ILLINOIS DATA SCIENCE INITIATIVE

TECHNICAL REPORTS

PySpark on Python 3: Configuration and Package Management Guide

Author:
Benjamin Congdon

March 7, 2017

PySpark on Python 3: Configuration and Package Management Guide

BENJAMIN CONGDON[1] AND PROFESSOR ROBERT J. BRUNNER[2]

[1] *National Center For Supercomputing Applications (NCSA)*
[2] *Laboratory for Computation, Data, and Machine Learning*

Compiled March 7, 2017

PySpark requires requires Python 2.6 or later, and does not officially support Python 3. However, with only minor configuration changes, it is possible to successfully run a Spark cluster using Python 3 for both PySpark drivers and workers.

https://github.com/lcdm-uiuc

1. INTRODUCTION

This technical report will describe a process for upgrading a Spark Cluster to operate with Python 3 as the version of Python used on worker instances and the PySpark driver. Additionally, this report will suggest a maintainable method for installing and upgrading a set of Python packages on a Spark Cluster for use in PySpark jobs.

2. ASSUMPTIONS

This technical report will make the following assumptions:

- The implementer already has a working Spark Cluster with a CentOS worker base image.
- Ideally, the Spark Cluster is managed by Apache Ambari. (This report will still contain valid information for non-Ambari-managed clusters, but the configuration files will have to be manually adjusted on every node).

3. MOTIVATION

While Python 3 is not yet the *de facto* standard in the Python community, the trend seams to be moving in the direction of Python 3 adoption. Thus, it behooves members of the data science community to begin to transition to Python 3 for cloud computing.

Additionally, we observed that running the default Python version that shipped with our CentOS image, Python 2.6.6, had many more issues being installed concurrently with Python 2.7 than any version of Python 3. PySpark requires Python 2.7+, and Ambari requires Python 2.6.6. As a result, we found that a more stable equilibrium was met while running our PySpark workers on Python 3 and leaving the Python 2.* installation on the workers undisturbed.

Furthermore, we researched the available solutions for maintaining and distributing Python packages across our cluster, and felt that the built-in mechanisms for distributing packages at runtime (i.e. the `--py-files` argument for spark-submit) to be lacking. Not only does this require the operator to have built versions of all the packages they wish to include in their job, but it also means that using packages like `nltk`, which download external datasets, are difficult to include in jobs.

4. INSTALLING PYTHON 3

To run the Spark Cluster on top of Python 3, an identical version Python 3 should be installed on all data nodes, and anywhere that a driver will be run.

Usually, this install process is rather simple: use the package manager for your OS to install the current Python 3 package.

```
# Ubuntu, Debian
$ apt-get install python3

# RHEL, CentOS
$ yum install python3
```

The complexity comes in the fact that you'll have to install Python 3 on all the nodes in your cluster. While this can be accomplished in a variety of ways, we suggest that you use Fabric to perform the operation. A link to the Fabfile used to install Python 3 is included in the references of this technical report.

5. CONFIGURING PYSPARK TO RUN PYTHON 3

There are 2 primary configuration settings necessary to configure PySpark to work with Python 3: `PYSPARK_PYTHON` and `PYSPARK_DRIVER_PYTHON`. The former dictates the path to the Python executable used for the workers, and the latter dictates the path to the Python executable used for the client driver, when applicable.

When running in a *client* mode, these variables can be set as environment variables on the node being used as the client driver. However, when PySpark is used in a *cluster* mode (such as *yarn-cluster*), this setting must be configured in the Spark configuration.

There is one additional complication due to the way that Python 3 handles hashing. Python 3 introduces randomness into its hashes to prevent hashing DoS attacks. However, in our

parallel computing, we need nodes to agree on the hash seed. Thus, we must also configure `PYTHONHASHSEED` to be identical on all nodes in the cluster.

To make the relevant changes to your Spark configuration, go to the Ambari Dashboard and navigate to *Services* > *Spark* > *Configs*.

Add the following entries to *spark-env template*:

```
export PYSPARK_PYTHON=/usr/bin/python3
export PYSPARK_DRIVER_PYTHON=/usr/bin/python3
export PYTHONHASHSEED=123
```

Add the following entries to *Custom spark-defaults*

- spark.executorEnv.PYTHONHASHSEED → 0
- spark.pyspark.driver.python → /usr/bin/python3
- spark.pyspark.python → /usr/bin/python3
- spark.yarn.appMasterEnv.PYSPARK_DRIVER_PYTHON → /usr/bin/python3
- spark.yarn.appMasterEnv.PYSPARK_PYTHON → /usr/bin/python3

Note that `/usr/bin/python3` is the default install location for most Python 3 installations (including the CentOS package). However, if you have Python 3 installed at a different location, you will have to adjust this path as necessary.

Once these changes have been made, save the configuration and restart all the affected nodes.

6. INSTALLING AND UPDATING PACKAGES

Python maintains a store of system *site-packages*. To use these packages in your PySpark jobs, each package you want to import will have to be installed on all nodes. As Python 3 will have a separate set of installed packages than Python 2, you will need to reinstall all downloaded packages following the Python 3 upgrade.

Again, we recommend using Fabric to manage Python package distributions. You can find a link to a Fabfile that installs packages to cluster nodes in the references of this technical report.

We have found it useful to maintain a `requirements.txt` file for the packages on the cluster. A `requirements.txt` file maintains a list of packages installed, and the current installed version. This is similar to `Gemfile.lock` for Ruby Gems. A typical `requirements.txt` file looks similar to this:

```
appdirs==1.4.0
beautifulsoup4==4.4.0
nose==1.3.7
packaging==16.8
pyparsing==2.1.10
requests==2.5.0
six==1.10.0
```

Keeping package versions consistent across nodes is important, as is assuring all required packages are installed on all nodes. Failure to do so will lead to unpredictable behavior and the possibility of Spark job failure.

A. Installing New Packages

When a cluster user wants to make add a package to the cluster, have them add the package and desired version to the cluster's `requirements.txt`. The installed version of a package can be obtained with this command:

```
$ pip freeze | grep <Package>
```

For example, we can search for the latest version of `beautifulsoup4`:

```
$ pip freeze | grep beautifulsoup4
beautifulsoup4==4.4.0
```

Once the cluster's `requirements.txt` has been updated, run the package installation Fabfile task to update the *site-packages* of all the nodes in the cluster.

If you are using the Fabfile distribution that comes with this technical report, run the following command:

```
$ fab -f package_management.py install_requirements
```

B. Updating Packages

Updating packages can be done manually, by changing the version required in the cluster's `requirements.txt`.

To do this in a clean way without disturbing the system's package, we will use `virtualenv` to create a new package environment. Running `virtualenv` creates a folder that contains the binary distribution of Python, Pip, and other tools. It also creates a new location within the virtual environment folder for packagse. `virtualenv` is a useful tool for testing Python environments without affecting the environment at the system level.

To update all packages automatically, we run `pip` in a Python `virtualenv` with its update flag to pull down the latest version numbers. (Note that this requires `virtualenv` to be installed.)

```
$ virtualenv venv
$ source venv/bin/activate
$ pip install -r requirements.txt --upgrade
$ pip freeze > requirements.txt
$ deactivate
$ rm -r venv/
```

Once the `requirements.txt` has been updated, run the install Fabric task again and the nodes will be updated similarly.

Updating the `requirements.txt` in this fashion can easily be automated using Fabric. If you are using our Fabfile distribution, run the following command:

```
$ fab -f package_management.py update_requirements
```

This Fabric task only updates the local `requirements.txt`, and does not update packages on cluster hosts. To do this, you will need to rerun the `install_requirements` task.

7. CONCLUSION

REFERENCES

- For information about setting up Fabric, refer to the *"Introduction to Fabric for Cluster Management"* technical report.
- Fabric tasks for installing Python can be found in the included `python3_install.py` fabfile
- Fabric tasks for installing and updating Python packages can be found in the included `package_management.py` fabfile