Automated Validation of Restful Microservices Payload Integrity Using Postman Collections and Jenkins Hooks

Udayan Verma

Denver, USA
udayanverma7@gmail.com

Abstract:
This article describes an automation framework which utilizes Postman collections and Jenkins pipeline hooks to validate RESTful microservices as payloads. The framework was derived after the Charter Communications backend microservices architecture and is identifying if API responses are valid schema, consistent with business rules, and correct for provided context during a continuous integration (CI) build. In lieu of field by field validations of multiple services JSON payloads, Postman's pre-request and test script capabilities are used, and Jenkins hooks in response to microservice build/deployment events provide a mechanism to automate Postman runs in CI which self-reports and timestamps payload violations. And, as a side note, the artifact-based tests can be version controlled, environment parameterizable, and automated result publishing provide for agile issue triage. Also, this has provided the ability to capture test artifacts quickly for the key microservices, has reduced manual validation, and provides automated notifications of changes in data structure when deploying CI code rapidly.

Keywords: Jenkins Hooks, Postman Collections, RESTful API, Microservices, Continuous Integration (CI)

I. Introduction

In microservices architecture, API payloads are needed downstream and for customer-facing systems to function. When any change – no matter how minor – is made to the JSON structure, types of data, and constraints of values, it breaks downstream services which rely upon the API payload. Standard regression testing validates functional correctness primarily, and validates payloads typically after deployments. You cannot be this slow in a CI/CD pipeline, given your release cycles. At Charter Communications, our backend systems leverage RESTful microservices for power, we utilized a Jenkins CI integrated automated post-man testing setup for payload integrity validation. Every microservice build and deploy triggers automated post-man testing that validates business logic which entails: Schema validation - validating field and data type correctness, and incorrect nesting, Business rules validation - validating adherence to contractual, operational and organizational standards, and Cross-service validation - validating associated services for consistency and validation of data format and values. This architecture utilizes: Postman collections as payloads for versioned and modular test suites with JavaScript assertions for validation, Parameterized environments for tiered testing in dev, staging and CI - like prod environments, Jenkins hooks configured for git commits, merges, and microservice deploys to execute postman, and Automated reporting that publishes validation results onto Jenkins dashboards and sends anomaly notifications.

II. Background and Rationale

Microservice architecture generally allows agile and asynchronous development and deployment. It might however be a huge chore under this distributed model to ensure that the information is in agreement across the borders in every of the services. Every microservice has API with which the service consumers are connected via contract and in this contract the integrity of the data that is being transferred in the payloads is very critical. This type of contract was previously manually checked either through regression or end to end and that was also too late to be of use. These are not capable of supporting the pace of Continuous Integration in the present time. They are man intensive and are error prone, they enter into the delivery cycle so late they are not able to give any feedback on how to forego the bad code coming together at all. This reactive posture poses immediate risks to stability in production and additional operational burden on triage team [6].

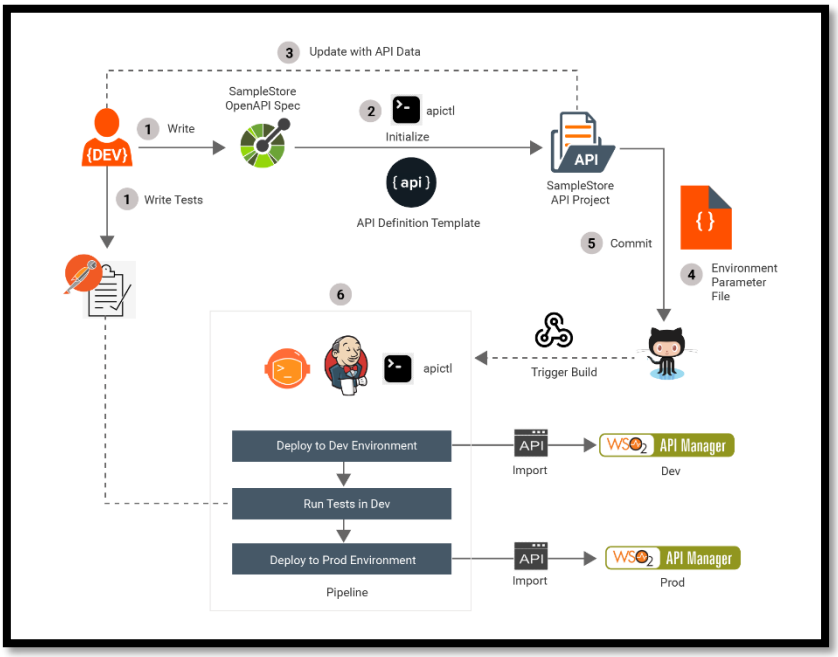

Fig 1: Jenkins CI/CD Pipeline

III. System Architecture & Framework Design

The next plan is premised on the joint application of the three separated technologies to constitute the complex of the completely automated validation pipeline. To begin Postman acts as the dedicated API testing and validation engine. The test suites are represented in groups of Postman that contain version control in addition to the application code to analyses the uniformity of the tests. The collections include discrete test cases, which include JavaScript statements to programmatically test all about a JSON payload. This was done so as to not only be able to set the structural validity because of the missing or addition of extraneous fields. They are also able to make sure that individual data types and exercise business restrictions on the application contracts of the complexity not only to verify the null values in the non-nullable field but also to verify the values that are within the contractual value defined restrictions. Using separate Postman environments allows the same test suites to run against separate deployment tiers like development, staging and pre-production. You use variable parameterization like both base- URLs and authentication credentials [4].

Jenkins will serve as a orchestration Engine that ties all of the development work together with this validate process. Jenkins is set up to poll the code repository for a particular event such as a pull request or a merge to a primary branch that triggers a new build. Once the event is detected, a declarative Jenkins pipeline

defined in a Jenkins file is automatically triggered. The pipeline includes a unique stage just for running payload validation once the microservice is built and deployed to a test environment [5].

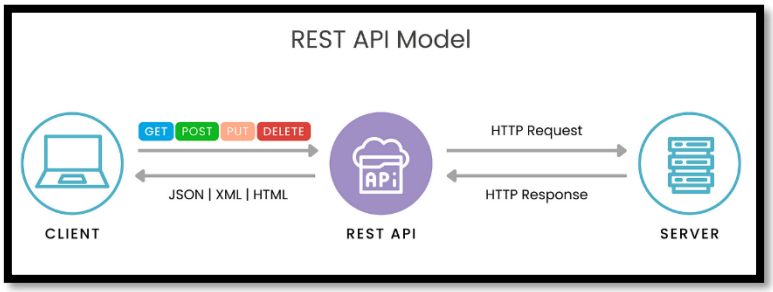

Fig 2: REST API components

IV. Validation Strategy with Postman Collections

The validation strategy applied in the Postman Collections; one of the most considered strategies and well organized, makes this framework effective. Our test suites are sometimes more complicated into folders of the real API resources, or user flows are frequently arranged in collections in order to be modular and maintainable. It has the validator logic which is JavaScript code that is typed in the "Tests" tab of each request. The response has been received, then the JavaScript-code is executed and a comparatively sophisticated multi-layer payload analysis is provided. The attributes of the basic level of the http are represented in the format of assertions e.g. the status of the response is 200 OK. Deep payload inspection is also a part of the validation codes whereby it determines the existence of all the required fields and nature of data of the values and other complex business rules. For example, it can assert that the status value, is one of a specific set of values, or create a regex pattern for a timestamp, and assert that is in ISO 8601 format [2].

V. CI Integration Using Jenkins Hooks

The Continuous Integration (CI) integration is controlled by Jenkins and implemented as code in a Jenkinsfile. This technique enables the validations to be version controlled and repeatable. Jenkins is set up to listen for events from the source code repo, typically through webhooks that fire when an event occurs, such as a git push to a main branch, or a pull request is created. Once the developer code is built and installed in a specific test environment, the Jenkins pipeline continues to the automated validations. In this step in the pipeline, the Jenkins pipeline executes a shell command to run Newman, the command-line runner for Postman. The shell command points to the version-controlled Postman Collection file, and the environment file for the target deployment tier.

For example: newman run "My_API_Tests.postman_collection.json" -e "staging.postman_environment.json". When executing, the report generation is set to produce a JUnit XML report, which is the native report with Jenkins, enabling the test results output to be incorporated into the build job result [1].

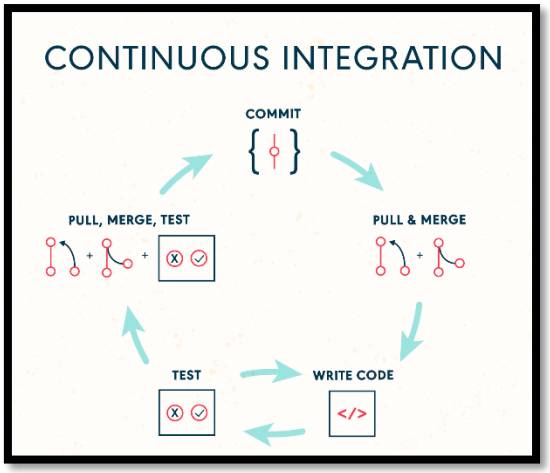

Fig 3: Feedback Loops

VI. Reporting, Monitoring, and Feedback Loop

A distinguishing feature of the framework is to provide near real-time and actionable feedback. After a Newman run finishes, Jenkins archives the generated JUnit XML report and depending on the plugins used sends the report input, to user friendly HTML dashboards in Jenkins interface. This dashboard allows users to see a quick overview of test results, pass and fail counts, time taken to execute, and historical trend graphs. The graphs are essential for teams to see how stable the API has been over time. This immediate notification helps to shorten the feedback loop considerably, making it possible for a developer to identify and address payload integrity problems in a matter of minutes of their introduction [3].

VII. Conclusion

The use of Postman collections integrated with Jenkins hooks enables automated framework to achieve the objective of preserving RESTful API payload integrity at Charter Communications. By inserting validations directly into the CI pipeline, it allows organizations to detect breaking changes sooner than later with greater confidence deploying the system and fewer post-release defects. Through an automated strategy to validate the API, the reliability of the entire microservices system is enhanced, while reducing the reliance on time consuming and error-prone manual validation effort. This framework can be further extended in the future as more possibilities may be included to the contract testing concepts: formalizing the contract testing concepts into more formal provider consumer checks, or being part of machine learning to search through our history of testing and find the flaky tests, or making observations when a test goes bad.

REFERENCES:

1. Afzal, W. and Piadehbasmenj, A., 2021, June. Cloud-based architectures for model-based simulation testing of embedded software. In *2021 10th Mediterranean Conference on Embedded Computing (MECO)* (pp. 1-8). IEEE.
2. Arcuri, A., 2020. Automated black-and white-box testing of restful api with evomaster. *IEEE Software*, *38*(3), pp.72-78.
3. Gong, S., Wang, C., Zhang, X., Liu, S., Xu, W., He, Z. and Wang, T., 2021, October. Analysis of Automatic Software Interface Test Framework based on Power Cloud. In *2021 IEEE 5th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)* (Vol. 5, pp. 101-105). IEEE.

4. Villalobos-Arias, L., Quesada-López, C., Martínez, A. and Jenkins, M., 2020, June. Assessing two graph-based algorithms in a model based testing platform for java applications. In *2020 15th Iberian Conference on Information Systems and Technologies (CISTI)* (pp. 1-6). IEEE.
5. Waseem, M., Liang, P., Shahin, M., Ahmad, A. and Nassab, A.R., 2021, June. On the nature of issues in five open source microservices systems: An empirical study. In *Proceedings of the 25th International Conference on Evaluation and Assessment in Software Engineering* (pp. 201-210).
6. Zeydan, E. and Mangues-Bafalluy, J., 2022. Recent advances in data engineering for networking. *Ieee Access*, *10*, pp.34449-34496.