International Journal of Science and Research (IJSR)
ISSN: 2319-7064
SJIF (2022): 7.942

Resource Validation Framework: Ensuring State and Lifecycle Consistency

Mahidhar Mullapudi

Abstract: ***In the intricate landscape of commerce, maintaining accurate and synchronized information about resources which are granular units in commerce like meters, Skus etc., is a multifaceted challenge. This paper introduces a robust solution - the Resource Validation Framework. At its core, this framework integrates critical components, leveraging a Commerce application that publishes configured resource data*** **[1]**. ***This data, vital for external partner systems, undergoes a meticulous journey—initially stored in a normalized format that business planners can configure and then transformed into an intermediate model with all possible combinations and permutations and converted into resources that are granular to be able to transact which are published and stored in Cosmos DB and subsequently archived in a Git repository through a seamless pull request mechanism. The resulting Cosmos DB and Git repository serve as authoritative sources for resource state and lifecycle*** **[2] [3] [4] [5].** ***The essence of the validation framework lies in guaranteeing the validity of this data, focusing particularly on the state of each resource—whether it's in a test, preview, GA, public, or decommissioned phase. The lifecycle of resources, encapsulating the start and end dates of those state changes, becomes paramount for operational transparency. To achieve this, a systematic approach is implemented, requiring service teams responsible for products and meters to validate the latest resource information against their local datasets. Subsequently, these teams attest to the accuracy of the data, signaling that the resources under their ownership are in a valid state*** **[6].** ***For streamlined management, the framework introduces a state - tracking mechanism. Resources with discrepancies are designated as 'pending, ' accompanied by a link to the respective incident for tracking. Meanwhile, unattested resources are systematically tracked, fostering a structured validation process. This monthly validation cycle aligns seamlessly with the regular billing and auditing processes of the company, establishing a proactive and integrated approach to resource management.***

Keywords: Modern Ingestion, Resource Validation, Lifecycle State Management, Large Scale Distributed Applications, Data Consistency, Attestation process.

1. Introduction

The modern paradigm of large - scale commerce applications demands a meticulous orchestration of resource data to ensure operational fluidity and financial integrity. At the heart of this challenge is the Resource Validation Framework, a comprehensive solution tailored to address the complexities of resource state and lifecycle management across diverse ecosystems [1].

A pivotal aspect of this framework is the Commerce application, acting as the conduit for publishing configured resource data. This data, forming an essential connection for external partner systems, goes through a complex process. It finds its initial repository in Cosmos DB [7], ensuring real - time access to the latest resource information. Subsequently, this data is archived in a Git repository through a seamless pull request mechanism, providing external partners with a dependable and up - to - date resource reference [8].

The essence of the validation framework lies in its dedication to guaranteeing the accuracy of this data, with a particular emphasis on the resource states and lifecycles. States, ranging from test to decommissioned, demand meticulous validation, while the lifecycle, encapsulating the chronological sequence of state changes, adds a layer of operational transparency [9] [10].

To operationalize this framework, service teams responsible for products and meters are entrusted with the validation process. Their task is to ensure that the latest resource information aligns with their local datasets, attesting to the validity of the data under their ownership. To streamline this validation, the framework introduces a state - tracking mechanism, categorizing resources with discrepancies as 'pending, ' each linked to the respective incident. Resources that are not verified are consistently monitored, creating a formal process of confirmation and accountability.

This monthly validation cycle, a cornerstone of the framework, aligns seamlessly with the regular billing and auditing processes of the company. It establishes a proactive and integrated approach to resource management, reinforcing a culture of accountability and precision in the ever - evolving landscape of commerce [11] [12].

2. Systems Overview

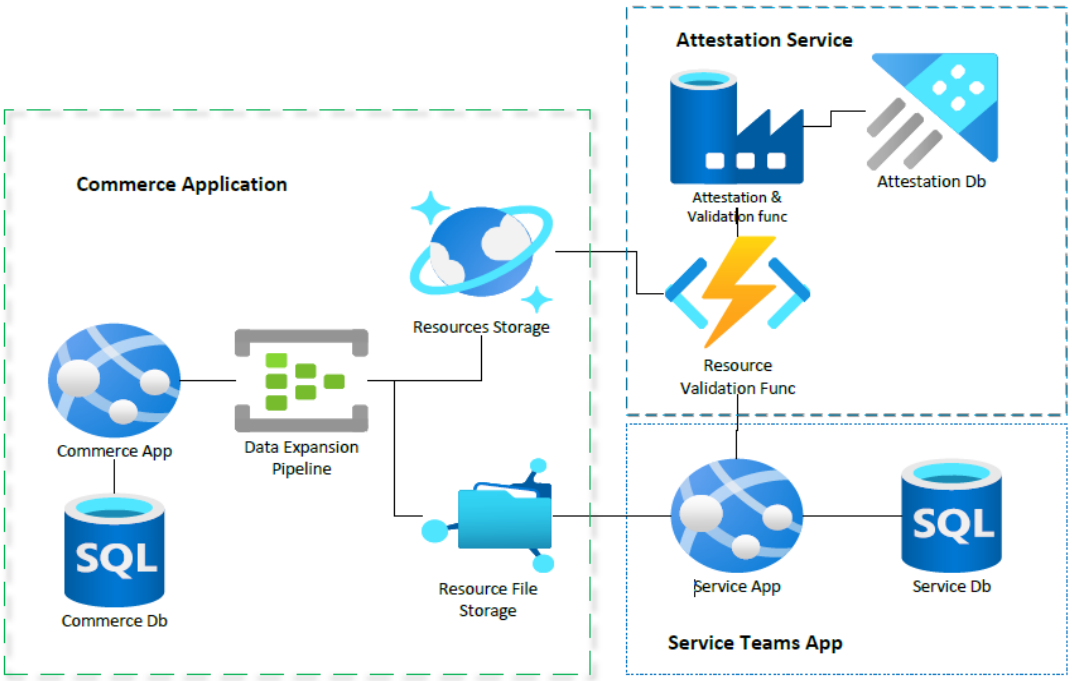

Figure 1: Resource Validation Framework Overview

As illustrated in Figure 1, the resource validation framework consists of the following major components:

Commerce Application
Commerce Application maintains the data for products and services that are configured by Business planners talking to different service owners and service providers. So, the data being configured or collected is in a denormalized format for simplicity and is stored that way in this application. Once the changes are complete, tasks picked up based on events and perform an operation called expansion where this normalized data is transformed into an intermediate model using all the combinations of latest data components and policies configured for the type of resource [13] [2].

This intermediate model data is transformed into granular units that can be transactable which we call *resources*. Resource has a state and lifecycle associated with it. The pipeline contains instructions on how to transform and store this resource information to multiple data streams – cosmos db and git repositories. Cosmos db offers better querying and performance capabilities, whereas the git repo offers easy access and visibility to the resource data that can be easily exposed to other teams for validation and other purposes [14].

Service Teams Data Validation
Service Teams in this context are the owners of those specific resources. The service teams would have the correct information and expectation about the state and lifecycle of the resources stored in their local database. They use the latest information that the Commerce application publishes and validates that information with their local database and performs a process which we call *Attestation*. Here the core of this framework is that the service owner performs these validations to ensure that the state of the resource and the lifecycle properties match their expectations [1] [6].

Permission Validation and Attestation Process
This Attestation function takes care of ensuring that the service owners have the right permissions to provide attestation/confirmation on the resources that they have validated and stores that attestation information. This function also ensures that the attestation information is stored and serves reporting and creating action item information. This attestation function connects to cosmos db to ensure the data that is being attested for those meters is latest and identify any unattested resources and aggregate or group them by product/service/service family [8] [15].

3. DEEP DIVE – ATTESTATION FUNC

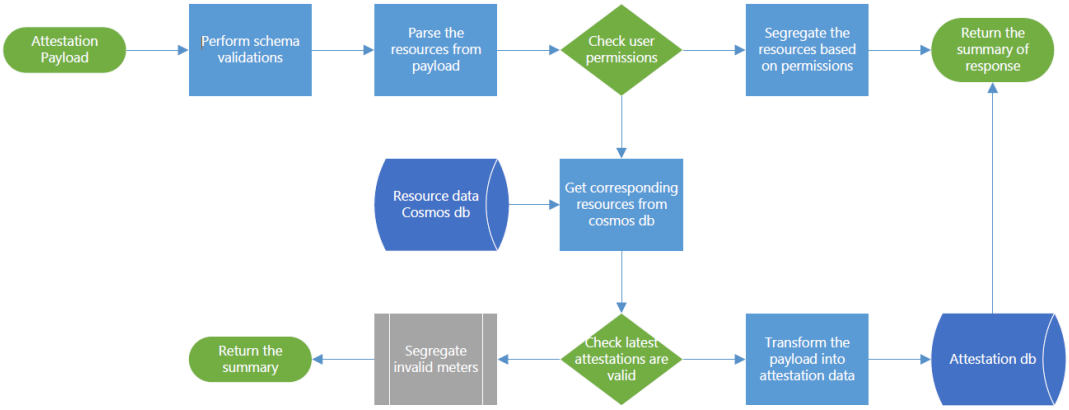

Figure 2: Attestation process - Deep Dive

In this comprehensive exploration, we intricately dissect the attestation validation process, shedding light on advanced design and implementation strategies that bolster efficiency and reliability, incorporating technical details for a thorough understanding.

1) Resource Validation by Service Owners:
Service owners engage in a meticulous validation process, leveraging advanced querying techniques to compare their local datasets against configurations published by the commerce system. This involves utilizing industry - standard data querying languages and techniques, such as SQL or NoSQL queries, to ensure a comprehensive alignment of resource states and lifecycles [5] [15].

2) Payload Creation:
Upon successful validation, service providers craft a payload containing resource information. This payload adheres to a well - defined schema, ensuring structured and standardized data representation. JSON formats may be employed, with each resource detail meticulously organized within the payload.

3) Attestation Function Overview:
The attestation function, a critical component, commences by validating authentication information using robust cryptographic methods. This ensures the integrity and security of user credentials during the attestation process [9].

4) Schema and Authorization Checks:
Following authentication, the attestation function rigorously checks the payload schema against predefined guidelines, employing JSON or XML schema validation techniques. Simultaneously, it verifies user permissions through OAuth or other industry - standard authorization protocols, ensuring only authorized personnel can provide attestations.

5) Payload Processing and Version Control:
The attestation function then processes the payload, converting it into a collection of resources. For version control, service - to - service calls are initiated, leveraging secure communication protocols such as HTTPS. Cosmos DB, acting as the source of truth, is queried for the latest resource information, ensuring that the attestation function works with the most up - to - date data.

6) Validation of Resource Versioning:
Each resource undergoes meticulous examination, with the attestation function ensuring that the version property aligns with the latest data. Techniques such as conditional requests or ETags may be employed for efficient version checking, minimizing unnecessary data transfer.

7) Aggregation and Result Summary:
Results, categorized as valid or stale based on version status, are aggregated by the attestation function. The summary, enriched with technical details like cryptographic hash functions for data integrity, is prepared for further processing.

8) Ingestion to Data Factory:
The summarized results are seamlessly ingested into the data factory, orchestrated through efficient data processing frameworks such as Apache Kafka or Apache Flink. This step ensures the secure and reliable storage of attestation results in storage.

9) Reporting and Autonomous Validations:
The database, now enriched with attestation results, has become a dynamic resource for reporting and validation systems. RESTful APIs or other communication protocols facilitate seamless integration, allowing these systems to generate actionable insights and create periodic action items for service teams [16] [17] [18].
Below is a sample algorithm that is language agnostic that outlines the list of steps that are performed in this validation framework:

```
function attestationValidationProcess (serviceOwnerData, commerceSystemConfig) {

// Step 1: Resource Validation by Service Owners
 validateResources (serviceOwnerData, commerceSystemConfig);

// Step 2: Payload Creation
payload = createPayload (serviceOwnerData);

// Step 3: Attestation Function
authenticationResult = validateAuthentication ();

if (authenticationResult. success) {
// Step 4: Schema and Authorization Checks
```

```
schemaCheckResult = validatePayloadSchema (payload);
authorizationResult = validateAuthorization ();

if (schemaCheckResult. success && authorizationResult. success) {
// Step 5: Payload Processing and Version Control
resourceCollection = processPayload (payload);
latestResourceInfo = queryCosmosDBForLatestInfo (resourceCollection);

// Step 6: Validation of Resource Versioning
validationResults = validateResourceVersioning (resourceCollection, latestResourceInfo);

// Step 7: Aggregation and Result Summary
summary = aggregateResults (validationResults)

// Step 8: Ingestion to DataFactory
ingestToDataFactory (summary);

// Step 9: Reporting and Validation Systems Integration
integrateWithReportingAndValidationSystems ();
} else {
// Handle authorization or schema check failures
handleValidationFailure (schemaCheckResult, authorizationResult);
 }
} else {
// Handle authentication failure
handleAuthenticationFailure (authenticationResult);
 }
}
```

Below are samples of some of the helper functions that would be useful in this validation process:

```
// Helper functions

function validateResources (serviceOwnerData, commerceSystemConfig) {
 // Implement resource validation logic
}

function createPayload (serviceOwnerData) {
 // Implement payload creation logic
}

function validateAuthentication () {
 // Implement authentication validation logic
}

function validatePayloadSchema (payload) {
 // Implement payload schema validation logic
}

function validateAuthorization () {
 // Implement authorization validation logic
}

function processPayload (payload) {
 // Implement payload processing logic
}

function queryCosmosDBForLatestInfo (resourceCollection) {
 // Implement querying Cosmos DB for the latest resource information
}

function validateResourceVersioning (resourceCollection, latestResourceInfo) {
 // Implement resource versioning validation logic
}

function aggregateResults (validationResults) {
 // Implement result aggregation logic
}

function ingestToDataFactory (summary) {
 // Implement ingestion to Data Factory logic
}

function integrateWithReportingAndValidationSystems () {
 // Implement integration with reporting and validation systems
}

function handleValidationFailure (schemaCheckResult, authorizationResult) {
 // Implement logic to handle validation failures
}

function handleAuthenticationFailure (authenticationResult) {
 // Implement logic to handle authentication failures
}
```

This information can be used by reporting and validation systems to create action items for the service teams to perform the attestation on a periodic basis [19].

4. Conclusion

The Resource Validation Framework, detailed in this paper, represents a significant advancement in ensuring the consistency of resource configurations across diverse ecosystems. By combining meticulous resource validation by service owners, a well - defined attestation process, and seamless integration with reporting and validation systems, this framework establishes a robust foundation for proactive resource management and precision in the dynamic landscape of modern commerce.

The attestation validation process, as explored in - depth, highlights the intricacies of authentication, schema checks, version control, and result aggregation. Leveraging secure

communication protocols, industry - standard authorization mechanisms, and efficient data processing frameworks, this process addresses the challenges posed by resource validation in a comprehensive and technically sound manner.

5. Future Directions

Looking ahead, future enhancements to the Resource Validation Framework could involve the integration of machine learning algorithms to augment the validation process. By analyzing usage metrics and patterns, machine learning models can be trained to identify anomalies and flag potential invalid states or configurations. This intelligent layer can significantly enhance the attestation validation process by automating the detection of discrepancies, reducing manual efforts, and providing real - time insights into the health of the resource ecosystem.

Moreover, the incorporation of machine learning can contribute to the continuous improvement of the validation process. Models trained on historical data can adapt to evolving patterns, offering a proactive approach to identifying emerging issues before they impact the integrity of resource configurations. In addition to anomaly detection, exploring techniques such as natural language processing (NLP) could further enhance the schema checks during the attestation process. NLP algorithms can parse and understand textual descriptions of resource configurations, ensuring a more nuanced validation approach that goes beyond the structural constraints of traditional schema checks.

As the landscape of commerce evolves, embracing machine learning - driven enhancements to the Resource Validation Framework positions organizations to stay ahead of the curve. These future developments not only streamline the attestation validation process but also empower businesses to proactively address challenges, ensuring the ongoing integrity of resource configurations in an increasingly complex and dynamic environment.

References

[1] B. P. D. G. Yogesh L. Simmhan, "A Survey of Data Provenance Techniques".
[2] Kleppmann, Martin, Designing Data - Intensive Applications, O'Reilly Media, 2017.
[3] "Azure Data Factory, " [Online]. Available: https: //azure. microsoft. com/en - us/products/data - factory.
[4] "Azure Data Explorer, " [Online]. Available: https: //learn. microsoft. com/en - us/azure/data - explorer/.
[5] "What is Azure Data Factory?, " [Online]. Available: https: //learn. microsoft. com/en - us/azure/data - factory/introduction.
[6] J. L. M. H. D. D. M. F. T. R. V. Kalavri, "Three steps is all you need: fast, accurate, automatic scaling decisions for distributed streaming dataflows., " *OSDI*, 2018.
[7] "Azure Cosmos Db, " [Online]. Available: https: //learn. microsoft. com/en - us/azure/cosmos - db/introduction.
[8] "Cassandra, " [Online]. Available: https: //cassandra. apache. org/.
[9] "Azure Functions, " [Online]. Available: https: //learn. microsoft. com/en - us/azure/azure - functions/.
[10] "Azure Event Hubs, " [Online]. Available: https: //learn. microsoft. com/en - us/azure/event - hubs/event - hubs - about.
[11] "Apache Kafka, " [Online]. Available: https: //kafka. apache. org/.
[12] "Apache Spark, " [Online]. Available: https: //spark. apache. org/.
[13] K. H. Robert Martin, Clean Architecture: A Craftsman's Guide to Software Structure and Design, Pearson.
[14] "Low latency system design, " [Online]. Available: https: //kayzen. io/blog/large - scale - low - latency - system - design.
[15] "microservices - best - practices, " [Online]. Available: https: //www.mulesoft. com/sem/lp/whitepaper/api/microservices - best - practices.
[16] K. Beck, Extreme Programming Explained: Embrace Change, Addison - Wesley, 2000.
[17] "12 Factor App, " [Online]. Available: https: //12factor. net/.
[18] "Owasp, " [Online]. Available: https: //owasp. org/.
[19] "Semantic Matching Wiki, " [Online]. Available: https: //en. wikipedia. org/wiki/Semantic_matching.