Computer Science & Information Systems

**DevOps for Cloud - Lab Sheet 2 – Module 4**

**(M4: Continuous Integration)**

This lab sheet needs to be administered along with Module 4: Continuous Integration

**Notation used in the document**

* ‘>’ represents the terminal, where we type the commands.
* The text mentioned within ‘[‘ and ‘]’ brackets provides additional documentation for the step.

1. Objective:
2. To demonstrate the basic process of “build” and “test” using PyBuilder in Python.
3. To demonstrate the concept of Continuous Integration (CI), which includes the steps of build and test. Use GitHub Actions as the CI tool to demonstrate the experiment.
4. Pre-requisite:

* Basic knowledge of GIT, GitHub Actions and Continuous Integration practice
* Valid GitHub credentials
* IDE such as Visual Studio Code

1. Lab Exercise:

**Task 1: Install Python**

* URL - <https://www.python.org/downloads/>
* Download the latest version of Python based on your OS type (Windows, Mac or Linux)
* [Neglect this step if Python is already installed in your system]

**Task 2: Install PyBuilder and create basic project scaffold**

[We shall use the “pyb” command to create a basic project scaffold comprising the code and unit test. Refer to the URL - <https://pybuilder.io/documentation/tutorial#creating-pybuilder-project> for detailed explanation of the steps]

* Create an empty folder called “pyb”
* Open any terminal (such as command prompt) and navigate to the folder
* > pyb --start-project

$ pyb --start-project

Project name (**default**: **'helloworld'**) : <enter>

Source directory (**default**: **'src/main/python'**) : <enter>

Docs directory (**default**: **'docs'**) : <enter>

Unittest directory (**default**: **'src/unittest/python'**) : <enter>

Scripts directory (**default**: **'src/main/scripts'**) : <enter>

**Use** plugin python.flake8 (Y/n)? (**default**: **'y'**) : <enter>

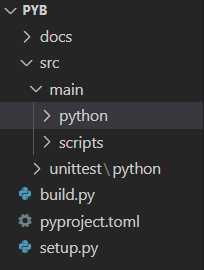
**Use** plugin python.coverage (Y/n)? (**default**: **'y'**) : <enter>

**Use** plugin python.distutils (Y/n)? (**default**: **'y'**) : <enter>

Created **'setup.py'**.

Created **'pyproject.toml'**.

* Open Visual Studio Code, open the project folder (pyb), and inspect the created files. The file structure should look like this.



**Task 3: Create the source code file**

* Navigate to src/main/python and create a file “helloworld.py” with the following sample code

import sys

def helloworld(out):

out.write("Hello world of Python\n")

**Task 4: Create the Unit test file**

* Navigate to src/unittest/python and create a file "helloworld\_tests.py" with the following content

from mockito import mock, verify

import unittest

from helloworld import helloworld

class HelloWorldTest(unittest.TestCase):

    def test\_should\_issue\_hello\_world\_message(self):

        out = mock()

        helloworld(out)

        verify(out).write("Hello world of Python\n")

**Task 5: Update the build.py file to include mockito**

* Include the following code in “build.py” file

@init

def set\_properties(project):

    project.build\_depends\_on("mockito")

Overall, the “build.py” file will look like

#   -\*- coding: utf-8 -\*-

from pybuilder.core import use\_plugin, init

use\_plugin("python.core")

use\_plugin("python.unittest")

use\_plugin("python.flake8")

use\_plugin("python.coverage")

use\_plugin("python.distutils")

name = "pyb-demo"

default\_task = "publish"

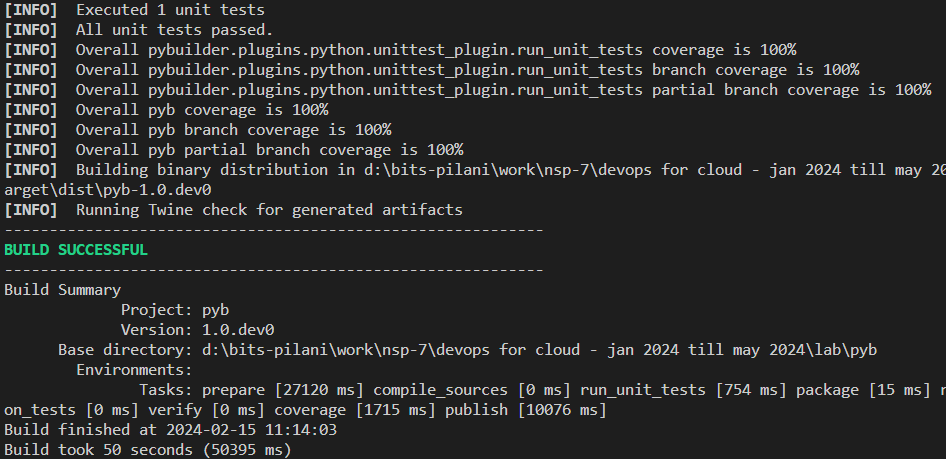
@init

def set\_properties(project):

    project.build\_depends\_on("mockito")

**Task 6: Run the “pyb” command to Build and Test the application**

* Open the Terminal in Visual Studio Code [View -> Terminal]
* > pyb
* Sample Screenshot of the build and test execution is shown below



**Task 7: Make the “pyb” folder as a Git repository**

* In the VS Code Terminal, type the following commands
* > git init
* > git add .
* > git commit -m “Committing all changes to git”

**Task 8: Integrate with GitHub and Push the Application to GitHub**

1. Login to GitHub
2. Create a new repository called “pyb”
3. In VS Code Terminal, run the command

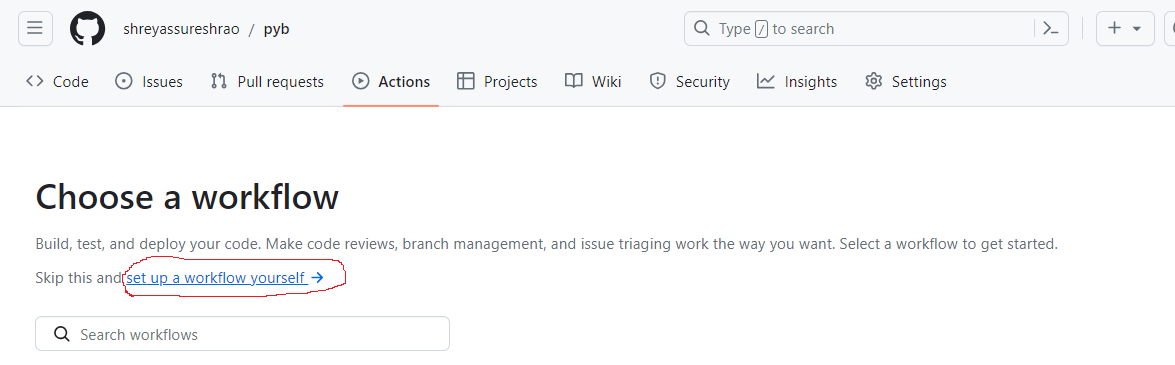
* > git remote add origin [https://github.com/<username of account owner>/pyb.git](https://github.com/%3cusername%20of%20account%20owner%3e/pyb.git)
* [Ex: git remote add origin <https://github.com/shreyassureshrao/pyb.git>]

1. Push the contents from local repo to remote repo

* > git push [https://github.com/<username of account owner>/pyb.git](https://github.com/%3cusername%20of%20account%20owner%3e/pyb.git)
* [Will push the local repo to remote location]

**Task 9: Setup Workflow in GitHub Actions and run the CI Pipeline**

1. Navigate to “pyb” repository in GitHub
2. Click on “Actions” tab [This will open the GitHub Actions]
3. Click on “set up a workflow yourself” link



1. Copy-paste the following code in the main.yml file under pyb/.github/workflows path

name: PyBuilder Workflow

on:

  push:

    branches:

      - main

  pull\_request:

    branches:

      - main

jobs:

  lint:

    runs-on: ubuntu-latest

    steps:

      - name: Checkout Repository

        uses: actions/checkout@v2

      - name: Set up Python

        uses: actions/setup-python@v2

        with:

          python-version: 3.9

      - name: Install pylint

        run: pip install pylint

      - name: Run pylint

        run: pylint src/main/python/helloworld.py

  build\_and\_test:

    runs-on: ubuntu-latest

    steps:

      - name: Checkout Repository

        uses: actions/checkout@v2

      - name: Set up Python

        uses: actions/setup-python@v2

        with:

          python-version: 3.9

      - name: Install PyBuilder

        run: |

          python -m pip install --upgrade pip

          pip install pybuilder

      - name: Build and Test

        run: pyb

      - name: Upload Test Results

        uses: actions/upload-artifact@v2

        with:

          name: test-results

          path: target/reports

      - name: List Files  # Optional step

        run: |

          ls -R

[In this file, two jobs are created. 1 -> lint; 2 -> Build and Test]

1. Commit the changes to the main branch
2. Click on “Actions” -> “PyBuilder Workflow” to see the result
3. You will get an “error” in the lint job, since the code is not ascribing to proper coding standards.
4. Perform “git pull” command to sync the GitHub changes (main.yaml) with the local (in VS Code)

> git pull [https://github.com/<account-name>/pyb](https://github.com/%3caccount-name%3e/pyb)

1. In order to rectify the code, replace the previous “helloworld.py” code with the below code, which is as per proper coding standards.

"""

This is a simple module to demonstrate a hello world function in Python.

"""

import sys

def helloworld(out):

    """

    Print a hello world message.

    Args:

        out: Output stream to write the message to.

    """

    out.write("Hello world of Python\n")

# Call the function with a file object

helloworld(sys.stdout)

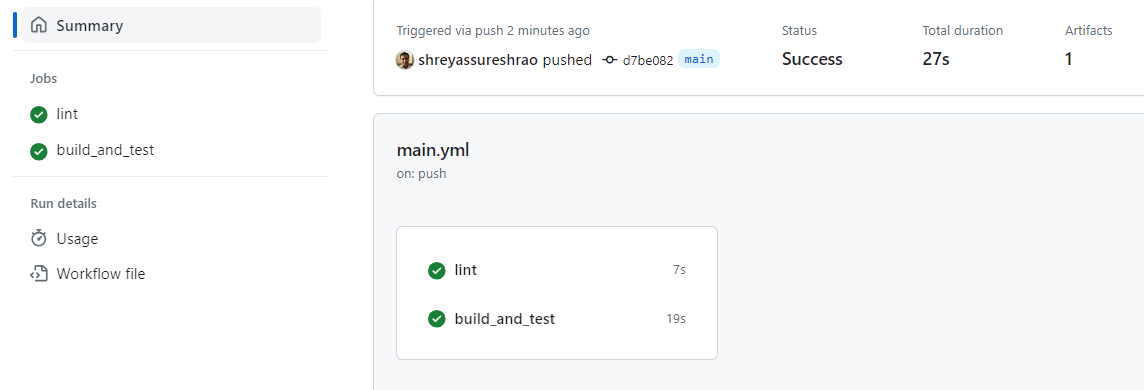
1. [The source code changes have to be staged, committed and pushed to GitHub. This will automatically trigger the “GitHub Actions” workflow]
2. In the VS code terminal, type the following commands

> git add .

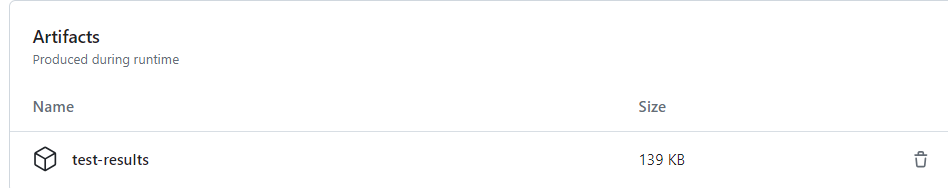
> git commit -m “Changed HelloWorld.py code”

> git push [https://github.com/<account-name>/pyb.git](https://github.com/%3caccount-name%3e/pyb.git)

1. This will automatically trigger the workflow. This time, both “Lint” and “build-and-test” jobs should run successfully



1. An artifact titled “test-results” will be generated and placed under the “Artifacts” section. View the folder, and verify the unit test execution report and test coverage reports.



1. Outputs/Results:

Students are expected to perform the tasks provided in the lab capsule, and thereby gain a practical understanding of the Build process, Testing, and Continuous Integration practice. Additionally, the students will gain a basic understanding of working on GitHub Actions as a Continuous Integration tool.

1. Observations:

* This lab capsule demonstrates the process of build and test (unit test) using the PyBuilder tool, which is specific to Python. Students are encouraged to explore other build tools such as Maven and Gradle, for Java applications.
* This lab capsule employs GitHub Actions as the tool to demonstrate the Continuous Integration practice. Students are encouraged to explore contemporary tools such as Jenkins, Jenkins X, GitLab etc., to perform the CI activity.

References:

* <https://pybuilder.io/documentation/tutorial#creating-pybuilder-project>
* <https://docs.github.com/en/actions>