Setting up a Python Project with Virtual Environment, PyBuilder, and PyCharm

## Abstract

Our goal for this article is to setup a tool-chain that builds Python “libraries” deployable to the *Databricks Community Edition* version of PySpark. We introduce *PyBuilder*, a Maven-like Python open source build tool, which should work well for Java programmers building Python components. Java programmers tend to have smaller files and more components than Python programmers who use “modules”, like C++. A Python module often contains several Python class definitions, as opposed to a Java class file which defines a single public class (but may include inner classes and package level classes.) We also setup the *PyCharm Community Edition*, a popular free Python IDE that well supports Python’s *virtual environment* mechanism.

This article is a part of a series of articles discussing Python modularization and dependency management practices for the Java programmer (see article: <http://www.tbd.com>.) A Windows 10 development environment is used for the article examples, but *NIX* environments are well documented and the steps are almost identical for those systems. Sadly, Python is not as machine independent as Java, and does not have as strong a backwards compatibility commitment as does Java. This means setting up these Python environments will change over time.

Discussed here are creating a *virtual environment* with the Python utility ***venv***, and completing the project structure using *PyBuilder*. A virtual environment is the primary dependency control mechanism for Python. Finally, we wrap the new project with the *PyCharm* Python IDE. Documentation resources are included in the reference section at the end of this article.

. We cover these steps for our Python library project creation:

* Create a Python project (and package) directory ***myapppy***.
* Create a Virtual Environment in ***myapppy*** (for Python interpreter-based dependency isolation.)
* Create a *PyBuilder* project for ***myapppy***.
* Create a *PyCharm Community Edition* IDE project for ***myapppy***.
* Add Python Source and Test files for sample library modules.
* Create deployable “binary” library using *PyBuilder*.
* Deploy the newly created library locally and verify.
* Deploy the newly created library to Databricks Community Edition and verify.

## Create the *myapppy* Project Directory

You will require access to the GitHub repository to repeat the steps outlined in this article. Please see reference #1 in the resources section at the end of the document. Our first step is to create a package directory for our test project and name it ***myapppy.*** We then create a *Virtual Environment* for our ***myapppy*** project as well:

D:\\Dependencies\myapppy>***python -m venv venv***

D:\\Dependencies\myapppy>***tree venv***

D:\\DEPENDENCIES\MYAPPPY\VENV *(abbreviated content!!)*

├───Include

├───Lib

│ └───site-packages

│ ├───pip

│ ├───pip-19.2.3.dist-info

│ ├───pkg\_resources

│ ├───setuptools

│ ├───setuptools-41.2.0.dist-info

└───Scripts

## Create a PyBuilder project for myapppy

Next, we setup a *PyBuilder* instance for our project by using the script ***loadPyBuilder.cmd***. After running the load script from directory ***myapppy***, the results are:

Directory of D:\\Dependencies\myapppy\venv\Scripts

10/21/2019 04:53 PM <DIR> .

10/21/2019 04:53 PM <DIR> ..

10/21/2019 04:48 PM 2,345 activate

10/21/2019 04:48 PM 1,022 activate.bat

10/21/2019 04:48 PM 1,553 Activate.ps1

10/21/2019 04:48 PM 368 deactivate.bat

10/21/2019 04:48 PM 98,235 easy\_install-3.7.exe

10/21/2019 04:48 PM 98,235 easy\_install.exe

10/21/2019 04:53 PM 103,342 pip.exe

10/21/2019 04:53 PM 103,342 pip3.7.exe

10/21/2019 04:53 PM 103,342 pip3.exe

10/21/2019 04:48 PM 886 pyb

10/21/2019 04:48 PM 98,217 pyb\_.exe

10/21/2019 04:48 PM 98,210 pytail.exe

10/21/2019 04:47 PM 522,768 python.exe

10/21/2019 04:47 PM 522,256 pythonw.exe

10/21/2019 04:48 PM 98,213 wheel.exe

We are now ready to install the *PyBuilder* dependencies, but first we add a ***builder.py*** bootstrap file obtained from the *PyBuilder* project’s GitHub repository (see reference #1.) This special build file “bootstraps” the *PyBuilder* installation. We are now able to create a *PyBuilder* environment for our project using steps recorded in file ***loadPyBuilder.cmd*** used to boot-strap *PyBuilder*:

D:\\Dependencies\myapppy>***installDependenciesPyBuilder.cmd > installDependenciesPyBuilder.log***

*PyBuilder* now has dependencies installed and has added a utility (***pygmentize.exe***). There are external dependencies that need to be added into *PyBuilder* as well, so use the script ***installExternalDependenciesPyBuilder.cmd*** to load them into the ***venv*** environment. We now create our directories and basic PyBuilder project infrastructure, first deleting the master ***build.py*** file:

D:\\Dependencies\myapppy>***del build.py***

D:\\Dependencies\myapppy>***venv\Scripts\activate.bat***

(venv) \d:\\dev-topics-dependencies\Dependencies\myapppy>pyb\_ --start-project

Project name (default: 'myapppy') :

Source directory (default: 'src/main/python') :

Docs directory (default: 'docs') :

Unittest directory (default: 'src/unittest/python') :

Scripts directory (default: 'src/main/scripts') :

Use plugin python.flake8 (Y/n)? (default: 'y') :

Use plugin python.coverage (Y/n)? (default: 'y') :

Use plugin python.distutils (Y/n)? (default: 'y') :

Created 'setup.py'.

This initial run of ***PyBuilder*** creates the ***setup.py*** and ***build.py*** files, along with the ***src, target*** and ***docs*** directories. The newly created ***build.py*** should look something like this:

from pybuilder.core import use\_plugin, init

use\_plugin("python.core")

use\_plugin("python.unittest")

use\_plugin("python.install\_dependencies")

use\_plugin("python.flake8")

use\_plugin("python.coverage")

use\_plugin("python.distutils")

name = "myapppy"

default\_task = "publish"

@init

def set\_properties(project):

pass

We execute a ***PyBuilder*** “verify” (I.e., Maven “test”) run on the no-source-yet project environment, and we get something like this:

(venv) D:\\Dependencies\myapppy>***pyb\_ verify***

PyBuilder version 0.12.0.dev20190116131423

Build started at 2019-10-21 17:24:08

------------------------------------------------------------

[INFO] Building myapppy version 1.0.dev0

[INFO] Executing build in \D:\\Dependencies\myapppy

[INFO] Going to execute task verify

Package(s) not found: coverage, flake8, pypandoc, twine, unittest-xml-reporting

[INFO] Installing plugin dependency coverage

[INFO] Installing plugin dependency flake8

[INFO] Installing plugin dependency pypandoc

[INFO] Installing plugin dependency twine

[INFO] Installing plugin dependency unittest-xml-reporting

[INFO] Running unit tests

[WARN] Not forking for <function do\_run\_tests at 0x000002B29AF87948> due to Windows incompatibilities (see #184). Measurements (coverage, etc.) might be biased.

[INFO] Executing unit tests from Python modules in \D:\\dependencies\myapppy\src\unittest\python

[WARN] No unit tests executed.

[INFO] All unit tests passed.

[INFO] Building distribution in \D:\\dependencies\myapppy\target\dist\myapppy-1.0.dev0

[INFO] Copying scripts to \D:\\dependencies\myapppy\target\dist\myapppy-1.0.dev0\scripts

[INFO] Writing setup.py as \D:\\dependencies\myapppy\target\dist\myapppy-1.0.dev0\setup.py

[INFO] Collecting coverage information

[WARN] coverage\_branch\_threshold\_warn is 0 and branch coverage will not be checked

[WARN] coverage\_branch\_partial\_threshold\_warn is 0 and partial branch coverage will not be checked

[WARN] Not forking for <function do\_coverage at 0x000002B29AFAF438> due to Windows incompatibilities (see #184). Measurements (coverage, etc.) might be biased.

[INFO] Running unit tests

[INFO] Executing unit tests from Python modules in \D:\\dependencies\myapppy\src\unittest\python

[WARN] No unit tests executed.

[INFO] All unit tests passed.

Coverage.py warning: No data was collected. (no-data-collected)

[INFO] Overall coverage is 100%

[INFO] Overall coverage branch coverage is 100%

[INFO] Overall coverage partial branch coverage is 100%

------------------------------------------------------------

BUILD FAILED - No data to report.

------------------------------------------------------------

Build finished at 2019-10-21 17:24:30

Build took 21 seconds (21780 ms)

As expected, the build failed. We have this directory structure for *PyBuilder*:

D:\\Dependencies\myapppy>dir & tree docs & tree src & tree target

Directory of D:\\Dependencies\myapppy

10/22/2019 02:45 PM <DIR> .

10/22/2019 02:45 PM <DIR> ..

10/21/2019 05:17 PM 339 build.py

10/21/2019 05:17 PM <DIR> docs

10/21/2019 05:07 PM 1,394 installDependenciesPyBuilder.cmd

10/21/2019 05:09 PM 2,176 installDependenciesPyBuilder.log

10/21/2019 04:42 PM 1,057 loadPyBuilder.cmd

10/21/2019 04:53 PM 83,384 loadPyBuilder.log

10/21/2019 05:17 PM 2,527 setup.py

10/21/2019 05:17 PM <DIR> src

10/21/2019 05:24 PM <DIR> target

10/21/2019 04:48 PM <DIR> venv

D:\\DEPENDENCIES\MYAPPPY\SRC

├───main

│ ├───python

│ └───scripts

└───unittest

└───python

D:\\DEPENDENCIES\MYAPPPY\TARGET

├───dist

│ └───myapppy-1.0.dev0

│ └───scripts

├───logs

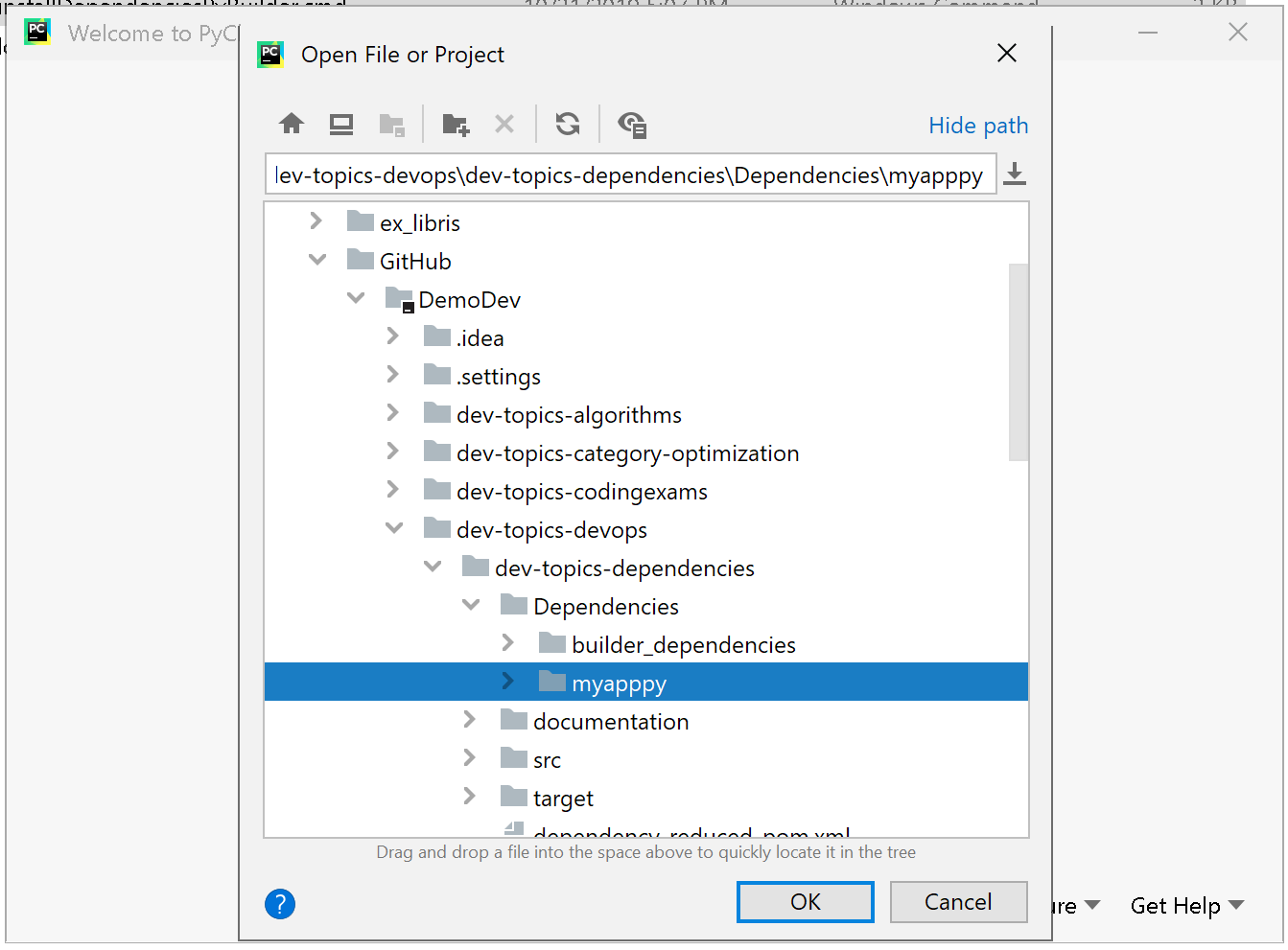
│ └───install\_dependencies

└───reports

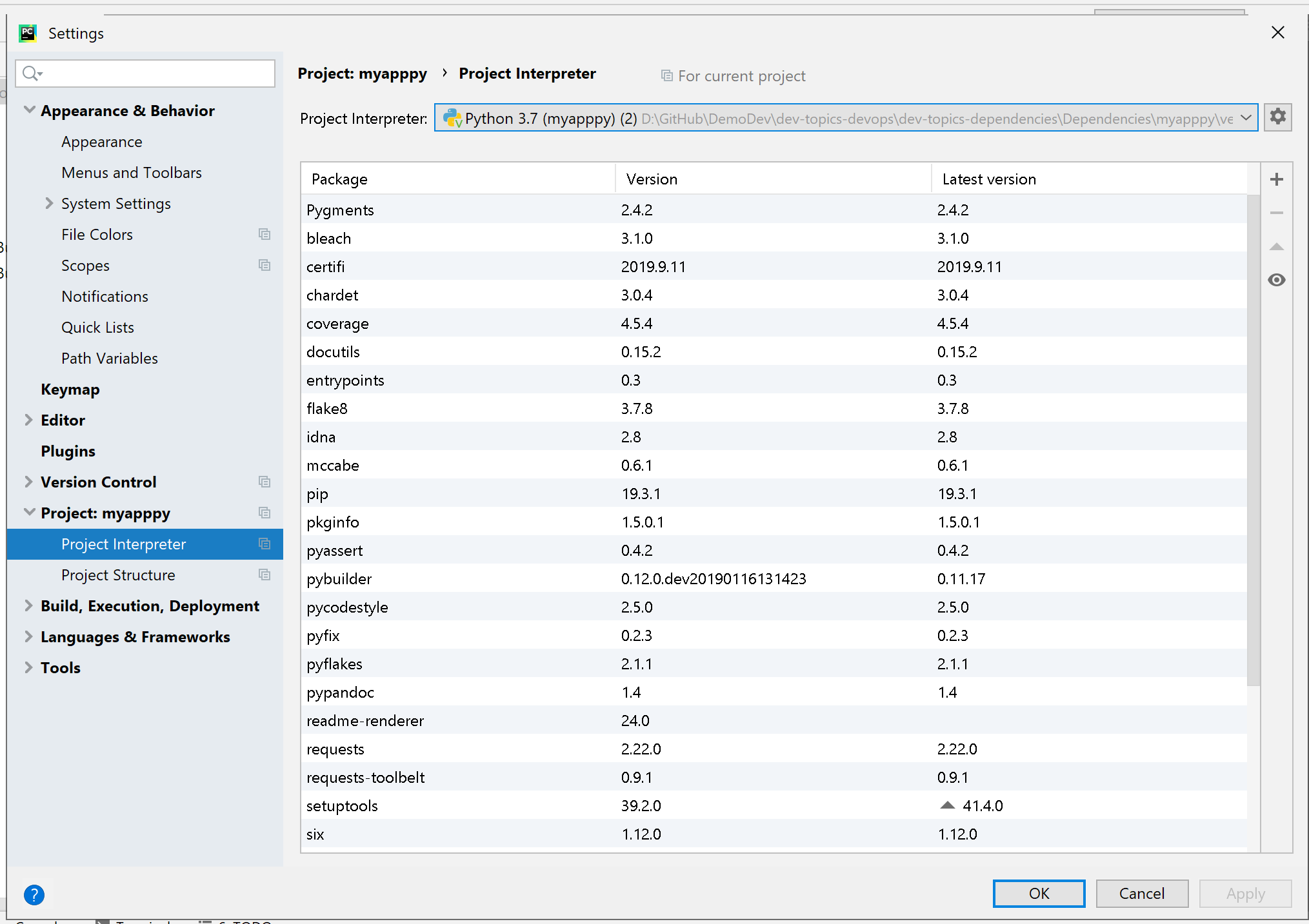
## Create the *PyCharm* Project

Create a *PyCharm Community Edition* Project over the *PyBuilder* Structure using the IDE.

1. Launch the PyCharm IDE.
2. Open, as an existing project, ***myapppy*** (the *myapppy* directory.)

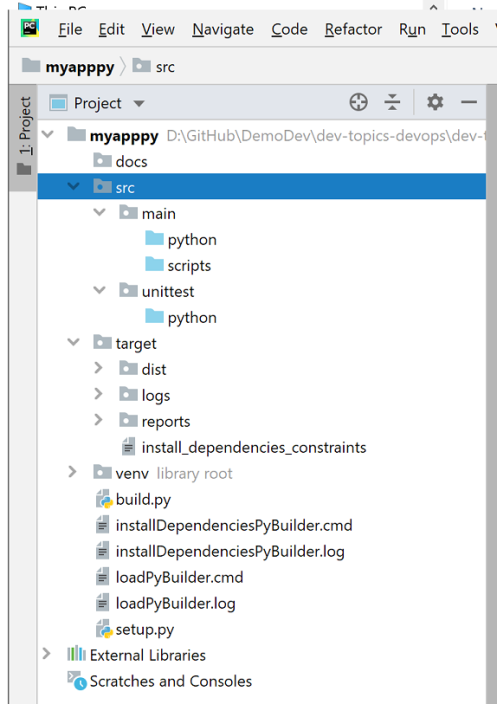


1. Select the virtual environment to associated with the project (***File>Settings>Project Interpreter>Show All>****{select venv}*)



1. Mark source code directories as “source root” (***highlight>right click>Mark as Sources Root***).

The required source directories are show in blue in the diagram below. The source directories are ***src\main\python***, ***src\main\scripts***, ***unittest\python***.



Now synchronize the project, delete compiled Python files, and prepare to add more source files.

## Add Python Source and Test files for sample library modules

We can now add source files for functionality and unit tests. We will refer to the GitHub repository for files and project dependencies (Please see reference #1 in the resources section at the end of the document):

* Update the ***build.py*** file to allow the build to continue regardless of code coverage, and include the *mock* testing utility dependency.
* Add the ***\_\_init\_\_.py*** file under the ***src\main\python\myapppy*** directory (project wide code.)
* Add the ***show\_me.py*** file under the ***src\main\scripts*** directory (standalone main entry for package)
* Add the ***version\_info\_tests.py*** unit test file under the ***scr\unittest\python*** directory.

We again run *PyBuilder* with the *verify* command on this initial project and get:

(venv) D:\\Dependencies\myapppy>***pyb\_ verify***

PyBuilder version 0.12.0.dev20190116131423

Build started at 2019-10-22 17:30:37

------------------------------------------------------------

[INFO] Building myapppy version 1.0.dev0

[INFO] Executing build in D:\\dependencies\myapppy

[INFO] Going to execute task verify

[INFO] Running unit tests

[WARN] Not forking for <function do\_run\_tests at 0x000002A6D24D0558> due to Windows incompatibilities (see #184). Measurements (coverage, etc.) might be biased.

[INFO] Executing unit tests from Python modules in D:\\dependencies\myapppy\src\unittest\python

[INFO] Executed 1 unit tests

[INFO] All unit tests passed.

[INFO] Building distribution in D:\\dependencies\myapppy\target\dist\myapppy-1.0.dev0

[INFO] Copying scripts to D:\\dependencies\myapppy\target\dist\myapppy-1.0.dev0\scripts

[INFO] Writing setup.py as D:\\dependencies\myapppy\target\dist\myapppy-1.0.dev0\setup.py

[INFO] Collecting coverage information

[WARN] coverage\_branch\_threshold\_warn is 0 and branch coverage will not be checked

[WARN] coverage\_branch\_partial\_threshold\_warn is 0 and partial branch coverage will not be checked

[WARN] Not forking for <function do\_coverage at 0x000002A6D25210D8> due to Windows incompatibilities (see #184). Measurements (coverage, etc.) might be biased.

[INFO] Running unit tests

[INFO] Executing unit tests from Python modules in D:\\dependencies\myapppy\src\unittest\python

[INFO] Executed 1 unit tests

[INFO] All unit tests passed.

[WARN] Test coverage below 70% for myapppy: 40%

[WARN] Overall coverage is below 70%: 40%

[INFO] Overall coverage branch coverage is 100%

[INFO] Overall coverage partial branch coverage is 100%

------------------------------------------------------------

BUILD SUCCESSFUL

------------------------------------------------------------

Build Summary

Project: myapppy

Version: 1.0.dev0

Base directory: D:\\dependencies\myapppy

Environments:

Tasks: prepare [859 ms] compile\_sources [0 ms] run\_unit\_tests [86 ms] package [16 ms] run\_integration\_tests [0 ms] verify [1776 ms]

Build finished at 2019-10-22 17:30:40

Build took 2 seconds (2853 ms)

We see that the low code-coverage values are just warnings, and they do not stop the build. Now we add three more source files (***generate.py***, ***fibber.py***, and ***generate\_tests.py***) to complete a deployable test package for use in *Databricks*, and we rerun the build:

D:\\Dependencies\myapppy>***venv\Scripts\activate.bat***

(venv) D:\\Dependencies\myapppy>***pyb\_***

PyBuilder version 0.12.0.dev20190116131423[0m

Build started at 2019-10-23 12:35:10

------------------------------------------------------------

[INFO] Building myapppy version 1.0.dev0

[INFO] Executing build in D:\\Dependencies\myapppy

[INFO] Going to execute task publish

[INFO] Running unit tests

[WARN] Not forking for <function do\_run\_tests at 0x00000230D4E489D8> due to Windows incompatibilities (see #184). Measurements (coverage, etc.) might be biased.

[INFO] Executing unit tests from Python modules in D:\\dependencies\myapppy\src\unittest\python

[INFO] Executed 2 unit tests

[INFO] All unit tests passed.

[INFO] Building distribution in D:\\dependencies\myapppy\target\dist\myapppy-1.0.dev0

[INFO] Copying scripts to D:\\dependencies\myapppy\target\dist\myapppy-1.0.dev0\scripts

[INFO] Writing setup.py as D:\\dependencies\myapppy\target\dist\myapppy-1.0.dev0\setup.py

[INFO] Collecting coverage information

[WARN] coverage\_branch\_threshold\_warn is 0 and branch coverage will not be checked

[WARN] coverage\_branch\_partial\_threshold\_warn is 0 and partial branch coverage will not be checked

[WARN] Not forking for <function do\_coverage at 0x00000230D4E714C8> due to Windows incompatibilities (see #184). Measurements (coverage, etc.) might be biased.

[INFO] Running unit tests

[INFO] Executing unit tests from Python modules in D:\\dependencies\myapppy\src\unittest\python

[INFO] Executed 2 unit tests

[INFO] All unit tests passed.

[WARN] Test coverage below 70% for myapppy: 40%

[WARN] Overall coverage is below 70%: 60%

[INFO] Overall coverage branch coverage is 100%

[INFO] Overall coverage partial branch coverage is 100%

[INFO] ***Building binary distribution in D:\\dependencies\myapppy\target\dist\myapppy-1.0.dev0***

------------------------------------------------------------

BUILD SUCCESSFUL

------------------------------------------------------------

Build Summary

Project: myapppy

Version: 1.0.dev0

Base directory: D:\\Dependencies\myapppy

Environments:

Tasks: prepare [2249 ms] compile\_sources [0 ms] run\_unit\_tests [350 ms] package [47 ms] run\_integration\_tests [0 ms] verify [2267 ms] publish [5556 ms]

Build finished at 2019-10-23 12:35:21

Build took 10 seconds (10509 ms)

Our unit tests were successful, and a ***myapppy*** deployable library was created (see directory ***target\dist\myapppy-1.0.dev0***).

## Deploy the newly created library locally and verify

We create a deployment testing directory with no files and a virtual environment. We next install the binary component for ***myapppy***. Finally, using the script files in the project (***show\_me.py*** and ***fibber.py***), we are able to verify that the ***myapppy*** package was installed. Here is the output:

D:\Temp>mkdir myapptest

D:\Temp>cd myapptest

D:\Temp\myapptest>python -m venv venv

D:\Temp\myapptest>venv\Scripts\activate.bat

(venv) D:\Temp\myapptest>***pip install D:\GitHub\DemoDev\dev-topics-devops\dev-topics-dependencies\Dependencies\myapppy\target\dist\myapppy-1.0.dev0\dist\myapppy-1.0.dev0-py3-none-any.whl***

Processing d:\github\demodev\dev-topics-devops\dev-topics-dependencies\dependencies\myapppy\target\dist\myapppy-1.0.dev0\dist\myapppy-1.0.dev0-py3-none-any.whl

Installing collected packages: myapppy

Successfully installed myapppy-1.0.dev0

(venv) D:\Temp\myapptest>***show\_me***

executing file \_\_init\_\_.py from show\_me.py

(venv) D:\Temp\myapptest>***fibber***

0 . . . 1

1 . . . 1

2 . . . 2

3 . . . 3

4 . . . 5

5 . . . 8

## Deploy the newly created library to Databricks Community Edition and verify

We tested deploying the “wheel” file locally, and now we can test adding it to our “Notebooks” on the *Databricks Community Edition* version of Apache Spark. The Databricks reference #1 below discusses establishing a free account on the community edition. We follow a three step process to test our library:

1. Upload our Python Wheel file “library” for ***myapppy*** to the Databricks file system.
2. Create a notebook to hold our test code, and upload the code into cells in the notebook.
3. Run the tests and validate library execution

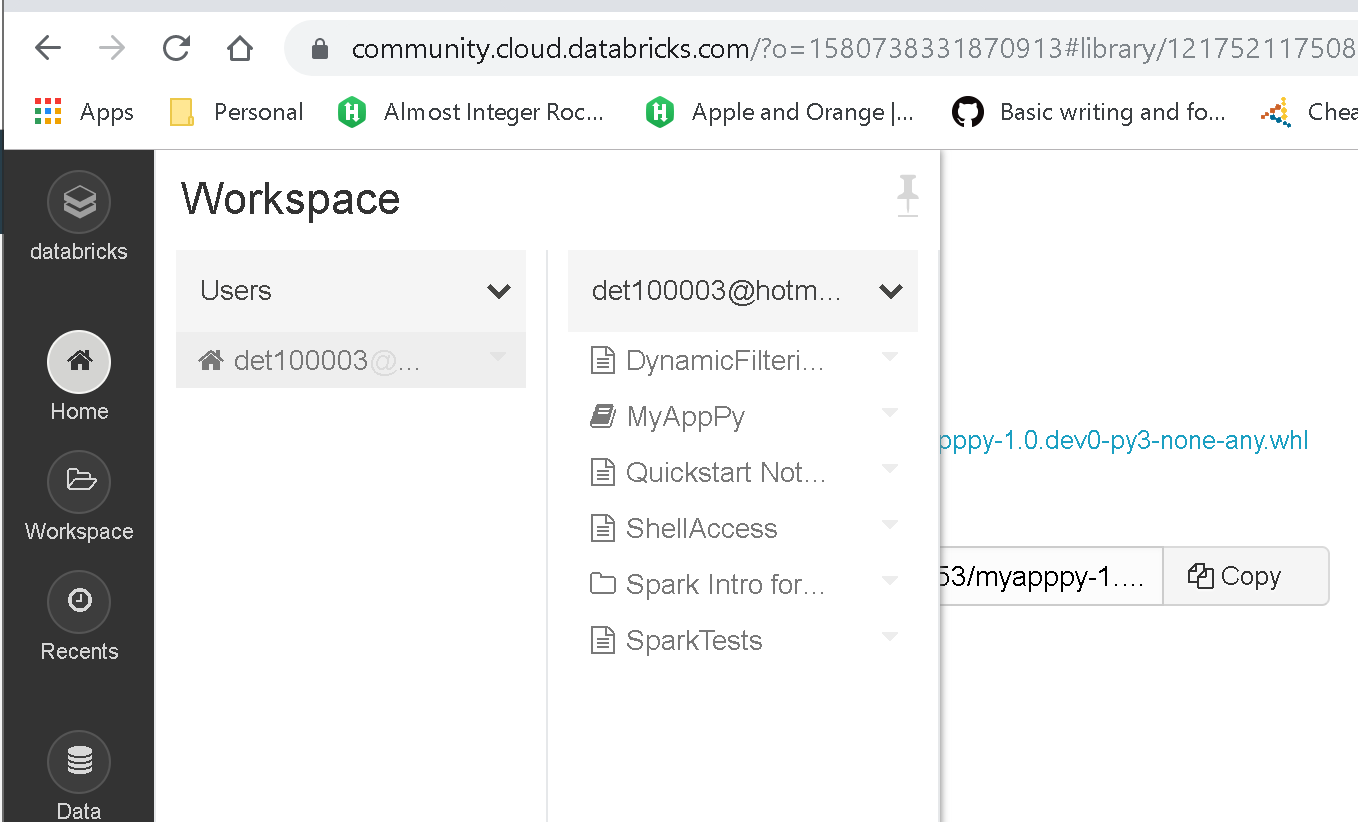
### Step One: Upload Library

1. Launch the *Databricks Community Edition* from your browser (see <https://community.cloud.databricks.com>.)
2. Select ***clusters***, and then select:
   1. An existing cluster (***interactive*** or ***automated***), or
   2. ***Create Cluster*** (a new cluster)
3. The selected cluster shows up in the ***interactive*** or ***automated*** list, so
4. High-light the desired cluster and select the ***Libraries*** link.
5. On the summary page, showing libraries, select the ***install new*** button on the upper left.
6. In the ***install library*** dialog box, select ***upload*** for *library source* and ***Python Whl*** for *library type*, and
   1. Drag the wheel file from your local project into the browser (e.g., ***Dependencies\myapppy\target\dist\myapppy-1.0.dev0\dist\myapppy-1.0.dev0-py3-none-any.whl***) into the rectangle labeled *Drop Whl Here*, and then
   2. Click on ***install***.
7. The installing dialog will appear, along with a DBFS storage location for the wheel file.
8. Click on the library description path and copy-and-save the path for later use (e.g., ***dbfs:/FileStore/jars/14d9ab94\_ffab\_40fa\_b6bc\_8b55f0f99045/myapppy-1.0.dev0-py3-none-any.whl***.)

At this point, we have a running cluster with access to a stored library. The Home tab shows our library. We can list it using *DbfsUtils*:



We can view the library in the Databricks GUI as well:

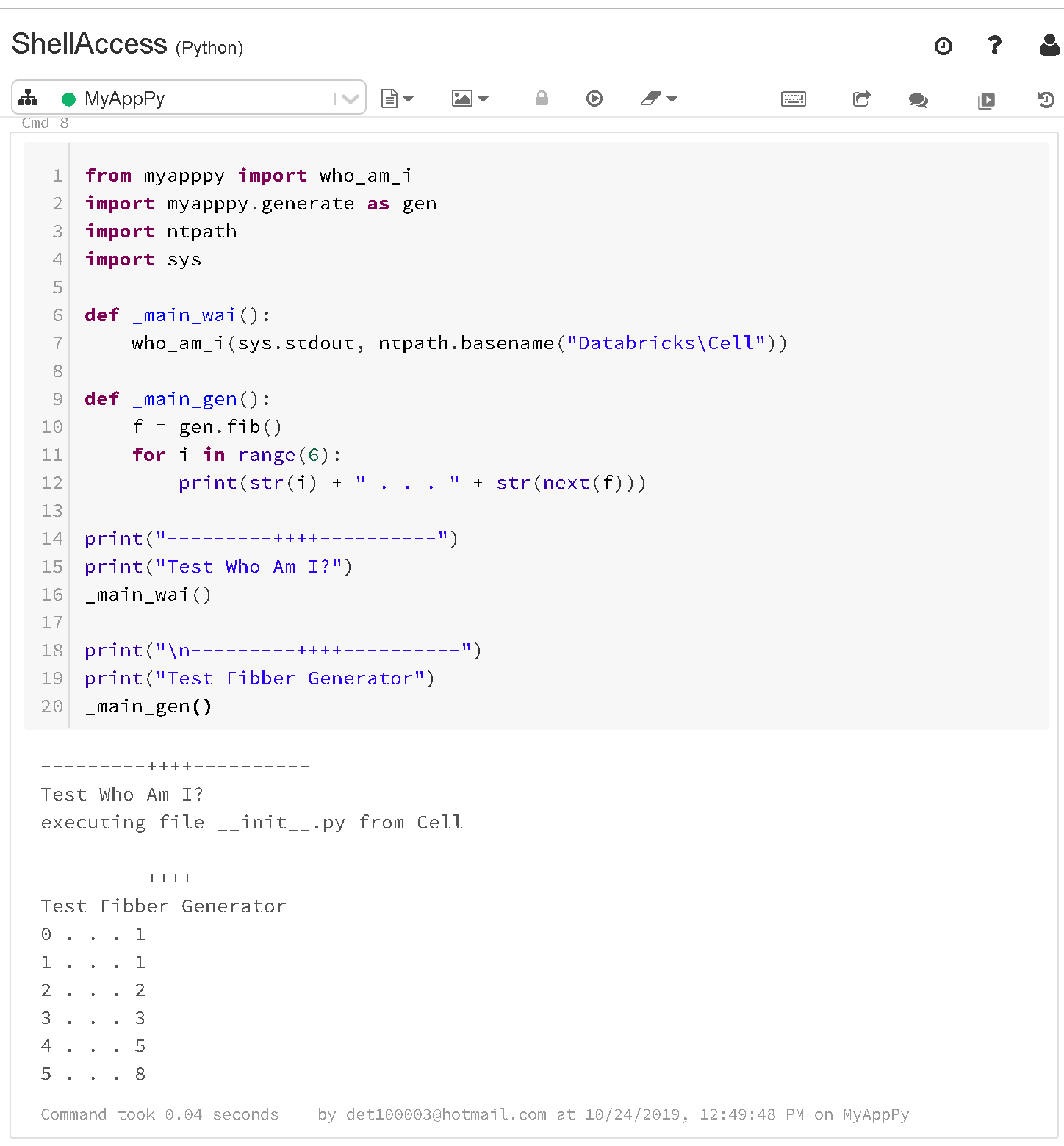


Now we install the library into the notebook so the Python code in the notebook can access the library. Library installation uses the *dbutils* utility like this:



### Steps Two and Three: Upload Installation Tests into Notebook Cell and Validate

We have installed the library into the notebook and are now able to access it in Python using the *import* mechanism. Here is the sample run:



## Conclusion

We have created a development environment that allows us to create source and debug in an IDE, test and build the source locally, and create a deployable library. We took one variant of that deployable library (the wheel file), installed it into the Databricks Community Edition, and verified that the library worked in that cloud environment.

# References - Resources

1. The DemoDev GitHub repository with required supporting files: <https://github.com/DonaldET/DemoDev/tree/master/dev-topics-devops/dev-topics-dependencies/Dependencies> and setup files <https://github.com/DonaldET/DemoDev/tree/master/dev-topics-devops/dev-topics-dependencies/Dependencies/builder_dependencies>.
2. Virtualenv – used to create a controlled Python runtime environment: <https://pypi.python.org/pypi/virtualenv>).
3. Additional Virtualenv documentation: <https://virtualenv.pypa.io/en/latest/>).
4. Venv background: <https://realpython.com/python-virtual-environments-a-primer/>.

## PyBuilder Documentation

1. PyBuilder Documentation Home: <http://pybuilder.github.io/>.
2. PyBuilder GitHub repository: <https://github.com/pybuilder/pybuilder>.
3. PyBuilder master ***build.py*** link in GitHub: <https://github.com/pybuilder/pybuilder/blob/master/build.py>.
4. PyBuilder tutorial (top level): <https://pybuilder.readthedocs.io/en/latest/walkthrough-new.html>.
5. Additional PyBuilder tutorials: <http://pybuilder.github.io/documentation/tutorial.html#.XaJXGkZKiUk>.
6. PyBuilder PDF: <https://buildmedia.readthedocs.org/media/pdf/pybuilder/stable/pybuilder.pdf>.

## PyCharm References

1. PyCharm download: <https://www.jetbrains.com/pycharm/download/#section=windows>.
2. PyCharm background: <https://en.wikipedia.org/wiki/PyCharm>.
3. PyCharm Getting Started: <https://www.jetbrains.com/help/pycharm/quick-start-guide.html>.

## Databricks References

1. Getting started: <https://www.c-sharpcorner.com/article/working-with-free-community-edition-databricks-spark-cluster/>.
2. Library description: <https://databricks.com/blog/2019/01/08/introducing-databricks-library-utilities-for-notebooks.html>.
3. AWS Libraries Documentation: <https://databricks.com/blog/2019/01/08/introducing-databricks-library-utilities-for-notebooks.html>.
4. DBUtils library: <https://docs.databricks.com/dev-tools/databricks-utils.html#dbutils-library>.