Web-scale Data Management

Group 2 - Pragmatic Project

Frank Bredius

Jan-Mark Dannenberg

Pepijn te Marvelde

Bailey Tjiong

TUDelft

Technologies used

docker

redis

HELM

Flask
web development,
one drop at a time

TUDelft

Technologies used

docker

redis

PostgreSQL

Flask SQLAlchemy

gunicorn

HELM

Flask
web development,
one drop at a time

TUDelft

Technologies used

docker

redis

PostgreSQL

Flask SQLAlchemy

gunicorn

RabbitMQ

HELM

Flask
web development,
one drop at a time

Grafana

Technologies used

docker

Quart

redis

Flask SQLAlchemy

PostgreSQL

gunicorn

RabbitMQ™

aio-pika

HELM

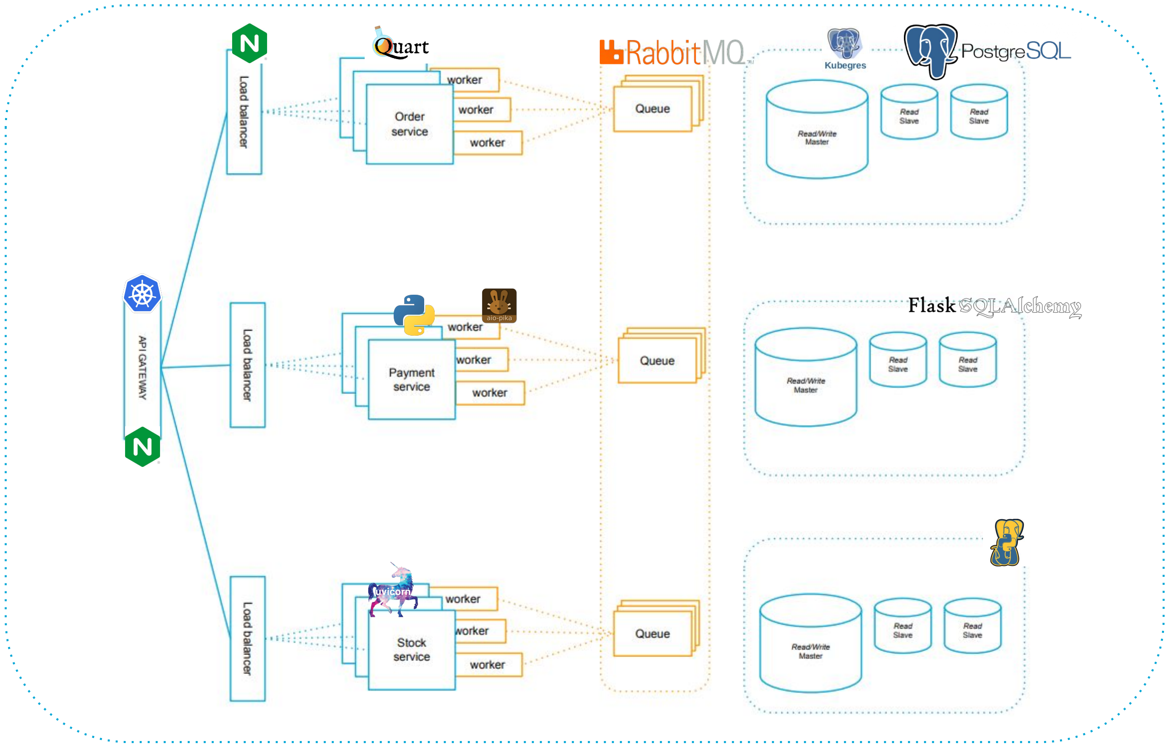
uvicorn

Flask
web development,
one drop at a time

Grafana

TUDelft

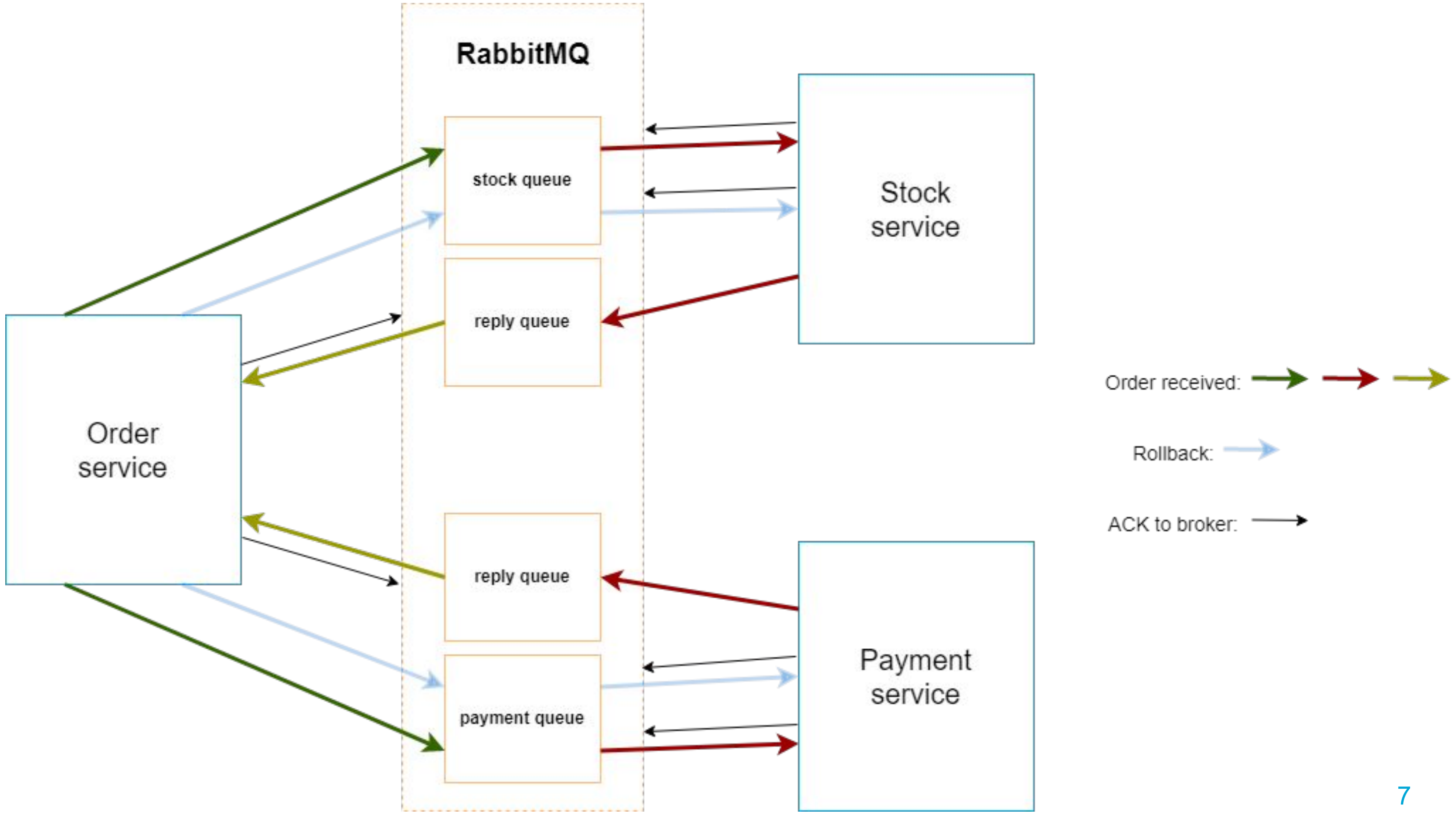
Architecture Diagram



Deployment

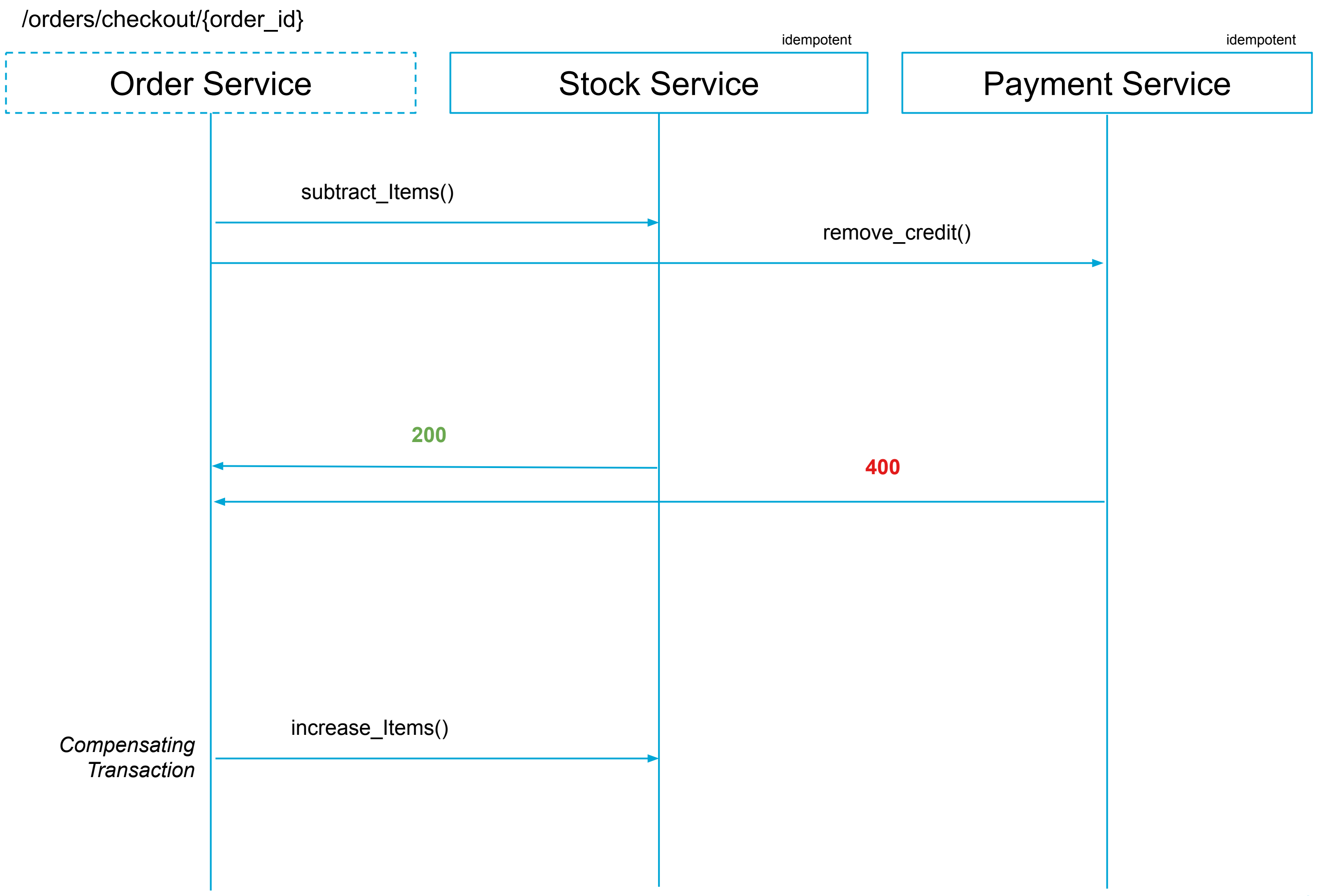
Monitoring & Observing

Messaging


RabbitMQ
stock queue
reply queue
reply queue
payment queue
Order service
Stock service
Payment service
Order received:
Rollback:
ACK to broker:

Orchestration based SAGA

Method of transaction execution



SAGA Guarantees

ACID

Atomicity: ✅

Consistency: ✅

Isolation: ❌

Durability: ✅

BASE

Base **A**vailability: ✅

Soft state: ✅

Eventual Consistency: ✅

Source: https://www.infoq.com/articles/saga-orchestration-outbox/

Consistency

Eventual consistency

- Eventual consistency using SAGA [1], see SAGA guarantees
- BASE guarantees: Basic Availability, Soft State, Eventual Consistency [2]

PostgreSQL consistency

- Enforcing correct data using database constraints [3]
- “READ COMMITTED” transaction level [4]
    - Read sees snapshot of database, dirty read not possible
    - Write waits for concurrent transactions on target rows

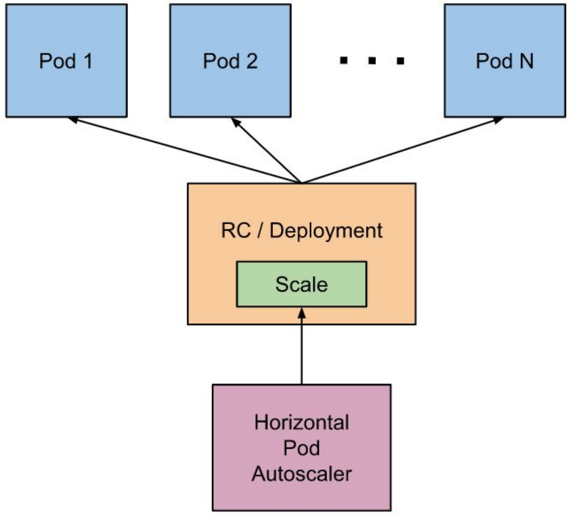
Correctness of transactions

- Correctness through code checks
- Distributed Transactions possible through acknowledgements using RabbitMQ

TUDelft

Scalability

- Kubernetes Horizontal Pod Autoscaling
- Load balancing using Ingress
- Asynchronous messaging
- Kubegres for PostgreSQL clusters + replicas
- **No sharding and multiple masters**



Fault Tolerance

Database Fault Tolerance

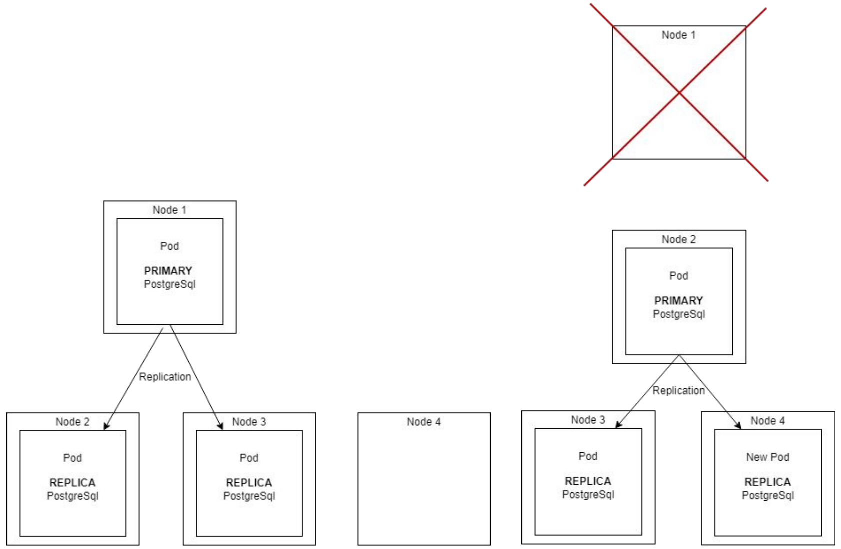
- Pod-anti affinity [5]
- Failover [5]

Microservice Fault Tolerance

- Stateless and replicated
- Self-healing [6]

Communication Fault Tolerance

- Acknowledgements [7]
- Quorum Queues [8]
- RabbitMQ Cluster [9]



Results setup

- **Deployed on GKE**
  - 4 nodes
  - 16 vCPUs
  - 64GB RAM

- **Load testing using provided Locust tests**
  - In cluster
  - 1 master, 5 workers
  - Spawning 1 user/sec

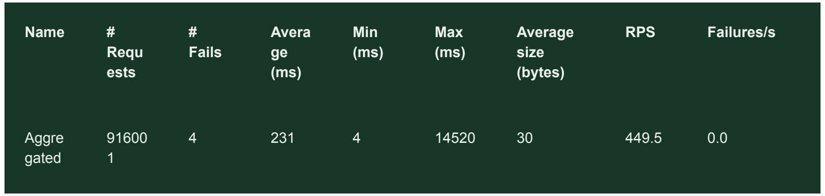
Google Kubernetes Engine

LOCUST

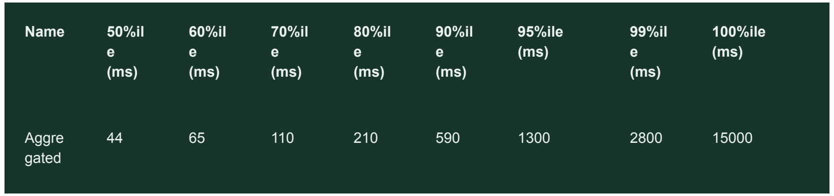
TU Delft

Results

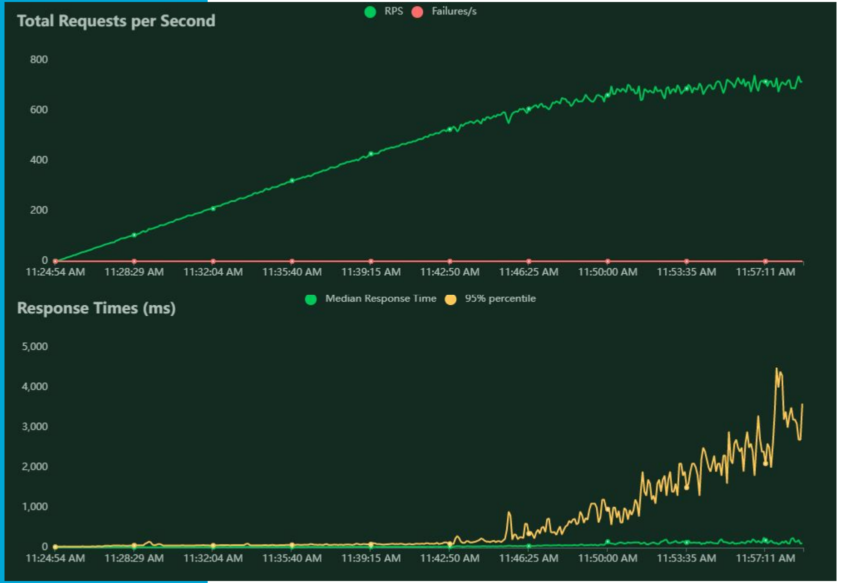
- **Throughput:** +/- 600 RPS
- **Response time** (at 600 RPS)
  - **Median:** 50ms
  - **95%ile:** 500ms

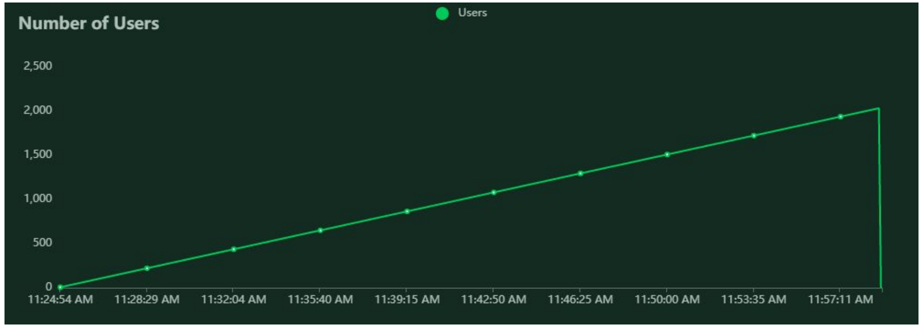


Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
Aggregated	916001	4	231	4	14520	30	449.5	0.0



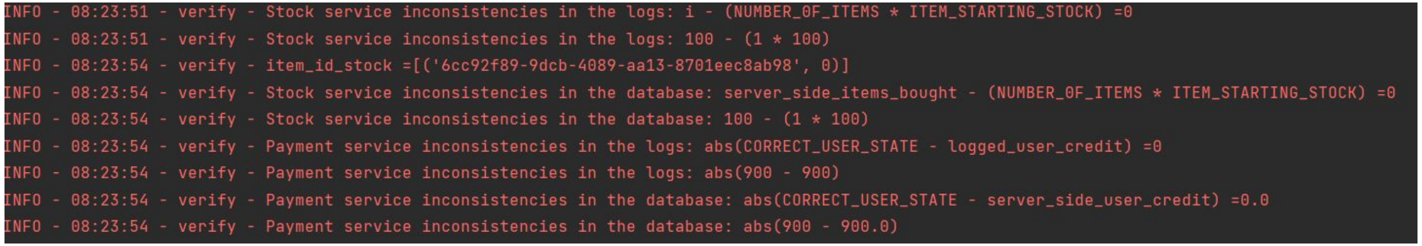
Name	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)	100%ile (ms)
Aggregated	44	65	110	210	590	1300	2800	15000





TUDelft

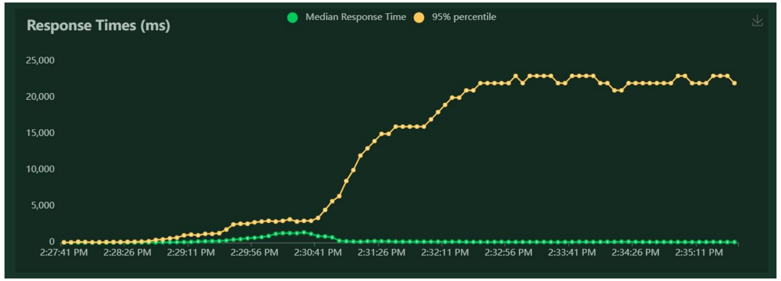
Results consistency

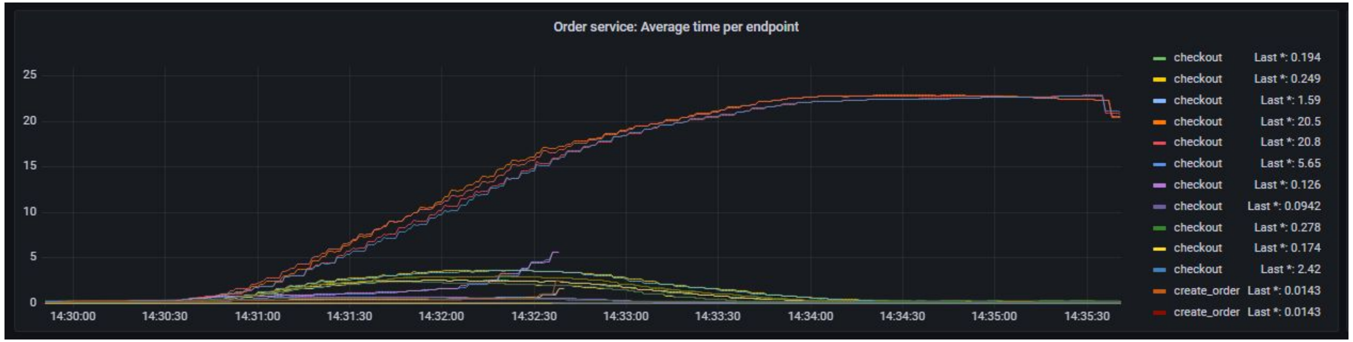


```
INFO - 08:23:51 - verify - Stock service inconsistencies in the logs: i - (NUMBER_OF_ITEMS * ITEM_STARTING_STOCK) =0
INFO - 08:23:51 - verify - Stock service inconsistencies in the logs: 100 - (1 * 100)
INFO - 08:23:54 - verify - item_id_stock =[('6cc92f89-9dcb-4089-aa13-8701eec8ab98', 0)]
INFO - 08:23:54 - verify - Stock service inconsistencies in the database: server_side_items_bought - (NUMBER_OF_ITEMS * ITEM_STARTING_STOCK) =0
INFO - 08:23:54 - verify - Stock service inconsistencies in the database: 100 - (1 * 100)
INFO - 08:23:54 - verify - Payment service inconsistencies in the logs: abs(CORRECT_USER_STATE - logged_user_credit) =0
INFO - 08:23:54 - verify - Payment service inconsistencies in the logs: abs(900 - 900)
INFO - 08:23:54 - verify - Payment service inconsistencies in the database: abs(CORRECT_USER_STATE - server_side_user_credit) =0.0
INFO - 08:23:54 - verify - Payment service inconsistencies in the database: abs(900 - 900.0)
```

Limitations

- *Kubegres* does not allow sharding or multi-master replication.
- Load tests sometimes show poor(er) latency.
- Fault-tolerance not tested





References

[1] https://medium.com/trendyol-tech/saga-pattern-briefly-5b6cf22dfabc

[2] https://www.scylladb.com/glossary/database-consistency/

[3] https://stackoverflow.com/questions/14225998/flask-sqlalchemy-column-constraint-for-positive-integer

[4] https://www.postgresql.org/docs/current/transaction-iso.html#XACT-READ-COMMITTED

[5] https://www.kubegres.io/doc/replication-and-failover.html

[6] https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

[7] https://www.rabbitmq.com/confirms.html

[8] https://www.rabbitmq.com/quorum-queues.html#usage

[9] https://www.rabbitmq.com/kubernetes/operator/operator-overview.html

[10] https://www.linkedin.com/pulse/multi-master-replication-relational-databases-scaling-ran-bechor/

TUDelft