# How Cloud Shell works



When you start Cloud Shell, it provisions a **Compute Engine virtual machine running** a Debian-based Linux operating system. Cloud Shell instances are provisioned on a per-user, per-session basis. The instance persists while your Cloud Shell session is active; after an hour of inactivity, your session terminates and its VM is discarded. For more on usage quotas, refer to the [limitations](https://cloud.google.com/shell/docs/limitations#usage_limits) guide.

With the default Cloud Shell experience, you are allocated with an ephemeral, pre-configured VM and the environment you work with is a Docker container running on that VM. You can also [customize your environment](https://cloud.google.com/shell/docs/custom-environments) automatically on VM boot to ensure that your Cloud Shell instance includes your preferred tools.

# GKE overview



Google Kubernetes Engine (GKE) provides a managed environment for deploying, managing, and scaling your containerized applications using Google infrastructure. The GKE environment consists of multiple machines (specifically, [Compute Engine](https://cloud.google.com/compute) instances) grouped together to form a [cluster](https://cloud.google.com/kubernetes-engine/docs/concepts/cluster-architecture).

## Cluster orchestration with GKE

GKE clusters are powered by the [Kubernetes](https://kubernetes.io/) open source cluster management system. Kubernetes provides the mechanisms through which you interact with your cluster. You use Kubernetes commands and resources to deploy and manage your applications, perform administration tasks, set policies, and monitor the health of your deployed workloads.

Kubernetes draws on the same design principles that run popular Google services and provides the same benefits: automatic management, monitoring and liveness probes for application containers, automatic scaling, rolling updates, and more. When you run your applications on a cluster, you're using technology based on Google's 10+ years of experience running production workloads in containers.

### Kubernetes on Google Cloud

When you run a GKE cluster, you also gain the benefit of advanced cluster management features that Google Cloud provides. These include:

* Google Cloud's [load-balancing](https://cloud.google.com/compute/docs/load-balancing-and-autoscaling) for Compute Engine instances
* [Node pools](https://cloud.google.com/kubernetes-engine/docs/concepts/node-pools) to designate subsets of nodes within a cluster for additional flexibility
* [Automatic scaling](https://cloud.google.com/kubernetes-engine/docs/cluster-autoscaler) of your cluster's node instance count
* [Automatic upgrades](https://cloud.google.com/kubernetes-engine/docs/concepts/node-auto-upgrades) for your cluster's node software
* [Node auto-repair](https://cloud.google.com/kubernetes-engine/docs/concepts/node-auto-repair) to maintain node health and availability
* [Logging and monitoring](https://cloud.google.com/monitoring/kubernetes-engine) with Google Cloud's operations suite for visibility into your cluster

## GKE workloads

GKE works with containerized applications. These are applications packaged into platform independent, isolated user-space instances, for example by using [Docker](https://www.docker.com/). In GKE and Kubernetes, these containers, whether for applications or batch jobs, are collectively called workloads. Before you deploy a workload on a GKE cluster, you must first package the workload into a container.

Google Cloud provides continuous integration and continuous delivery tools to help you build and serve application containers. You can use [Cloud Build](https://cloud.google.com/build) to build container images (such as Docker) from a variety of source code repositories, and [Artifact Registry](https://cloud.google.com/artifact-registry) or [Container Registry](https://cloud.google.com/container-registry) to store and serve your container images.

## Modes of operation

The level of flexibility, responsibility, and control that you require for your clusters determines the mode of operation to use in GKE. GKE clusters have two modes of operation to choose from:

* **Autopilot**: Manages the entire cluster and node infrastructure for you. Autopilot provides a hands-off Kubernetes experience so that you can focus on your workloads and only pay for the resources required to run your applications. Autopilot clusters are pre-configured with an optimized cluster configuration that is ready for production workloads.
* **Standard**: Provides you with node configuration flexibility and full control over managing your clusters and node infrastructure. For clusters created using the Standard mode, you determine the configurations needed for your production workloads, and you pay for the nodes that you use.

For more information about these modes, and to learn more about Autopilot, see the [Autopilot overview](https://cloud.google.com/kubernetes-engine/docs/concepts/autopilot-overview).

# Node images



This page describes the node images available for Google Kubernetes Engine (GKE) nodes.

When you create a GKE cluster or node pool, you can choose the operating system image that runs on each node. You can also upgrade an existing cluster to use a different node image type. For instructions on how to set the node image, see [Specifying a node image](https://cloud.google.com/kubernetes-engine/docs/how-to/node-images).

<https://cloud.google.com/kubernetes-engine/docs/concepts/node-images>

Run a Spark job on Dataproc on Google Kubernetes Engine.

[Kubernetes](https://kubernetes.io/) (K8s), the open source container orchestration platform,

released of Cloud Dataproc for Kubernetes (K8s Dataproc), allowing Spark to run directly on [Google Kubernetes Engine](https://cloud.google.com/kubernetes-engine/) (GKE)-based K8s clusters. The service promises to reduce complexity, in terms of open source data components' inter-dependencies, and portability of Spark applications. That should allow data engineers, analytics experts and data scientists to run their Spark workloads in a streamlined way, with less integration and versioning hassles.

While the DIY approach of deploying Spark to your own K8s cluster is good, it's essentially an IaaS (Infrastructure as a Service) approach. As such, it requires a K8s skill set and puts the customer in charge of everything, including software deployment and cluster maintenance. K8s Dataproc is better because it offers Dataproc's service level agreement (SLA), Google Cloud Platform-optimized open source components and -- via the Dataproc API -- abstraction of the K8s details and skill set requirements, supplying integrated management and security.

**Google Dataproc uses image versions to bundle operating system, big data components, and Google Cloud Platform connectors into one package that is deployed on a cluster.**

### Debian images

The following **Debian 10**-based image versions are supported in Dataproc clusters. Note that new clusters will be created to include any sub-minor patches that have been made to a version since its release.

### Ubuntu images

The following **Ubuntu 18.04 LTS**-based image versions are supported in Dataproc clusters. Note that new clusters will be created to include any sub-minor patches that have been made to a version since its release

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

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

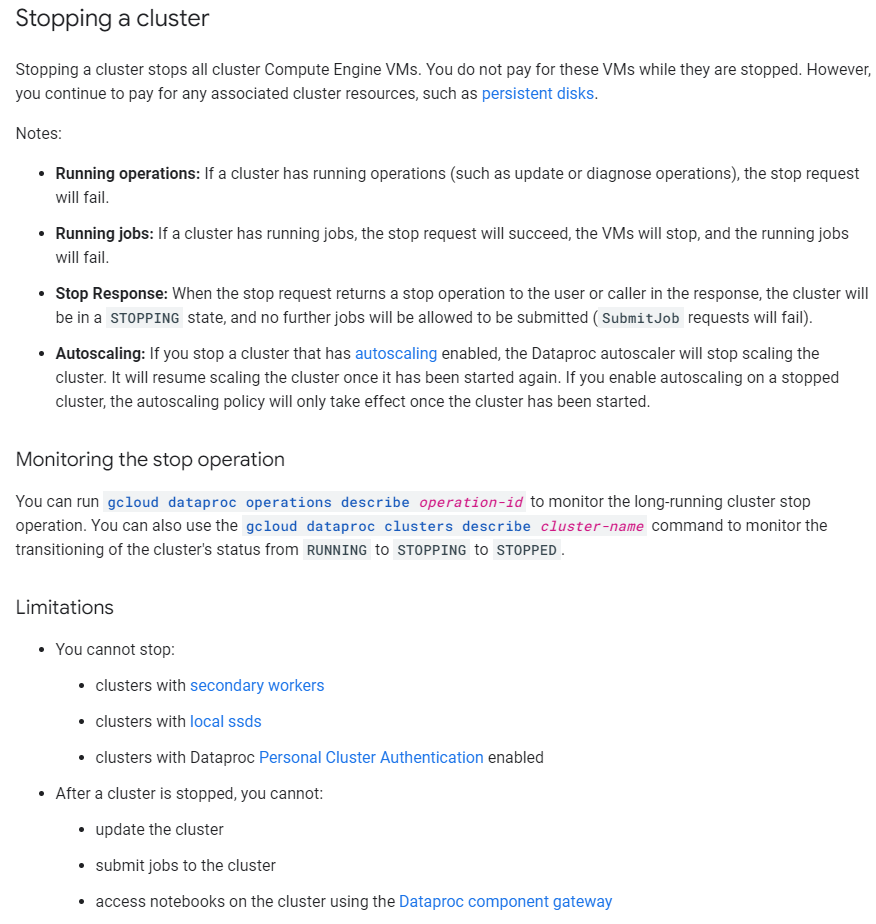
# Starting and stopping clusters



After you create a cluster, **you can stop it, then restart it** when you need it. Stopping an idle cluster avoids incurring charges and avoids the need to delete an idle cluster, then create a cluster with the same configuration later.

**Fetaure Notes:**

* The cluster start/stop feature is only supported with the following Dataproc image versions or above:
  + 1.4.35-debian10/ubuntu18
  + 1.5.10-debian10/ubuntu18
  + 2.0.0-RC6-debian10/ubuntu18
* Stopping individual cluster nodes is not recommended since the status of a stopped VM may not be in sync with cluster status, which can result in errors.



**Starting a cluster**

* When you start a stopped cluster, any initialization actions will not be re-run. Initialization actions are only run on cluster nodes when the cluster is created and when nodes are added when the cluster is scaled up.
* After the start operation completes, you can immediately submit jobs to the cluster. However, execution of these jobs can be delayed (approximately 30 seconds) to allow HDFS and YARN to become operational.

**Using Stop/Start**

You can stop and start a cluster using the gcloud CLI or the Dataproc API.

[gcloud command](https://cloud.google.com/dataproc/docs/guides/dataproc-start-stop#gcloud-command)[REST API](https://cloud.google.com/dataproc/docs/guides/dataproc-start-stop#rest-api)[Console](https://cloud.google.com/dataproc/docs/guides/dataproc-start-stop#console)

**Stop a cluster**

gcloud dataproc clusters stop *cluster-name* \

    --region=*region*

**Start a cluster**

gcloud dataproc clusters start *cluster-name* \

    --region=*region*

# Submit a job

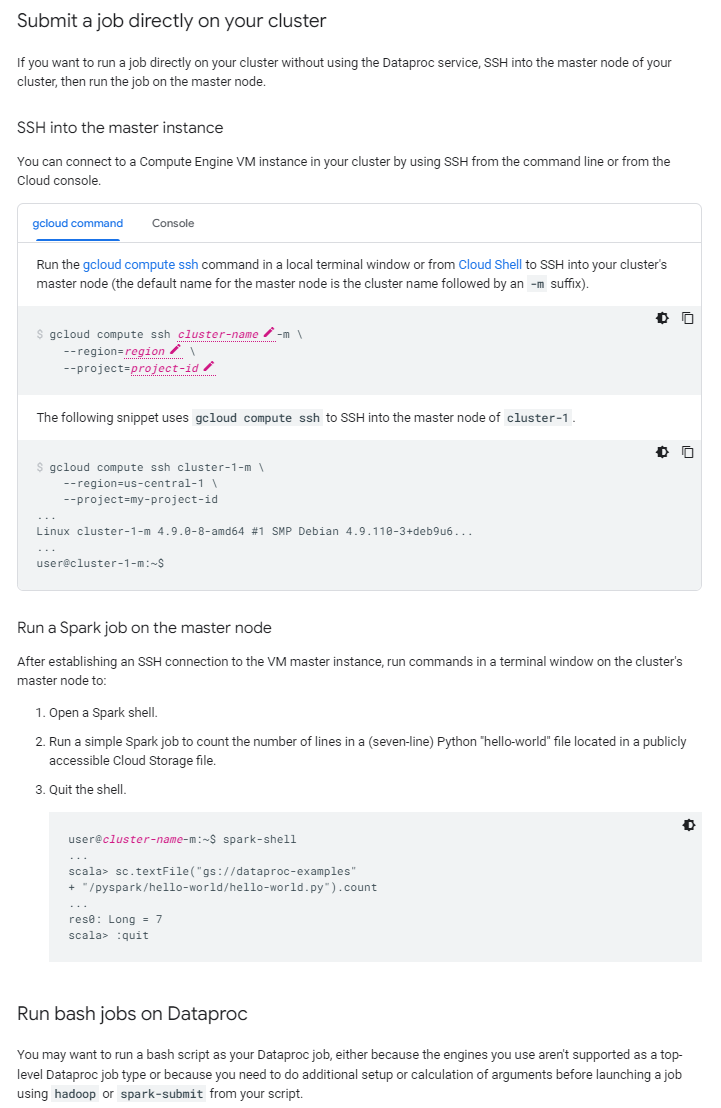


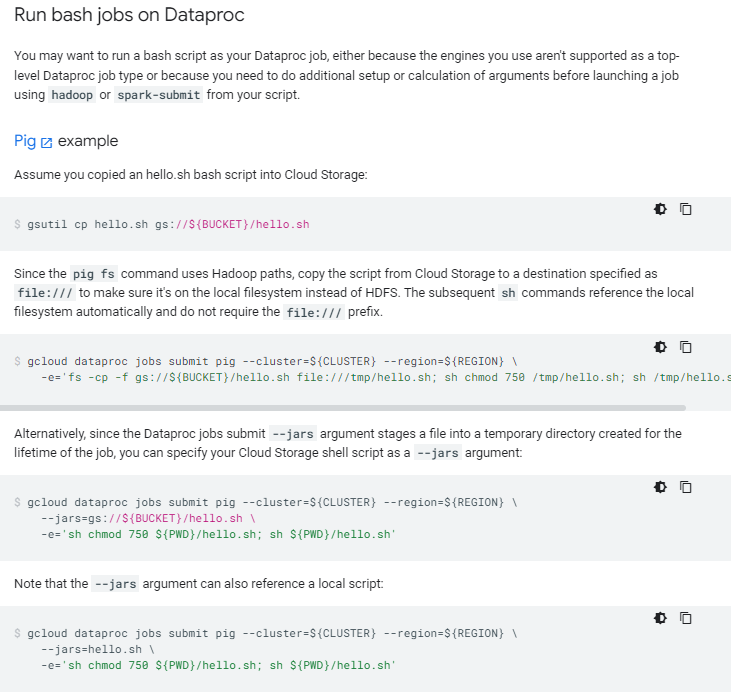
You can submit a job to an existing Dataproc cluster via a Dataproc API [jobs.submit](https://cloud.google.com/dataproc/docs/reference/rest/v1/projects.regions.jobs/submit) HTTP or programmatic request, using the Google Cloud CLI [gcloud](https://cloud.google.com/sdk/gcloud/reference/dataproc/jobs/submit) command-line tool in a local terminal window or in [Cloud Shell](https://console.cloud.google.com/?cloudshell=true), or from the [Google Cloud console](https://console.cloud.google.com/dataproc/jobs/jobsSubmit) opened in a local browser. You can also [SSH into the master instance](https://cloud.google.com/dataproc/docs/guides/submit-job#ssh_into_the_master_instance) in your cluster, and then run a job directly from the instance without using the Dataproc service.

<https://cloud.google.com/dataproc/docs/reference/libraries#installing_the_client_library>

<https://cloud.google.com/dataproc/docs/guides/submit-job>

import re  
  
  
from google.cloud import dataproc\_v1 as dataproc  
from google.cloud import storage  
  
  
def submit\_job(project\_id, region, cluster\_name):  
    # Create the job client.  
    job\_client = dataproc.JobControllerClient(  
        client\_options={"api\_endpoint": "{}-dataproc.googleapis.com:443".format(region)}  
    )  
  
    # Create the job config. 'main\_jar\_file\_uri' can also be a  
    # Google Cloud Storage URL.  
    job = {  
        "placement": {"cluster\_name": cluster\_name},  
        "spark\_job": {  
            "main\_class": "org.apache.spark.examples.SparkPi",  
            "jar\_file\_uris": ["file:///usr/lib/spark/examples/jars/spark-examples.jar"],  
            "args": ["1000"],  
        },  
    }  
  
    operation = job\_client.submit\_job\_as\_operation(  
        request={"project\_id": project\_id, "region": region, "job": job}  
    )  
    response = operation.result()  
  
    # Dataproc job output gets saved to the Google Cloud Storage bucket  
    # allocated to the job. Use a regex to obtain the bucket and blob info.  
    matches = re.match("gs://(.\*?)/(.\*)", response.driver\_output\_resource\_uri)  
  
    output = (  
        storage.Client()  
        .get\_bucket(matches.group(1))  
        .blob(f"{matches.group(2)}.000000000")  
        .download\_as\_string()  
    )  
  
    print(f"Job finished successfully: {output}")





# Dataproc on GKE overview

Dataproc on GKE allows you to execute Big Data applications using the Dataproc jobs API on GKE clusters. Use the Google Cloud console, Google Cloud CLI or the Dataproc API (HTTP request or Cloud Client Libraries) to [create a Dataproc on GKE virtual cluster](https://cloud.google.com/dataproc/docs/guides/dpgke/quickstarts/dataproc-gke-quickstart-create-cluster), then submit a Spark, PySpark, SparkR, or Spark-SQL job to the Dataproc service.

Dataproc on GKE supports [Spark 2.4 and Spark 3.1 versions](https://cloud.google.com/dataproc/docs/guides/dpgke/dataproc-gke-versions).

## How Dataproc on GKE works

Dataproc on GKE deploys Dataproc**virtual clusters on a GKE cluster**. Unlike legacy [Dataproc on Compute Engine clusters](https://cloud.google.com/dataproc/docs/guides/create-cluster),

Dataproc on GKE virtual clusters do not include separate master and worker VMs. Instead,

when you create a Dataproc on GKE virtual cluster, Dataproc on GKE creates **node pools** within a GKE cluster.

Dataproc on GKE jobs are run as **pods** on these **node pools**. The node pools and scheduling of pods on the node pools are managed by GKE.

**google Dataproc on GKE vs Dataproc on Compute Engine clusters**

Our organisation has recently moved its infrastructure from aws to google cloud compute and I figured dataproc clusters are a good solution to running our existing spark jobs .

But when it comes to comparing the pricing , I also realised that I can just fire up a **google kubernetes engine cluster and install spark in it to run spark applications on it .**

Now my question is , how do “running spark on gke “ and using dataproc compare ? Which one would be the best option in terms of autoscaling , pricing and infrastructure . I’ve read googles documentation on gke and dataproc but there isn’t enough for to be sure in terms of advantages and disadvantages of using GKE or dataproc over the other .

Spark on DataProc is proven and it's in use at many organizations, though its not fully managed, you can automate cluster creation and tear down, submitting jobs etc through GCP api, but still it's another stack you have to manage.

Spark on GKE is something new, Spark started adding features from 2.4 onwards to support Kubernetes, and even Google updated the Kubernetes for the preview couple of days back, [Link](https://www.datanami.com/2019/01/30/google-brings-kubernetes-operator-for-spark-to-gcp/)

I would just go with DataProc if I have to run Jobs in Prod environment as we speak otherwise you could just experiment yourself with Docker and see how it fares, but I think it needs little more time to be stable, from purely cost perspective it would be cheaper with Docker as you can share resources with your other services.

* I would favor DataProc, because its managed and supports Spark out of the box. **No hazzles**. More importantly, cost optimized. **You may not need clusters all the time**, you can have **ephemeral clusters** with dataproc.
* With GKE, I need to explicitly discard the cluster and recreate when necessary. Additional care needs to be taken care of.
* I could not come across any direct service offering from GCP on data lineage. In that case, I would probably use Apache Atlas with Spark-Atlas-Connector on Spark installation managed by myself. In that case, running Spark on GKE with all the control lying with myself would make a compelling choice.

**Running Spark on Kubernetes with Dataproc**

Today, we are announcing the general availability of [Dataproc on Google Kubernetes Engine](https://cloud.google.com/dataproc/docs/guides/dpgke/dataproc-gke-overview) (GKE), enabling you to leverage k8s to manage and optimize your compute platforms. You can now create a Dataproc cluster and submit Spark jobs on a self-managed GKE cluster.

 We are observing a trend towards building applications as containers, to simplify application management among the many other benefits such as, improved agility, security, portability.

Dataproc on GKE, now in GA, allows you to run Spark workloads on a self-managed GKE cluster

<https://cloud.google.com/blog/products/infrastructure-modernization/running-spark-on-kubernetes-with-dataproc>

### **Key Benefits**

Preview customers with expertise in GKE were able to easily integrate Dataproc into their environments and are now looking forward to migrating Spark workloads and optimizing their execution environments to improve efficiency and save costs. Our advanced customers are exploring GPUs for improved job performance to meet their stringent SLAs needs. As we go GA, these customers are excited about utilizing the advanced k8s compute management and resource sharing for their production workloads.

Running on GKE enables you to take advantage of the advanced capabilities of k8s enabling you optimize costs and performance by running:

* Completely independent jobs on the same cluster.
  + You can now share a Dataproc cluster among multiple applications with distinct libraries and dependencies. Each Job can run its own container. Allowing independent Jobs with conflicting dependencies to run at the same time on the same Cluster. Earlier, each job with a distinct environment required an exclusive cluster. Relaxing this constraint enables customers to further optimize their execution environment.
* Multiple clusters on the same node pool.
  + You can share the same infrastructure across multiple Dataproc clusters. You can run multiple Dataproc clusters on the same node pools, thereby allowing you further optimize costs. Some customers are now sharing multiple development environments on the same infrastructure. The same is applicable for testing, validation and certification environments.
* Multiple Spark versions on the same infrastructure
  + You can easily migrate from one version of Spark to another with the support for multiple versions on the same node pool. Your cluster management is simplified as you do not need to create two distinct environments and do not have to plan scaling down ‘existing’ cluster and scaling up the ‘upgraded’ cluster.

**Objective:** Create a Dataproc on GKE virtual cluster, then run a Spark job on the cluster.

**Before you begin**

1. You must have created a standard (not autopilot) [Google Kubernetes Engine (GKE) **zonal** or **regional** cluster](https://cloud.google.com/kubernetes-engine/docs/how-to#creating-clusters) that has [Workload Identity enabled on the cluster](https://cloud.google.com/kubernetes-engine/docs/how-to/workload-identity).**Performance tips:**
   * Enable [image streaming](https://cloud.google.com/kubernetes-engine/docs/how-to/image-streaming) for faster workload initialization.

## Create a Dataproc on GKE virtual cluster

A Dataproc on GKE virtual cluster is created as the deployment platform for Dataproc components. It is a **virtual** resource, and unlike a legacy Dataproc on Compute Engine cluster, does not include separate Dataproc master and worker VMs. Instead, Dataproc on GKE creates node pools within a GKE cluster when you create a Dataproc on GKE virtual cluster. Dataproc on GKE jobs are run as pods on these node pools. The node pools and scheduling of pods on the node pools are managed by GKE.

* **Create multiple virtual clusters.** You can create and run multiple virtual clusters on a GKE cluster to obtain improved resource utilization by sharing node pools across the virtual clusters. Each virtual cluster is created with separate properties, including Spark engine version and workload identity, and is isolated within a separate GKE namespace on the GKE cluster. Deletion of one or more Dataproc on GKE clusters does not delete associated node pools; node pools are deleted when the GKE cluster is deleted (see [Node pool Deletion](https://cloud.google.com/dataproc/docs/guides/dpgke/dataproc-gke-nodepools#node_pool_deletion)).
* In the Cloud console, go to the Dataproc **Clusters** page.

[Go to Clusters](https://console.cloud.google.com/dataproc/clusters)

* Click **Create cluster**.
* In the **Create Dataproc cluster** dialog, click **Create** in the **Cluster on GKE** row.

