

Big Data Applications

Contents

- [Linux](#)

Introduction

Pre-requisites

- Be comfortable with basic Python. Familiarity with Numpy and Pandas is beneficial but not required.
- Have an environment ready for the several hands-on exercises (e.g., own laptop or cloud resource)

Syllabus

Something broken?

If you encounter any problem or bug in these materials, please remember to add an issue to the [course repo](#), explaining the problem and, potentially, its solution. By doing this, you will improve the instructions for future users. 

Installation Instructions

Introduction

This document contains instructions for installation of the software we'll be using during the course. If you want to follow the training on your own machines, then please complete these instructions.

If you encounter any problem during installation and you manage to solve them, please remember to add an issue, explaining the problem and solution. By doing this you will be helping to improve the instructions for future users! :tada:

What we're installing

- the [Python](#) programming language (version 3.7 or greater)
- [git](#) for version control
- your favourite text editor
- [Apache Spark](#)
- [Jupyter notebooks](#)

Please ensure that you have a computer or a cloud-based workbench with all of these installed.

Linux

Package Manager

[Linux](#) users should be able to use their package manager to install all of the software from this page.

However note that if you are running an older [Linux](#) distribution you may get older versions with different look and features. We target [Ubuntu 21.10](#) for the sake of homogeneity.

💡 Tip

We recommend to upgrade the system and get the latest security packages: `sudo apt-get upgrade`

Please do not forget to restart after the upgrade finishes.

You can test the [Python](#) version by issuing:

```
python3 --version
```

You should get an output similar to the following:

```
Python 3.9.7
```

Python via package manager

Recent versions of [Ubuntu](#) come with mostly up to date versions of all needed packages.

The version of [IPython](#) might be slightly out of date. Thus, you may wish to upgrade this using [pip](#).

You should ensure that the following packages are installed using [apt-get](#):

- `python3-pip`
- `jupyter`
- `ipython3`

```
sudo apt-update  
sudo apt-get install -y gcc make perl python3-pip jupyter ipython3
```

You can test the installation by issuing:

```
pip --version
```

You should get an output similar to the following:

```
pip 20.3.4 from /usr/lib/python3/dist-packages/pip (python 3.9)
```

Editor

You have many different text editors suitable for programming at your fingertips. Here is an opinionated list of editors in case you do not already have a favourite:

- [Visual Studio Code](#)
- [Atom](#)
- [Neovim](#)
- [Vim](#)

Apache Spark

We will setup [Apache Spark](#) now. You need to open the [downloads](#) page, and download a spark distribution. We suggest to choose the same options as on the screenshot below. If you see a newer version is available, please feel free to choose it.

Download Apache Spark™

1. Choose a Spark release: [3.2.1 \(Jan 26 2022\) ▾](#)
2. Choose a package type: [Pre-built for Apache Hadoop 3.3 and later](#)
3. Download Spark: [spark-3.2.1-bin-hadoop3.2.tgz](#)
4. Verify this release using the [3.2.1 signatures](#), [checksums](#) and [project release KEYS](#).

Note that Spark 3 is pre-built with Scala 2.12 in general and Spark 3.2+ provides additional pre-built distribution with Scala 2.13.

... [Image Link](#)

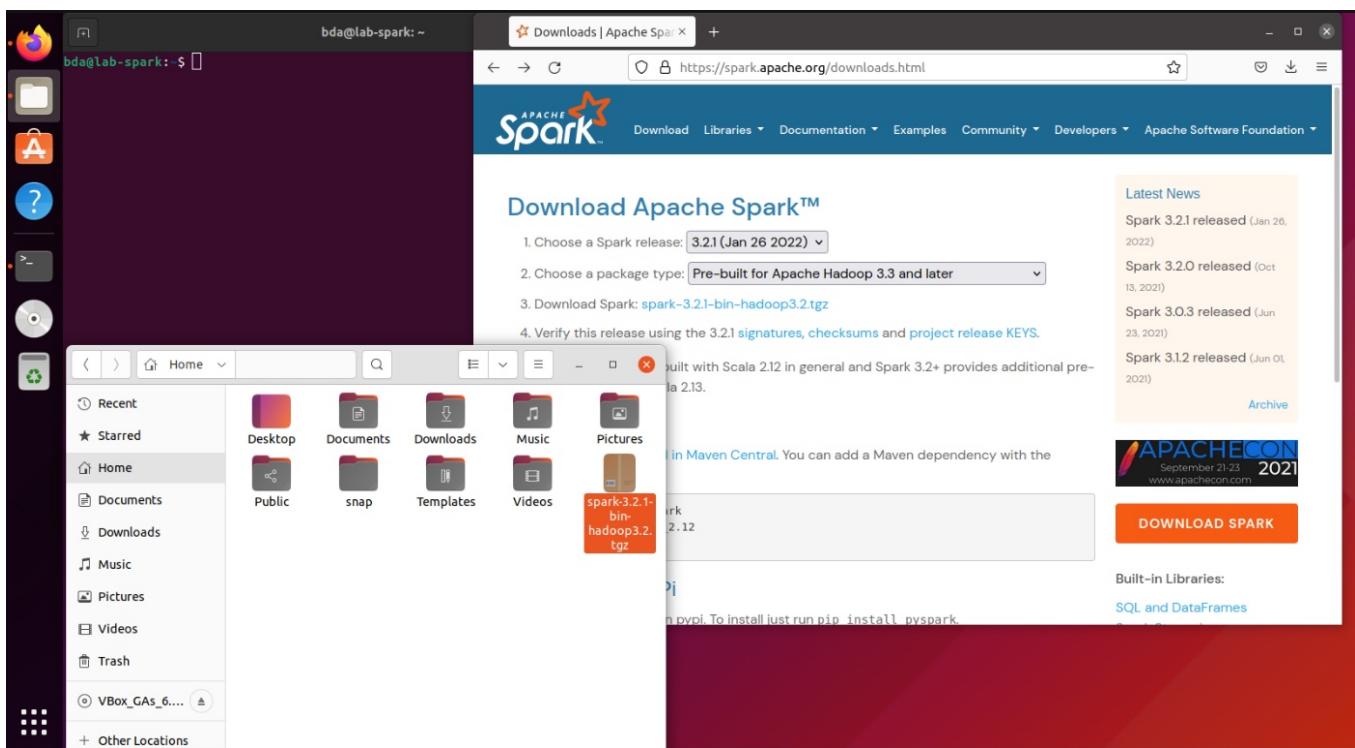
Latest News

- [Spark 3.2.1 released \(Jan 26, 2022\)](#)
- [Spark 3.2.0 released \(Oct 13, 2021\)](#)
- [Spark 3.0.3 released \(Jun 23, 2021\)](#)
- [Spark 3.1.2 released \(Jun 01, 2021\)](#)

[Archive](#)

💡 Tip

We want the package in the right location, so open the file explorer and place it into your home folder.



Then go to your command line and issue the following to unzip the downloaded file:

```
sudo tar -xvf spark-3.2.1-bin-hadoop3.2.tgz
```

❗ Attention

The above command assumes you downloaded version 3.2.1 with hadoop 3.2 binaries. Please amend it as needed.

Now what we need to do is telling [Python](#) how to find Spark:

```
echo "export SPARK_HOME=~/spark-3.2.1-bin-hadoop3.2" >> ~/.bashrc
echo "export PATH=$SPARK_HOME:$PATH" >> ~/.bashrc
echo "export PYTHONPATH=$SPARK_HOME/python:$PYTHONPATH" >> ~/.bashrc
echo "export PYSPARK_DRIVER_PYTHON=jupyter" >> ~/.bashrc
echo "export PYSPARK_DRIVER_PYTHON_OPTS=notebook" >> ~/.bashrc
echo "export PYSPARK_PYTHON=python3" >> ~/.bashrc
```

! Important

Please do not forget to reload your interactive shell session. [1]

[1] <https://www.delftstack.com/howto/linux/reload-bashrc/>

Introduction to RDD

In this first session, we will introduce what is a [Resilient Distributed Dataset \(RDD\)](#). Please refer to the original paper for in-depth details [1].

To understand how Spark works, we need to understand the essence of RDD. Long story short, an RDD **represents a fault-tolerant collection of elements partitioned across the nodes of a cluster that can be operated in parallel**. We will chop the last sentence into pieces now.

Processing Logic Expressed as RDD Operations

We use RDDs to express a certain processing logic we want to apply to a dataset. For example, finding the average number of days required to recover from a certain disease.

Then, Spark makes its magic to **schedule** and **execute** our data processing logic on a distributed fashion.

If we look behind the scenes, the Spark runtime requires to determine the order of execution of RDDs, and make sure the execution is fault-tolerant. It uses the following pieces of information for the aforementioned purposes:

- Lineage
 - Dependencies on parent RDDs
- Fault-tolerance
 - The partitions that makes the whole dataset: used to execute in **parallel** to speed up the computation with **executors**.
 - The function for computing all the rows in the dataset: provided by users (i.e., “you”). Each **executor** in the cluster execute this function against each row in each partition.

With the above information, Spark can reproduce the RDD in case of failure scenarios.

Quick Demo

Prepare a Jupyter Kernel

You will need to prepare a Jupyter kernel in order to run this notebook on your local environment.

i Note

This course relies on [Poetry](#) as package manager. Commands below can might slightly differ if you are using a different strategy (e.g., [Pipenv](#) or [Conda](#), just to mention a few).

```
poetry shell
ipython kernel install --name "bda-labs" --user
```

The output from above command is a JSON file located under your Jupyter installation. For example:

/path_to_jupyter_installation/kernels/bda-labs/kernel.json

Its content will be similar to the following:

```
{
  "argv": [
    "<some_path_here>/bda-course/.venv/bin/python",
    "-m",
    "ipykernel_launcher",
    "-f",
    "{connection_file}"
  ],
  "display_name": "bda-labs",
  "language": "python",
  "metadata": {
    "debugger": true
  }
}
```

Initialize Spark

We need to do some preparation first. In the next snippet, we will import the [SparkContext](#) and [SparkConf](#) classes.

```
from pyspark import SparkContext, SparkConf
```

Now we proceed to create the **configuration** for our application and a **context** which tells Spark how to access the execution environment (local or a cluster).

```
# we use "local" to indicate we're running in local mode
conf = SparkConf().setAppName("into-to-rdd").setMaster("local")
sc = SparkContext(conf=conf)
```

```
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.apache.spark.unsafe.Platform
(file:/opt/hostedtoolcache/Python/3.9.10/x64/lib/python3.9/site-
packages/pyspark/jars/spark-unsafe_2.12-3.2.1.jar) to constructor
java.nio.DirectByteBuffer(long,int)
WARNING: Please consider reporting this to the maintainers of
org.apache.spark.unsafe.Platform
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective
access operations
WARNING: All illegal access operations will be denied in a future release
```

```
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use
setLogLevel(newLevel).
```

```
22/02/22 19:57:23 WARN NativeCodeLoader: Unable to load native-hadoop library for
your platform... using builtin-java classes where applicable
```

Define an RDD

Let's start by defining a very simple RDD. We can think of it as data points indicating the recovery days for a certain disease.

```
recovery_days_per_disease = [("disease_1", [3, 6, 9, 10]), ("disease_2", [11, 11,
10, 9])]
rdd = sc.parallelize(recovery_days_per_disease)
```

Now we can operate on it. For example, we can compute the average.

```
rdd.mapValues(lambda x: sum(x) / len(x)).collect()
```

```
[Stage 0:> (0 + 1) / 1]
```

```
[('disease_1', 7.0), ('disease_2', 10.25)]
```

References

- [1] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauly, Michael J Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, 15–28. 2012.
-

By Carlos Montemuiño

© Copyright 2022.