IN4331 - Group 2

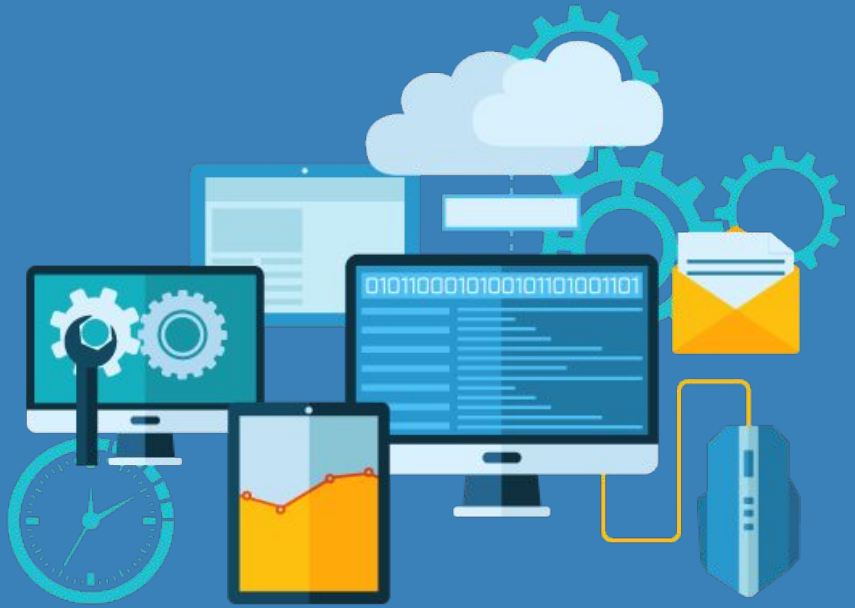
A reactive solution using Celery/RabbitMQ

…to serve and protect data.

Enrique Barba Roque, Philippe de Bekker, Arjan Hasami, Jokūbas de Kort, Simcha Vos

TU Delft

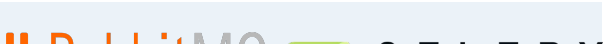
Our System



Databases & Logging

- For the main services: MongoDB
  - Easy to use, highly mature and popular.
  - Horizontal scaling through sharding.
  - Atomic, by checking on the database server.
  -
- Logging and recovery: Redis
  - Flexible data structures, highly mature and popular.
  - In-memory, so extremely fast.
  - Logs the steps in the saga.
  - Eventually scrapped

Task Management & Messaging

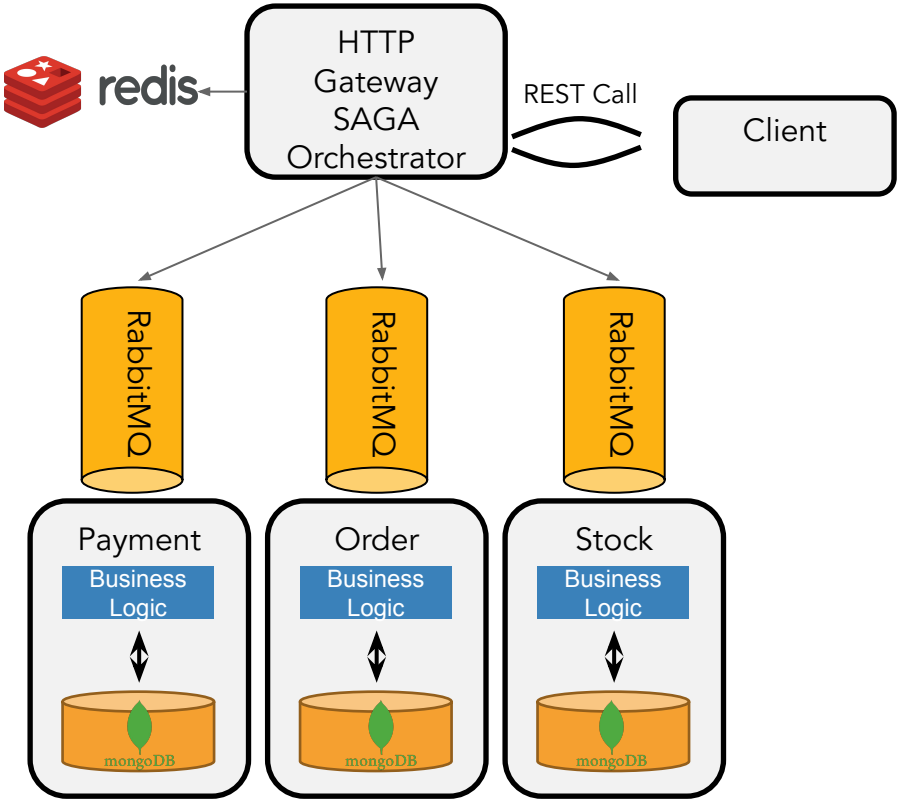


- Distributed task queue: Celery
  - Allows distributed task processing, ideal for large amounts of concurrent tasks.
  - Highly mature and popular; robust and flexible.

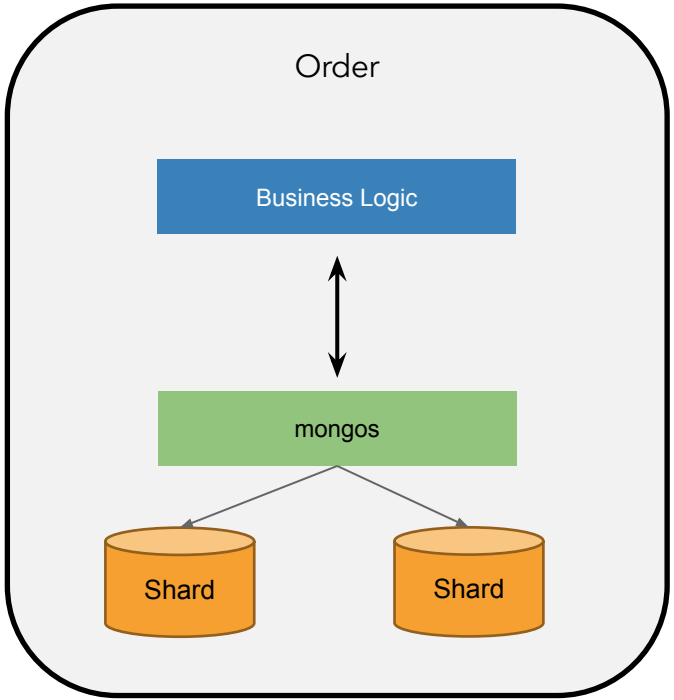
- Message broker: RabbitMQ
  - Implements Advanced Message Queuing Protocol; reliable message delivery. Highly mature and popular.
  - Celery workers grab from the queues.

Our Solution

Global overview

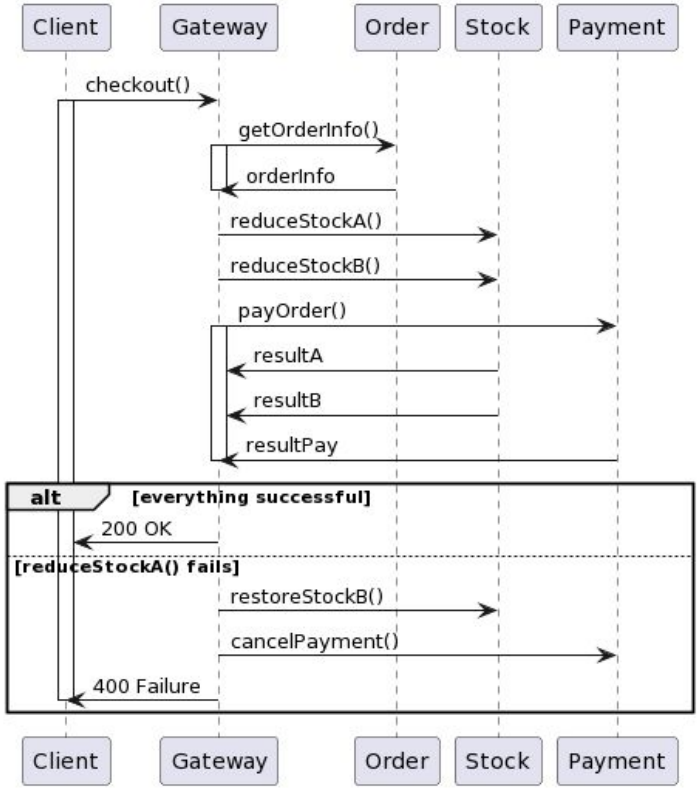


MongoDB Sharding



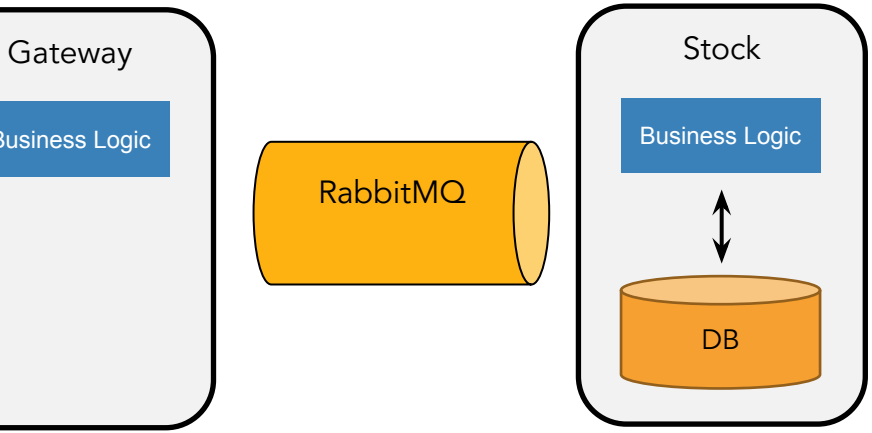
- MongoDB Sharding for DBscaling capabilities
- Operations by entity ID
- Partitioned by ID hashing
  - Equal distribution across shards

SAGA



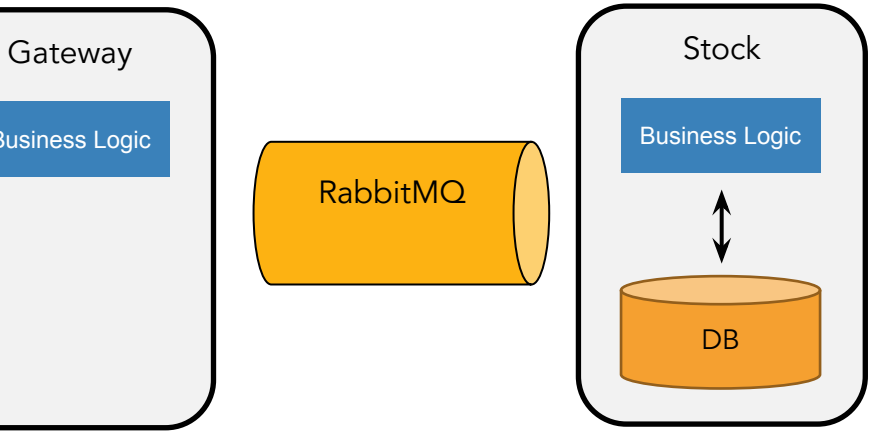
- Gateway acts as SAGA orchestrator.
  - Launch tasks to workers
  - Wait and track status
  - Compensate in case of failure

RabbitMQ for internal communication



- RabbitMQ for communication between services
  1. Order publishes a message on RabbitMQ's message queue;
  2. Stock consumes messages on the queue and processes;
  3. Stock publishes response on queue;
  4. Order service consumes messages on the queue and continues processing the request.

But what about REST?



- Classic microservices:
  - REST calls in the place of the queues makes all services synchronous
  - Result: horizontal scaling becomes less trivial due to synchronous exchanges
  - Services need to wait for a response and cannot handle other requests in the meantime

➔ Asynchronous communication makes scaling specific services easy

Celery

- If the API is different this would improve performance
- Celery used in code through `.delay()` and `.get()` functions

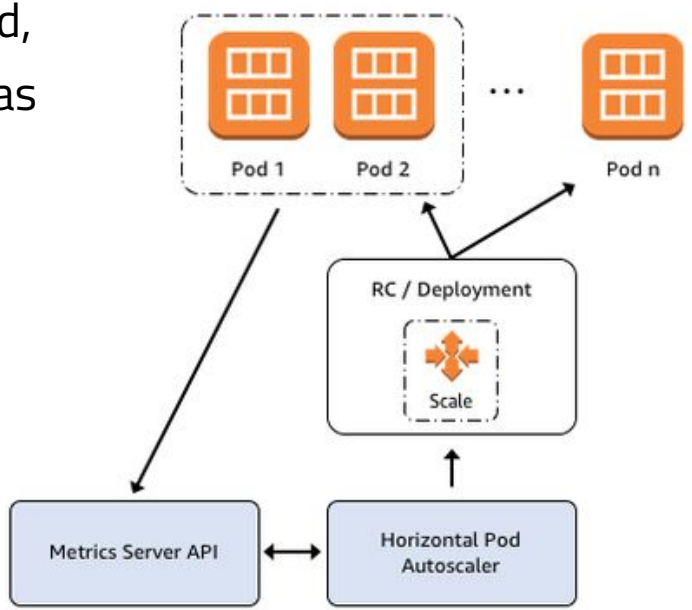
```
@router.get('/stock/find/{item_id}', status_code=status.HTTP_200_OK)
async def find_item(item_id: str):
    task = stock.find_item.delay(item_id)
    item = task.get()
    if item and not task.failed():
        return item
    else:
        raise HTTPException(status_code=404, detail="Item not found")
```

`.delay()` returns temporary object while queueing the execution of the task

`.get()` blocks until it gets the actual result of the task

Potential theoretical optimization: Load balancing

- Not effective in local environment with hardware limitations
- Dynamic scaling through **Horizontal Pod Autoscaling (HPA)**
- If CPU utilization reaches predefined threshold, scale between **min** and **max** number of replicas



Results

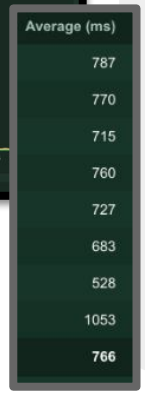
No AWS credits received; ran locally on Macbook Pro M2 Pro 16GB RAM

Throughput

Overview of total requests per second

maximum of 1100+ RPS

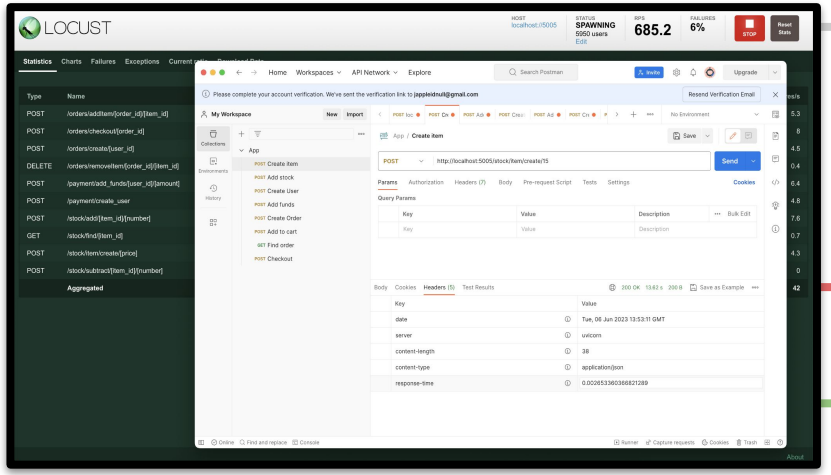
700 RPS on average



considering 2K-50K users

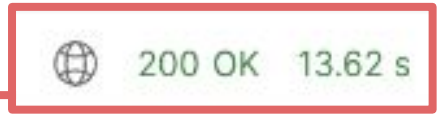
latency slope indicates bottleneck

Latency



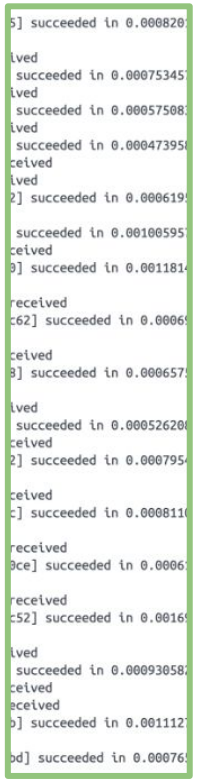
POST Request to create stock item *(using Postman)*

Number of users and requests per second
(General example scenario; consider extensive runtime)

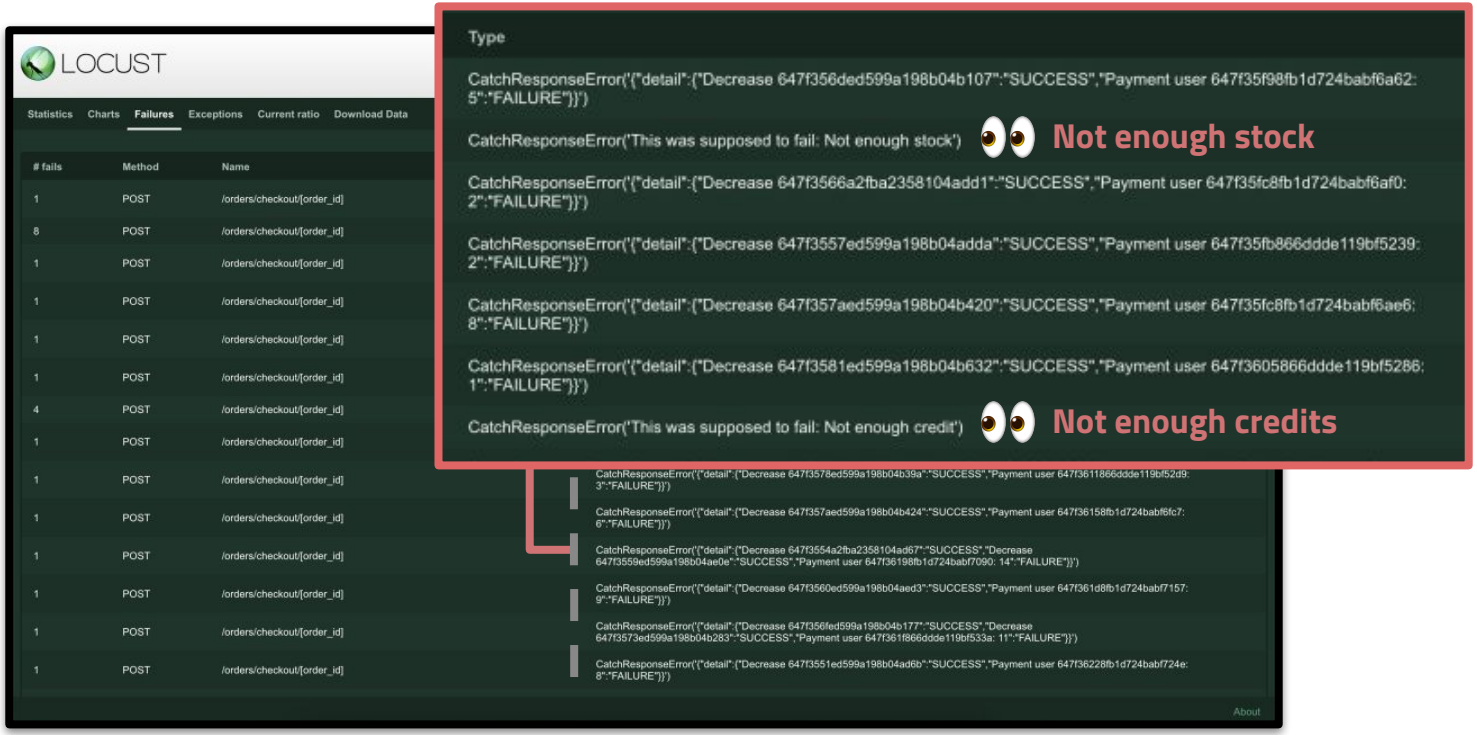


Bottleneck occurs in processing requests, gateway = suspect

Task completion is still fast, which is also confirmed by the Docker logs:



Consistency



Missing *processed-by-test-design* requests due to performance bottleneck causes (chain of) failures, however, failure =/= consistency error

What can be done better?

Considering a few more months of project time

Potential improvements

- Try different frameworks (e.g. Spring WebFlux)
- Make the API calls asynchronous
- Amazon Web Services
- Crash Recovery

Any questions?

And thanks for listening!