Pipeline

Document metadata: LO5 5.2 EVI 1

Contents:

General

Concept: Constructing a CI pipeline is a process of implementing automated testing.

Motivation: Automated testing and analysis is effective both for quickly and cheaply checking simple properties, and for more expensive checks that are necessary for critical properties that resist cheaper forms of verification (Y&P, 447).

Description of CI Pipeline Suited for Our System

Repository environment: For repository management and as a git UI client, we use GitHub and not GitLab. Since the provided sample project is configured for GitLab and corresponding runners, we would follow an initial GitHub workflow draft, using a nodejs env:

```
# This workflow will do a clean installation of node dependencies, cache/restore them, build the source code and run tests across diff

# For more information see: https://docs.github.com/en/actions/automating-builds-and-tests/building-and-testing-nodejs

name: Node.js CI

on:

push:

branches: [ "main" ]

pull_request:

branches: [ "main" ]

jobs:

build:

runs-on: ubuntu-latest

strategy:

matrix:

node-version: [18.x]

# See supported Node.js release schedule at https://nodejs.org/en/about/releases/


steps:

- uses: actions/checkout@v3

- name: Use Node.js ${{ matrix.node-version }}

uses: actions/setup-node@v3

with:

node-version: ${{ matrix.node-version }}

cache: 'npm'

- run: cd website && npm ci

- run: npm run build --if-present
```

```
- run: npm test
```

Potential improvements tailored towards our system

- We could use multiple node versions for the runners to check for backwards and forward compatibility of used node modules in the system by changing the workflow file:
  `node-version: [14.x, 16.x, 18.x]`
- If the CI testing would take to long, we could detect code pushes which do not change any code but only content (e.g. links, names, titles) and not trigger the embedded testing for those code changes. This would need to be considered carefully as for some systems, a change of a link, for instance, could still be fatal.

Adequacy of the pipeline

- **Why is this pipeline suited for the system?**
  This pipeline takes into account the importance of availability of the system. Checking for build and embedded tests at every code push is relevant as the system is part of a research project where the web service the system is deployed to needs to be continuously available. The CI pipeline is a first step of ensuring so; if the schedule would have allowed us we would have constructed a CD pipeline on the side for further availability integrity. Furthermore, the pipeline suits the developer's DevOps workflow.

[Excellence Items] Proposed Further Changes to Pipeline and Embedded testing which Exceed Course Content

N.B. For each of these proposals, we would need to check individually how adequate they are for the given system.

1. Implement automated security testing: Include security testing tools such as OWASP ZAP or Snyk in our pipeline to automatically check for vulnerabilities and security issues.
2. Incorporate further performance testing: Use tools like Apache JMeter or Gatling to automatically test the performance of our system and identify any bottlenecks or issues.
3. Implement chaos engineering: Incorporate chaos engineering techniques to test the resilience and robustness of our system under unexpected conditions.
4. Automated testing on multiple platforms: Test our system on multiple platforms to ensure that it works correctly on all of them. For this system, this could mean checking for browser compatibility.
5. Auto-scaling test environment: Create an auto-scaling environment that can run tests against different number of users, to check the performance of the application under varying loads.
6. Automated canary releases: Use canary releases, where we deploy a small portion of new version of the system, to a subset of users, and monitor the behaviour of the release before rolling out to the entire user base.
7. [If using more sophisticated scheduling unit with prediction functionality at some point] Machine learning testing: Incorporate machine learning testing to ensure that our models are working correctly and that they are not biased.
8. Build a chatbot that interacts with our pipeline and provides information on pipeline status, alerts, and results.