International Journal on Science and Technology (IJSAT)
E-ISSN: 2229-7677 ● Website: www.ijsat.org ● Email: editor@ijsat.org

IMPROVING API TEST ACCURACY THROUGH SCHEMA VALIDATION AND REAL-TIME JSON ASSERTIONS IN JENKINS PIPELINES

Udayan Verma

Denver, USA
udayanverma7@gmail.com

Abstract:
The paper presents a framework, which can make API testing more accurate by integrating JSON schema validation and real-time JSON assertions as steps in the Jenkins pipeline. The structural regressions are known to bring about catastrophic integration mistakes and the conventional functional testing may not be effective within a large scale microservices ecosystem. The proposed model applies a two-level validation plan. A quality gate is a versioned JSON schema against which API responses are checked to be correct, in order to ensure integrity. The satisfied pay loads are then passed to a next stage of real-time JSON assertions which accept the data and compare it against dynamic business logic and situational rule sets. The combination of the two layers into the CI/CD process will guarantee the attention to both structural and functional correctness. The effect is that there will be a massive decrease in defect leakage, better issues will be resolved, and the confidence of production deployments will grow.

Keywords: Jenkins Pipeline, API Testing, Schema Validation, JSON Assertions, CI/CD.

I. Introduction

Verification of functional criteria is not an adequate endpoint integrity in a distributed and complex API ecosystem. New payloads should also comply with a strict structural contract to avoid integration failures. In a federated microservices architecture of an organisation such as Charter Communications with functional tests, a collapsing underlying payload schema may succeed with field additions or deletions. This type of schema drift may result in otherwise malicious but expensive downstream failures, especially when a single API endpoint is shared across many applications that use it. In order to reduce this risk, a two-level verification mechanism was introduced in the Jenkins CI/CD pipelines. The first level isSchema Validation where the JSON Schema definitions are employed to test the validity of the API response schema to contain the required fields and data types without making any calls to dedicated functional test cases. The Real-Time JSON Assertions layer follows and explores the content of a payload by comparing it to some data-driven and dynamically-managed rules to ensure the data meets some business logic and environmental condition.

II. Background and Rationale

The transition of Charter Communications to a microservices-based infrastructure was incredibly swift, resulting in a microcloud of mutually reinforcing APIs raising significant questions regarding the reliability of the data and the compatibility of the systems. The primary issue was the question of schema drift in which an API changes its response in a manner that does not mean its services that consume the API change accordingly. As an example, a production service may change the type of a field, such as a string to a numeric or vice versa, resulting in parsing errors in subsequent applications. The historical testing methods used at the organisation, which were mainly aimed at functional validation, were not

sufficient to identify these structural regressions. A test could confirm the correctness of one data point while failing to identify that another critical field was absent from the response payload entirely. These omissions were frequently discovered only after deployment, necessitating reactive hotfixes and eroding confidence in the release process. Consequently, a preventative, automated solution was deemed essential [1]. The rationale for the Jenkins-based framework was to establish the API contract as a non-negotiable quality gate, thereby shifting quality control earlier into the development lifecycle. By validating the schema first, the framework ensures the contract is honoured before proceeding to verify that the data contained within that contract is contextually correct.

III. System Architecture & Pipeline Design

The framework is architecturally founded upon a standardised, declarative Jenkins pipeline, engineered for reusability across all microservice projects to enforce a uniform standard of quality assurance. Central to its design is a 'fail-fast' philosophy, whereby a failure in an early validation stage immediately halts the pipeline, preventing the execution of subsequent, more resource-intensive processes. The pipeline is structured into a sequence of distinct stages. Following the initial Checkout and Build & Deploy stages, the process enters the first critical validation gate: Schema Validation. In this stage, a script programmatically invokes an API endpoint and validates the structure of its JSON response against a version-controlled JSON Schema file [2]. This check rigorously assesses the presence of required fields, correct data types, and adherence to predefined patterns. Any non-conformance results in immediate pipeline termination, with detailed error reporting. Only upon successful schema validation does the pipeline proceed to the Real-Time Assertions stage. Here, a functional test suite, employing libraries such as RestAssured, executes dynamic, context-aware assertions that scrutinise the data against specific business rules and expected values. The final Reporting stage aggregates the outcomes from both validation layers. This modular design, which separates structural validation from functional logic testing, is highly effective at isolating the root cause of failures, enabling development teams to rapidly distinguish between a broken API contract and flawed business logic [3].

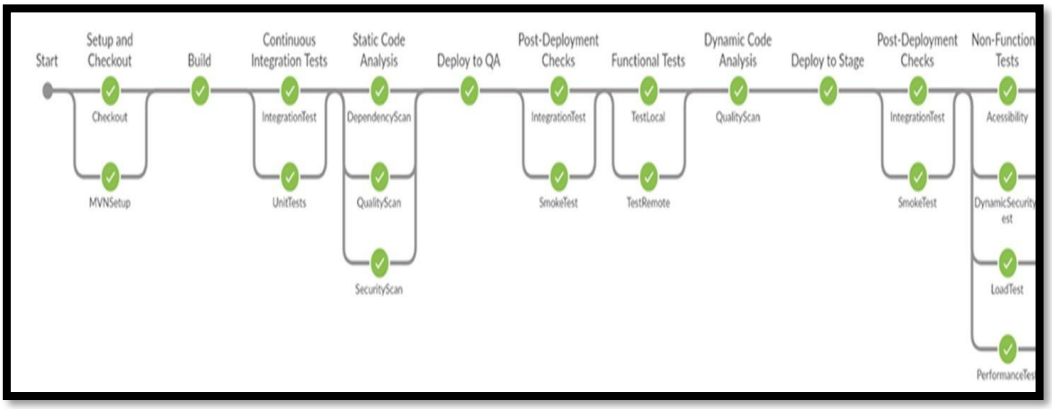

Fig 1: Importance of Pipeline Quality Gates

IV. Integration & Maintenance Strategy

A seamless integration approach and sustainable maintenance support the long-term effectiveness of the framework. This is done by considering API contracts and test logic as versioned artifacts. All JSON Schema files are consolidated within a centralised Git repository, which functions as the single source of truth for API contracts. Any proposed modification to a schema must be submitted via a pull request, a process that inherently facilitates peer review and ensures consuming teams are notified of impending changes. To streamline integration across a multitude of microservice projects, the core validation logic is encapsulated within a Jenkins Shared Library. This approach abstracts the implementation details, enabling development teams to incorporate the entire two-tier validation process into their project by

adding only a minimal configuration to their Jenkinsfile [4]. This centralisation of logic ensures that any enhancements or fixes to the validation framework are automatically propagated to all inheriting pipelines, thereby minimising redundant effort and enforcing a consistent quality standard across the organisation. The test suites containing the real-time assertions remain co-located with the microservice's source code, allowing them to evolve in tandem with the application's business logic [5].

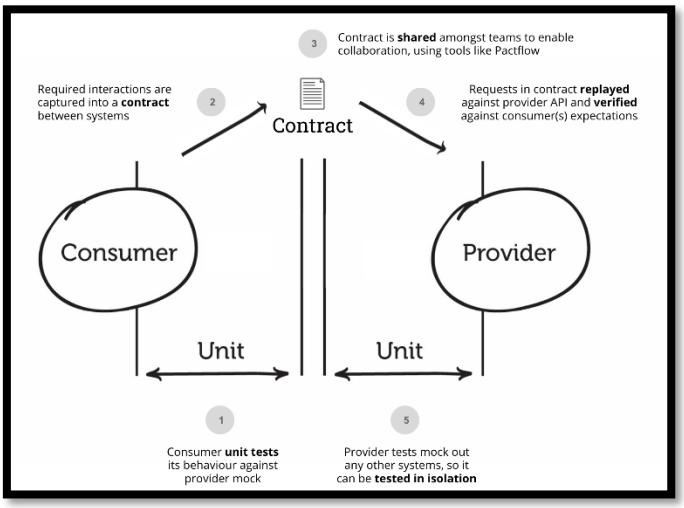

Fig 2: Contract Testing & Uses

V. Scalability & Optimization

To accommodate the expanding microservices landscape at Charter Communications without introducing bottlenecks, the framework was designed for both scalability and performance optimization. Scalability is addressed primarily through the parallel execution capabilities of Jenkins. Test stages for multiple API endpoints or even different services can be configured to run concurrently, which significantly curtails the total execution time required for comprehensive regression suites. The principal optimization, however, is derived from the pipeline's intrinsic 'fail-fast' architecture. By strategically positioning the lightweight schema validation stage before the more computationally expensive functional assertion tests, the system avoids wasting resources on payloads that are already structurally non-compliant. This sequencing provides rapid, early feedback and conserves compute cycles [6]. Furthermore, the utilisation of containerized Jenkins agents ensures the provision of ephemeral, consistent, and isolated test environments. This facilitates elastic scaling, allowing the infrastructure to dynamically provision resources in response to fluctuating demand during peak testing periods.

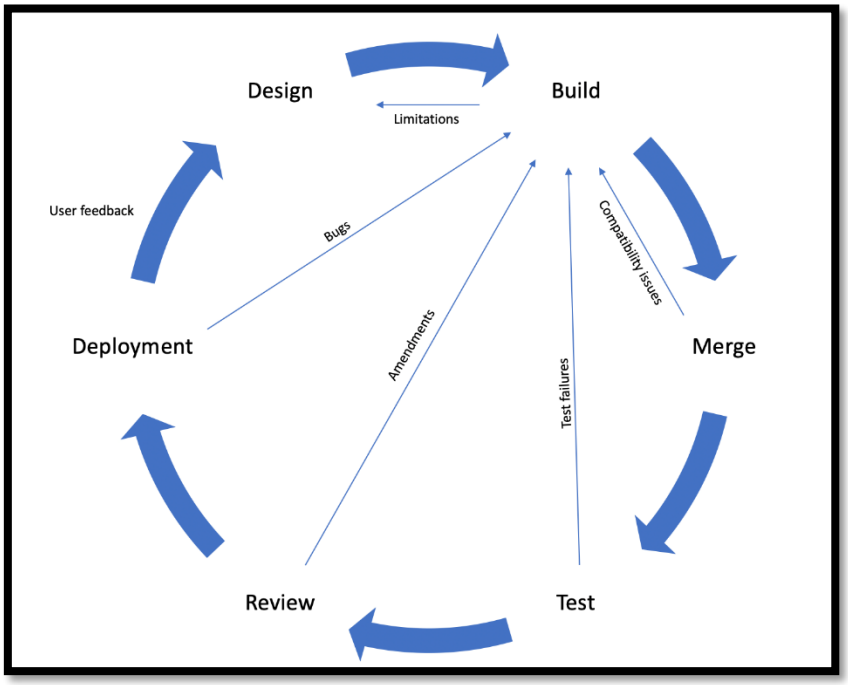

Fig 3: DevOps Feedback Loop

VI. Reporting & Monitoring

The framework relies heavily on actionable feedback, which is provided through reporting and monitoring systems aimed at providing clear, immediate, and context-rich information. When the pipeline is executed, Jenkins dashboards show a high-level view of the build statuses, showing the exact point at which the failure occurred, either Schema Validation or Real-Time Assertions. In the case of schema validation failure, the build logs are set to provide specific error messages that indicate the specific discrepancy, which may be a missing field or a different data type and thus removes ambiguity. In the case of assertion failures, common test reporting extensions create informative summaries describing the expected and actual outcomes. The pipeline can be linked to enterprise communication tools, such as Slack, to promptly send failure notifications to the appropriate team channel, including a direct attachment to the build logs.

VII. Conclusion

The implementation of JSON Schema validation and real-time assertions into a standardised Jenkins pipeline has significantly improved the quality and reliability of API testing within Charter Communications. By enforcing a two-layer model of validation, API responses undergo validation against their defined structural contracts prior to any functional logic being tested, thus preventing proactively a whole set of integration failures. Through a Contract-as-Code approach and pipeline libraries, the framework provides a scalable and sustainable solution that enhances the integrity of the microservices ecosystem of the organisation leading to more reliable and confident continuous delivery.

REFERENCES:

[1] Révész, Á., & Pataki, N. (2021). Visualisation of Jenkins Pipelines. Acta Cybernetica, 25(4), 877–895. https://doi.org/10.14232/actacyb.284211

[2] Ehsan, A., Abuhaliqa, M. A. M. E., Catal, C., & Mishra, D. (2022). RESTful API Testing Methodologies: Rationale, Challenges, and Solution Directions. Applied Sciences, 12(9), 4369. https://doi.org/10.3390/app12094369.

[3] Georgiev, A., Valkanov, V., & Georgiev, P. (2024). A comparative analysis of Jenkins as a data pipeline tool in relation to dedicated data pipeline frameworks. In Proceedings of the 2024 International Conference on Automatics and Informatics (ICAI), pp. 508–512. Available: https://www.semanticscholar.org/paper/da32933bf3f9d0015b594749837ce89619a9f6f5.

[4] H. da Gião, A. Flores, R. Pereira, and J. Cunha, "Chronicles of CI/CD: A Deep Dive into its Usage Over Time," *arXiv.org*, Feb. 27, 2024. https://arxiv.org/abs/2402.17588

[5] Attouche, L., Baazizi, M.-A., Colazzo, D., Ghelli, G., Sartiani, C., & Scherzinger, S. (2022). Witness Generation for JSON Schema. PVLDB, 15(13), 4002–4014. https://www.vldb.org/pvldb/vol15/p4002-sartiani.pd.

[6] Attouche, L., Baazizi, M.-A., Colazzo, D., Ghelli, G., Sartiani, C., & Scherzinger, S. (2022). Witness Generation for JSON Schema. PVLDB, 15(13), 4002–4014. https://www.vldb.org/pvldb/vol15/p4002-sartiani.pd