Submit Apache Spark Applications Lab

Estimated time needed: 20 minutes

In this lab, you will learn how to submit Apache Spark applications from a Python script. This exercise is straightforward, thanks to Docker Compose.

Learning objectives

In this lab, you will:

- Install a Spark Master and Worker using Docker Compose
- Create a Python script containing a Spark job
- Submit the job to the cluster directly from python (Note: you'll learn how to submit a job from the command line in the Kubernetes lab)

Prerequisites

```
**Note**: If you are running this lab within the Skills Network Lab environment, all prerequisites are already installed for you
```

The only prerequisites for this lab are:

- The *wget* command line tool
- A Python development environment

Start the Spark Master

On the right hand side of these instructions, you'll find the Cloud IDE. Select the *Lab* tab. Then, in the menu bar, select *Terminal* > *New Terminal*.

2. Enter the following commands in the terminal to download the Spark environment:

```
wget https://archive.apache.org/dist/spark/spark-3.3.3/spark-3.3.3-bin-hadoop3.tgz && tar xf spark-3.3.3-bin-hadoop3.tgz && rm -rf spark-3.3.3-bin-ha
```

   This process may take some time. It downloads Spark as a zipped archive and then unzips it into the current directory.

3. Run the following commands to set up `JAVA_HOME` (preinstalled in the environment) and `SPARK_HOME` (which you just downloaded):

```
export JAVA_HOME=/usr/lib/jvm/java-1.11.0-openjdk-amd64
export SPARK_HOME=/home/project/spark-3.3.3-bin-hadoop3
```

4. Run the following command to create a config file for the master:

```
touch /home/project/spark-3.3.3-bin-hadoop3/conf/spark-defaults.conf
```

5. Click the button below to begin the Spark Master configuration process.

Open **spark-defaults.conf** in IDE

6. Paste the following content into the `spark-defaults.conf` file. This will configure the number of cores and the amount of memory that the Master will allocate to the workers.

```
spark.executor.memory 4g
spark.executor.cores 2
```

7. Navigate to the `SPARK_HOME` directory:

```
cd $SPARK_HOME
```

8. Run the Spark master by executing the following command:

```
./sbin/start-master.sh
```

9. Once it starts up successfully, click the button below to verify that the Master is running as expected.

Spark Master

If the application has started up successfully, you will see a page as given below.

10. Copy the URL of the Master as shown in the image and note it down in a text editor, such as Notepad, on your computer.

Start the worker

1. Click `Terminal` from the menu, and click `New Terminal` to open a new terminal window.

2. Once the terminal opens up at the bottom of the window, run the following commands to set up `JAVA_HOME` and `SPARK_HOME`:

```
export JAVA_HOME=/usr/lib/jvm/java-1.11.0-openjdk-amd64
export SPARK_HOME=/home/project/spark-3.3.3-bin-hadoop3
```

3. Navigate to the `SPARK_HOME` directory:

```
cd $SPARK_HOME
```

4. Run the Spark worker by executing the following command. Remember to replace the placeholder `yourname` in the command below with your name as given in the Spark master URL that you noted down in the previous step.

```
./sbin/start-worker.sh spark://theiadocker-yourname:7077 --cores 1 --memory 1g
```

5. Once it starts up successfully, click the button below to verify that the Worker is running as expected.

Spark Master

You should see that the Worker is now registered with the Master.

Create code and submit

1. Click `Terminal` from the menu, and click `New Terminal` to open a new terminal window.

2. Once the terminal opens up at the bottom of the window, run the following command to create the Python file:

```
touch submit.py
```

A new Python file called `submit.py` is created.

3. Open the file in the file editor by clicking the button below or following the visual guidance in the image.

Open **submit.py** in IDE

4. Paste the following code to the file and save it. Remember to replace the placeholder `yourname` in the code below with your name as in the Spark master URL.

```
import findspark
findspark.init()
from pyspark import SparkContext, SparkConf
from pyspark.sql import SparkSession
from pyspark.sql.types import StructField, StructType, IntegerType, StringType
spark = SparkSession.builder \
    .master('spark://theiadocker-yourname:7077') \
    .config('spark.executor.cores', '1') \
    .config('spark.executor.memory', '512m') \
    .getOrCreate()
df = spark.createDataFrame(
    [
        (1, "foo"),
        (2, "bar"),
    ],
    StructType(
        [
            StructField("id", IntegerType(), False),
            StructField("txt", StringType(), False),
        ]
    ),
)
print(df.dtypes)
print("\nDataFrame:")
df.show()
```

3. Run the following commands to set up `JAVA_HOME` and `SPARK_HOME`:

```
export JAVA_HOME=/usr/lib/jvm/java-1.11.0-openjdk-amd64
export SPARK_HOME=/home/project/spark-3.3.3-bin-hadoop3
```

4. Install the required packages to set up the Spark environment.

```
pip3 install findspark
```

5. Type in the following command in the terminal to execute the Python script:

```
python3 submit.py
```

You will see output as below:

```
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
25/01/27 23:50:53 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
[('id', 'int'), ('txt', 'string')]
+---+---+
| id|txt|
+---+---+
|  1|foo|
|  2|bar|
+---+---+
```

Experiment yourself

Please take a look at the UI of the Apache Spark Master and Worker.

1. Click the button below to launch the `Spark Master`. Alternatively, click the Skills Network button on the left. This will open the “Skills Network Toolbox.” Then, click `Other` followed by `Launch Application`. From there, you can enter the port number as `8080` and launch the application.

Spark Master

2. This will take you to the Spark Master's admin UI (if your popup blocker doesn't interfere).

▾ Running Applications (0)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration

▾ Completed Applications (1)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
app-20250128003023-0000	pyspark-shell	1	512.0 MiB		2025/01/28 00:30:23	theia	FINISHED	13 s

3. Please note that you can see all registered workers (one in this case) and all running or completed jobs (also one in this case).

   Note: Due to the limitations of the lab environment, you cannot click on links within the UI. In a typical production environment, these links would be functional.

4. Click the button below to open the `Spark Worker` on 8081. Alternatively, click the Skills Network button on the left, it will open the "Skills Network Toolbox." Then, click `Other` followed by `Launch Application`. From there, you should be able to enter the port number as `8081` and launch the application.

Spark Worker

You should be able to find your currently running job listed here.

Summary

In this lab, you've learned how to set up an experimental Apache Spark cluster using Docker Compose. You are now able to submit a Spark job directly from Python code. In the subsequent Kubernetes lab, you will learn how to submit Spark jobs from the command line as well.

Author(s)

Romeo Kienzler
Lavanya T S