Software project

Analysis and monitoring of product upload dynamics on the marketplace platform

OZON tech

Ibragimova Taisiya

About

This project develops a system to collect, process, and analyze large volumes of seller product-upload events on the Ozon marketplace

About

This project develops a system to collect, process, and analyze large volumes of seller product-upload events on the Ozon marketplace

Focus areas

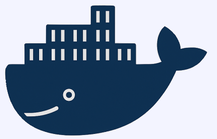
- Development of architecture
- Statistical metrics computation
- Visualization
- Automated alerting

Technologies


Clickhouse


Prometheus


Grafana


Docker


Consul

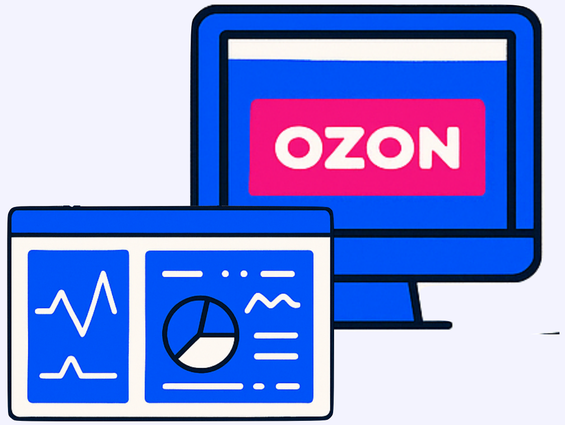

Zookeeper


Python


Kafka

OZON tech

Problem statement



Ozon's marketplace generates a high-velocity stream of product load events from thousands of sellers.

Key challenges

- Handling millions of small, frequent events per day
- Ensuring data consistency and replication
- Delivering real-time analytics and alerts

Tasks

Step 1. Draft requirements

Domain description, functional & non-functional specs, constraints

Step 2. Compare alternatives

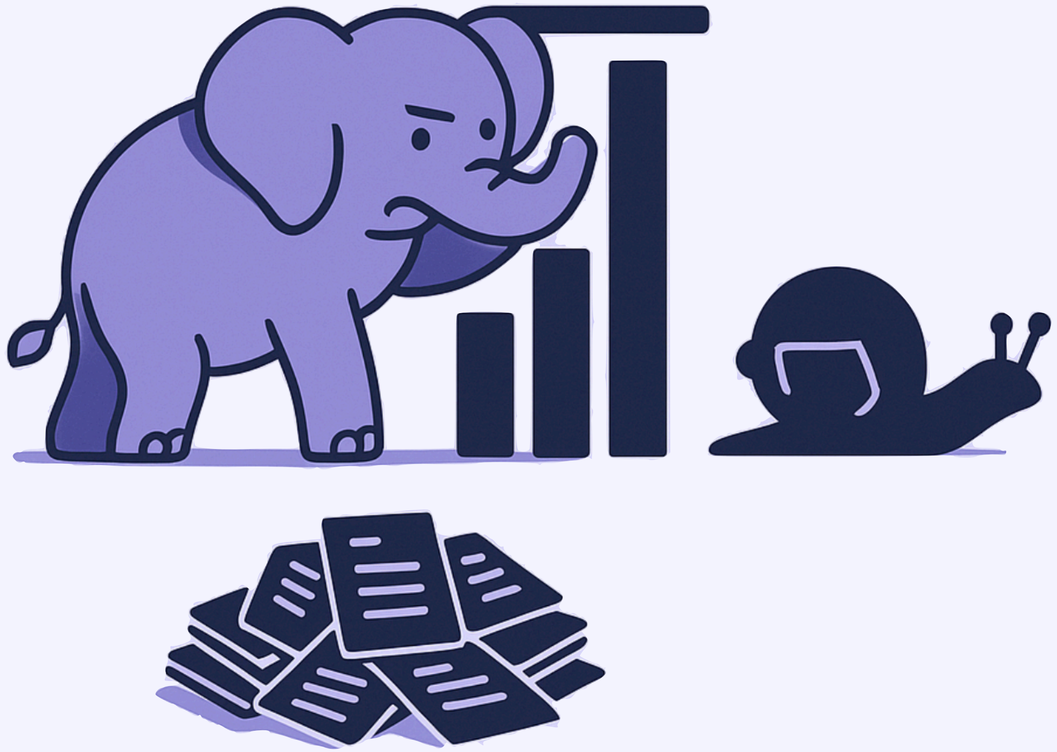
Step 3. Implement solution

- Implement database sharding locally (via zookeeper)
- Deploy ClickHouse, Grafana, Prometheus (via docker-compose)
- Create microservices that will simulate data stream and compute metrics
- Connect & register microservices (via consul)
- Write simulation script
- Create leader election for microservices
- Develop ETL pipelines & pre-aggregations
- Build anomaly detection modules

Step 4. Test the system

Limitations of PostgreSQL

- Vertical scaling limits
- WAL/MVCC overhead under heavy write load
- Slow OLAP queries on growing datasets



Limitations of PostgreSQL

Scalability

only vertical or external sharding (Citus/Postgres-XL)

Storage

row-oriented, sub-optimal for analytics.

Transactions

single-node ACID; no built-in 2PC across cluster

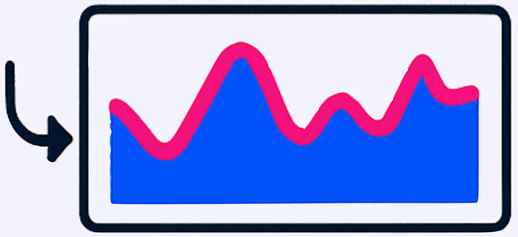
Maintenance

long VACUUM, index rebuilds on TB-scale tables

Write throughput

WAL contention and locking delays

Functional Requirements



The system has to:

- Survive the ingestion of maximum 1000 rps
- Keep one year of history online
- Provide **control-charts** for upload latency, failure rate, and seller anomalies
- Trigger **Prometheus alerts** when specific metrics exceed the limit

Alternative approaches

Cassandra

A highly scalable database optimized for fast writes, but less suitable for real-time analytics due to eventual consistency.

Apache Druid

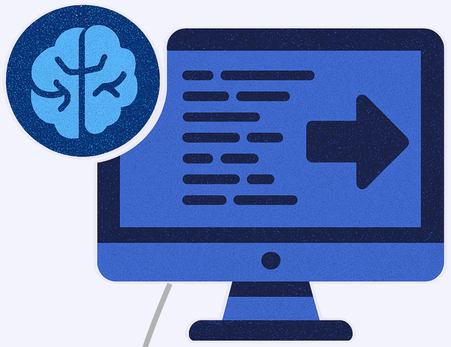
A fault-tolerant, distributed analytics engine built for fast, interactive queries over large-scale time-series data.

Implementation

OZONtech

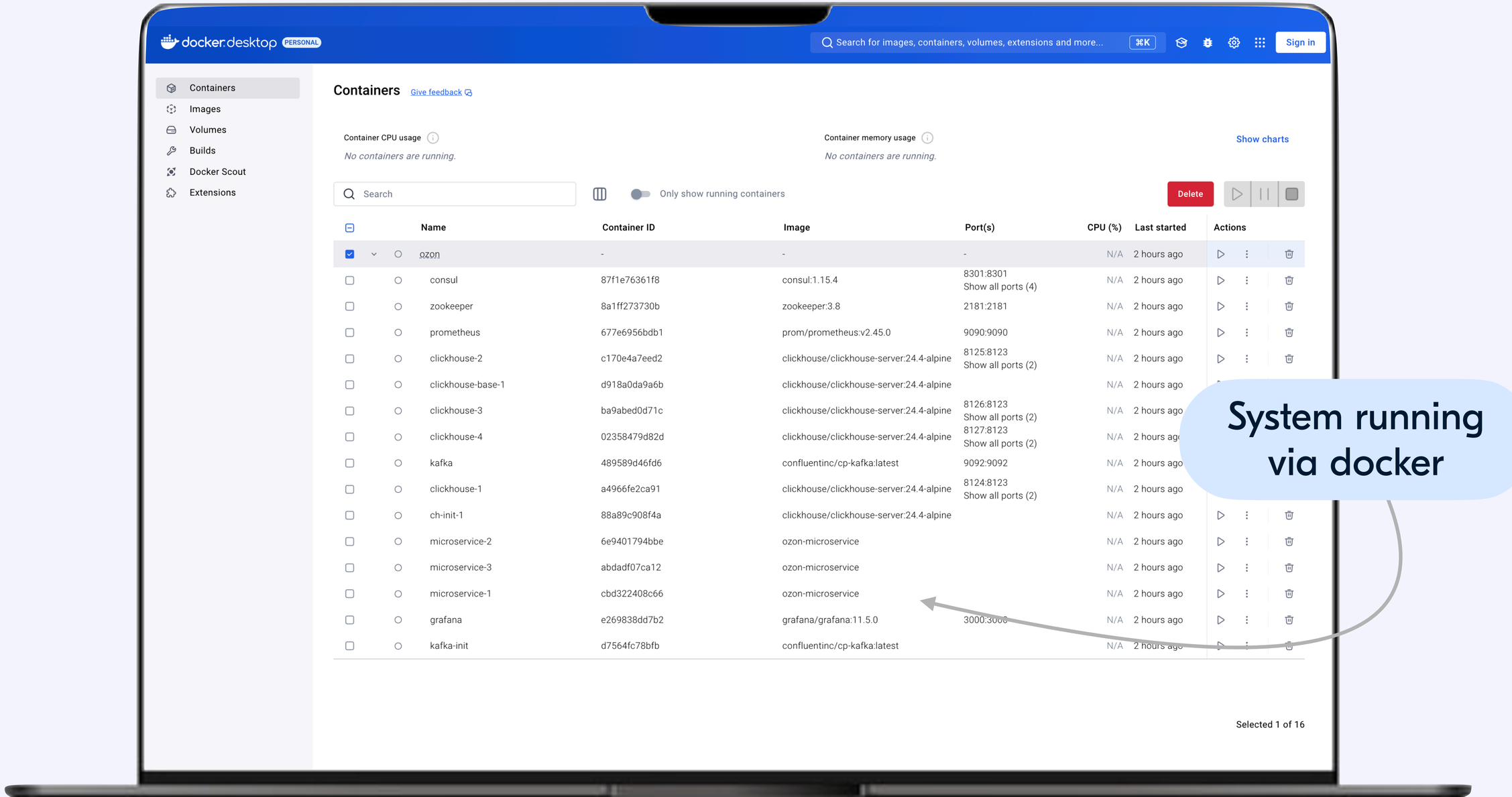
Architecture Overview

Synthetic data generation



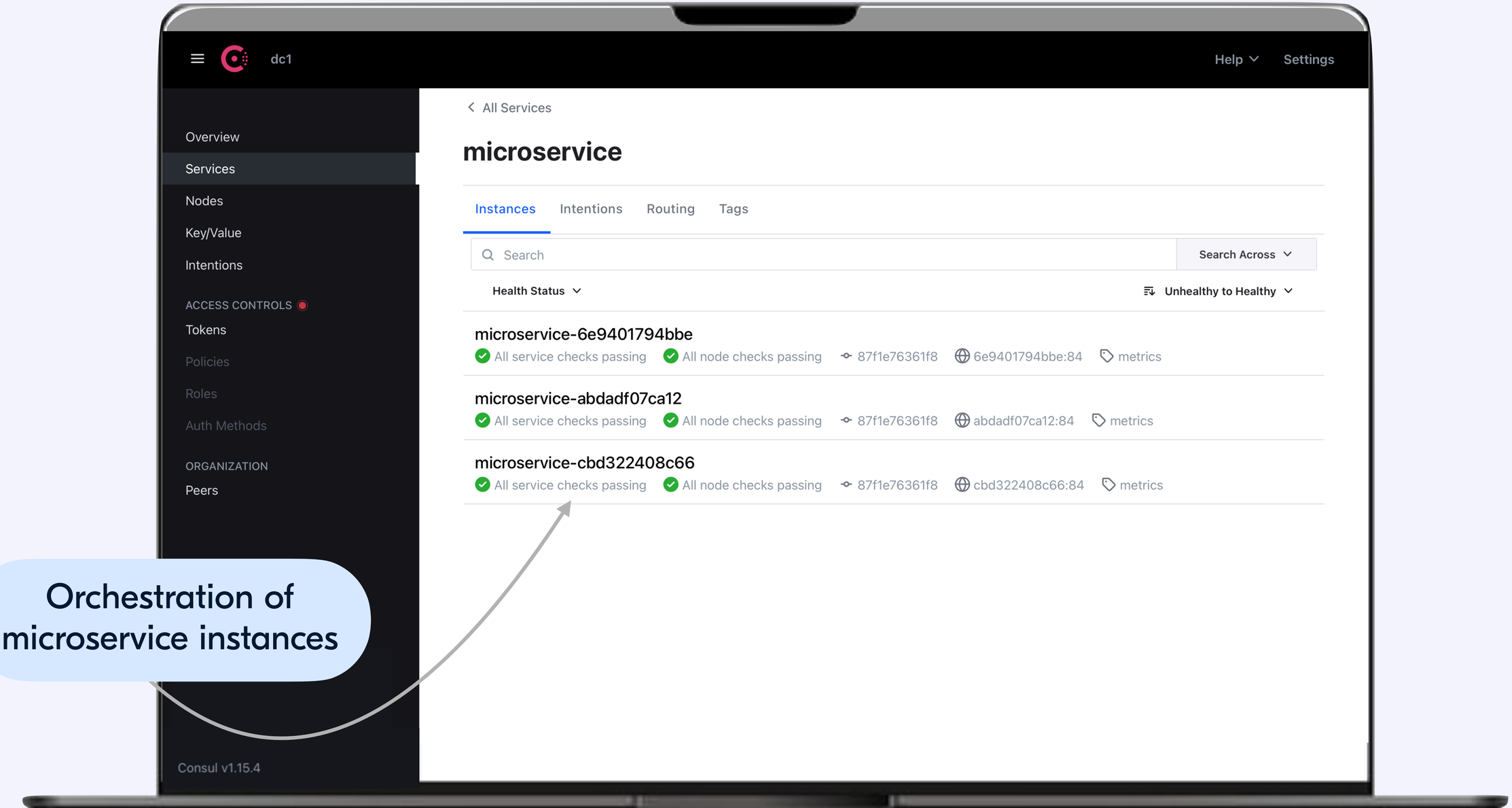
id	Unique identifier of the item
ts_ns	Event timestamp in nanoseconds
event	Type of event: "SAVE", "UPDATE", "ERROR"
attempt_id	Identifier for this upload attempt
is_created	Boolean flag
origin	Source of the operation
up_ts	If is_created=1, a ns ts when creation occurred; or 0
country	Production country code
data	JSON: info, company ID and list of media files

Architecture Overview



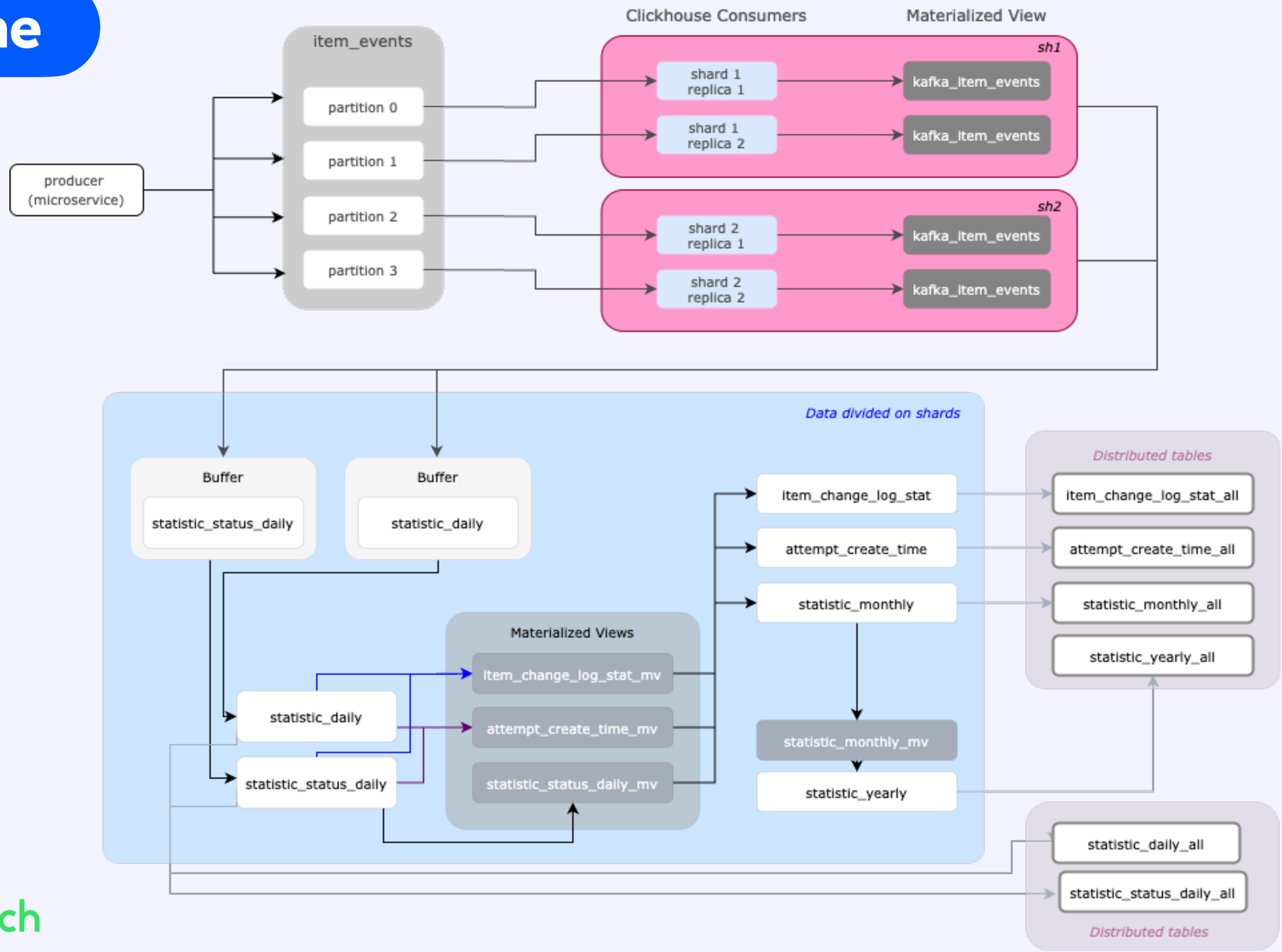
System running via docker

Architecture Overview



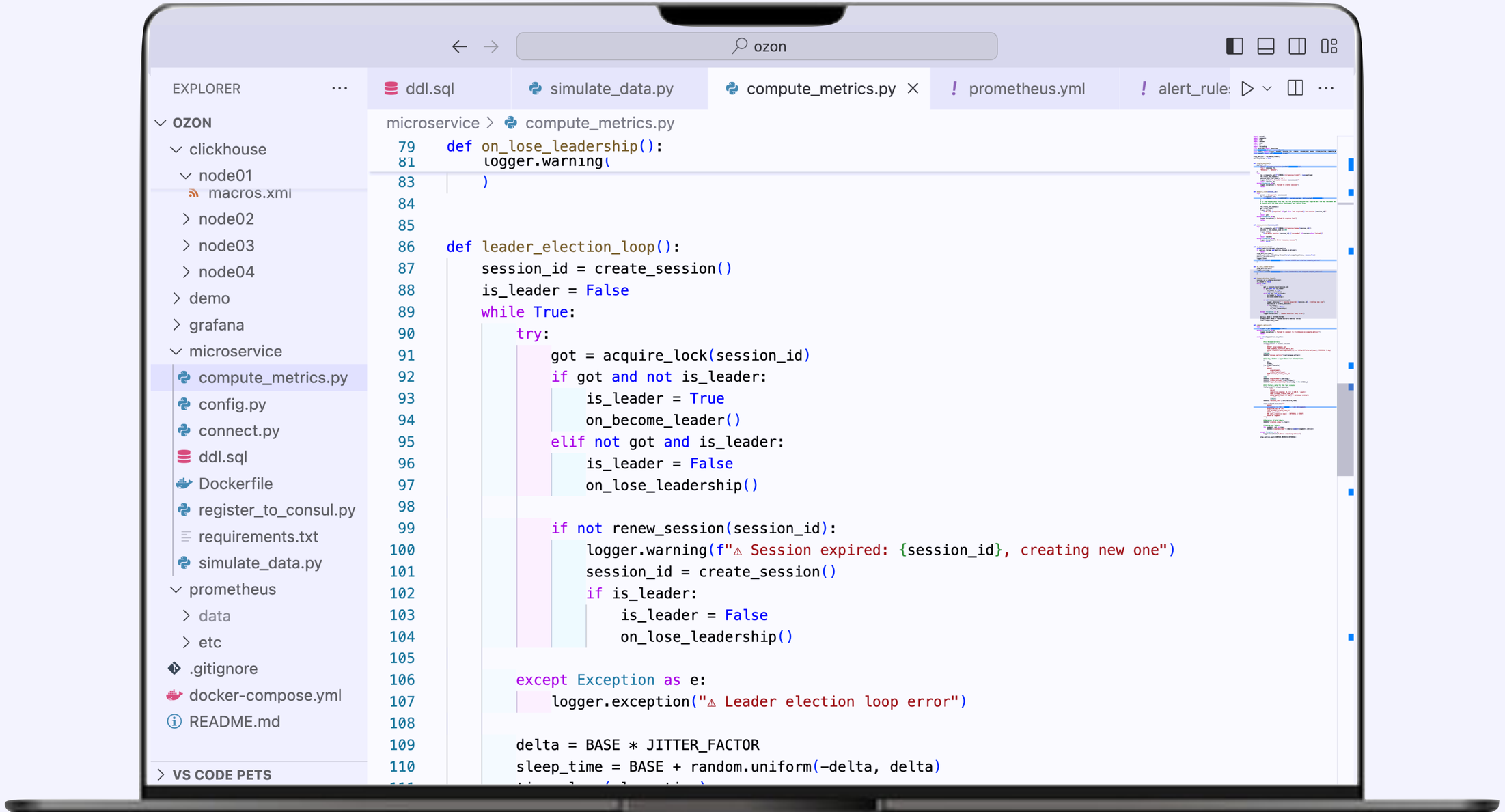
Orchestration of microservice instances

Pipeline

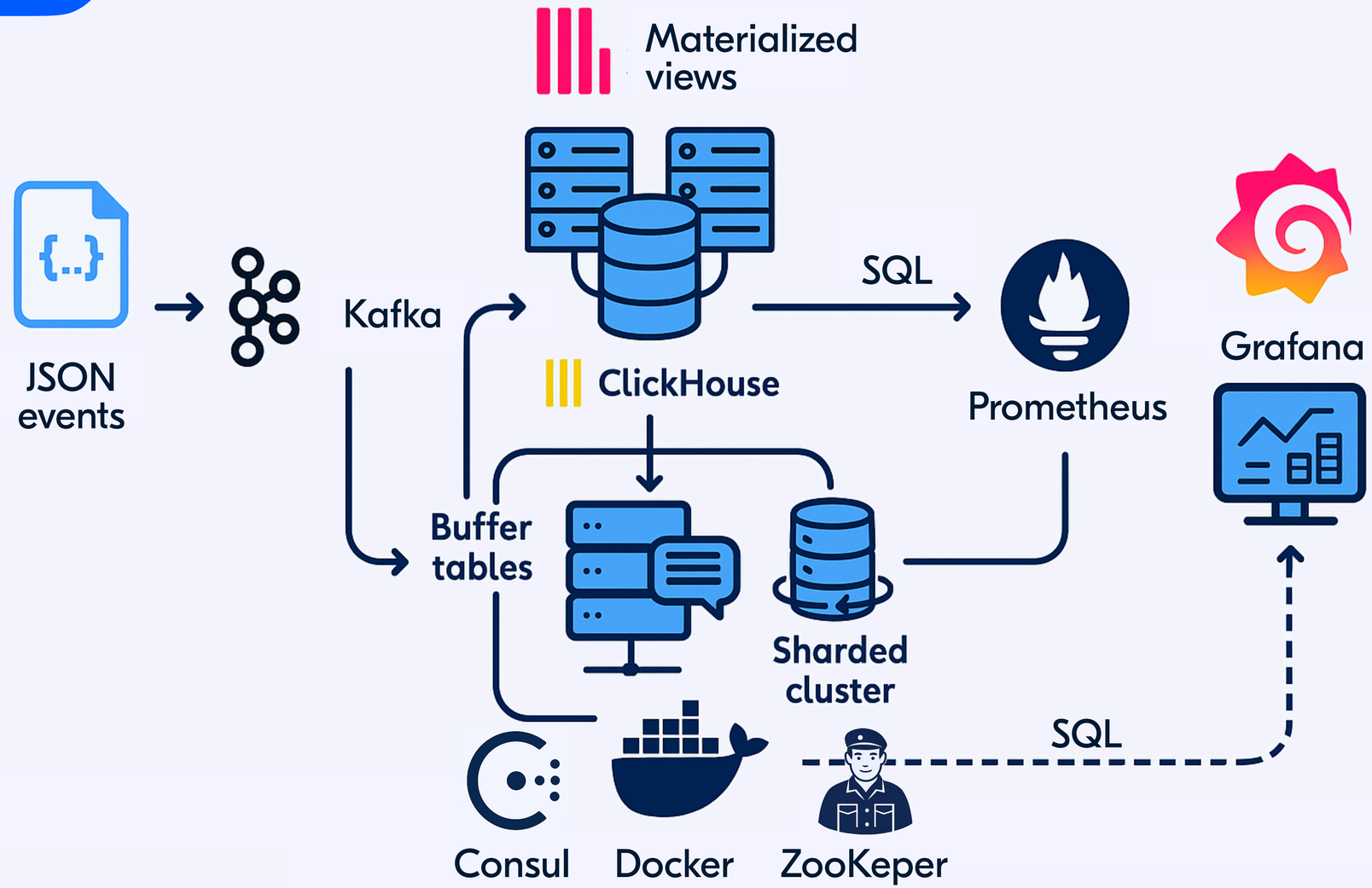


OZONtech

Pipeline


ozon
EXPLORER
OZON
clickhouse
node01
macros.xml
node02
node03
node04
demo
grafana
microservice
compute_metrics.py
config.py
connect.py
ddl.sql
Dockerfile
register_to_consul.py
requirements.txt
simulate_data.py
prometheus
data
etc
.gitignore
docker-compose.yml
README.md
VS CODE PETS
ddl.sql
simulate_data.py
compute_metrics.py
prometheus.yml
alert_rule
microservice > compute_metrics.py
def on_lose_leadership():
logger.warning(
)
def leader_election_loop():
session_id = create_session()
is_leader = False
while True:
try:
got = acquire_lock(session_id)
if got and not is_leader:
is_leader = True
on_become_leader()
elif not got and is_leader:
is_leader = False
on_lose_leadership()
if not renew_session(session_id):
logger.warning(f"⚠ Session expired: {session_id}, creating new one")
session_id = create_session()
if is_leader:
is_leader = False
on_lose_leadership()
except Exception as e:
logger.exception("⚠ Leader election loop error")
delta = BASE * JITTER_FACTOR
sleep_time = BASE + random.uniform(-delta, delta)

Pipeline


Materialized views
JSON events
Kafka
ClickHouse
SQL
Prometheus
Grafana
Buffer tables
Sharded cluster
SQL
Consul
Docker
ZooKeper

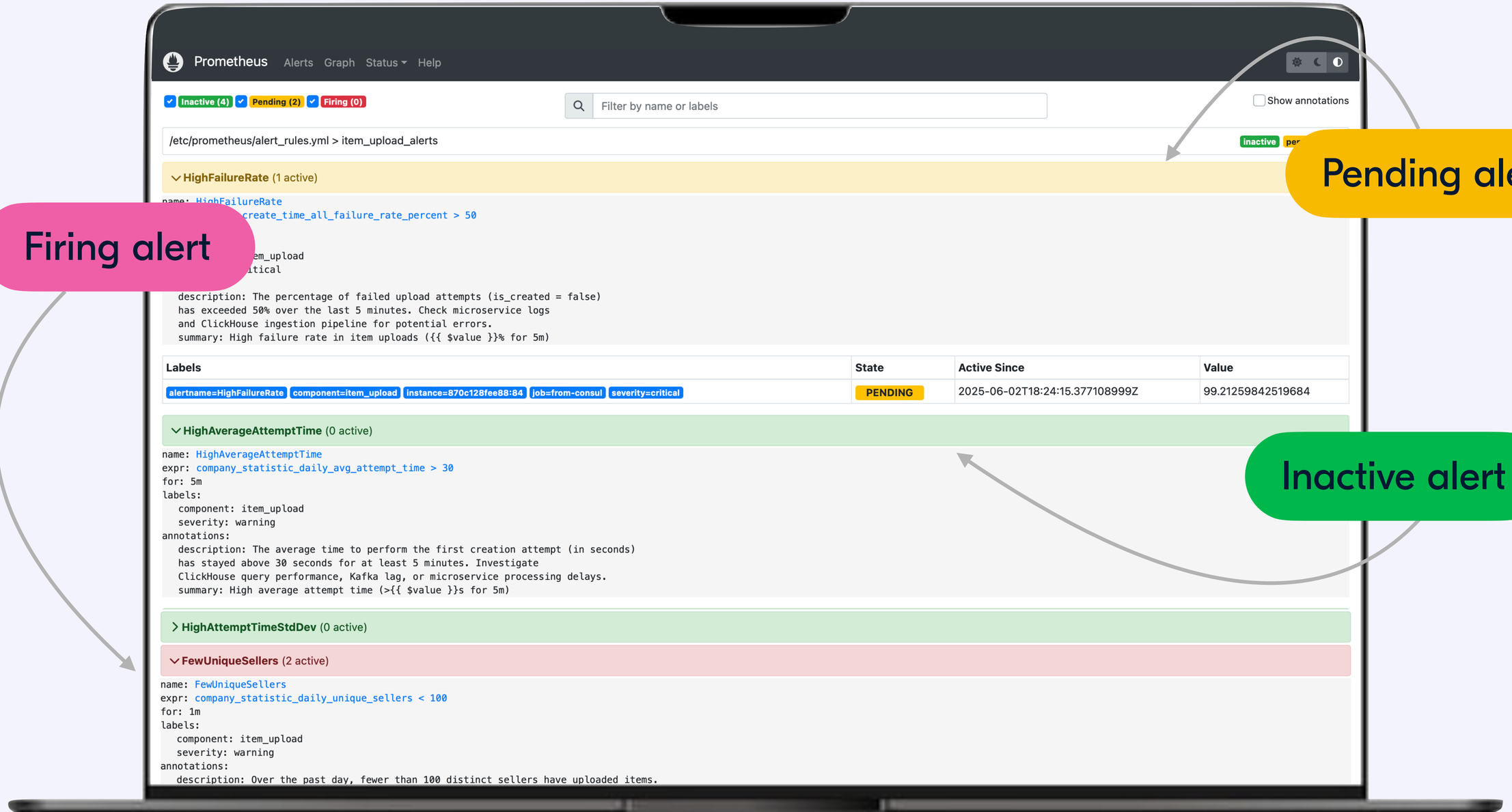
Results & Testing

OZONtech

Vizualization

Vizualization

Alerts



Firing alert

Pending alert

Inactive alert

Tests

Reference:

Ozon Production Cluster

- 2 replicas (non-sharded)
- 8 vCPU, 48 GB RAM, 700 GB SSD
- 1,000 RPS peak

Test setup:

Local Sharded Cluster

- 2 shards × 2 replicas
- Load balanced across shards
- No increase in latency

- **Linear scalability** by adding shards;
- Ready to handle higher loads

Issues

Data skew

Effect of random distribution in Python

Synthetic traffic

Lacks specific patterns

Limited testing

Pipeline with scripted chaos tests required

Cluster orchestration

Deployment with Kubernetes may be an improvement

References

[1] ACID-transactions in PostgreSQL. Accessed: 2025-05-10. 2025. url: https://blog.ydb. tech/sharded-is-not-distributed

[2] Denis Sheahan Adrian Cockcroft. Benchmarking Cassandra Scalability on AWS. url: https://netflixtechblog.com/benchmarking-cassandra-scalability

[3] T. Akidau et al. Streaming Systems: The What, Where, When, and How of Large-Scale Data Processing. O'Reilly Media, 2018.

[4] Apache Cassandra Documentation. Accessed: 2025-02-03. 2025. url: https://cassandra. apache.org/doc/latest/.

[5] Apache Druid Documentation. Accessed: 2025-02-03. 2025. url: http : / / druid . apache . org/docs/latest/.

[6] J. Carpenter and E. Hewitt. Cassandra: The Definitive Guide: Distributed Data at Scale. 2nd. O'Reilly Media, 2016.

[7] Citus. Accessed: 2025-20-05. 2025. url: https://demirhuseyinn-94.medium.com/scaling-horizontally-on-postgresql-cituss-impact-on-database-architecture

[8] ClickHouse Documentation. Accessed: 2025-02-03. 2025. url: https://clickhouse.com/docs/en/.

[9] Horizontal Scaling PostgreSQL. Accessed: 2025-28-05. 2025. url: https://www.cockroachlabs. com/blog/limitations-of-postgres/.

[10] Multiversion concurrency control. Accessed: 2025-18-05. 2025. url: https://en.wikipedia. org/wiki/PostgreSQL#Multiversion_concurrency_control_(MVCC).

[11] Postgre XL, github. Accessed: 2025-18-05. 2025. url: https : //github.com/pharosnet/postgres-xl.

[12] Round-Robin scheduling algorithm. Accessed: 2025-07-10. 2025. url: https://en.wikipedia. org/wiki/Round-robin_scheduling.

[13] Write-ahead logic. Accessed: 2025-18-05. 2025. url: https : //en.wikipedia.org/wiki/PostgreSQL#Storage_and_replication.