Add Continuous Integration and Continuous Deployment

Estimated time needed: 60 minutes

Your team is growing! Management has decided to hire front-end and back-end engineers to ensure features on the roadmap are developed in time for future releases. However, this means that multiple engineers will need to work in parallel on the repository. You are tasked with ensuring the code being pushed to the main branch meets the team coding style and is free of syntax errors.

In this lab, you will add linting to your repository that automatically checks for such errors whenever a developer creates a pull request or whenever a branch is merged into the default main branch. Before we dive into the lab, here is a primer on GitHub Actions.

GitHub Actions

GitHub actions provide an event-driven way to automate tasks in your project. There are several kinds of events you can listen to. Here are a few examples:

- **push**: Runs tasks when someone pushes to a repository branch.
- **pull_request**: Runs tasks when someone creates a pull request (PR). You can also start tasks when certain activities happen, such as:
  - PR opened
  - PR closed
  - PR reopened
- **create**: Run tasks when someone creates a branch or a tag.
- **delete**: Run tasks when someone deletes a branch or a tag.
- **manually**: Jobs are kicked off manually.

GitHub Action Components

You will use one or more of the following components in this lab:

- **Workflows**: A collection of jobs you can add to your repository.
- **Events**: An activity that launches a workflow.
- **Jobs**: A sequence of one or more steps. Jobs are run in parallel by default.
- **Steps**: Individual tasks that can run in a job. A step may be an action or a command.
- **Actions**: The smallest block of a workflow.

GitHub Workflow

You are provided with a workflow template below. Let's examine it.

```
name: 'Lint Code'
on:
  push:
    branches: [master, main]
  pull_request:
    branches: [master, main]
jobs:
  lint_python:
    name: Lint Python Files
    runs-on: ubuntu-latest
    steps:
    - name: Checkout Repository
      uses: actions/checkout@v3

    - name: Set up Python
      uses: actions/setup-python@v4
      with:
        python-version: 3.12
    - name: Install dependencies
      run: |
        python -m pip install --upgrade pip
        pip install flake8
    - name: Print working directory
      run: pwd
    - name: Run Linter
      run: |
        pwd
        # This command finds all Python files recursively and runs flake8 on them
        find . -name "*.py" -exec flake8 {} +
        echo "Linted all the python files successfully"
  lint_js:
      name: Lint JavaScript Files
      runs-on: ubuntu-latest
      steps:
      - name: Checkout Repository
        uses: actions/checkout@v3
      - name: Install Node.js
        uses: actions/setup-node@v3
        with:
          node-version: 14
      - name: Install JSHint
        run: npm install jshint --global
      - name: Run Linter
        run: |
          # This command finds all JavaScript files recursively and runs JSHint on them
          find ./server/database -name "*.js" -exec jshint {} +
          echo "Linted all the js files successfully"
```

1. The first line names the workflow.
2. The next line defines when this workflow will run. The workflow should run when developers push a change to the main branch or create a PR. These two ways are captured as follows:
   - run on push to the main branch (main or master):
     ```
     push:
       branches: [master, main]
     ```
   - run when PR is created on main branches (main or master):
     ```
     pull_request:
       branches: [master, main]
     ```
3. You will then define all the jobs. There are two jobs in this workflow:
   - lint_python: Linting JavaScript function
   - lint_js: Linting Python function

GitHub Jobs

Let's look at each of these jobs:

1. lint_python

   - Set up the Python runtime for the action to run using the `actions/setup-python@v4` action.

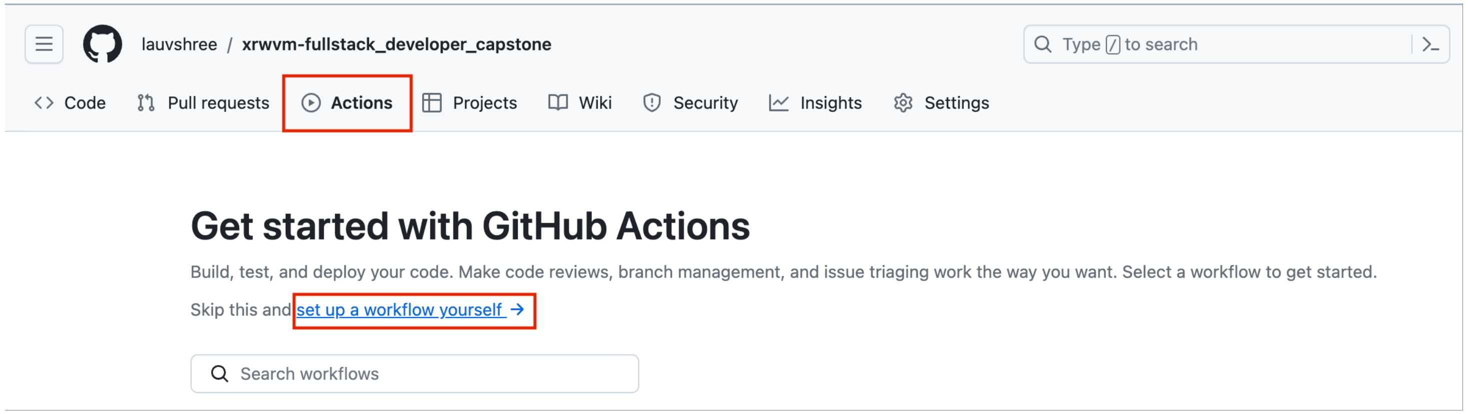
  - Install all dependencies using `pip install`.
  - Run the linting command `flake8 *.py` in all files in server directory recursively.
  - Print a message saying the linting was completed successfully.

2. lint_function_js

  - Set up the Node.js runtime for the action to run using the `actions/setup-node@v3` action.
  - Install all JSHint linter `npm install jshint`.
  - Run the linting command on all the .js files in the database directory recursively.
  - Print a message saying the linting was completed successfully.

Enable GitHub Actions

1. To enable GitHub action, log into GitHub and open your forked repo. Next, go to the `Actions` tab and click `Set up a workflow yourself`.

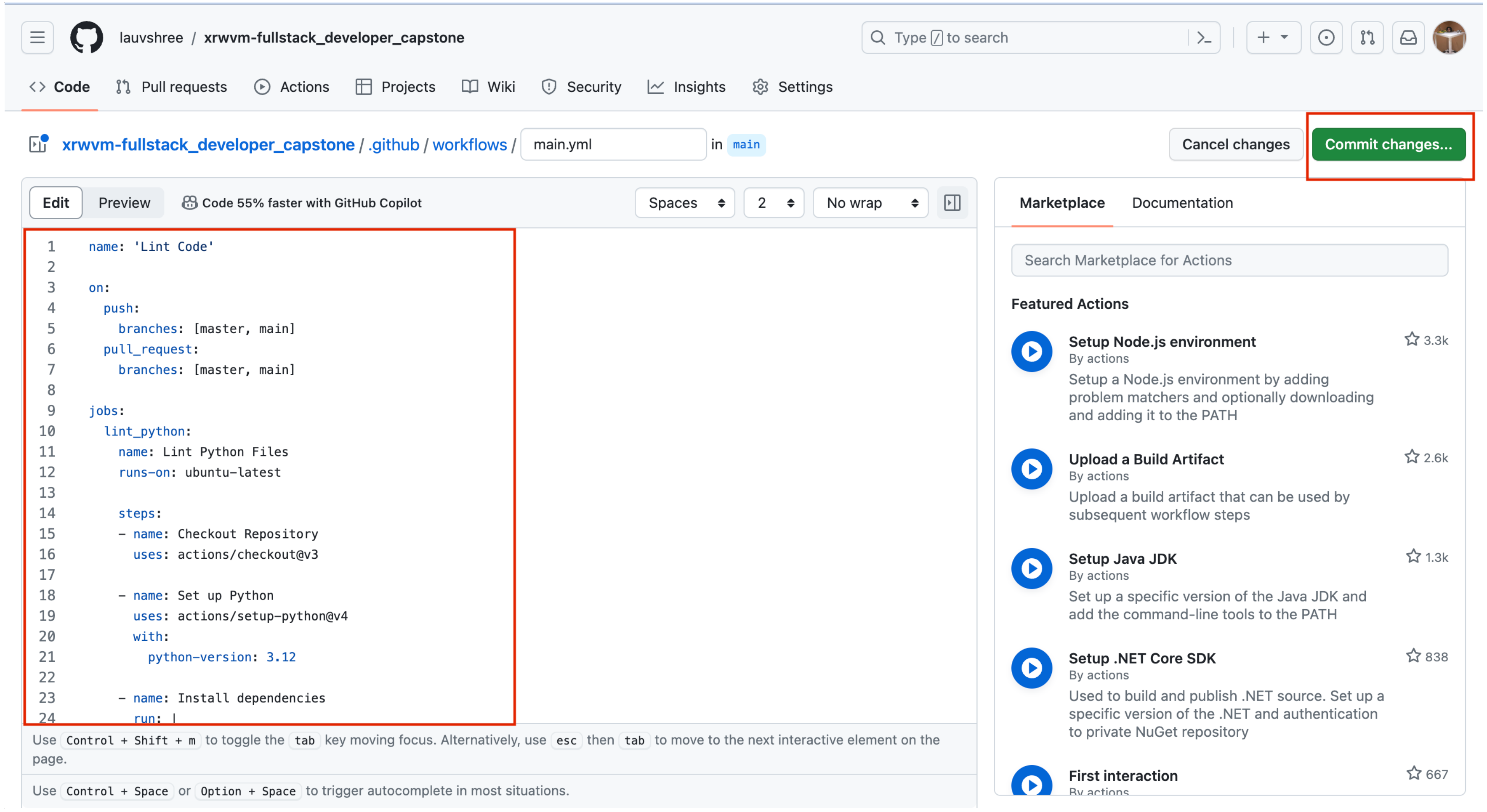


Get started with GitHub Actions

Build, test, and deploy your code. Make code reviews, branch management, and issue triaging work the way you want. Select a workflow to get started.

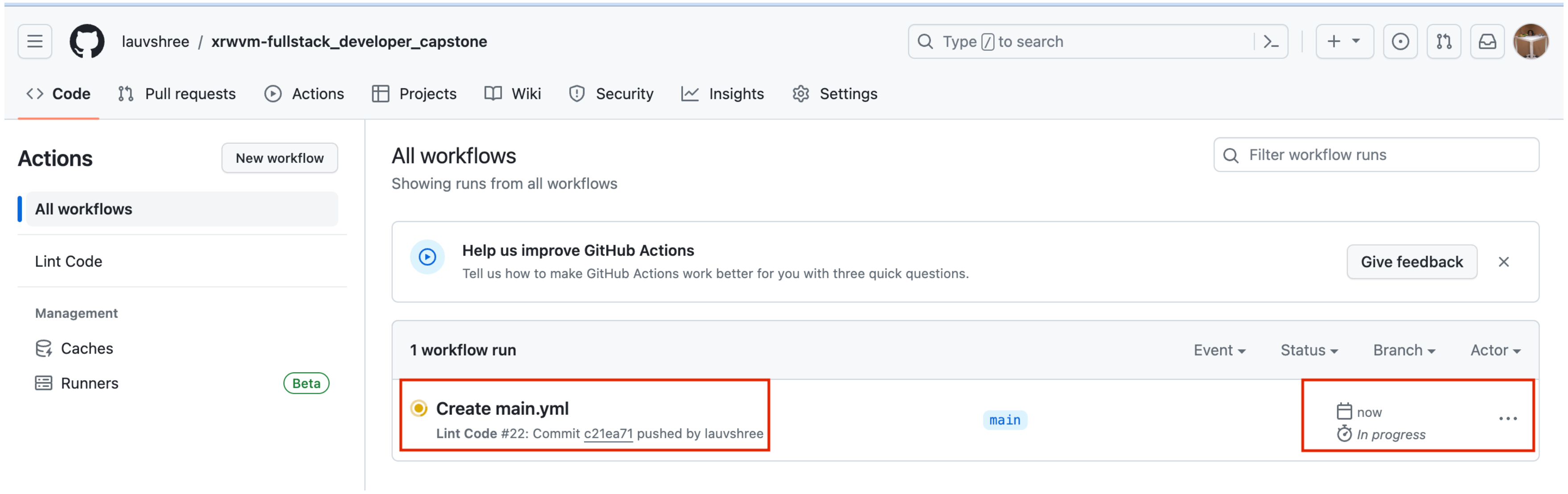
Skip this and set up a workflow yourself →

2. Paste the lint code given above inside `main.yml` and commit it.



```yaml
name: 'Lint Code'

on:
  push:
    branches: [master, main]
  pull_request:
    branches: [master, main]

jobs:
  lint_python:
    name: Lint Python Files
    runs-on: ubuntu-latest

    steps:
    - name: Checkout Repository
      uses: actions/checkout@v3

    - name: Set up Python
      uses: actions/setup-python@v4
      with:
        python-version: 3.12

    - name: Install dependencies
      run: |
```

3. Open the `Actions` tab again, and you will see that the commit has automatically started the lint workflow.



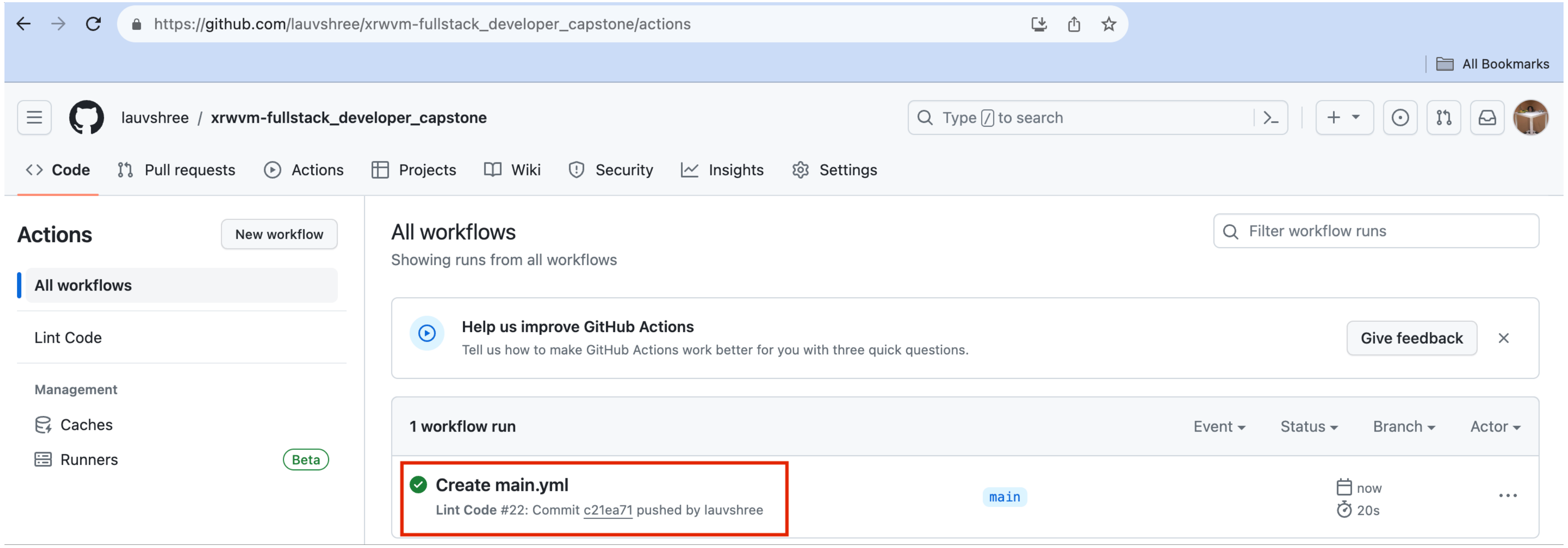
All workflows
Showing runs from all workflows

Help us improve GitHub Actions
Tell us how to make GitHub Actions work better for you with three quick questions.

1 workflow run

Create main.yml
Lint Code #22: Commit c21ea71 pushed by lauvshree
main
now
In progress

4. You can click the workflow run to see the individual jobs and the logs for each job. When the workflow successfully completes, you will see the green tick indicating it went well. A red cross would mean there were errors found in the code while linting.



All workflows
Showing runs from all workflows

Help us improve GitHub Actions
Tell us how to make GitHub Actions work better for you with three quick questions.

1 workflow run

Create main.yml
Lint Code #22: Commit c21ea71 pushed by lauvshree
main
now
20s

5. Check these hints to resolve usual Linting errors you could come across.

▾ Click here

Flake-8 Lint (Python) Linting errors:

1. If you receive one or more errors as listed below:

```
E117 over-indented
E128 continuation line under-indented for visual indent
```

Resolution:

Verify that all code maintains appropriate indentation—neither under-indented nor over-indented.

Note: Use a text editor to ensure accurate implementation.

2. Error:

```
E501 line too long (xxx > 79 characters)
```

Resolution:

Split the code into multiple lines, ensuring that each line has a maximum of 79 characters or less.

3. Error:

```
F401 'xxx' imported but unused
```

Resolution:

Verify whether the mentioned entity or variable (‘xxx’) is utilized in subsequent code segments. Remove the line containing it if the entity/variable is unused.

4. Error:

```
W292 no newline at end of file
```

Resolution:

Insert a new line after the final code in the file and position the cursor at the far-left vertical pane (without any rightward indentation).

5. Error:

```
 E302 expected 2 blank lines, found 1
```

Resolution:

Ensure there are exactly two empty lines (not more, not less) between each pair of adjacent functions.

eg:

```
....... "1st function ends"..
def "Next_function"():
```

6. Error:

```
E231 missing whitespace after ':'
```

Resolution:

Make sure to leave a space after the semicolon in all dictionary key-value pairs.

For eg. If the existing code is `"a":"b"` , please change it to `"a": "b"`

7. Error:

```
E275 missing whitespace after keyword
```

Resolution:

Ensure there is one white space after every keyword.

For instance, if your existing code is `if("condition"):`, please change it to `if ("condition"):` as demonstrated.

8. Error:

```
E722 do not use bare 'except'
```

Resolution:

Use `except Exception:` instead of `except` as a best practice for catching exceptions and handling them comprehensively.

For instance, if the existing code is:

```
except:
    print("Error")
```

You can change this to:

```
except Exception as e:
    print(f"Error: {e}")
```

JS Hint (Javascript) Linting errors:

1. If you receive one or more errors as listed below:

```
'const' is available in ES6 (use 'esversion: 6') or Mozilla JS extensions (use moz).
'arrow function syntax (=>)' is only available in ES6 (use 'esversion: 6').
'async functions' is only available in ES8 (use 'esversion: 8').
'let' is available in ES6 (use 'esversion: 6') or Mozilla JS extensions (use moz).
'template literal syntax' is only available in ES6 (use 'esversion: 6').
```

Resolution:

Add the line below at the start of the file(s) where this error is reported:

```
/*jshint esversion: 8 */
```

2. Error:

```
['xxxxxx'] is better written in dot notation.
```

- Resolution:
  This issue arises when Dictionary/JSON key-value pairs are formatted as `key[value]`. To resolve it, switch the format to `key.value`.

For example, if you have the existing code as: `keyA['valA']`, update it to `keyA.valA`.

Submission

Take a screenshot of the action workflow succeeding and save it as `CICD.png`.

Summary

In this lab, you added a linting service to your application. As a result, all new code will automatically get checked for syntax errors, and this will ensure all developers are following the team coding guidelines.

Author(s)

Upkar Lidder

Lavanya T S

Other Contributor(s)

Yan Luo
Priya