Automation Testing with Docker and CI/CD

Project Overview

This project demonstrates automation testing using Docker and CI/CD principles, aimed at making software testing efficient, scalable, and repeatable. It was developed as part of my learning experience in the "Proficient Automation Tester by Leveraging Docker with CI/CD" course on Udemy.

Key Concepts Covered

Docker: Containerized test environments for easy setup and consistency across different environments.

CI/CD Pipelines: Integration of automated testing into Continuous Integration and Continuous Delivery pipelines for faster feedback and better software quality.

Automation Testing: Streamlining testing processes to ensure faster and more reliable test execution.

Framework Structure

This project is a Selenium Test Automation Framework designed to run tests on different platforms such as LambdaTest and Selenium Grid, with the flexibility to configure execution dynamically using Maven commands.

```
selenium-docker/
|-- .github/
|   └── workflows/
|       └── maven.yml              # GitHub Actions CI/CD workflow for running tests
|-- docker/
|   ├── docker-compose.yml         # Docker configuration for standalone browser execution
|   └── docker-compose-grid.yml    # Docker configuration for Selenium Grid setup
|-- logs/                          # Directory to store test logs
|-- reports/                       # Directory to store test execution reports
|-- screenshots/                   # Directory to store failure screenshots
|-- src/
|   ├── main/
|   |   ├── java/
|   |   |   ├── base/
|   |   |   |   ├── BasePage.java      # Base class for page objects
|   |   |   |   └── BaseTest.java      # Base class for test execution setup
|   |   |   ├── config/
|   |   |   |   ├── config.properties  # Configuration file for test settings
|   |   |   |   └── ConfigLoader.java  # Utility to load configuration
|   |   |   ├── page_objects/
|   |   |   |   ├── LoginPage.java     # Page object for login functionality
|   |   |   |   └── ProductsPage.java  # Page object for products functionality
|   |   |   ├── utils/
|   |   |   |   ├── ExtentReport.java              # Extent report logging utilities
|   |   |   |   └── WebElementsInteractions.java   # Common interactions with web elements
|   |   └── resources/
|   |       └── log4j2.xml           # Logging configuration
|   └── test/
|       ├── java/
|       |   ├── test_cases/
|       |   |   ├── LoginTest.java       # Test case for login functionality
|       |   |   └── ProductTest.java     # Test case for product pages functionality
|-- target/                        # Compiled output and generated reports
|-- .gitignore                     # Git ignore file for unnecessary files
|-- customtestng.xml               # Custom TestNG suite XML
|-- testng.xml                     # Default TestNG suite XML
|-- pom.xml                        # Maven project configuration file
```

 : Shivam Kumar

Features

Cross-platform execution: Run tests on local Selenium Grid or cloud platforms like LambdaTest.

Parallel execution: Configure parallel test execution via TestNG.

Logging and reporting: Uses Log4j2 for logging and Extent Reports for detailed reporting.

Docker support: Execute tests using Docker containers for isolated environments.

Config-driven execution: Modify config.properties to change test parameters dynamically.

GitHub Actions Integration: Leverage an integrated GitHub Actions workflow to automate testing, streamline CI/CD pipelines, and provide continuous feedback on code changes.

Prerequisites

Ensure you have the following installed:

Java (JDK 11 or above)

Apache Maven

Docker (for running Selenium Grid)

TestNG (included via Maven dependencies)

Setup and Execution

1. Clone the repository:

git clone https://github.com/QA-Shivam/selenium-docker.git

cd selenium-docker

2. Run tests using Maven:

mvn clean test -Dplatform=local

3. Run tests on Selenium Grid using Docker:

docker-compose -f docker/docker-compose-grid.yml up -d

mvn clean test -Dplatform=seleniumgrid

4. Run tests on LambdaTest:

mvn clean test -Dplatform=lambdatest -Dbrowser=chrome -Drunmode=remote

5. Generate reports: *Reports are generated in the reports/ folder after test execution.*

Configuration

The framework uses a config.properties file to control test execution settings. Key parameters include:

browser=chrome

platform=local

gridURL=http://localhost:4444/wd/hub

Modify these values based on your test environment requirements.

Logging

Logging is handled via Log4j2, with the configuration specified in log4j2.xml. Logs are saved in the logs/ directory.

Reporting

Test execution results are captured using Extent Reports, providing a detailed, visually appealing report.

: Shivam Kumar