INTUIT turbotax creditkarma quickbooks mailchimp

Trino for Observability at Intuit

December 2024

Riya John, SSE, Observability - Logging
Ujjwal Sharma, SE2, Observability - Logging

Speakers

Riya John
Senior Software Engineer

Ujjwal Sharma
Software Engineer 2

INTUIT

turbotax

creditkarma

quickbooks

mailchimp

Agenda

Logging at Intuit - Overview

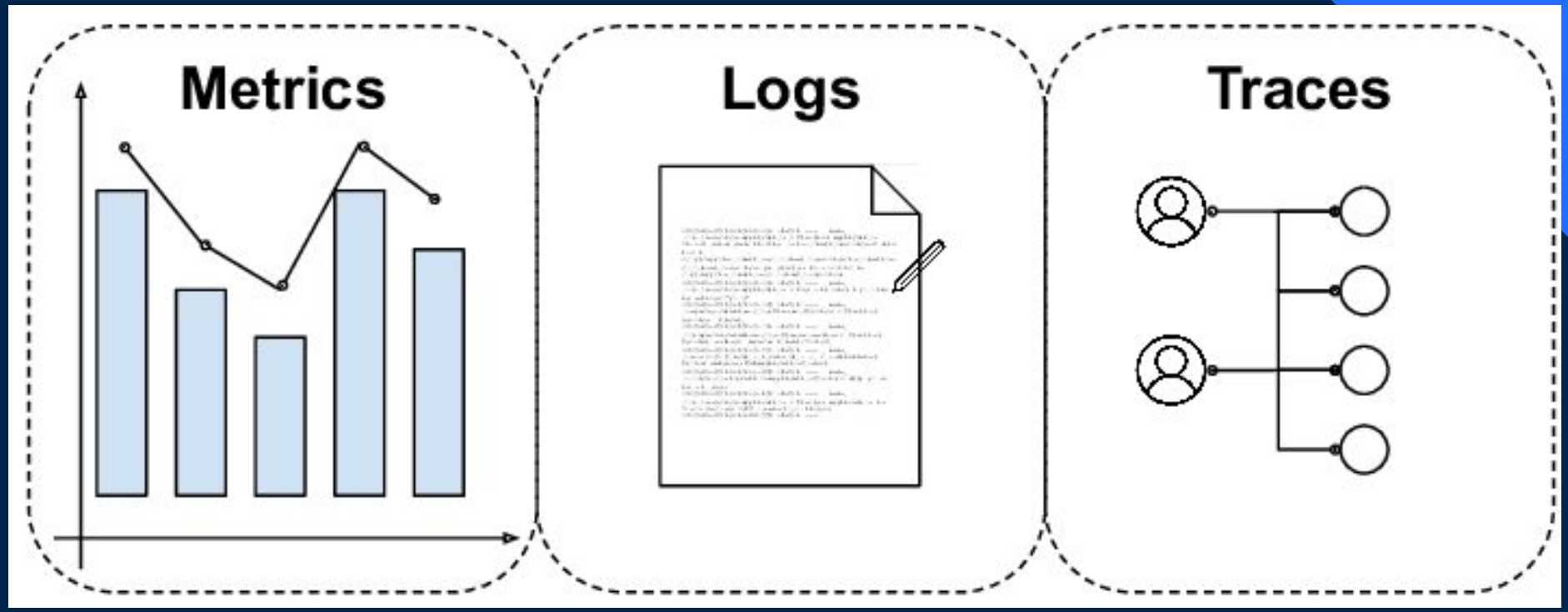
Use Case

Solution with Trino + Iceberg

Ongoing efforts

AI and Trino

You cannot fix or improve what you cannot measure, Observability helps you measure.


Metrics
Logs
Traces

Logging Platform @ Intuit

We have a high performing, scalable, and fault tolerant infrastructure handling all the logs at intuit

800 TB+	< 10s	2k+
Avg. Volume per Day	E2E Avg Latency	Services

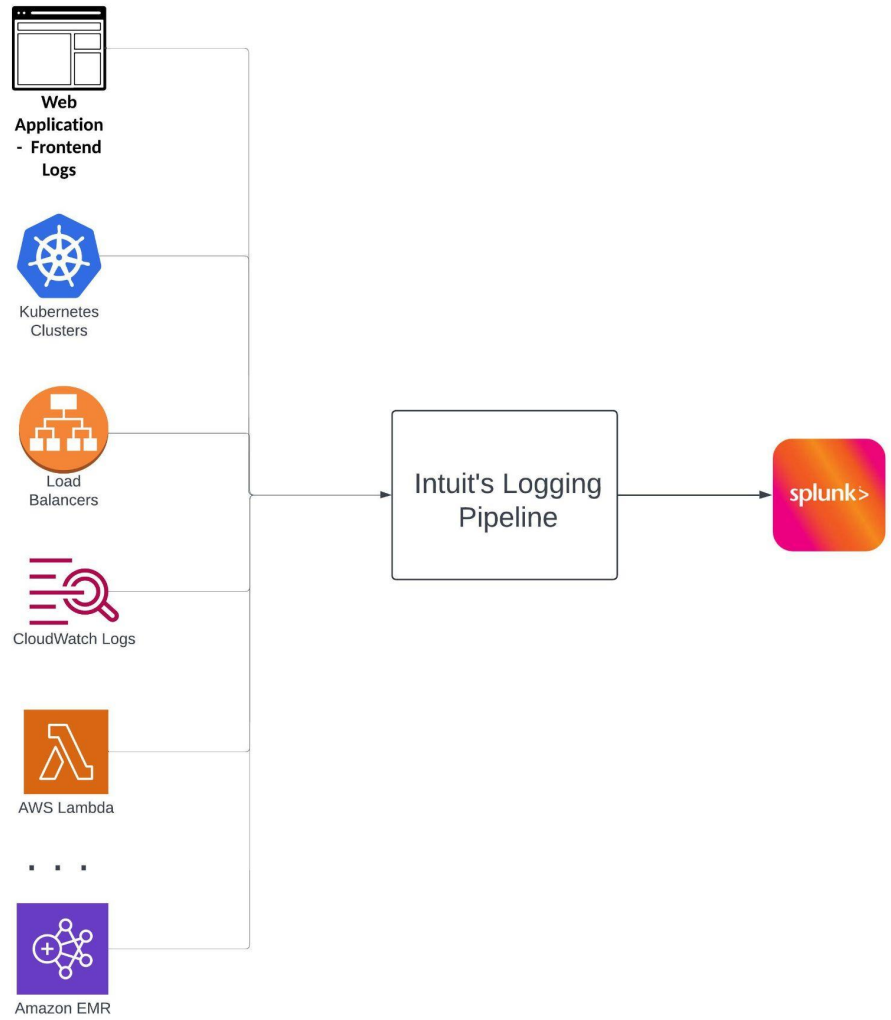
Use-case

The Problem

Splunk alone today handles the ingestion and querying of all the logs at Intuit

We have an ever increasing log volume that Splunk handles gracefully from storing to important reporting of essential logs.

With huge volume we incur scalability issues with regular pileups which brings necessity for an alternate solution for Logs which are easier to maintain and can be stored in a cost effective way.



The Goal

- Reduce the operational costs associated with Splunk

- Offload the ingestion and querying of a subset of logs to an alternate solution where the operational cost is less

- A new solution that should be
    - Reliable
    - Scalable
    - At Speed comparable to Splunk

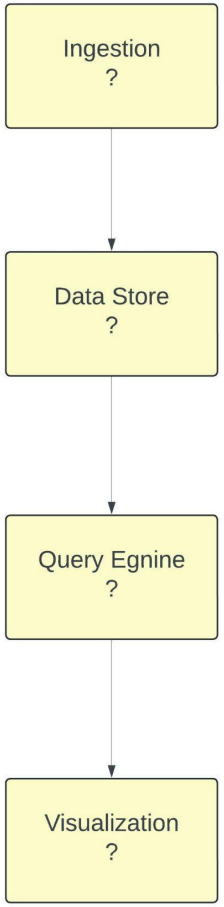
- Provide a good Customer Experience so that users can easily migrate from the existing solution.

Happy Customers

Journey to our solution begins

An E2E pipeline that handles the ingestion and querying of logs using open source applications

- Which subset of the current log volume is a good candidate to offload?
- What is the cost effective alternate store?
- What is the Alternate format to store that helps querying?
- What is the most compatible writer with this Store?
- How do we efficiently Query this Store?
- How do we Visualize the Data?



What is the Best Subset?

Candidate: Logs that follow a predefined schema

```
http 2018-07-02T22:23:00.186641Z app/my-loadbalancer/50dc6c495c0c9188
192.168.131.39:2817 10.0.0.1:80 0.000 0.001 0.000 200 200 34 366
"GET http://www.example.com:80/ HTTP/1.1" "curl/7.46.0" - -
arn:aws:elasticloadbalancing:us-east-2:123456789012:targetgroup/my-targets/73e2d6bc24d8a067
"Root=1-58337262-36d228ad5d99923122bbe354" "-" "-"
0 2018-07-02T22:22:48.364000Z "forward" "-" "-" "10.0.0.1:80" "200" "-" "-"
```


```
[
  {
    "type": "http",
    "time": "2018-07-02T22:23:00.186641Z",
    "elb": "app/my-loadbalancer/50dc6c495c0c9188",
    "client_port": "192.168.131.39:2817",
    "target_port": "10.0.0.1:80",
    "request_processing_time": 0.001,
    "target_processing_time": 0.091,
    "response_processing_time": 0,
    "elb_status_code": 200,
    "target_status_code": 200,
    "received_bytes": 3459,
    "sent_bytes": 3428,
    "request": "GET http://www.example.com:80/ HTTP/1.1",
    "user_agent": "gSOAP/2.8",
    "ssl_cipher": "ECDHE-RSA-AES128-GCM-SHA256",
    "ssl_protocol": "TLSv1.2",
    "target_group_arn": "arn:aws:elasticloadbalancing:us-east-2:123456789012:targetgroup/my-targets/73
    "trace_id": "Root=1-58337262-36d228ad5d99923122bbe354",
    "domain_name": "example.com",
    "chosen_cert_arn": "arn:aws:acm:us-west-2:123456789012:certificate/a6457test",
    "matched_rule_priority": 1,
    "account": "123456789012",
    "request_creation_time": "2024-11-22T04:10:13.445000Z",
    "actions_executed": "waf,forward",
    "redirect_url": "-",
    "error_reason": "-",
    "target_port_list": "10.0.0.1:80",
    "target_status_code_list": "200",
    "classification": "-",
    "classification_reason": "-",
    "conn_trace_id": "Root=1-58337262-36d228ad5d99923122bbe354",
    "request_url": "http://www.example.com:80/",
    "request_protocol": "GET",
    "host": "example.com",
    "elb_region": "us-west-2",
    "elb_ip": "10.0.0.1:80"
  }
]
```

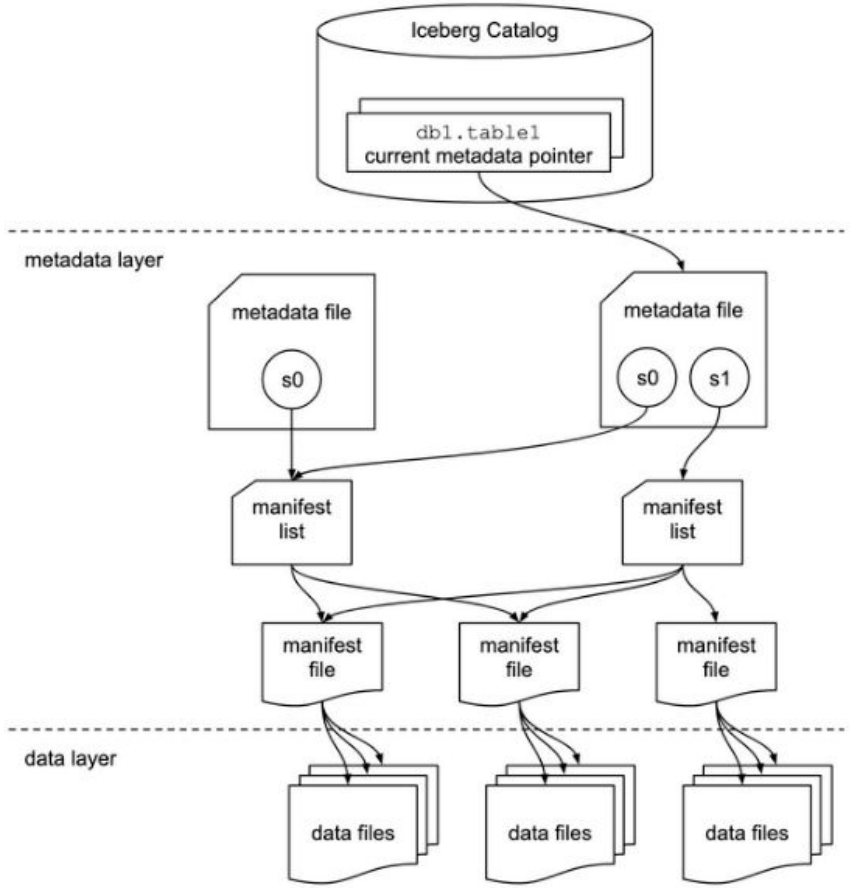

Use a Table based design for logs that have a predefined schema of key value pairs

Where and How to Store the Data?

Apache Iceberg tables on AWS S3

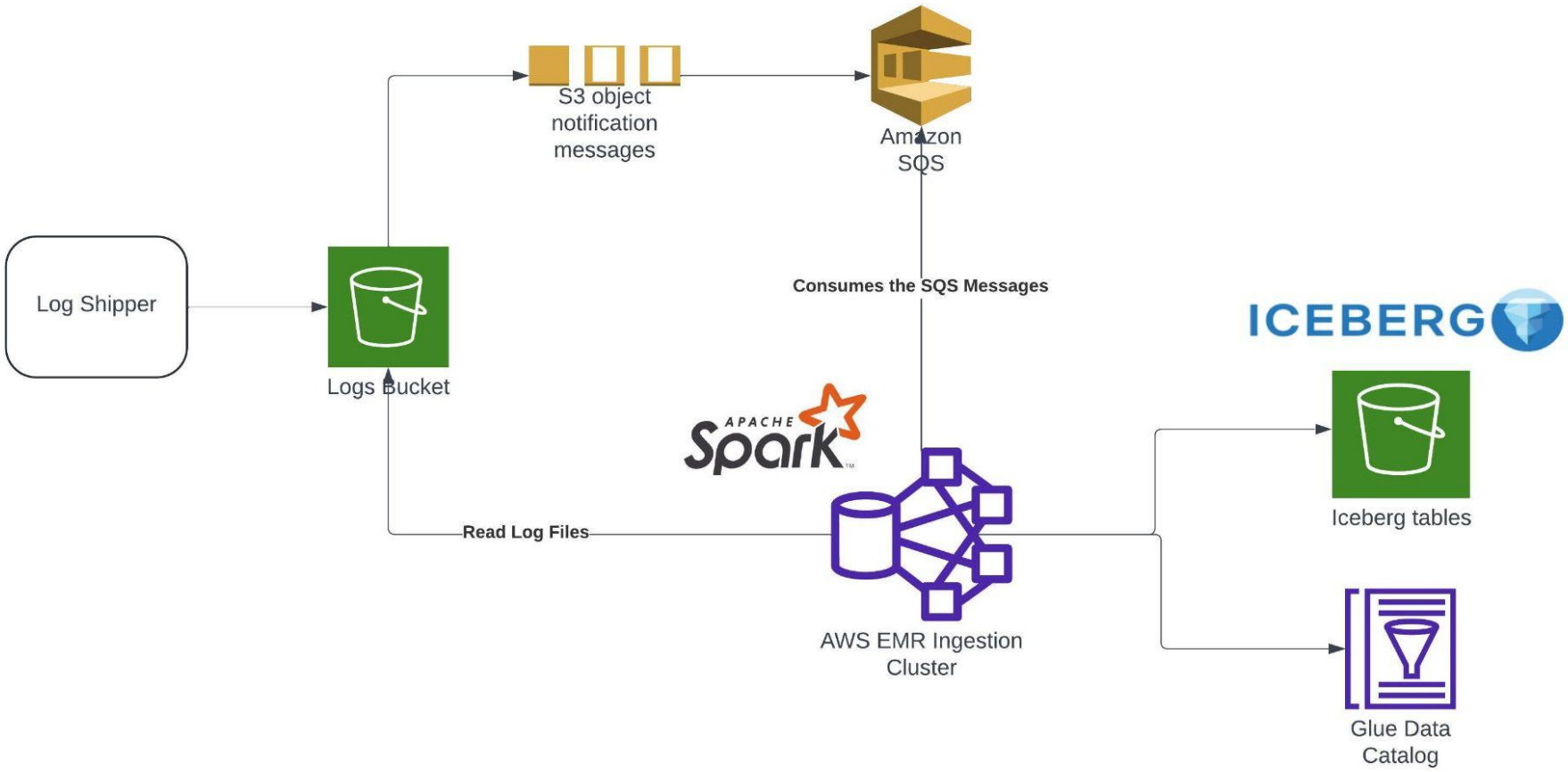
What is Apache Iceberg?

ICEBERG



How do we setup Ingestion?

Ingestion Pipeline



Search for a Query Engine

Query Engine Test Requirements

1. Target State Requirements for Discovery Tests
   a. **Support 50 Concurrent Users**
   b. **Highly Available Query Engine**
   c. **Queries can range from simple Scan, Filter, Aggregation, Percentile and Regex based.**
   d. **Complete Configuration & Infrastructure Control on the Query Engine and Cost Efficiency.**

2. Query Engines considered

Experimentation Scenarios

1. Identification of Type of Queries to run
   a. Aggregate
   b. Percentile
   c. Regex
2. Load should be corresponding to 50 concurrent users running Ad-hoc Queries
3. Query Engines should have the same underlying infrastructure (m5d-32xlarge nodes) (With exception of Athena as it is a managed solution).

Query Engine	CPU for Nodes	Memory for Nodes	Worker Replicas
Trino	32 / 16	64 Gi	20
Starrocks	32 / 16	64 Gi	20

Single Query Results

Test			Athena	Starrocks	Trino
Single Query					
	Aggregation	Time	19.753 sec	**7.56 seconds**	9.93 seconds
		Data	**4.22 GB**	4.57 GB	4.41GB
		S3 Call	5304	4430	**4380**
	Percentile	Time	36.342 sec	30.56 sec	**19.56 sec**
		Data	**8.19 GB**	8.84GB	8.53GB
		S3 Call	4732	N/A	**3816**
	Regex	Time	20.646 sec	**8.39 sec**	15.53 sec
		Data	**3.04 GB**	3.30GB	3.21GB
		S3 Call	3945	3132	**3028**

Load Test Results

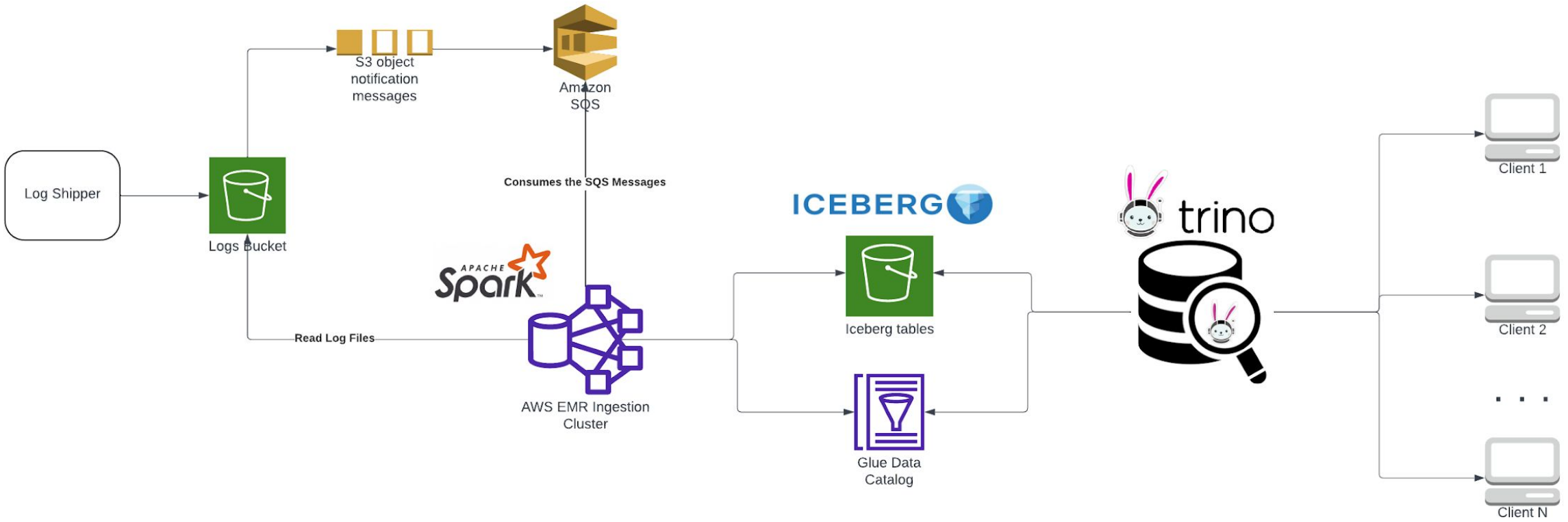
Test		Time	Starrocks	Trino
Load Test				
	Aggregation	MIN	**3.28 seconds**	9.60 seconds
		MAX	3 minutes 5 seconds	**3 minutes 4 sec**
		AVG	**34.42 seconds**	36.91 seconds
	Percentile	MIN	**OOM***	**10.5 sec**
		MAX	**OOM**	**35 minutes**
		AVG	**OOM**	**9.4 minutes**
	Regex	MIN	**OOM**	**5.55 sec**
		MAX	**OOM**	**5 mins**
		AVG	**OOM**	**47.11 sec**

* Starrocks showed OOM with the exact config given to Trino with limited resources

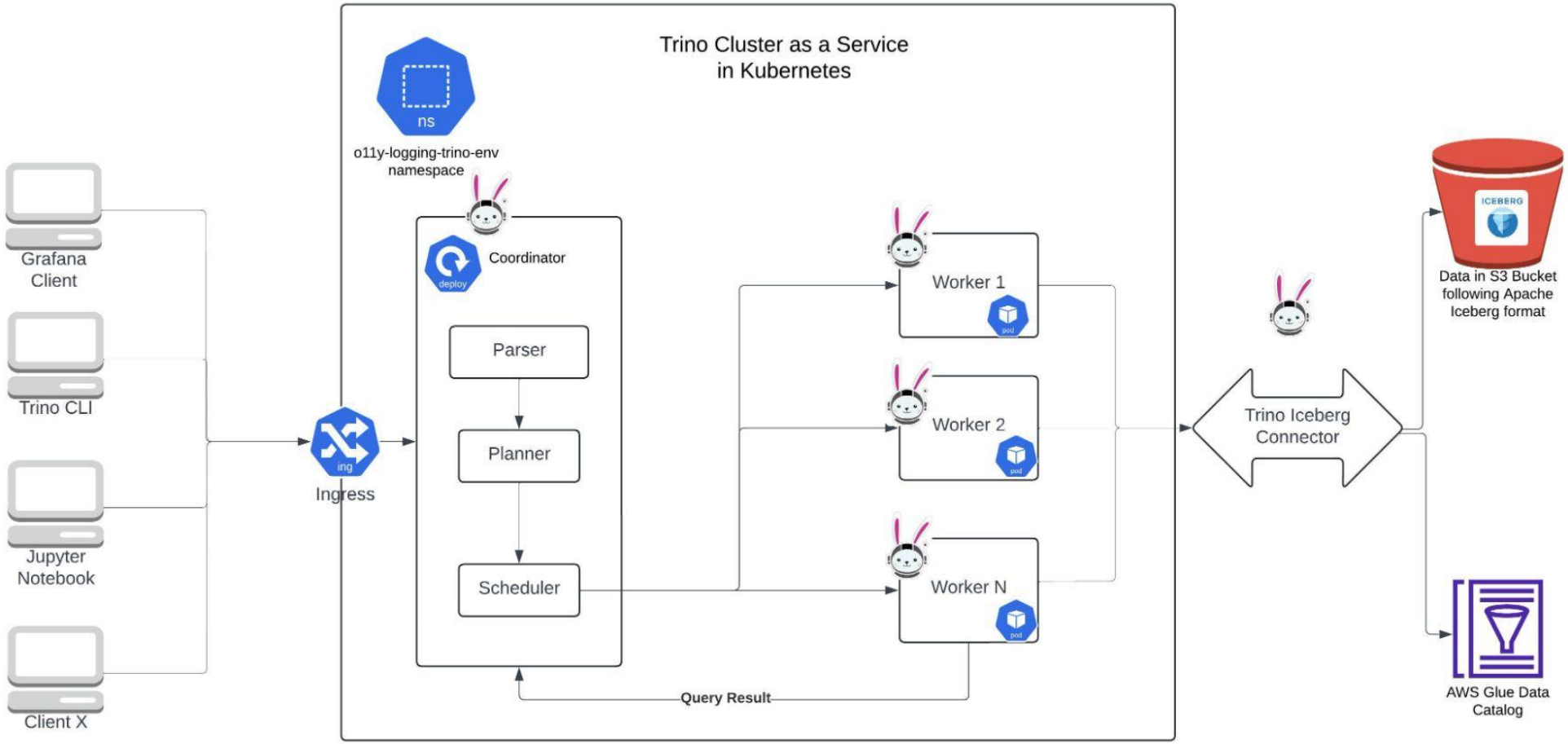
End-to-End Pipeline with Trino and Iceberg

INTUIT

E2E Pipeline



Setup of Trino on Kubernetes



Deployment using ArgoCD

Auto-Scaling

Horizontal Pod AutoScaling & Pod Distribution Budget is set on Trino Worker Pods

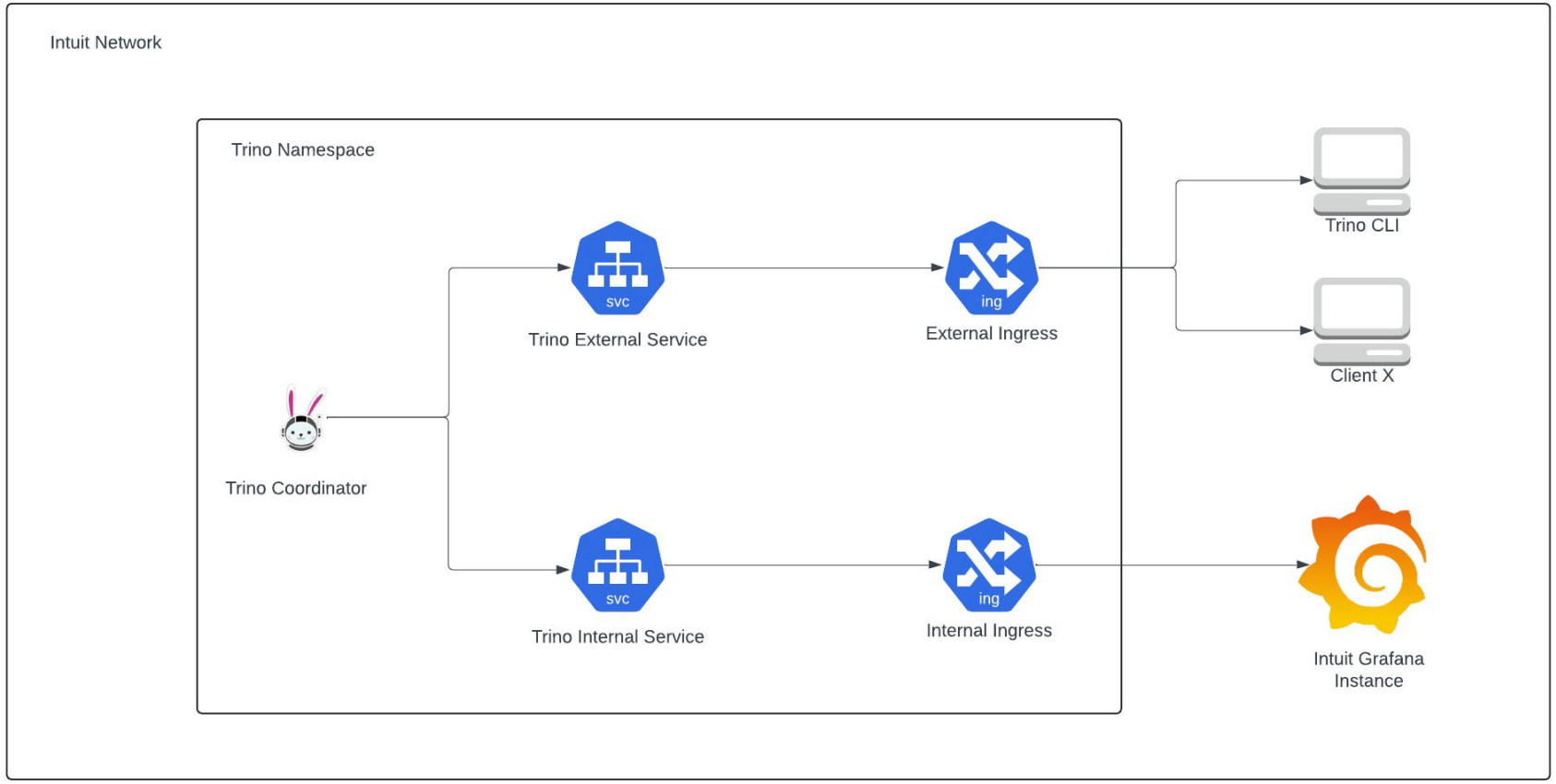
HPA is set on Trino Worker pods such that the pods will auto-scale as the CPU utilization increases(>70%) on Trino Workers with increase in traffic or with a complex query

We can optimize it further to evaluate other metrics like

- Memory Utilization of Workers
- Number of Queries in the Queue
- Number of Running Queries
- Average response time of queries

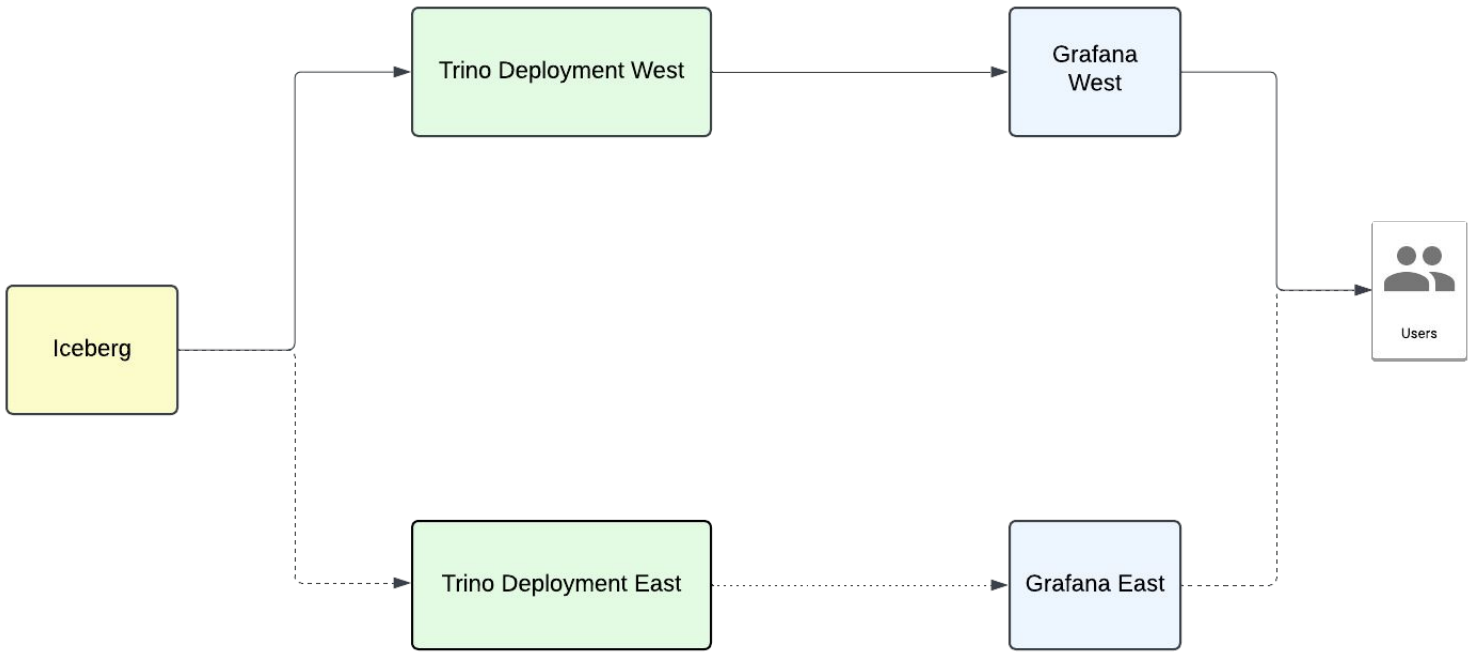
Enable HPA on Trino Worker Pods

Exposing Trino to Customers



Dual Ingress Setup helped us expose Trino to Grafana and any other client

HA/DR Setup



We have Active-Active Setup for Trino Enabled, with cross-region support to the backend.

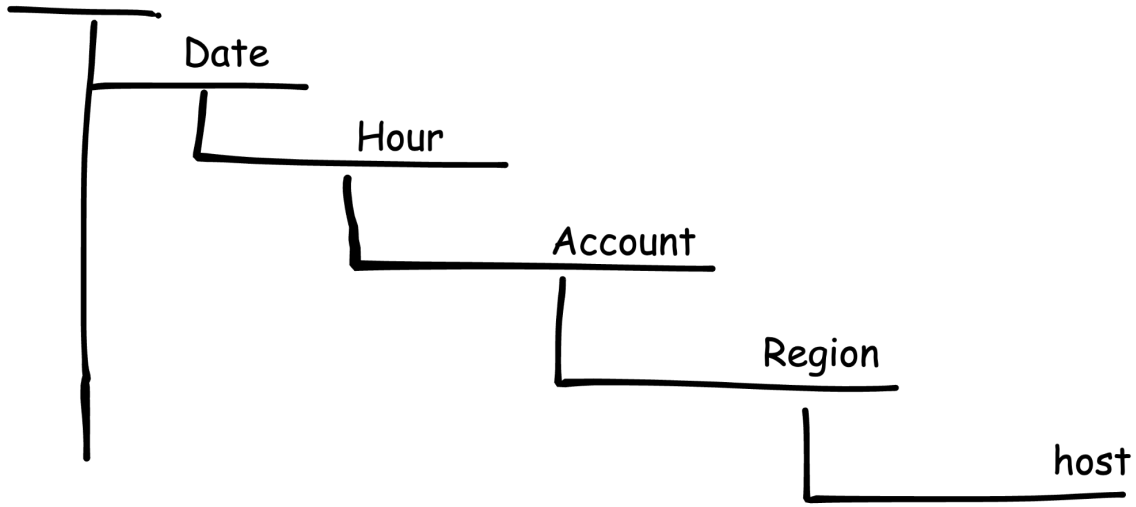
First Customer: AWS ELB Logs

AWS ELB Logs Schema

ELB Logs follow key value pairs and a predefined schema

- All logs are time bound and are specific to given AWS Account and furthermore teams are interested at a specific AWS region and/or by their host name

```
http 2018-07-02T22:23:00.186641Z app/my-loadbalancer/50dc6c495c0c9188
192.168.131.39:2817 10.0.0.1:80 0.000 0.001 0.000 200 200 34 366
"GET http://www.example.com:80/ HTTP/1.1" "curl/7.46.0" - -
arn:aws:elasticloadbalancing:us-east-2:123456789012:targetgroup/my-targets/73e2d6bc24d8a067
"Root=1-58337262-36d228ad5d99923122bbe354" "-" "-"
0 2018-07-02T22:22:48.364000Z "forward" "-" "-" "10.0.0.1:80" "200" "-" "-"
```



Field Name	*Data Type*
account	string
type	string
time	timestamp
elb	string
client_port	string
target_port	string
request_processing_time	float
target_processing_time	float
response_processing_time	float
elb_status_code	int
target_status_code	int
received_bytes	long
sent_bytes	long
request	string
user_agent	string
ssl_cipher	string
ssl_protocol	string
target_group_arn	string
trace_id	string
domain_name	string
chosen_cert_arn	string
matched_rule_priority	int
request_creation_time	string
actions_executed	string
redirect_url	string
error_reason	string
target_port_list	string
target_status_code_list	string
classification	string
classification_reason	string
conn_trace_id	string
request_url	string
request_protocol	string
host	string
source	string
root_trace_id	string
self_trace_id	string
elb_region	string
elb_ip	string

Trino Iceberg Connector

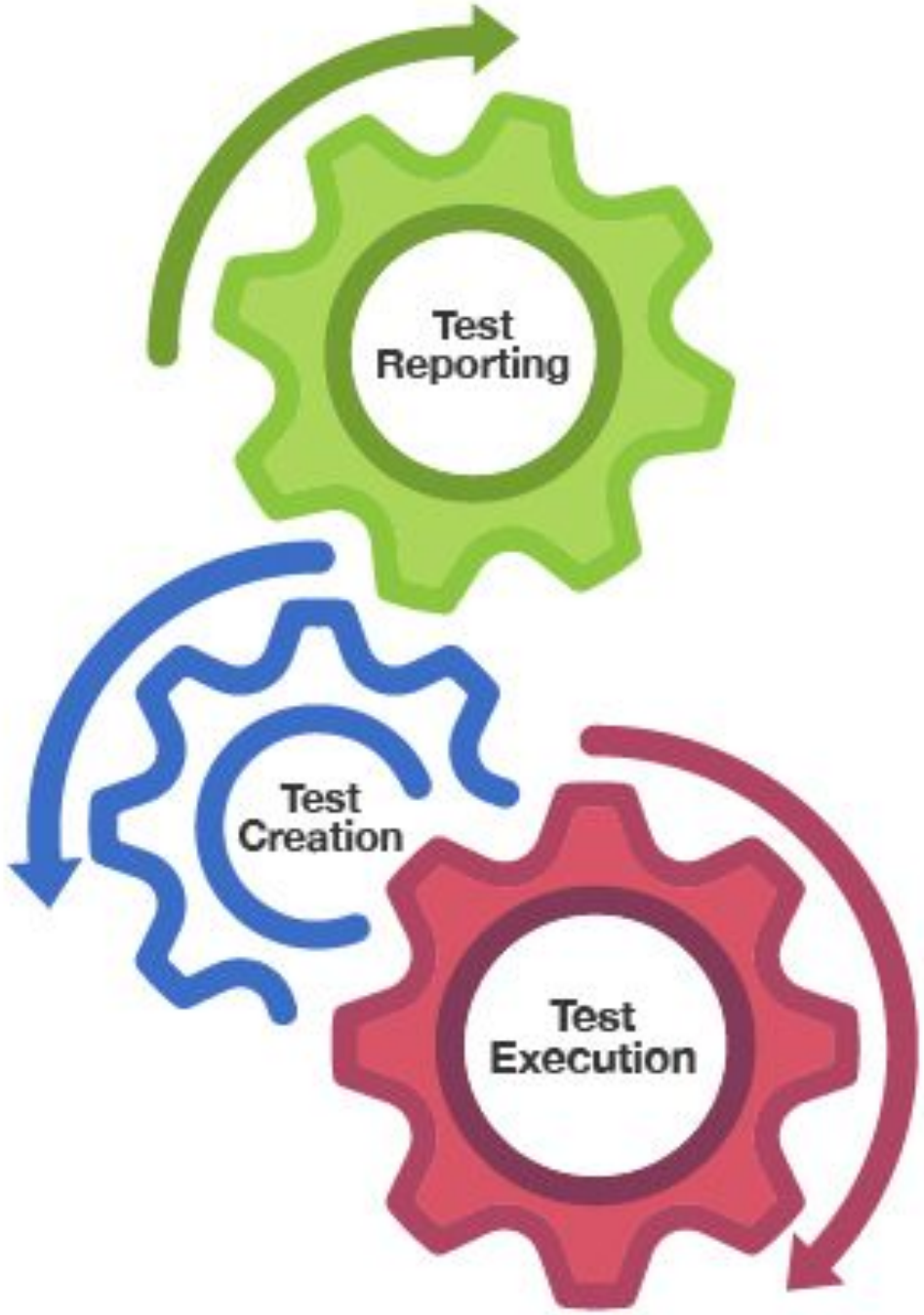

```
# Source: trino/templates/configmap-catalog.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: catalog
  labels:
    app: o11y-logging-trino
    app.kubernetes.io/name: trino
    app.kubernetes.io/component: catalogs
data:
  tpch.properties: |
    connector.name=tpch
    tpch.splits-per-node=4
  tpcds.properties: |
    connector.name=tpcds
    tpcds.splits-per-node=4
  structuredlogs.properties: |
    connector.name=iceberg
    iceberg.catalog.type=glue
    hive.s3.region=us-west-2
    hive.s3.iam-role=arn:aws:iam::[redacted]:role/o11y-logging-trino-access
    hive.metastore.glue.region=us-west-2
    hive.metastore.glue.iam-role=arn:aws:iam::[redacted]:role/o11y-logging-trino-access
```

Use Trino Iceberg Connector to connect to data following Apache Iceberg table format

A Test Bed

Test Suite in Golang - from test creation to report generation

1. Splunk Comparison Test
   a. Data Parity check
   b. Query Performance check
2. Functional Test - to evaluate the functional parity when switching from a Splunk SPL based query language to Trino SQL based query language
3. Performance and Load Test to evaluate the
   a. Speed
   b. Scalability
   c. Reliability
   d. Resource utilization



Functional Test

FUNCTION	STATUS	DESCRIPTION
Standard SQL Clauses		Support for SELECT, GROUP BY, HAVING, ORDER BY, UNION
Comparison operators		Support for IS NULL, DISTINCT, <,>, != etc in queries
Nested queries		Nested SQL Queries
Aggregate functions		Run Aggregates like count, avg, sum etc on columns
Percentiles		Calculate percentiles P95,P99 etc on columns like average response time etc
Regex Based		Use Regex based functions on search columns

Current Load Test Numbers

Node size	• m5.8xLarge Nodes
Coordinator CPU/Memory	• 32 cores/128 GiB
Worker CPU/Memory	• 16 cores/8 GiB
Worker Nodes	• 5
Test queries	• Aggregate queries
Data scan time window	• 15 mins
Test run period	• 15 mins

5

Queries Per Second

2.17s

Avg Query Execution Time

0%

Error Rate

N Concurrent Queries at a given second

Node size Coordinator CPU/Memory	• m5.8xLarge Nodes • 32 cores/128 GiB
Worker CPU/Memory	• 16 cores/8 GiB
Worker Nodes	• 5

Test queries	• Aggregate queries
Data scan time window	• 4 hours
Data scan volume	• 4 GB

250

Concurrent Queries

2m 11 s

Avg Query Execution Time

0%

Error Rate

This is just the beginning

Rigorous Testing and Optimizations at all levels to meet our requirements -

- Instance Type Selections and Resource Allocations
- JVM Optimizations
- Tweaking Query and Task properties
- Partitioning Strategy on Ingestion Side

Explore Multiple Trino Clusters

- Trino Gateway Setup
- Explore Conditional Routing for Different sized Trino Clusters.

Evaluate Caching opportunities for improved query performance

A little bit of AI with Trino

Querying on a Jupyter Notebook

Explored querying Trino engine from a Jupyter Notebook seamlessly using JupySQL library in Python

- Good for Log Analytics using MatPlotLib, Plotly etc

JupySQL

```
pip install jupysql
```

Explored text-to-SQL generation using GenAI

Explored generating Trino SQL query from user input in plain English text

Exploring Conversion of Splunk SPL to Trino SQL using GPT 4

Using Langchain and Prompt Engineering, we could generate Trino SQL queries for Splunk search queries following Splunk SPL

- Helpful for conversion of Splunk search queries into Trino SQL queries

LangChain X OpenAI

Example: Text to Log Results with GenAI

```
TEXT QUERY :
Get records for domain name custlicvdt-prf.api.intuit.com and limit to 1 record


TRINO QUERY :
SELECT * FROM elblogs.sk_iceberg_db.alb_table_v4 WHERE domain_name = 'custlicvdt-prf.api.intuit.com' LIMIT 1

FETCHING OUTPUT FOR TRINO
['3[redacted]9', 'https', datetime.datetime(2024, 11, 18, 23, 5, 5, 202778, tzinfo=zoneinfo.ZoneInfo(key='UTC')), 'app/k8s-services-usw2prfd-44[redacted]3
7a22a69cd31b', '51.15.213.8:58080', '-', -1.0, -1.0, -1.0, 403, None, 392, 266, 'POST https://custlicvdt-prf.api.intuit.com:443/minio/webrpc HTTP/1.1', 'Moz
illa/5.0 (Macintosh; Intel Mac OS X 11) AppleWebKit/616.16 (KHTML, like Gecko) Version/17.0.90 Safari/616.16', 'ECDHE-RSA-AES128-GCM-SHA256', 'TLSv1.2', 'ar
n:aws:elasticloadbalancing:us-west-2:3[redacted]9:targetgroup/k8s-services-usw2prfd-[redacted]', 'Root=1-673bc821-6c1c36d603a607df5f4cb28b'
, 'custlicvdt-prf.api.intuit.com', 'arn:aws:acm:us-west-2:301532132469:certificate/a6457ab1-2069-4ca5-a51e-dabeeb181268', -1, '2024-11-18T23:05:05.055000Z',
 'waf', '-', '-', '-', '-', '-', '-', 'TID_402bc9a18b0d7a4e84a3ebd5acf46e35', 'https://custlicvdt-prf.api.intuit.com:443/minio/webrpc', 'POST', 'custlicvdt-
prf.api.intuit.com', 's3a://iks-logs-3[redacted]_elasticloadbalancing
_us-west-2_app.k8s-services-usw2prfd-448[redacted]', 'Root=1-673bc821-6c1c36d603a607df5f4cb28b', '
', 'us-west-2', '[redacted]']
```

With Prompt Engineering, we can achieve Text to Log Results with GenAI like in this case.

Example: Splunk SPL query to SQL

```
query = """index=aws_elb_access app/crmprd/* \"eai/start.swe\" target_status_code=5* | stats count by request,target_status_code,elb_status_code | where count > 5"""
final_prompt = prompt.format(splunk_query=query)
response =  llm.invoke(final_prompt)
# Notice how you get back a `parsed` section in output
# JSON(response.json())
output = pydantic_parser.parse(response.content)
print(output.sql_query)
```

```
SELECT request, target_status_code, elb_status_code, COUNT(*) as count FROM elblogs.table_v4 WHERE request LIKE '%eai/start.swe%' AND target_status_code LIKE '5%' GROUP BY request, target_status_code, e
b_status_code HAVING COUNT(*) > 5
```

```
query = """index=aws_elb_access app/crmprd/* \"eai/start.swe\" target_status_code=5* | stats count by request,target_status_code,elb_status_code | where count > 5"""
sql_query = "SELECT request, target_status_code, elb_status_code, COUNT(*) as count FROM elblogs.table_v4 WHERE request LIKE '%eai/start.swe%' AND target_status_code LIKE '5%' GROUP BY request, target_sta
error = "[58] Query failed (#20241025_053819_00021_rqcs2): line 3:42: Left side of LIKE expression must evaluate to a varchar (actual: integer) io.trino.spi.TrinoException: line 3:42: Left side of LIKE ex
final_error_prompt = error_prompt.format(splunk_query=query, sql_query=sql_query, error=error)
# print(final_error_prompt)

response =  llm.invoke(final_error_prompt)
# Notice how you get back a `parsed` section in output
# JSON(response.json())
output = pydantic_parser.parse(response.content)
print(output.sql_query)
```

```
SELECT request, target_status_code, elb_status_code, COUNT(*) as count FROM elblogs.table_v4 WHERE request LIKE '%eai/start.swe%' AND CAST(target_status_code AS VARCHAR) LIKE '5%' GROUP BY request, targe
t_status_code, elb_status_code HAVING COUNT(*) > 5
```

```
query = '''SELECT elb_status_code, count(elb_status_code) as count FROM test_alb_table_v02
WHERE account='738495399770' AND  time >= timestamp '2024-11-04 00:00:00.000Z'
and time <= timestamp '2024-11-04 11:59:59.000Z'
GROUP BY elb_status_code'''
```

```
res = %sql {{query}}
```

```
Running query in 'trino://admin@o11y-logging-trino-service:8080/elblogs/iceberg_db'
```

```
print(res)
```

```
+-----------------+---------+
| elb_status_code |  count  |
+-----------------+---------+
|       200       | 7654477 |
|       404       |  22075  |
|       460       |  264215 |
|       413       |   4006  |
|       505       |    4    |
|       500       |  47430  |
|       204       | 2851038 |
|       502       |    50   |
|       400       |  602312 |
|        0        |    4    |
+-----------------+---------+
```

Splunk Query

1st SQL Output - Error

Error passed to the prompt

2nd SQL Output - Valid

Output Verified

Thank you