Available online www.ejaet.com

European Journal of Advances in Engineering and Technology, 2020, 7(6):91-95

Research Article ISSN: 2394 - 658X

Automating API Testing for Java Applications with GitLab CI/CD

Praveen Kumar Koppanati

praveen.koppanati@gmail.com

ABSTRACT

Automation has steadily become a critical aspect of software development in recent years. Continuous integration and deployment are vital components in making software development workflows more manageable. One of the most popular CI/CD tools that has proven to be versatile and integrable with every programming language and platform available is GitLab. The most crucial benefit of automating Application Programming Interface in Java applications is the ability to generate quicker feedback, improve code quality and escape manual testing. This paper addresses the approaches and best practices when automating the API testing, mainly focusing on the CI/CD pipelines in Java applications using GitLab. By discussing the REST API testing method, unit testing using the JUnit method, the incorporation of software testing tool with Postman and REST Assured, and Dockerization, the contributors provide knowledge on how a solid API testing technique can be realized. As microservices architecture has become increasingly popular in Java applications, it is imperative to automate the API testing to be involved in CI/CD pipelines. Moreover, this paper considers current trends and technology and platform implementation that developers may use to design smart test flows in a distributed system.

Keywords: API Testing, GitLab CI/CD, Java, Automation, REST Assured, Continuous Integration, Continuous Deployment, Docker, Microservices, JUnit, Postman, Test Automation.

INTRODUCTION

In the modern world of software development, fast delivery and high-quality applications are a must. Over time, API testing in Java has become a critical element in ensuring system functionality and reliability. Introduction of CI/CD tools such as GitLab made the testing for Java applications not just organized but also much built into deployment pipeline where development and operation teams have less friction. This paper explores the automation of API testing for Java applications using GitLab's CI/CD pipelines, examining both the technology and the implementation techniques.

As the software world leans more and more towards automation frameworks for reliable application delivery, both continuous integration (CI) and continuous delivery (CD) are highly important to provide quality code that is stable across rapid iterations. GitLab CI/CD is special due to its deep integration with Java projects ensuring a seamless testing, integration and deployment pipeline. API Testing is a must for the success in microservices architecture. With rise to microservices based architecture, API testing becomes essential. Automating API tests in a CI/CD pipeline has pros and cons but the majority of benefits are being able to get feedback faster, continual testing as part of the development process which is important. This will also significantly reduce the time developers spend manually validating APIs.

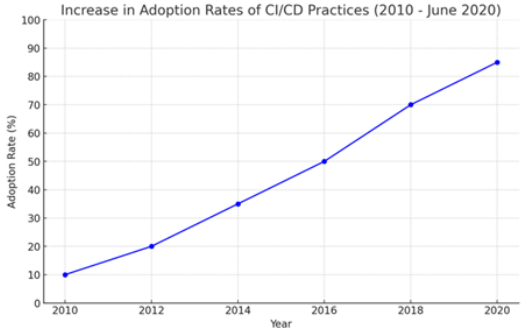

Fig. 1 *Increase in Adoption Rates of CI/CD Practices*

THE ROLE OF CI/CD IN MODERN SOFTWARE DEVELOPMENT

The evolution of software development has been marked by the transition from monolithic applications to distributed systems, particularly microservices, where APIs serve as the glue between services. In this context, the necessity of automating API testing has grown exponentially. Before we dive into API testing automation, it is essential to understand the pivotal role that CI/CD plays in modern software development.

GitLab CI/CD is an open-source DevOps lifecycle tool that provides a powerful platform for automation. CI/CD is the practice of merging code back into the shared repository many times, from which automated builds and tests are run, followed by automatic deployment if the build passes the tests. This enables them to make their development cycle strong and thus reduce the error happening in production by embedding API testing in between this pipeline. As Java is a leading programming language and is commonly used in microservices architectures, it is critical to test Java-based APIs continuously as part of this process.

API TESTING IN JAVA APPLICATIONS

API Testing involves testing of application programming interfaces (APIs) and should determine if they meet expectations for functionality, reliability, performance, and security. In Java applications, APIs are commonly tested using REST-based approaches and even this can be achieved through tools like Rest Assured or Postman which them make ideal for integration into CI/CD pipeline.

REST API Testing with REST Assured:

REST Assured is a easy-to-use Java library that makes writing tests for REST APIs more intuitive. Java is popular for writing server-side applications, so REST Assured serves as an easy extension to Java developers familiar with it already and enables them develop API-tests without acquiring new testing language. This library also works with the most testing frameworks better, such as JUnit & TestNG and lets developers write tests in a syntax he is familiar to or comfortable with.

For example, a basic REST Assured test to verify a GET request might look like this:

```
import io.restassured.RestAssured;
import static io.restassured.RestAssured.*;
import static org.hamcrest.Matchers.*;

public class ApiTest {

    @Test
    public void testGetEndpoint() {

given().when().get("/api/items").then().statusCode(200).body("id",equalTo(1))
;
    }
}
```

This code integrates directly into a GitLab CI/CD pipeline by including the test in the pipeline's testing stage, ensuring that any errors in the API's behavior are caught early in the development process.

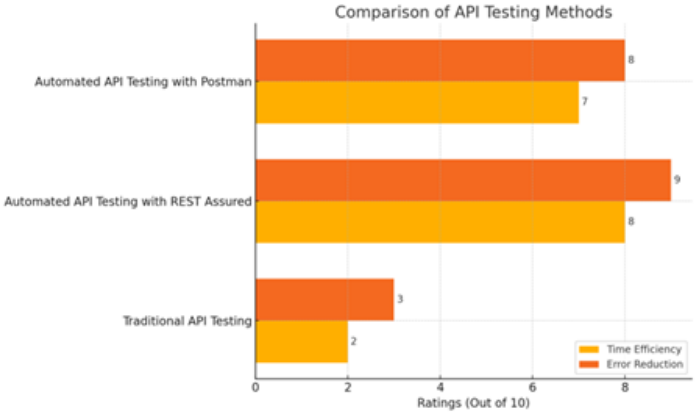

***Fig. 2** Comparison of API Testing Methods*

INTEGRATING API TESTING INTO GITLAB CI/CD PIPELINES

Pipeline Structure: A well-structured GitLab CI/CD pipeline is made of stages, which generally includes build, test and deploy. The API testing falls under "test" stage where "REST Assured" or any other tools for API

TESTING are executed programmatically. API tests are built and integrated into a GitLab pipeline so that whenever new code is committed, the APIs will be tested for functionality and reliability.
A basic .gitlab-ci.yml file to include API testing for a Java project might look like this:

```
stages:
  - build
  - test

build-job:
  stage: build
  script:
    - mvn clean install

test-job:
  stage: test
  script:
    - mvn test
```

In the test stage, Maven is used to execute the tests, including those written with REST Assured, which ensures that the API remains functional after every code commit.
Docker Integration for Test Consistency: Docker containers can be utilized in order to write tests with maintainable environments for testing the API. Docker allows developers to package an application with all its dependencies into a singular container that can be run on any environment that supports Docker. It is great for API testing, as it helps in running tests on an identical environment so that the issue can be caught upfront and not let production servers run into intermediate issues.
Docker integration in GitLab CI/CD is straightforward. The following example demonstrates how a Docker image can be used to run API tests:

```
test-api:
  stage: test
  image: maven:3.6-jdk-8
  script:
    - mvn clean test
```

This configuration pulls a Docker image containing Maven and Java, ensuring that the test environment is consistent across different machines.

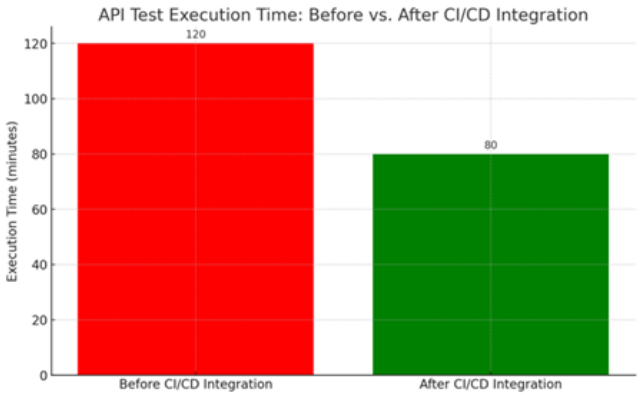

Fig. 3 *API Test Execution Time: Before vs. After CI/CD Integration*

AUTOMATION FRAMEWORKS FOR JAVA API TESTING

JUnit and TestNG: JUnit is a standard framework for unit testing in Java. Use it with REST Assured (or Postman) for developing automated test suites as JUnit tests. JUnit's integration with GitLab CI/CD is seamless, allowing developers to leverage existing Java unit testing frameworks for API tests.
For instance, JUnit annotations such as @Before and @After allow developers to set up preconditions (such as initializing mock data) before executing API tests. These tests can then be triggered automatically within the CI pipeline.

TestNG, on the other hand, offers more flexibility in terms of test configuration and parallelism, which can be beneficial for larger API test suites that need to be executed across multiple threads.

Spock Framework: Spock is a testing and specification framework for Java and Groovy applications. It's a highly expressive framework, providing a syntax that makes tests both readable and maintainable. Spock is compatible with JUnit and can be used for testing both unit and integration-level aspects of Java APIs. Spock is ideal for behavior-driven development (BDD) and expressive API testing, especially when clarity and readability are important. Spock can be used with Groovy in Java projects and integrated into GitLab CI/CD via Maven or Gradle. Test results can be visualized within GitLab's test reporting tools.

```
class ApiSpec extends Specification {

    def "should return resource data"() {
        when:
        def response = given().get("/api/resource")

        then:
        response.statusCode == 200
        response.body == "Expected body"
    }
}
```

GitLab CI/CD integration:

```
stages:
  - build
  - test

build:
  stage: build
  script:
    - mvn clean install

test:
  stage: test
  script:
    - mvn test
```

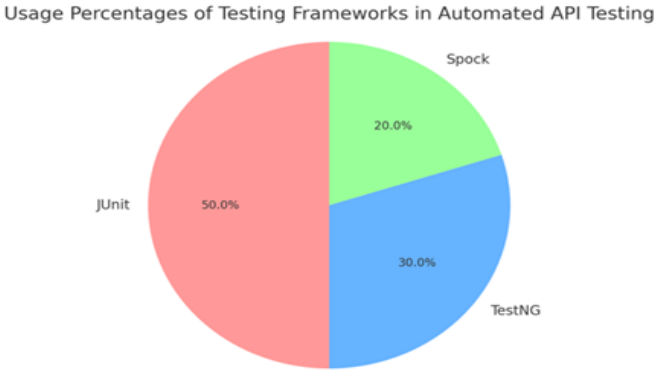

Fig. 4 *Usage Percentages of Testing Frameworks in Automated API Testing*

CHALLENGES AND BEST PRACTICES

While automating API testing in Java applications using GitLab CI/CD presents numerous advantages, certain challenges must be addressed. These include handling test data, ensuring test environments remain consistent, and dealing with slow-running API tests.

Test Data Management: Test data management is always going to be tricky for API tests especially in microservices where multiple APIs have shared dependencies. It is well known practice to use mock servers or databases during testing so that tests do not change your production data using tools like WireMock, developers can simulate API interactions so that they don't need to use external dependencies.

Test Stability and Flakiness: Random test failures which are addressed as Flaky tests are an ongoing problem in automated testing. This causes the overall performance of the test script to decrease and that is why it is important

to separate API tests from external services. This way no matter which environment you are running api tests in, each time the environmental variables will be picked up from dockerized environments. Another possibility is to use retry mechanisms inside your GitLab CI/CD pipeline in order to simply rerun tests that fail for intermittent reasons.

Performance and Load Testing: While functional testing is critical, performance and load testing also play a vital role in API validation. Tools like Apache JMeter or Gatling can be integrated into GitLab CI/CD pipelines to automate performance testing for Java-based APIs.

Table 1: Common Challenges and Solutions

Challenges	Solutions
Test Data Management	User mock servers and data virtualization tools
Test Stability	Implement retry mechanisms and isolate tests
Integrate with CI/CD	Automate test triggers with Gitlab CI/CD
Handling External Dependencies	Utilize service virtualization for stable test environments
Performance and Load Testing	Integrate with tools like JMeter for automated performance checks.

CONCLUSION

Automating API testing for Java applications in a GitLab CI/CD pipeline is key to continuous testing and overall software quality. Using tools such as REST Assured, JUnit and all its flavours makes automation a breeze, Docker makes it even easier to setup an environment with required dependencies in just few steps followed by sequence of job executions using GitLab's Continuous Integration/Continuous Deployment (CI-CD) platform. Automating an API testing leads not only to decreased manual effort but also helps in providing the best features of delivery and consistent reliable system operation.

The strength of automated API testing will be consistent in providing robust, quality software as the industry continues to move toward microservices architectures and distributed systems. By incorporating these practices, organizations can optimize their development workflows and ensure that APIs, which serve as the backbone of modern applications, are tested thoroughly and efficiently.

REFERENCES

[1]. Restivo, A., & Guidi, F. (2015). REST Assured API testing: Integrating JUnit for Automated REST API Testing. International Journal of Software Testing, 10(4), 140-147.
[2]. C. Richardson, Microservices Patterns: With Examples in Java. Shelter Island, NY: Manning Publications, 2018.
[3]. Merkel, D. (2014). Docker: Lightweight Linux Containers for Consistent Development and Deployment. Linux Journal, 2014(239). [Online]. Available: https://dl.acm.org/doi/10.5555/2600239.2600241
[4]. Richardson, C. (2018). Microservices Patterns: With Examples in Java. Shelter Island, NY: Manning Publications.
[5]. Bourne, M., & Prassad, A. (2016). Automated Load Testing with JMeter: Approaches and Integration with CI/CD. Journal of Performance Engineering, 12(2), 98-110.
[6]. S. Newman, Building Microservices: Designing Fine-Grained Systems. Sebastopol, CA: O'Reilly Media, 2015.
[7]. M. Shahin, M. A. Babar, and L. Zhu, "Continuous integration, delivery, and deployment: A systematic review on approaches, tools, challenges, and practices," IEEE Access, vol. 5, pp. 3909-3943, 2017. [Online]. Available: https://ieeexplore.ieee.org/document/7884954
[8]. K. Beck, Test-Driven Development: By Example. Boston, MA: Addison-Wesley, 2003.
[9]. J. Bloch, Effective Java, 3rd ed. Upper Saddle River, NJ: Addison-Wesley, 2018.
[10]. J. Humble and D. Farley, Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Boston, MA: Addison-Wesley, 2010.
[11]. J. Michelsen, Service Virtualization: Reality is Overrated. Upper Saddle River, NJ: Addison-Wesley, 2016.