## **Dataproc**

This exam was heavy on on-prem Hadoop implementations and how to migrate to GCP.

* Migrate jobs to the cloud.
* Which role needs the service account to work properly with Dataproc (Dataproc Worker).
* SOCKS and YARN for web Interface.
* Custom images.
* Use Storage instead of HDFS.
* Always remember that Google recommends one cluster for one task. If you need analytics and transactional solutions with Dataproc, it is better to create two clusters for that kind of implementation.

Cloud Dataproc is a managed Apache Spark and Apache Hadoop service that lets you use open source data tools for batch processing, querying, streaming, and machine learning.  
Reference: <https://cloud.google.com/dataproc/docs/>

It is recommended to use Dataproc to run Apache Spark & Hadoop clusters When you want to move Hadoop & Spark workloads from an on-premises environment to Google Cloud Platform (GCP).  
Local HDFS storage is a good option if you have workloads that involve heavy I/O. For example, you have a lot of partitioned writes. It is a good option if you also have I/O workloads that are especially sensitive to latency. For example, you require single-digit millisecond latency per storage operation.

You can create a Cloud Dataproc cluster with internal IP addresses only. However, attempts to access the Internet in an initialization action will fail unless you have configured routes to direct the traffic through a NAT or a VPN gateway. Without having access to the Internet, you can enable Private Google Access, and place job dependencies in Cloud Storage; cluster nodes can download the dependencies from Cloud Storage from internal IPs.

Cloud Storage with Dataproc for long-term storage that does not require a Dataproc cluster to be persistent and constantly running. Modifying the data storage would require merely repointing the source and destination location in existing scripts.

<https://cloud.google.com/dataproc/docs/concepts/configuring-clusters/flex>

Dataproc has a BigQuery connector library which allows it directly to interface with BigQuery.

You can use a BigQuery connector to enable programmatic read/write access to BigQuery. This is an ideal way to process data that is stored in BigQuery. No command-line access is exposed. The BigQuery connector is a Java library that enables Hadoop to process data from BigQuery using abstracted versions of the Apache Hadoop InputFormat and OutputFormat classes.

A Cloud Dataproc cluster has the Spark components, including Spark ML, installed. Dataproc reads data directly from the BigQuery using Sparc connector.

Refer: <https://cloud.google.com/dataproc/docs/tutorials/bigquery-sparkml>

the internal services can be accessed using the SOCKS proxy server.  
Refer GCP documentation – Dataproc – Connecting to web interfaces:- <https://cloud.google.com/dataproc/docs/concepts/accessing/cluster-web-interfaces>

You can connect to web interfaces running on a Cloud Dataproc cluster using your project’s Cloud Shell or the Cloud SDK gcloud command-line tool:

Cloud Shell: The Cloud Shell in the Google Cloud Platform Console has the Cloud SDK commands and utilities pre-installed, and it provides a Web Preview feature that allows you to quickly connect through an SSH tunnel to a web interface port on a cluster. However, a connection to the cluster from Cloud Shell uses local port forwarding, which opens a connection to only one port on a cluster web interface—multiple commands are needed to connect to multiple ports. Also, Cloud Shell sessions automatically terminate after a period of inactivity (30 minutes).

gcloud command-line tool: The gcloud compute ssh command with dynamic port forwarding allows you to establish an SSH tunnel and run a SOCKS proxy server on top of the tunnel. After issuing this command, you must configure your local browser to use the SOCKS proxy. This connection method allows you to connect to multiple ports on a cluster web interface.

Storing persistent data off the cluster allows the cluster to be shut down when not processing data. And it allows separate clusters to be started per job or per kind of work, so tuning is less important.  
Refer GCP documentation – Dataproc Cloud Storage: - <https://cloud.google.com/dataproc/docs/concepts/connectors/cloud-storage>

Direct data access – Store your data in Cloud Storage and access it directly, with no need to transfer it into HDFS first.  
HDFS compatibility – You can easily access your data in Cloud Storage using the gs:// prefix instead of hdfs://.  
Interoperability – Storing data in Cloud Storage enables seamless interoperability between Spark, Hadoop, and Google services.  
Data accessibility – When you shut down a Hadoop cluster, you still have access to your data in Cloud Storage, unlike HDFS.  
High data availability – Data stored in Cloud Storage is highly available and globally replicated without a loss of performance.  
No storage management overhead – Unlike HDFS, Cloud Storage requires no routine maintenance such as checking the file system, upgrading, or rolling back to a previous version of the file system, etc.  
Quick startup – In HDFS, a MapReduce job can’t start until the NameNode is out of safe mode—a process that can take from a few seconds to many minutes depending on the size and state of your data. With Cloud Storage, you can start your job as soon as the task nodes start, leading to significant cost savings over time.

Spark can improve the performance as it performs lazy in-memory execution.

Spark is important because it does part of its pipeline processing in memory rather than copying from disk. For some applications, this makes Spark extremely fast. With a Spark pipeline, you have two different kinds of operations, transforms and actions. Spark builds its pipeline using an abstraction called a directed graph. Each transform builds additional nodes into the graph, but spark does not execute the pipeline until it sees an action.

Spark waits until it has the whole story, all the information. This allows Spark to choose the best way to distribute the work and run the pipeline. The process of waiting on transforms and executing on actions is called lazy execution. For a transformation, the input is an RDD, and the output is an RDD. When Spark sees a transformation, it registers it in the directed graph and then it waits. An action triggers Spark to process the pipeline, the output is usually a result format, such as a text file, rather than an RDD.

Pig is wrapper and would initiate Map Reduce jobs

Dataproc Preemptible VMs:- <https://cloud.google.com/dataproc/docs/concepts/compute/preemptible-vms>

Preemptible instances are disposable and should not be used to store data. Dataproc cluster cannot be with only preemptible instances. It needs to have two non-preemptible worker nodes.

preemptible nodes can have persistent disks.

Dataproc handles the addition and removal of preemptible nodes.

Cloud Dataproc provides out-of-the box and end-to-end support for many of the most popular job types, including Spark, Spark SQL, PySpark, MapReduce, Hive, and Pig jobs.  
Reference: <https://cloud.google.com/dataproc/docs/resources/faq#what_type_of_jobs_can_i_run>

Service accounts authenticate applications running on your virtual machine instances to other Google Cloud Platform services. For example, if you write an application that reads and writes files on Google Cloud Storage, it must first authenticate to the Google Cloud Storage API. At a minimum, service accounts used with Cloud Dataproc need permissions to read and write to Google Cloud Storage, and to write to Google Cloud Logging.  
Reference: <https://cloud.google.com/dataproc/docs/concepts/service-accounts#important_notes>

Service accounts used with Cloud Dataproc must have Dataproc/Dataproc Worker role (or have all the permissions granted by Dataproc Worker role).  
Reference: <https://cloud.google.com/dataproc/docs/concepts/service-accounts#important_notes>

A Cloud Dataproc Viewer is limited in its actions based on its role. A viewer can only list clusters, get cluster details, list jobs, get job details, list operations, and get operation details.  
Reference: <https://cloud.google.com/dataproc/docs/concepts/iam#iam_roles_and_cloud_dataproc_operations_summary>

At a minimum, you must specify four values when creating a new cluster with the projects.regions.clusters.create operation:  
The project in which the cluster will be created  
The region to use –  
The name of the cluster –  
The zone in which the cluster will be created  
You can specify many more details beyond these minimum requirements. For example, you can also specify the number of workers, whether preemptible compute should be used, and the network settings.  
Reference: <https://cloud.google.com/dataproc/docs/tutorials/python-library-example#create_a_new_cloud_dataproc_cluste>

The following rules will apply when you use preemptible workers with a Cloud Dataproc cluster: Processing onlySince preemptibles can be reclaimed at any time, preemptible workers do not store data. Preemptibles added to a Cloud Dataproc cluster only function as processing nodes.  
. No preemptible-only clustersTo ensure clusters do not lose all workers, Cloud Dataproc cannot create preemptible-only clusters.  
. Persistent disk sizeAs a default, all preemptible workers are created with the smaller of 100GB or the primary worker boot disk size. This disk space is used for local caching of data and is not available through HDFS.  
The managed group automatically re-adds workers lost due to reclamation as capacity permits.  
Reference: <https://cloud.google.com/dataproc/docs/concepts/preemptible-vms>

When using Cloud Dataproc clusters, configure your browser to use the SOCKS proxy. The SOCKS proxy routes data intended for the Cloud Dataproc cluster through an SSH tunnel.  
Reference: <https://cloud.google.com/dataproc/docs/concepts/cluster-web-interfaces#interfaces>

To make updating files and properties easy, the –properties command uses a special format to specify the configuration file and the property and value within the file that should be updated. The formatting is as follows: file\_prefix:property=value.  
Reference: <https://cloud.google.com/dataproc/docs/concepts/cluster-properties#formatting>

Scaling a Cloud Dataproc cluster typically involves

After creating a Cloud Dataproc cluster, you can scale the cluster by increasing or decreasing the number of worker nodes in the cluster at any time, even when jobs are running on the cluster. Cloud Dataproc clusters are typically scaled to:  
1) increase the number of workers to make a job run faster  
2) decrease the number of workers saving money  
3) increase the number of nodes to expand available Hadoop Distributed Filesystem (HDFS) storage  
Reference: <https://cloud.google.com/dataproc/docs/concepts/scaling-clusters>

One of the advantages of Cloud Dataproc is its low cost. Dataproc charges for what you really use with **minute-by-minute billing** and a low, ten-minute-minimum billing period.  
Reference: <https://cloud.google.com/dataproc/docs/concepts/overview>

The YARN ResourceManager and the HDFS NameNode interfaces are available on a Cloud Dataproc cluster master node. The cluster master-host-name is the name of your Cloud Dataproc cluster followed by an -m suffixfor example, if your cluster is named “my-cluster,” the master-host-name would be “my-cluster-m”.  
Reference: <https://cloud.google.com/dataproc/docs/concepts/cluster-web-interfaces#interfaces>

You can access the master node of the cluster by clicking the SSH button next to it in the Cloud Console.  
You can easily use the –properties option of the dataproc command in the Google Cloud SDK to modify many common configuration files when creating a cluster.  
When creating a Cloud Dataproc cluster, you can specify initialization actions in executables and/or scripts that Cloud Dataproc will run on all nodes in your Cloud Dataproc cluster immediately after the cluster is set up. [https://cloud.google.com/dataproc/docs/concepts/configuring-clusters/init-actions]  
Reference: <https://cloud.google.com/dataproc/docs/concepts/configuring-clusters/cluster-properties>

To securely transfer web traffic data from your computer’s web browser to the Cloud Dataproc cluster you should use a(n) \_\_\_\_\_.

To connect to the web interfaces, it is recommended to use an SSH tunnel to create a secure connection to the master node.  
Reference: <https://cloud.google.com/dataproc/docs/concepts/cluster-web-interfaces#connecting_to_the_web_interfaces>

non-critical experiments, there is no need for high-CPU worker machine types.

BigQuery Connector for dataproc:You can use a BigQuery connector to enable programmatic read/write access to BigQuery. This is an ideal way to process data that is stored in BigQuery. No command-line access is exposed. The BigQuery connector is a Java library that enables Hadoop to process data from BigQuery using abstracted versions of the Apache Hadoop InputFormat and OutputFormat classes.  
You can access BigQuery from Dataproc by installing BigQuery connector to the Dataproc cluster using initialization actions. When a Dataproc spark job reads from BigQuery, it writes the BigQuery table’s content temporarily to Google Storage using Dataproc cluster’s assigned bucket. If the job is completed successfully, temporary files are automatically deleted from the cluster. If the job fails, you need to delete temp files manually.

<https://cloud.google.com/dataproc/docs/concepts/connectors/bigquery>

<https://cloud.google.com/dataproc/docs/concepts/connectors/bigquery><https://cloud.google.com/dataproc/docs/concepts/configuring-clusters/init-actions>

**Migration**:

When you want to move Hadoop & Spark workloads from an on-premises environment to Google Cloud Platform (GCP), It’s recommended to use Dataproc to run Apache Spark & Hadoop clusters.  
Cloud Storage is a good option if:  
Your data in ORC, Parquet, Avro, or any other format will be used by different clusters or jobs, and you need data persistence if the cluster terminates.  
You need high throughput, and your data is stored in files larger than 128 MB.  
You need cross-zone durability for your data.  
You need data to be highly available—for example, you want to eliminate HDFS NameNode as a single point of failure.

[https://cloud.google.com/solutions/migration/hadoop/ migrating-apache-spark-jobs-to-cloud-dataproc](https://cloud.google.com/solutions/migration/hadoop/%C2%A0migrating-apache-spark-jobs-to-cloud-dataproc)