DevOps with GitHub Actions

Alessandro Bocci
name.surname@unipi.it

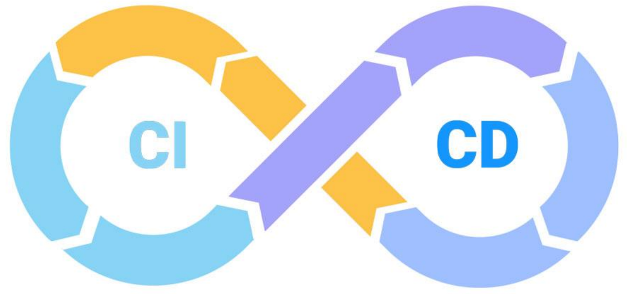
Advanced Software Engineering (Lab)
07/11/2025

Software Prerequisites

- Microservice architecture (**`microase`** or a project)
- GitHub account (with Unipi you can have PRO)

  Istructions: https://unipisa.github.io/join_organization
- With GitHub PRO you have unlimited Actions in public repos with standard runners and a fixed amount of Actions usage in private ones.

CI/CD

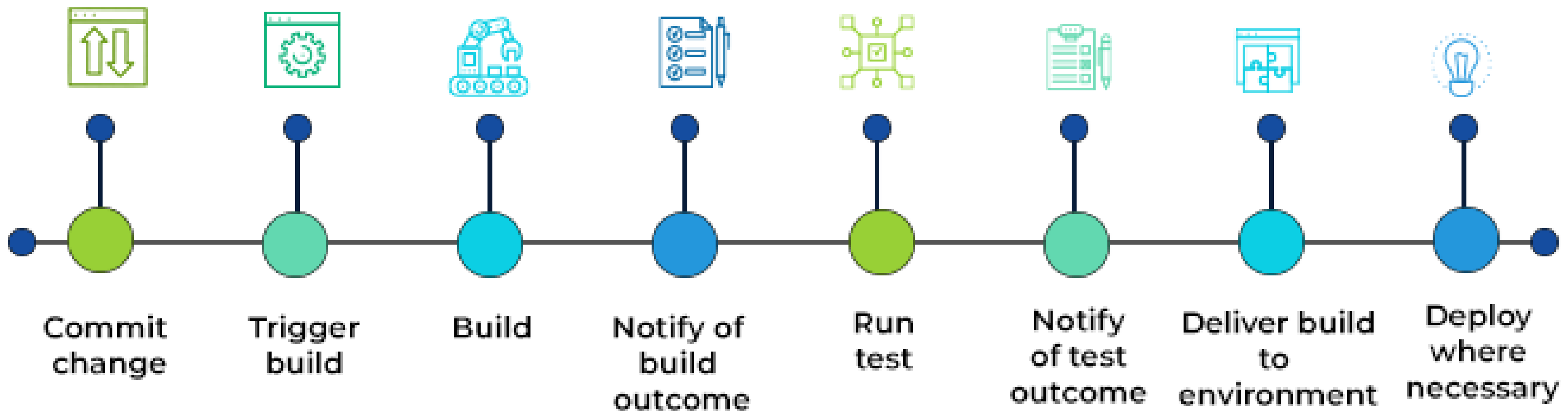


- Continuous Integration (CI):
  1. Developers regularly merge their code changes into a shared repository, usually multiple times a day.
  2. Automated tools then build the code and run tests to ensure that the changes don't break anything, allowing for early detection of issues.
- Continuous Deployment/Delivery (CD):
  1. **Continuous Delivery**: After CI, the application is automatically prepared for deployment to any environment (like testing or production), but a manual approval is still required to push it to production.
  2. **Continuous Deployment**: This goes one step further—code is automatically deployed to production without manual intervention, as long as all tests pass.

CI/CD

- **Continuous Integration (CI):**
  1. Developers regularly merge their code changes into a shared repository, usually multiple times a day.
  2. Automated tools then build the code and run tests to ensure that the changes don't break anything, allowing for early detection of issues.
- **Continuous Deployment/Delivery (CD):**
  1. **Continuous Delivery**: After CI, the application is automatically prepared for deployment to any environment (like testing or production), but a manual approval is still required to push it to production.
  2. **Continuous Deployment**: This goes one step further—code is automatically deployed to production without manual intervention, as long as all tests pass.

CI/CD Pipeline



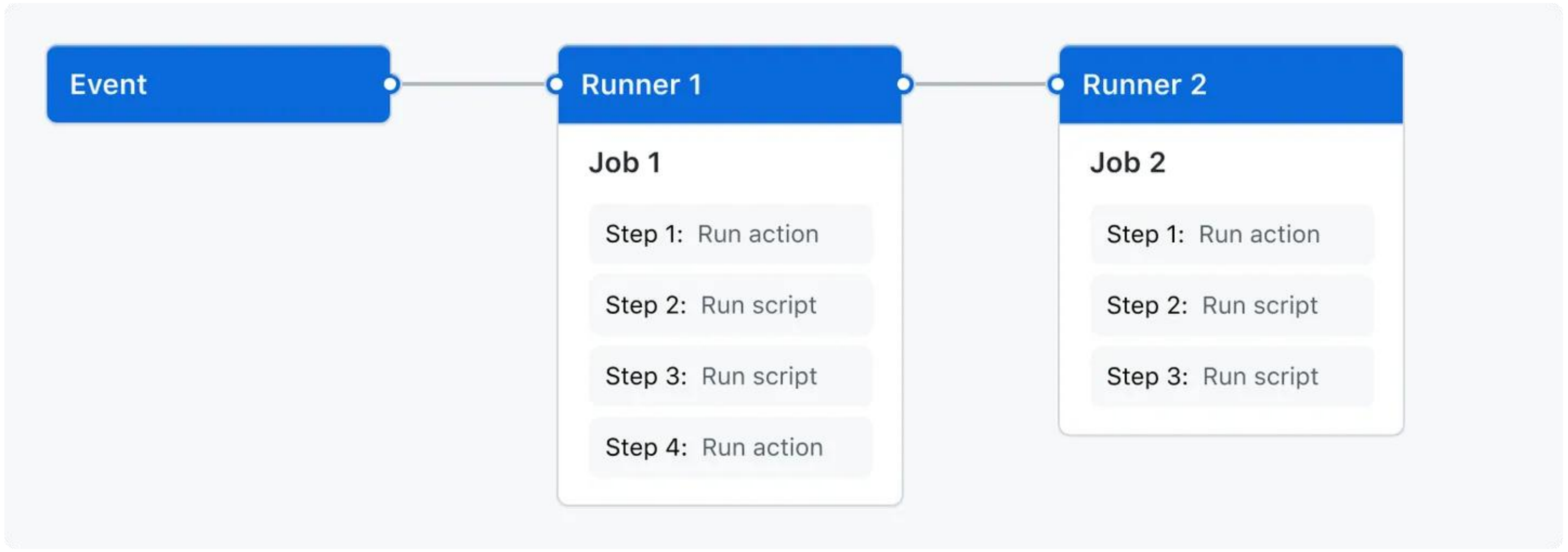
An automated series of **processes** triggered by an event that halts if any process fails, ensuring code quality and streamlined delivery.

GitHub Actions

GitHub Actions

GitHub Actions is a powerful CI/CD platform that integrates directly with GitHub repositories.

It allows to automate tasks like testing, building, and deploying your code.



GitHub Actions

GitHub Actions

Key Components:

- **Workflows**: Automated processes defined in YAML files. It must be stored in `.github/workflows/` within the repository.
- **Events**: Triggers for workflows. E.g. code push, pull request, or on a schedule.
- **Jobs**: Groups of steps that execute commands in a specified environment. Jobs can run in parallel or sequentially.
- **Steps**: Individual tasks within a job, such as running a script, installing dependencies, or deploying code.
- **Runners**: Servers that execute the steps in jobs. GitHub provides cloud-hosted runners or you can set up self-hosted ones.

IN SUPREMÆ DIGNITATIS · 1343 ·

Workflow example

```yaml
name: Python Application CI

on:
  push:
    branches:
      - main

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
    - name: Check out the repository
      uses: actions/checkout@v4

    - name: Set up Python
      uses: actions/setup-python@v5
      with:
        python-version: ‘3.12'

    - name: Install dependencies
      run: |
        python -m pip install --upgrade pip
        pip install -r requirements.txt

    - name: Run tests
      run: |
        pip install pytest
        pytest
```

Workflow example

Event

```yaml
name: Python Application CI

on:
  push:
    branches:
      - main

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
    - name: Check out the repository
      uses: actions/checkout@v4

    - name: Set up Python
      uses: actions/setup-python@v5
      with:
        python-version: ‘3.12'

    - name: Install dependencies
      run: |
        python -m pip install --upgrade pip
        pip install -r requirements.txt

    - name: Run tests
      run: |
        pip install pytest
        pytest
```

Workflow example

Event

Job

```yaml
name: Python Application CI

on:
  push:
    branches:
      - main

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
    - name: Check out the repository
      uses: actions/checkout@v4

    - name: Set up Python
      uses: actions/setup-python@v5
      with:
        python-version: ‘3.12'

    - name: Install dependencies
      run: |
        python -m pip install --upgrade pip
        pip install -r requirements.txt

    - name: Run tests
      run: |
        pip install pytest
        pytest
```

Workflow example

Event

Job

Runner

```yaml
name: Python Application CI

on:
  push:
    branches:
      - main

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
    - name: Check out the repository
      uses: actions/checkout@v4

    - name: Set up Python
      uses: actions/setup-python@v5
      with:
        python-version: ‘3.12'

    - name: Install dependencies
      run: |
        python -m pip install --upgrade pip
        pip install -r requirements.txt

    - name: Run tests
      run: |
        pip install pytest
        pytest
```

Workflow example

Event

Job

Runner

Steps

```yaml
name: Python Application CI

on:
  push:
    branches:
      - main

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
    - name: Check out the repository
      uses: actions/checkout@v4

    - name: Set up Python
      uses: actions/setup-python@v5
      with:
        python-version: ‘3.12'

    - name: Install dependencies
      run: |
        python -m pip install --upgrade pip
        pip install -r requirements.txt

    - name: Run tests
      run: |
        pip install pytest
        pytest
```

Workflow example

Steps can be predefined.
You can find them in public repositories, Docker hub or the same repo of the workflow.

```yaml
name: Python Application CI

on:
  push:
    branches:
      - main

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
    - name: Check out the repository
      uses: actions/checkout@v4

    - name: Set up Python
      uses: actions/setup-python@v5
      with:
        python-version: ‘3.12'

    - name: Install dependencies
      run: |
        python -m pip install --upgrade pip
        pip install -r requirements.txt

    - name: Run tests
      run: |
        pip install pytest
        pytest
```

Workflow example

`actions/checkout@v4`
is a special step required to use repository code in the workflow. It clones the repo in the runner.

```yaml
name: Python Application CI

on:
  push:
    branches:
      - main

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
    - name: Check out the repository
      uses: actions/checkout@v4

    - name: Set up Python
      uses: actions/setup-python@v5
      with:
        python-version: ‘3.12'

    - name: Install dependencies
      run: |
        python -m pip install --upgrade pip
        pip install -r requirements.txt

    - name: Run tests
      run: |
        pip install pytest
        pytest
```

Workflow example

Steps can be set of commands executed on the runner environment.

```
name: Python Application CI

on:
  push:
    branches:
      - main

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
    - name: Check out the repository
      uses: actions/checkout@v4

    - name: Set up Python
      uses: actions/setup-python@v5
      with:
        python-version: ‘3.12'

    - name: Install dependencies
      run: |
        python -m pip install --upgrade pip
        pip install -r requirements.txt

    - name: Run tests
      run: |
        pip install pytest
        pytest
```

GitHub Actions Cheat sheet

Basic Structure of a Workflow

A workflow is defined in a YAML file within the `.github/workflows/` directory.

```yaml
name: Workflow Name
on:
  push:
    branches:
      - main
  pull_request:
    branches:
      - main

jobs:
  job_name:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Run a command
        run: echo "Hello, World!"
```

Common events that trigger Workflows

- `push`: Triggered on code push.
- `pull_request`: Triggered on pull requests.
- `schedule`: Triggered on a schedule
- `workflow_dispatch`: Manually triggered via the GitHub UI.

```yaml
on:
  push:
    branches:
      - main
  schedule:
    - cron: '0 0 * * *'#Daily at midnight
```

Jobs and Runners

- `jobs`: A workflow can contain multiple jobs.
- `runs-on`: Specifies the type of runner (e.g., `ubuntu-latest`, `windows-latest`, `macos-latest`).

```yaml
jobs:
  build:
    runs-on: ubuntu-latest
```

Job Parallelism and Dependencies

- Parallel Jobs - Jobs run in parallel by default

```yaml
jobs:
  job1:
    runs-on: ubuntu-latest
    steps:
      - run: echo "Job 1"

  job2:
    runs-on: ubuntu-latest
    steps:
      - run: echo "Job 2"
```

- Job Dependencies - Use `needs` to specify dependencies

```yaml
jobs:
  job1:
    runs-on: ubuntu-latest
    steps:
      - run: echo "Job 1"

  job2:
    needs: job1  # job2 will run after job1
    runs-on: ubuntu-latest
    steps:
      - run: echo "Job 2"
```

Steps within Jobs

- `steps`: A sequence of tasks that are executed in order.
- `uses` to execute an action from the GitHub Marketplace.
- `run` to execute shell commands.

```yaml
steps:
  - name: Checkout
    uses: actions/checkout@v4

  - name: Set up Node.js
    uses: actions/setup-node@v4
    with:
      node-version: '14'

  - name: Install dependencies
    run: npm install
```

Use `with` to pass input parameters or configuration options to an action.

Conditionals

- Run a Step Based on a Condition:

```yaml
steps:
  - name: Run if on main branch
    if: github.ref == 'refs/heads/main'
    run: echo "Runs only on the main branch"
```

- Run Based on Success or Failure:

```yaml
steps:
  - name: Run on failure
    if: failure()
    run: echo "This runs on failure"
```

Common uses Actions

- Checkout Repository:

```yaml
uses: actions/checkout@v4
```

- Set up Python:

```yaml
uses: actions/setup-python@v5
with:
  python-version: '3.x'
```

- Set up Docker:

```yaml
uses: docker/setup-buildx-action@v3
```

Environment Variables

- Set Environment Variable:

```yaml
env:
  MY_VAR: Hello
```

- Use Environment Variables:

```yaml
steps:
  - name: Use ENV var
    run: echo "$MY_VAR"
```

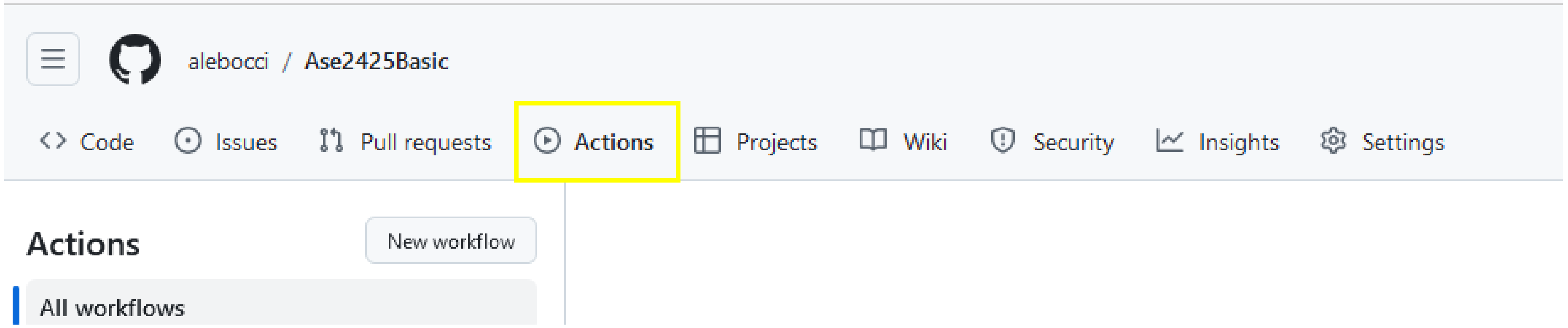
GitHub Action Docs:
https://docs.github.com/en/actions

Action's Marketplace:
https://github.com/marketplace?type=actions

Docker setup action docs:
https://github.com/docker/setup-buildx-action

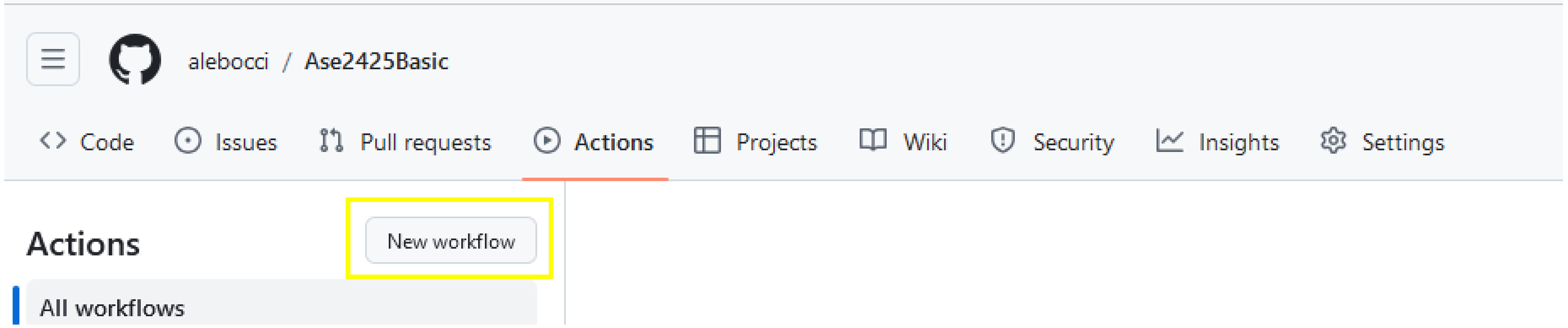
GitHub Actions

After enabling actions by allowing permissions of workflows in the repository settings…



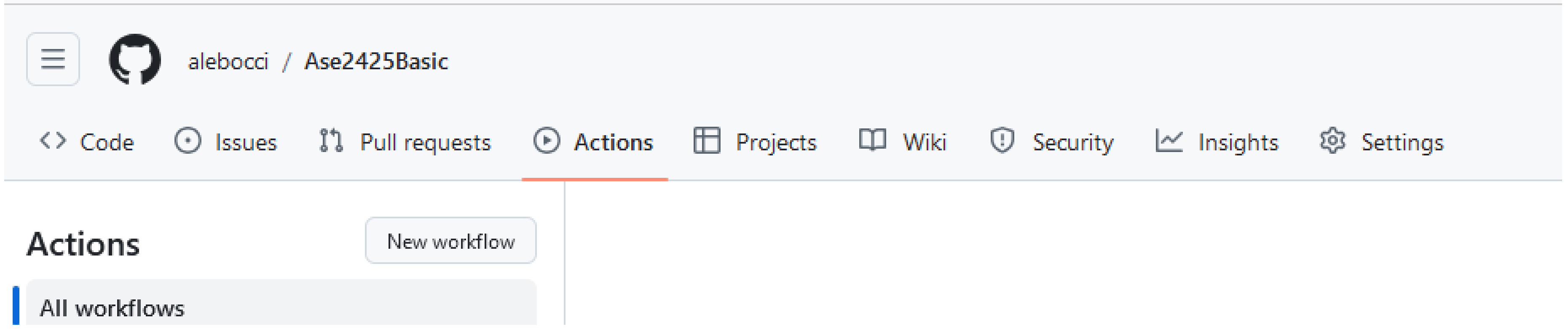
GitHub Actions

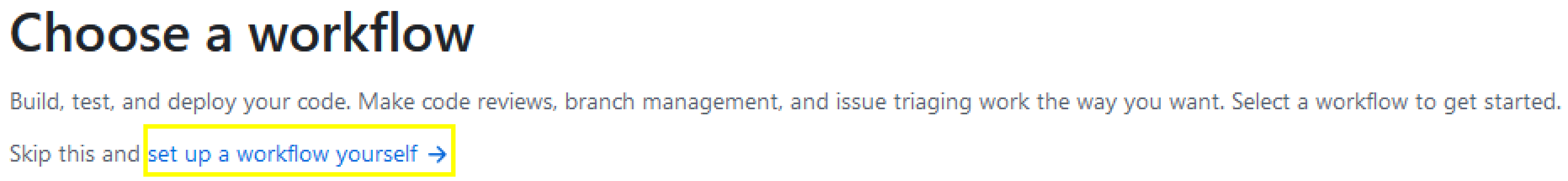
After enabling actions by allowing permissions of workflows in the repository settings…



GitHub Actions

After enabling actions by allowing permissions of workflows in the repository settings…





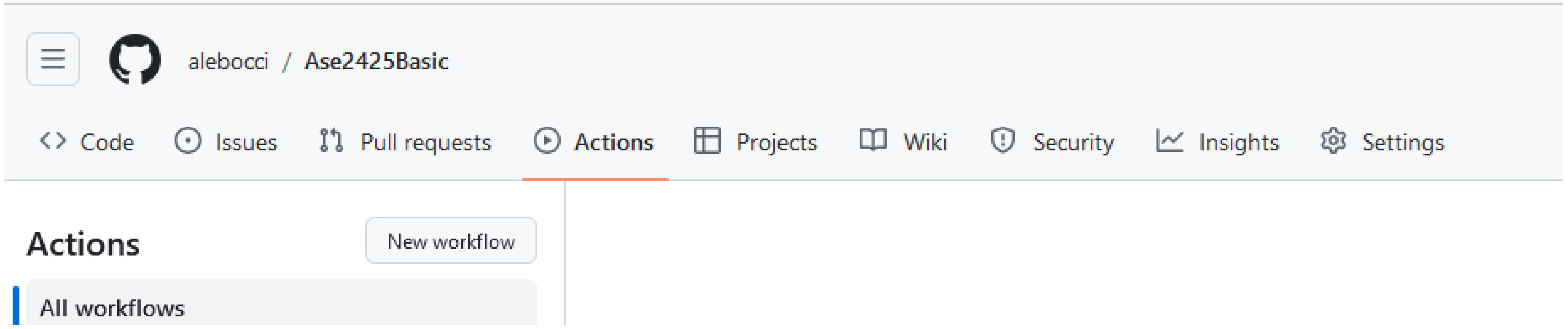
Choose a workflow

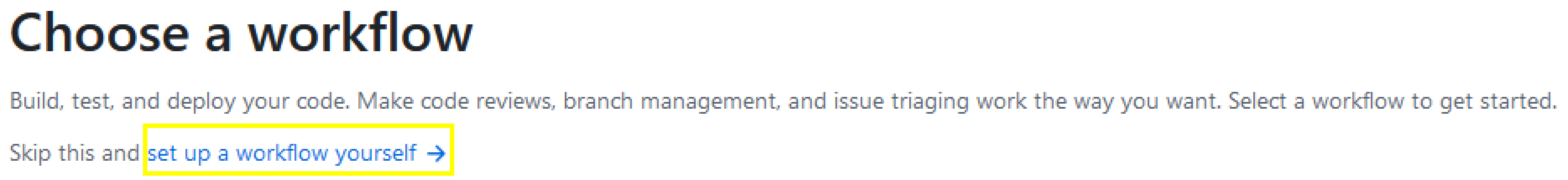
Build, test, and deploy your code. Make code reviews, branch management, and issue triaging work the way you want. Select a workflow to get started.

Skip this and set up a workflow yourself →

GitHub Actions

After enabling actions by allowing permissions of workflows in the repository settings…





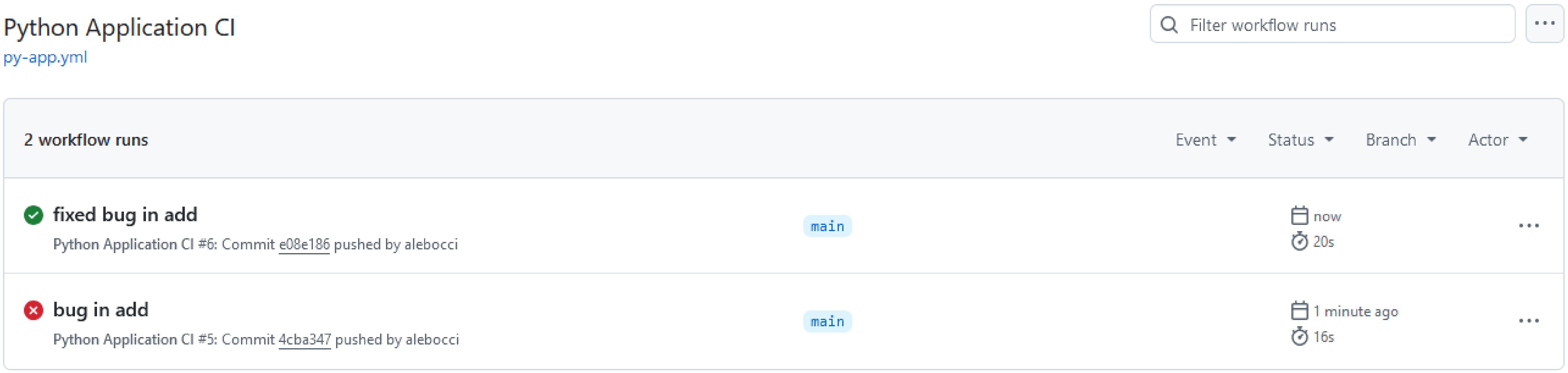
Choose a workflow

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 →

Otherwise, create and edit a file locally and push it.

Workflow execution view



Python Application CI

py-app.yml

Filter workflow runs

2 workflow runs

Event Status Branch Actor

✓ fixed bug in add

Python Application CI #6: Commit e08e186 pushed by alebocci

main

now

20s

✗ bug in add

Python Application CI #5: Commit 4cba347 pushed by alebocci

main

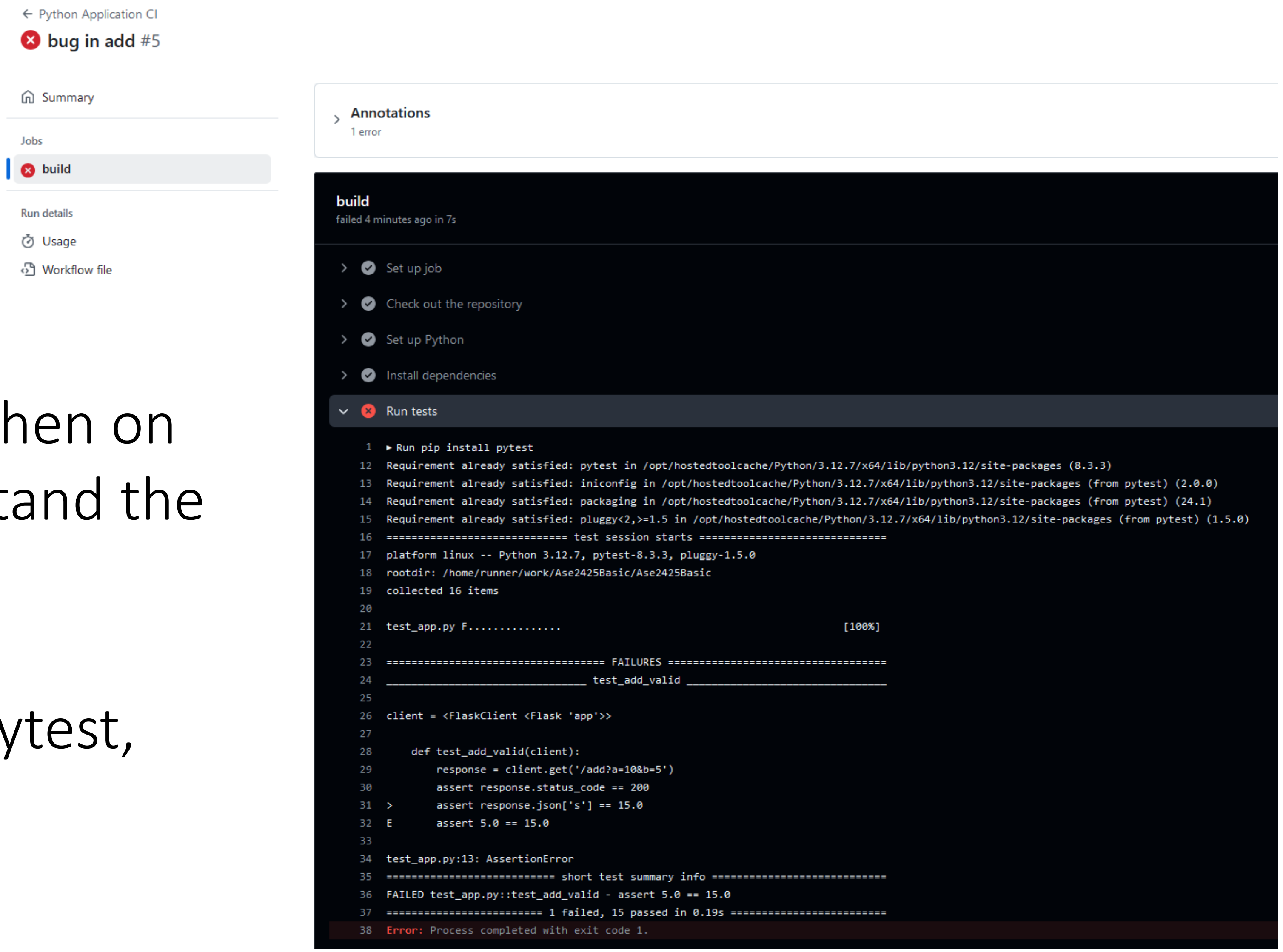
1 minute ago

16s

The first run did not pass the tests and it failed (bottom one).

The second one succeded.

Workflow execution view



← Python Application CI

bug in add #5

Summary

Jobs

build

Run details

Usage

Workflow file

Annotations
1 error

build
failed 4 minutes ago in 7s

Set up job

Check out the repository

Set up Python

Install dependencies

Run tests

```
▸ Run pip install pytest
Requirement already satisfied: pytest in /opt/hostedtoolcache/Python/3.12.7/x64/lib/python3.12/site-packages (8.3.3)
Requirement already satisfied: iniconfig in /opt/hostedtoolcache/Python/3.12.7/x64/lib/python3.12/site-packages (from pytest) (2.0.0)
Requirement already satisfied: packaging in /opt/hostedtoolcache/Python/3.12.7/x64/lib/python3.12/site-packages (from pytest) (24.1)
Requirement already satisfied: pluggy<2,>=1.5 in /opt/hostedtoolcache/Python/3.12.7/x64/lib/python3.12/site-packages (from pytest) (1.5.0)
============================= test session starts ==============================
platform linux -- Python 3.12.7, pytest-8.3.3, pluggy-1.5.0
rootdir: /home/runner/work/Ase2425Basic/Ase2425Basic
collected 16 items

test_app.py F...............                                             [100%]

=================================== FAILURES ===================================
________________________________ test_add_valid ________________________________

client = <FlaskClient <Flask 'app'>>

    def test_add_valid(client):
        response = client.get('/add?a=10&b=5')
        assert response.status_code == 200
>       assert response.json['s'] == 15.0
E       assert 5.0 == 15.0

test_app.py:13: AssertionError
=========================== short test summary info ============================
FAILED test_app.py::test_add_valid - assert 5.0 == 15.0
========================= 1 failed, 15 passed in 0.19s =========================
Error: Process completed with exit code 1.
```

Clicking on the failed one and then on the job's name you can understand the reason of the failure.

Note: this is an example with pytest, you will use Postman.

When a workflow fails…

You can play with the repo settings to put rules about workflows execution.
For example you can stop a merge if a workflow fails (you need to use an event prior to the merge, i.e. not ‘push’).

Today’s Lab

Use `microase` code (download it or use yours) or work in group on your project code.

LAB TODO

1. Create or re-use a public GitHub repository.
2. Enable GitHub Actions.
3. Define a CI pipeline that on each push:
   1) Performs unit tests on microservices in isolation.
   2) Builds the microservice architecture.
   3) Performs tests on the whole architecture.

   Use Newman to run Postman collections.
4. Test the pipeline (a.k.a. workflow).

Note: if you use my code I put 2 bugs to avoid passing the tests ☺

Pipeline detailed steps

1. Checkout the code
2. Set up Node
3. Install Newman
4. Set up Docker

For the microservices `calc` and `string`:

    1. Build docker image to test in isolation
    2. Run the containter in detached mode
    3. Run tests with Newman
    4. Stop the container (this steps should be done even if other steps fail)

5. Build the overall architecture
6. Run the overall architecture in detached mode
7. Run tests with Newman

The microase folder

The content is:

- `src/` with all the code of `microase`.
- `tests/` with 3 Postman collections: one to test `calc`, one to test `string`, and one to test the overall architecture.

All the tests are done toward `localhost:5000`

Bonus stage

Modify the workflow to:

- Run unit tests in parallel jobs.
- Run the integration test after them.

Note: when you use different jobs, each job needs its setup.

Lab take away

- ❑ Familiarise with CI/CD.
- ❑ Define a CI pipeline and write a GitHub Actions workflow.
- ❑ Test the pipeline and resolve bulding errors.

Project take away

- ❑ In the project you will have to write a workflow to perform unit tests and integration tests when updating the code.