evolve

Leading the Big Data Revolution

Cookbook

Guidelines for using PySpark 3.X on EVOLVE dashboard

ICCS Group

This document describes the guidelines for using PySpark 3.X through zep-pelin notebook on the EVOLVE dashboard. We provide a simple ETL example that loads a 2.5 GB dataset and performs an SQL query. Finally, we provide the configuration for enabling CPU only as well as GPU accelerated execution in PySpark 3.X.

For any issues or questions please contact **aferikoglou@microlab.ntua.gr**

Contents

evolve
Leading the Big Data Revolution

1.

PySpark 3.X Zeppelin Notebook Basics

1. PySpark 3.X Zeppelin Notebook Basics

After logging in to the EVOLVE platform, select the provided template.

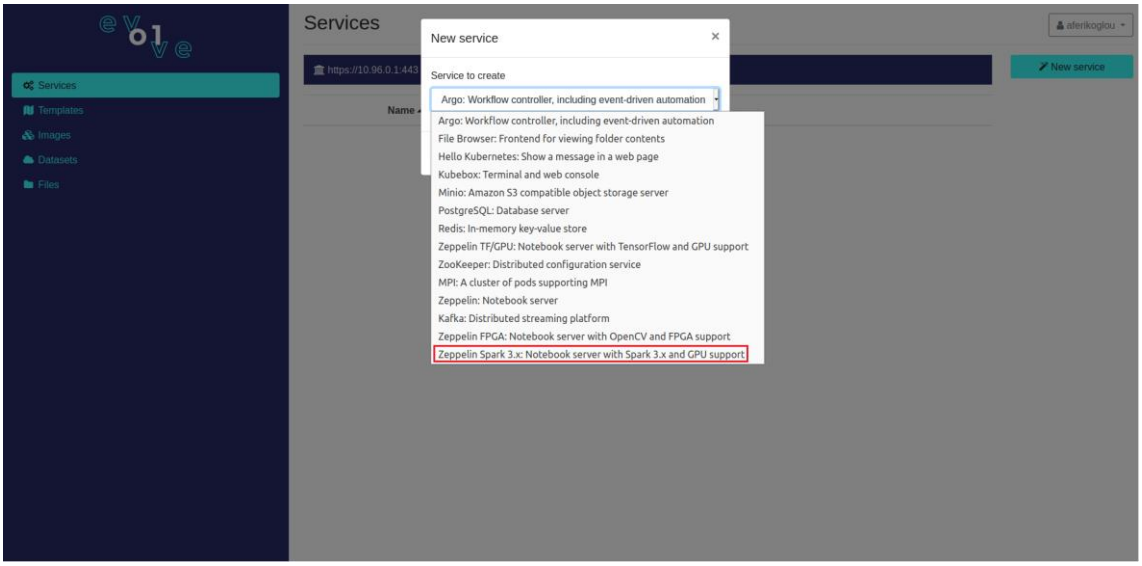

Figure 1: Spark 3.X Template Selection

In order to execute your PySpark application you must add at least 2 para-graphs. One configuration paragraph (starts with %spark.conf tag) and one or more code paragraphs (starts with %spark.pyspark tag). The configuration paragraph describes the way your code is going to be executed on the underlying Kubernetes cluster. The code paragraphs contain the PySpark code you want to execute.

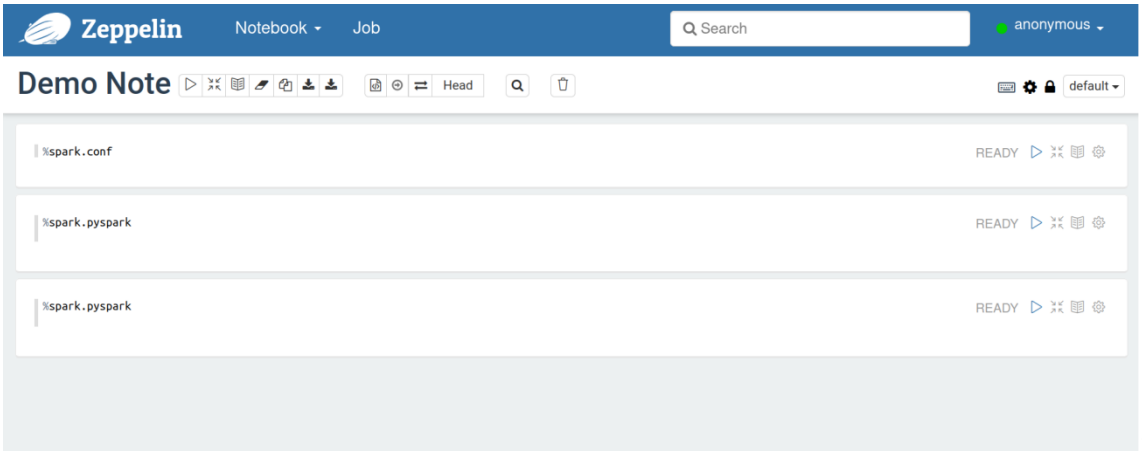

Figure 2: PySpark 3.X Demo Note

Note that you should allow the created zeppelin pod in karvdash-USERNAME namespace to create pods and services. This can be done by executing the fol-lowing commands.

```
$ kubectl create serviceaccount spark --namespace=karvdash-USERNAME
$ kubectl create clusterrolebinding spark-role --clusterrole=edit
      --serviceaccount=karvdash-USERNAME:spark
      --namespace=karvdash-USERNAME
```

Listing 1: Create Spark Service Account

The first command provides an identity for processes that run in a pod. In this way, processes in containers inside pods can also contact the API Server. When they do, they are authenticated as a particular Service Account (for example, spark). The second command provides edit permissions across a whole cluster for the previously created spark service account.

evolve
Leading the Big Data Revolution

2.

ETL Example

2. ETL Example

As we already mentioned, we created a simple ETL example which loads a 2.5 GB dataset and performs an SQL query. The used code is presented below.

```
% spark . pyspark

from __future__ import print_function

import sys

from pyspark . sql import SparkSession

if __name__ == " __main__ ":
   # Absolute path to data folder
   data_dir = '/ opt / spark / examples / src / main / python / '
   # Name of input file
   fn = 'nf - chunk2 . csv '

   spark = SparkSession \
      . builder \
      . appName (" ETL Demo ") \
      . getOrCreate ()

netflow_df = spark . read . format ( ' com . databricks . spark . csv '). options ( header = ' true ', inferschema = ' true '). load ( data_dir + fn )

netflow_df . createOrReplaceTempView ( ' netflow ')

query = '''
      SELECT
          a.firstSeenSrcIp as source ,
          a. firstSeenDestIp as destination ,
          count (a. firstSeenDestPort ) as targetPorts ,
          SUM (a. firstSeenSrcTotalBytes ) as bytesOut ,
          SUM (a. firstSeenDestTotalBytes ) as bytesIn ,
          SUM (a. durationSeconds ) as durationSeconds ,
          MIN ( parsedDate ) as firstFlowDate ,
          MAX ( parsedDate ) as lastFlowDate ,
          COUNT (*) as attemptCount
      FROM
          netflow a
      GROUP BY
          a.firstSeenSrcIp ,
          a. firstSeenDestIp
      '''

      edges_df = spark . sql ( query )

      edges_df . show (50)

      spark . stop ()
```

Listing 2: PySpark ETL Example

The code is pretty straightforward. At first, the path to input file is defined and the spark session is initialized. After specifying the path, the input data are loaded into a dataframe. Then the SQL query is defined and executed while the first 50 rows of the output table are shown. Finally, the spark session is stopped.

Note that you must put the input data (in our case nf-chunk2.csv) in the same path in both of the zeppelin and executor pods.

In the following subsections we provide configuration files for CPU-only and GPU accelerated execution in PySpark 3.X. These configuration files provide simple templates. It should be clear that users can add lots of different config-urations in order to fine-tune their applications.

2.1. CPU Only Execution Configuration

PySpark 3.X can perform computation in a CPU-only manner just like its pre-vious versions. The following configuration file creates 1 executor which uses only CPU resources, binds 32 GB of memory and is executed on node ns66.

```
% spark.conf

master k8s: // https: // kubernetes.default.svc
spark.submit.deployMode client
name etlexample
spark.executor.instances 1
spark.executor.memory 32 g
spark.kubernetes.authenticate.driver.serviceAccountName spark 9
spark.kubernetes.container.image 172 .9.0.240:5000 /
    zeppelin-spark3rapids-executor:v0.1
spark.kubernetes.node.selector.kubernetes.io / hostname=ns66
```

Listing 3: CPU Configuration

2.2 GPU Accelerated Execution Configuration

PySpark 3.X provides GPU operations in order to accelerate the overall execu-tion. The following configuration file creates 1 executor which binds 32 GB of memory and 1 GPU. The executor is executed on ns66 node because PySpark 3.X supports specific NVIDIA GPU architectures (NVIDIA Pascal or better).

In particular, GPU accelerated executors can be executed only on nodes ns64, ns65 and ns66 which are provisioned with Tesla P20 and V100 cards.

```
% spark.conf

master k8s: // https: // kubernetes.default.svc
spark.submit.deployMode client
name etlexample
spark.executor.instances 1
spark.rapids.sql.concurrentGpuTasks 1
spark.rapids.memory.gpu.pooling.enabled false
spark.task.resource.gpu.amount 1
spark.executor.memory 32 g
spark.executor.resource.gpu.amount 1
spark.executor.resource.gpu.vendor nvidia.com
spark.plugins com.nvidia.spark.SQLPlugin
spark.executor.resource.gpu.discoveryScript / opt / sparkRapidsPlugin /
    getGpusResources.sh
spark.kubernetes.authenticate.driver.serviceAccountName spark
spark.kubernetes.container.image 172 .9.0.240:5000 /
    zeppelin-spark3rapids-executor:v0.1
spark.kubernetes.node.selector.kubernetes.io / hostname=ns66
```

Listing 4: GPU Configuration

A detailed explanation of each of the following configuration parameters is presented below :

- **master :** The cluster manager to connect to (in our case the Kuber-netes cluster master). Specifies the Kubernetes API Server through which Zeppelin Server communicates with the Kubernetes cluster. (**do NOT change**)

- **spark.submit.deployMode :** The deploy mode of the Spark driver pro-gram. It is either cluster (the driver is created in a pod and executed on one of the cluster nodes) or client (the driver is created inside the Zeppelin Server pod). Client mode is only supported. (**do NOT change**)

- **spark.executor.instances :** Specifies the number of executor instances that will be created.

- **spark.rapids.sql.concurrentGpuTasks :** Specifies the number of con-current tasks per executor for the RAPIDS plugin. Some queries benefit significantly from setting this to a value between 2 and 4, with 2 typi-cally providing the most benefit, and higher numbers giving diminishing returns.

- **spark.task.resource.gpu.amount :** Specifies the number of GPUs per task. Note that spark.task.resource.gpu.amount can be a decimal amount, so if you want multiple tasks to be run on an executor at the same time and assigned to the same GPU you can set this to a decimal value less than
    - You would want this setting to correspond to the spark.executor.cores setting. For instance, if you have spark.executor.cores=2 which would allow 2 tasks to run on each executor and you want those 2 tasks to run on the same GPU then you would set spark.task.resource.gpu.amount=0.5.

- **spark.executor.memory :** Specifies the amount of memory to use per executor process, in the same format as JVM memory strings with a size unit suffix (”k”, ”m”, ”g” or ”t”).

- **spark.executor.resource.gpu.amount :** Specifies the number of GPUs per executor. Note that the RAPIDS accelerator plugin only supports a one-to-one mapping between GPUs and executors.

- **spark.executor.resource.gpu.vendor :** Specifies the GPU vendor. In our case, the vendor is nvidia.com as the Nvidia device plugin is used in order to use the Kubernetes cluster GPUs. (**do NOT change**)

- **spark.plugins :** Specifies the plugins that are going to be used. In our case, the GPU accelerated com.nvidia.spark.SQLPlugin plugin is defined.(**do NOT change**)

- **spark.executor.resource.gpu.discoveryScript :** Specifies the script that will be used for the discovery of GPUs in the Kubernetes cluster nodes. (**do NOT change**)

- **spark.kubernetes.authenticate.driver.serviceAccountName :** Specifies the name of the service account. This configuration defines the service that allows the Zeppelin Server to create other pods and services. (**do NOT change**)

- **spark.kubernetes.container.image :** Specifies the Docker container image that is going to be used for the executor.

- **spark.kubernetes.node.selector.kubernetes.io/hostname :** Specifies the node the executor pod is going to be executed (note that the defined node should have available cards). This node selector is used in order to force the executor pod to be executed in a node with a Tesla architecture GPU.

Both of these examples can be found in dashboard in the examples directory in the shared files as zeppelin notebooks.

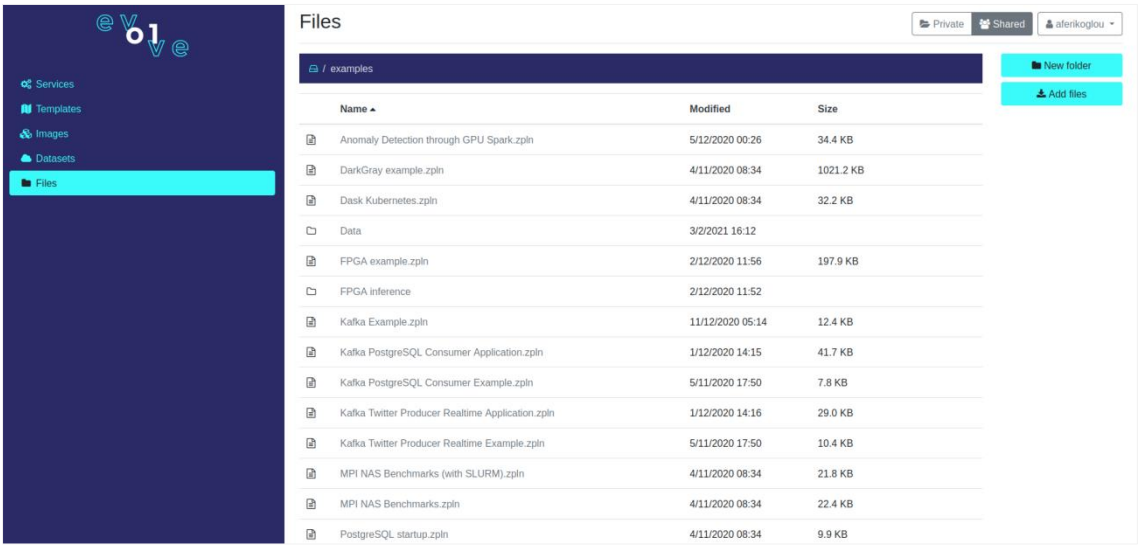

Figure 3: Zeppelin Notebook Examples

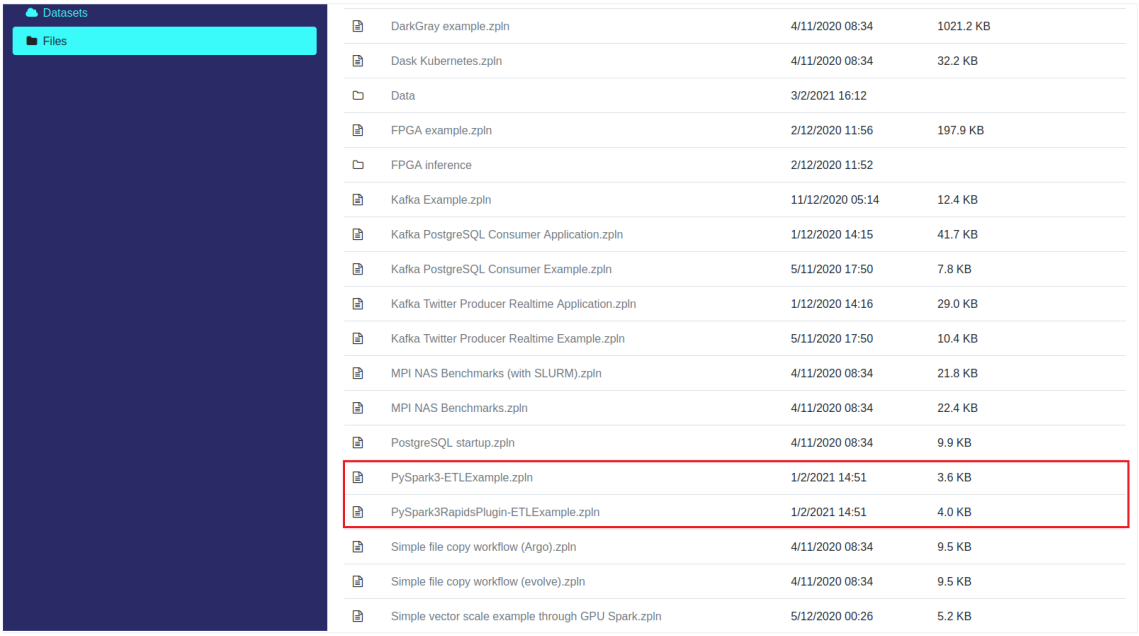

Figure 4: PySpark 3.X Zeppelin Notebook

evolve

Leading the Big Data Revolution

@evolve-h2020

@evolve_h2020

info@evolve-h2020.eu

www.evolve-h2020.eu

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825061