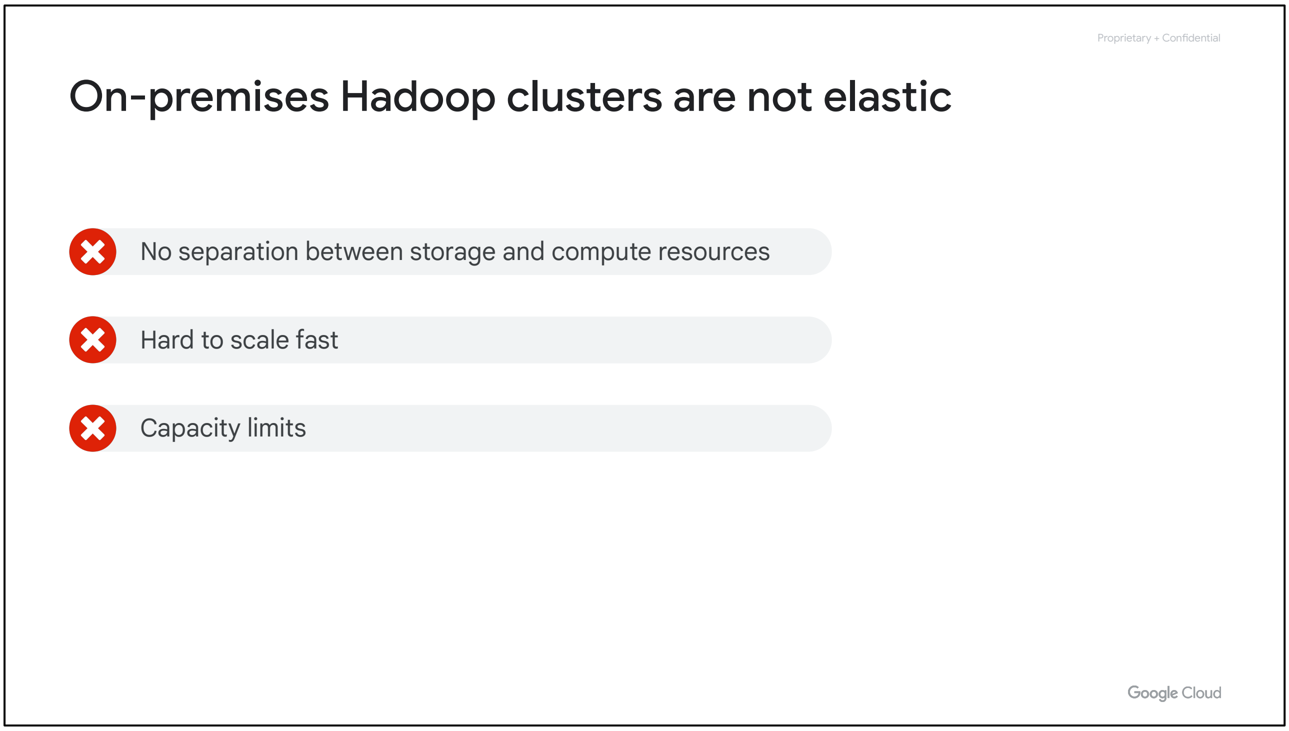
Apache Hadoop is an open source software project that maintains the framework for distributed processing of large data sets across clusters of computers using simple programming models. HDFS is the main file system Hadoop uses for distributing work to nodes on the cluster.

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 few 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 data center. 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.

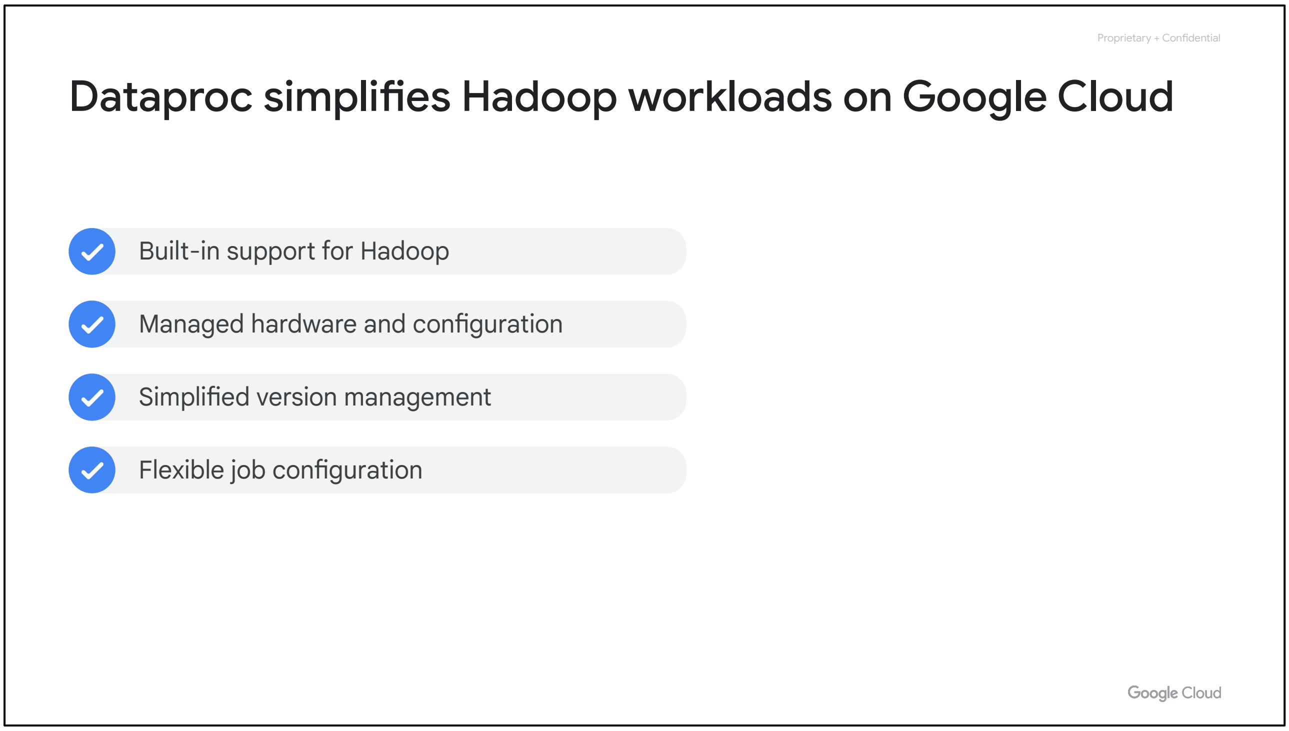
In many cases, using Dataproc as designed will overcome these limitations.



On-premises Hadoop clusters are not elastic

- ✗ No separation between storage and compute resources
- ✗ Hard to scale fast
- ✗ Capacity limits

On-premises Hadoop clusters, due to their physical nature, suffer from limitations. The lack of separation between storage and compute resources results in capacity limits and an inability to scale fast. The only way to increase capacity is to add more physical servers.



Dataproc simplifies Hadoop workloads on Google Cloud

- ✓ Built-in support for Hadoop
- ✓ Managed hardware and configuration
- ✓ Simplified version management
- ✓ Flexible job configuration

Google Cloud

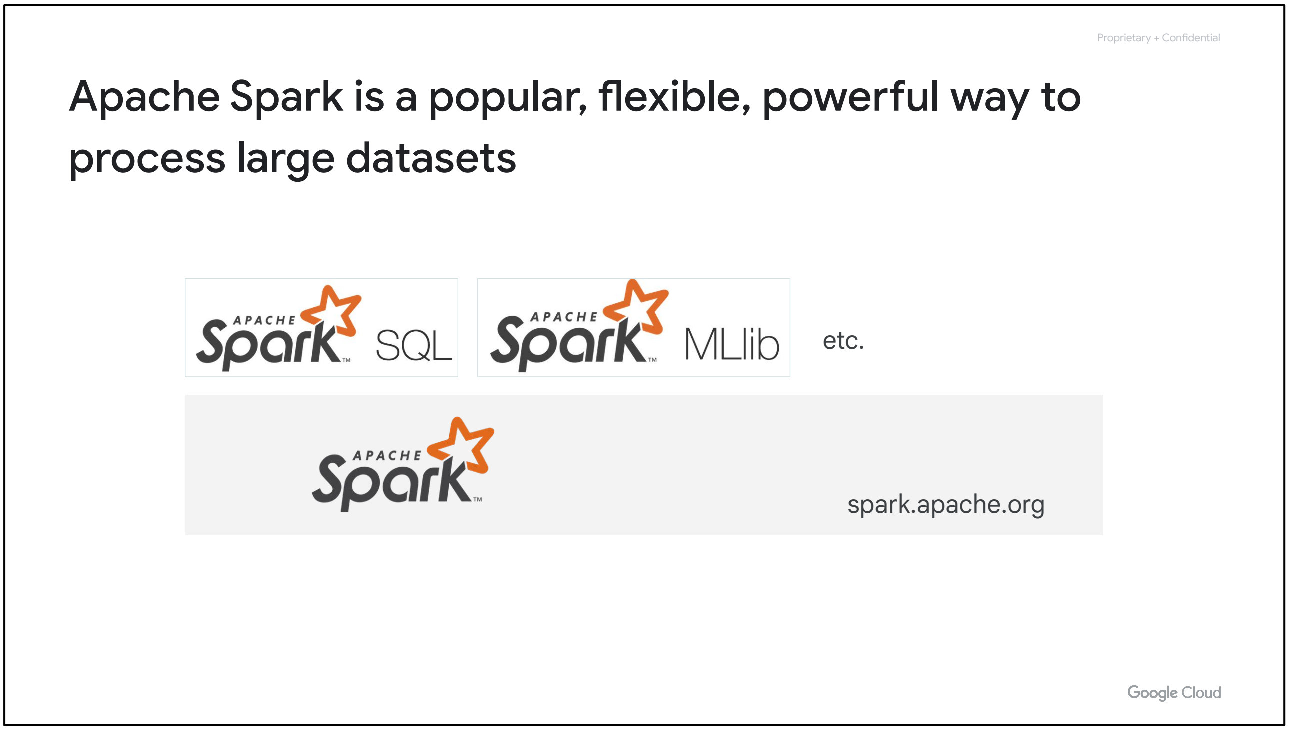
There are many ways in which using Google Cloud can save you time, money, and effort compared to using an on-premises Hadoop solution. In many cases, adopting a cloud-based approach can make your overall solution simpler and easy to manage.

Built-in support for Hadoop
Dataproc is a managed Hadoop and Spark environment. You can use 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 Google Cloud, you never need to worry about physical hardware. You specify the configuration of your cluster, and 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 Dataproc, much of that work is managed for you by Dataproc versioning.

Flexible job configuration
A typical on-premises Hadoop setup uses a single cluster that serves many purposes. When you move to Google Cloud, 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.



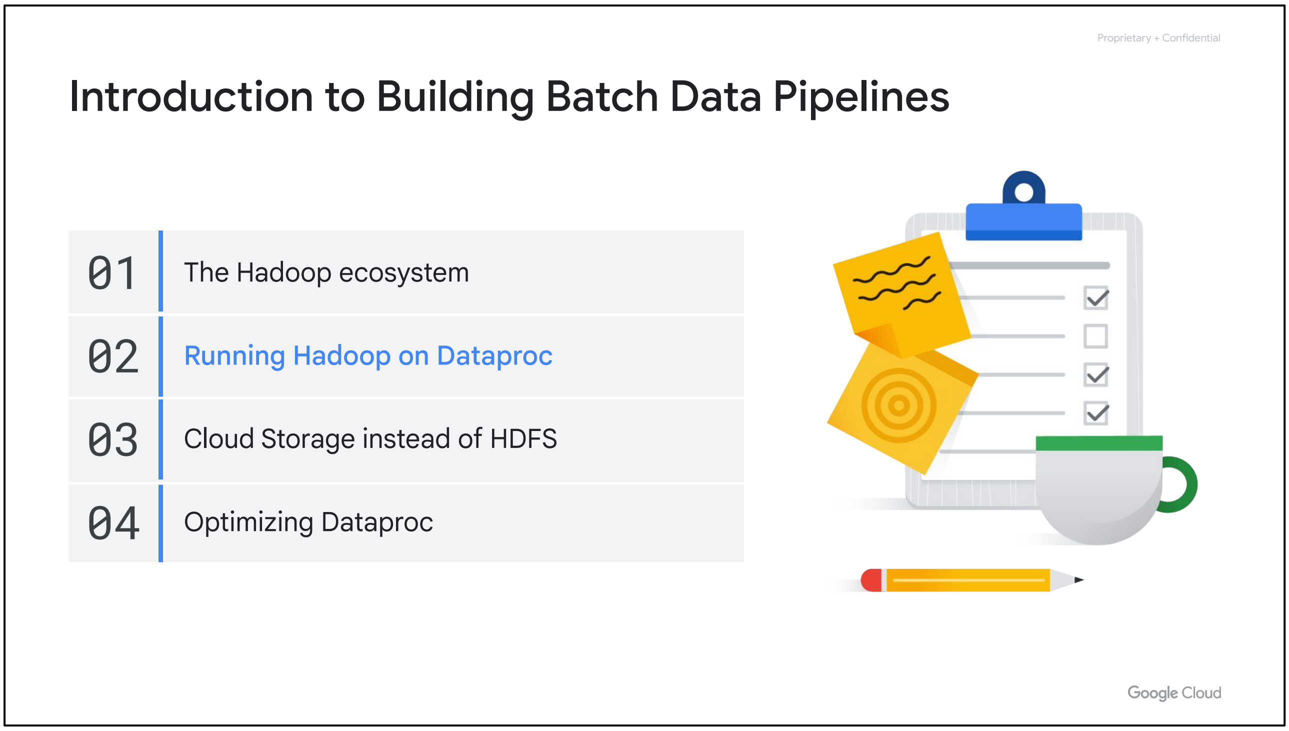
Apache Spark is a popular, flexible, powerful way to process large datasets

SQL
MLlib
etc.
spark.apache.org

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. 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.

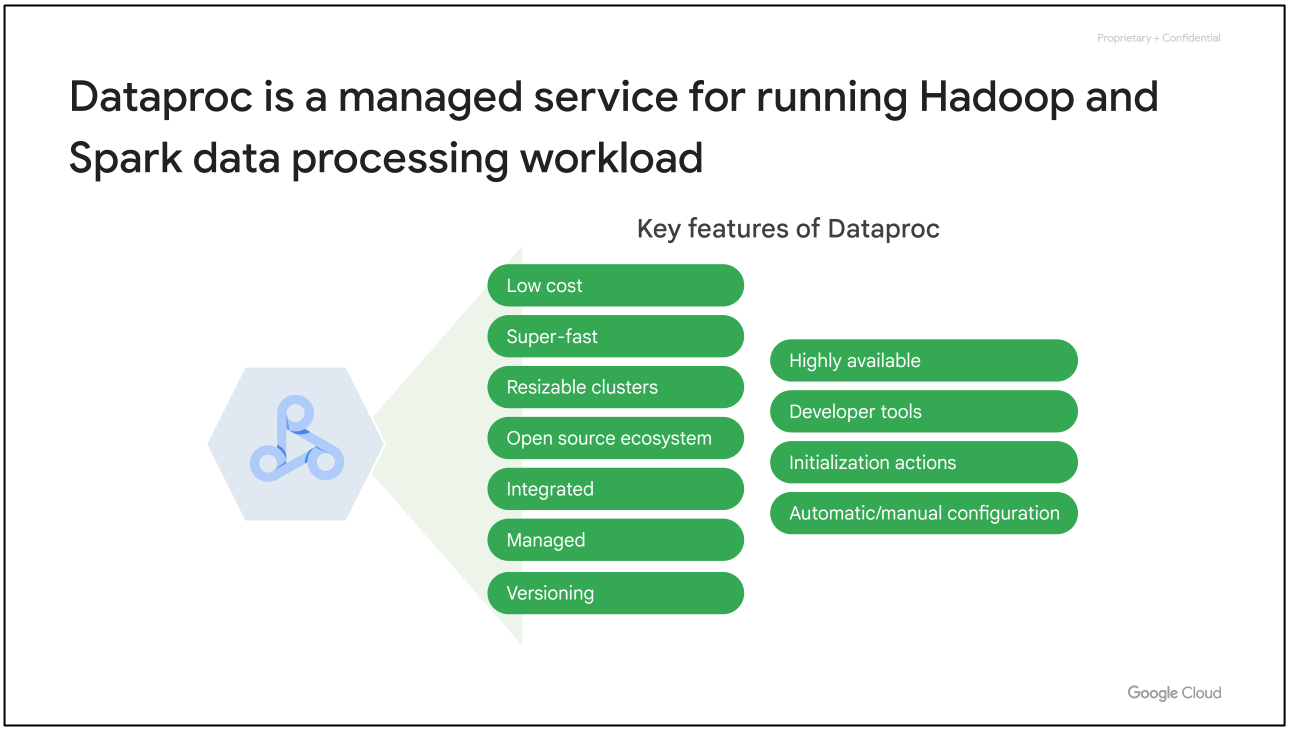
Spark uses a declarative programming model. In imperative programming you tell the system what to do and how to do it. In declarative programming you tell the system what you want and it figures out how to implement it. You will be learning to work with Spark in the labs later.

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.



Introduction to Building Batch Data Pipelines

Next, we'll discuss how and why you should consider processing your same Hadoop job code in the cloud using Dataproc on Google Cloud.



Proprietary + Confidential

Dataproc is a managed service for running Hadoop and Spark data processing workload

Key features of Dataproc

- Low cost
- Super-fast
- Resizable clusters
- Open source ecosystem
- Integrated
- Managed
- Versioning
- Highly available
- Developer tools
- Initialization actions
- Automatic/manual configuration

Google Cloud

Dataproc 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, and 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.

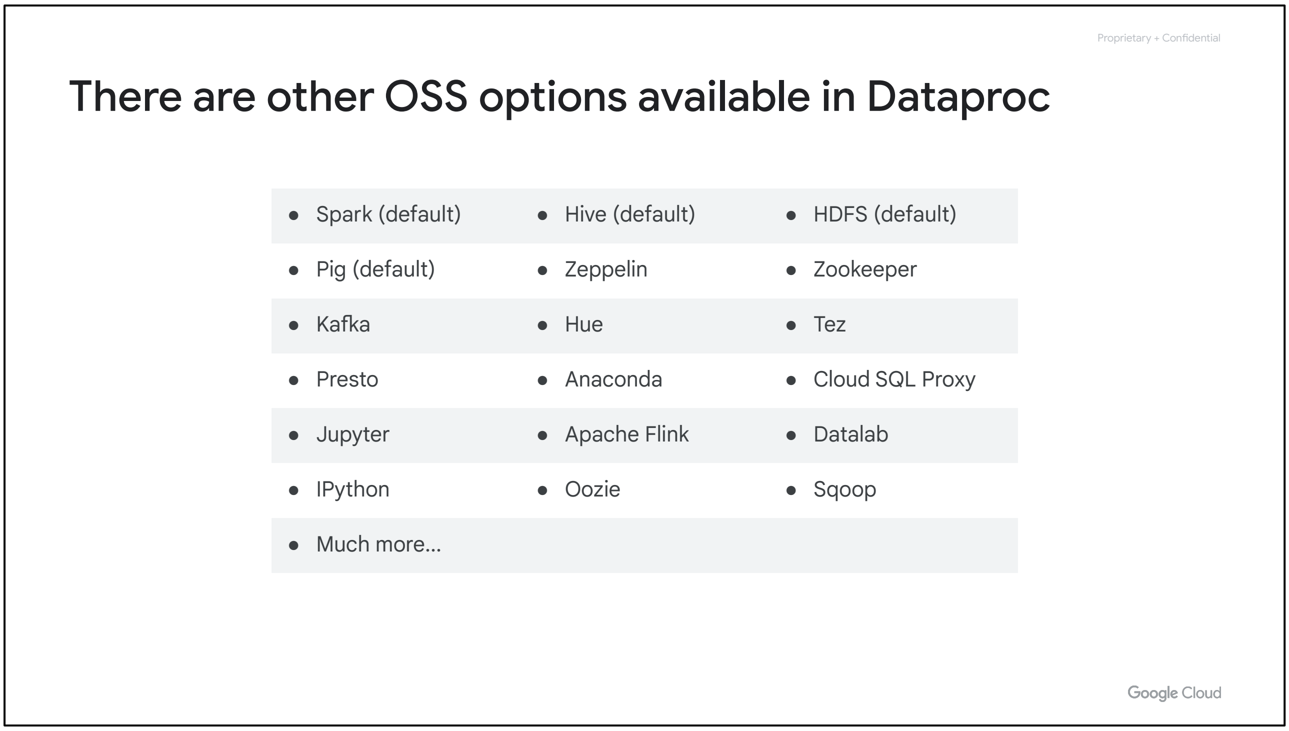
Versioning: Image versioning allows you to switch between different versions of Apache Spark, Apache Hadoop, and other tools.

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.

And **automatic or manual configuration:** Dataproc automatically configures hardware and software on clusters for you while also allowing for manual control.


Proprietary + Confidential

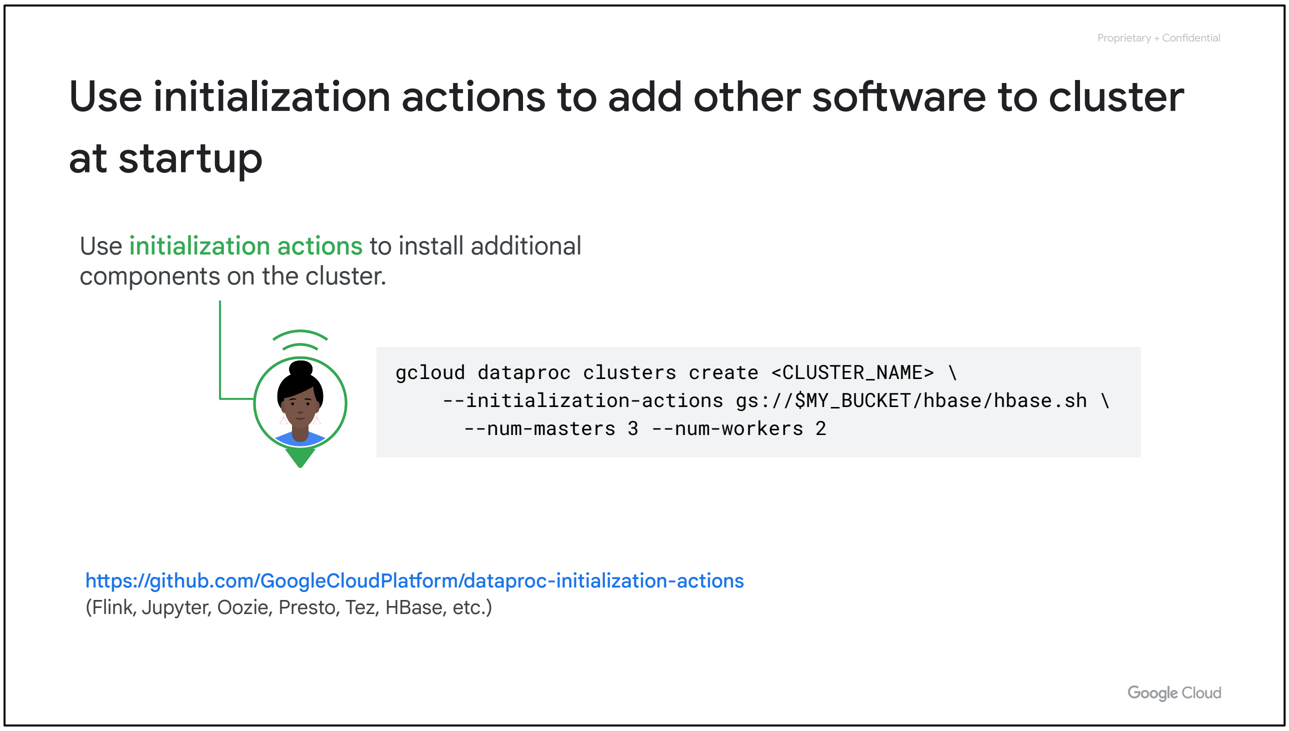
There are other OSS options available in Dataproc

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

Google 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 Dataproc will run on all nodes in your Dataproc cluster immediately after the cluster is set up.



Use initialization actions to add other software to cluster at startup

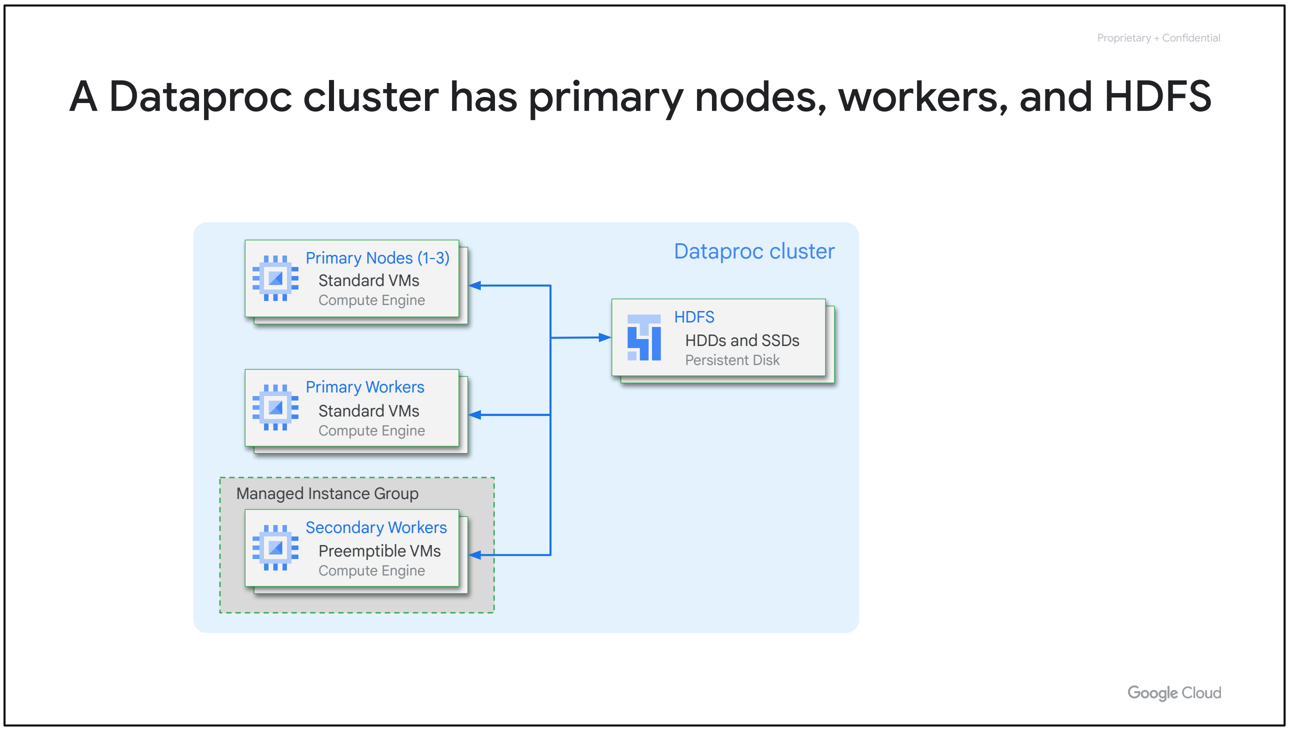
Use initialization actions to install additional components on the cluster.

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

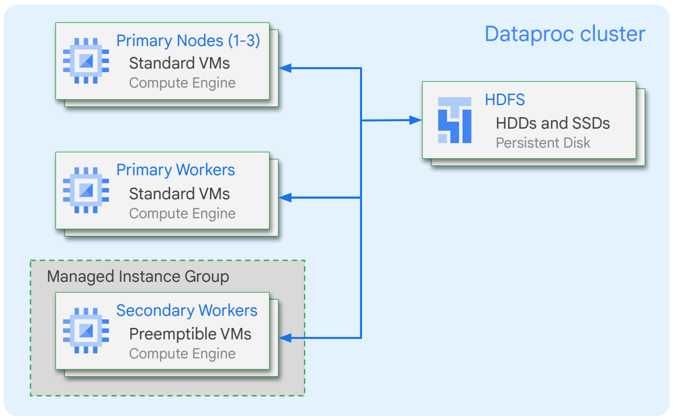
https://github.com/GoogleCloudPlatform/dataproc-initialization-actions
(Flink, Jupyter, Oozie, Presto, Tez, HBase, etc.)

Here's an example of how you can create a Dataproc cluster using the Cloud SDK. And we're going to specify an HBase shell script to run on the clusters initialization. There are a lot of pre-built startup scripts that you can leverage for common Hadoop cluster setup tasks, like Flink, Jupyter and more. You can check out the GitHub repo link to learn more. Do you see the additional parameter for the number of primary and worker nodes in the script? Let's talk more about the architecture of the cluster.

https://github.com/GoogleCloudPlatform/dataproc-initialization-actions



A Dataproc cluster has primary nodes, workers, and HDFS



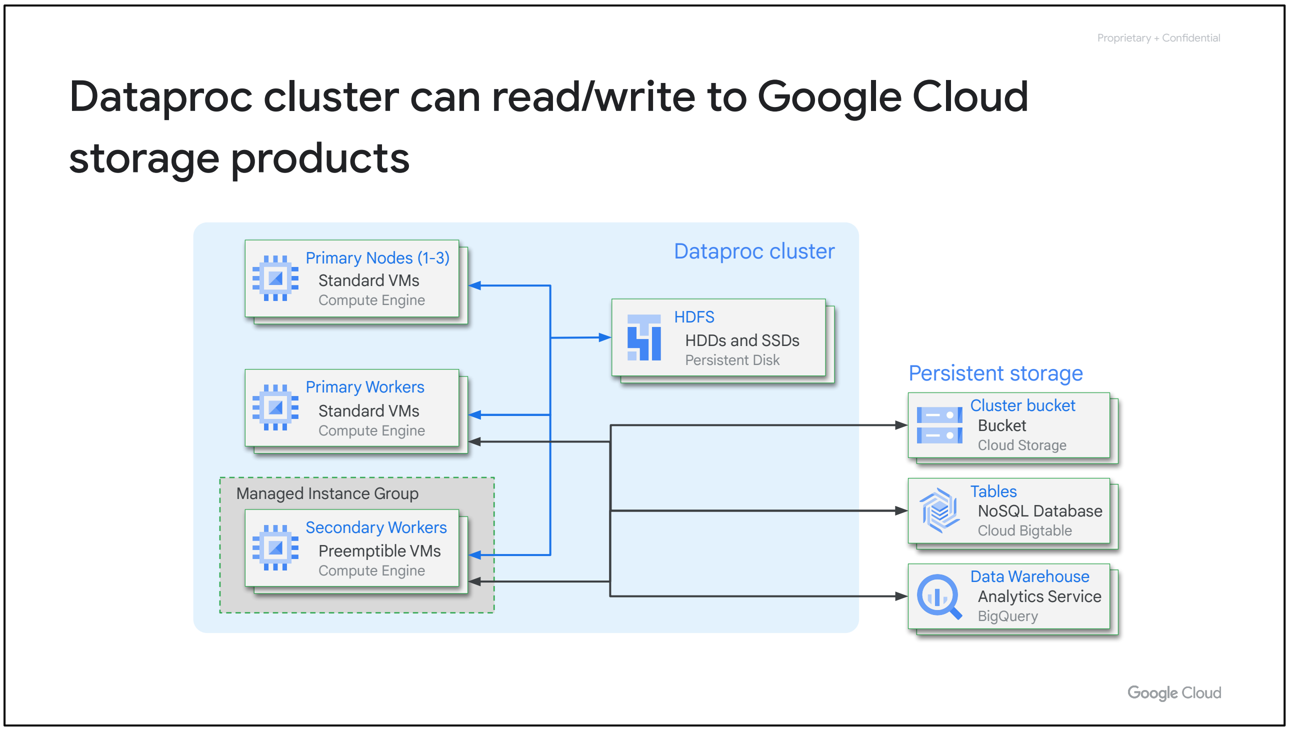
A Dataproc cluster can contain either preemptible secondary workers or non-preemptible secondary workers, but not both.

The standard setup architecture is much like you would expect on-premise. You have a cluster of virtual machines for processing and then persistent disks for storage via HDFS. You've also got your primary node VMs in a set of worker nodes.

Worker nodes can also be part of a managed instance group, which is just another way of ensuring that VMs within that group are all of the same template. The advantage is that you can spin up more VMs than you need to automatically resize your cluster based on the demands. It also only takes a few minutes to upgrade or downgrade your cluster.

Generally, you shouldn't think of a Dataproc cluster as long-lived. Instead you should spin them up when you need compute processing for a job and then simply turn them down. You can also persist them indefinitely if you want to.

What happens to HDFS storage on disk when you turn those clusters down? The storage would go away too, which is why it's a best practice to use storage that's off cluster by connecting to other Google Cloud products.

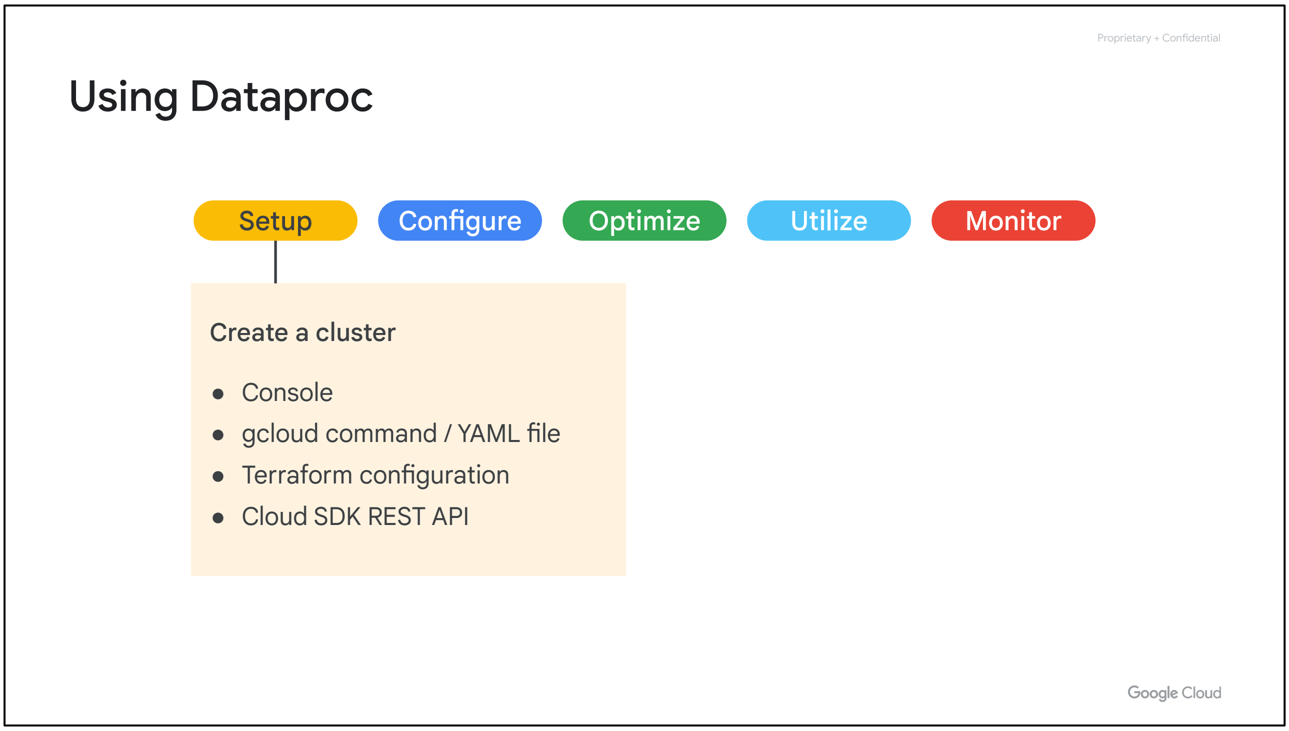


Dataproc cluster can read/write to Google Cloud storage products

Instead of using native HDFS on a cluster, you could simply use a cluster of bucket on Cloud Storage via the HDFS connector.

It's pretty easy to adapt existing Hadoop code to use Cloud Storage instead of HDFS. Change the prefix for this storage from hdfs// to gs//.

What about Hbase off-cluster? Consider writing in the Cloud Bigtable instead. What about large analytical workloads? Consider reading that data into BigQuery and doing those analytical work loads there.



Using Dataproc

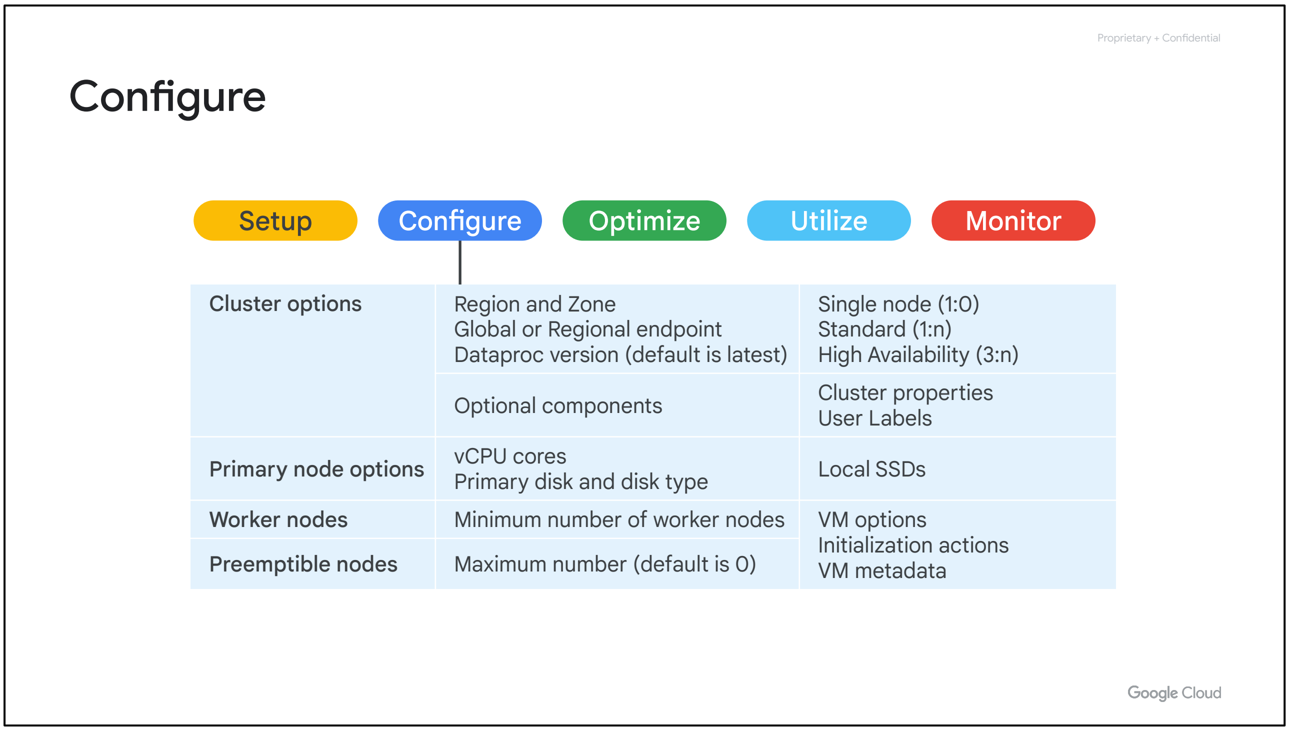
Setup
Configure
Optimize
Utilize
Monitor

Create a cluster

- Console
- gcloud command / YAML file
- Terraform configuration
- Cloud SDK REST API

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

Setup means creating a cluster. And you can do that through the Cloud Console, or 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 Terraform configuration, or use the REST API.



Configure

Setup
Configure
Optimize
Utilize
Monitor

Cluster options	Region and Zone Global or Regional endpoint Dataproc version (default is latest)	Single node (1:0) Standard (1:n) High Availability (3:n)
	Optional components	Cluster properties User Labels
Primary node options	vCPU cores Primary disk and disk type	Local SSDs
Worker nodes	Minimum number of worker nodes	VM options Initialization actions VM metadata
Preemptible nodes	Maximum number (default is 0)	

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 Primary Node, and High Availability has three Primary 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 Primary Node is where the HDFS Namenode runs, as well as the YARN node and job drivers. HDFS replication defaults to 2 in Dataproc.

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

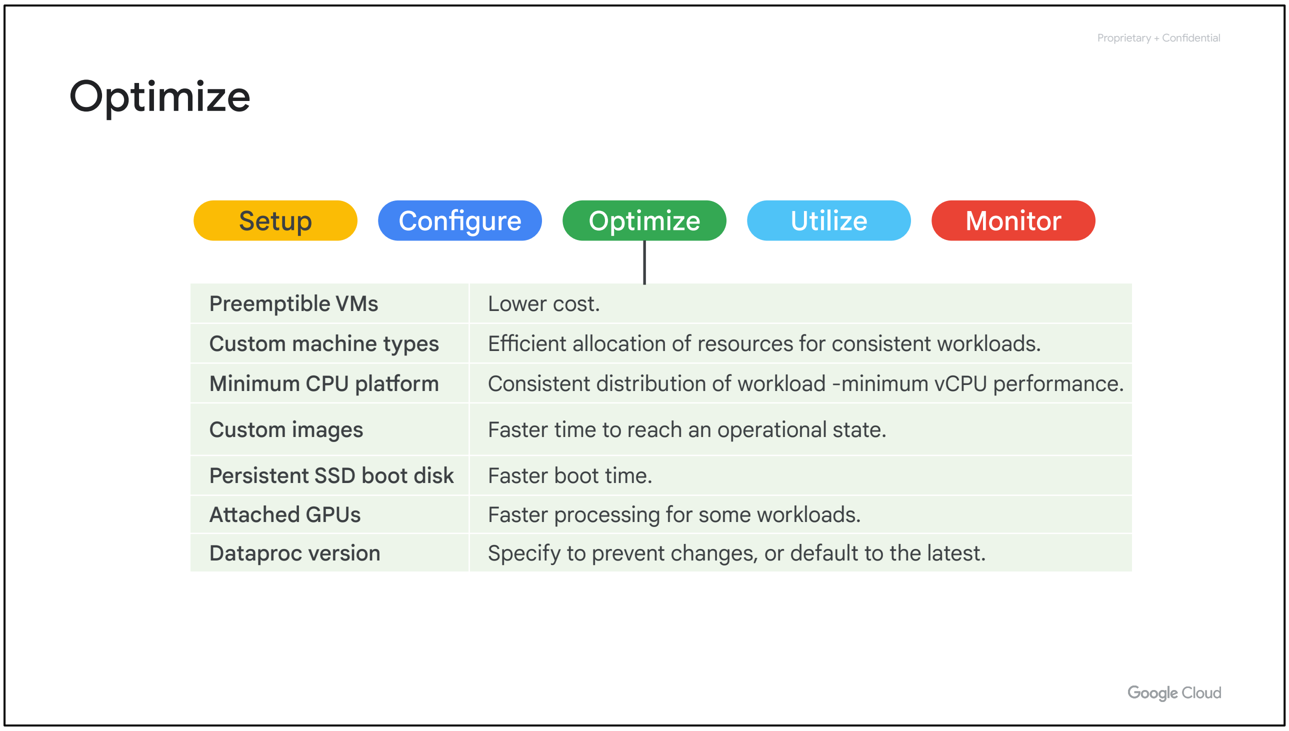
Cluster properties are run-time values that can be used by configuration files for more dynamic startup options. And user labels can be used to tag the cluster for your own solutions or reporting purposes.

The Primary 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.



Optimize

Setup
Configure
Optimize
Utilize
Monitor

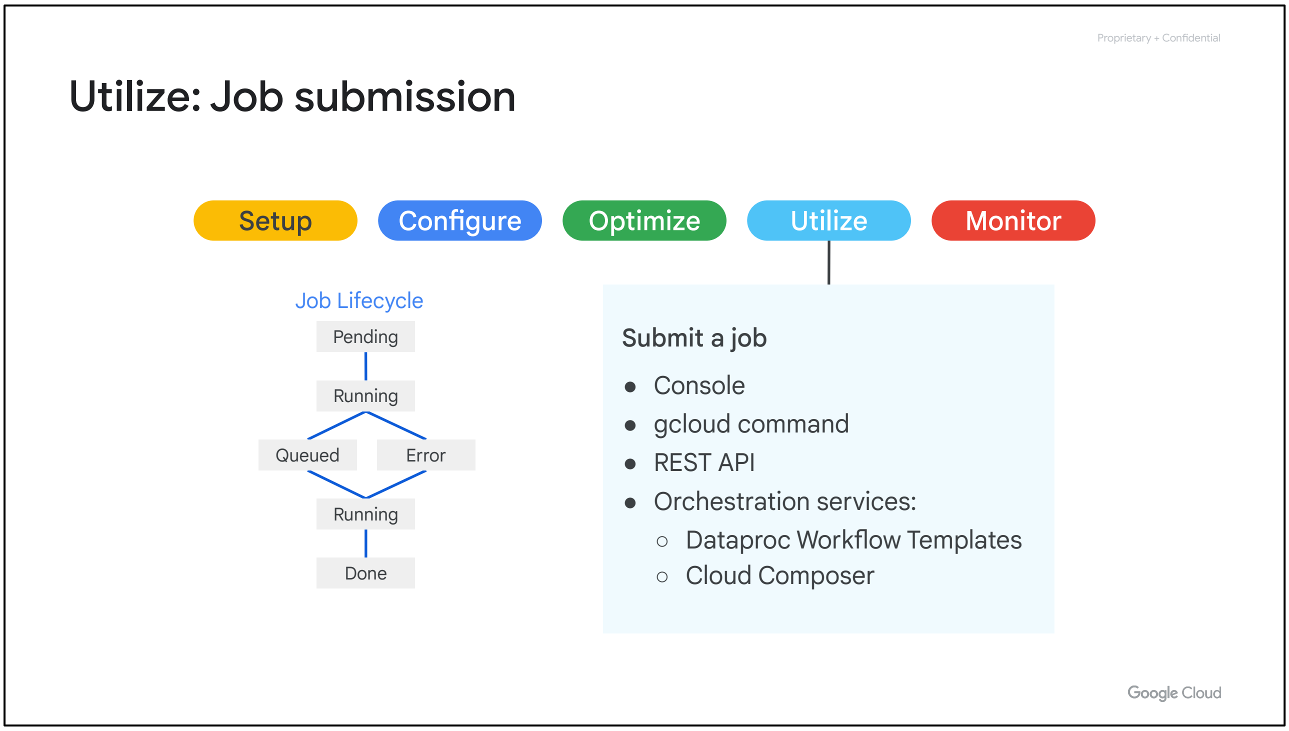
Preemptible VMs	Lower cost.
Custom machine types	Efficient allocation of resources for consistent workloads.
Minimum CPU platform	Consistent distribution of workload -minimum vCPU performance.
Custom images	Faster time to reach an operational state.
Persistent SSD boot disk	Faster boot time.
Attached GPUs	Faster processing for some workloads.
Dataproc version	Specify to prevent changes, or default to the latest.

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 at boot-time using and initialization script.

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



Utilize: Job submission

Setup
Configure
Optimize
Utilize
Monitor
Job Lifecycle
Pending
Running
Queued
Error
Running
Done

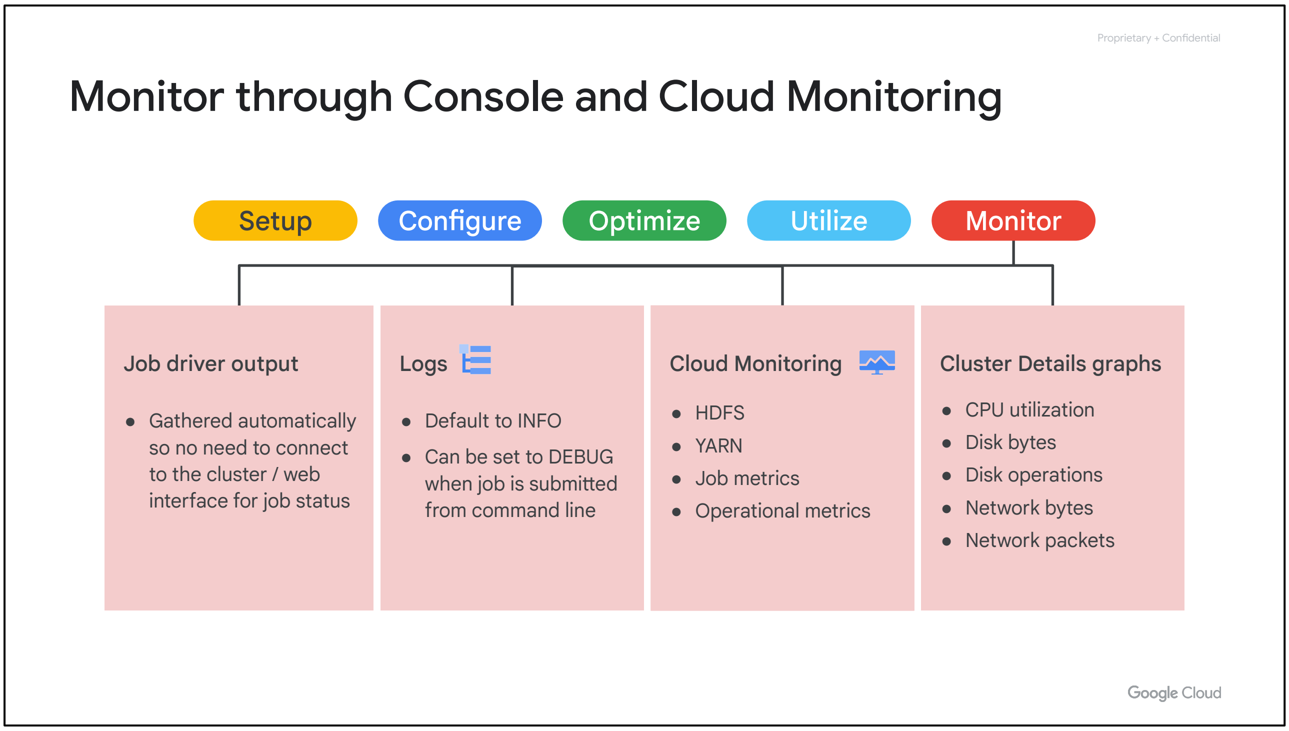
Submit a job

- Console
- gcloud command
- REST API
- Orchestration services:
  - Dataproc Workflow Templates
  - Cloud Composer

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

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

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.



Monitor through Console and Cloud Monitoring

Setup
Configure
Optimize
Utilize
Monitor

Job driver output

- Gathered automatically so no need to connect to the cluster / web interface for job status

Logs

- Default to INFO
- Can be set to DEBUG when job is submitted from command line

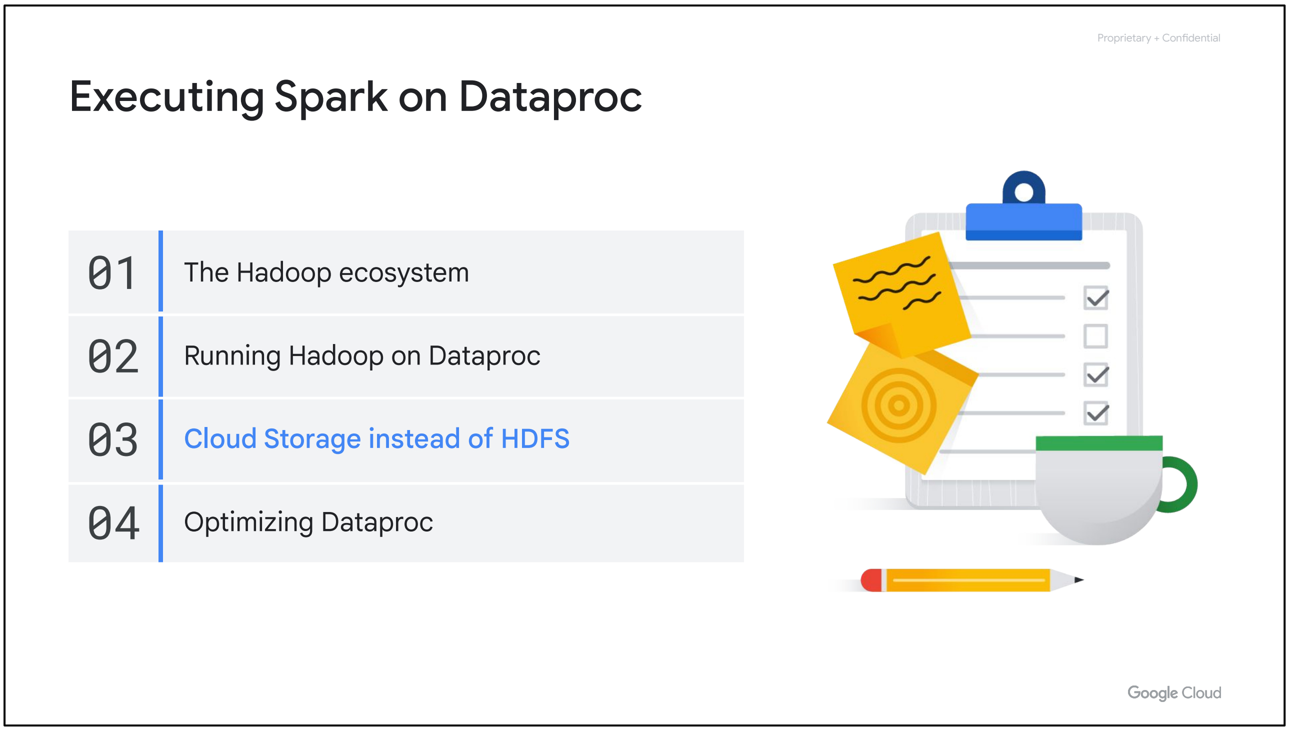
Cloud Monitoring

- HDFS
- YARN
- Job metrics
- Operational metrics

Cluster Details graphs

- CPU utilization
- Disk bytes
- Disk operations
- Network bytes
- Network packets

Lastly, after you submit your job you'll want to monitor it, you can do so using Cloud Monitoring. Or you can also build a custom dashboard with graphs and set up monitoring of alert policies to send emails for example, where you can notify if incidents happen. Any details from HDFS, YARN, metrics about a particular job or overall metrics for the cluster like CPU utilization, disk and network usage, can all be monitored and alerted on with Cloud Monitoring.



Executing Spark on Dataproc

Let's discuss more about using Google Cloud Storage, instead of the native Hadoop file system, or HDFS.

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

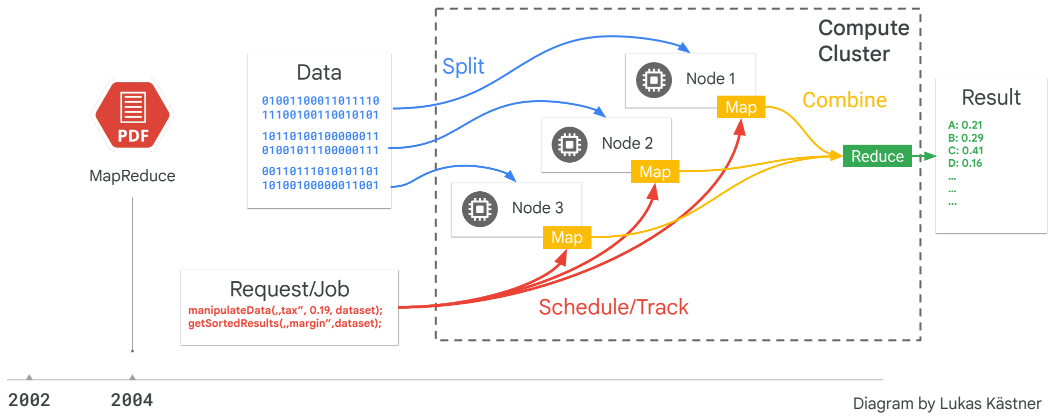
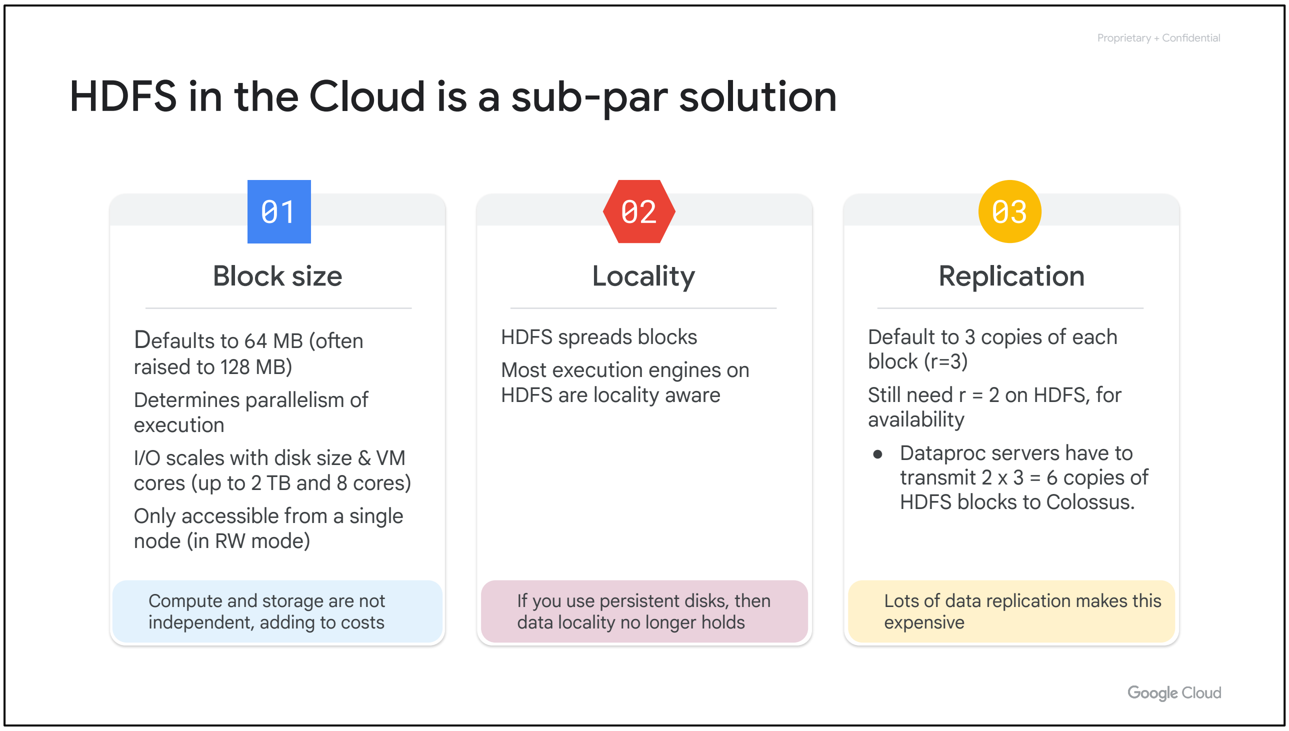


Diagram by Lukas Kästner

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.



HDFS in the Cloud is a sub-par solution

01
Block size

Defaults to 64 MB (often raised to 128 MB)

Determines parallelism of execution

I/O scales with disk size & VM cores (up to 2 TB and 8 cores)

Only accessible from a single node (in RW mode)

Compute and storage are not independent, adding to costs

02
Locality

HDFS spreads blocks

Most execution engines on HDFS are locality aware

If you use persistent disks, then data locality no longer holds

03
Replication

Default to 3 copies of each block (r=3)

Still need r = 2 on HDFS, for availability

- Dataproc servers have to transmit 2 x 3 = 6 copies of HDFS blocks to Colossus.

Lots of data replication makes this expensive

Google Cloud

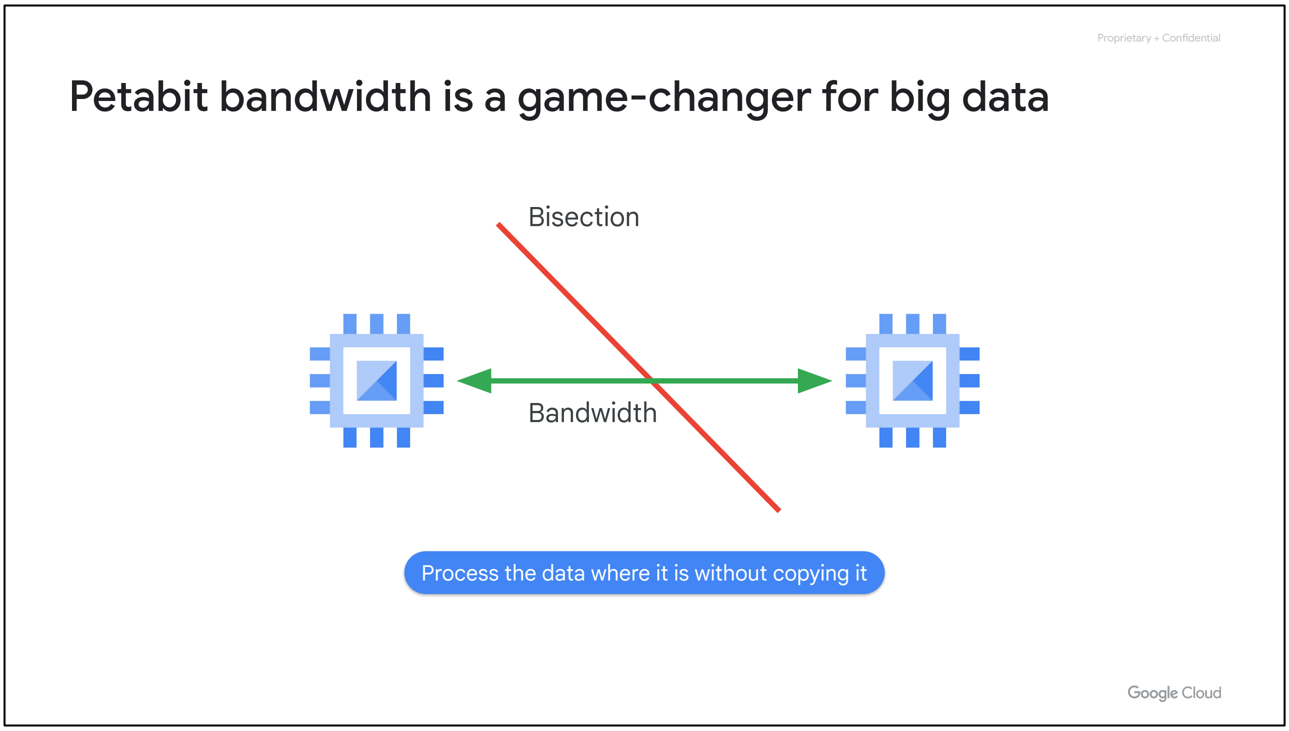
Your on-premise Hadoop clusters need local storage on its disk, since the same server runs, computes, and stores jobs. That's one of the first areas for optimization. You can run HDFS in the Cloud just by lifting and shifting your Hadoop workloads to Dataproc. This is often the first step to the Cloud, 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.

Block size
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 in 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.

Locality
For data locality, there are similar concerns about storing data on individual persistent disks.

Replication
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.


Proprietary + Confidential

Petabit bandwidth is a game-changer for big data

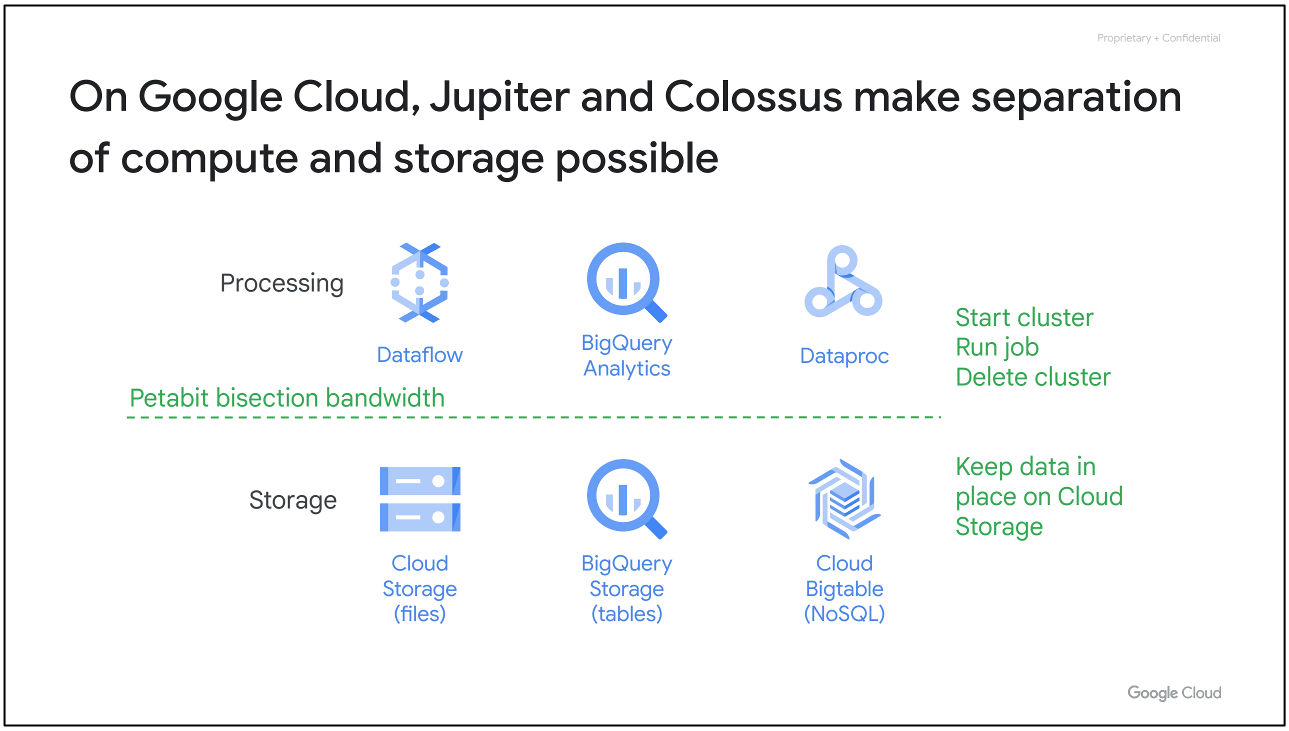
Bisection
Bandwidth
Process the data where it is without copying it
Google Cloud

Google's network enables new solutions for Big Data. The Jupiter networking fabric within a Google data center delivers over 1 petabit per second of bandwidth.

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.)

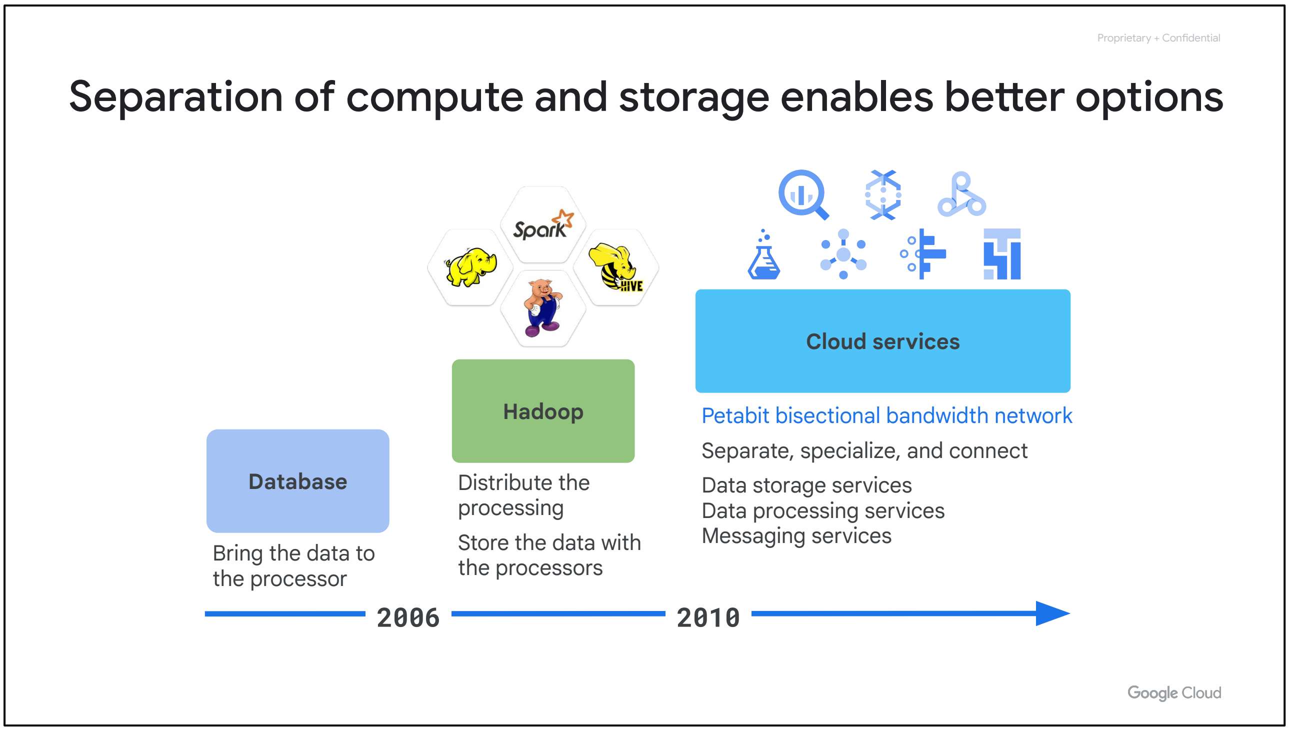
If you draw a line somewhere in a network, bisectional bandwidth is the rate of communication at which servers on one side of the line can communicate with servers on the other side. With enough bisectional bandwidth any server can communicate with any other server at full network speeds.

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.



On Google Cloud, Jupiter and Colossus make separation of compute and storage possible

Inside of a Google data center, the internal name for the massively distributed storage layer is called Colossus, and the network inside the data center 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 controlled by Colossus behind the scenes.



Proprietary + Confidential

Separation of compute and storage enables better options

Database: Bring the data to the processor

Hadoop: Distribute the processing. Store the data with the processors

Cloud services: Petabit bisectional bandwidth network. Separate, specialize, and connect. Data storage services, Data processing services, Messaging services

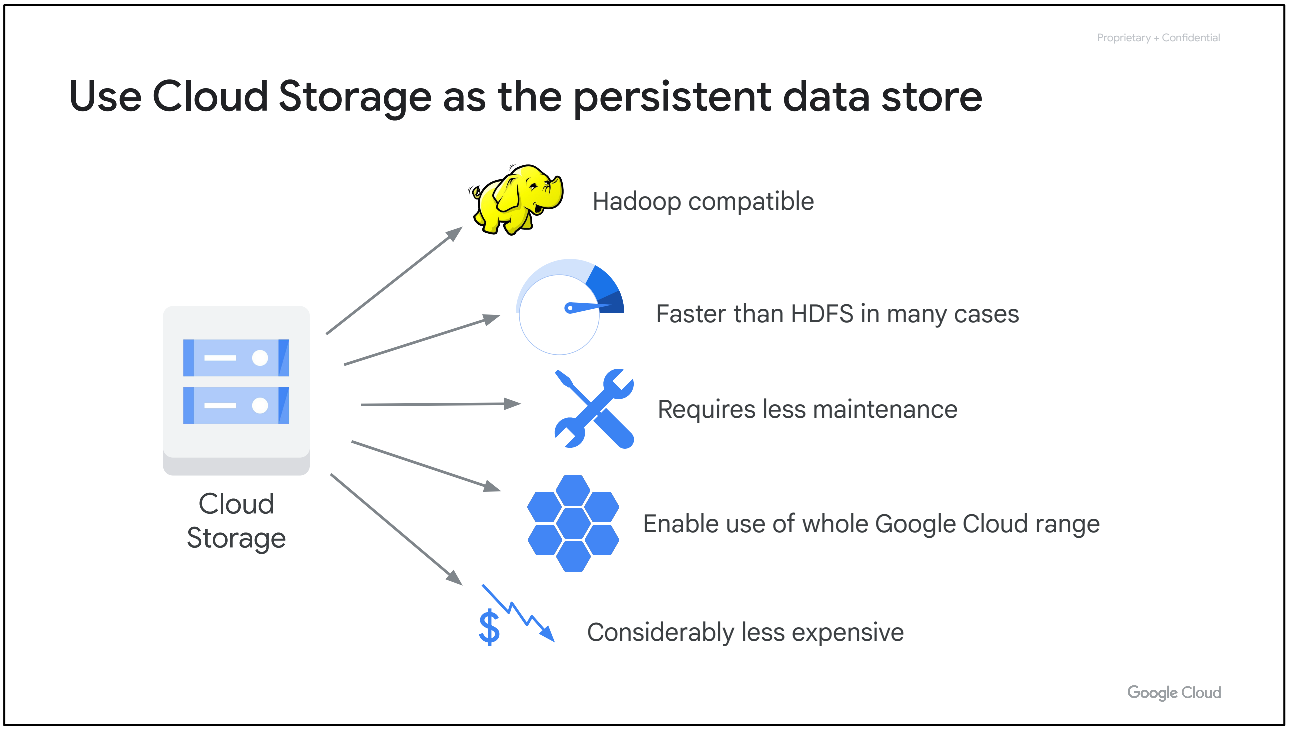
Google Cloud

A historical continuum of data management is a follows:

Before 2006 big data meant big databases. Database design came from a time when storage was relatively cheap and processing was expensive.

Around 2006, distributed processing of big data became practical with Hadoop.

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



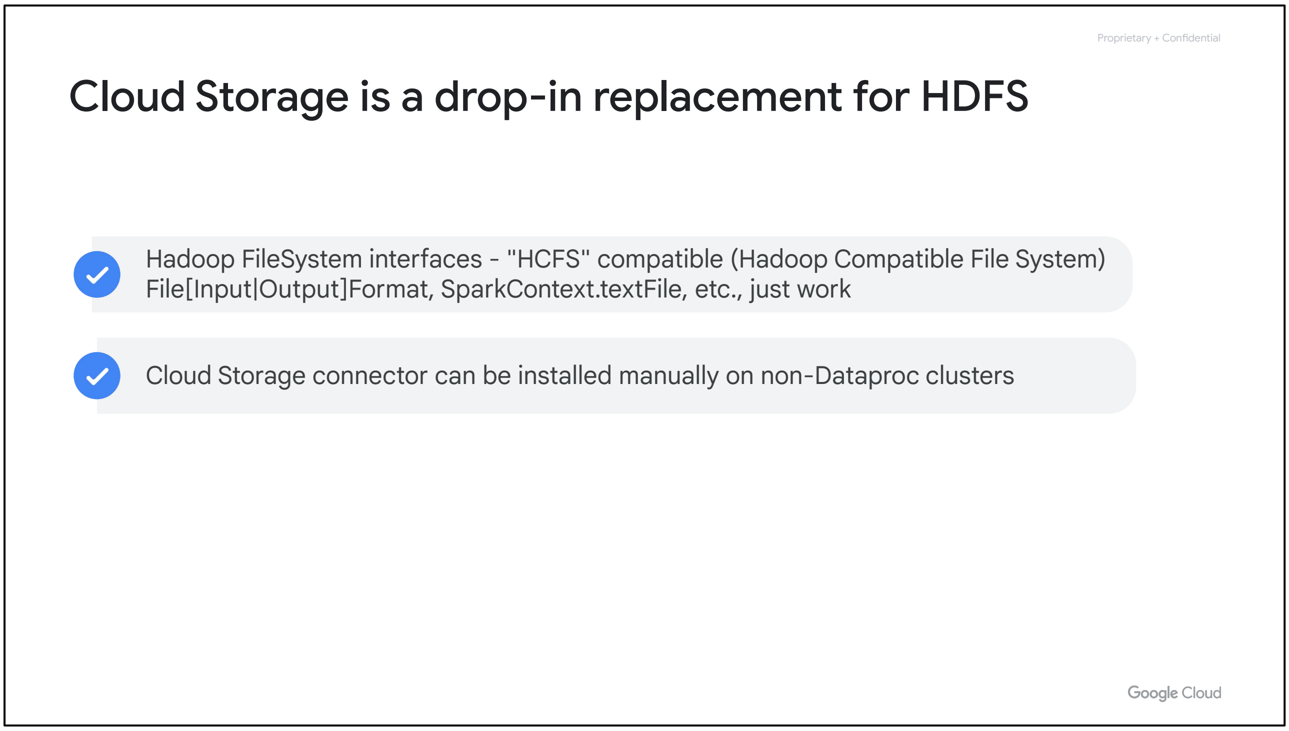
Proprietary + Confidential

Use Cloud Storage as the persistent data store

Cloud Storage
Hadoop compatible
Faster than HDFS in many cases
Requires less maintenance
Enable use of whole Google Cloud range
Considerably less expensive

Google Cloud

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 Google Cloud projects.


Proprietary + Confidential

Cloud Storage is a drop-in replacement for HDFS

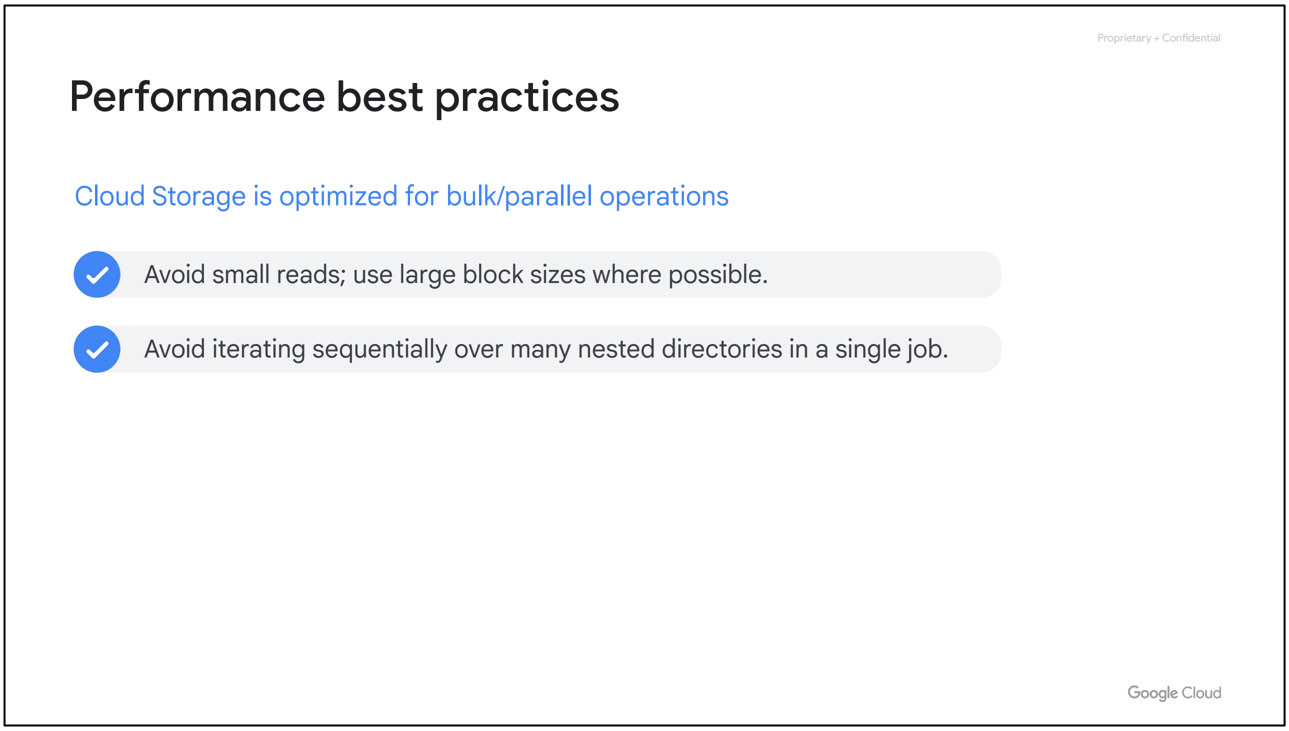
- ✓ Hadoop FileSystem interfaces - "HCFS" compatible (Hadoop Compatible File System) File[Input|Output]Format, SparkContext.textFile, etc., just work
- ✓ Cloud Storage connector can be installed manually on non-Dataproc clusters

Google Cloud

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.

With HDFS, you must overprovision for current data and for data you might have, and you must use persistent disks throughout.

With Cloud Storage however, you pay for exactly what you need, when you use it.



Proprietary + Confidential

Performance best practices

Cloud Storage is optimized for bulk/parallel operations

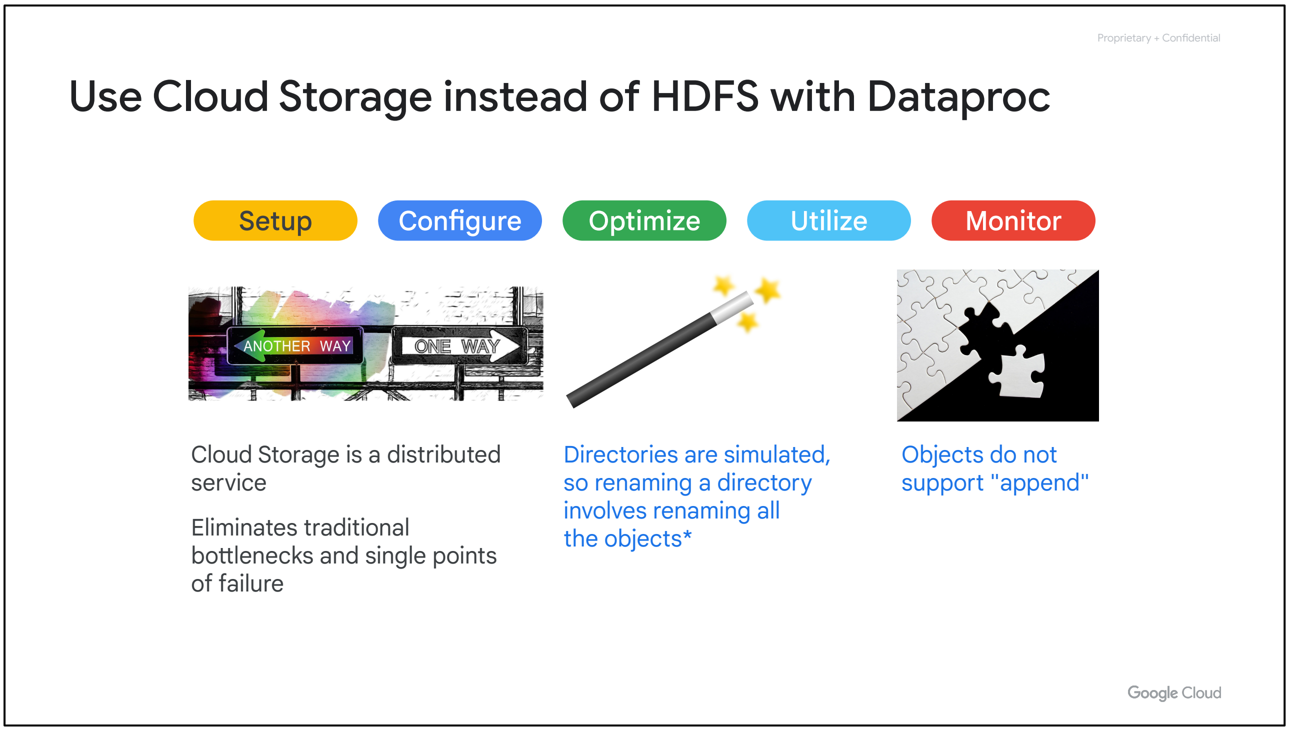
- ✓ Avoid small reads; use large block sizes where possible.
- ✓ Avoid iterating sequentially over many nested directories in a single job.

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 Dataproc

Setup Configure Optimize Utilize Monitor

Cloud Storage is a distributed service

Eliminates traditional bottlenecks and single points of failure

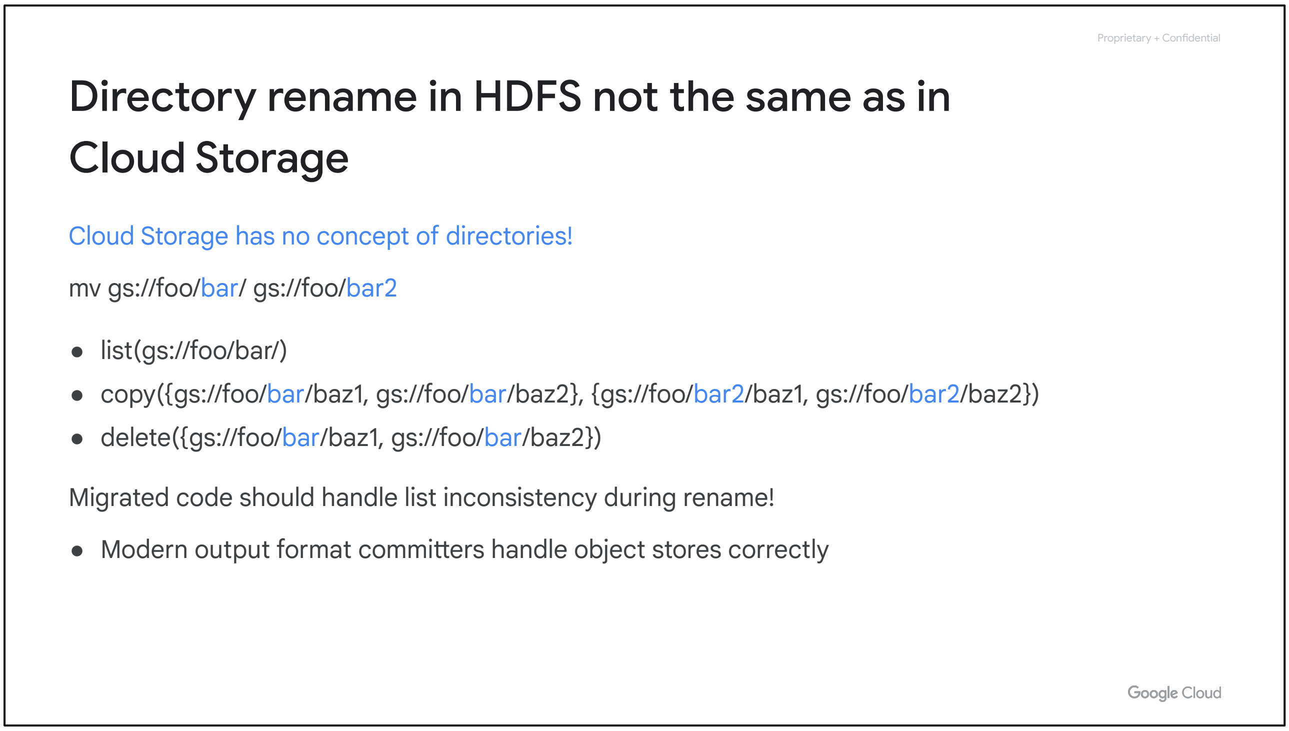
Directories are simulated, so renaming a directory involves renaming all the objects*

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

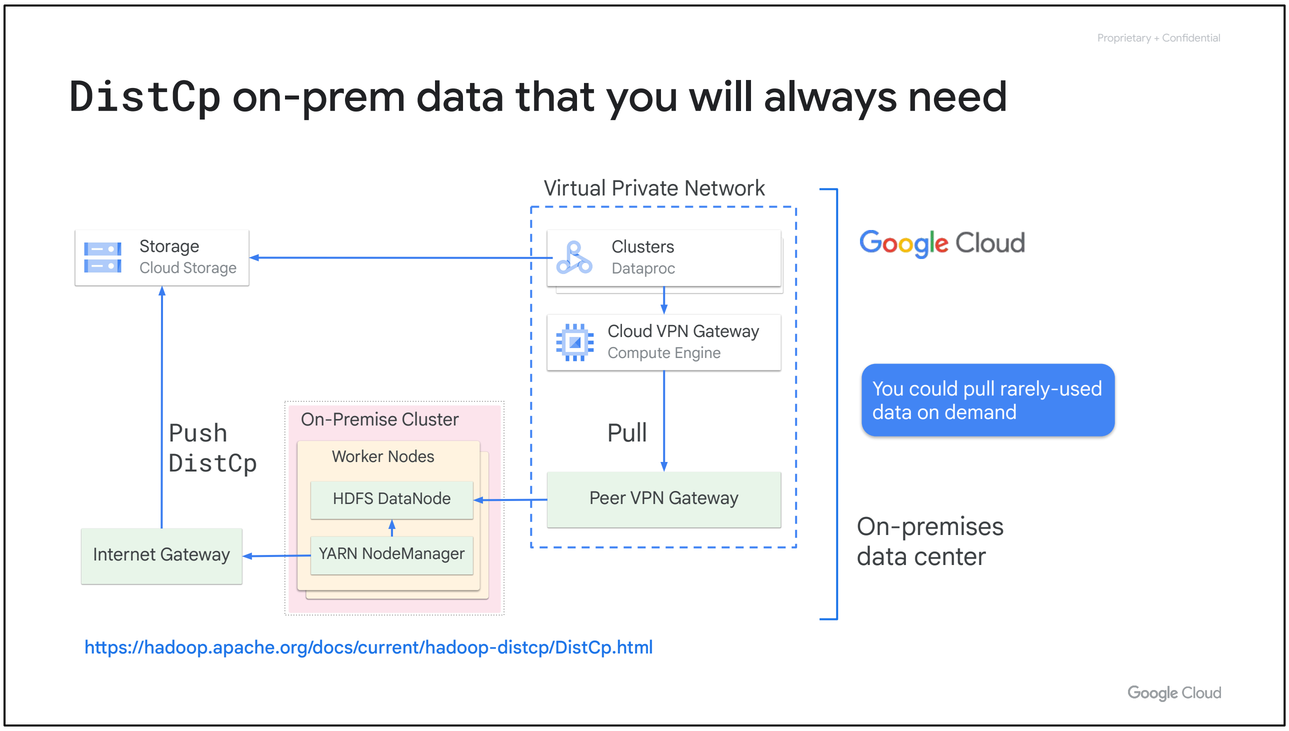
- list(gs://foo/bar/)
- copy({gs://foo/bar/baz1, gs://foo/bar/baz2}, {gs://foo/bar2/baz1, gs://foo/bar2/baz2})
- delete({gs://foo/bar/baz1, gs://foo/bar/baz2})

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.

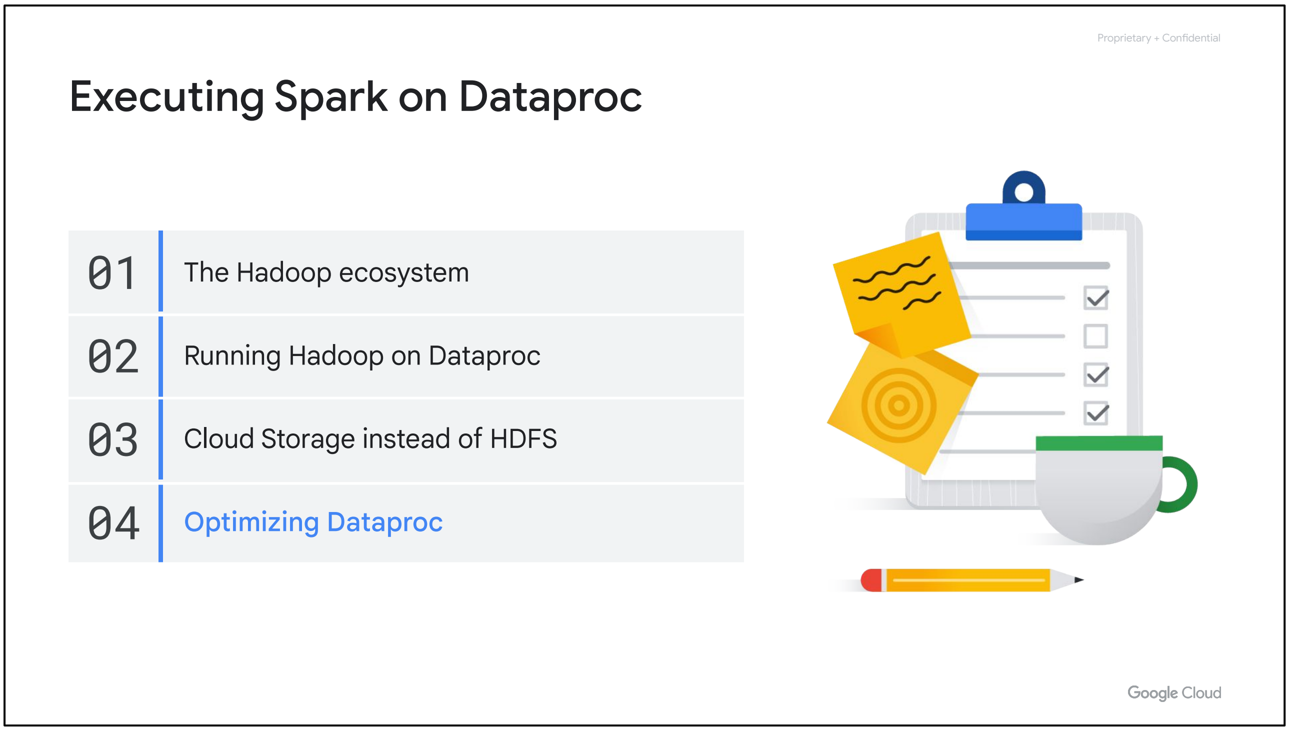


DistCp on-prem data that you will always need

https://hadoop.apache.org/docs/current/hadoop-distcp/DistCp.html

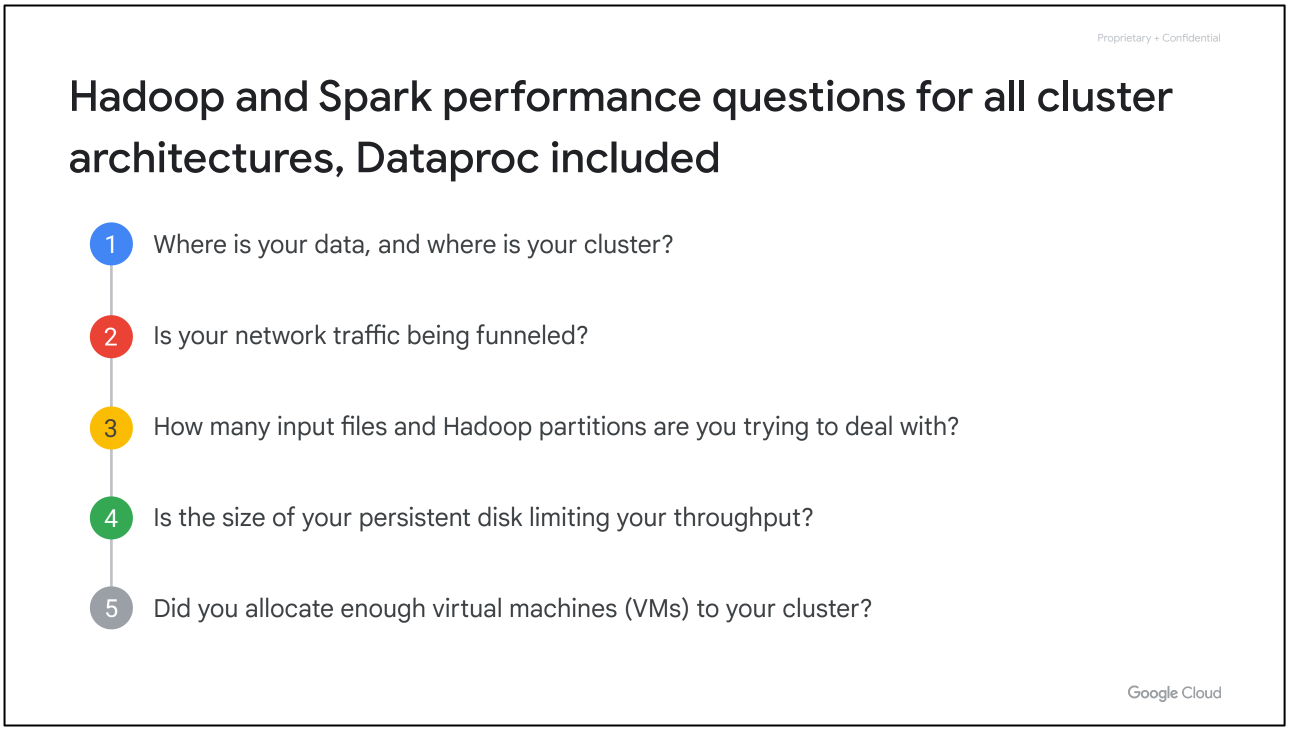
DistCp is a key tool for moving data. In genera,l you want to use a push-based model for any data that you know you will need while pull-based may be a useful model if there is a lot of data that you might not ever need to migrate.

https://hadoop.apache.org/docs/current/hadoop-distcp/DistCp.html



Executing Spark on Dataproc

Next, let's look at optimizing Dataproc.


Proprietary + Confidential

Hadoop and Spark performance questions for all cluster architectures, Dataproc included

1. Where is your data, and where is your cluster?
2. Is your network traffic being funneled?
3. How many input files and Hadoop partitions are you trying to deal with?
4. Is the size of your persistent disk limiting your throughput?
5. Did you allocate enough virtual machines (VMs) to your cluster?

Google Cloud

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 Dataproc, you can omit the zone and have the 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

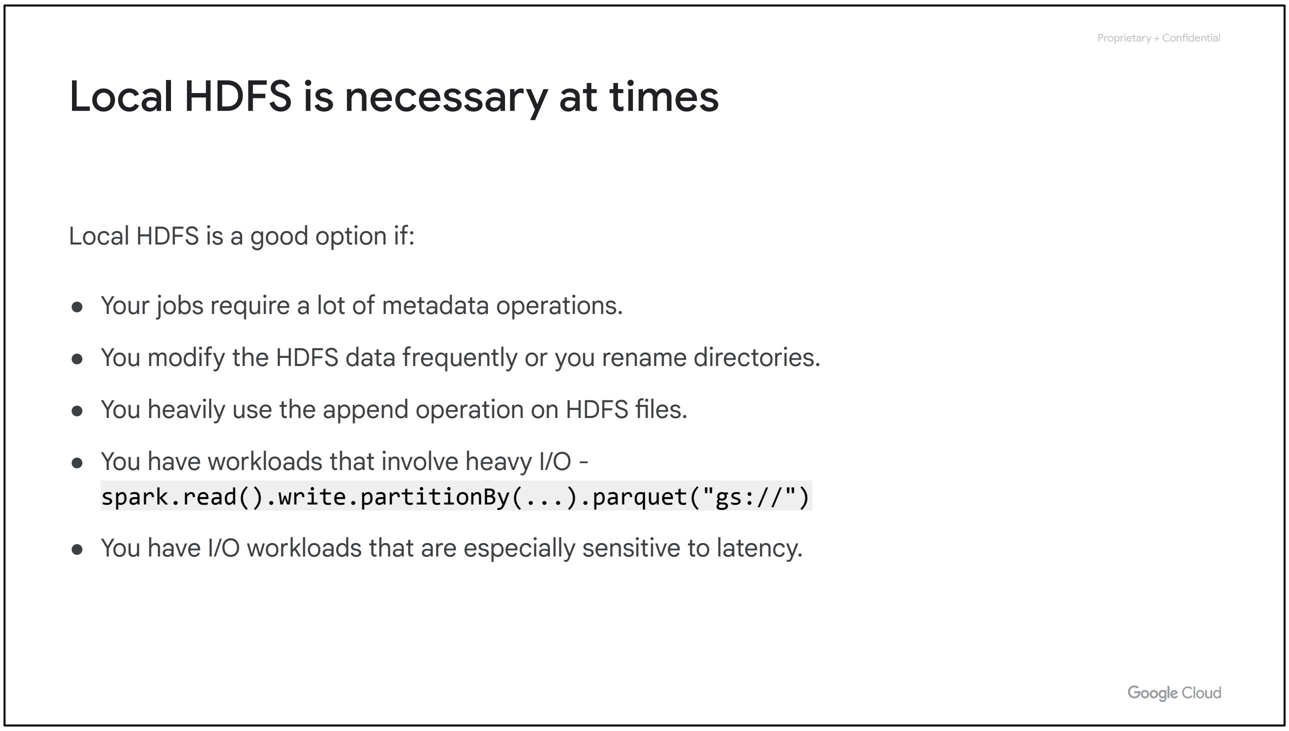
Is your network traffic being funneled?
Be sure that you do not have any network rules or routes that funnel Cloud Storage traffic through a small number of VPN gateways before it reaches your cluster. There are large network pipes between Cloud Storage and Compute Engine. You don't want to throttle your bandwidth by sending traffic into a bottleneck in your Google Cloud 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 approximately 50,000 Hadoop partitions), you should consider adjusting the setting fs.gs.block.size to a larger value accordingly.

Is the size of your persistent disk limiting your throughput?
Oftentimes, 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 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.



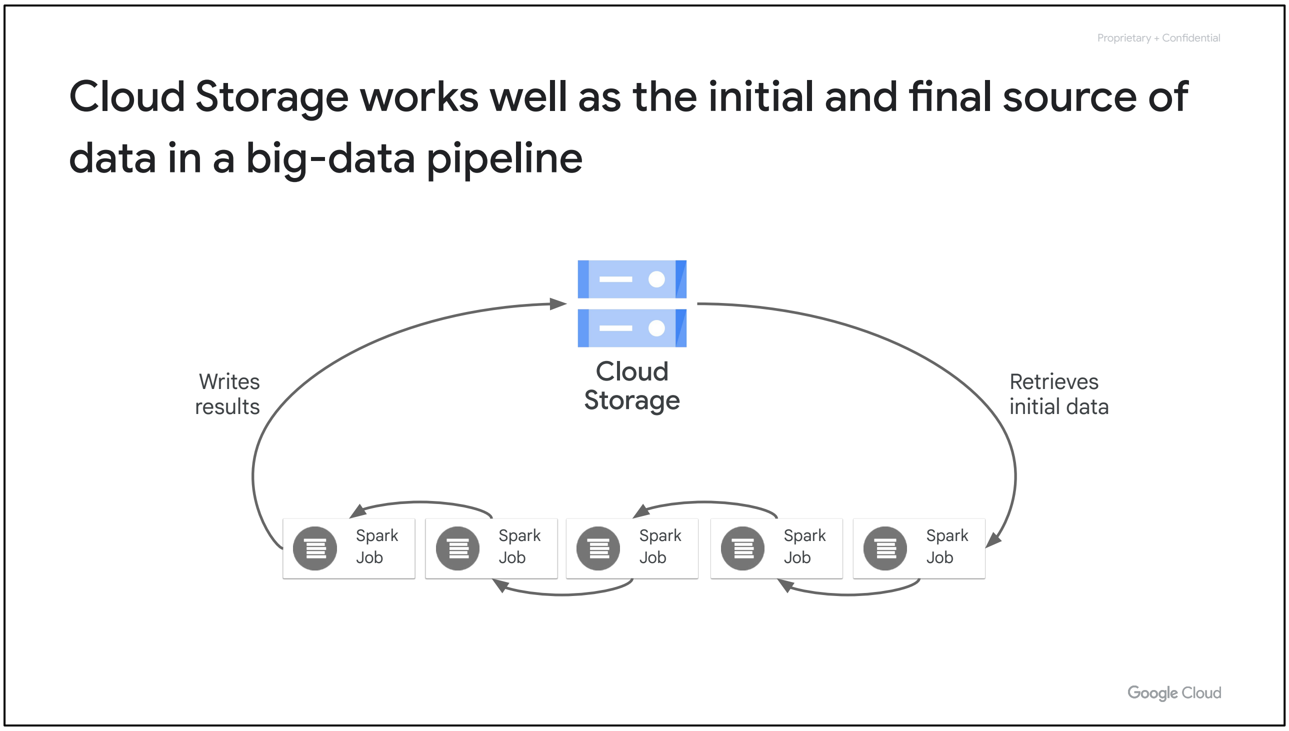
Local HDFS is necessary at times

Local HDFS is a good option if:

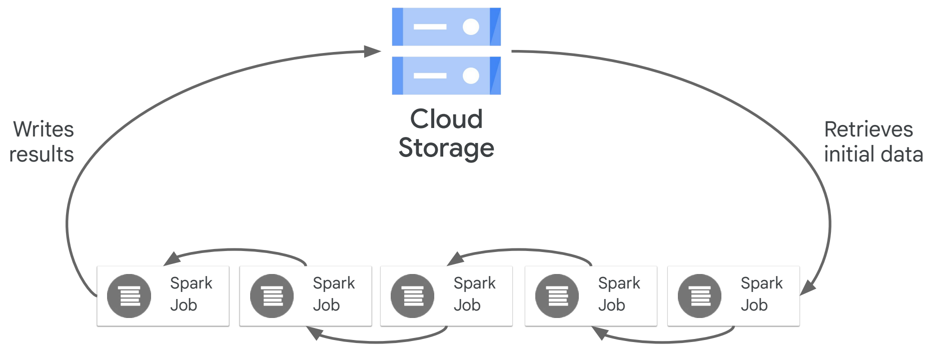
- Your jobs require a lot of metadata operations.
- You modify the HDFS data frequently or you rename directories.
- You heavily use the append operation on HDFS files.
- You have workloads that involve heavy I/O - `spark.read().write.partitionBy(...).parquet("gs://")`
- You have I/O workloads that are 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 in this example.
- You have I/O workloads that are especially sensitive to latency. For example, you require single-digit millisecond latency per storage operation.



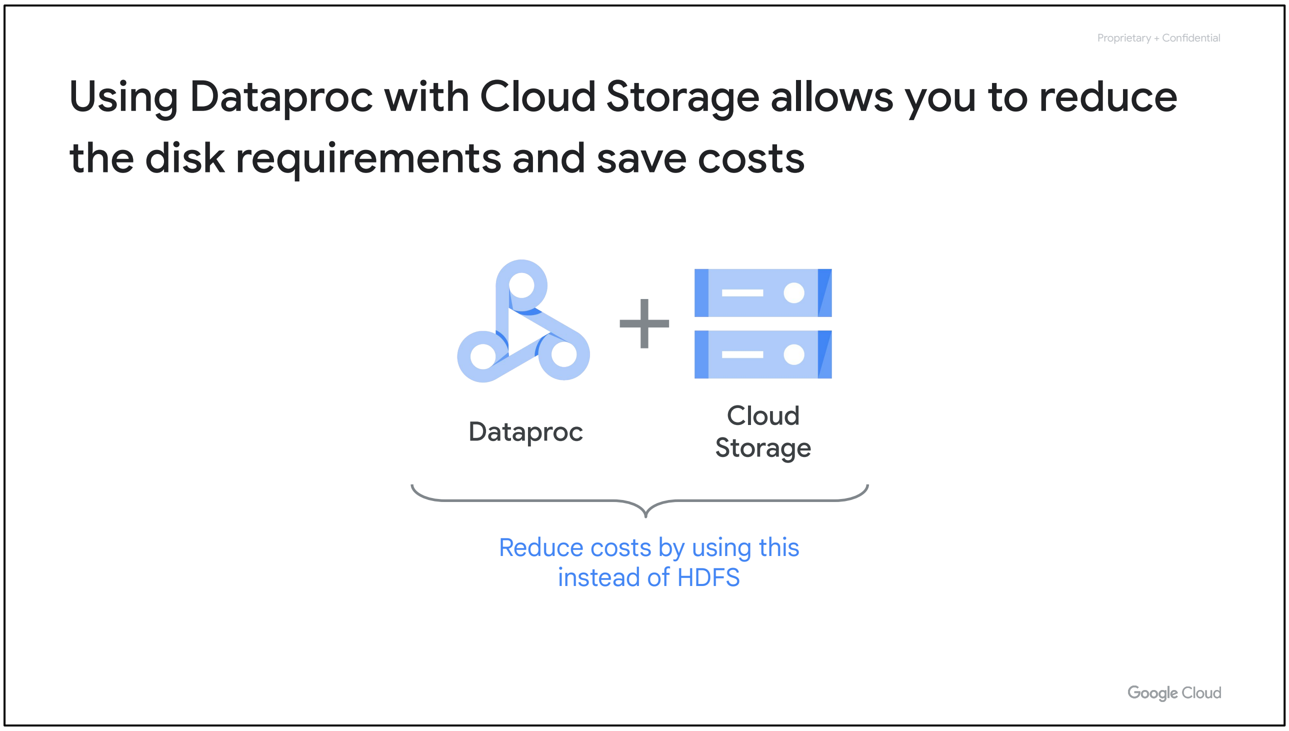
Cloud Storage works well as the initial and final source of data in a big-data pipeline



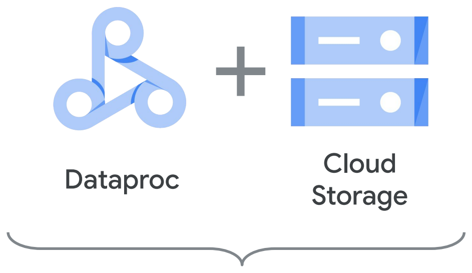
In general, we recommend using Cloud Storage as the initial and final source of data in a big-data pipeline.

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.



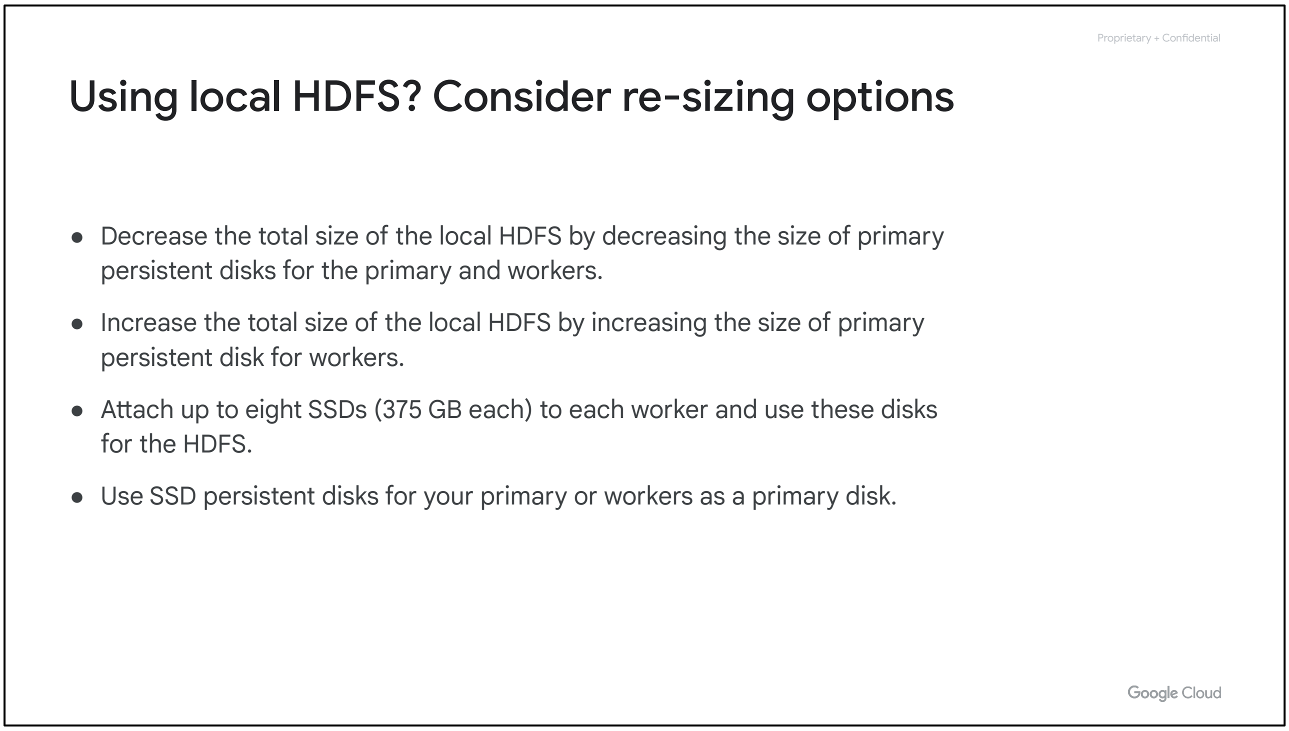
Using Dataproc with Cloud Storage allows you to reduce the disk requirements and save costs



Reduce costs by using this instead of HDFS

Using 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 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.



Proprietary + Confidential

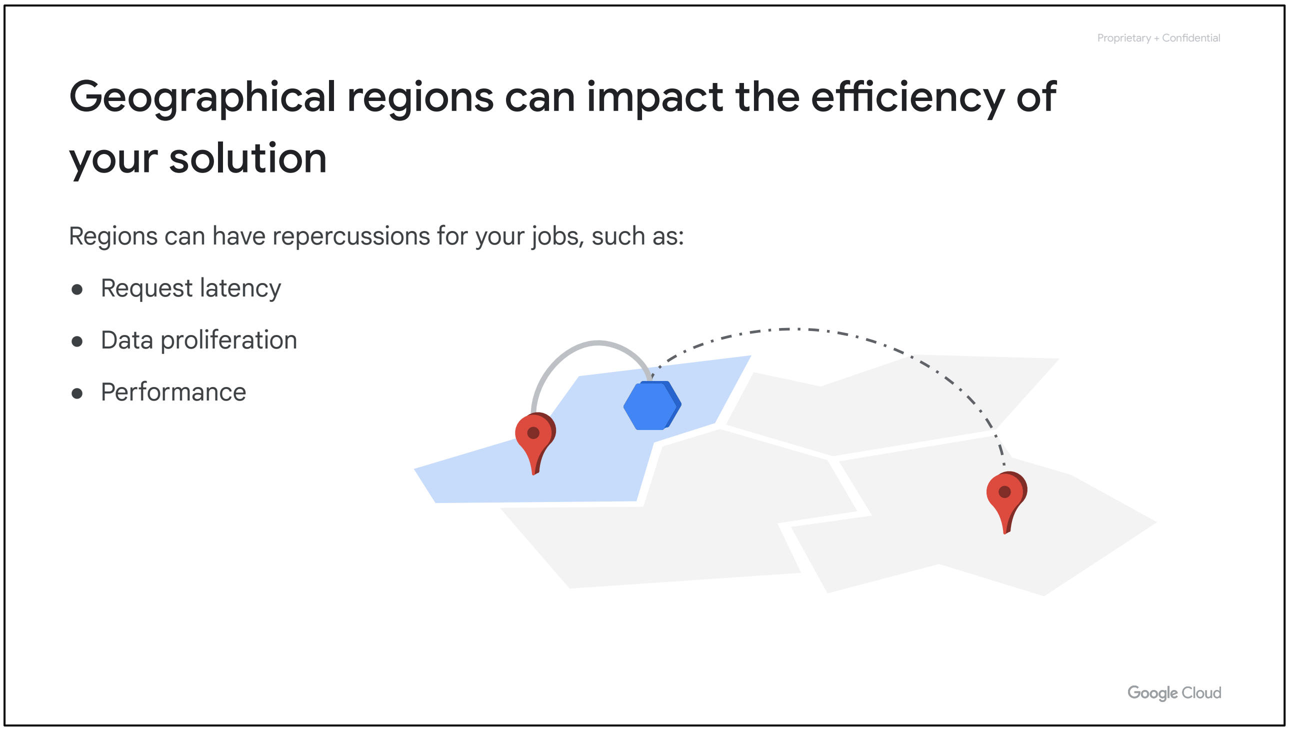
Using local HDFS? Consider re-sizing options

- Decrease the total size of the local HDFS by decreasing the size of primary persistent disks for the primary 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 primary or workers as a primary disk.

Google Cloud

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

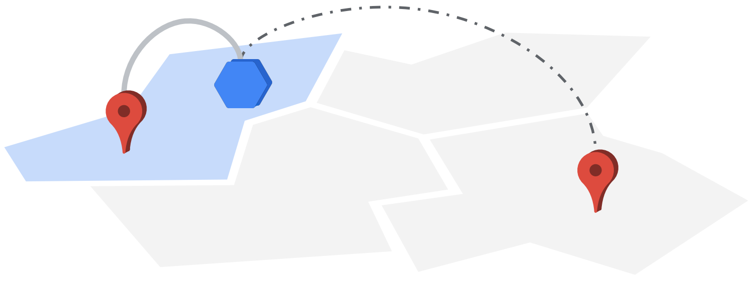
- Decrease the total size of the local HDFS by decreasing the size of primary persistent disks for the primary and workers. The primary persistent disk also contains the boot volume and system libraries, so allocate at least 100 GB.
- Increase the total size of the local HDFS by increasing the size of 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 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.
- And use SSD persistent disks for your primary or workers as a primary disk.



Geographical regions can impact the efficiency of your solution

Regions can have repercussions for your jobs, such as:

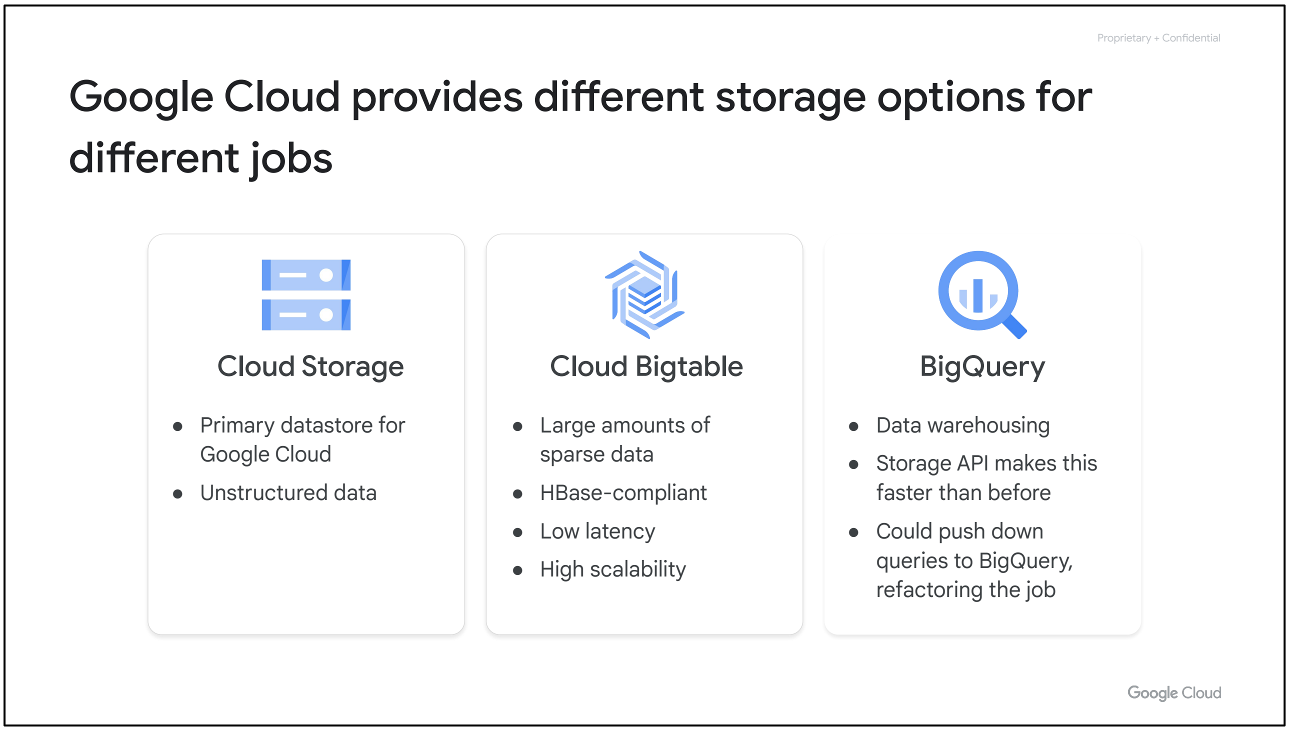
- Request latency
- Data proliferation
- Performance



You should understand the repercussions of geography and regions before you configure your data and jobs. Many Google Cloud 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 Google Cloud services might copy all of the required data from one zone to another before processing. This can have a severe impact on performance.



Google Cloud provides different storage options for different jobs

Cloud Storage

- Primary datastore for Google Cloud
- Unstructured data

Cloud Bigtable

- Large amounts of sparse data
- HBase-compliant
- Low latency
- High scalability

BigQuery

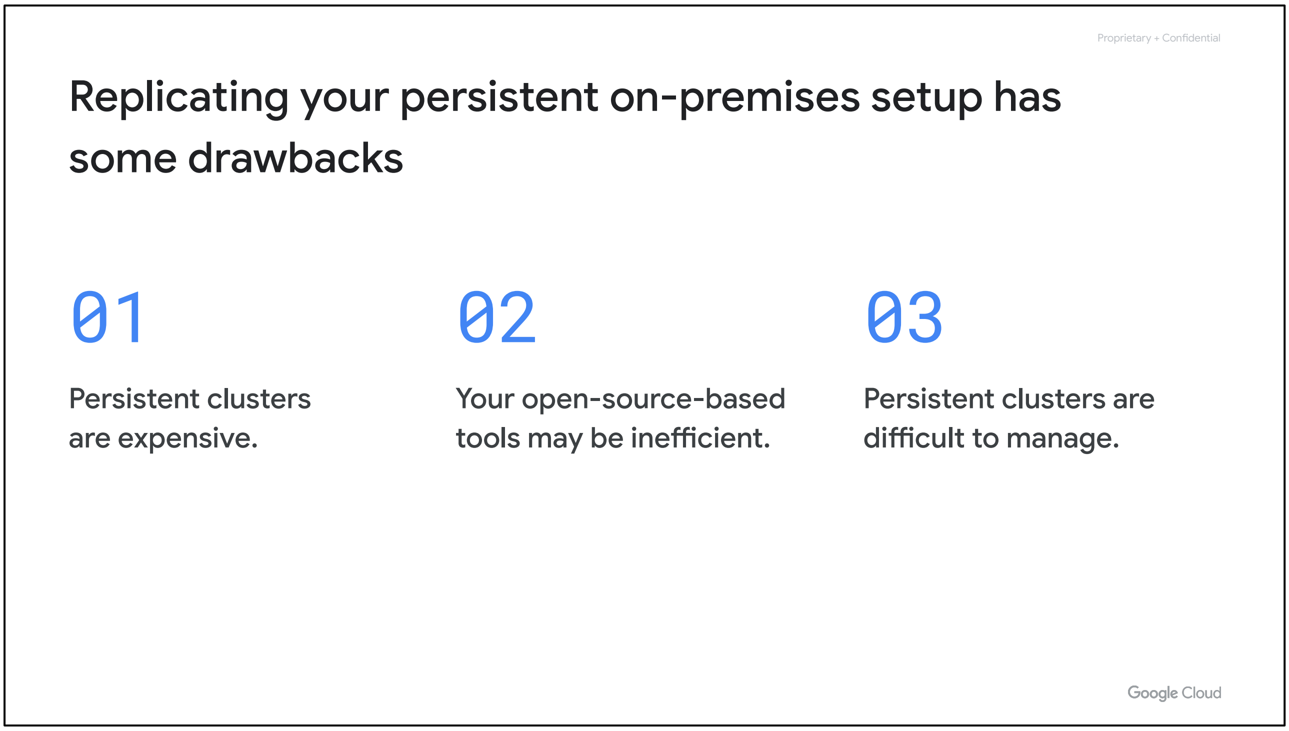
- Data warehousing
- Storage API makes this faster than before
- Could push down queries to BigQuery, refactoring the job

Google Cloud

Cloud Storage is the primary way to store unstructured data in Google Cloud, 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.



Replicating your persistent on-premises setup has some drawbacks

01

Persistent clusters are expensive.

02

Your open-source-based tools may be inefficient.

03

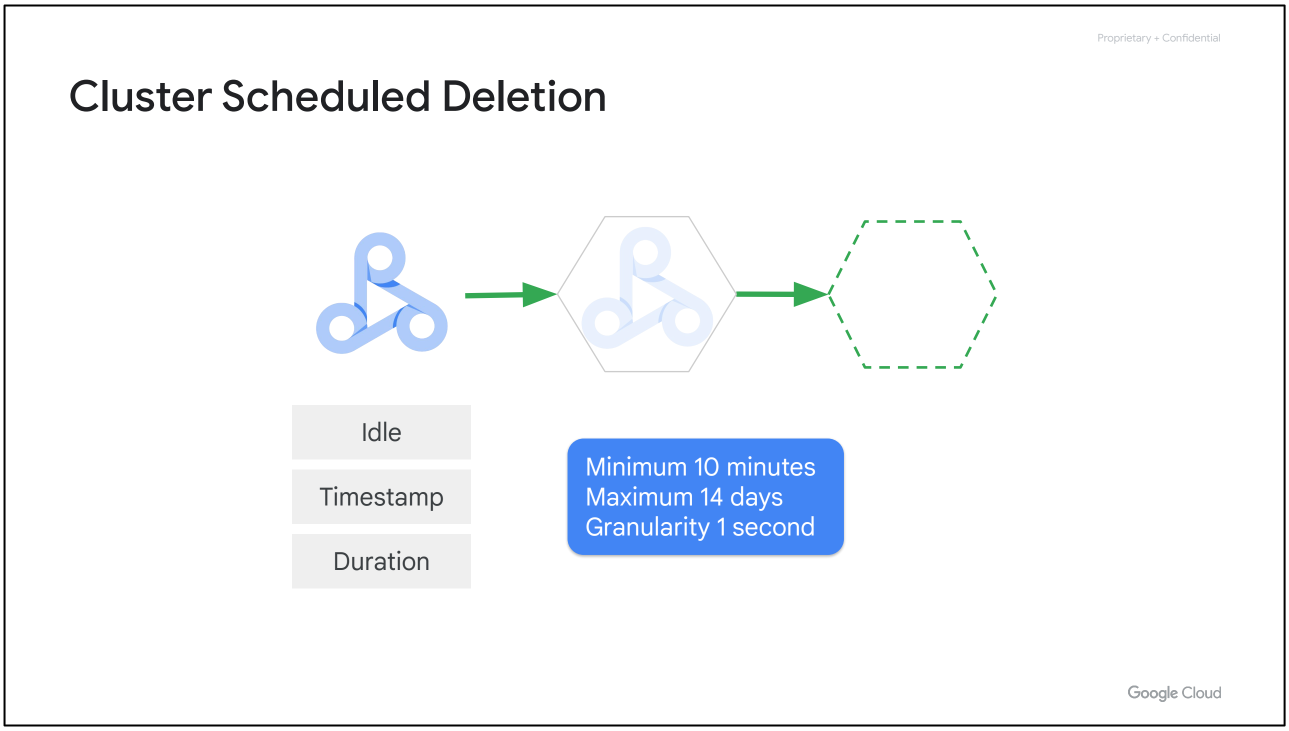
Persistent clusters are difficult to manage.

Because Dataproc runs Hadoop on Google Cloud, using a persistent 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 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 Google Cloud products.
- Augmenting or replacing some of your open-source-based tools with other related Google Cloud services can be more efficient or economical for particular use cases.
- Using a single, persistent 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 Google Cloud 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.



Proprietary + Confidential

Cluster Scheduled Deletion

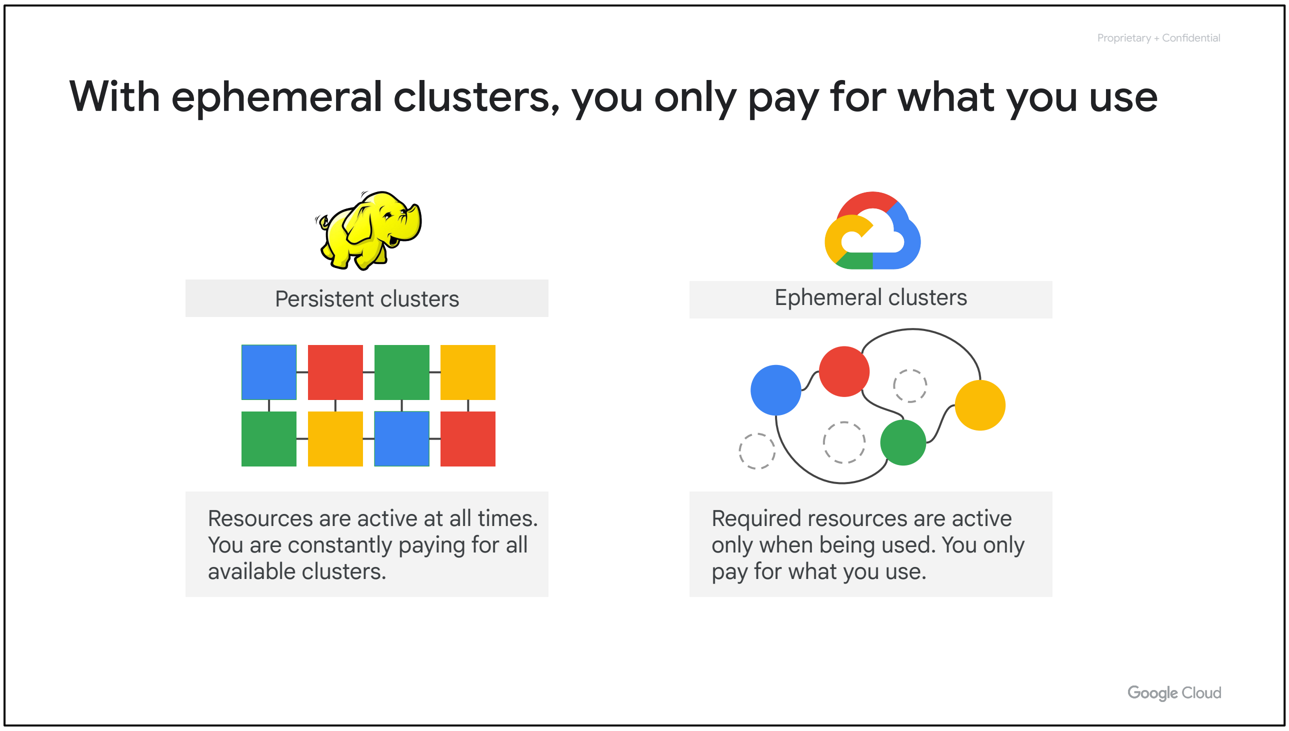
Idle
Timestamp
Duration
Minimum 10 minutes
Maximum 14 days
Granularity 1 second

Google Cloud

If you have efficient utilization, don't pay for resources that you don't use - employ scheduled deletion.

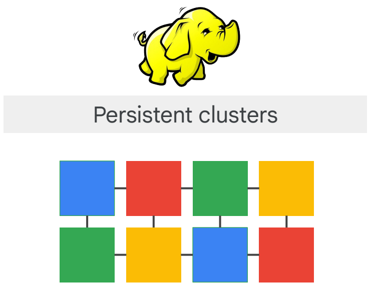
A fixed amount of time after the cluster enters an idle state, you can automatically set a timer. You can give it a timestamp, and the count starts immediately once the expiration has been set.

You can set a duration, the time in seconds to wait before automatically turning down the cluster. You can range from ten minutes as a minimum, to 14 days as a maximum at a granularity of one second.

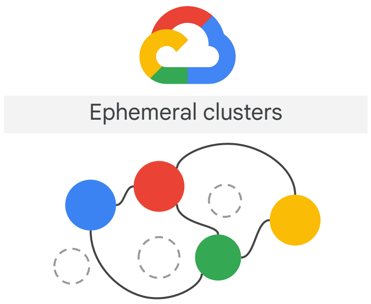


Proprietary + Confidential

With ephemeral clusters, you only pay for what you use



Resources are active at all times. You are constantly paying for all available clusters.



Required resources are active only when being used. You only pay for what you use.

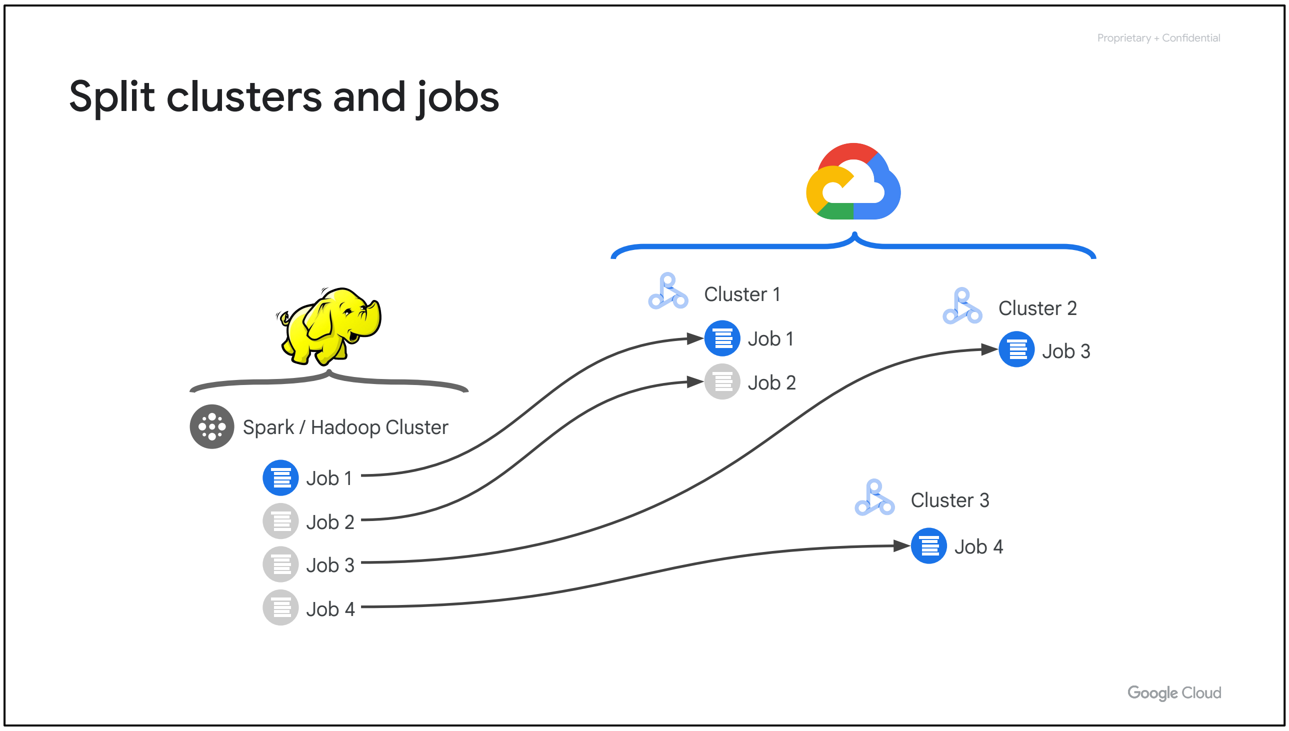
Google Cloud

The biggest shift in your approach between running an on-premises Hadoop workflow and running the same workflow on Google Cloud 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 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 Cloud Storage data persistence and fast boot of Dataproc, however, a persistent cluster is a waste of resources. If a persistent cluster is needed, make it small. Clusters can be resized anytime.

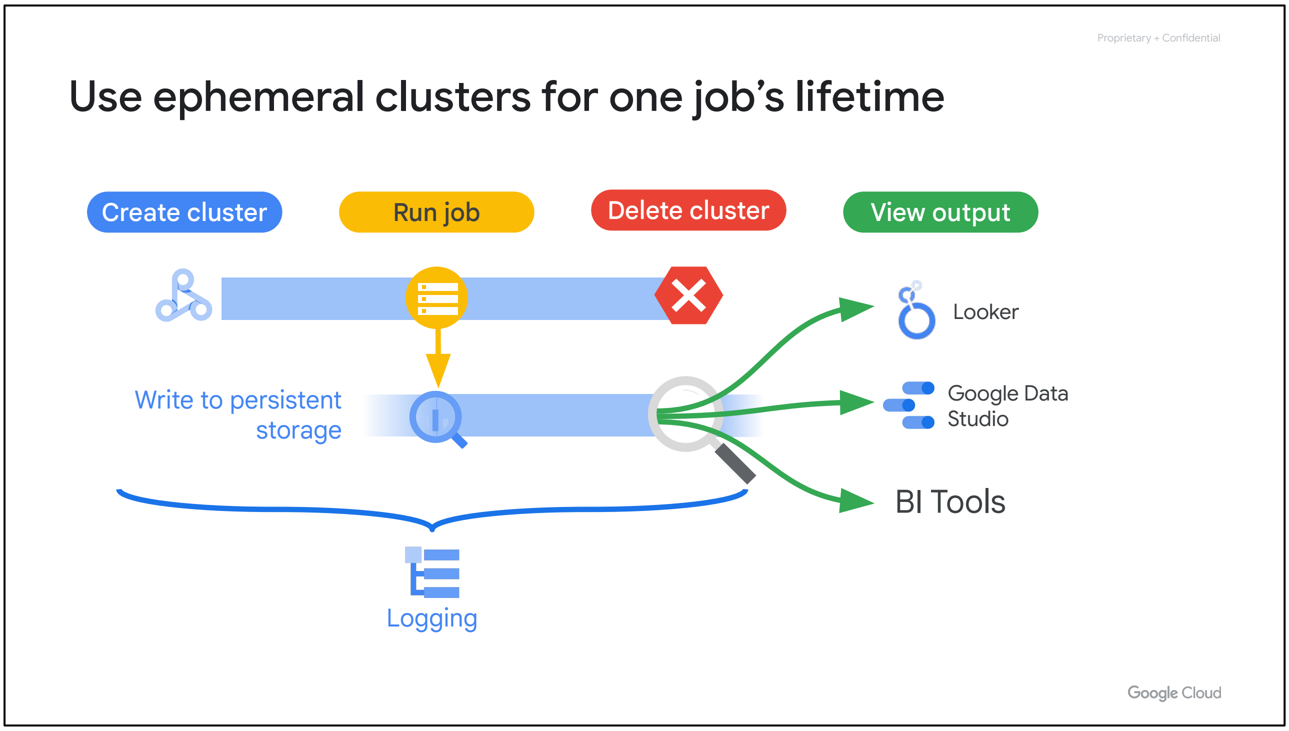
Ephemeral model is the recommended route but it requires storage to be decoupled from compute.



Split clusters and jobs

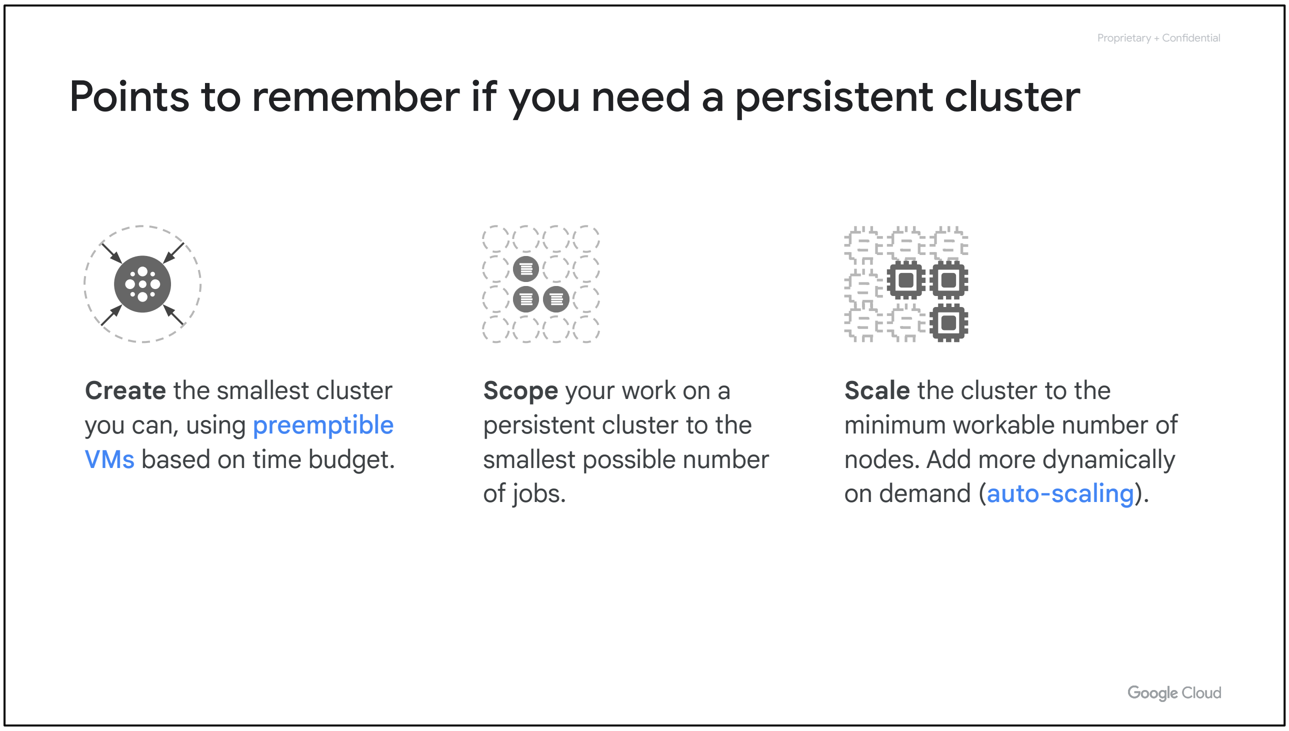
Separate job shapes and separate clusters. Decompose even further with job-scoped clusters.

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


Proprietary + Confidential
Use ephemeral clusters for one job's lifetime
Create cluster
Run job
Delete cluster
View output
Write to persistent storage
Looker
Google Data Studio
BI Tools
Logging
Google Cloud

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.



Points to remember if you need a persistent cluster

Create the smallest cluster you can, using preemptible VMs based on time budget.

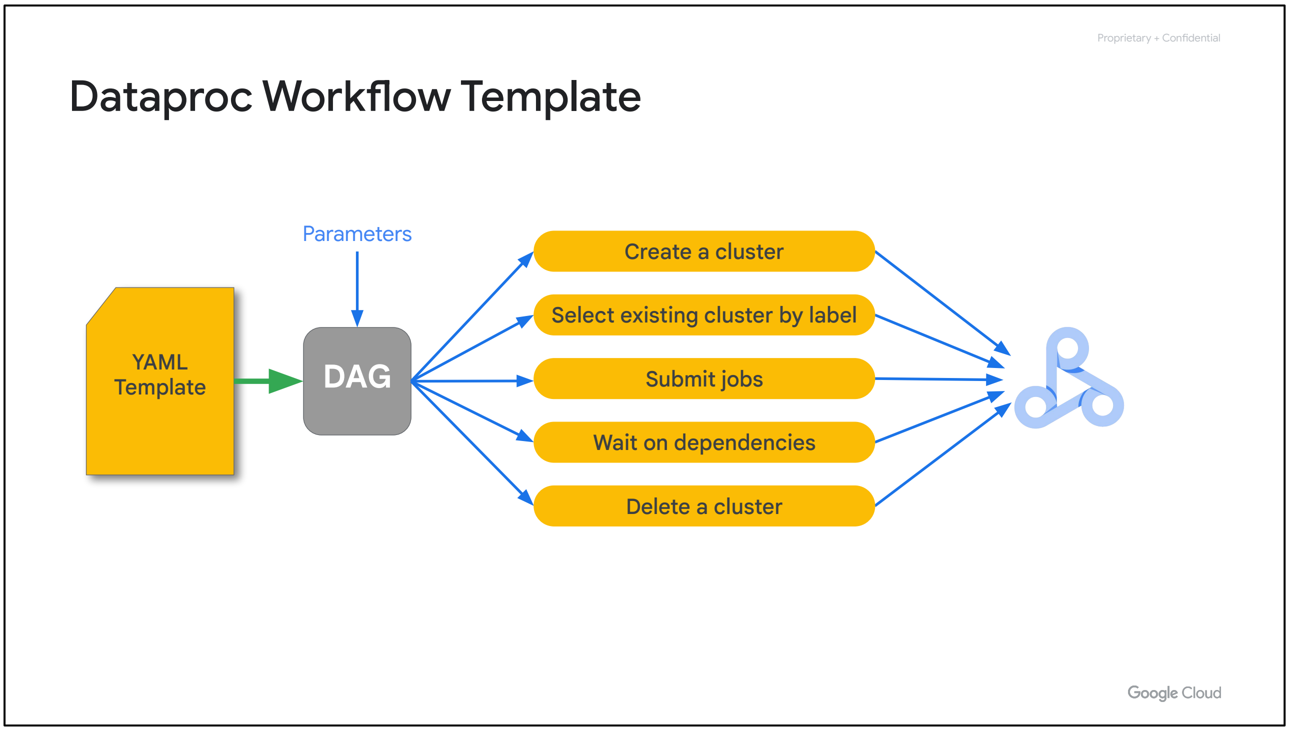
Scope your work on a persistent cluster to the smallest possible number of jobs.

Scale the cluster to the minimum workable number of nodes. Add more dynamically on demand (auto-scaling).

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.
- And scaling the cluster to the minimum workable number of nodes, adding more dynamically to meet demand.



Dataproc Workflow Template

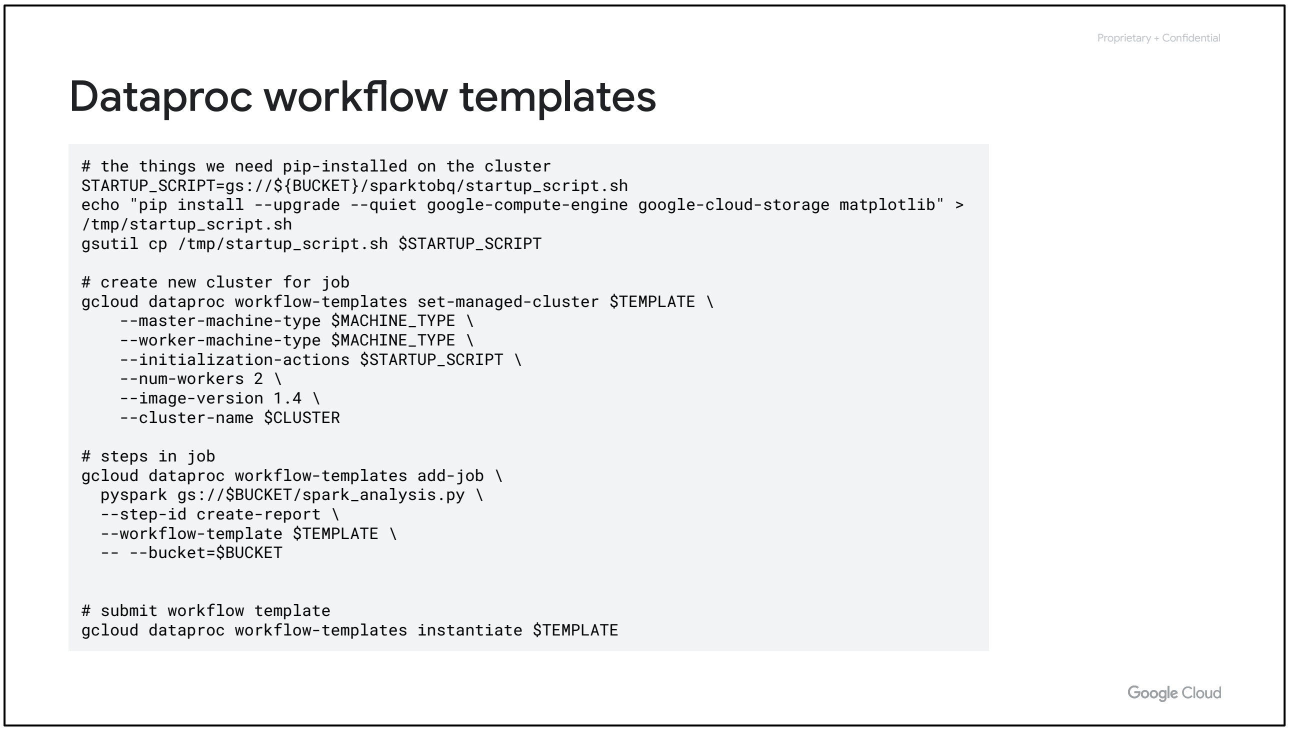
The Dataproc Workflow Template is a YAML file that is processed through a Directed Acyclic Graph or DAG.

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's currently only available through the gcloud command and the REST API. It cannot be accessed through the Cloud Console.

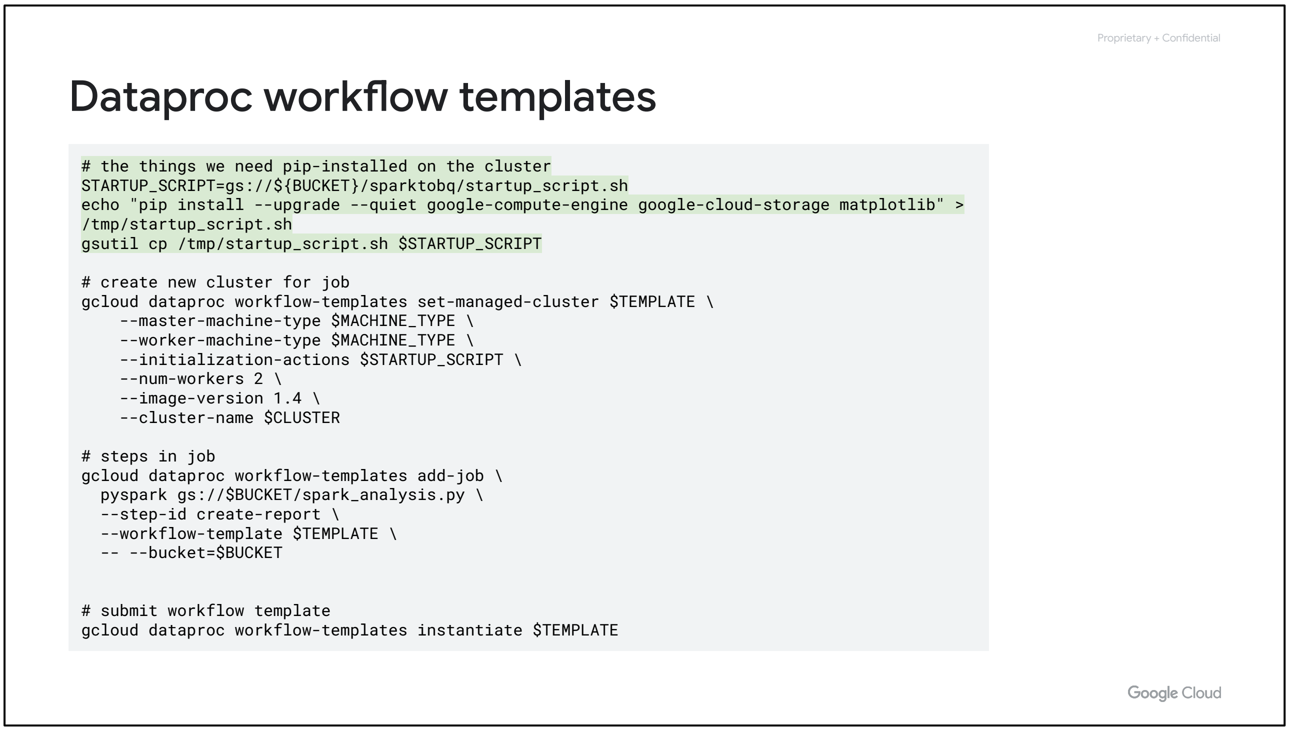
The Workflow Template becomes active when it is instantiated into the DAG. The Template can be submitted multiple times with different parameter values. You can also write a template inline in the the gcloud command, and you can list workflows and workflow metadata to help diagnose issues.



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
```

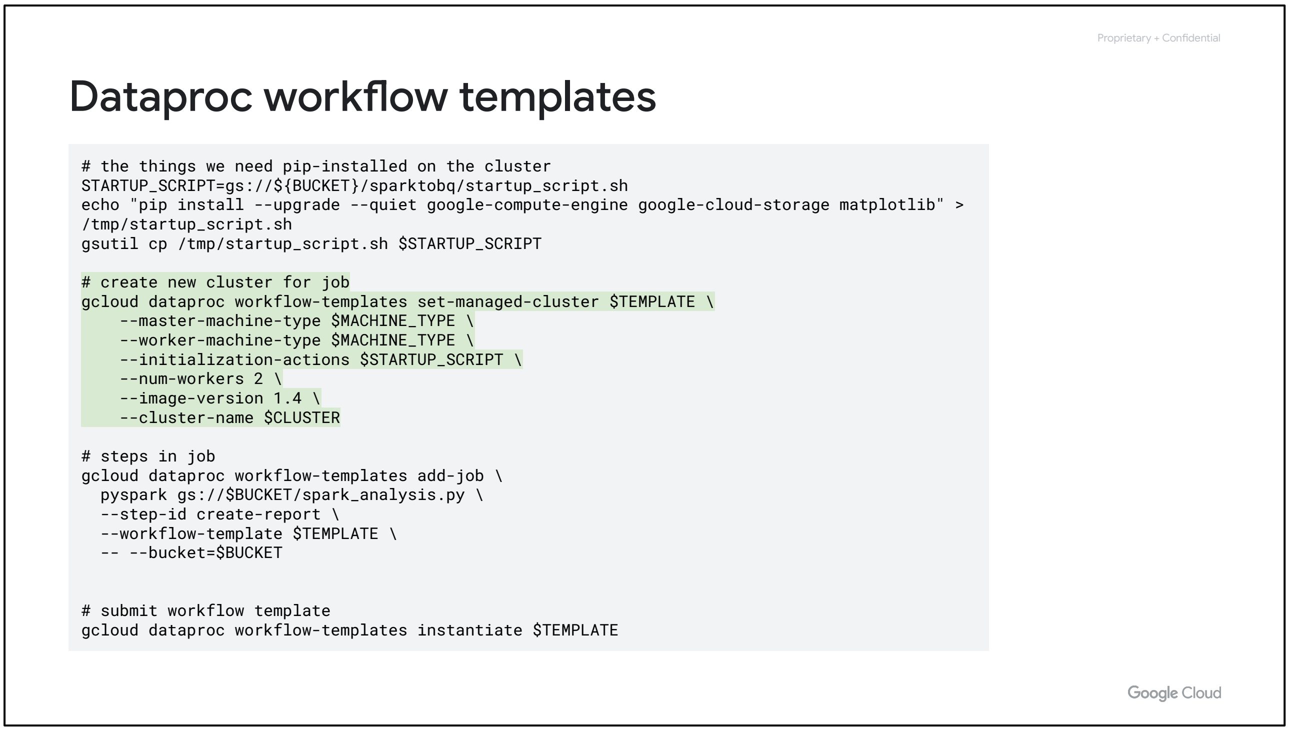
Here's an example of a Dataproc workflow template.



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
```

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.

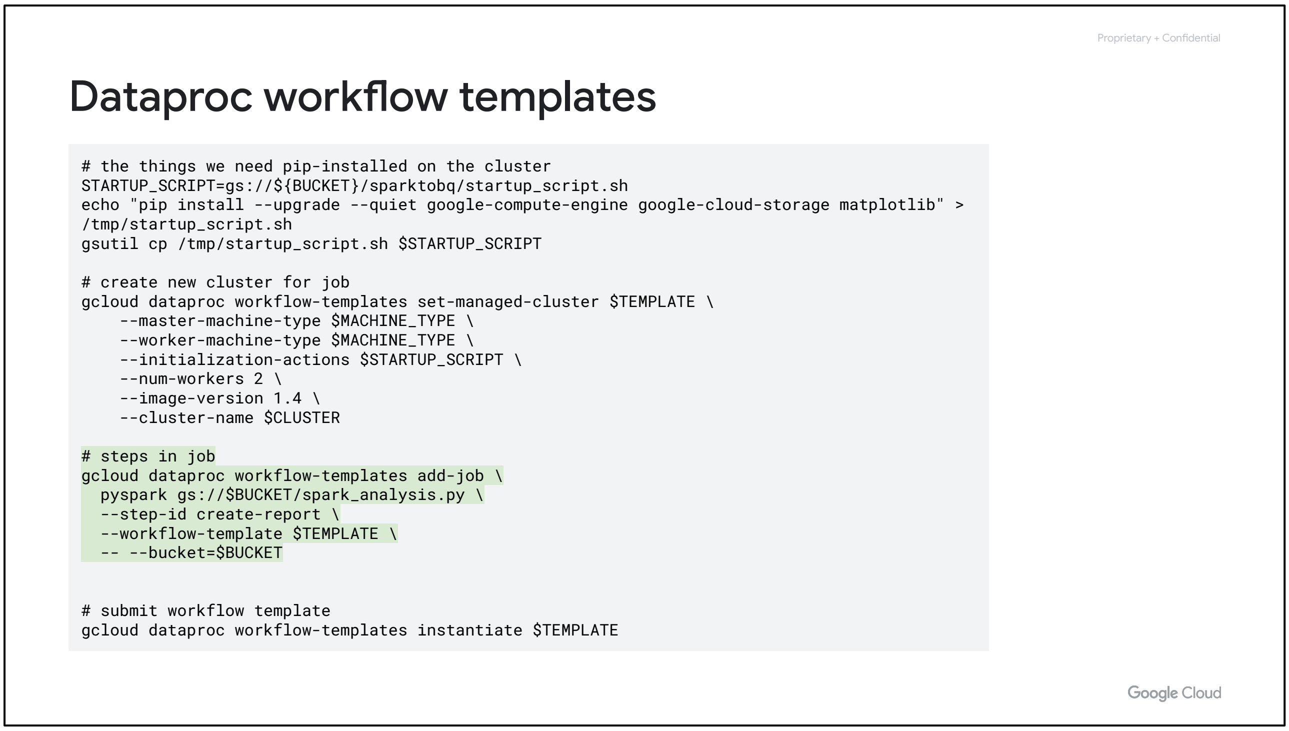


Dataproc workflow templates

```
# 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

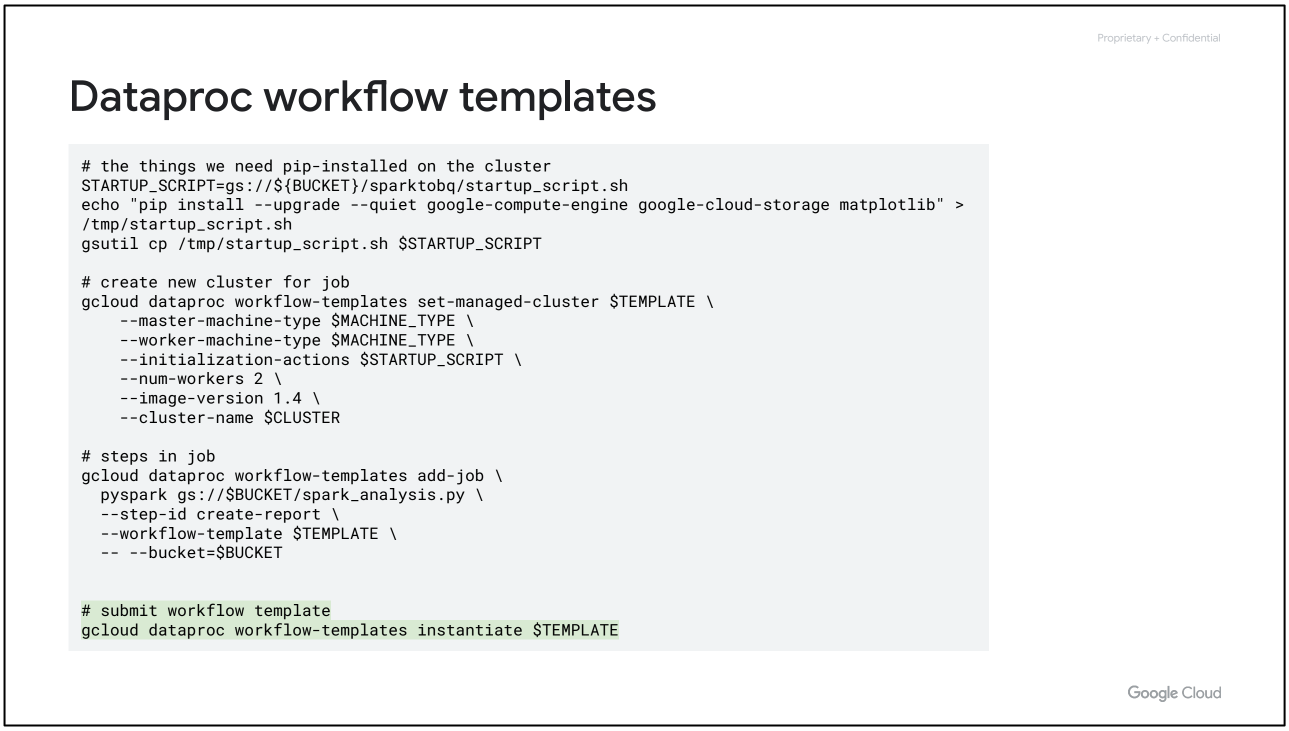
Next, we use the gcloud command for creating a new cluster in advance of running our job. We specify cluster parameters like the template to be used in our desired architecture and what machine types and image versions we want for hardware and software.



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
```

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 Cloud Storage bucket that we control.

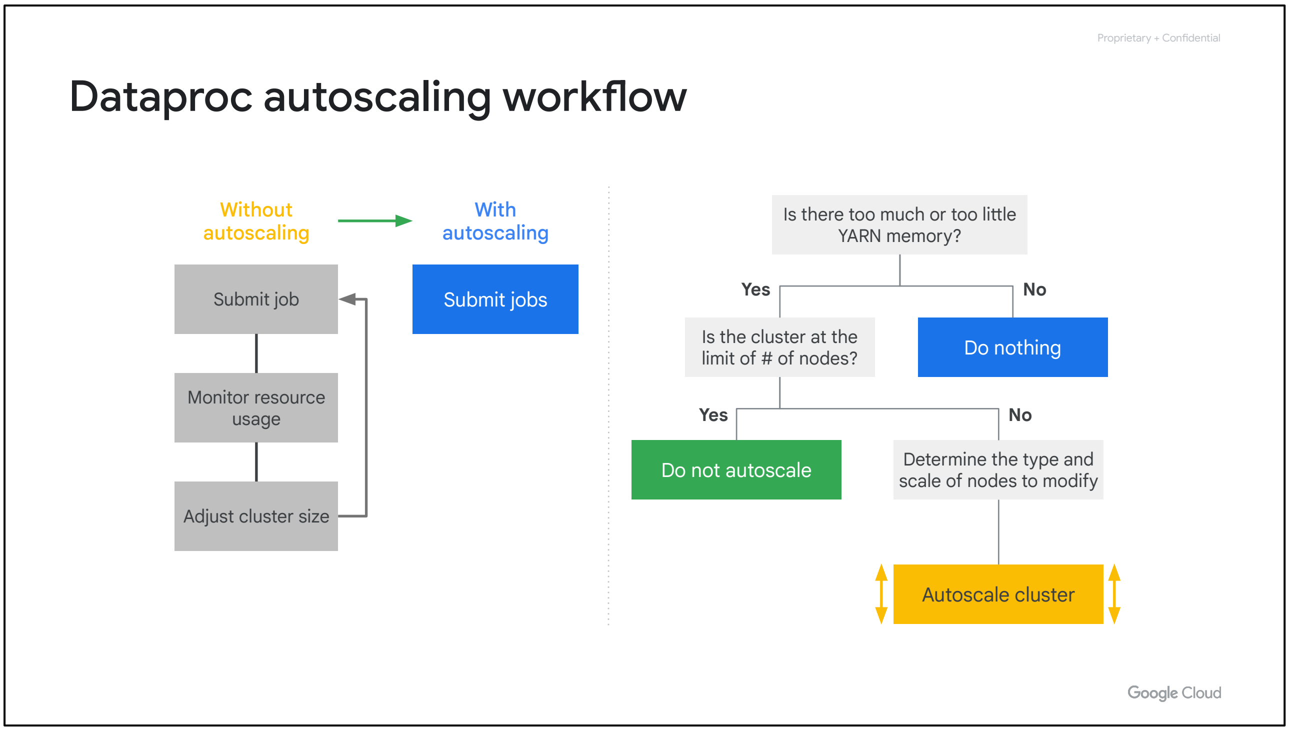


Dataproc workflow templates

```
# 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

Lastly, we need to submit this template itself as a new workflow template as you see with the last command.

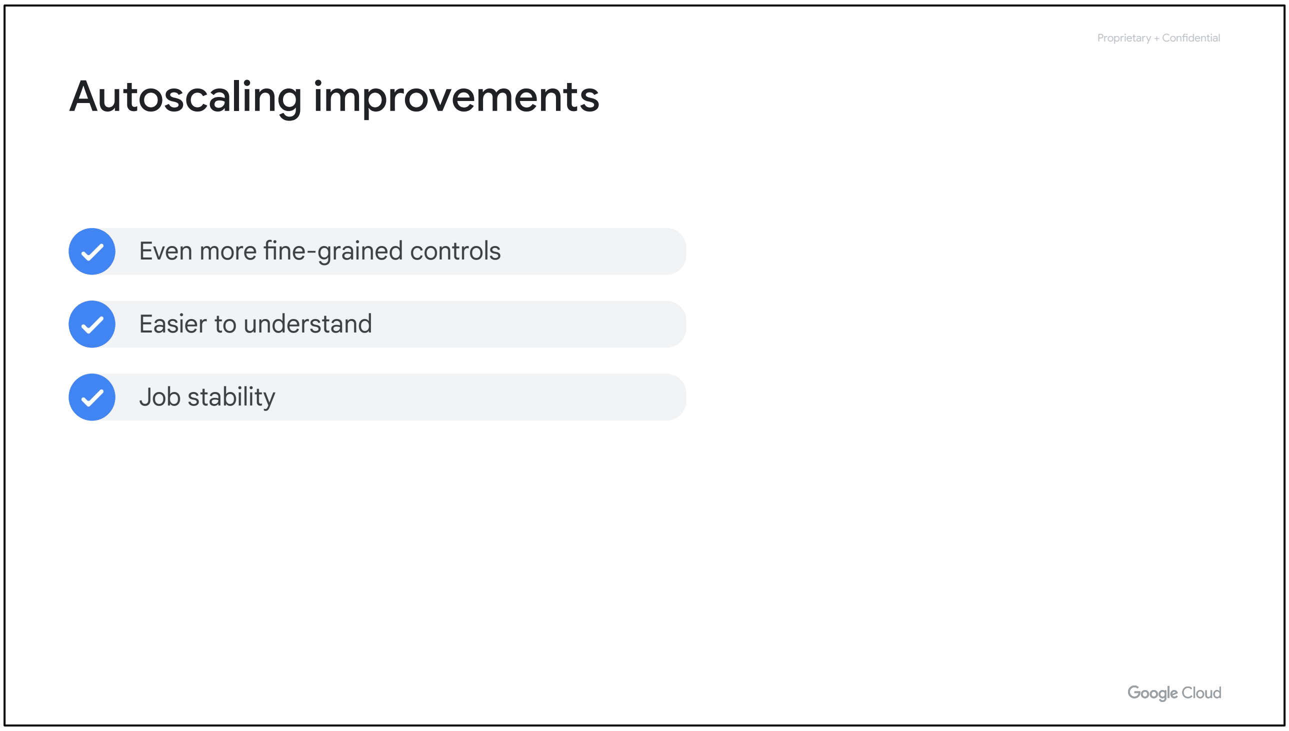


Dataproc autoscaling workflow

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

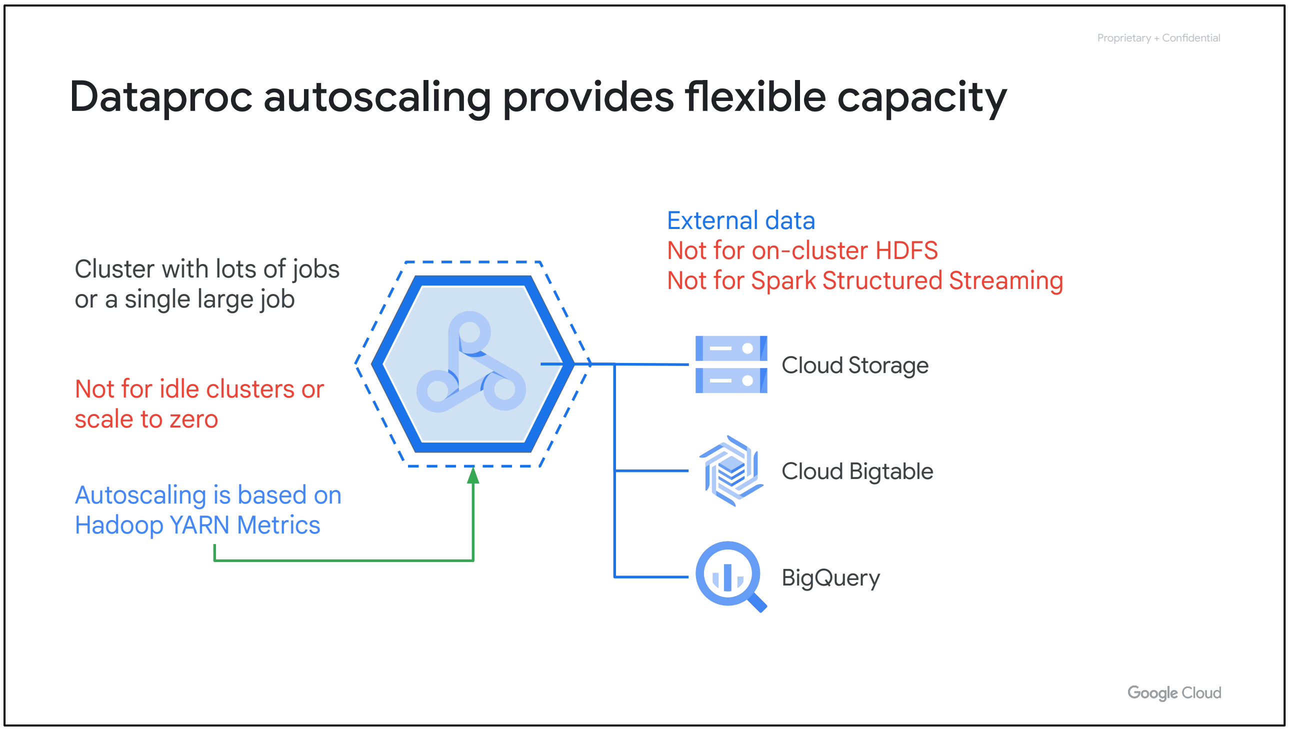
- ✓ Even more fine-grained controls
- ✓ Easier to understand
- ✓ Job stability

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, and autoscaling policies can be shared between multiple clusters.

- Autoscaling is now **easier to understand**.
  YARN and HDFS dashboards can be viewed in a in cluster page, and the autoscaling decision history is available in Cloud Logging.

- And **job stability** is provided through the ability to scale MapReduce and Spark jobs without losing progress.



Proprietary + Confidential

Dataproc autoscaling provides flexible capacity

Cluster with lots of jobs or a single large job
Not for idle clusters or scale to zero
Autoscaling is based on Hadoop YARN Metrics
External data
Not for on-cluster HDFS
Not for Spark Structured Streaming
Cloud Storage
Cloud Bigtable
BigQuery

Google Cloud

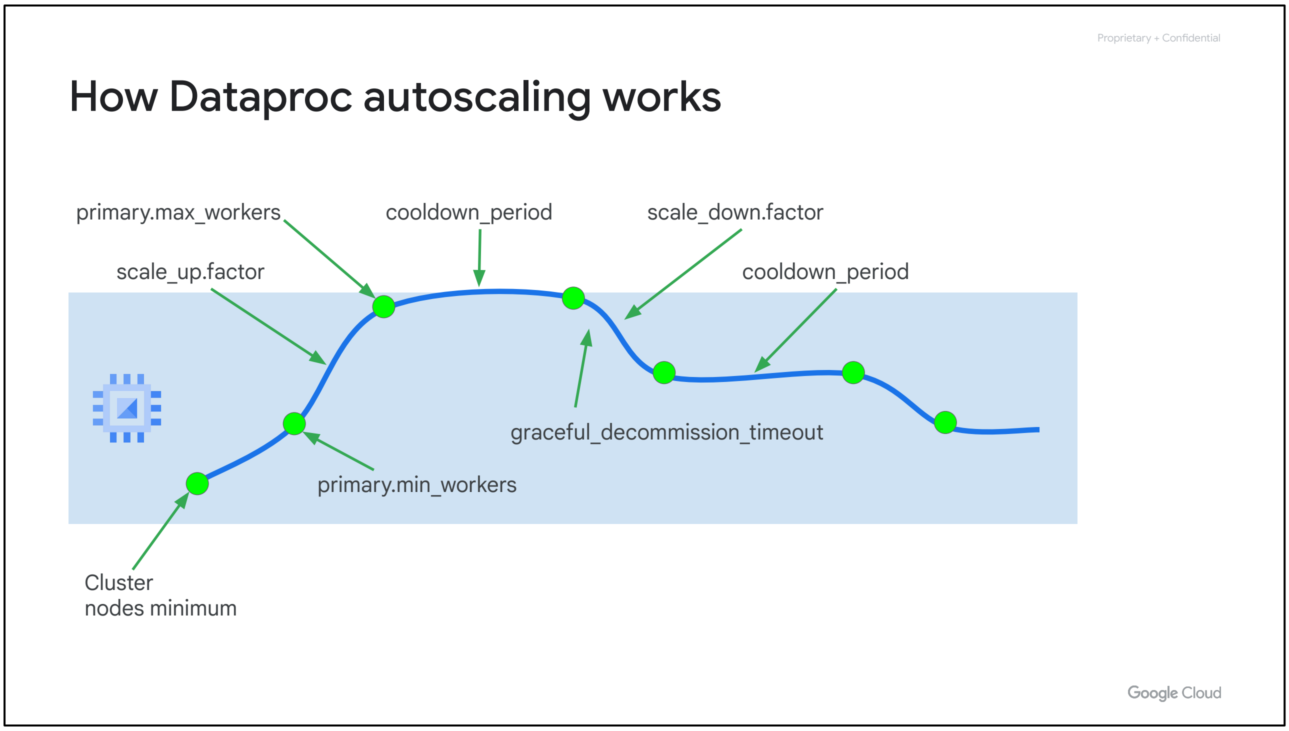
Dataproc autoscaling provides flexible capacity for more efficient utilization, making scaling decisions based on Hadoop YARN Metrics.

It’s designed to be used only with off-cluster persistent data, not on-cluster HDFS or HBase.

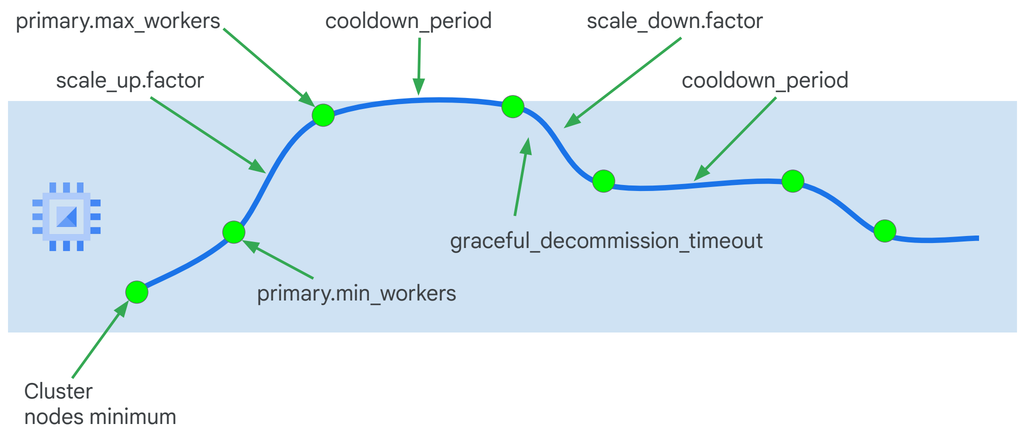
It works best with a cluster that processes a lot of jobs or that processes a single large job.

It doesn’t support Spark Structured Streaming, a streaming service built on top of Spark SQL.

It’s also not designed to scale to zero. So it’s not the best for sparsely utilized or idle clusters. In these cases it’s equally fast to terminate a cluster that’s idle and create a new cluster when it’s needed. For that purpose you would look at Dataproc Workflows or Cloud Composer, and Cluster Scheduled Deletion.



How Dataproc autoscaling works



One of the things that you want to consider when working with autoscaling is setting the 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.

If there is heavy load on the cluster, 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.

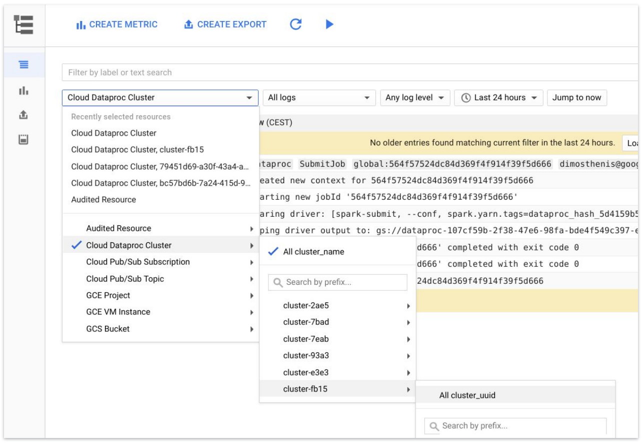
Notice there is a **scale down factor**. In this case it is scaling down by one node at a time for a more leisurely reduction in capacity.

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.

Use Cloud Operations logging and performance monitoring



```
w (CEST)
No older entries found matching current filter in the last 24 hours.
taproc  SubmitJob  global:564f57524dc84d369f4f914f39f5d666  dimosthenis@goog
eated new context for 564f57524dc84d369f4f914f39f5d666
arting new jobId '564f57524dc84d369f4f914f39f5d666'
aring driver: [spark-submit, --conf, spark.yarn.tags=dataproc_hash_5d4159b5
ping driver output to: gs://dataproc-107cf59b-2f38-47e6-98fa-bde4f549c397-e
d666' completed with exit code 0
d666' completed with exit code 0
24dc84d369f4f914f39f5d666
```

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 executors.

Note however, that If you submit a Spark job by connecting directly to the primary 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.

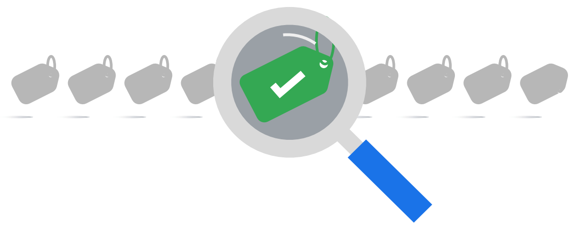
All other logs are located in different files inside the machines of the cluster. It's possible to see the logs for each container from the Spark app web UI (or from the History Server after the program ends) in the Executors tab. You need to browse through each Spark container to view each log. If you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

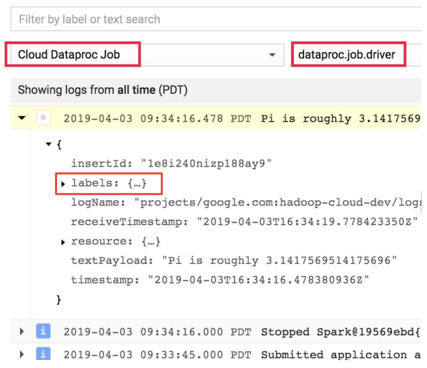
In a Dataproc cluster, YARN is configured to collect all these logs by default, and they're available in Cloud Logging. Logging provides a consolidated and concise view of all logs so that you don't need to spend time browsing among container logs to find errors.

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 by its ID. You can get the application ID from the driver output.

Create labels on clusters and jobs to find logs faster

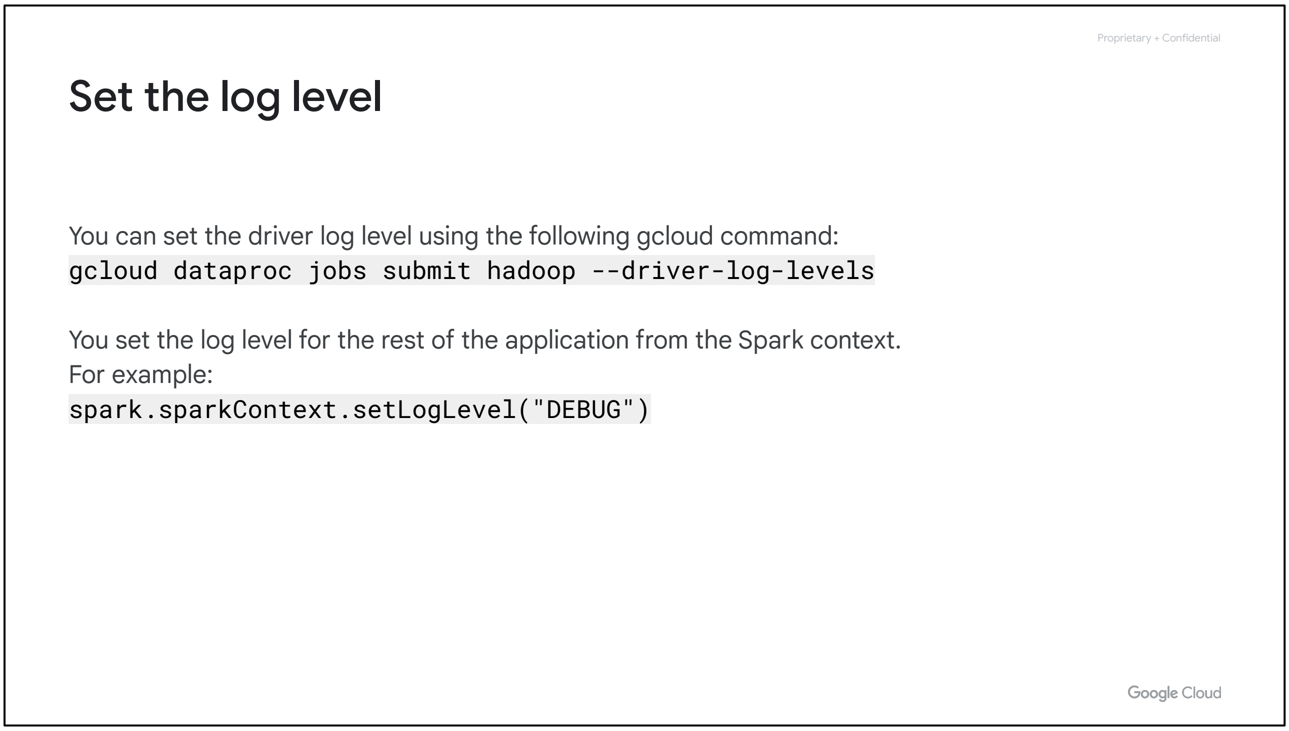



Filter by label or text search
Cloud Dataproc Job
dataproc.job.driver
Showing logs from **all time** (PDT)

```
▾ 2019-04-03 09:34:16.478 PDT Pi is roughly 3.1417569
    ▾ {
        insertId: "1e8i240nizp188ay9"
      ▸ labels: {…}
        logName: "projects/google.com:hadoop-cloud-dev/logs
        receiveTimestamp: "2019-04-03T16:34:19.778423350Z"
      ▸ resource: {…}
        textPayload: "Pi is roughly 3.1417569514175696"
        timestamp: "2019-04-03T16:34:16.478380936Z"
      }
▸ 2019-04-03 09:34:16.000 PDT Stopped Spark@19569ebd{
▸ 2019-04-03 09:33:45.000 PDT Submitted application a
```

Google Cloud

To find logs faster, you can create and use your own labels for each cluster or for each Dataproc job. For example, you can create a label with the key environment, or ENV, as the value in the exploration and use it for your data exploration job. You can then get logs for all exploration job creations by filtering with the label *environment* with a value *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:

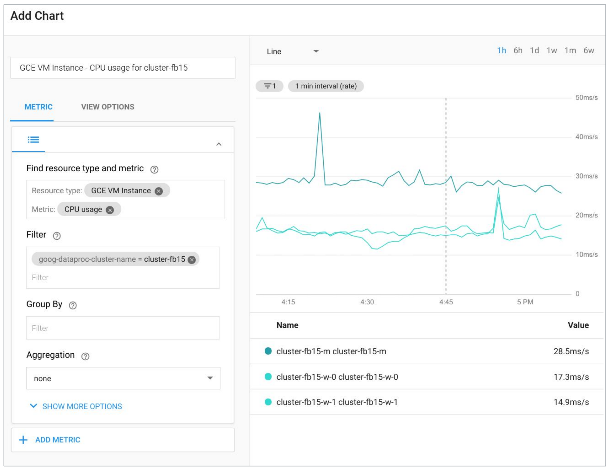
```
spark.sparkContext.setLogLevel("DEBUG")
```

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

You set the log level for the rest of the application from the Spark context. For example: spark.sparkContext.setLogLevel. For here, we’ll just say the example is DEBUG.

Proprietary + Confidential

Monitor your jobs


Add Chart

GCE VM Instance - CPU usage for cluster-fb15

METRIC | VIEW OPTIONS

Find resource type and metric

Resource type: GCE VM Instance

Metric: CPU usage

Filter

goog-dataproc-cluster-name = cluster-fb15

Group By

Aggregation: none

SHOW MORE OPTIONS

ADD METRIC

Line — 1h 6h 1d 1w 1m 6w

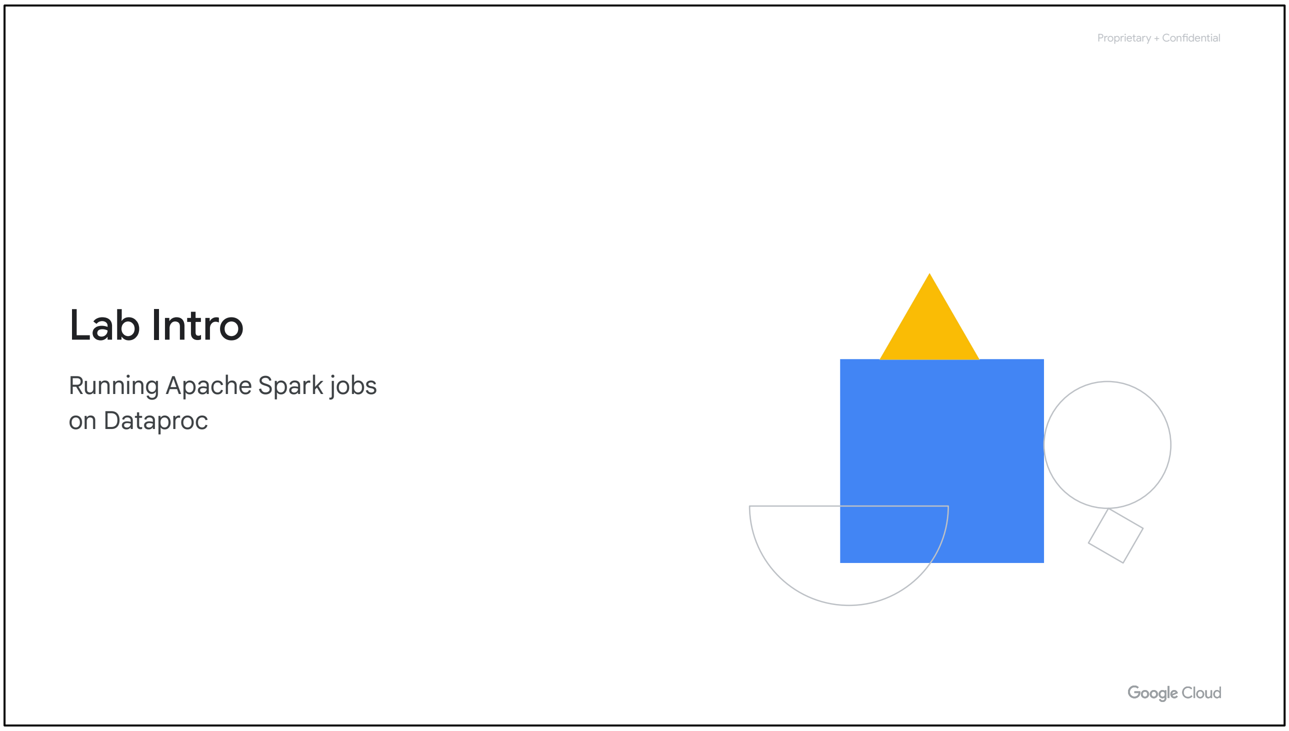
1 min interval (rate)

Name	Value
● cluster-fb15-m cluster-fb15-m	28.5ms/s
● cluster-fb15-w-0 cluster-fb15-w-0	17.3ms/s
● cluster-fb15-w-1 cluster-fb15-w-1	14.9ms/s

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. 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.

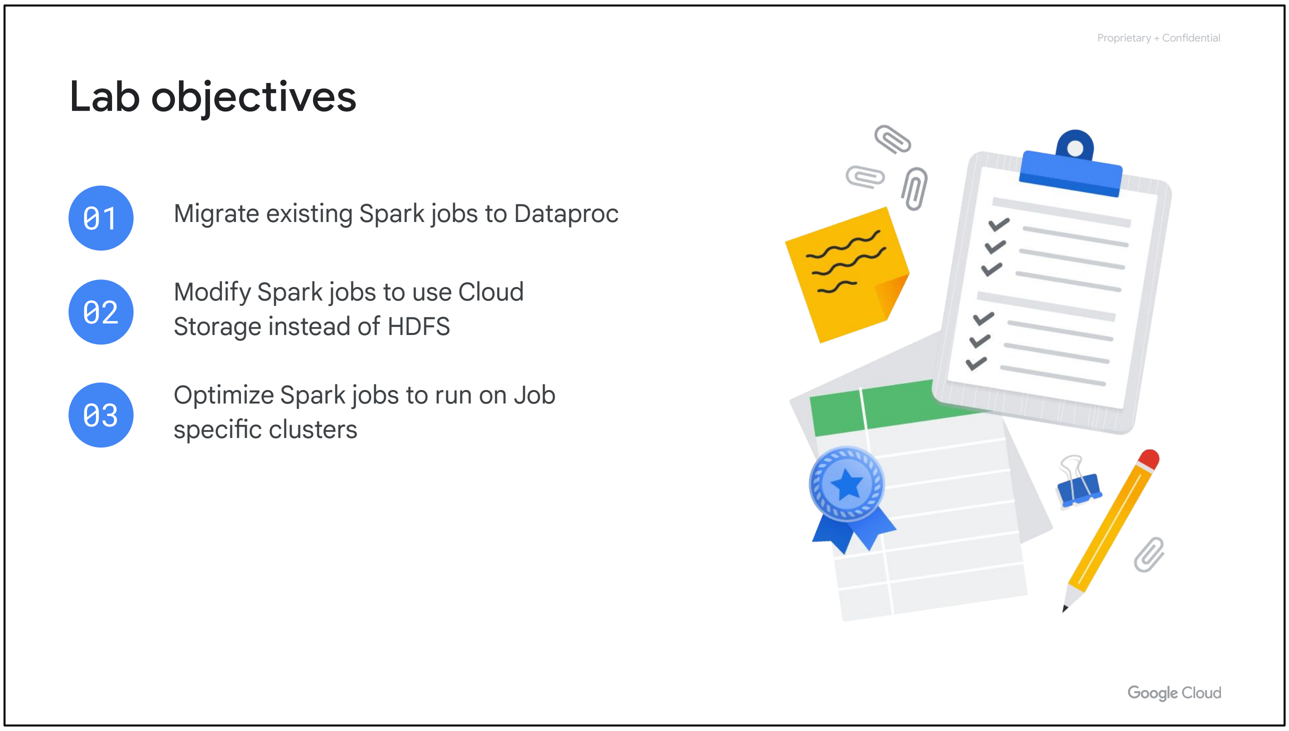
To view metrics for Spark queries, jobs, stages, or tasks, connect to the Spark application's web UI.



Lab Intro

Running Apache Spark jobs
on Dataproc

Now it's time for our lab. In this lab, you'll be running Apache Spark jobs on Dataproc. Let's take a look at where you're going do.



Lab objectives

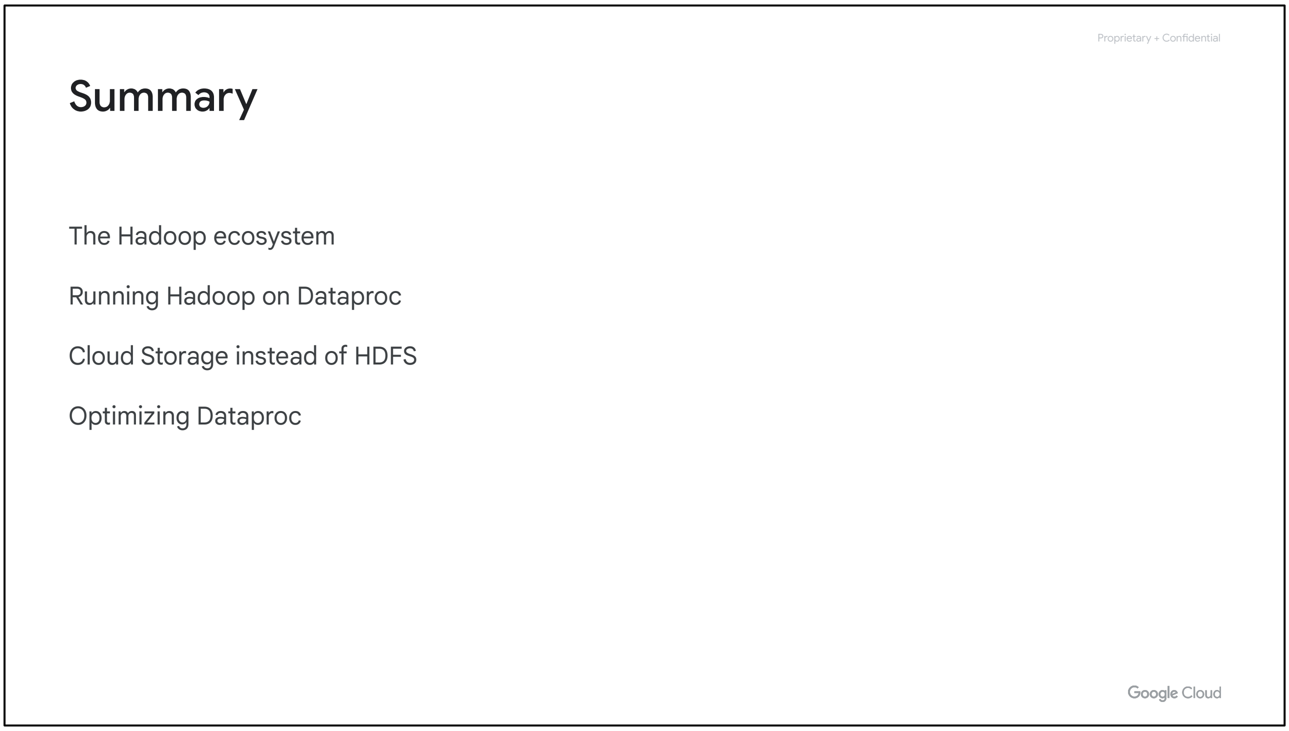
01 Migrate existing Spark jobs to Dataproc

02 Modify Spark jobs to use Cloud Storage instead of HDFS

03 Optimize Spark jobs to run on Job specific clusters

Google Cloud

First, you're going migrate existing Spark job code to Dataproc. Then, you'll be modifying your Spark jobs to use a different back-end, that's Google Cloud Storage instead of HDFS. Finally, you optimize Spark jobs to run on job specific clusters. Good luck.



Summary

The Hadoop ecosystem

Running Hadoop on Dataproc

Cloud Storage instead of HDFS

Optimizing Dataproc

Welcome to the end of the module. Let's do a brief recap.

You saw how you can run your entire Hadoop ecosystem on the Cloud, with Dataproc.

We covered the advantages of separating compute and storage for cost efficiency and performance by using Cloud Storage instead of HDFS.

Lastly, we discussed how you can optimize Dataproc by resizing your cluster as your needs change and enabled smart features like automatically turning down the cluster after a certain period of non-use.