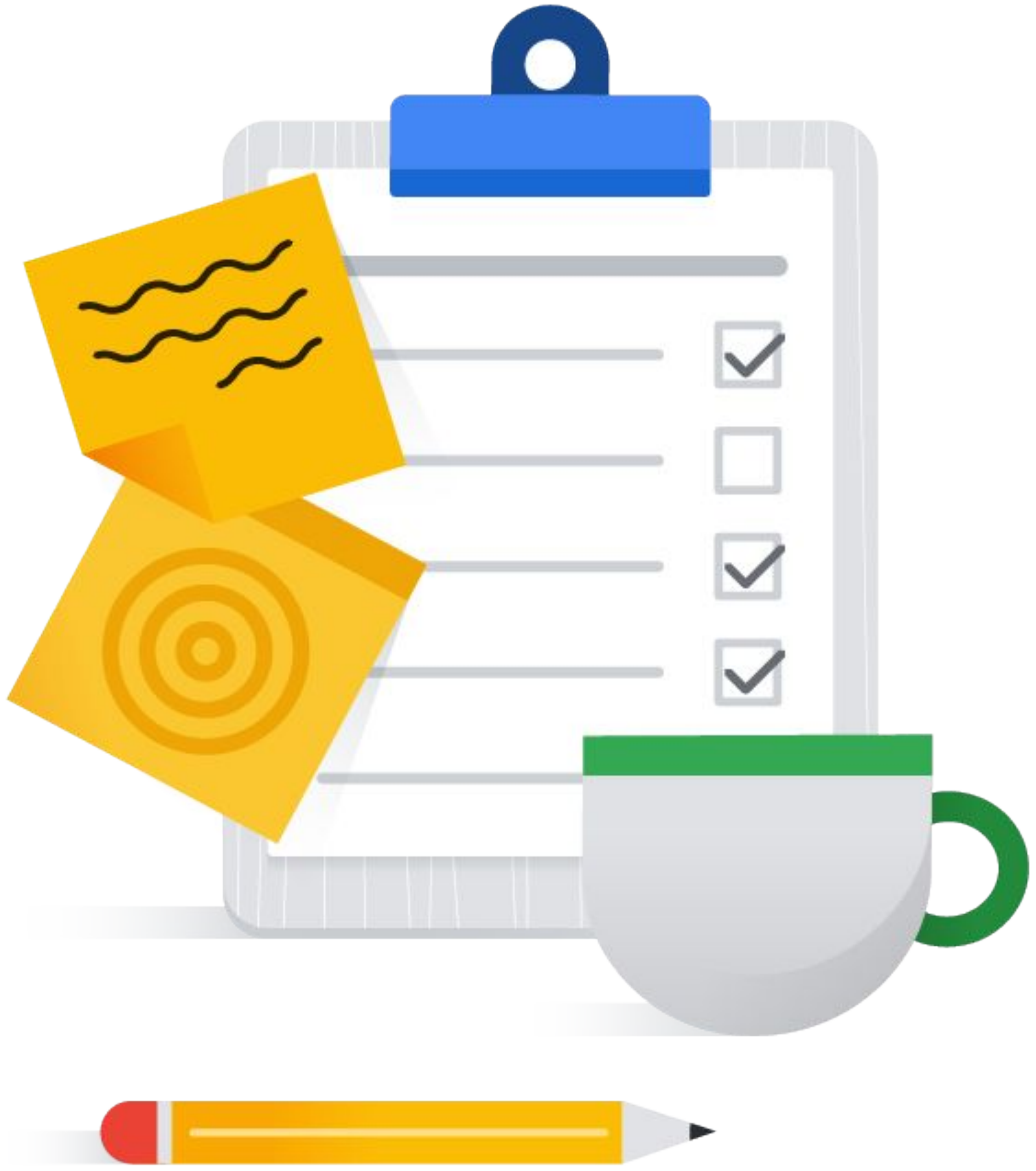
02

Executing Spark on Dataproc

Executing Spark on Dataproc



Executing Spark on Dataproc

01	The Hadoop ecosystem
02	Running Hadoop on Dataproc
03	Cloud Storage instead of HDFS
04	Optimizing Dataproc

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

The Hadoop ecosystem is very popular for Big Data workloads

Sqoop
(Import and
expo
rt
of relational data)

APACHE HBASE

HBase
(NoSQL
datastore)

HCatalog (Metadata)

Hive (SQL DW)

Pig (Scripting)

Spark
(Cluster data processing)

Mahout & Spark ML
(Machine learning)

Flink (Streams)

Presto
(Distributed SQL query)

Zookeeper
(Coordination)

OOZIE

Oozie
(Workflow
automation)

FLUME

Flume
(Log aggregation
and transport)

MapReduce (Cluster data processing)

YARN (Cluster resource management)

HDFS (Hadoop Distributed File System)

Ambari
(Management and
monitoring)

On-premises Hadoop clusters are not elastic

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

Dataproc simplifies Hadoop workloads on Google Cloud

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

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

Apache Spark SQL

Apache Spark MLlib

etc.

Apache Spark

spark.apache.org

Introduction to Building Batch Data Pipelines

01	The Hadoop ecosystem
02	Running Hadoop on Dataproc
03	Cloud Storage instead of HDFS
04	Optimizing Dataproc

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

Key features of Dataproc

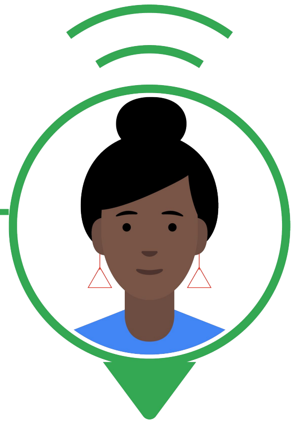
- Low cost
- Super-fast
- Resizable clusters
- Open source ecosystem
- Integrated
- Managed
- Versioning
- Highly available
- Developer tools
- Initialization actions
- Automatic/manual configuration

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...		

Use initialization actions to add other software to cluster at startup

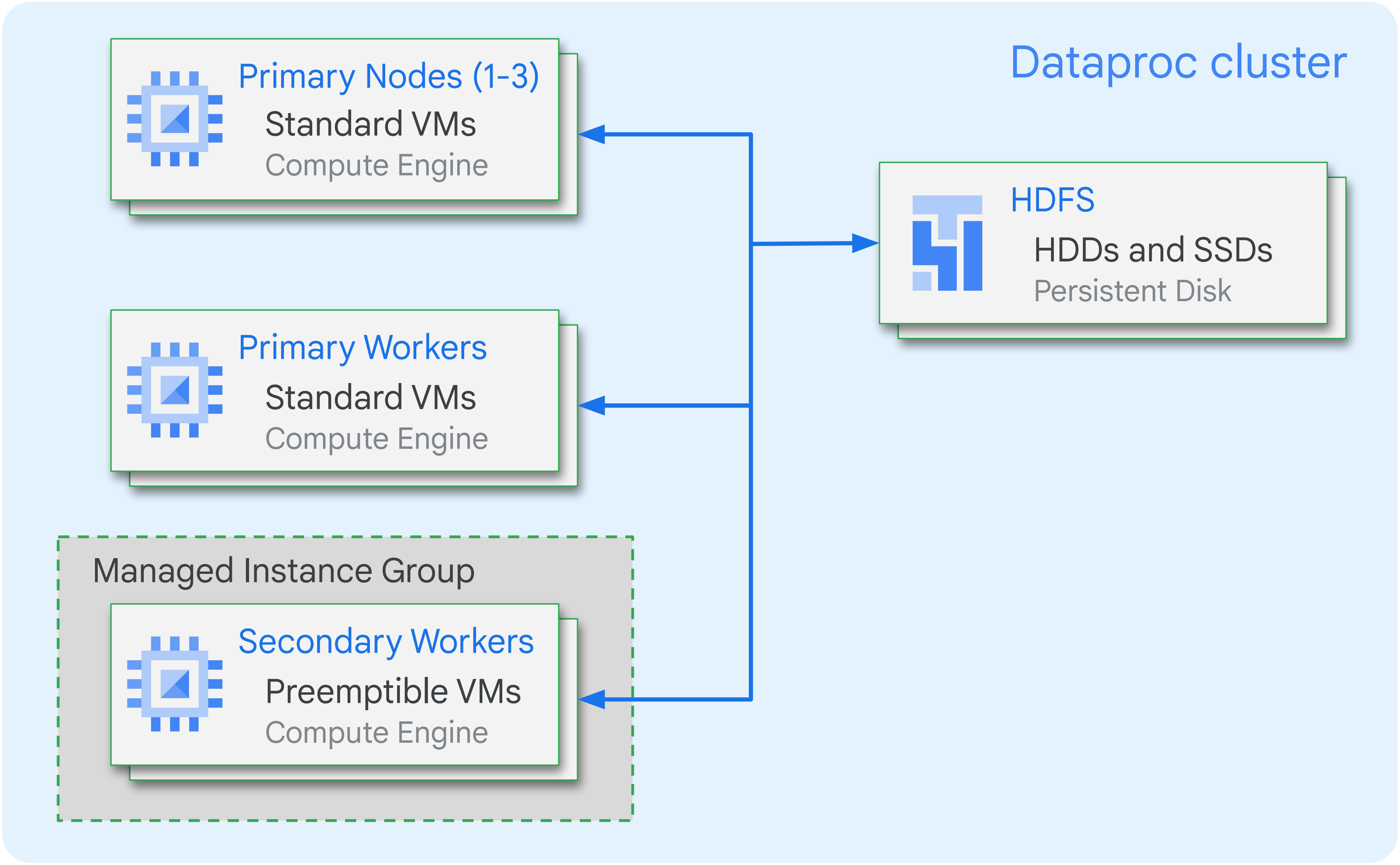
Use initialization actions to install additional components on the cluster.



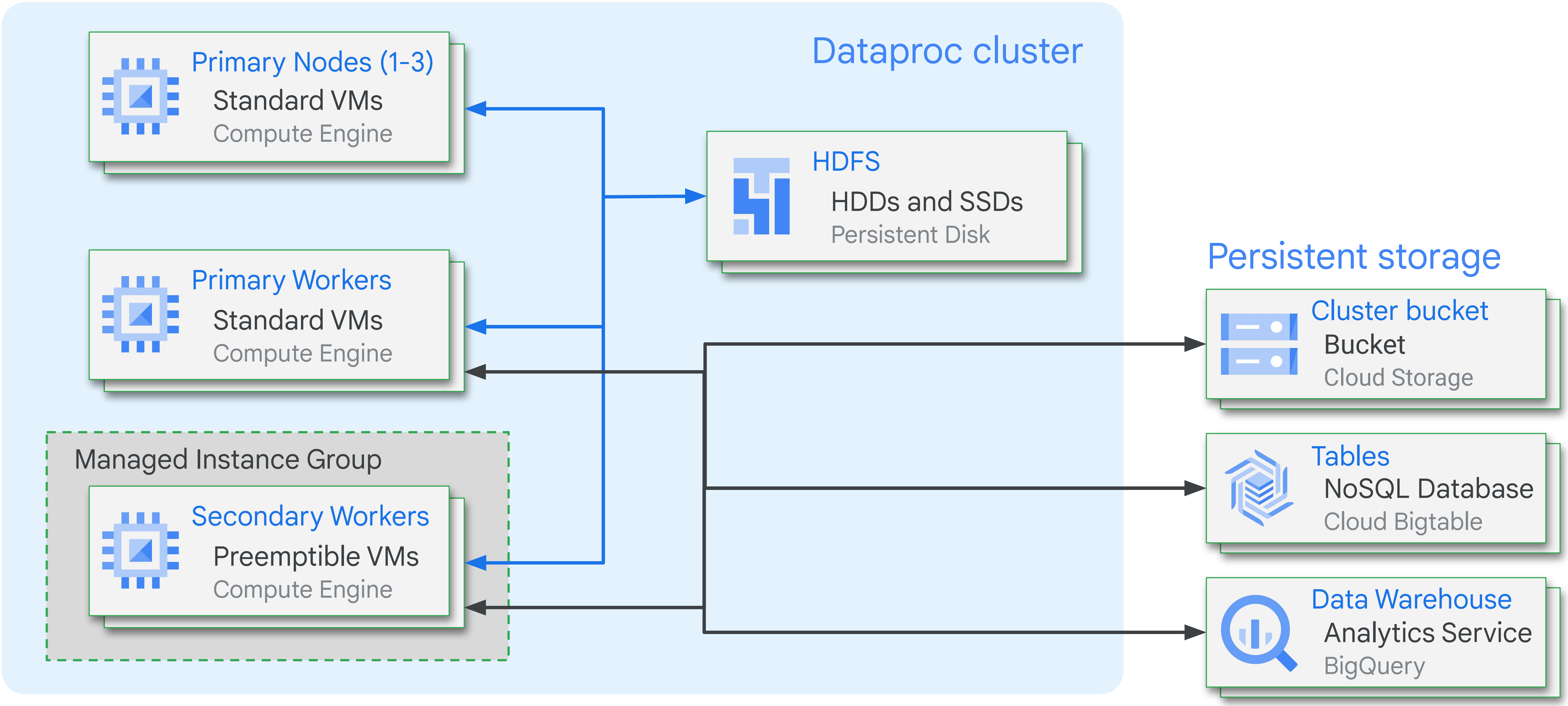
```
gcloud dataproc clusters create <CLUSTER_NAME> \
    --initialization-actions gs://$MY_BUCKET/hbase/hbase.sh \
      --num-masters 3 --num-workers 2
```

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

A Dataproc cluster has primary nodes, workers, and HDFS



Dataproc cluster can read/write to Google Cloud storage products



Using Dataproc

Setup | Configure | Optimize | Utilize | Monitor

Create a cluster

- Console
- gcloud command / YAML file
- Terraform configuration
- Cloud SDK 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)	

Optimize

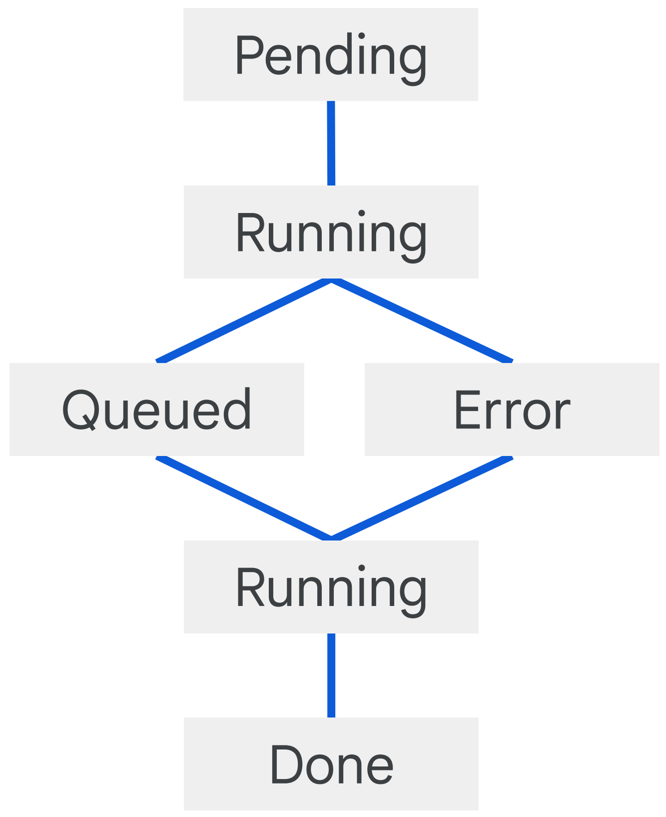
Setup | Configure | Optimize | Utilize | Monitor

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.

Utilize: Job submission

Setup | Configure | Optimize | Utilize | Monitor

Job Lifecycle



Submit a job

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

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

Executing Spark on Dataproc

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

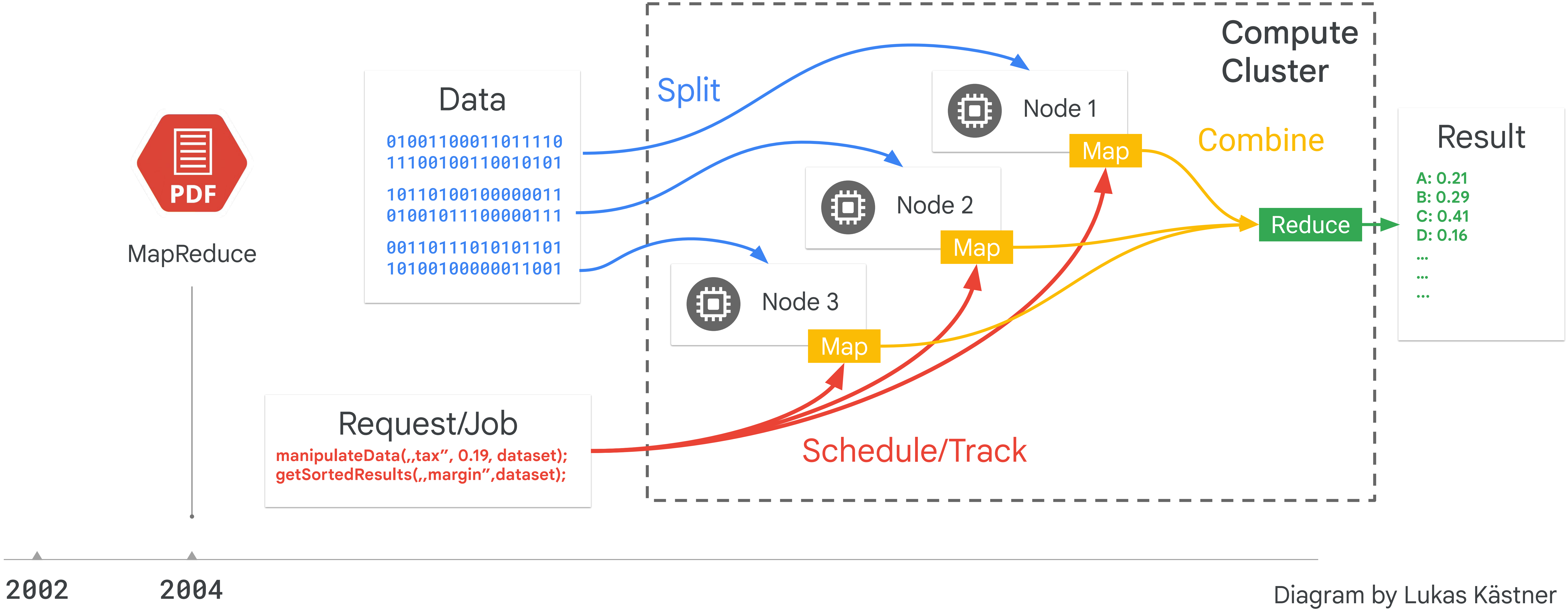


Diagram by Lukas Kästner

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

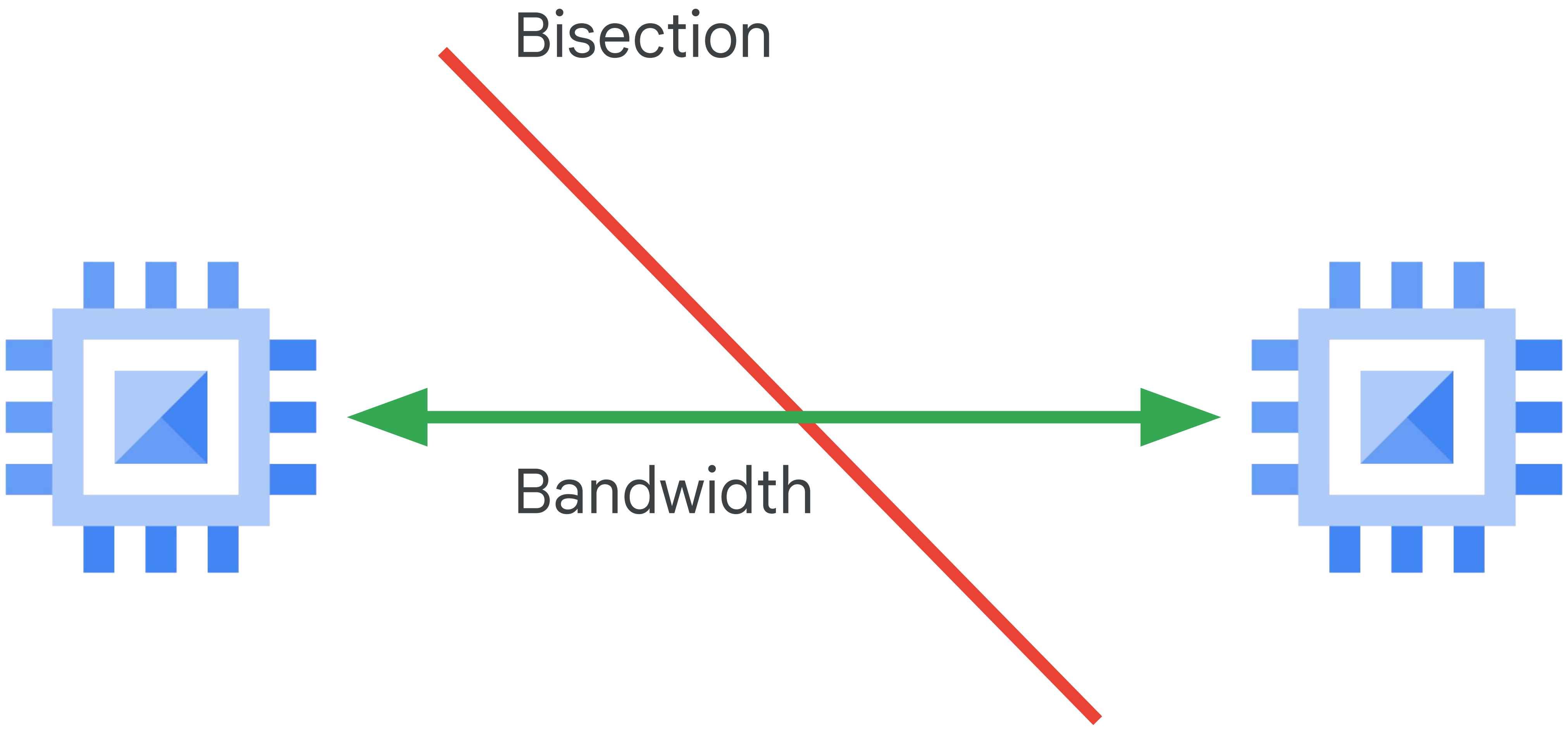
Default to 3 copies of each block (r=3)

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

Petabit bandwidth is a game-changer for big data



Process the data where it is without copying it

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

Processing

Dataflow

BigQuery Analytics

Dataproc

Start cluster
Run job
Delete cluster

Petabit bisection bandwidth

Storage

Cloud Storage (files)

BigQuery Storage (tables)

Cloud Bigtable (NoSQL)

Keep data in place on Cloud Storage

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

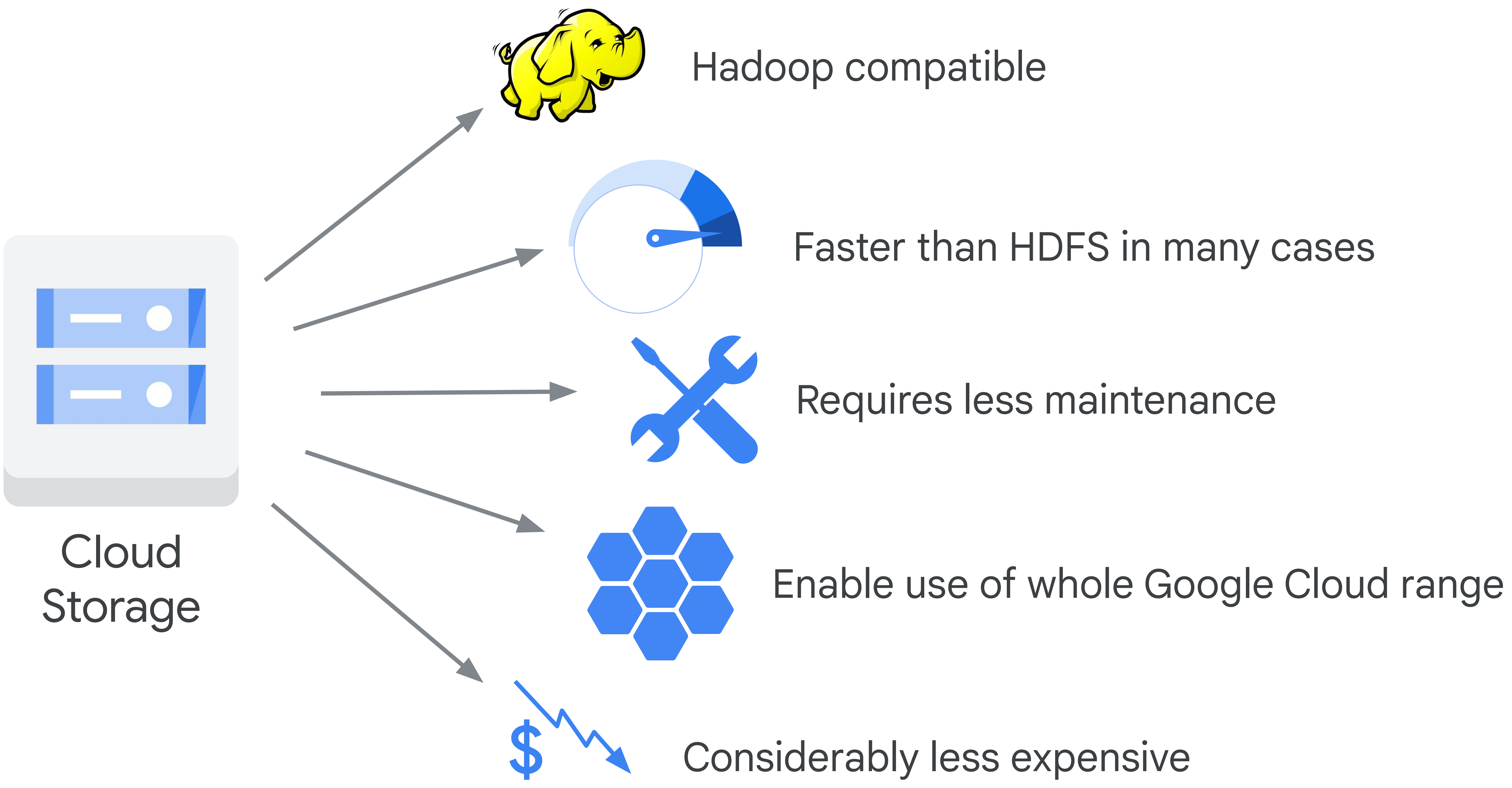
Petabit bisectional bandwidth network

Separate, specialize, and connect

Data storage services
Data processing services
Messaging services

2006 — 2010

Use Cloud Storage as the persistent data store



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

Performance best practices

Cloud Storage is optimized for bulk/parallel operations

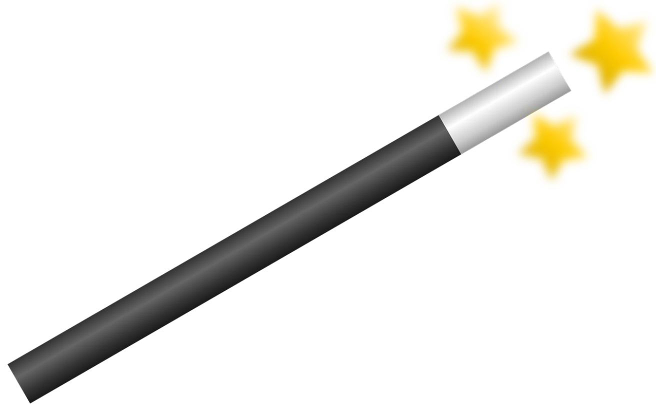
- Avoid small reads; use large block sizes where possible.
- Avoid iterating sequentially over many nested directories in a single job.

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"

Directory rename in HDFS not the same as in Cloud Storage

Cloud Storage has no concept of directories!

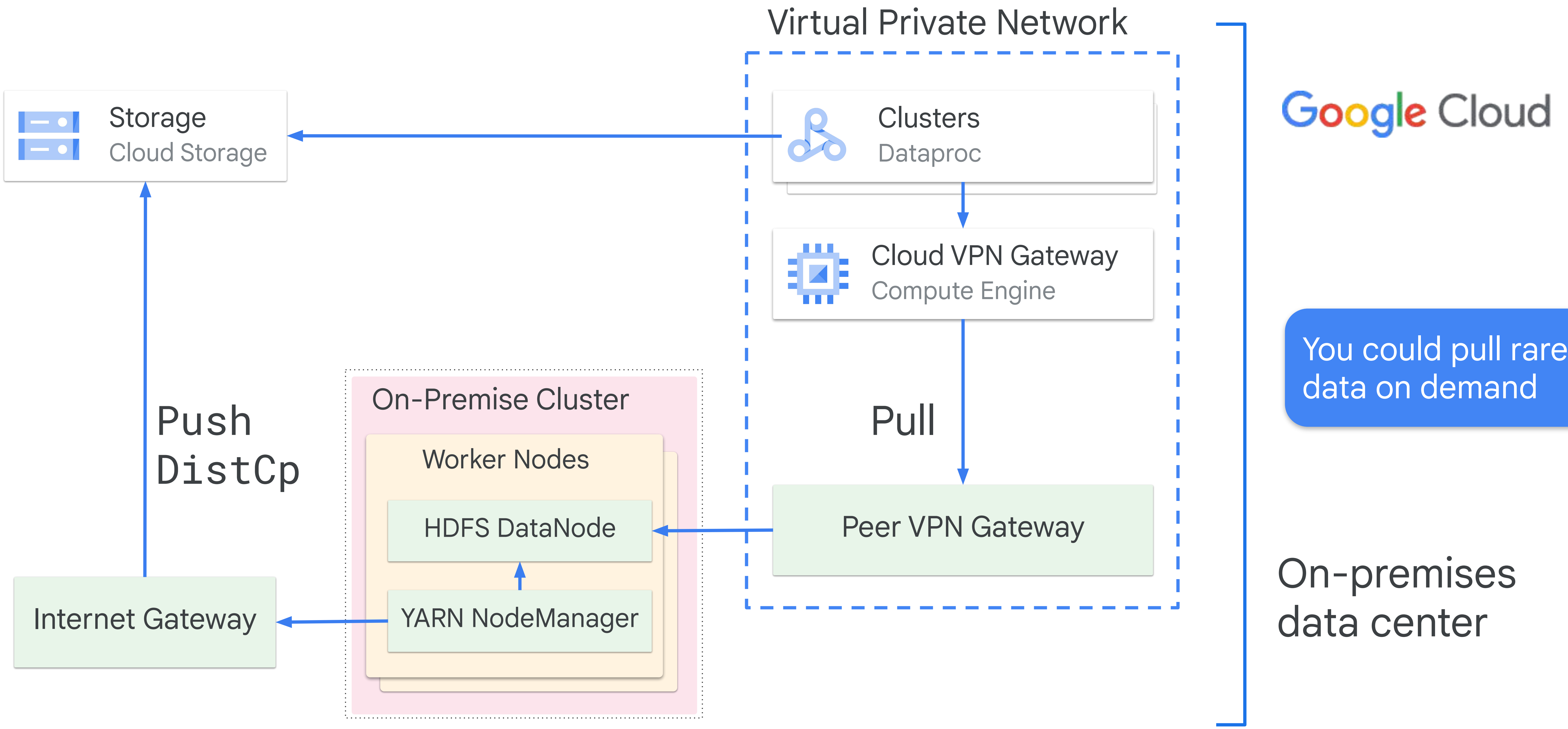
mv gs://foo/bar/ gs://foo/bar2

- 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

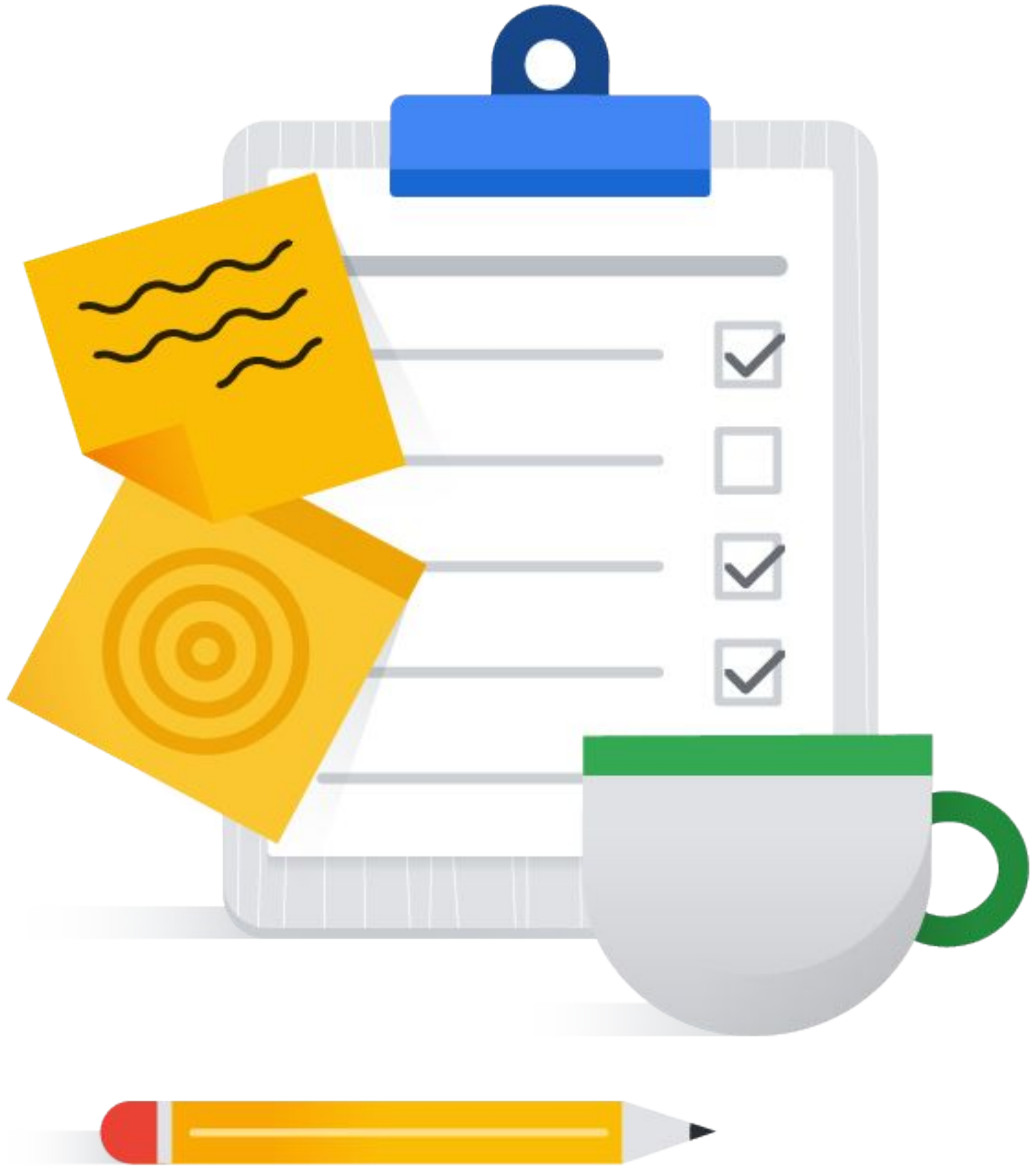
DistCp on-prem data that you will always need



You could pull rarely-used data on demand

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

Executing Spark on Dataproc



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

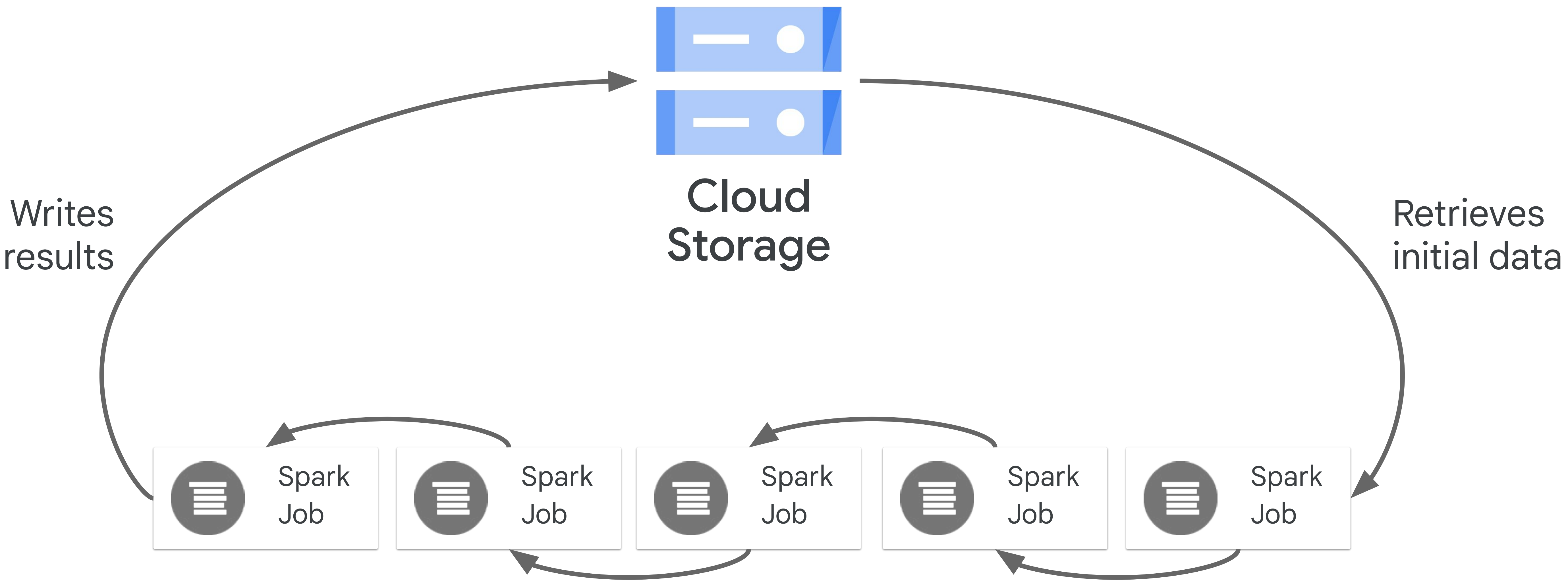
1. Where is your data, and where is your cluster?
2. Is your network traffic being funneled?
3. How many input files and Hadoop partitions are you trying to deal with?
4. Is the size of your persistent disk limiting your throughput?
5. Did you allocate enough virtual machines (VMs) to your cluster?

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.

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



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

Dataproc + Cloud Storage

Reduce costs by using this instead of HDFS

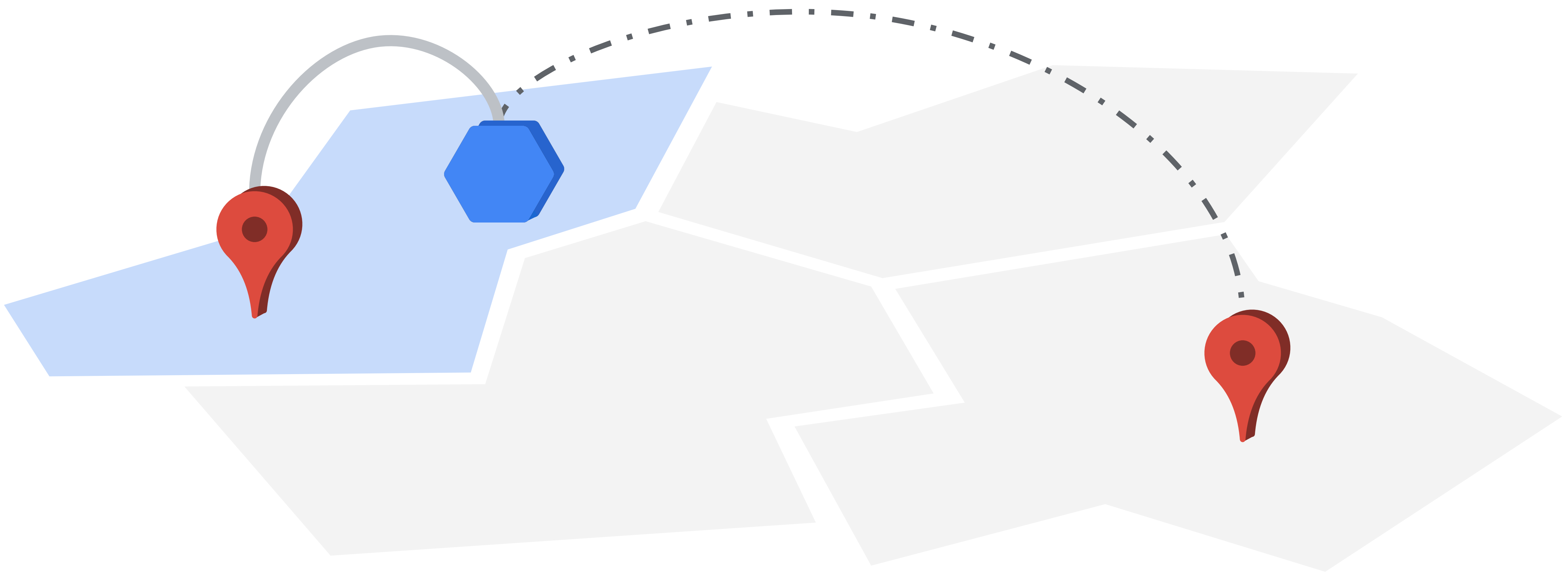
Using local HDFS? Consider re-sizing options

- 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.

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 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

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.

Cluster Scheduled Deletion

Idle

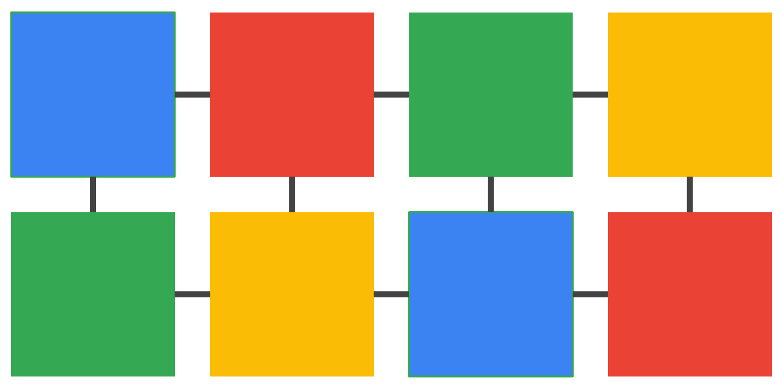
Timestamp

Duration

Minimum 10 minutes
Maximum 14 days
Granularity 1 second

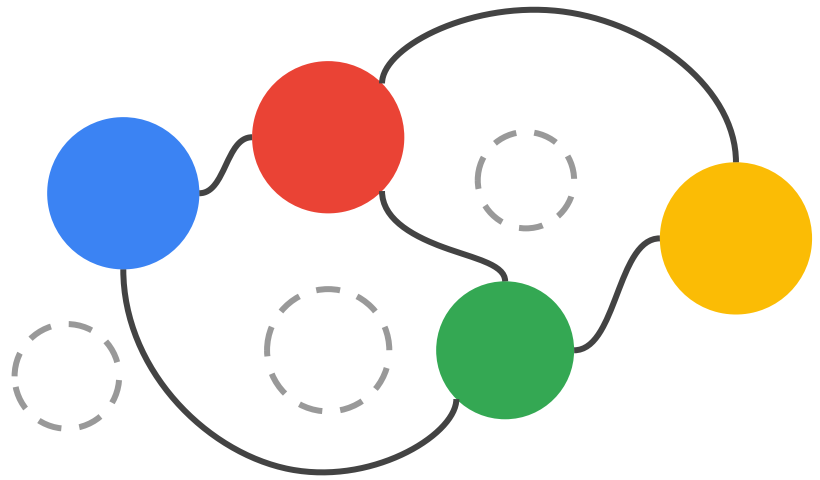
With ephemeral clusters, you only pay for what you use

Persistent clusters



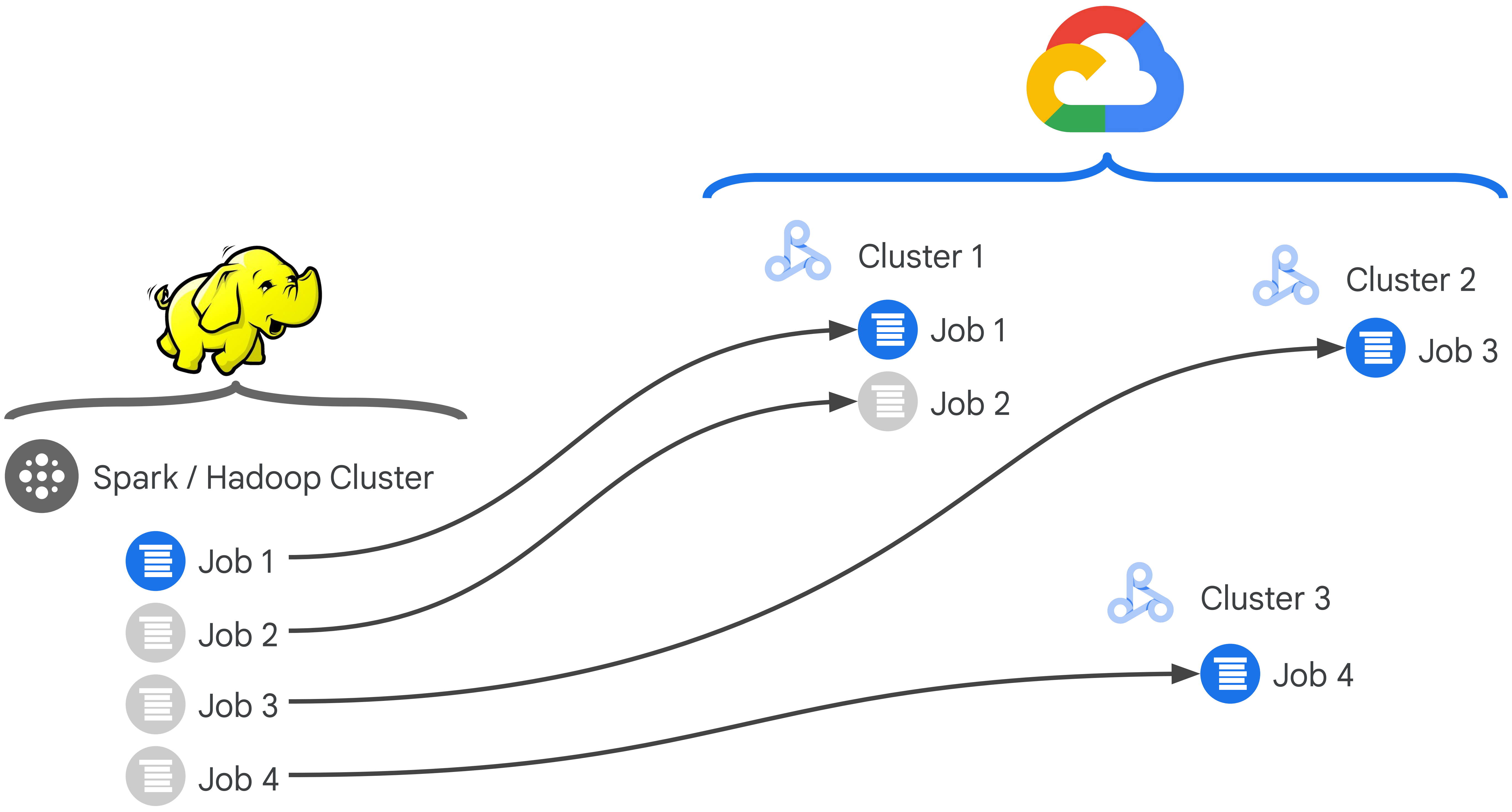
Resources are active at all times. You are constantly paying for all available clusters.

Ephemeral clusters

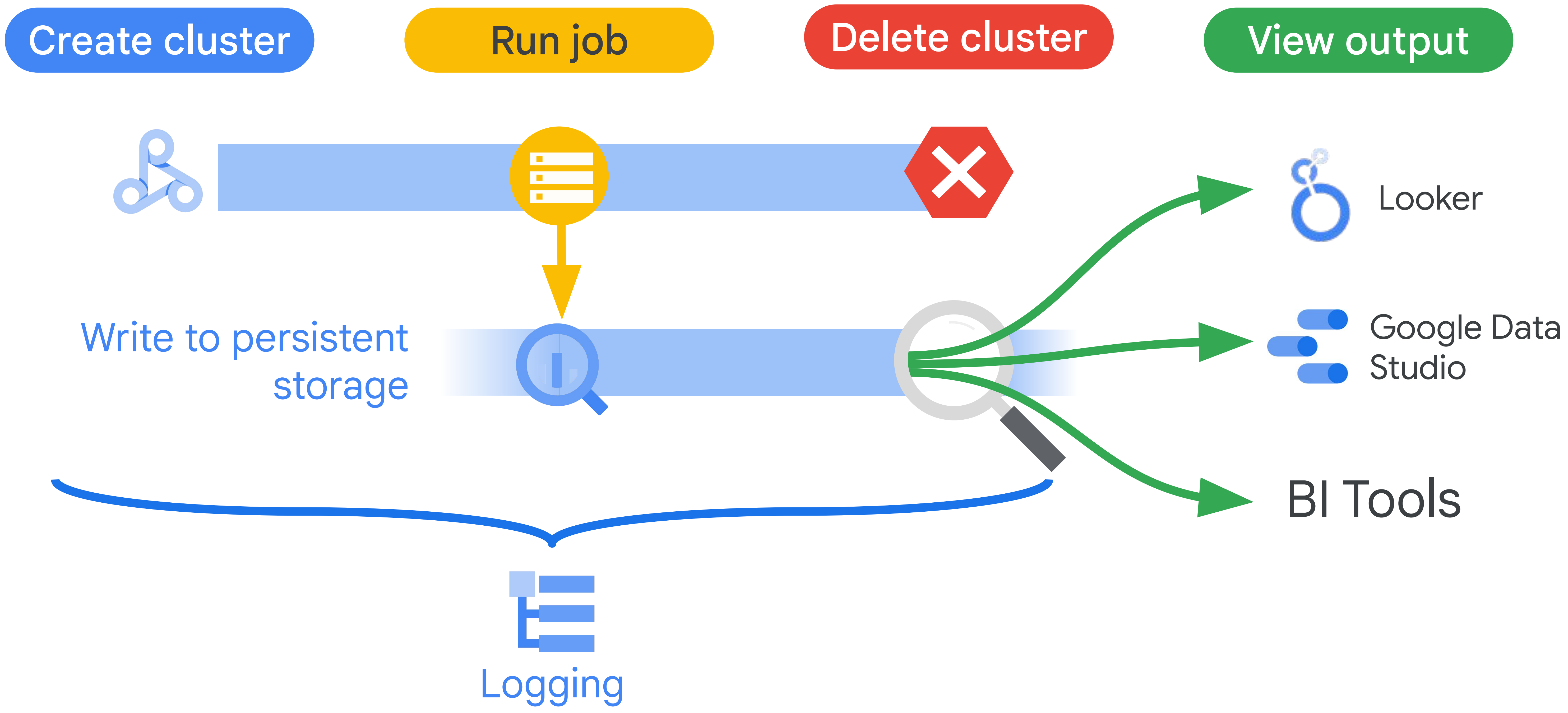


Required resources are active only when being used. You only pay for what you use.

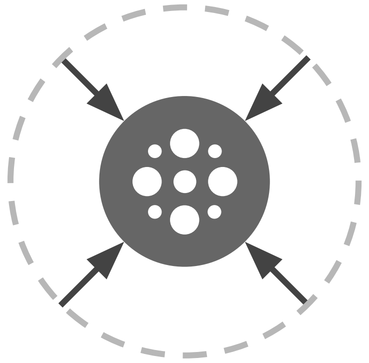
Split clusters and jobs



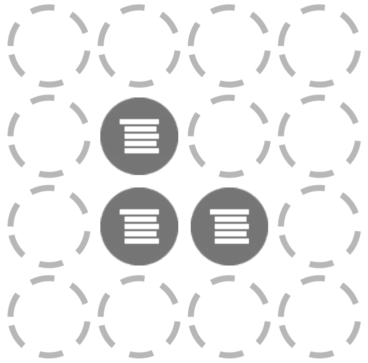
Use ephemeral clusters for one job's lifetime



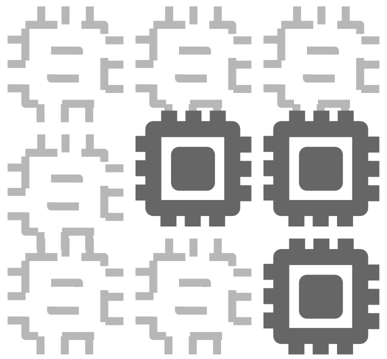
Points to remember if you need a persistent cluster



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

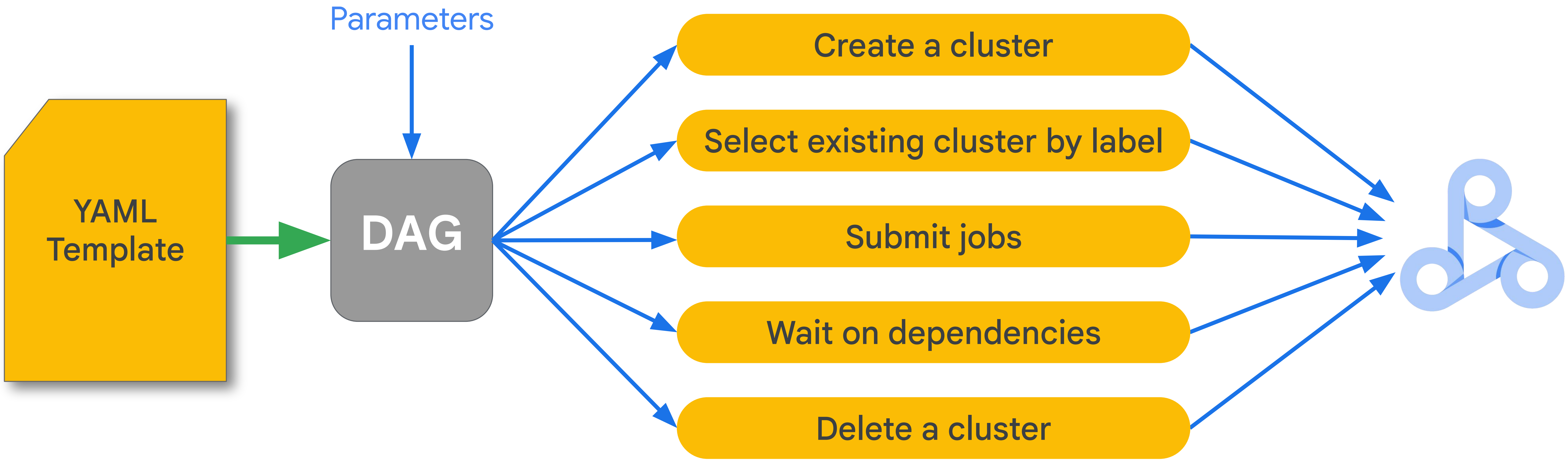


Scope your work on a persistent cluster to the smallest possible number of jobs.



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

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
```

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
```

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
```

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
```

Dataproc workflow templates

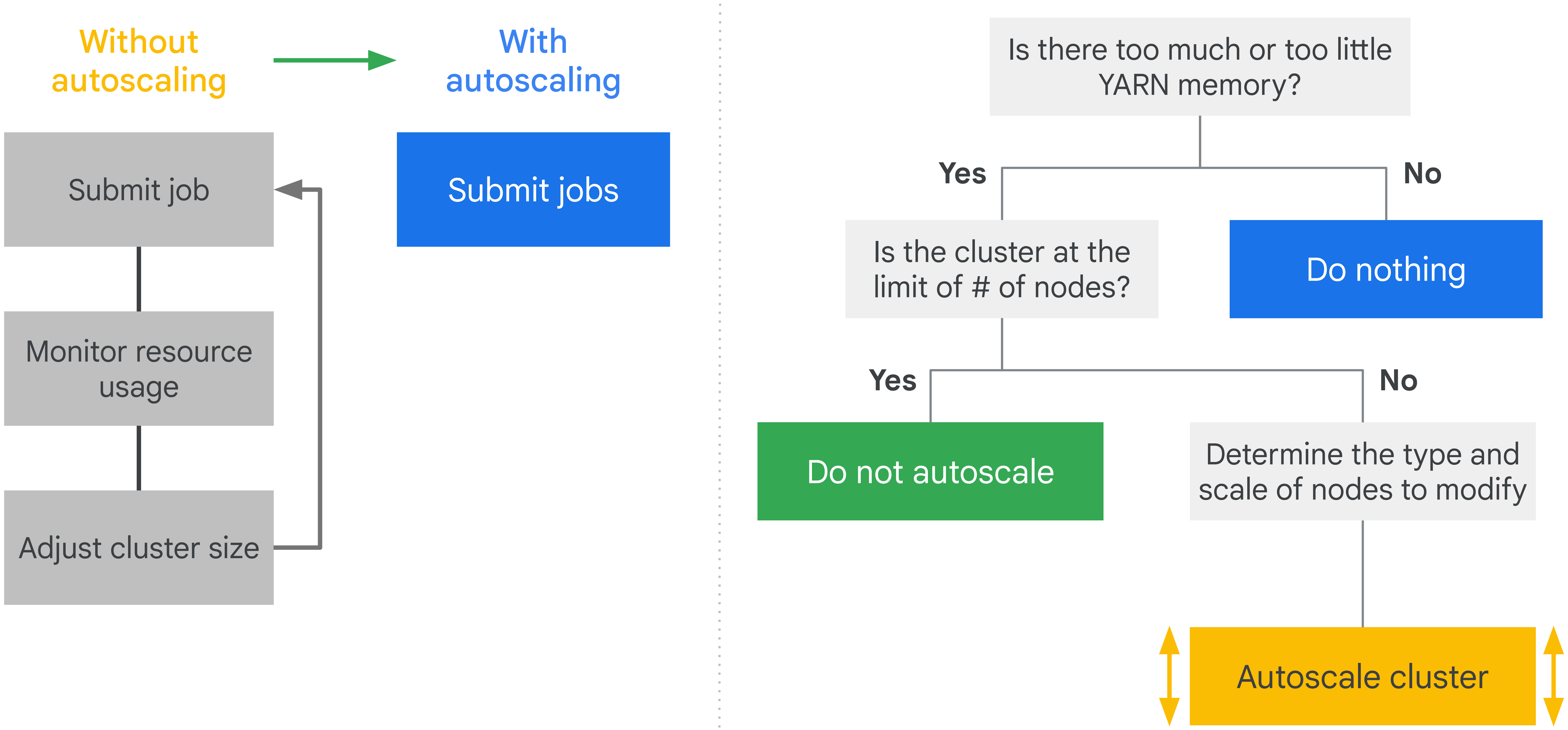
```
# the things we need pip-installed on the cluster
STARTUP_SCRIPT=gs://${BUCKET}/sparktobq/startup_script.sh
echo "pip install --upgrade --quiet google-compute-engine google-cloud-storage matplotlib" >
/tmp/startup_script.sh
gsutil cp /tmp/startup_script.sh $STARTUP_SCRIPT

# 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
```

Dataproc autoscaling workflow



Autoscaling improvements

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

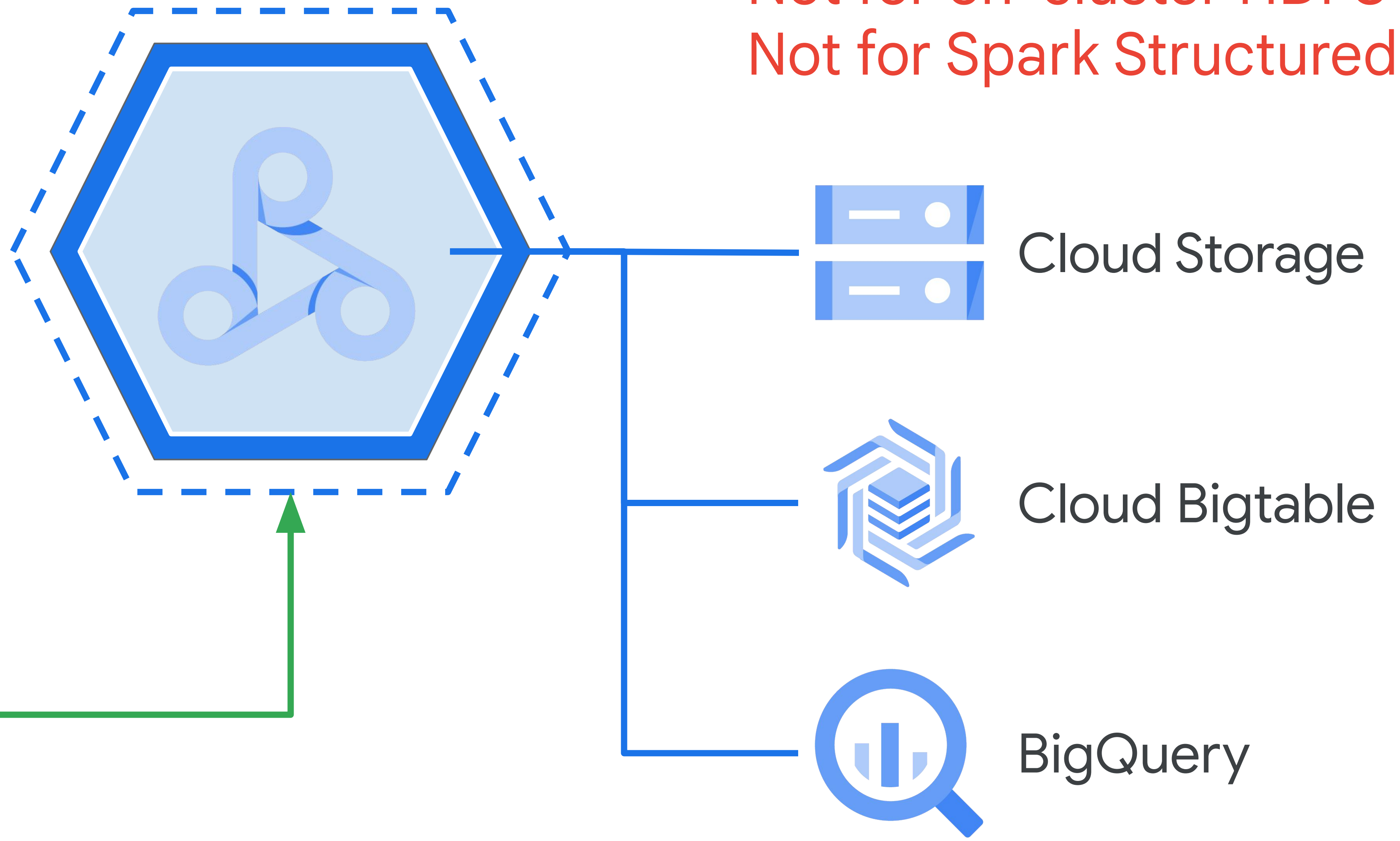
Dataproc autoscaling provides flexible capacity

Cluster with lots of jobs or a single large job

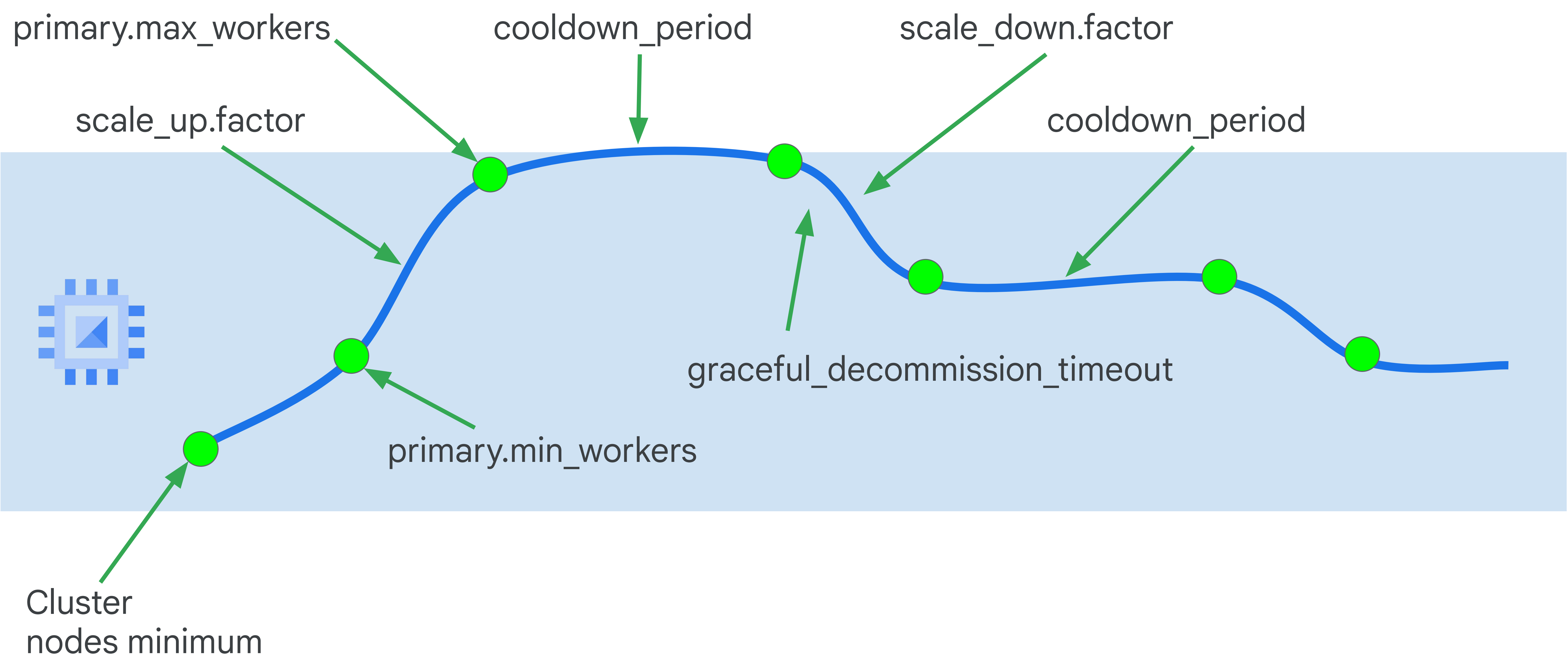
Not for idle clusters or scale to zero

Autoscaling is based on Hadoop YARN Metrics

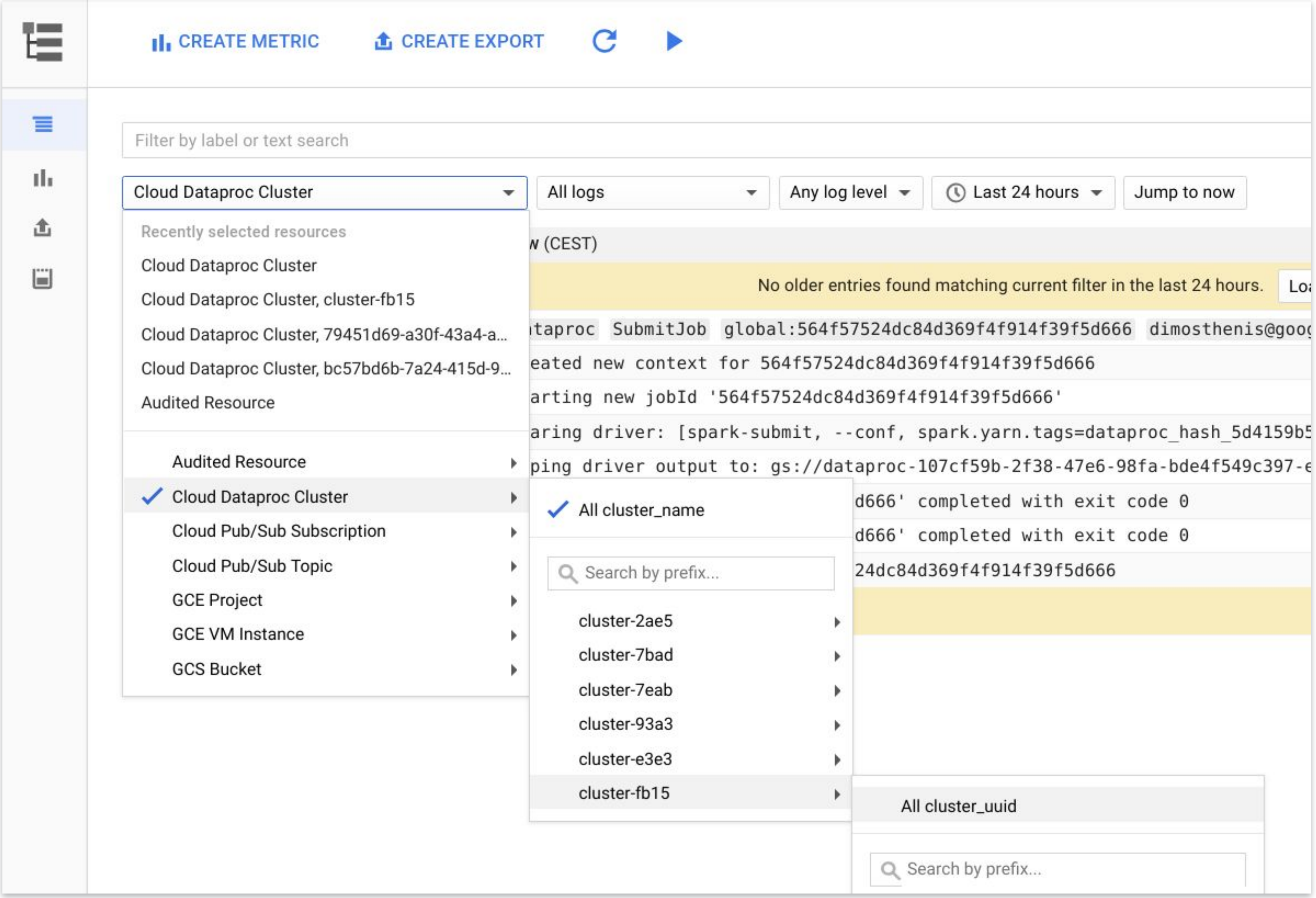
External data
Not for on-cluster HDFS
Not for Spark Structured Streaming



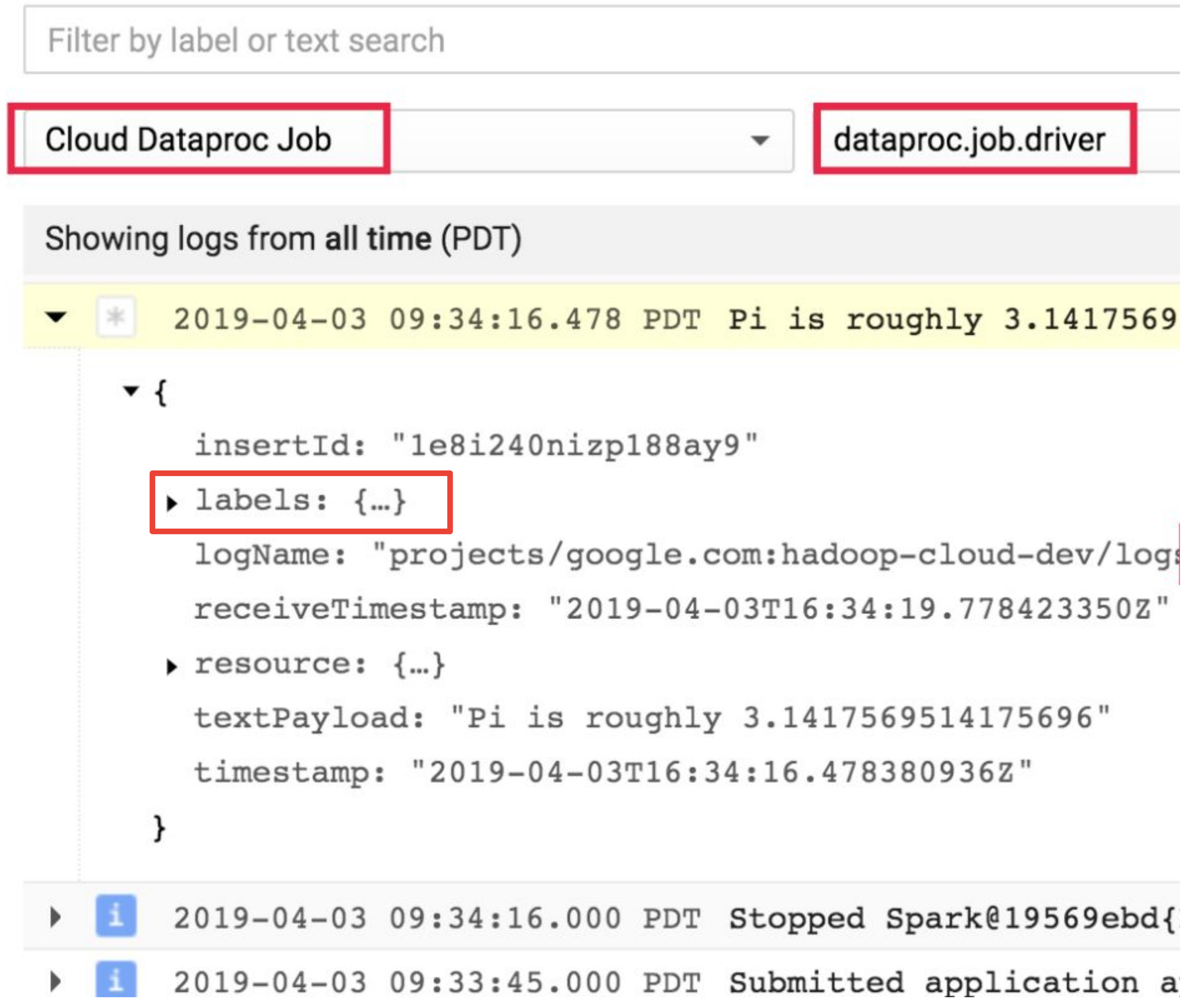
How Dataproc autoscaling works



Use Cloud Operations logging and performance monitoring



Create labels on clusters and jobs to find logs faster



Set the log level

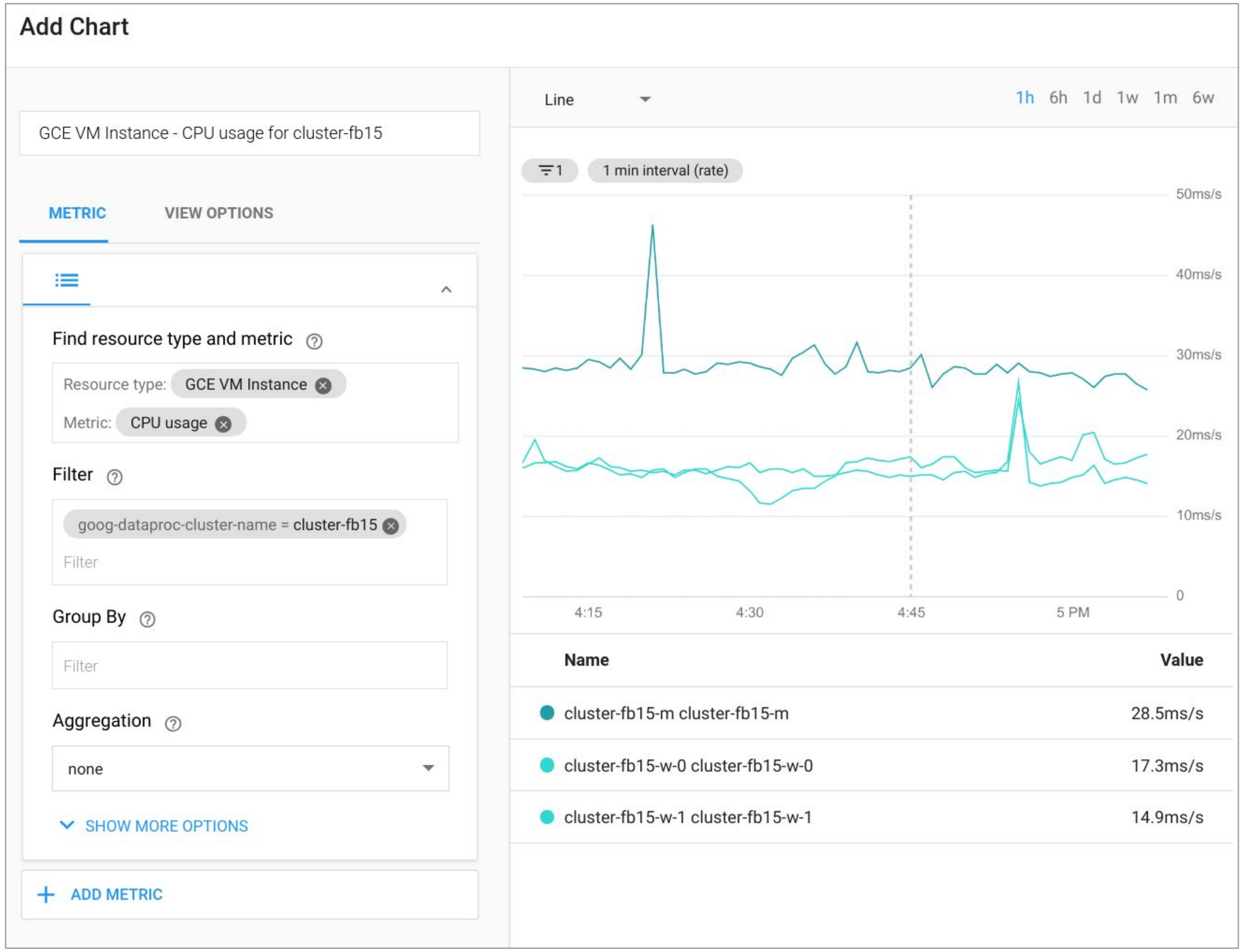
You can set the driver log level using the following gcloud command:

```
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")
```

Monitor your jobs



Lab Intro

Running Apache Spark jobs
on Dataproc

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

Summary

The Hadoop ecosystem

Running Hadoop on Dataproc

Cloud Storage instead of HDFS

Optimizing Dataproc