IT UNIVERSITY OF COPENHAGEN
BIG DATA MANAGEMENT 2022

Guide: PySpark on the Ambari Cluster

Niels Ørbæk Chemnitz, niec@itu.dk updated by Thorvald Sørensen thso@itu.dk

28.09.2022

1 INTRODUCTION

This document will help you getting access to the Ambari-managed Hadoop Cluster on which you will run small Spark applications to do data analysis.

This guide will describe the specifications of the cluster, how to get command-line access to the cluster and through Jupyter notebooks. Some recommendations are given on how to set up your development environment.

The 3rd part of the document contains some useful resources, like links to the documentation for the version of Spark we are using (2.3.1), and parts of the MIT "Missing Semester", which is hugely recommendable.

Finally this document also includes a template file that can be used when running scripts in PySpark, although for the BDM course, we will run code cells in Jupyter notebooks.

NOTE: This guide assumes that you have sent your ssh public key to Vincent Ajoubi (viaj@itu.dk) and that he has set up a user for you on the name node. See LearnIT for a guide on how to.

2 CONNECTING TO THE CLUSTER

2.1 Technical Specifications

The machines you will be running your code on is a Hadoop/Spark cluster, comprised of 5 desktop machines with the following specs:

- Intel Xeon quad-core processors
- 32Gb memory each
- Fairly old 512GB SSD disks
- Running CentOS 7.7

The cluster is running Apache Ambari for cluster management. There is one NameNode and four DataNodes, and the installed services include: HDFS, YARN, MapReduce2, Hive, Spark2 (v 2.3.1), Zeppelin and others.

2.2 Setting up your development environment

To execute code on the cluster you have 3 options:

1. Use Jupyter notebooks outside of IDE such as VS Code (recommendeded for BDM course)
2. Use your favorite editor via ssh
3. Use a ssh shell and launch an interactive python shell or run scripts

Unfortunately, it is not possible to connect to the Spark cluster through a notebook opened in (an IDE such as) Visual Studio Code. We need to run Jupyter notebooks "standalone", and use `sparkmagic` and `livy` to connect your Jupyter notebook to the cluster. If you do not wish to use a Jupyter notebook, you can connect directly to the cluster using a terminal and ssh, although it is more difficult to work with.

NOTE: *If you wish to continue working with Jupyter Notebooks for assignment 2 (recommended), please have Jupyter notebook installed e.g. through Anaconda3. It is strongly recommended to launch "jupyter notebook" from a Linux/Mac terminal. Windows users can use an "Ubuntu terminal" with Windows Subsystem for Linux (recommended) or the Anaconda Navigator app (not recommended on Windows).*

2.2.1 Connecting through Jupyter notebooks (with sparkmagic over Livy)

It is possible to setup your local notebooks to execute as PySpark on the cluster. The setup is not trivial, but I have tried to make it as simple as possible. We can only offer limited support if you are having issues, due to the high variety of Python and virtual environments students use.

INSTALLING THE SPARKMAGIC JUPYTER NOTEBOOK EXTENSION USING ANACONDA (RECOMMENDED)

Sparkmagic is set of tools for interactively working with remote Spark clusters through Livy. You can install it locally, but it can be incompatible with other pip/conda packages. Therefore, we recommend creating a new virtual environment (using conda, or optionally venv) where you can install Sparkmagic, to prevent compatibility issues. See resources on LearnIT regarding Conda environments, specifically this page. Once you have created and activated your virtual environment, you can install sparkmagic (original guide here, check it out if you're using JupyterLab instead of notebook). Here's how to do it using anaconda. Make sure to shut down any running Jupyter notebook server before continuing. Open a Linux/Mac/WSL/Anaconda terminal and write:

Activate your conda eviroment, e.g. if you named it "sparkmagicEnv" it will be:

```
conda activate sparkmagicEnv
conda install sparkmagic -y
```

Make sure that ipywidgets is properly installed by running:

```
jupyter nbextension enable --py --sys-prefix widgetsnbextension
```

Verify you successfully installed sparkmagic by running:

```
conda list sparkmagic
```

If installed, you should see something like:

```
# packages in environment at <path>/anaconda3/envs/sparkmagic:
#
# Name                    Version                   Build  Channel
sparkmagic                0.20.0          py310h06a4308_1
```

Once installed, you can change directory to where you want your notebook to be stored. Then launch Jupyter notebook from the same terminal session as normal by running:

```
cd <path to directory where notebook should be created>
jupyter notebook
```

Open your browser using the URL displayed in the terminal, e.g. http://localhost:8888/?token=a50993cb8be4bf67b1db01cb35f93cc15842332655d788aa

Now we need to connect to the cluster using SSH before we can run Python code.

CONNECTING THROUGH SSH WITH PORT FORWARDING To make your notebook "send" your Python code to be executed on ITU's Spark cluster, Jupyter notebook needs to send it to the cluster's "Livy" service. Apache Livy is a REST service for submitting jobs to a remote Spark cluster. In our cluster it is running on port 8999. In order to access it, you need to forward that port to your localhost using ssh(additional info how that works). Once your user has been setup by a TA on the cluster.

Open a *new* terminal window, activate your "sparkmagic" conda enviroment (if you created one earlier).

```
conda activate sparkmagicEnv
```

You should be able to access the cluster using this ssh command (replace `abcd` with your ITU-username):

```
 ssh abcd@130.226.142.166 -p 8022 -L 8998:localhost:8999
```

Depending on how you generated your public ssh key, you might need to specify the path to your private key.[1] This can be done with the `-i` option:

```
 ssh abcd@130.226.142.166 -p 8022  -L 8998:localhost:8999 -i /full_path/to
   /public_key_file
```

For example:

```
 ssh abcd@130.226.142.166 -p 8022  -L 8998:localhost:8999 -i ~/.ssh/abcd
```

NOTE: Do not write *-i /.ssh/abcd.pub*

After this you should arrive at the `ambari0` machine. Anything you write in this terminal will be commands exectuted on the abari0 machine. If you are using a Jupyter notebook setup, you do not need to write anything here. When you are done working for the day, write "exit" to disconnect from the cluster.

Verify that the ssh port forwarding worked by opening your browser. You should now see the Livy UI if you go to http://localhost:8998/ui. It can be used to keep track of the code you send to the cluster.

DON'T SEE WITH LIVY UI? If you don't see the Livy UI, after successfully SSH'ing into the cluster, disconnect from the cluster with "exit".

While still in the new terminal, try replacing the port "8998" above with something else, like "8765". Then edit the file located at ~/.sparkmagic/config.json using a text editor. (On windows: it's located probably in your own user folder, eg C:\Users\thorv). First we check if the config file exists by changing into the directory:

```
cd ~/.sparkmagic/
ls
```

You should see "config.json logs"

Edit the file with eg nano or another text editor.

```
nano config.json
```

Input this exact config. Beware of copying from the PDF; it may copy unintended characters.

[1] This is the other part of your key-pair, where you sent the private key to a TA

```
  {
  "kernel_python_credentials" : {
    "username": "",
    "password": "",
    "url": "http://localhost:8765",
    "auth": "None"
  }
}
```

Save and close the file by clicking CTRL + O then ENTER then CTRL + X

Restart the "PySpark" kernel in the Jupyter notebook. Try SSH again with the new port:

```
ssh abcd@130.226.142.166 -p 8022  -L 8765:localhost:8999 -i /full_path/to
  /public_key_file
```

Verify that the ssh port forwarding worked by opening your browser. You should now see the Livy UI if you go to http://localhost:8765/ui. It can be used to keep track of the code you send to the cluster.

If that does not work, something on your local machine may block access to the port "8765" or "8998". Try disabling your anti-virus firewall, or allowing local traffic for that port in your firewall.

If you still cannot get Livy to work, try below section *"Installing the Sparkmagic Jupyter notebook extension using Docker (advanced users)"*

CREATING A JUPYTER NOTEBOOK USING THE PYSPARK KERNEL In http://localhost:8888/, Click **New** on the top right corner, and then click **PySpark**. This should open a new notebook with the kernel you selected. Verify that by checking if it says "PySpark" at the top right in the notebook editor. When using the PySpark kernel instead of Python 3, we tell Jupyter to execute code on the cluster (through Livy) instead of running it on your laptop.

All students can view which applications are running on the cluster, and how many resources people use at the cluster scheduler overview.

See LearnIT for an example notebook that shows how to run code on the cluster.

INSTALLING THE SPARKMAGIC JUPYTER NOTEBOOK EXTENSION USING DOCKER (ADVANCED USERS) From experience, you may have issues with virtual environments. Thus it may be easier to use a Docker container instead to install everything you need. Using the sparkmagic-notebook docker image by fabiobatsilva, we can create a connection using this configuration:

```
Docker run -p 8888:8888 -v "$(pwd)":/home/jovyan/work -e
  SPARKMAGIC_CONFIG_CREDENTIALS_URL='http://127.0.0.1:8999' -e
  SPARKMAGIC_CONFIG_SESSION_CONFIG='{"driverMemory":"512M","numExecutors
  ":2,"executorMemory":"1G","proxyUser":"INSERT_YOUR_USERNAME_HERE"}' --
  network host fabiobatsilva/sparkmagic-notebook
```

NOTE: This command will bind your current working directory to the container, so it will only be those files that are available.
NOTE: Remember to insert your user name in the command above.
NOTE: You can alter the configs in the command above to get more resources for your spark job.
NOTE: You may need to change the ports when using SSH to these: *ssh abcd@130.226.142.166 -p 8022 -L 8999:localhost:8999*

TL;DR

1. Forward port 8999 to your localhost

2. Install sparkmagic on your local machine or start Docker image
3. Access the notebooks in your browser.

2.2.2 Note on accessing the Cluster through SSH

You will need to `ssh` onto the machine. Actually the machines are only accessible through a portal machine which can be reach at ip `130.226.142.166`. Normally we use port 22 for ssh, but as port 22 is already used on the portal machine (by its own ssh service), we will instead use port 8022, which the portal machine will forward to the NameNode of the cluster (`ambari0`).

Due to this setup (that the cluster is behind a proxy) many of the URL forwardings you might encounter will not work. This is not an important issue and you can safely ignore this.

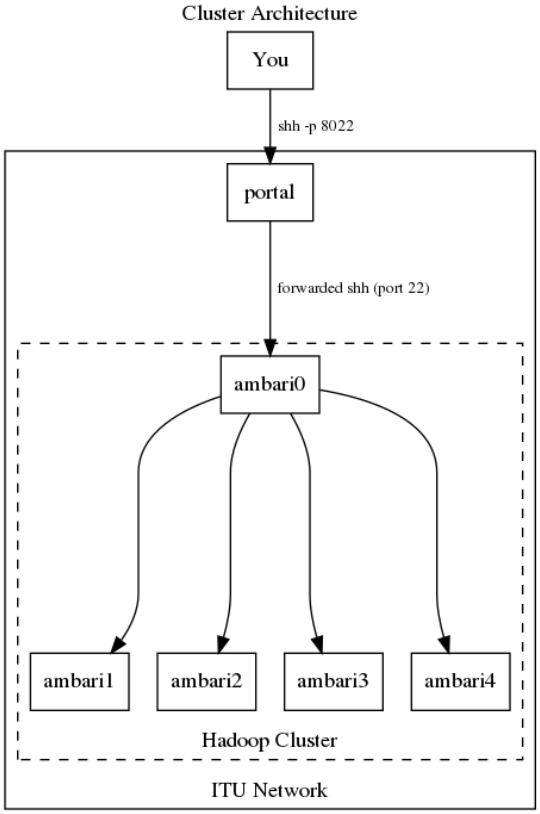

Figure 2.1: Cluster access architecture

2.2.3 Editing remote files (Advanced users)

If you are not used to working with ssh and python directly from the terminal, you should instead follow the above guide, to access the cluster through a Jupyter Notebook.

You may need to edit files that are on the remote machine, which can be done in a couple of ways. You can either set up a ssh connection though your local editor (e.g. VS Code[2] or Atom[3]), or you can work with the command-line editor that is already on the remote machine.[4]

I find that the VS Code SSH extension works very well. It will forward a lot of unnecessary ports for you, but you can ignore that.

[2] https://code.visualstudio.com/docs/remote/ssh-tutorial

[3] https://atom.io/packages/remote-ftp

[4] One of the editors you can almost always find on linux machines is `vim`. If you're new to it, vim is really weird, but once you've worked with it a bit, it is really cool and powerful. The MIT "missing" course has a nice intro to vim as well.

2.2.4 Running Spark interactively (Advanced users)

Once you're on `ambari0` (the NameNode), you probably need to run the below command (in bash), otherwise some review text can't be printed

```
export PYTHONIOENCODING=utf8
```

If terminal session is killed by remote host, you need to run above command again.

Then you can run `pyspark` to access a Python REPL shell, where the Spark context is available.[5] The Spark context is the entry point for the Spark API.

By default the PySpark shell will be allocated a modest amount of resources on the cluster, but you can specify this through the commandline options.

```
--name NAME                 A name of your application.
--conf PROP=VALUE           Arbitrary Spark configuration property.
--driver-memory MEM         Memory for driver (e.g. 1000M, 2G) (Default:
    1024M).
--executor-memory MEM       Memory per executor (e.g. 1000M, 2G) (Default:
    1G).
--driver-cores NUM          Number of cores used by the driver, only in
    cluster mode (Default: 1).
--executor-cores NUM        Number of cores per executor. (Default: 1)
--num-executors NUM         Number of executors to launch (Default: 2).
```

For example if you want a pyspark session with 4 executors, with 4 gigabytes of memory each, you would write:

```
pyspark --executor-memory 4G --num-executors 4
```

Instead of typing everything into the pyspark shell every time, you probably want to have your scripts in files. A quick way to run your script file in the pyspark shell is to *feed it to the standard input*.[6]

```
pyspark < your_script_file.py
```

While not the recommended approach to launch real applications,[7] the upside to this approach is that you do not get a swarm of logging messages. The printed output you get is the same as you would get if you inserted each line into the pyspark shell. You do need to initialize the SparkSession in your file, though. An example of how to do this is in the CODE TEMPLATE at the end of the document.

There is a significant overhead when launching Spark applications, as Spark is designed for high throughput, not low latency. So doing your experimentation in the pyspark shell is often a good idea.[8]

3 RESOURCES

3.1 The "Missing Semester"

For this assignment you will do practical work on a remote Linux machine. This is extremely common, as our laptops are not powerful enough for large data, and Linux is the de facto standard for servers. This requires you to work on the command line, use ssh to log-in to the machine remotely, and navigate in a Linux environment. These skills are often not directly taught at universities, but instead assumed that students can figure out on their own.

[5] You might want to consider using `pyspark -conf spark.ui.enabled=false`. You will not be able to access the ui anyway, and by disabling it Spark will not try out a bunch of ports before it starts

[6] Theres an example of such a script file at the end of this document

[7] For larger scale or long running applications, you would typically use `spark-submit`

[8] If you want to save your commands from the pyspark shell, you can export your history to a local file.

The student-run MIT course The Missing Semester of Your CS Education is a great intro to many of these topics. For this assignment the lectures that are especially relevant are:

- The Shell
- Remote Machines (SSH)

3.2 Suggested reading and useful links

- Spark 2.3.1 Quick Start (Remember to choose the python tab)
- Spark SQL Programming Guide (Much more detail)
- Our cluster scheduler overview (See how many resources are free on the cluster)
- Architecture Overview of Cluster-based Spark
- The PySpark API Documentation for version 2.3.1 (Remember to always use the documentation for this version)
  Especially these modules will be useful to you:
  - DataFrame
  - Column
  - Spark SQL Functions
- A Tale of Three Apache Spark APIs: RDDs vs DataFrames and Datasets

4 Script file template (advanced users)

Use the below code if you intend to create python scripts to execute your code on the cluster, rather than using notebooks.

```
import pyspark
from pyspark.sql import SparkSession
from pyspark import SparkConf

# You can alter these configurations
conf = SparkConf()
conf.set("spark.executor.memory", "1G")
conf.set("spark.executor.instances", "2")
# And you can add other configuration options here.

# You should change the appName
spark = SparkSession.builder \
                    .appName('template-application') \
                    .config(conf=conf) \
                    .getOrCreate()


# Insert your own code here

##########################

## EXAMPLE
# Read the csv-file from HDFS
df = spark.read\
    .option("header",True)\
    .csv("/datasets/retail/retail.csv")

```

```
28 # Show the first 20 rows as text
29 df.show()
```