DevOps Report

Our team :

- LE Vu Thuy Tu
- VU Thanh Tung
- NGUYEN Le Hoang

In this DevOps report, we will provide an overview of the technologies employed and clarify our implementation for automating of our DevOps processes within the project.

Outline

Tools explaniation

Test

Back-end

- **Code convention** : We use Checkstyle (an open source tool that checks code against a configurable set of rules)
- **Unit test** : We use JUnit (a unit test tool)
- **Linters** : We use SpotBugs (a static analysis tool that help identify potential bugs in Java code)
- **Code coverage** : We use JaCoCo (a code coverage reports generator for Java projects)
- **Security analysis** : we use OWASP (a dependency-check plugin)

We include all the mentioned plugins in the pom.xml file of our Maven project. These plugins will be executed during the packaging of the Maven project in test phase by running the following command:

```
mvn clean package spring-boot:repackage
```

Front-end

- **Front-end unit test** : During front-end unit testing, we simulate the behavior of the backend and its REST API for the purpose of testing. `Jasmine` and `Karma` serve as our testing tools for this process. In this context, we employ mock techniques to create simulated versions of the backend and its API, allowing us to assess the functionality and responsiveness of the front-end components in isolation. This approach helps ensure that the front-end elements interact seamlessly with the backend and that the application as a whole functions as intended. To run our unit tests :

```
cd/game/game-frontend
npm run test
```

- **System test (E2E testing)** : We employ `Cypress` for comprehensive end-to-end testing, including four distinct test types:
  - *Full Play Test*: Validate the entire player experience by testing the complete game cycle (end2end test). This includes entering a player name, selecting the play option with suggestions, choosing a level (here is easy level), and engaging in Sudoku gameplay until completion and return to the menu page
  - *Error Display Test*: Verify error handling by examining if the system correctly displays errors (red color in invalid cases) when the same number is present in a row, column, or sub-grid.
  - *Suggestion Test*: Check if the suggestions mode is running.
  - *Get Board Test*: Verify if the game can get the board from api or generate a new board.
  - *Name Input Test*: Check if the player input name function is working.

  To execute `Cypress` , follow these steps:

  You are currently projet-cpoo-thuy-tu-thanh-tung

  - Initiate the frontend application, open a new terminal:

```
cd/game/game-frontend
ng serve
```

  - In a separate terminal, run `cypress` :

```
cd/game/cypress
npm start
```

  - Choose E2E Testing, select Chrome and then click on start button, there will be 5 tests for you to choose
  - We have created a video demonstration of a successful test, which is stored in the `game-doc` directory. This video serves as a reference in case you encounter anormal issues when running Cypress tests : https://gitlab.insa-rennes.fr/devops-2023/projet-cpoo-thuy-tu-thanh-tung/-/blob/main/game-doc/video_cypress.webm?ref_type=heads

- **Code convention and linters** : In adhering to our established code conventions and ensuring code quality in the frontend development, we employ `ESLint` as our designated linter. `ESLint` plays an important role in analyzing and enforcing coding standards, enhancing the maintainability and readability of our codebase. To initiate the `ESLint` process :

  ```
  npm run eslint
  ```

  You can see our predefined rules in this file `.eslintrc.json` (in `game-frontend` directory) We don't run eslint in the pipeline because we have errors from the previous cpoo project which should take a lot of time to fix all the style errors

- **Code coverage** : The specification of the percentage is established within the file named `karma.config.js` . To run test with code coverage :

  ```
  npm run test --code-coverage
  ```

- **Security analysis** : We use `Snyk` (a platform and tool designed to help developers find, fix, and monitor vulnerabilities in their open-source dependencies). How to run Snyk:

  ```
  snyk auth $SNYK_TOKEN
  ```

  ```
  snyk test
  ```

Global security

- **Git leaks**: To enhance global security in DevOps, we employ GitLeaks to identify commits that may inadvertently disclose sensitive information. Our approach involves utilizing a basic configuration for effective detection.

Build

We modified `.gitlab-ci.yml` file to specify how the back and the front are built and customised it for docker. We precise in detail this stage in Pipeline Explanation part.

Release

We authored two Dockerfiles to initiate the backend and frontend as microservices.

For the frontend, initially, we crafted an NGINX configuration file (NGINX acting as a web server) to serve as a reverse proxy. The configuration details can be found in nginx.conf within the game-

frontend directory.

In terms of continuous delivery, upon each successful build, the Docker images are uploaded to the GitLab INSA Docker instance registry.

As an alternative to using an NGINX server proxy, we have transitioned to employing a Kubernetes Ingress. This is defined in the k8s-deployment.yml file.

Deploy

For our deployment processes, we leverage `MicroK8s`. We've authored two manifest files, namely `k8s-namespace.yml` and `k8s-deployment.yml`, designed to deploy our backend and frontend on Kubernetes servers. The document provides comprehensive steps for executing the deployment : https://gitlab.insa-rennes.fr/devops-2023/projet-cpoo-thuy-tu-thanh-tung/-/blob/main/README.md?ref_type=heads

Pipeline explaniation

Our GitLab CI pipeline consists of three stages: `leaks`, `test`, `build`, and `docker-build`. Each stage has specific jobs related to security checks, code compilation, and Docker image building.

Stage 1: Leaks

This stage checks for sensitive information leaks using the gitleaks tool.

Job: gitleaks

- Uses the `zricethezav/gitleaks` Docker image.
- Executes the gitleaks tool to scan for leaks in the repository.
- Includes commands to protect the repository using gitleaks.

Stage 2 : Test

Job: test-frontend

This job tests the frontend of the project using the Node.js image (insa-node:16.13.2). It installs Node.js dependencies, installs Chromium, runs Snyk for security checks, and runs frontend tests using ChromeHeadless. `Eslint` is deactivated due to lingering errors from the preceding CPoo project, as resolving all style errors would be time-consuming. Additionally, the backend tests are skipped, as Maven is responsible for conducting tests prior to packaging.

Stage 3 : Build

This stage involves building the backend and frontend components of the application.

Job: build-back

- Uses a custom Maven image for building the backend.
- Compiles and packages the backend code using Maven. Test Backend included in this step. Before packaging, Maven will test with `CheckStyle`, `SpotBugs`, `JaCoCo`, `JUnit` and `owasp` to verify all test passed.
- Specifies rules for triggering the job based on changes in relevant files.
- Generates artifacts such as Jacoco coverage report and JAR file.

Job: build-front

- Uses a custom Node.js image for building the frontend.
- Installs dependencies, builds the frontend, and generates artifacts.

Stage 4 : Docker Build

This stage involves building Docker images for the frontend and backend components.

Job: build-job-docker-frontend

- Depends on the `build-front` job.
- Logs in to the INSA Docker registry.
- Builds a Docker image for the frontend using the artifacts from the previous stage.
- Pushes the Docker image to the INSA Docker registry.

Job: build-job-docker-backend

- Depends on the `build-back` job.
- Logs in to the INSA Docker registry.
- Builds a Docker image for the backend using the artifacts from the previous stage.
- Pushes the Docker image to the INSA Docker registry.

Difficulties encountered

In the microk8s deployment process, we've successfully deployed both the frontend and backend on the server. Both applications are functional; the frontend pod can successfully make curl requests to the backend. However, we are currently facing challenges in configuring Nginx to properly redirect to the backend service. This issue remains unresolved at the moment.