Introduction to the Cloud and Apache Spark

Overview

- Introduction to the Cloud
- Context
- Basics of Apache Spark
- Run Apache Spark on the Cloud

Why would you go with the Cloud?

On-demand self-service — No human intervention needed to get resources

Broad network access — Access from anywhere

Resource pooling — Provider shares resources to customers

Rapid elasticity — Get more resources quickly as needed

Measured service — Pay only for what you consume

What is Cloud in the REAL World?

- **"Cloud"** refers to large Internet services running on 10,000s of machines (Amazon, Google, Microsoft, etc)
- **"Cloud computing"** refers to services by these companies that let external customers rent cycles and storage
    - Amazon EC2: virtual machines at 8.5¢/hour
    - Amazon S3: storage at 21¢/GB/month
    - Google Cloud AppEngine
    - Windows Azure

Redeem your Google Cloud Coupon

- Redeem your coupon by copying and pasting the coupon URL that can be found on Canvas into your browser and type your @andrew.cmu.edu email.

- After typing your @andrew email, you will be emailed a URL to use your Google Cloud coupon. Click on the link and switch the email to your @gmail.com email. In other words, relogin to your GMAIL account and make sure that the account shown on your profile page when redeeming is your @gmail.com (Check the screenshot next page)

- Remember that you get only one coupon. Make sure your google account on the **top right** corner of the screen shows your GMAIL account.

- As a student, you get a 50 dollars cloud credit coupon. This coupon can be refilled by the instructor.
- You **MUST** disable your billing account when you finish using your cloud account. Failure to do so will lead to your account running out of balance.
- Do not redeem the free trial on your Google Cloud account. If you redeemed Google cloud credit earlier, use different GMAIL email. If you are not sure, use different email. Your coupon will be lost (and unredeemable) if it was redeemed in an account that used free trial credits before.

You agree that we may share the following information with your educational
institution and course instructor: (1) personal information that you provide to us
during the coupon redemption process and (2) information regarding your use of
the coupon and Google Cloud Platform products.

Accept and continue   Clear

- If it is your first time on this cloud account, you will be asked to create a project and associate the billing account with your project. The billing account used by this grant is named "Billing Account for Education"

- Remember, each time you finish the use of your cloud, you need to disable the billing on your account. In order to do so, navigate to the **Billing** section from left navigation menu and view your projects. You will see three vertical dots next to your project, from which you will be able to disable the billing on your account (Check the image below). You may enable the billing back using the same approach.

Context

Last lecture, we talked about the data science process. Now, let's switch gears and discuss our technical infrastructure that can be deployed later to the cloud. You will need Spark installed on your machine in order to run the code snippets in this lecture.

In this class, we will use **Apache Spark** for data ingestion, wrangling and EDA. But why Spark?

Why Spark?

- Developed in 2009 at UC Berkeley AMPLab, then open sourced in 2010,

- Spark is the next revolution of the popular Hadoop MapReduce framework.

- Gartner, Advanced Analytics and Data Science (2014) "Organizations that are looking at big data challenges – including collection, ETL, storage, exploration and analytics – should consider Spark for its in-memory performance and the breadth of its model. It supports advanced analytics solutions on Hadoop clusters, including the iterative model required for machine learning and graph analysis."

Pandas vs PySpark (Spark developed in Python)

In a research done by Databricks, an industrial leader in the domain of big data storage and processing, PySpark shows superior performance compared to traditional implementations.
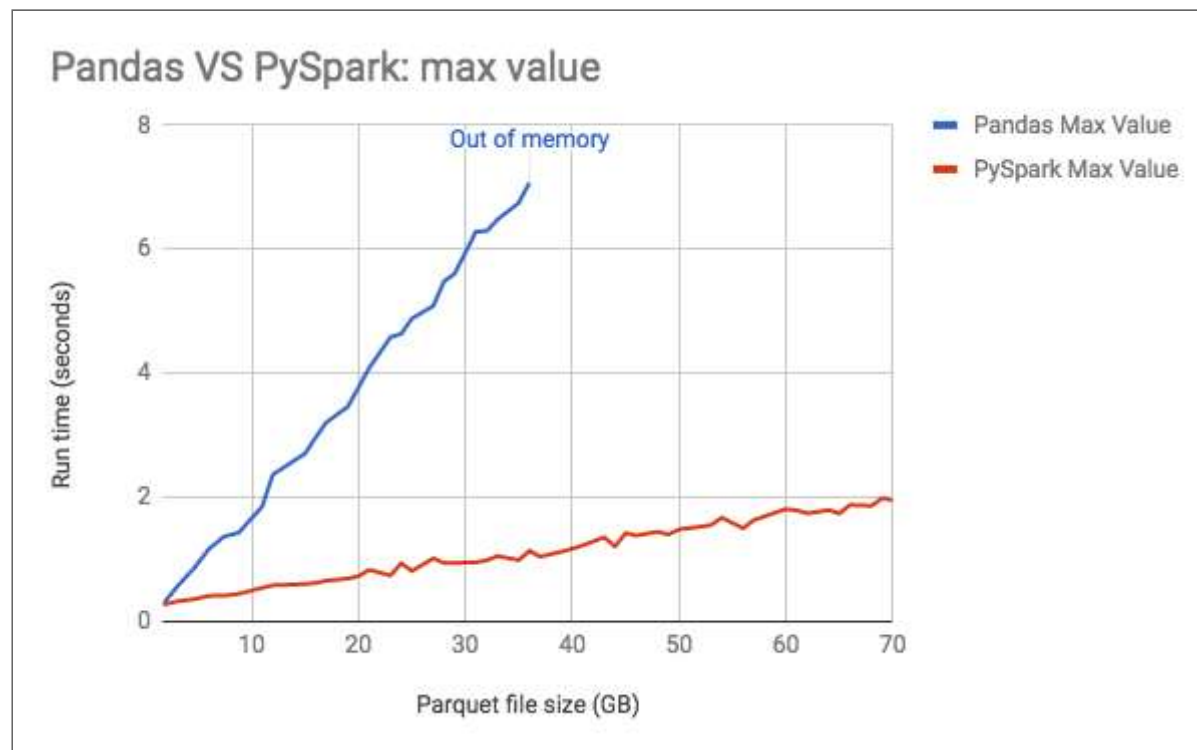
What is Spark?

- Apache Spark is a fast and general-purpose cluster computing system for large scale data processing.

- Spark was originally written in Scala, which allows concise function syntax and interactive use.

- Apache Spark provides High-level APIs in Java, Scala, Python (PySpark)and R.

- Apache Spark combines two different modes of processing:

  - **Batch-based Processing** which can be provided via Apache Hadoop MapReduce

  - **Real-time Processing** which can be provided via Apache Storm.

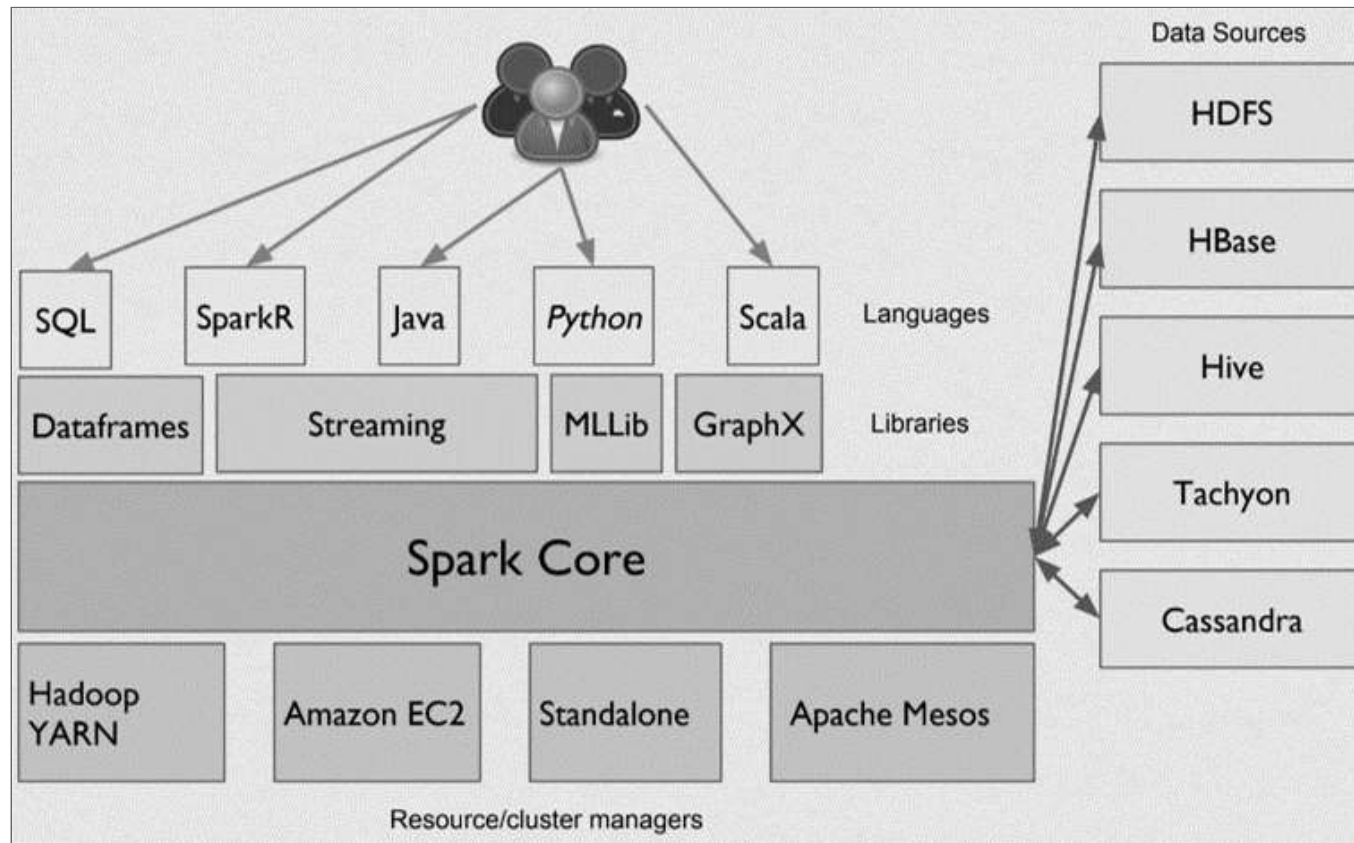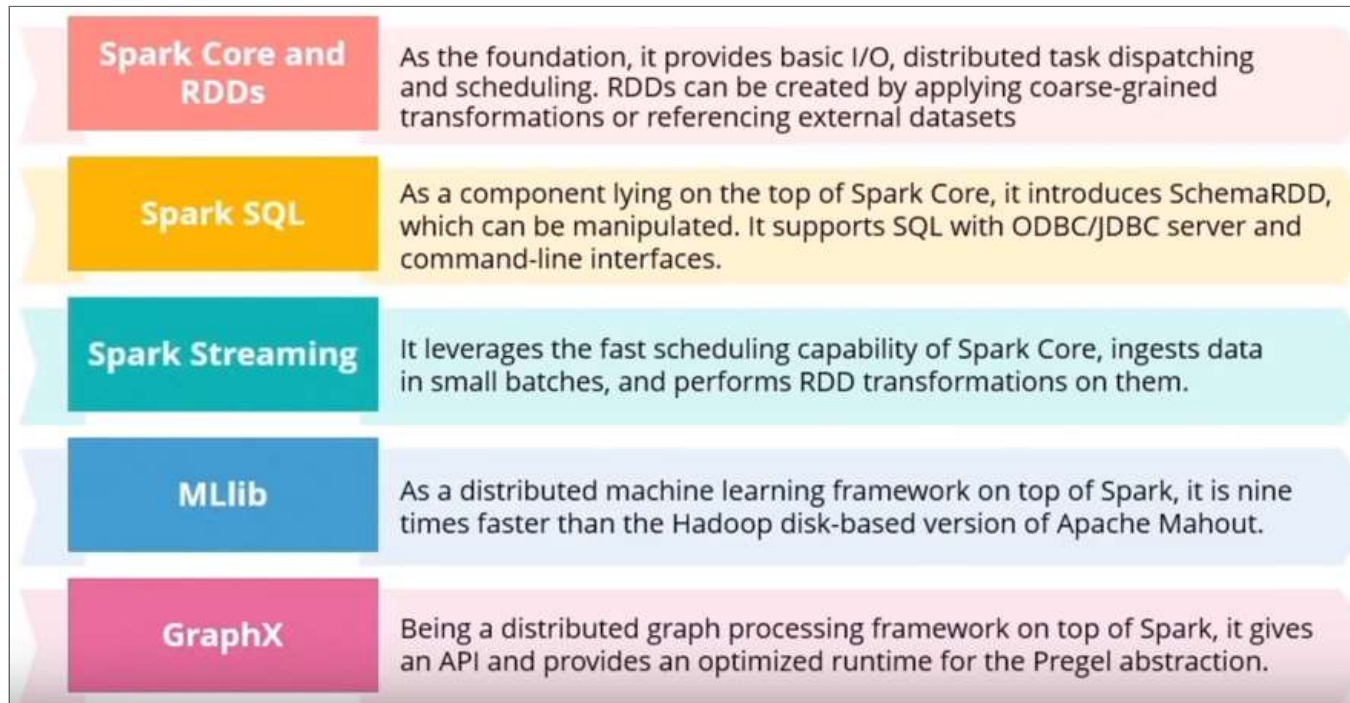| Batch Processing	Real-Time Processing
• Large group of data/transactions is processed in a single run.	• Data processing takes place upon data entry or command receipt instantaneously.
• Jobs run without any manual intervention.	• It must execute on response time within stringent constraints.
• The entire data is pre-selected and fed using command-line parameters and scripts.	
• It is used to execute multiple operations, handle heavy data load, reporting, and offline data workflow.	
Example: Regular reports requiring decision making	**Example**: Fraud detection

Spark Ecosytem

Spark Componenets

Spark Core and RDDs	As the foundation, it provides basic I/O, distributed task dispatching and scheduling. RDDs can be created by applying coarse-grained transformations or referencing external datasets
Spark SQL	As a component lying on the top of Spark Core, it introduces SchemaRDD, which can be manipulated. It supports SQL with ODBC/JDBC server and command-line interfaces.
Spark Streaming	It leverages the fast scheduling capability of Spark Core, ingests data in small batches, and performs RDD transformations on them.
MLlib	As a distributed machine learning framework on top of Spark, it is nine times faster than the Hadoop disk-based version of Apache Mahout.
GraphX	Being a distributed graph processing framework on top of Spark, it gives an API and provides an optimized runtime for the Pregel abstraction.

Spark Core

Spark Core is the general execution engine for the Spark platform that other functionalities are built on top of it. Spark has several advantages:

- Speed: runs programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk
- Ease of Use: Write applications quickly in Java, Scala, Python, R
- Generality: Combine SQL, streaming, and complex analytics
- Runs Everywhere: Spark runs on Hadoop, Mesos, standalone, or in the cloud. It can access diverse data sources including HDFS, Cassandra, HBase, and S3

```
In [ ]:
            # if you installed Spark on windows,
    # you may need findspark and need to initialize it prior to being able to use pyspark

    !pip install findspark
```

```
In [ ]:
            # Uncomment the following lines if you are using Windows!
#import findspark
#findspark.init()
#findspark.find()

import pyspark
from pyspark.sql import SparkSession


spark = SparkSession.builder.master("local[*]").appName('SparkTest').getOrCreate()
```

Download Data Files Remotely

```
In [ ]:
        !pip install wget
```

```
In [ ]:
        # Downloading and preprocessing KDD Train Data
!python -m wget https://www.andrew.cmu.edu/user/mfarag/14763/KDDTrain+.txt
```

Cloud Consideration

Your data need to be moved to a special storage server if you are running Spark on the Cloud. This special storage is called HDFS. The following command is used to move your data from your local storage to the special storage server.

```
In [ ]:
        # Uncomment and Execute this line if you are running your notebook on the Cloud
#!hadoop fs -put KDDTrain+.txt /
```

Spark Dataframes

Inspired by pandas DataFrames in structure, format, and a few specific operations, Spark DataFrames are like distributed in-memory tables with named columns and schemas, where each column has a specific data type: integer, string, array, map, real, date, timestamp, etc. To a human's eye, a Spark DataFrame is like a table

When data are visualized as a structured table, it's not only easy to digest but also easy to work with when it comes to common operations you might want to execute on rows and columns. Also, DataFrames are immutable and Spark keeps a lineage of all transformations. You can add or change the names and data types of the columns, creating new DataFrames while the previous versions are preserved. A named column in a DataFrame and its associated Spark data type can be declared in the schema.

Let's examine the generic and structured data types available in Spark before we use them to define a schema. Then we'll illustrate how to create a DataFrame with a schema.

Spark's Basic Data Types

Spark supports basic internal data types. These data types can be declared in your Spark application or defined in your schema

Data type	Value assigned in Python	API to instantiate
ByteType	int	DataTypes.ByteType
ShortType	int	DataTypes.ShortType
IntegerType	int	DataTypes.IntegerType
LongType	int	DataTypes.LongType
FloatType	float	DataTypes.FloatType
DoubleType	float	DataTypes.DoubleType
StringType	str	DataTypes.StringType
BooleanType	bool	DataTypes.BooleanType
DecimalType	decimal.Decimal	DecimalType

```
In [ ]:
            # Load data from csv to a dataframe on a local machine.
# header=False means the first row is not a header
# sep=',' means the column are seperated using ','
df = spark.read.csv('KDDTrain+.txt', header=False, sep=",")
# on the Cloud, the files will have to be at the root level. So, the cloud version is:
#df = spark.read.csv('/KDDTrain+.txt', header=False, sep=",")

df.show(5, vertical=True)
```

Avoiding Auto-assigned Column Names: Read CSV and Specify the Column Names

```
In [ ]:
        col_names = ["duration","protocol_type","service","flag","src_bytes",
    "dst_bytes","land","wrong_fragment","urgent","hot","num_failed_logins",
    "logged_in","num_compromised","root_shell","su_attempted","num_root",
    "num_file_creations","num_shells","num_access_files","num_outbound_cmds",
    "is_host_login","is_guest_login","count","srv_count","serror_rate",
    "srv_serror_rate","rerror_rate","srv_rerror_rate","same_srv_rate",
    "diff_srv_rate","srv_diff_host_rate","dst_host_count","dst_host_srv_count",
    "dst_host_same_srv_rate","dst_host_diff_srv_rate","dst_host_same_src_port_rate",
    "dst_host_srv_diff_host_rate","dst_host_serror_rate","dst_host_srv_serror_rate",
    "dst_host_rerror_rate","dst_host_srv_rerror_rate","classes","difficulty_level"]

df = spark.read.csv("KDDTrain+.txt",header=False, inferSchema= True).toDF(*col_names)

# on the Cloud, the files will have to be at the root level. So, the cloud version is:
# df = spark.read.csv("/KDDTrain+.txt",header=False, inferSchema= True).toDF(*col_names)

df.show(5, vertical=True)
```

More ways to Display Dataframes

1. `df.take(5)` will return a list of five Row objects.

2. `df.collect()` will get all of the data from the entire DataFrame. Be really careful when using it, because if you have a large data set, you can easily crash the driver node.

3. `df.show()` is the most commonly used method to view a dataframe. There are a few parameters we can pass to this method, like the number of rows and truncaiton. For example, `df.show(5, False)` or `df.show(5, truncate=False)` will show the entire data wihtout any truncation.

4. `df.limit(5)` will **return a new DataFrame** by taking the first n rows. As spark is distributed in nature, there is no guarantee that `df.limit()` will give you the same results each time.

Schemas and Creating DataFrames

You can think about the dataframes as a table. A schema in Spark defines the column names and associated data types for a DataFrame. Most often, schemas come into play when you are reading structured data from an external data source. Defining a schema up front as opposed to taking a schema-on-read approach offers three benefits:

- You relieve Spark from the onus of inferring data types.

- You prevent Spark from creating a separate job just to read a large portion of your file to ascertain the schema, which for a large data file can be expensive and time-consuming.

- You can detect errors early if data doesn't match the schema.

You may also create your own schema using data field name followed by data type.

```
In [ ]:
            schema = "id INT, firstName STRING, WEBSITE STRING"
data = [[1, "John", "https://tinyurl.1"],
        [2, "Brooke", "https://tinyurl.2"]]

test_df = spark.createDataFrame(data, schema)
test_df.show()
test_df.printSchema()
```

Display Schema Information for Your Dataframe

```
In [ ]:
        df.printSchema() # or df.dtypes
```

Print Column Names in Your Dataframe

```
In [ ]:
        print(df.columns)
```

Print Total Number of Your Records in Your Dataframe

```
In [ ]:
        print(df.count())
```

Print Sample Record from Your Dataframe

```
In [ ]:
        df.show(1, vertical=True)
```

DataFrame Operations on Columns

1. Selecting Columns & Creating Subset Dataframes
2. Adding New Columns
3. Renaming Columns
4. Removing Columns

Create a Subset Dataframe from Your Dataframe

```
In [ ]:
        small_df = df.select("duration","protocol_type","service","classes","difficulty_level")
small_df.show(5)
```

Display Summary Statistics in Your Dataframe

```
In [ ]:
        df.describe().show(vertical=True)
```

Display Unique Values from a Column in Your Dataframe

```
In [ ]:
        df.select("classes").distinct().show(40)
```

Add a Column to Your Dataframe

```
In [ ]:
            # We will add a new column called 'first_column' at the end
from pyspark.sql.functions import lit
df = df.withColumn('first_column',lit(1))
# lit means literal. It populates the row with the literal value given.
# When adding static data / constant values, it is a good practice to use it.
df.show(1,vertical=True)
```

Renaming a Column in Your Dataframe

```
In [ ]:
        df = df.withColumnRenamed('first_column', 'new_column_one')

df.printSchema()
```

Delete a Column from Your Dataframe

```
In [ ]:
            df = df.drop('new_column_one')
df.printSchema()
```

Lab: Run Spark on the Cloud

Create Clusters (e.g. Hadoop Clusters)

• A cluster is group of machines, servers, or nodes. It helps providing the sum of the computational power offered by all incorporated machines. It's difficult to build a local machine with 64GB RAM and 20TB of Storage but that is not difficult when you are running on the cloud.

• You may start by navigating to Dataproc and click on the Clusters section
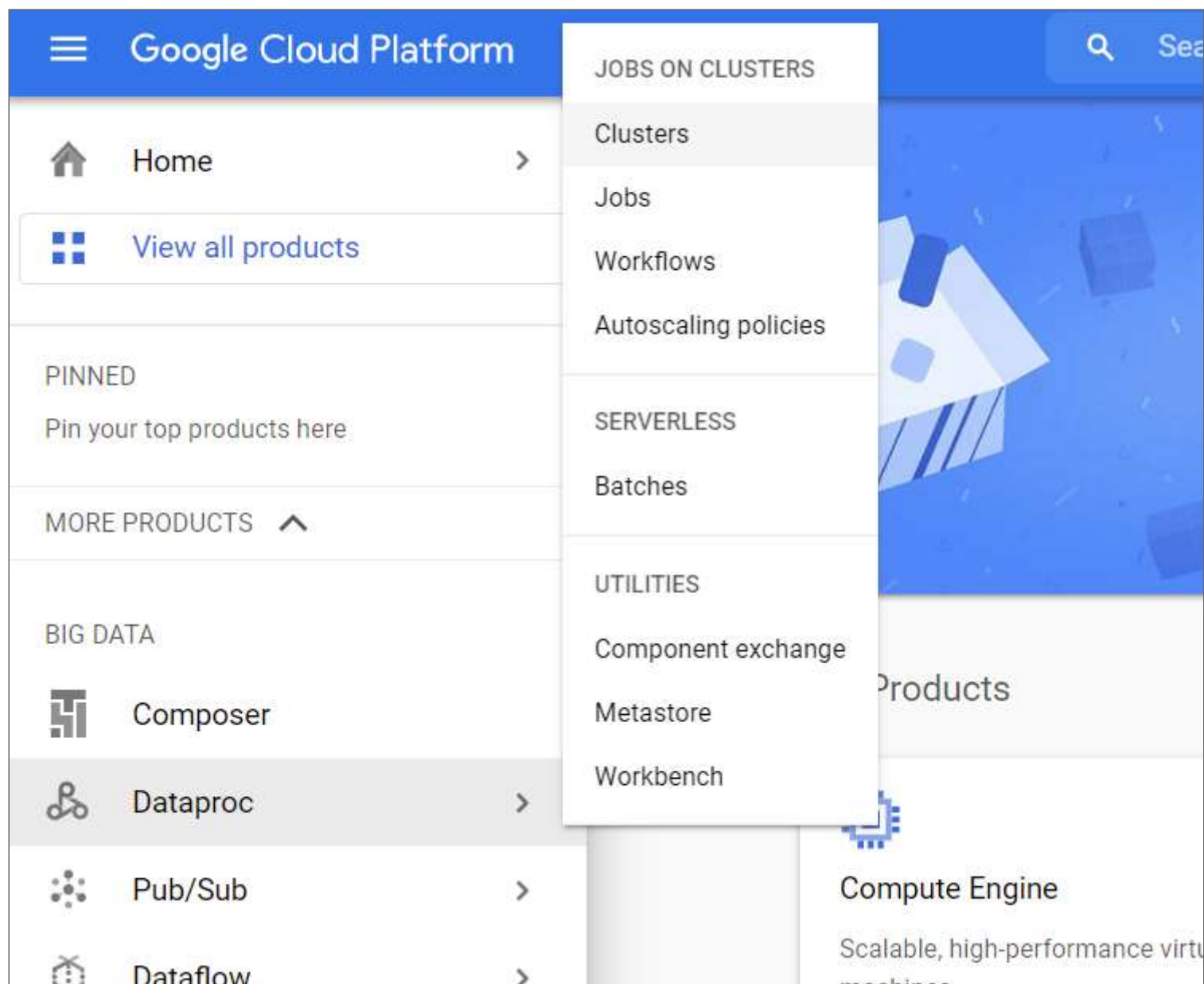
Cluster - Setup

• Next, you need to create your cluster and choose the cluster to use "Google Compute Engine"
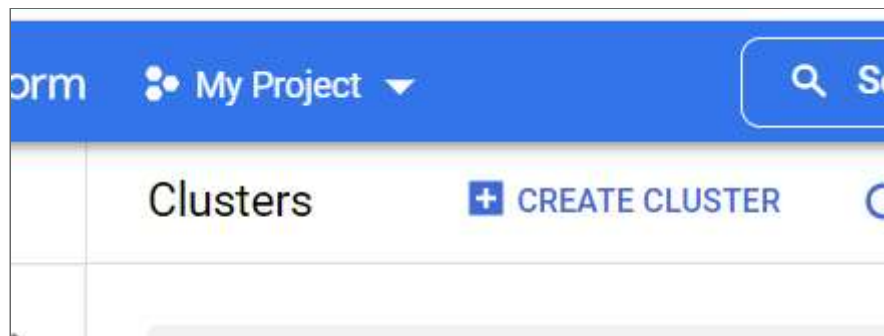
Cluster Configuration

Make sure to conduct the following:

1. Enable component gateway
2. Select to add `Docker` and `Jupyter Notebooks` from the list
   of available components

Running Cluster

Once you click on the create button, Google Cloud will work on creating your own cluster and if it's successful, you will see your cluster running.

Connect to Your Cluster

From the Web Interfaces, open the Jupyter Notebook. Upload your Notebook to `Local Disk/home` folder and run the cells.