Deployment automation

Continuous deployment depends on automation so that as soon as a change is committed, a series of automated activities is triggered to test the software. It is often used to automate end-to-end processes and improve the speed and quality of software development.

Continuous deployment with Github Action & Python

GitHub offers a feature called GitHub Actions, enabling the implementation of continuous deployment (CD) directly within a repository. This automated workflow platform allows you to streamline tasks such as building, testing, and deploying code, all triggered by changes made in the repository.

- **Step 1**:

  To integrate this project into GitHub Actions, the first step is to allow the Github Action in setting page:

  ```
  Settings -> Actions -> Allow all actions and reusable workflows
  ```

- **Step 2:**

To set up using GitHub Actions, begin by creating a folder named ".github/workflows" in your repository. Then, add a YAML-formatted workflow configuration file inside this folder. In this example, we will create a file called python-ci.yml.

The sample configuration file defines a workflow named "Python CI", which is triggered automatically every time code is pushed to the repository. This workflow ensures compatibility across different environments by running tests on four Python versions: 3.7, 3.8, 3.9, and 3.10. It begins by setting up the appropriate Python environment, followed by installing the project's dependencies, and finally executing the unit tests.

For code specifics, refer to the Moodle resources provided.

- **Step 3:**

Once you've completed the steps mentioned above, the .github/workflows directory will be visible at the root of your GitHub repository.

Now, let's suppose you have a simple string manipulation script with the following functions: reversing a string, checking if a string is a palindrome, and converting a string to uppercase. Here's how the main script and its test script should look like:

```python
# StringProcessor.py
def reverse_string(s):
    """Reverses the given string."""
    return s[::-1]

def is_palindrome(s):
    """Checks if the given string is a palindrome."""
    return s == s[::-1]
```

```python
def to_uppercase(s):
    """Converts the given string to uppercase."""
    return s.upper()
```

```python
# myTest.py
import unittest
from StringProcessor import reverse_string, is_palindrome, to_uppercase

class TestStringProcessor(unittest.TestCase):
    def test_reverse_string(self):
        self.assertEqual(reverse_string("hello"), "olleh")
        self.assertEqual(reverse_string("abc"), "cba")
        self.assertEqual(reverse_string(""), "")

    def test_is_palindrome(self):
        self.assertTrue(is_palindrome("madam"))
        self.assertFalse(is_palindrome("hello"))
        self.assertTrue(is_palindrome(""))

    def test_to_uppercase(self):
        self.assertEqual(to_uppercase("hello"), "HELLO")
        self.assertEqual(to_uppercase("world"), "WORLD")
        self.assertEqual(to_uppercase(""), "")

if __name__ == '__main__':
    unittest.main()
```

Then evey time you push your code to GitHub, the Action will automatically run.