Distributed Image Processing in Cloud Dataproc

Overview

In this hands-on lab, you will learn how to use Apache Spark on Cloud Dataproc to distribute a computationally intensive image processing task onto a cluster of machines. This lab is part of a series of labs on processing scientific data.

What you'll learn

- How to create a managed Cloud Dataproc cluster with Apache Spark pre-installed.
- How to build and run jobs that use external packages that aren't already installed on your cluster.
- How to shut down your cluster.

Introduction

Cloud 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. Cloud 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.

Consider using Cloud Dataproc to scale out compute-intensive jobs that meet these characteristics:

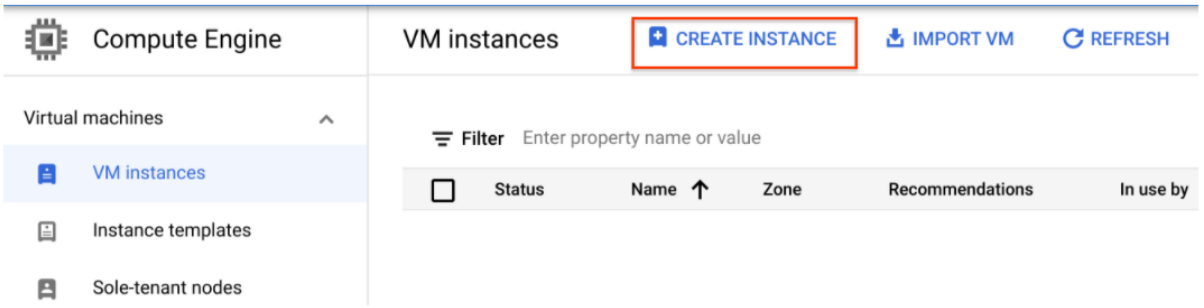
1. The job is **embarrassingly parallel** —in other words, you can process different subsets of the data on different machines.
2. You already have **Apache Spark** code that does the computation or you are familiar with Apache Spark.
3. The distribution of the work is pretty **uniform** across your data subsets.

If different subsets require different amounts of processing (or if you don't already know Apache Spark), Apache Beam on Cloud Dataflow is a compelling alternative because it provides autoscaling data pipelines.

In this lab, the job that you will run outlines the faces in the image using a set of image processing rules specified in OpenCV. The Vision API is a better way to do this, since these sort of hand-coded rules don't work all that well, but this lab is an example of doing a compute-intensive job in a distributed way.

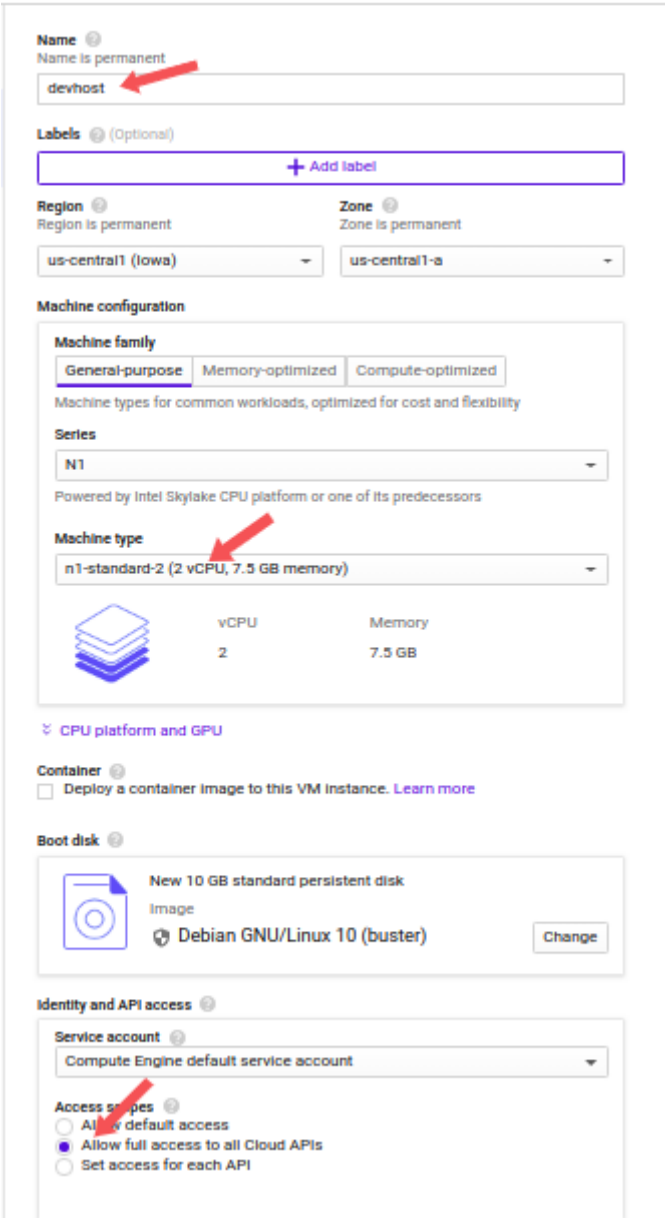
Create a development machine in Compute Engine

First, you create a virtual machine to host your services. In the Cloud Console, go to **Compute Engine** > **VM Instances** > **Create Instance**.



Configure the following fields, leave the others at their default value.

- **Name**: devhost
- **Series**: N1
- **Machine Type**: 2 vCPUs (n1-standard-2 instance)
- **Identity and API Access**: Allow full access to all Cloud APIs.



Click **Create**. This will serve as your development 'bastion' host.

Now SSH into the instance by clicking the **SSH** button on the Console.

Install Software

Now set up the software to run the job. Using `sbt`, an open source build tool, you'll build the JAR for the job you'll submit to the Cloud Dataproc cluster. This JAR will contain the program and the required packages necessary to run the job. The job will detect faces in a set of image files stored in a Cloud Storage bucket, and write out image files with the faces outlined, to either the same or to another Cloud Storage bucket.

Step 1: Set up Scala and sbt

In the SSH window, install `Scala` and `sbt` with the following commands so that you can compile the code:

```
sudo apt-get install -y dirmngr unzip
sudo apt-get update
sudo apt-get install -y apt-transport-https
echo "deb https://repo.scala-sbt.org/scalasbt/debian all main" | sudo tee
/etc/apt/sources.list.d/sbt.list
echo "deb https://repo.scala-sbt.org/scalasbt/debian /" | sudo tee
/etc/apt/sources.list.d/sbt_old.list
curl -sL
"https://keyserver.ubuntu.com/pks/lookup?op=get&search=0x2EE0EA64E40A89B84B
2DF73499E82A75642AC823" | sudo apt-key add
sudo apt-get update
sudo apt-get install apt-transport-https curl gnupg -yqq
echo "deb https://repo.scala-sbt.org/scalasbt/debian all main" | sudo tee
/etc/apt/sources.list.d/sbt.list
echo "deb https://repo.scala-sbt.org/scalasbt/debian /" | sudo tee
/etc/apt/sources.list.d/sbt_old.list
curl -sL
"https://keyserver.ubuntu.com/pks/lookup?op=get&search=0x2EE0EA64E40A89B84B
2DF73499E82A75642AC823" | sudo -H gpg --no-default-keyring --keyring
gnupg-ring:/etc/apt/trusted.gpg.d/scalasbt-release.gpg --import
sudo chmod 644 /etc/apt/trusted.gpg.d/scalasbt-release.gpg
sudo apt-get update
sudo apt-get install -y bc scala sbt
sudo apt install default-jdk
```

Step 2: Set up the Feature Detector Files

Now you'll build the Feature Detector files. The code for this lab is a slight modification of a solution which exists in the Cloud Dataproc repository on GitHub. You'll download the code, then `cd` into the directory for this lab and build a "fat JAR" of the feature detector so that it can be submitted to Cloud Dataproc. Run the following commands **in the SSH window**:

```
sudo apt-get update
gsutil cp gs://spls/gsp124/cloud-dataproc.zip .
unzip cloud-dataproc.zip
cd cloud-dataproc/codelabs/opencv-haarcascade
```

Step 3: Launch build

This command builds a "fat JAR" of the Feature Detector so that it can be submitted to Cloud Dataproc:

```
sbt assembly
```

Note: This step will take a while to process, approximately five or more minutes. Please be patient.

Create a Cloud Storage bucket and collect images

Now that you built your Feature Detector files, create a Cloud Storage bucket and add some sample images to it.

Step 1

Fetch the Project ID to use to name your bucket.

```
GCP_PROJECT=$(gcloud config get-value core/project)
```

Name your bucket and set a shell variable to your bucket name. The shell variable will be used in commands to refer to your bucket.

```
MYBUCKET="${USER//google}-image-${RANDOM}"
echo MYBUCKET=${MYBUCKET}
```

Step 2

Use the `gsutil` program, which comes with `gcloud` in the Cloud SDK, to create the bucket to hold your sample images:

```
gsutil mb gs://${MYBUCKET}
```

Step 3

Download some sample images into your bucket:

```
curl
https://www.publicdomainpictures.net/pictures/20000/velka/family-of-three-87129096
3799xUk.jpg | gsutil cp - gs://${MYBUCKET}/imgs/family-of-three.jpg
```

```
curl
https://www.publicdomainpictures.net/pictures/10000/velka/african-woman-3312879125
08yqXc.jpg | gsutil cp - gs://${MYBUCKET}/imgs/african-woman.jpg
curl
https://www.publicdomainpictures.net/pictures/10000/velka/296-1246658839vCW7.jpg |
gsutil cp - gs://${MYBUCKET}/imgs/classroom.jpg
```

You just downloaded the following images into your Cloud Storage bucket:

Step 4

Run this to see the contents of your bucket:

```
gsutil ls -R gs://${MYBUCKET}
```

Create a Cloud Dataproc cluster

Step 1

Run the following commands **in the SSH window** to name your cluster and to set the MYCLUSTER variable. You'll be using the variable in commands to refer to your cluster:

```
MYCLUSTER="${USER/_/-}-qwiklab"
echo MYCLUSTER=${MYCLUSTER}
```

Step 2

Set a global Compute Engine region to use and create a new cluster:

```
gcloud config set dataproc/region us-central1
```

```
gcloud dataproc clusters create ${MYCLUSTER} --bucket=${MYBUCKET}
--worker-machine-type=n1-standard-2 --master-machine-type=n1-standard-2
--initialization-actions=gs://spls/gsp010/install-libgtk.sh
--image-version=2.0
```

If prompted to use a zone instead of a region, enter **Y**.

This might take a couple minutes. The default cluster settings, which include two worker nodes, should be sufficient for this lab. `n1-standard-2` is specified as both the worker and master machine type to reduce the overall number of cores used by the cluster.

For the `initialization-actions` flag, you are passing a script which installs the `libgtk2.0-dev` library on each of your cluster machines. This library will be necessary to run the code.

If your cluster fails to create, try deleting your cluster (`gcloud dataproc clusters delete ${MYCLUSTER}`) and then retrying the previous cluster creation command.

Submit your job to Cloud Dataproc

In this lab the program you're running is used as a face detector, so the inputted `haar` classifier must describe a face. A `haar` classifier is an XML file that is used to describe features that the program will detect. You will download the haar classifier file and include its Cloud Storage path in the first argument when you submit your job to your Cloud Dataproc cluster.

Step 1

Run the following command **in the SSH window** to load the face detection configuration file into your bucket:

```
curl
https://raw.githubusercontent.com/opencv/opencv/master/data/haarcascades/ha
arcascade_frontalface_default.xml | gsutil cp -
gs://${MYBUCKET}/haarcascade_frontalface_default.xml
```

Step 2

Use the set of images you uploaded into the `imgs` directory in your Cloud Storage bucket as input to your Feature Detector. You must include the path to that directory as the second argument of your job-submission command.

Submit your job to Cloud Dataproc:

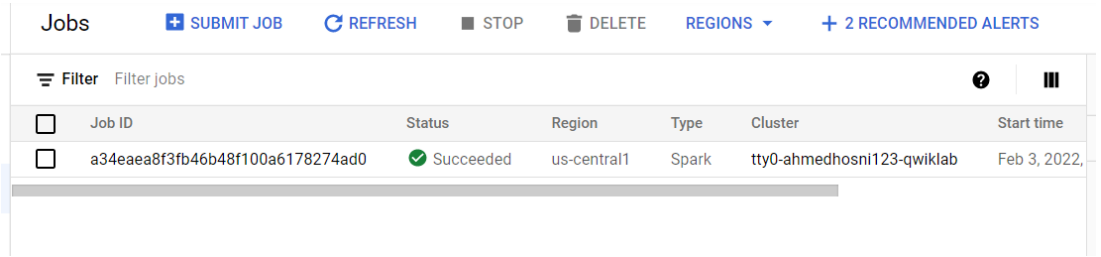
```
cd ~/cloud-dataproc/codelabs/opencv-haarcascade

gcloud dataproc jobs submit spark \
--cluster ${MYCLUSTER} \
--jar target/scala-2.12/feature_detector-assembly-1.0.jar -- \
gs://${MYBUCKET}/haarcascade_frontalface_default.xml \
gs://${MYBUCKET}/imgs/ \
gs://${MYBUCKET}/out/
```

You can add any other images to use to the Cloud Storage bucket specified in the second argument.

Step 3

Monitor the job, in the Console go to **Navigation menu** > **Dataproc** > **Jobs**. Move on to the next step when you get a similar output:



Step 4

When the job is complete, go to **Navigation menu** > **Cloud Storage** and find the bucket you created (it will have your username followed by `student-image` followed by a random number) and click on it. Then click on an image in the **Out** directory. Click on **Download** icon, the image will download to your computer.

How accurate is the face detection? The Vision API is a better way to do this, since this sort of hand-coded rules don't work all that well. You can see how it works next.