Document Strategy:

Improving Test Efficiency with GitHub Actions, Docker, and Selenium Grid

V01

JuanTous 11/24

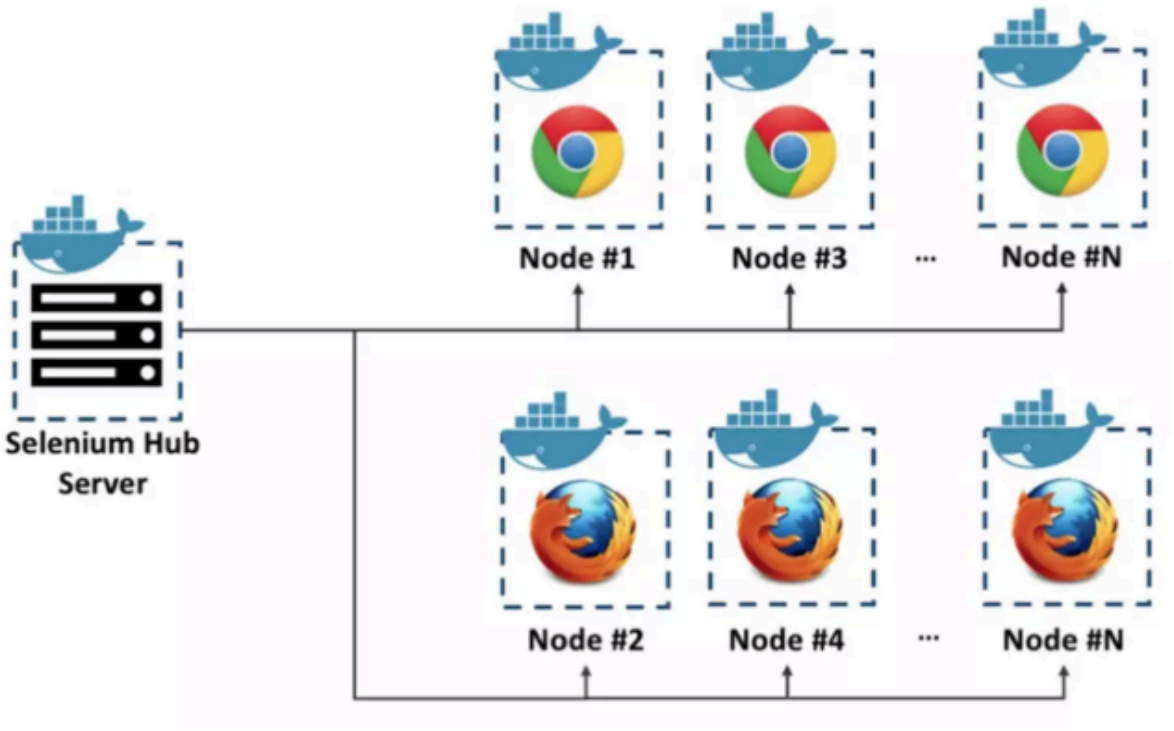
@Context

1. Introduction

In today's fast-paced software development environment, where DevOps practices are essential, the need for efficient and reliable testing strategies is critical. This document outlines a strategy to enhance test efficiency through parallel testing using GitHub Actions, Docker, and Selenium Grid. A key focus of this strategy is the execution of various regression test suites specifically on the Safari browser. This ensures that our applications not only perform seamlessly across all major browsers but also meet the unique requirements and behaviors of Safari users.

However, it is important to note that configuring the Safari driver in Selenium can be challenging.

Fortunately, the architecture of our testing strategy allows for the execution of the test suite from any operating system, thanks to the use of jobs in GitHub Actions. This flexibility enables us to run our tests in diverse environments, ensuring comprehensive coverage and compatibility across different platforms. Additionally, with a growing number of users accessing applications on mobile devices, testing on Safari allows us to validate touch interactions and gestures that are critical for a seamless mobile experience. Leveraging Safari's developer tools can also aid in debugging and performance optimization, ensuring that our applications not only function correctly but also deliver high performance across all platforms. By incorporating Safari testing into our regression suite, we can detect potential issues early in the development cycle, ultimately leading to a more reliable and user-friendly product.



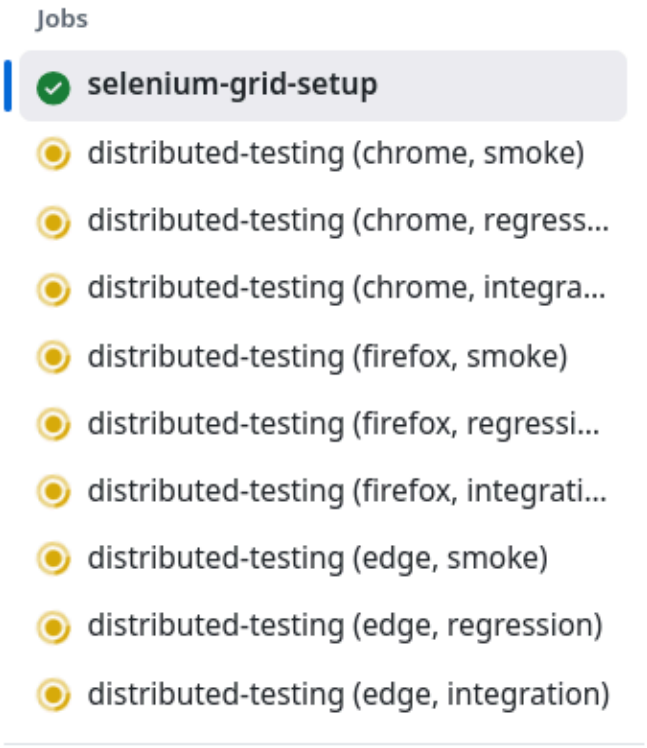
2. Objectives

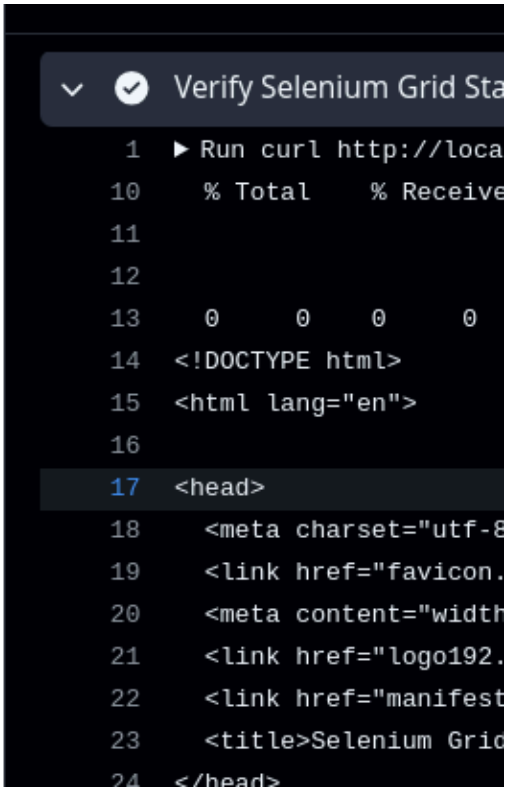
- Enhance Test Efficiency: Utilize Selenium Grid to run tests in parallel across different browsers and operating systems.
- Automate Testing Process: Leverage GitHub Actions for continuous integration and continuous deployment (CI/CD) to automate the testing workflow.
- Containerization: Use Docker to create isolated environments for tests, minimizing dependency issues and ensuring consistency across different environments.
- Simplify Configuration: Address the complexities associated with browser driver configurations, particularly for Safari, across various operating systems.

3. Strategy Overview

3.1 Selenium Grid Gh Actions

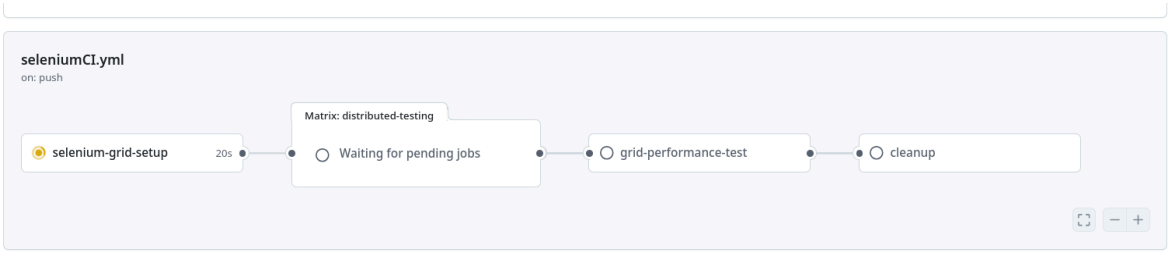
- Parallel Testing: Set up a Selenium Grid to allow multiple tests to run concurrently across different browsers and OS configurations.
- Scalability: Easily scale the grid by adding more nodes as needed, accommodating increased testing demands.





3.2 GitHub Actions

- CI/CD Integration: Create workflows in GitHub Actions to automate the execution of tests on code commits, pull requests, and merges.
- Custom Workflows: Define custom workflows that can trigger tests based on specific events or schedules.
- Event-Driven Automation: Workflows can be triggered by various events, ensuring immediate feedback for developers.
- Parallel Execution: Run multiple jobs simultaneously, drastically reducing the time required to execute tests.



3.3 Docker

- Containerized Testing: Use Docker to encapsulate the test environment, ensuring that all dependencies are included and versions are controlled.
- Cross-Platform Compatibility: Build Docker images for different OS/browser combinations to overcome local resource limitations and driver compatibility issues.

4. Implementation Steps

4.1 Set Up Selenium Grid

1. Install Selenium Grid: Deploy a Selenium Grid hub and configure nodes for various browsers (Chrome, Firefox, Safari) and operating systems (Windows, macOS, Linux).
2. Configure Node Capabilities: Define the capabilities for each node to ensure the correct browser versions and configurations are used.

4.2 Configure Docker

1. Create Docker Images: Develop Dockerfiles for each browser/OS combination required for testing.
2. Docker Compose: Utilize Docker Compose to orchestrate the Selenium Grid setup, including the hub and nodes.

3. Environment Variables: Use environment variables to customize configurations for different testing scenarios.

4.3 Set Up GitHub Actions

1. Create Workflow File: Define a `.github/workflows/test.yml` file to outline the testing workflow.
2. Trigger Events: Configure the workflow to trigger on specific events such as push, pull request, or on a schedule.
3. Run Tests: Use the Docker images to run tests against the Selenium Grid, capturing results and artifacts.

Test Automation Strategy

Prepared by: [@JuanTM]

Position: Senior QA Automation Engineer

Date: [Insert Date]

Executive Summary

This document outlines a comprehensive Test Automation Strategy designed to enhance the efficiency, reliability, and scalability of our testing processes. By leveraging modern technologies such as GitHub Actions, Docker, and Selenium Grid, we aim to

create a robust framework that supports continuous integration and continuous delivery (CI/CD), ultimately improving software quality and accelerating release cycles.

1. Introduction

In today's fast-paced software development environment, the need for efficient and effective testing practices is paramount. This strategy serves as a roadmap for implementing an automated testing framework that aligns with our organizational goals and addresses the challenges faced by the QA team.

2. Objectives

- Increase Test Coverage: Ensure comprehensive testing across multiple application components and user scenarios.
- Reduce Time to Market: Automate repetitive testing tasks to shorten the feedback loop and accelerate release cycles.
- Enhance Test Reliability: Minimize human error and environmental inconsistencies through standardized testing environments.
- Facilitate Continuous Integration: Integrate automated tests into the CI/CD pipeline to provide immediate feedback on code quality.

3. Scope

This strategy encompasses the following areas:

- Test Automation Framework Development
- Tool Selection and Integration
- Test Design and Implementation
- Maintenance and Continuous Improvement
- Reporting and Metrics

4. Test Automation Framework Development

4.1 Framework Design

- Modular Architecture: Design the framework with a modular approach to allow easy addition and maintenance of test cases.
- Reusable Components: Create reusable functions and libraries to streamline test case development and reduce redundancy.
- Data-Driven Testing: Implement data-driven testing to enhance test coverage and validate various input scenarios.

4.2 Tool Selection

- Selenium Grid: Utilize Selenium Grid for cross-browser testing, enabling parallel execution of tests across different environments.
- Docker: Leverage Docker for containerization, ensuring consistent test environments and simplifying setup and teardown processes.
- GitHub Actions: Integrate GitHub Actions for CI/CD automation, allowing for automatic execution of tests on code commits and pull requests.

4. Test Automation Framework Development

4.1 Framework Design

- Modular Architecture: Design the framework with a modular approach to allow easy addition and maintenance of test cases.
- Reusable Components: Create reusable functions and libraries to streamline test case development and reduce redundancy.
- Data-Driven Testing: Implement data-driven testing to enhance test coverage and validate various input scenarios.

4.2 Tool Selection

- Selenium Grid: Utilize Selenium Grid for cross-browser testing, enabling parallel execution of tests across different environments.
- Docker: Leverage Docker for containerization, ensuring consistent test environments and simplifying setup and teardown processes.
- GitHub Actions: Integrate GitHub Actions for CI/CD automation, allowing for automatic execution of tests on code commits and pull requests.

4.3 Set Up GitHub Actions

In this section, we will define the workflow for our test automation strategy using GitHub Actions. The workflow is designed to set up a Selenium Grid and run distributed tests across different browsers, including Safari, Chrome, Firefox, and Edge.

Create Workflow File: Define a `.github/workflows/test.yml` file to outline the testing workflow. Below is the YAML configuration for the workflow:

```yaml
name: Selenium Grid Distributed Testing

on:
 push:
   branches: [ master ]
 pull_request:
   branches: [ master ]
 workflow_dispatch:

env:
 JAVA_VERSION: '22'
 SELENIUM_GRID_VERSION: '4.19.0'

jobs:
 # Job to set up the Selenium Grid
 selenium-grid-setup:
   runs-on: ubuntu-latest
   steps:
     # 1. Checkout Repository
     - name: Checkout Repository
       uses: actions/checkout@v4

     # 2. Setup Java
     - name: Set up Java
       uses: actions/setup-java@v4
       with:
         java-version: ${{ env.JAVA_VERSION }}
         distribution: 'temurin'
         cache: 'maven'

     # 3. Setup Docker
     - name: Set up Docker
       uses: docker/setup-buildx-action@v3
```

```
    # 4. Pull Selenium Grid Images
    - name: Pull Selenium Grid Images
      run: |
        docker pull selenium/hub:${{ env.SELENIUM_GRID_VERSION }}
        docker pull selenium/node-chrome:${{ env.SELENIUM_GRID_VERSION }}
        docker pull selenium/node-firefox:${{ env.SELENIUM_GRID_VERSION }}
        docker pull selenium/node-edge:${{ env.SELENIUM_GRID_VERSION }}

    # 5. Start Selenium Grid
    - name: Start Selenium Grid
      run: |
        docker network create grid

        # Start Selenium Hub
        docker run -d --net grid --name selenium-hub \
          -p 4442:4442 -p 4443:4443 -p 4444:4444 \
          selenium/hub:${{ env.SELENIUM_GRID_VERSION }}

        # Start Chrome Nodes
        docker run -d --net grid --name chrome-node-1 \
          -e SE_EVENT_BUS_HOST=selenium-hub \
          -e SE_EVENT_BUS_PUBLISH_PORT=4442 \
          -e SE_EVENT_BUS_SUBSCRIBE_PORT=4443 \
          selenium/node-chrome:${{ env.SELENIUM_GRID_VERSION }}

        docker run -d --net grid --name chrome-node-2 \
          -e SE_EVENT_BUS_HOST=selenium-hub \
          -e SE_EVENT_BUS_PUBLISH_PORT=4442 \
          -e SE_EVENT_BUS_SUBSCRIBE_PORT=4443 \
          selenium/node-chrome:${{ env.SELENIUM_GRID_VERSION }}

        # Start Firefox Nodes
        docker run -d --net grid --name firefox-node-1 \
          -e SE_EVENT_BUS_HOST=selenium-hub \
          -e SE_EVENT_BUS_PUBLISH_PORT=4442 \
          -e SE_EVENT_BUS_SUBSCRIBE_PORT=4443 \
          selenium/node-firefox:${{ env.SELENIUM_GRID_VERSION }}

        # Start Edge Nodes
        docker run -d --net grid --name edge-node-1 \
          -e SE_EVENT_BUS_HOST=selenium-hub \
          -e SE_EVENT_BUS_PUBLISH_PORT=4442 \
          -e SE_EVENT_BUS_SUBSCRIBE_PORT=4443 \
          selenium/node-edge:${{ env.SELENIUM_GRID_VERSION }}
```

```
        # Start Safari Nodes
        docker run -d --net grid --name safari-node-1 \
          -e SE_EVENT_BUS_HOST=selenium-hub \
          -e SE_EVENT_BUS_PUBLISH_PORT=4442 \
          -e SE_EVENT_BUS_SUBSCRIBE_PORT=4443 \
          selenium/node-safari:${{ env.SELENIUM_GRID_VERSION }}

        docker run -d --net grid --name safari-node-2 \
          -e SE_EVENT_BUS_HOST=selenium-hub \
          -e SE_EVENT_BUS_PUBLISH_PORT=4442 \
          -e SE_EVENT_BUS_SUBSCRIBE_PORT=4443 \
          selenium/node-safari:${{ env.SELENIUM_GRID_VERSION }}

        docker run -d --net grid --name safari-node-3 \
          -e SE_EVENT_BUS_HOST=selenium-hub \
          -e SE_EVENT_BUS_PUBLISH_PORT=4442 \
          -e SE_EVENT_BUS_SUBSCRIBE_PORT=4443 \
          selenium/node-safari:${{ env.SELENIUM_GRID_VERSION }}

        docker run -d --net grid --name safari-node-4 \
          -e SE_EVENT_BUS_HOST=selenium-hub \
          -e SE_EVENT_BUS_PUBLISH_PORT=4442 \
          -e SE_EVENT_BUS_SUBSCRIBE_PORT=4443 \
          selenium/node-safari:${{ env.SELENIUM_GRID_VERSION }}

        docker run -d --net grid --name safari-node-5 \
          -e SE_EVENT_BUS_HOST=selenium-hub \
          -e SE_EVENT_BUS_PUBLISH_PORT=4442 \
          -e SE_EVENT_BUS_SUBSCRIBE_PORT=4443 \
          selenium/node-safari:${{ env.SELENIUM_GRID_VERSION }}

        # Wait for Grid to be ready
        sleep 20
        docker ps

    # 6. Verify Grid Status
    - name: Verify Selenium Grid Status
      run: |
        curl http://localhost:4444/ui/index.html#/
        docker logs selenium-hub

# Job to run distributed tests
distributed-testing:
  needs: selenium-grid-setup
```

```yaml
  strategy:
    matrix:
      browser: [chrome, firefox, edge]
      test-group: [smoke, regression, integration]
  runs-on: ubuntu-latest
  steps:
    # 1. Checkout Repository
    - name: Checkout Repository
      uses: actions/checkout@v4

    # 2. Setup Java
    - name: Set up Java
      uses: actions/setup-java@v4
      with:
        java-version: ${{ env.JAVA_VERSION }}
        distribution: 'temurin'
        cache: 'maven'

    # 3. Maven Dependencies
    - name: Install Dependencies
      run: mvn clean install -DskipTests

    # 4. Run Distributed Tests
    - name: Run Distributed Selenium Grid Tests
      env:
        BROWSER: ${{ matrix.browser }}
        TEST_GROUP: ${{ matrix.test-group }}
      run: |
        mvn test \
          -Dselenium.grid.url=http://localhost:4444/wd/hub \
          -Dbrowser=$BROWSER \
          -Dtest.group=$TEST_GROUP

          # 5. Upload Test Results
    - name: Upload Test Results
      if: always()
      uses: actions/upload-artifact@v4
      with:
        name: test-results-${{ matrix.browser }}-${{ matrix.test-group }}
        path: |
          target/surefire-reports
          target/screenshots
        retention-days: 5
```

5. Test Design and Implementation

5.1 Test Case Development

- Identify Test Scenarios: Collaborate with stakeholders to identify critical test scenarios based on business requirements and user stories.
- Prioritize Test Cases: Prioritize test cases based on risk assessment, focusing on high-impact areas of the application.
- Implement Automation: Develop automated test scripts using industry-standard programming languages and frameworks (e.g., Java, Python, TestNG, or JUnit).

5.2 Test Execution

- Continuous Testing: Integrate automated tests into the CI/CD pipeline to ensure tests are executed automatically with each code change.
- Parallel Execution: Leverage Selenium Grid to run tests in parallel, significantly reducing execution time.

6. Maintenance and Continuous Improvement

- Regular Review: Conduct regular reviews of the test suite to identify obsolete or redundant test cases and ensure alignment with current application features.
- Refactor Tests: Continuously refactor test scripts to improve maintainability and performance.
- Feedback Loop: Establish a feedback loop with developers and stakeholders to gather insights and improve the testing process.

7. Reporting and Metrics

- Test Reporting: Implement reporting tools to generate comprehensive reports on test execution results, defect rates, and code coverage.
- Key Performance Indicators (KPIs): Define KPIs to measure the effectiveness of the test automation strategy, including:
    - Test execution time
    - Defect detection rate

- Test coverage percentage
- Automation ROI

8. Conclusion

The proposed Test Automation Strategy aims to establish a robust framework that enhances our testing capabilities while supporting the overall goals of [Company Name]. By leveraging modern tools and methodologies, we can improve software quality, reduce time to market, and foster a culture of continuous improvement within the QA team.

With my extensive experience as a Senior QA Automation Engineer and Advanced DevOps practitioner, particularly in working with Docker, I have been able to build an effective YAML configuration file that orchestrates our testing processes seamlessly. This expertise allows us to utilize GitHub Actions for executing our test suite across various environments, ensuring that we maintain high standards of quality and compatibility, especially for critical browsers like Safari.

9. Next Steps

- Stakeholder Approval: Present this strategy to stakeholders for feedback and approval.
- Pilot Implementation: Initiate a pilot project to validate the framework and tools selected.
- Training Sessions: Organize training sessions for the QA team to familiarize them with the new tools and processes