

API Logs for Product Metrics - ADT

- Summary
- Context
 - High Availability and disaster recovery
 - Performance
 - Service Level Objectives (SLOs)
 - Disaster Recovery (DR) Objectives
 - Applicable regulations
 - Principles
 - Constraints
 - Design
 - High Level Architecture
 - Presto Table : api_log (Run MSCK REPAIR TABLE every day for updating partitions).
- Analysis
 - Maintainability
 - Observability
 - Testability
- Discarded Alternatives
- Appendix
 - Definitions/Acronyms
 - References
 - Contributors

Status	DRAFT
APM ID	
SSP ID	
PRD link	
Reviewed by	
Approved by	

Instructions

1. Update the title of the page to include the name of the system described and an appropriate qualifier like "Architecture", "Design", "Blueprint", or "Pattern".
2. Review the outline and instructions. Note that you are expected to expand and adjust the outline as needed, especially in the [Design](#) section.
3. Populate the template sections in whatever order you see fit, the process is expected to be iterative.
4. Fill in and update the document status and other metadata above as the document matures.

Status	Definition
DRAFT	Document may be incomplete, incorrect, or misleading, and is subject to change
IN PROGRESS	Document substantially complete and in process of review
REVIEWED	Reviewers agree that the document is complete and accurate
APPROVED	Stakeholders agree to move forward with the direction described in the document
STALE	Document is abandoned or obsolete
WORKING COPY	Document is unofficial, and should be ignored for tracking purposes.

5. Delete the "Instructions" boxes as you complete the document.

Solicit feedback early and often, on both content and form. Start with your own team and then expand upward and outward.

Writing

Pay attention to the quality of your prose. Write the document so that it is easy to read and flows like a story. Follow [best practices for technical writing](#). Confluence was chosen as the place to host these documents in part because it naturally supports consistency in format, avoiding many of the pitfalls and distractions of more free-form document editors. Please take advantage of that and avoid introducing superfluous customization or other unnecessary changes in format. That said, don't let any of this guidance keep you from effectively communicating with your audience.

Drawing

Diagrams are often the best way to illustrate relationships among entities, but they are best combined with text that:

- Provides context via a formal introduction, a segue in the preceding paragraph, or an adequately descriptive section heading
- Explains the visual vocabulary, ideally in a legend
- Expands abbreviations used, especially if not already introduced earlier in the document
- Adds nuance that would otherwise introduce unnecessary complication to the diagram

Focus each diagram on a single level of abstraction and a specific set of points to be made. Avoid the temptation to include all facets of the architecture in a single diagram. Use common entity names and consistent formatting across multiple diagrams to help readers link the various aspects together within their own mental model.

Remember that while a picture can be worth a thousand words, a diagram with extraneous elements or inconsistencies can be confusing or misleading – like text with random words sprinkled throughout. Make sure that every element and variation is meaningful, intentional, and required.

Focus

Similarly, consider breaking up a complex architecture into multiple subsystems or specializations, each with their own architecture document and development lifecycle. A simpler and higher-level architecture document can then provide a unifying model that pulls them all together. This higher-level document can also provide common language and concepts that in turn allow the subordinate documents to be simpler and more focused without losing precision or context.

Further Study

While this template is intended to be usable without training in specific architecture methodologies, please consider further study of the art of [software architecture](#). In addition to the more specific references above and in the template itself, the design of this template was particularly influenced by:

- Simon Brown's [software guidebook](#) concept and [C4 model](#)
- Phil Koopman's [Embedded Software Engineering](#) course, especially his lectures on [Software Development Processes \(video\)](#) and [Software Architecture and High-level Design \(video\)](#)
- [Architecture Decision Records](#)
- The Thor [Development Design Template](#)
- The GTP App Data "-ilities" - [Template](#), [Pattern-Template](#), [Architecture Scope Document Template](#), and [_DF Tech Profile - Template](#)
- [Documenting Architecture Patterns](#)
- The Heap [Problem Brief](#) and [Design Brief](#)
- The [FURPS](#) requirements framework
- The [TOGAF](#) architecture framework

I have a deadline to meet, why do I need to spend time filling this out?

You needn't bother documenting your architecture if all of the below are true:

1. You are the sole stakeholder and funding source for the system
2. You will be building, and operating the system by yourself
3. You have plenty of time to rebuild the system a few times as you discover the limitations of earlier approaches
4. You have a perfect memory, so you won't forget requirements discovered and lessons learned

Otherwise, you need architecture documentation in order to:

1. Align stakeholders and sponsors around value proposition, requirements, and expectations
2. Align developers and operators so that their work converges into a working and reliable system
3. Minimize the risk that fundamental aspects of the system will need to be changed later
4. Prevent architectural regressions as this system evolves and inspires others that follow

If you already have documentation that appears to meet all these needs, then feel free to link to it from the relevant sections (or excerpt it, if on Confluence). Make sure that readers of this template can count on being able to access linked documents now and into the future.

My team uses ADRs, can't I just refer to those?

[Architecture Decision Records](#) are an excellent way of documenting the decisions that go into creating and iterating on architecture, but they form a log of the journey the team took to get to an architecture, rather than a concise and easy to understand description of the architecture as a whole. This is fine for reviewers of individual decisions, but not great for reviewers of the overall design.

ADRs and Architecture Design documents complement each other, and each can help simplify and focus the other. For instance, an Architecture Design document can use references to ADRs to simplify its explanation, maintaining focus on how the aspects covered by various ADRs come together to achieve the overall business goals. Similarly, an ADR can link to a specific version of an Architecture Design document to provide background information that might otherwise need to be repeated across multiple ADRs.

Why should I use this template in particular?

This template is intended as a guide to develop a document that meets the goals described above. It primarily offers two types of guidance:

1. **Effective communication:** structure and best practices known to help convey architectural information in a manner that is both unambiguous and understandable
2. **Design checklist:** a set of perspectives, categories of requirements, and aspects of implementation specific to software architecture at Walmart that anticipate the questions and expectations of typical stakeholders

This template is also specifically designed to expedite Walmart governance processes, and is an integral part of the [Every Day Engineering Excellence \(EDEE\)](#) program.

Wow, the template is long. Do I really have to fill everything out?

While it is expected that most of the guidance provided by this template will be relevant to most architectures, it is up to the authors, reviewers, and consumers of the document to decide what topics are relevant and necessary to cover and how the material is best presented. The only measure of success is how well the document helps stakeholders operate as a team to deliver the intended business value.

A good rule of thumb is to focus on the big rocks. Describe the aspects of the system that are fundamental to its definitions or will be hard to change once it is built. Omit detail about things that will either sort themselves out or are easily changed later.

Example

Imagine an application that performs a specific business function that involves some business logic. It's important to clearly describe the business function itself, including who or what it serves and where it must provide that service to them. It's also important to call out what language(s) and framework(s) will be used to implement that logic, as it is the codification of that logic that represents the primary investment being made and asset being delivered.

Assuming an iterative development process with frequent feedback from stakeholders, is *not* important to detail the logic itself in this document. The development process will ensure that the logic ends up being complete and correct, and that rework is minimized. A good architecture anticipates the range of detailed business and technical requirements that will be discovered through the development process, accelerating it by ensuring the necessary technical capabilities and qualities are in place. This minimizes the technical complexity of the logic and allows the project's developers to focus on solving business rather than technical problems.

Likewise, future changes to the logic will go through that same development process. The mark of a great architecture is that it anticipates those changes, even some changes to the business function itself, allowing them to be accommodated in a straightforward fashion without having to move or change those big rocks.

If the logic is particularly complex, it may be appropriate to create a separate design document as an early artifact in that process. That design document should refer to the corresponding architecture document for context, and it's potentially useful for the architecture document to also (eventually) refer to the design document. Such a design document describes another layer of "medium-sized rocks" that are useful to think through and align on early in order to parallelize development and minimize later rework.

This template assumes that I'm describing a concrete solution, but I need to document a pattern or platform -- what should I do?

Use the same template but specify abstract requirements instead of concrete ones (e.g., a range of supported SLOs instead of a specific number). As with any use of the template, if a field or section does not apply, mark it accordingly or simply delete it. If a new section is needed, add it.

Summary

[Back to Top...](#)

Why: Given the value Near Real Time endpoints provide, and their adoption within Walmart store operations, it is important to have a real-time dashboard to support product metrics. Product metrics tell us if the feature is creating value. Product metrics helps us with greenlight future investment in this feature.

What: We want to capture all of our HTTP requests with request and response body and persist them in storage so we can build reports on top of it to see what APIs suppliers are using most.

How: We want to publish the request body and response body from application to log publish service that write the logs to storage. We can plug this in Power BI to see the reports.

Business Use cases:

- Lack of product metrics make it difficult to understand if feature should be expanded in relation to others and causes extra work for product and engineering teams to pull data for executives.
- Able to see what suppliers and consumer IDs are using the endpoints and proactively reach out to suppliers if they are not consuming.
- Able to see what stores, GTINs are used mostly by suppliers.
- Patterns on IAC and DSC data.



Instructions

Aim for three short paragraphs that briefly summarize, in turn:

1. **Why:** describe the key business problem(s) addressed by this architecture, including goals relative to previous or alternative approaches
2. **What:** the key facet(s) of the architecture that enable it to address the business problem(s)
3. **How:** connect the dots between the key facet(s) of the architecture and the business value it will deliver

It's usually best to work through at least the [Context](#), [Non-functional Requirements](#), and [Design](#) sections below before attempting to distill the result here.

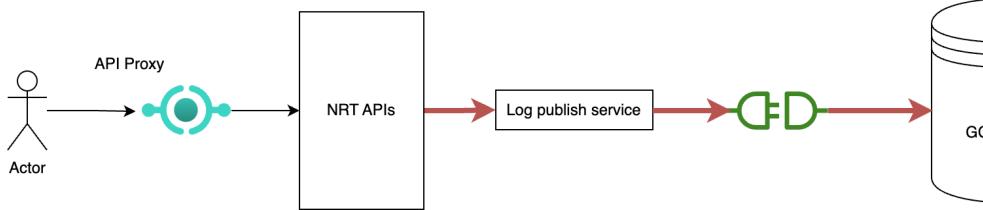
Context

[Back to Top...](#)

Suppliers use multiple endpoints depending on their usecase.

- PFS suppliers use IAC endpoint.
- DSD suppliers that deliver directly to store by trailer use DSC endpoints.
- Other suppliers look at the inventory in store, in transit and in DC.

- we also need to store request body to get some of the metrics because some key fields are inside the body like store, gtin etc.,



DSD Suppliers - DSD (Direct Store Delivery) suppliers are companies that deliver products directly to retail stores.

PFS Suppliers - PFS(Pay for Scan) Suppliers stock directly in sales floor and get paid on item scans.

Instructions

Elaborate on the *Why* described in the [Summary](#) by illustrating (ideally with a [diagram](#)) the relationships among the system and:

1. Users and other stakeholders served by the system
2. External components with which the system must interact
3. Environment(s) in which the system must operate

Help the reader build a mental model of the business functions and constraints that the architecture must satisfy. This is often best done using a simple diagram. Refer to entities that act as landmarks for the reader, typically ones that most readers will already recognize, or that are at least well-described elsewhere. Provide links to relevant definitions if possible, ideally to their own architecture design documents.

Keep this at a high level and focused on the describing the problem space – the next section will provide further detail and later sections will connect it to the solution.

It can be helpful to include high-level descriptions of lessons learned from previous approaches, but consider using the [Discarded Alternatives](#) section to document them in more detail.

For further discussion, see the [Context section](#) of the [Software guidebook](#) and the [C4 System Context diagram](#).

Non-functional Requirements

[Back to Top...](#)

Non-Functional	Standard
Testing	Testing Best Practices
Security	Global Compliance Program Top 12 Controls

Living Design (User Experience)	Living Design System
Accessibility	Web Content Accessibility Guidelines (WCAG) Overview

High Availability and disaster recovery

Business Application Tier
Tier 1

Metric	Trigger	Operations Context	Component	Response and Response Measure
Recovery Time Objective (RTO)	Health Check	Regional Failure	APIs	resume operation in alternate region within 24 hours
Recovery Point Objective (RPO)	Health Check	Regional Failure	APIs	lose no more than 30 minutes of data
Percent Availability	Health	Nominal	APIs	shall be available for use 99.9% of the time
Scheduled Downtime	Maintenance	Maintenance	APIs	Not expected to have downtime.
Availability Time Window	Health	Nominal	APIs	24*7 availability

SCALABILITY

US market -				
Metric	Operations Context	Component	Response and Response Measure	Comments
Requests per Second per consumer	Nominal	API	500 TPM	
Requests per Second per consumer	Peak	API	3000 TPM	
Concurrent Users	Nominal	API	3	
Concurrent Users	Peak	API	10	

Performance

Non-Functional	Standard
Availability	99.9%
Latency	< 200ms
TTL for Data	2 years
Volume	1billion/year

Service Level Objectives (SLOs)

[Back to Top...](#)

Type	Service Level Indicator (SLI)	Objective
Availability	Valid (non-4xx) requests successfully served (2xx or 3xx) within 100 milliseconds as measured by its local service mesh proxy	>= 99.9%
Latency in log publish service	95th percentile response time for successful requests (2xx or 3xx) as measured by its local service mesh proxy	<=200ms

Error Rate	SLO goal for 4xx against 2xx in % SLO goal for 5xx against 2xx in %	<=1% < 0.5%
-------------------	--	----------------

Instructions

Declare the relevant service level indicators (SLIs) and corresponding **service level objectives (SLOs)** for this system. Add or remove rows as needed, but define at least an availability SLO. A service's availability SLO must accommodate scheduled and unscheduled downtime, including downtime due to disasters.

See [SRE fundamentals: SLIs, SLAs and SLOs](#) and [Available . . . or not? That is the question](#) for a short primer on SLOs and SLIs and how to define them. Definition of service level agreements, if applicable, may be included or referenced here but are less architecturally relevant. At a minimum, every service should provide an availability SLO defined relative to an SLI that is meaningful to users of the service.

Note that definition of an SLI must include how it will be measured, and that the mechanism specified becomes a constraint for the design and requirement for the implementation.

Disaster Recovery (DR) Objectives

[Back to Top...](#)

Recovery Point Objective (RPO)	30 minutes
Recovery Time Objective (RTO)	Recovery not to exceed 24 hrs



Instructions

Define the maximum amount of data that may be lost due to any possible disruption, known as the system's **Recovery Point Objective (RPO)**, and how quickly that service must be fully restored with any recoverable data, known as the system's **Recovery Time Objective (RTO)**.

An RPO is defined as the maximum period of time between the recovery point and the disruption event. Data generated during that period (e.g., transactions) may be lost. An RTO is the period of time between the disruption event and the point at which the data up to the recovery point has been fully recovered and processed by the system.

It is possible for a system to support an availability SLO that allows for less downtime than its RTO might imply, as it might be able to generate and process new data during the recovery period in parallel with the recovery process.

Applicable regulations

[Back to Top...](#)

NO	Payment Card Industry Data Security Standard (PCI DSS)
NO	Sarbanes–Oxley Act (SOX)
NO	Health Insurance Portability and Accountability Act (HIPAA) / Health Information Technology for Economic and Clinical Health Act (HITECH Act)
NO	General Data Protection Regulation (GDPR)
NO	California Consumer Privacy Act (CCPA) / California Privacy Rights Act (CRPA)
NO	Video Privacy Protection Act (VPPA)
NO	Children's Online Privacy Protection Act (COPPA)



Instructions

Identify any industry or governmental standards, laws, and other regulations that apply to this system.

If the connection is non-obvious, provide a brief summary of why the system is in scope or expected to be in scope for the identified regulation(s). Likewise, please explain if this system has been specifically determined to be out of scope for a regulation that might otherwise seem to apply.

This section should call out applicable regulations even if the design (described later in the document) delegates implementation of compliance to underlying platforms or external services. For instance if the document describes a system that handles transactions involving payment cards, it should still list PCI DSS as an applicable regulation even if the design allows it to avoid PCI scope by handling only opaque tokens provided by a separate service that handles the actual payment card information itself.

Principles

[Back to Top...](#)

[Guiding Principles & Best Practices](#)

[Architecture Decision Records](#)

Key principles for the design

1. **Scalability** : The system should be able to handle an increasing number of suppliers and endpoints without compromising performance.
2. **Reusability** : The same service can be used across multiple rest api to track the request body, response body, status etc.,
3. **Interoperability**: The system should be able to work with other systems and be compatible with different platforms and technologies.
4. **Avoid event loss** : Using Kafka/GCS to ensure no event is lost when it is send by suppliers.
5. **Performance**: The system should be designed to optimize performance and minimize latency, ensuring efficient communication between distributed components.



Instructions

What general rules should be followed in the design?

List any guidance that the design should follow. These might include:

- Build or buy preference
- Preferences related to strategic goals
- Relative prioritizations of speed, quality, and cost
- Specific design methodologies
- Use or avoidance of specific sets of technologies

The [Principles](#) section of the [software guidebook](#) has a longer list. Focus on principles that guide the levels of abstraction addressed by this document.

It is often appropriate to simply link to a separate document that describes the principles to which the team or department developing the system has decided to adhere. This helps to ensure consistency across projects, which is the main purpose of such principles in the first place.

Constraints

[Back to Top...](#)

Walmart WCNP has global fluentd config in wcnp and cannot tweak log appenders.

API Proxy Gateway is not logging the request body and response body.

Masking is one of the constraint. we do not see the need for masking in KYS, but KYC may have PII data, we will build masking logic in log publish service when we see the need. This service does not need Tokenization kind of masking as it is logs, and we can do randomization by maintaining configuration about PII data or on the fly masking by taking masking paths as request param. Source system has to take the responsibility of configuring in log publish service or mask the data before calling service.



Instructions

What imposed or practical considerations place limits on the range of acceptable solutions?

List any bounds to which the design described in this document must conform. These might include:

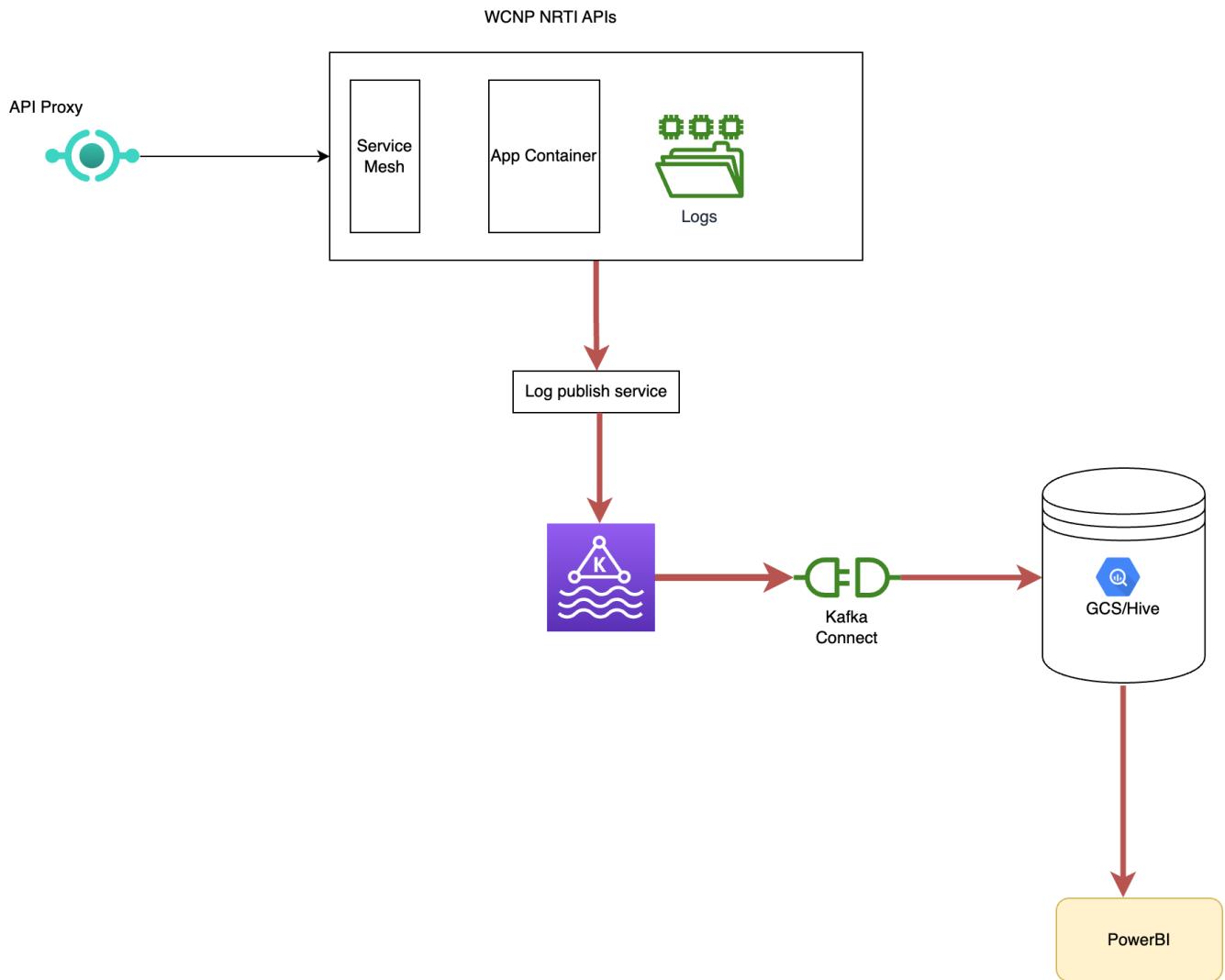
- Dependent project schedules
- Timing with respect to business events
- Capital or operational budgets
- Availability of human resources, specific skill sets, or support providers
- Directives to use or avoid specific technologies, frameworks, platforms, or services
- Directives to further specific strategic goals
- Expected lifetime of system
- Expected lifetime or disposition of intellectual property (e.g., open source)
- Migrations from existing systems

Read the description of the [Constraints](#) section of the [software guidebook](#) for more help.

Design

Service that accepts rest api log with request payload, response payload and other metadata (endpoint, version etc) from multiple services and write to Kafka topic. The information from kafka will be persisted to storage. The information would be used to derive metrics and compare YoY data etc., and use it for troubleshooting.

High Level Architecture



Service Boundary

API operation summary	Endpoint	Description	Sample request payload	Sample response payload

Log http request with request and response body and status.	POST v1/logs /api-requests	Rest endpoint to publish the api log into kafka topic.	<pre>{ "request_id" : "97e6f2ff-691d-449d-ac6d-4e3a9bf27b6f", "service_name" : "NRT", "endpoint_name" : "Transaction history", "version" : "v1", "path" : "/store/100/gtin/000123223213/transactionHistory", "host" : "http://api.walmartdataventures.com", "supplier_company" : "luminate_company_id", "method" : "GET", "request_body" : null, "response_body" : {"gtin": "000123223213", "event": "SALES"}, "response_code" : 200, "error_reason" : "if any", "request_ts" : 1732760547, "response_ts" : 1732760750, "request_size_bytes" : 2000, "response_size_bytes" : 5000, "created_ts" : 1732760900, "trace_id" : "hjahsjdkhasjdhad", "headers" : {"header1": "value1"} }</pre>	204 No Response
---	----------------------------	--	--	-----------------

Log Publish service details:

- **Responsibilities:**
 - publish an event to persisting the log.
- **APIs:**
 - API namespace: logs
 - Resource Collection: api-requests
 - API Operations:
 - Publish log details : **POST /v1/logs/api-requests**
- **Kafka Connect :**
 - Map data from kafka topic to presto table

Recommendation to consumers is to make non blocking async call.

Presto Table : api_log (Run MSCK REPAIR TABLE every day for updating partitions).

Partition key : service_name, date, endpoint_name

Format : Parquet.

Attribute name	type	Desc
source_request_id	String	source request id, if source wants to track.
service_name	String	service name. eg : NRT
api_version	String	version of api, v1 or v2
endpoint_name	String	Name for endpoint. eg : transaction history
endpoint_path	String	path eg : /inventory/status
consumer_id	String	UUID of the consumer accessing the endpoint.
supplier_company	String	optional for internal endpoint
method	String	http method.
request_ts	Timestamp	
response_ts	Timestamp	

response_code	Integer	Http response code. 200, 400 etc..
request_body	String	
response_body	String	optional
error_reason	String	
request_size_bytes	numeric	
response_size_bytes	Numeric	
trace_id	String	
created_ts	Timestamp	
headers	key-value pair	

Analysis

[Back to Top...](#)

Maintainability

[Back to Top...](#)

- *Infrastructure and Load Balancing* : auto-scaling , load balancing , isolated resource pools
- *Separation of Concerns* : Each tenant's data, configurations, and custom logic to be logically isolated from others while maintaining a shared infrastructure. Every low level design must keep this approach intact.
- *Automated Monitoring and Alerts* : Golden signals , alerts and onboarding to nucleus health track system to be done. tenant specific queries to be configured.
- *Documentation* : Run-books & knowledge bases [**DR & Failure recovery playbooks critical**]=
- *Security and Compliance Maintenance* : application scales, maintaining security and compliance becomes more challenging, especially with multiple tenants and differing regulatory needs.



Instructions

Given the design described above, what are the maintenance, recovery, and other operational processes and tools needed to manage the system over time?

Examples are procedures, automation, and capabilities to support:

- Backups
- Deletion of data that is no longer required
- Failure recovery
- Disaster recovery
- Component upgrades
- Replacement of components
- Migration of components

Observability

[Back to Top...](#)

- Golden signals dashboard for application health monitoring.
- Istio dashboard for incoming requests and status.
- Splunk alerts for API failures .
- Alerts enabled for application health, failures and response latency.
- Latency tracking to measure end-to-end processing time.
- Error rate monitoring to detect and respond to processing errors.

- Resource utilization monitoring (CPU, memory, disk I/O) to optimize performance.
- Checkpoint and offset tracking to ensure no data loss or duplication.
- Log aggregation and analysis to troubleshoot issues quickly.



Instructions

Identify the measurements needed to predict, prevent, and diagnose violations of the system's [Service level objectives](#).

Consider both resource-oriented measurements and service-level measurements, which can be addressed by the [USE](#) and [RED](#) methodologies, respectively.

Note that while literature about the USE method is often focused on physical infrastructure and operating system-level resources, it also applies well to software infrastructure and managed services. For instance, a distributed database or messaging system provided as a managed service is a compound resource composed of component compute and storage resources. A team maintaining a system that uses such a compound resource must be concerned with measurement of utilization, saturation, and errors with respect to the compound resource as a unit, while the team that maintains that compound resource itself would be the one concerned with lower-level USE metrics for the components of that resource.

Testability

[Back to Top...](#)

Due to the stateless nature of our API, the testing process is more streamlined and efficient.

- Unit testing with code coverage of more than 80%
- Regression Testing for existing APIs.
- Functional Testing
- Performance Testing
 - Load testing - Automaton tests to be run to check the SLA response time during load tests.
 - Stress testing - Automaton tests to be run to measure the application performance for the throughput.
 - Scaling - Automaton tests to be run to measure the application performance for the throughput.
 - Maintenance
 - Disaster recovery
- Security testing
 - Penetration - PEN tests will be completed during SSP review.



Instructions

How is the system designed to be tested?

Describe at a high level how the system is expected to be tested in its entirety to verify delivery of all requirements. Highlight requirements that the implementation must satisfy in order to support such testing in addition to unit and other build-time testing. Consider support for at least the following types of testing, both before and in production:

- Functional Testing
 - Load testing
 - Stress testing
 - Failure testing
- Performance Testing
 - Scaling
 - Maintenance
 - Disaster recovery
- Operational Testing
 - Scaling
 - Maintenance
 - Disaster recovery
- Analytics Testing
- Experimentation
 - Business impact testing
 - Experience optimization
- Security testing
 - Penetration
 - Data integrity
 - Denial of service

Discarded Alternatives

[Back to Top...](#)

Reading logs from api proxy, API Proxy is logging the request and response code, but not the body.



Instructions

Describe alternatives considered, commonly suggested, or previously implemented and why they were discarded in favor of the design described above.

This section acts as a log of [architectural decisions](#). Unlike a "traditional" [architecture decision log](#), the decisions here are written with reference to the design described above and should be updated as needed to reflect changes in that design (or even its requirements) over time. For instance, given the following sequence of events:

1. The design initially depends on technology X
2. After a POC, technology X is found to be deficient and the design is updated to use technology Y instead
3. It is found that the POC for X was flawed, and because Y has been found to be more expensive than X, the design is reverted back to use of X

Assuming no other decisions to document, this section should go through this sequence of states:

1. No content, as the reasoning behind use of X should already be covered in the [Design](#) section
2. A single subsection rejecting the use of technology X that describes the POC and its findings, with the Design section now covering the choice to use technology Y
3. A single subsection rejecting the use of technology Y because of its cost relative to X, with the Design section reverted back to its original discussion of X

	<p>This structure allows a reader to quickly understand the entire current design and the key learnings that have been factored into it without needing to wade through the entire journey. Should it be of interest, the journey is still preserved because previous versions of the document can be retrieved.</p> <p>While it permissible (and recommended if applicable) to have subsections describing discarded alternatives at the outset, it is not required.</p>
--	---

Appendix

[Back to Top...](#)

Definitions/Acronyms

[Back to Top...](#)

Instructions

Add the definitions of jargon used in the document, especially if not defined upon first use.

References

[Back to Top...](#)

Instructions

Add any relevant references, especially if not otherwise introduced as links in the text.

Contributors

[Back to Top...](#)

- [Asok Palugulla](#)
- [Andrei Komolov](#)