Building High Scale Backend Systems With Kafka

15 May 2023

Dineshkumar, Founder @ Scalescape, Inc

@devdineshkumar devdinu.github.io

About

@relyonmetrics, we're building a platform to self-serve opensource components

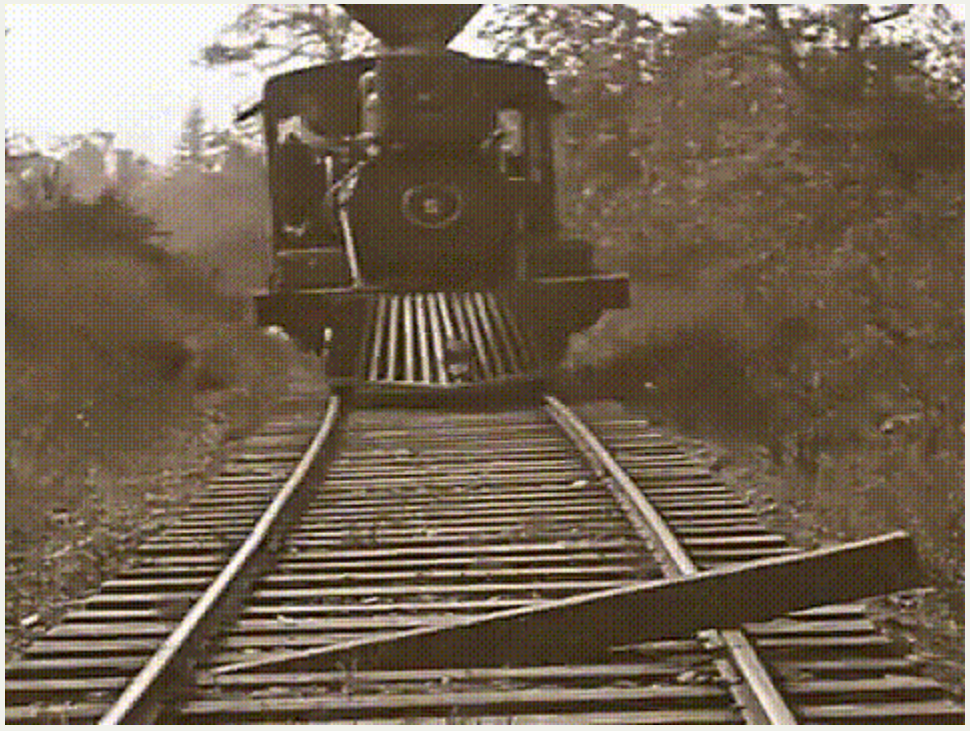
- Built backend for security platform @ Elastic, backend system for cmd.com
- Built event driven system for high scale @ Gojek
- Go, Kafka, distributed-systems enthusiast

Agenda

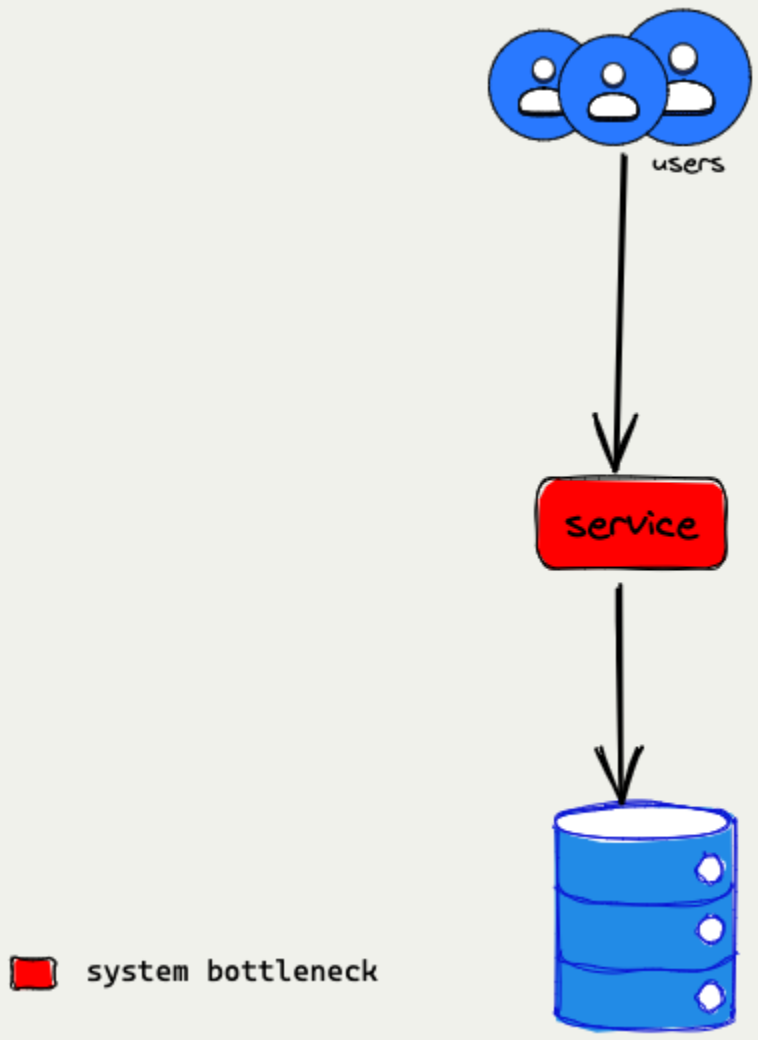
- Basics
- Monolith to Microservices
- System Level / Architecture
- Kafka / Event-Driven systems
- Scaling with Kafka
- Best Practices

Basics

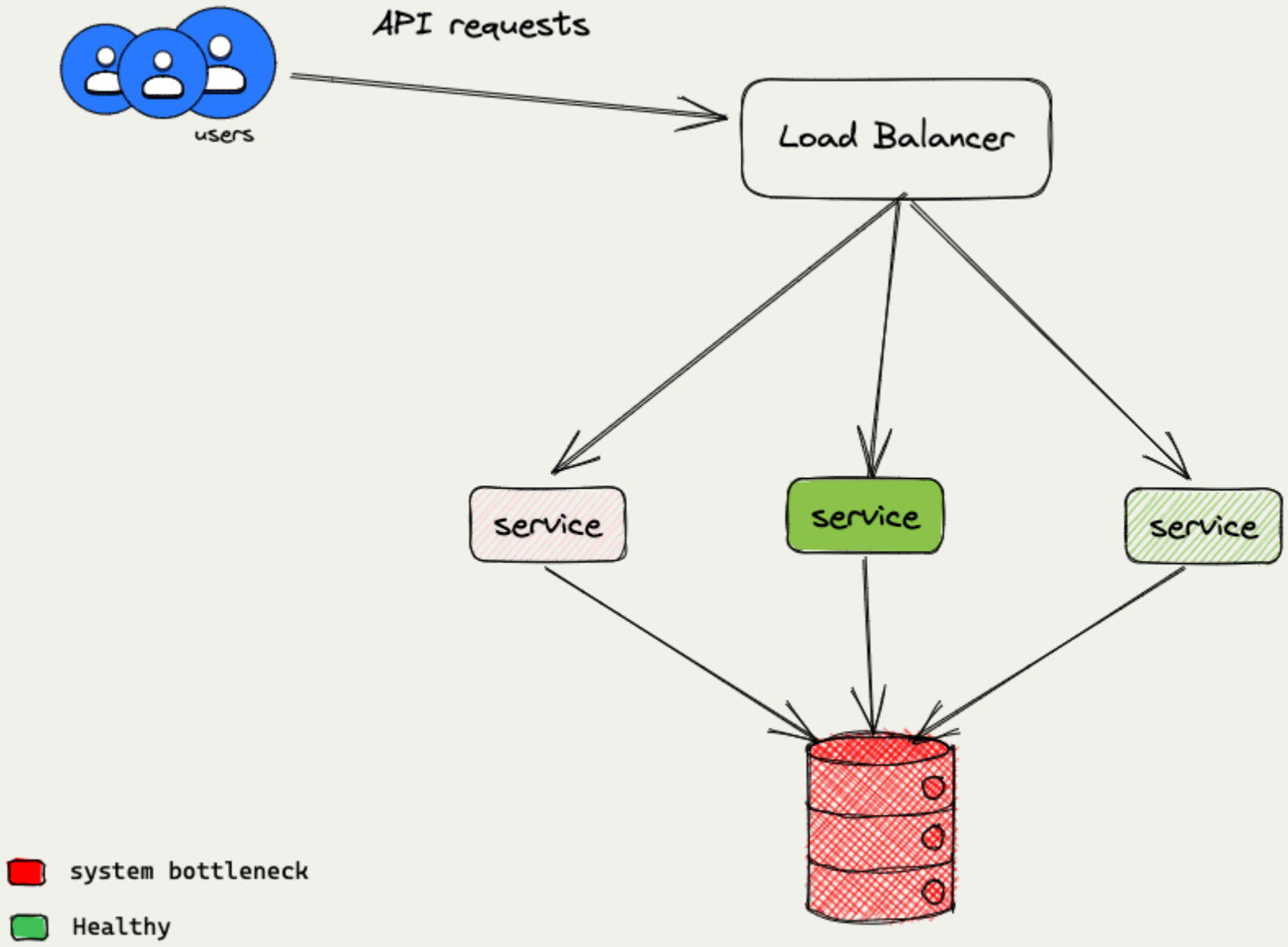
Assume everything will fail! 🔥



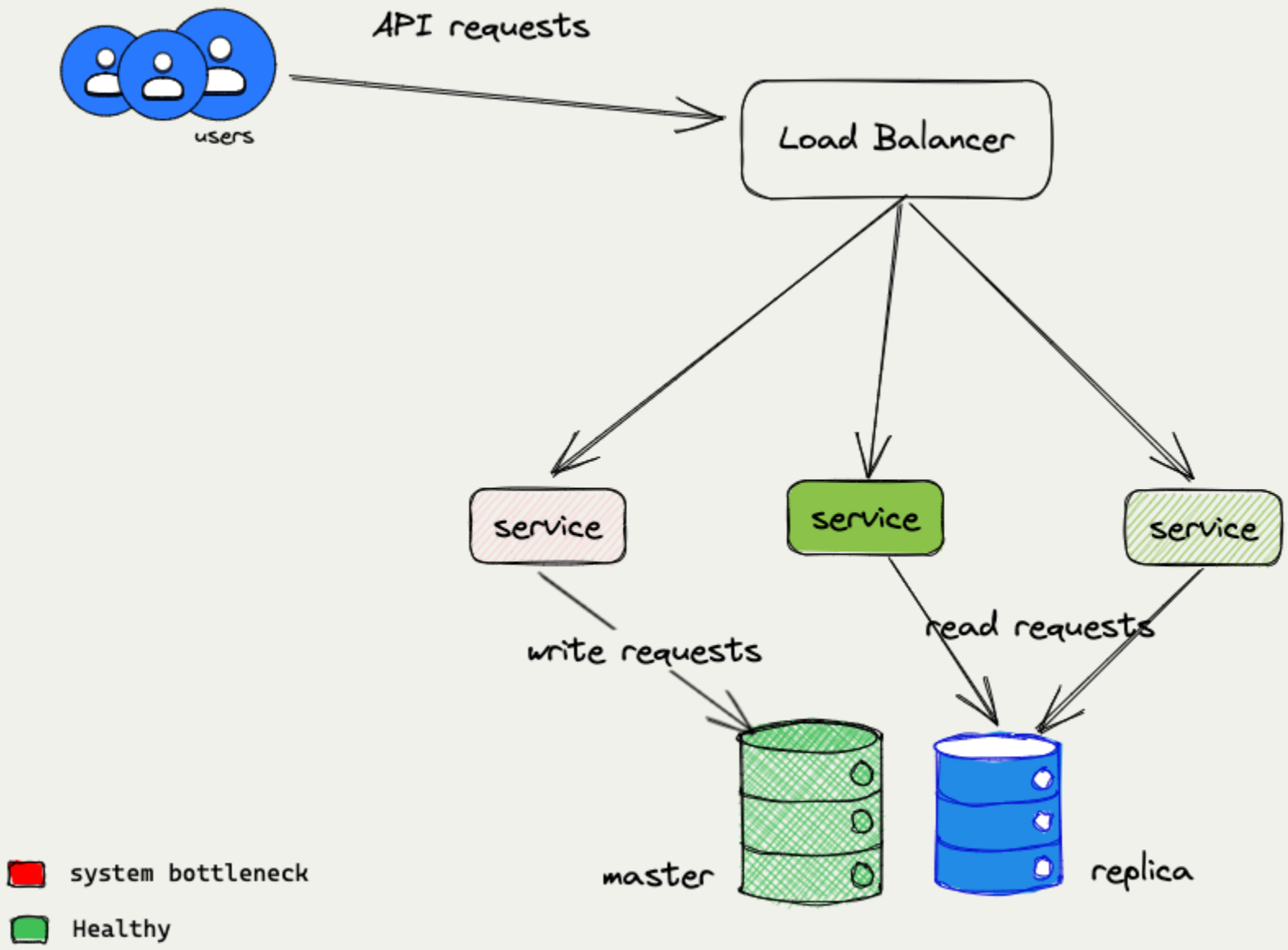
Monolith service with DB



Database becoming a bottleneck



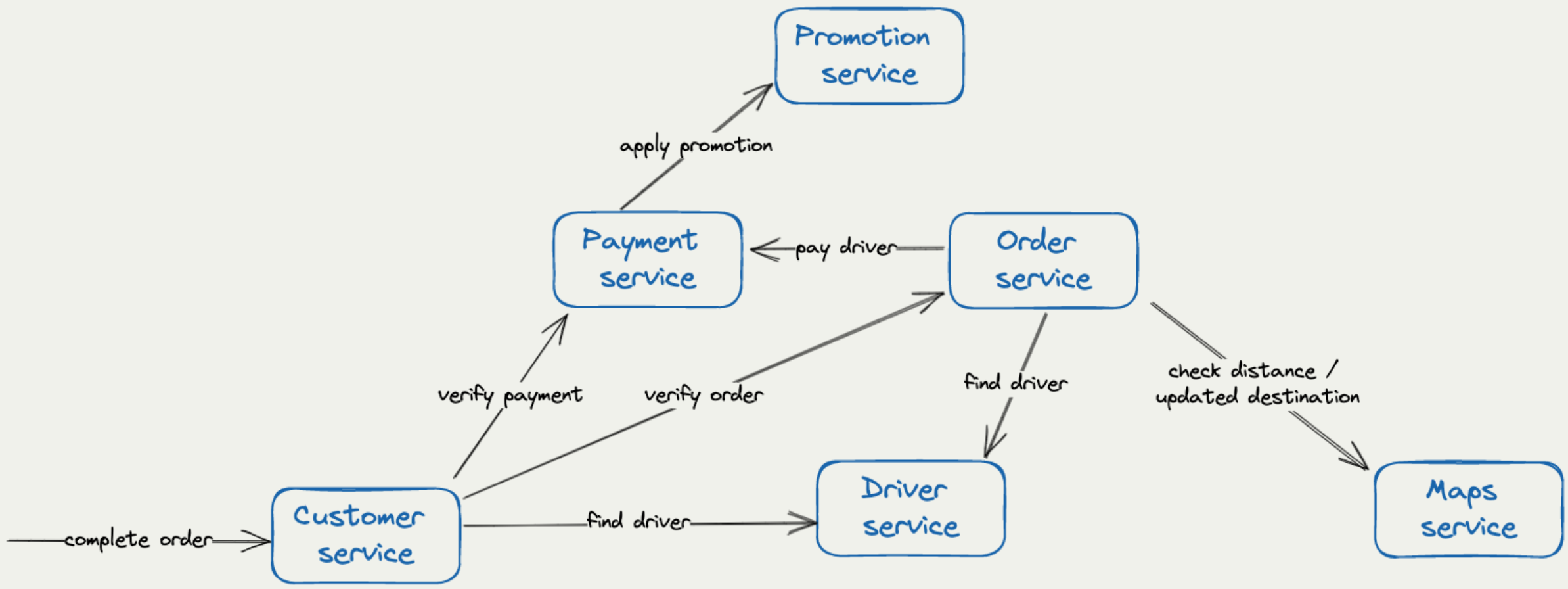
Simple service with DB replication



Variables / Factors

- Scaling independently
- Independent Deployments
- Blast Radius
- choice of stack
- build (CI/CD), deployment and development time
- ownership, team structure, ...

Growing microservices

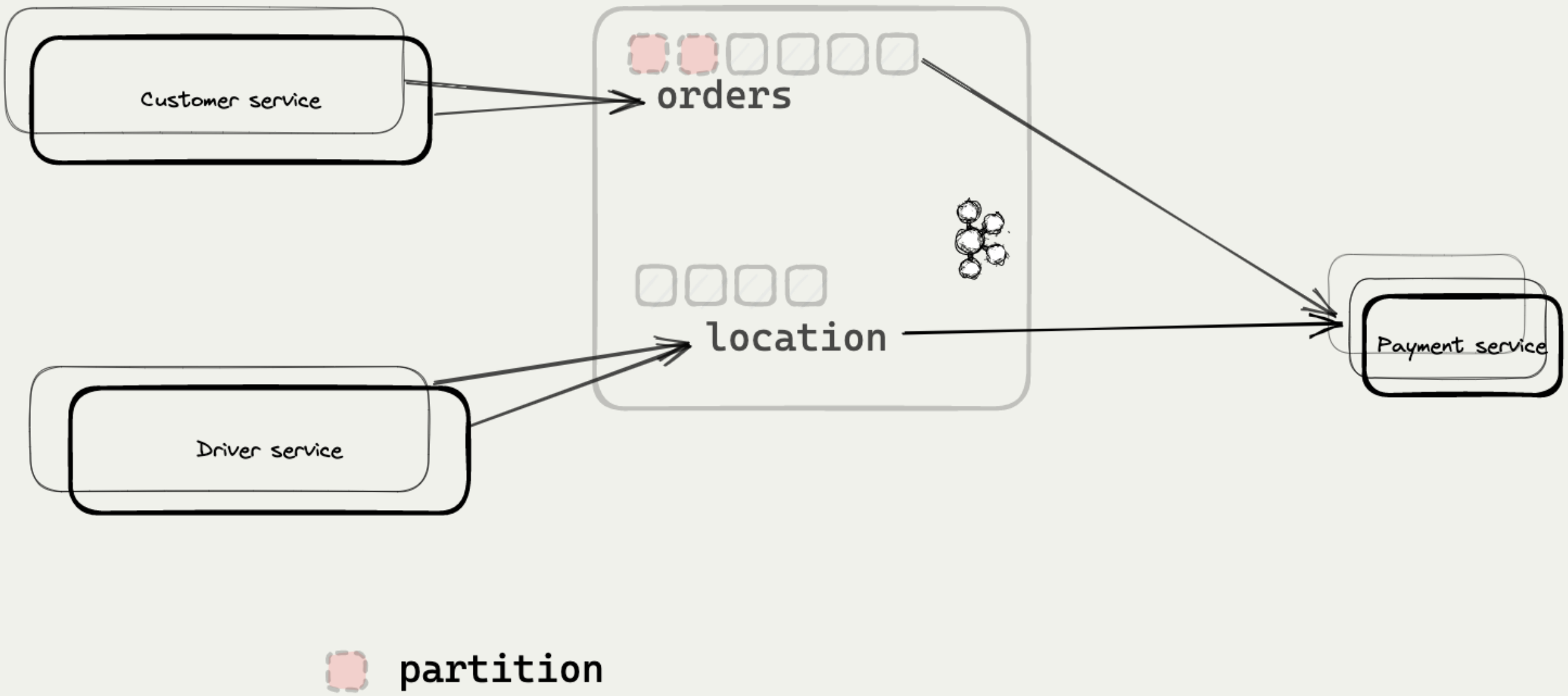


Kafka

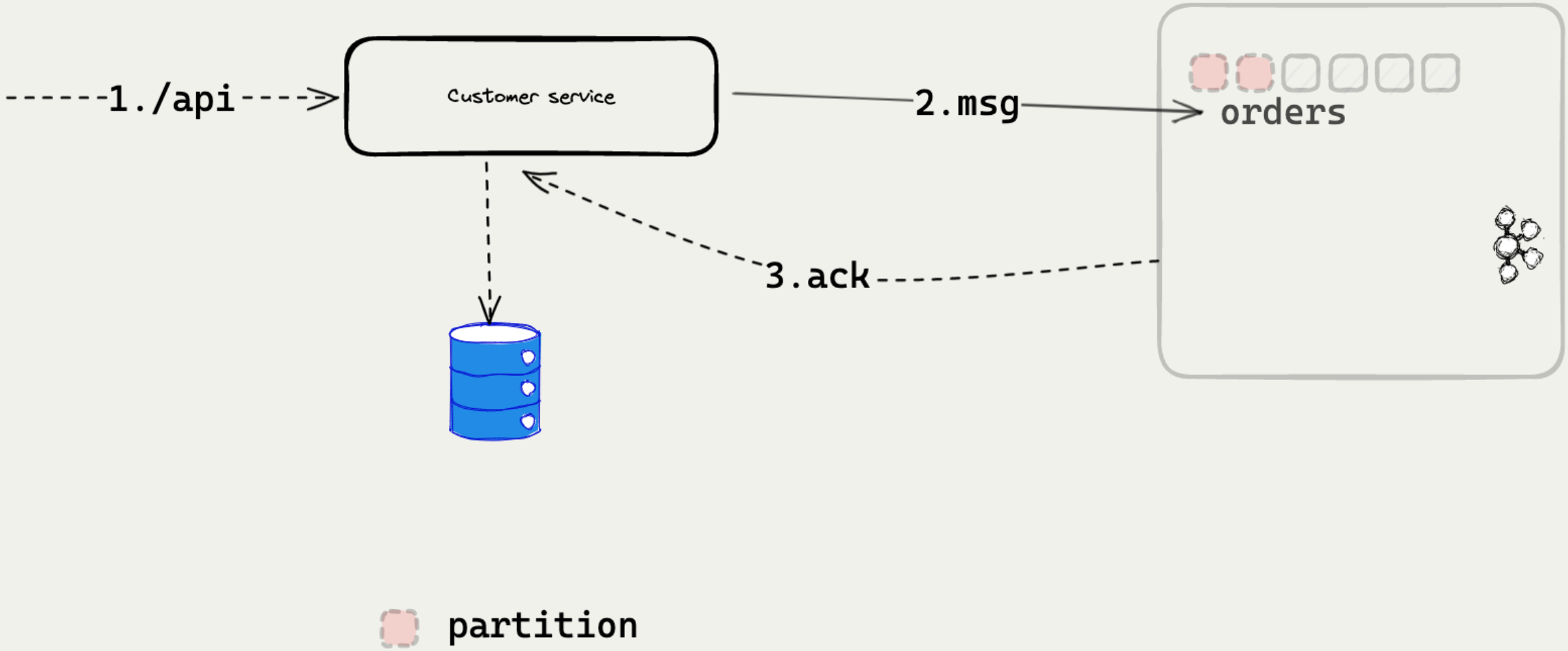
Event Driven Systems

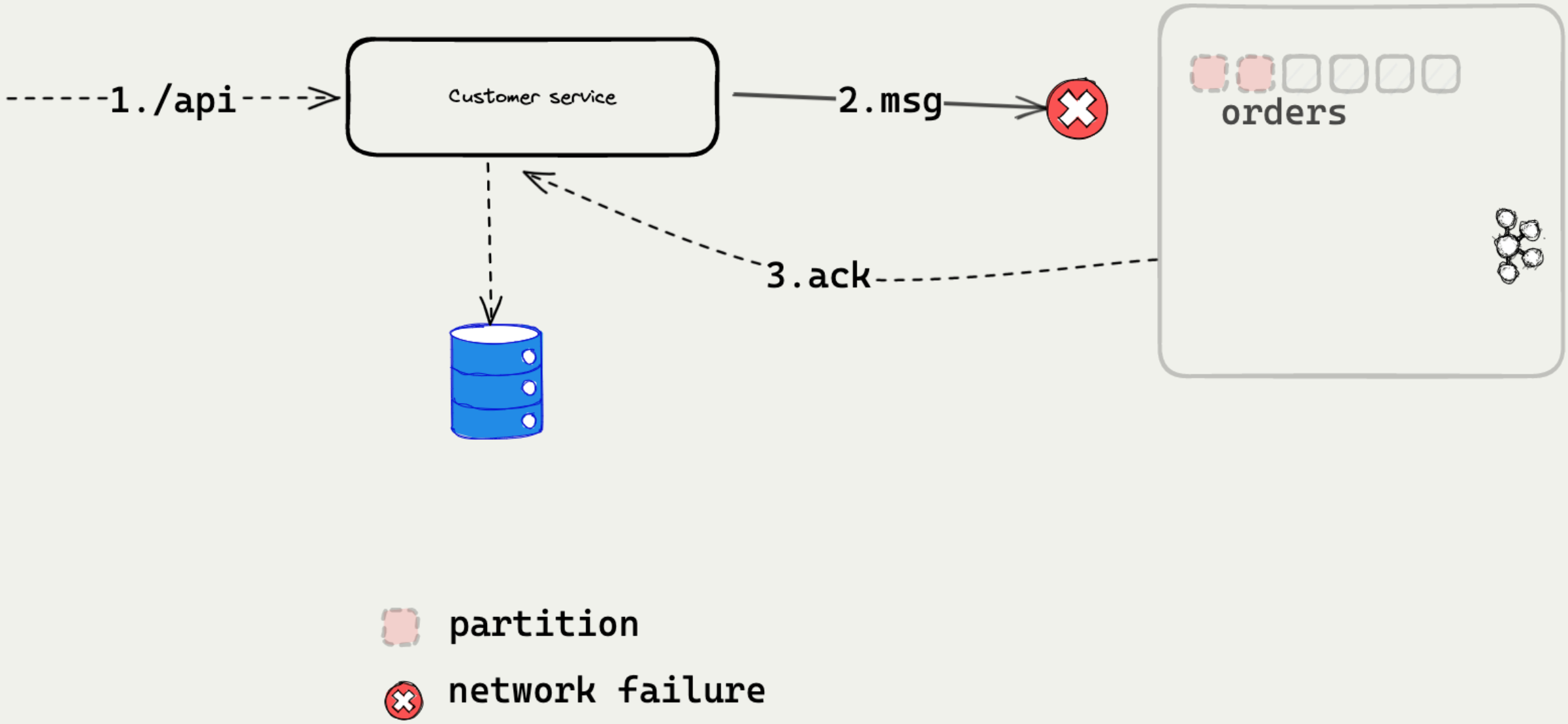
Kafka Terminologies

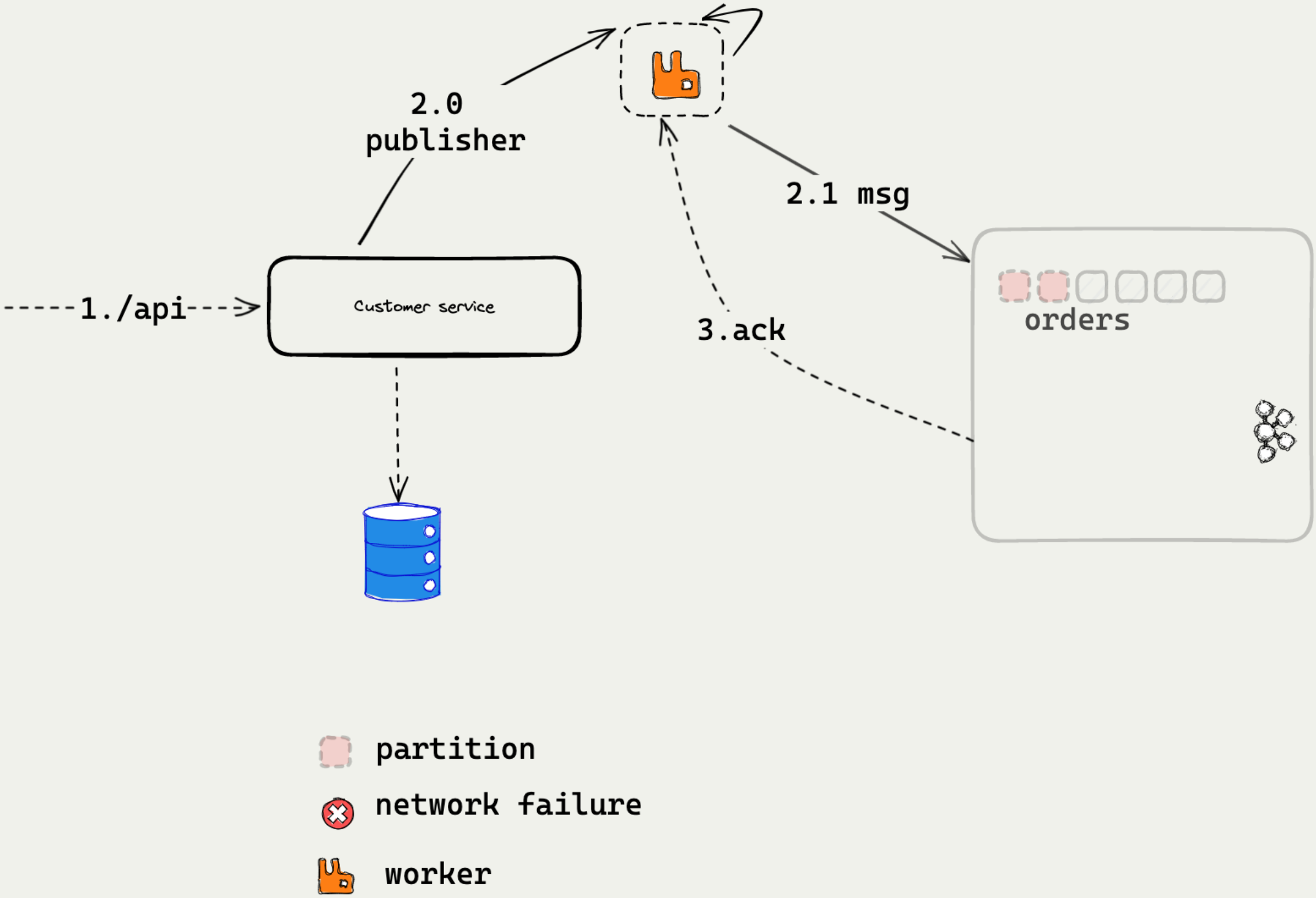
- Producer
- Consumer
- Topics (Offset, Partitions)
- Consumer groups



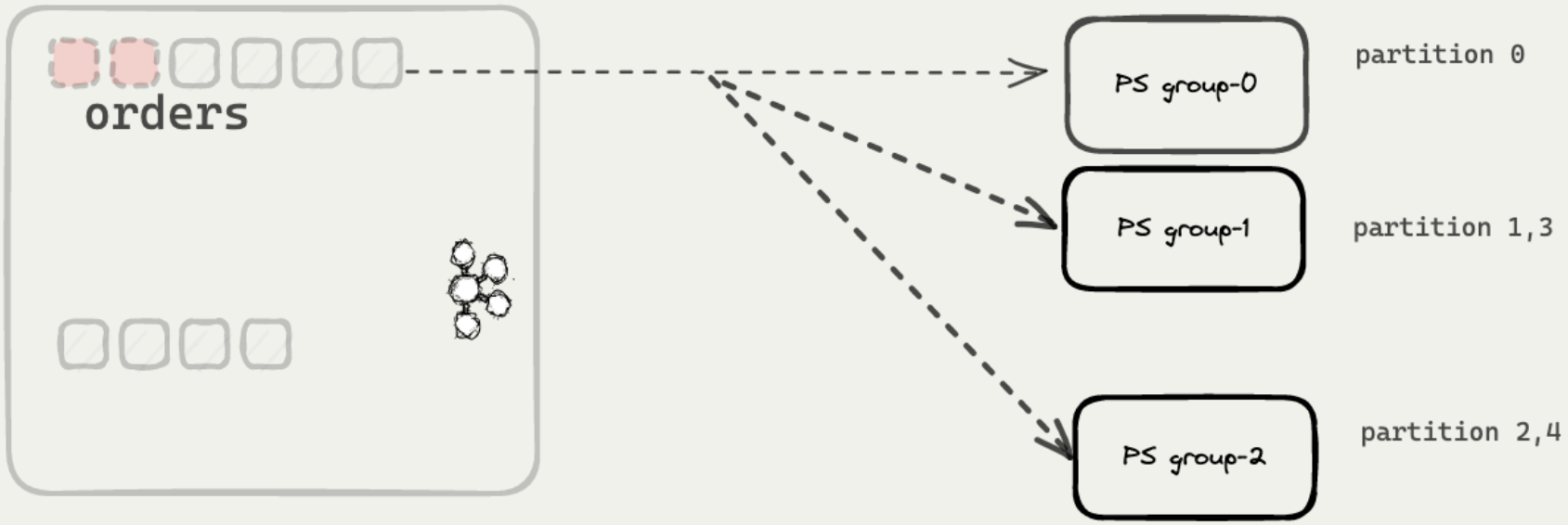
Produce Messages




1./api
Customer service
2.msg
orders
3.ack
partition
network failure


2.0
publisher
2.1 msg
1./api
Customer service
3.ack
orders
partition
network failure
worker

Consumers



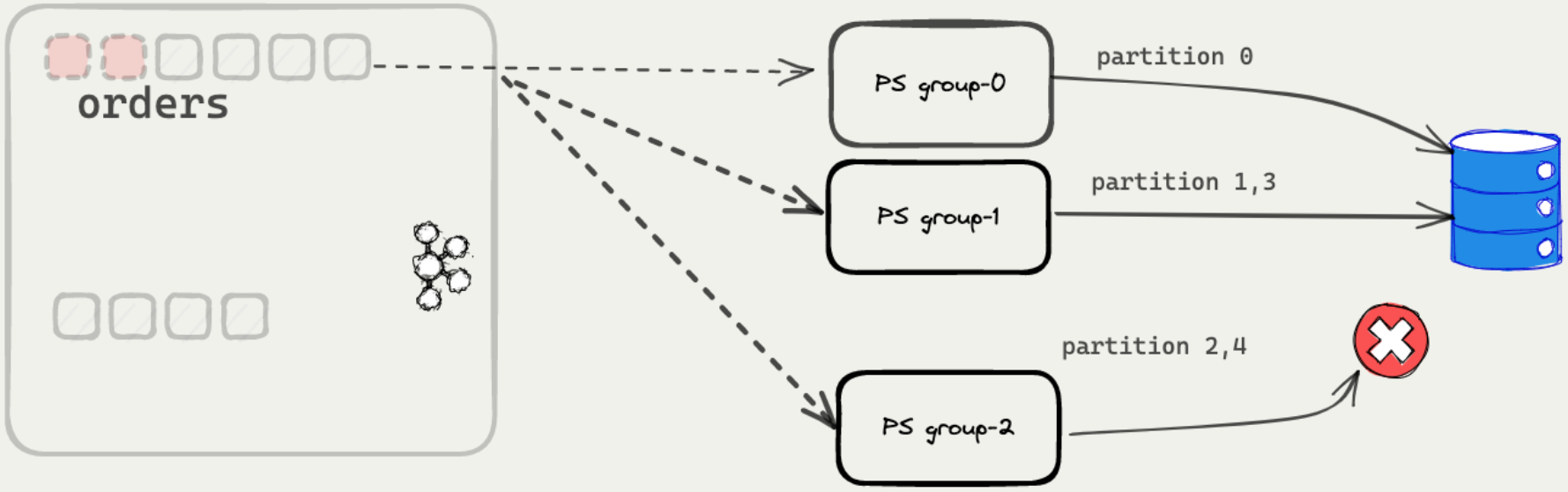
partition

network failure

worker

```
kafka-consumer-groups --bootstrap-server=localhost:9090 --group=consumer-group-0 -
```

```
GROUP           TOPIC           PARTITION  CURRENT-OFFSET  LOG-END-OFFSET  LAG
consumer-1      timed-data      1          1553            1553            0
consumer-1      timed-data      2          1592            1592            0
consumer-1      timed-data      0          1459            1461            2
```



partition

network failure

worker

Demo

- Auto commit offset
  - Consume messages and marks as read successfully in background
- Commit explicitly
  - More control and reliable | Production ready

```
enable.auto.commit
```

Production Readiness

Service Level

- Load Testing
- Logging
- Metrics
- Circuit breaker to prevent cascading failure

System

- Replication
- Sharding
- Load Balancer
- Metrics
- Event Driven Systems
- Architecture optimisation

HowTo / Tools

- Kubernetes
- HaProxy
- Nginx
- GLB / AWS LB
- Google PubSub
- Chaos Testing
- Observability

Load Testing

Ensure whether the service will be fine for higher production load.
Do 10x of estimated traffic for a future timeline

Tools: *wrk2, gatling, ab, vegeta, k6, ...*

Service metrics plotted as histogram

- service-0 has high latency

Architecture

As we build systems, we accomodate *hacks, tech debts and legacy decisions*.

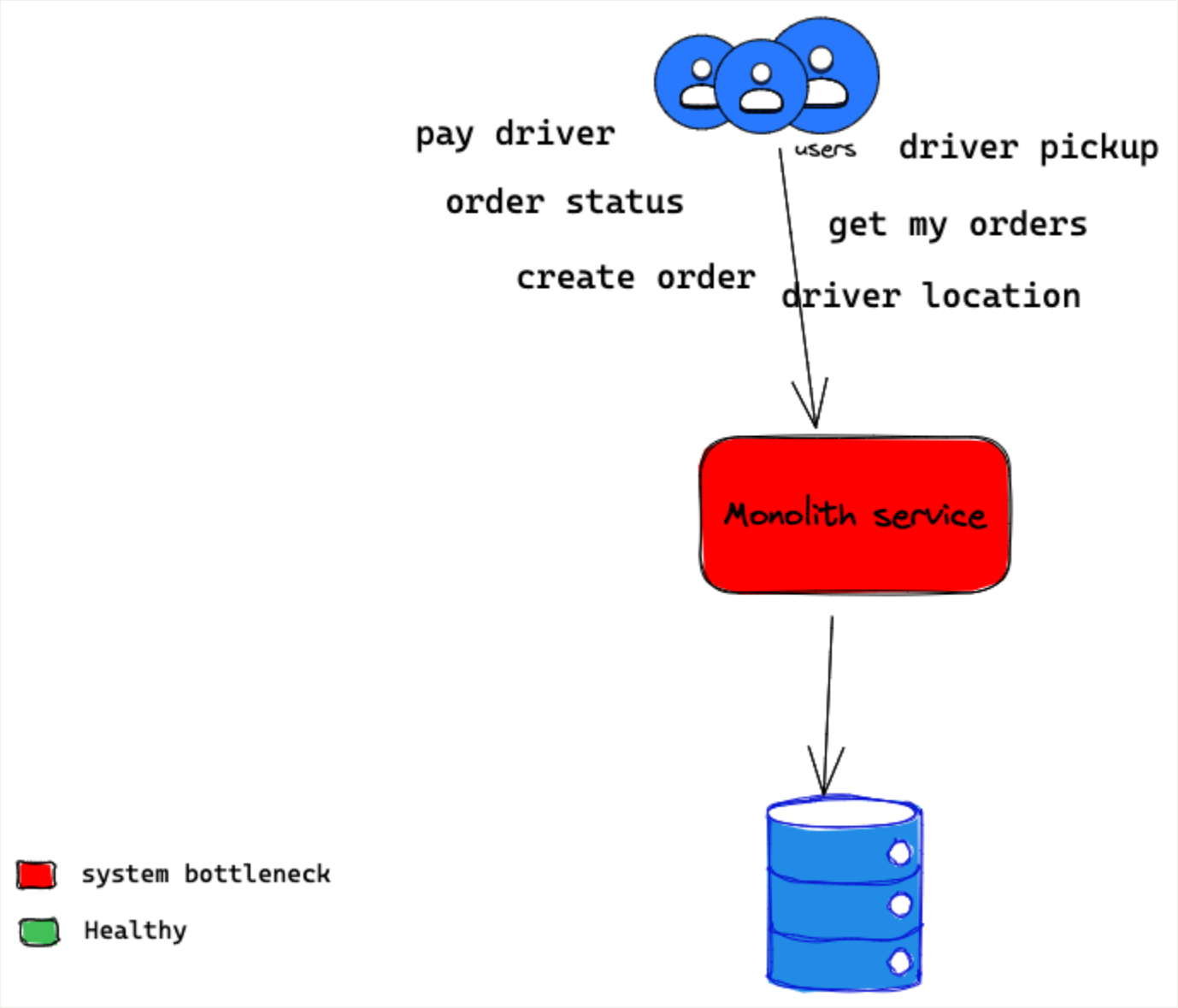
We have to rearchitecture or rebuild or remove complexity and extend architecture at times to scale our system further

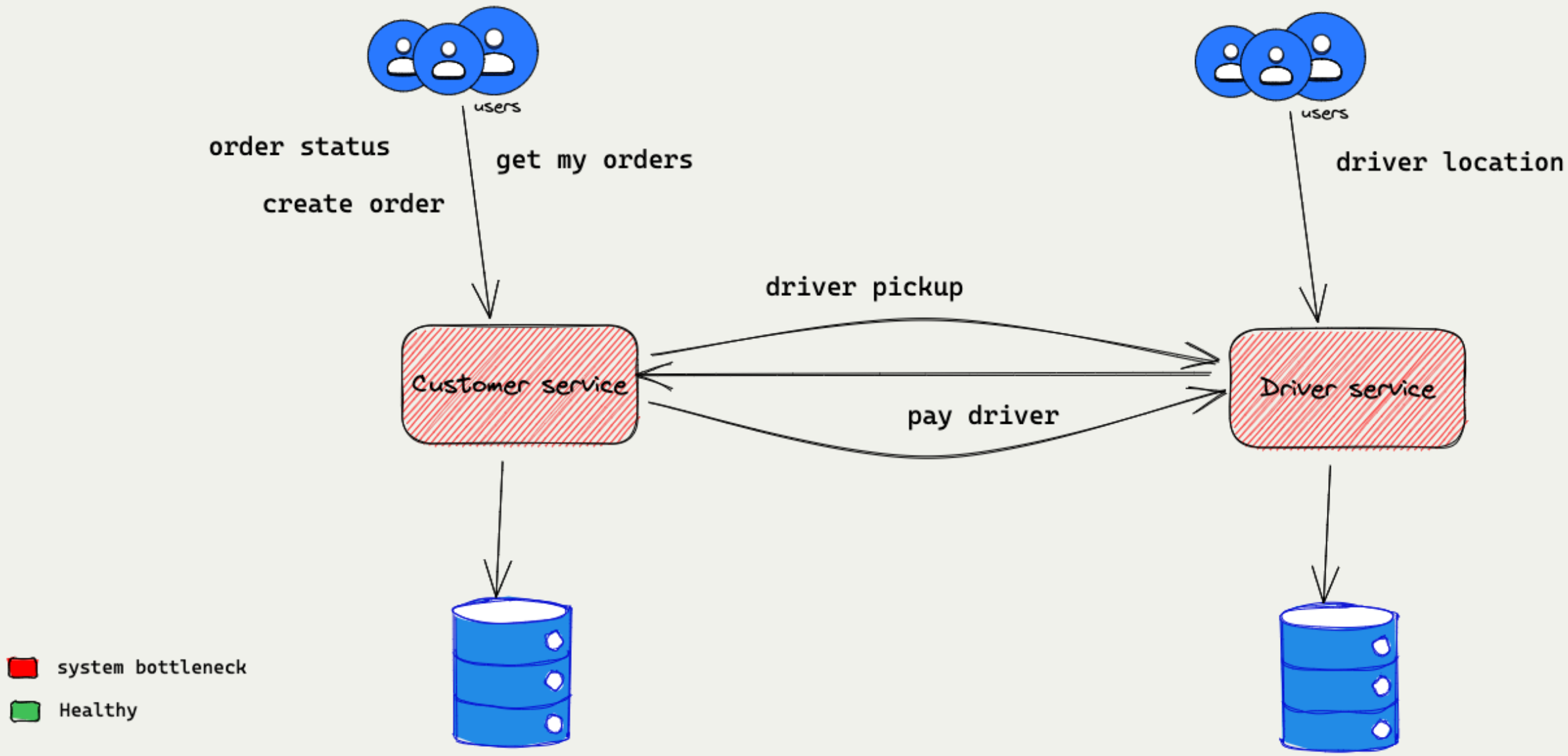
Adopting different stack (e.g, Go, rust as per need)

Adopting technologies (*kafka/rabbitmq/pubsub, BigQuery, etc*)

Monolith service

considering a domain like food ordering, ride hailing, etc.

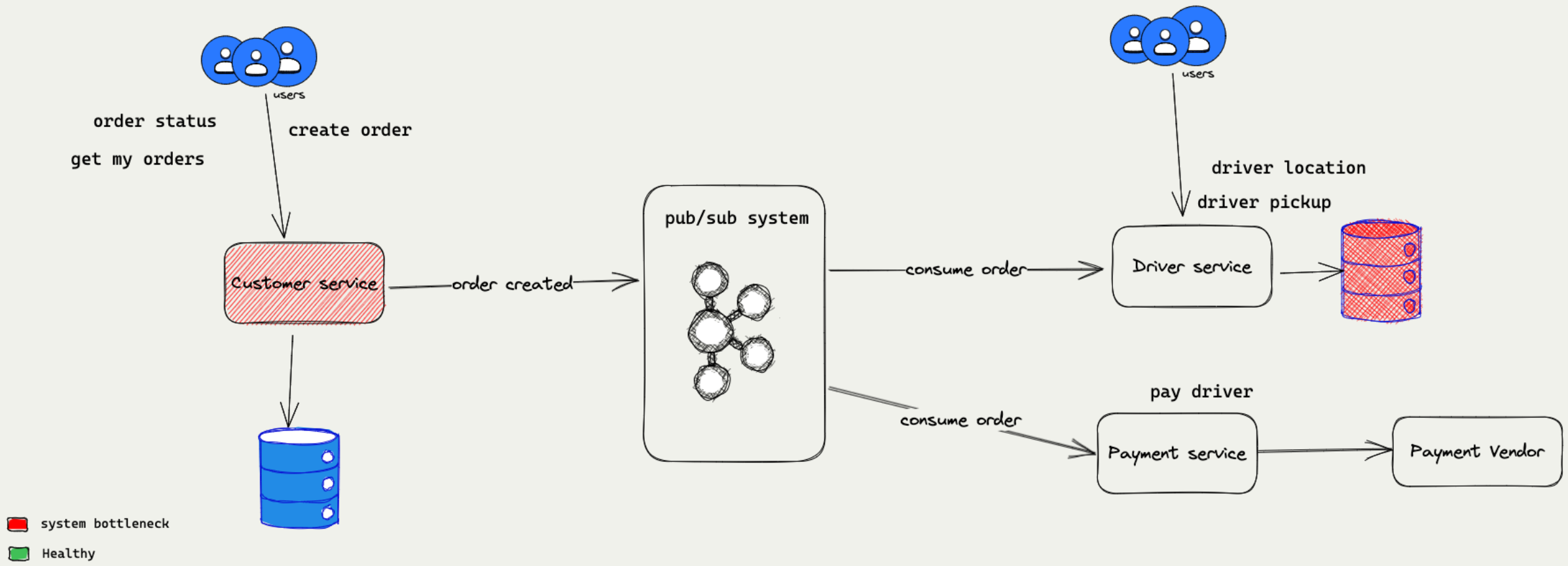




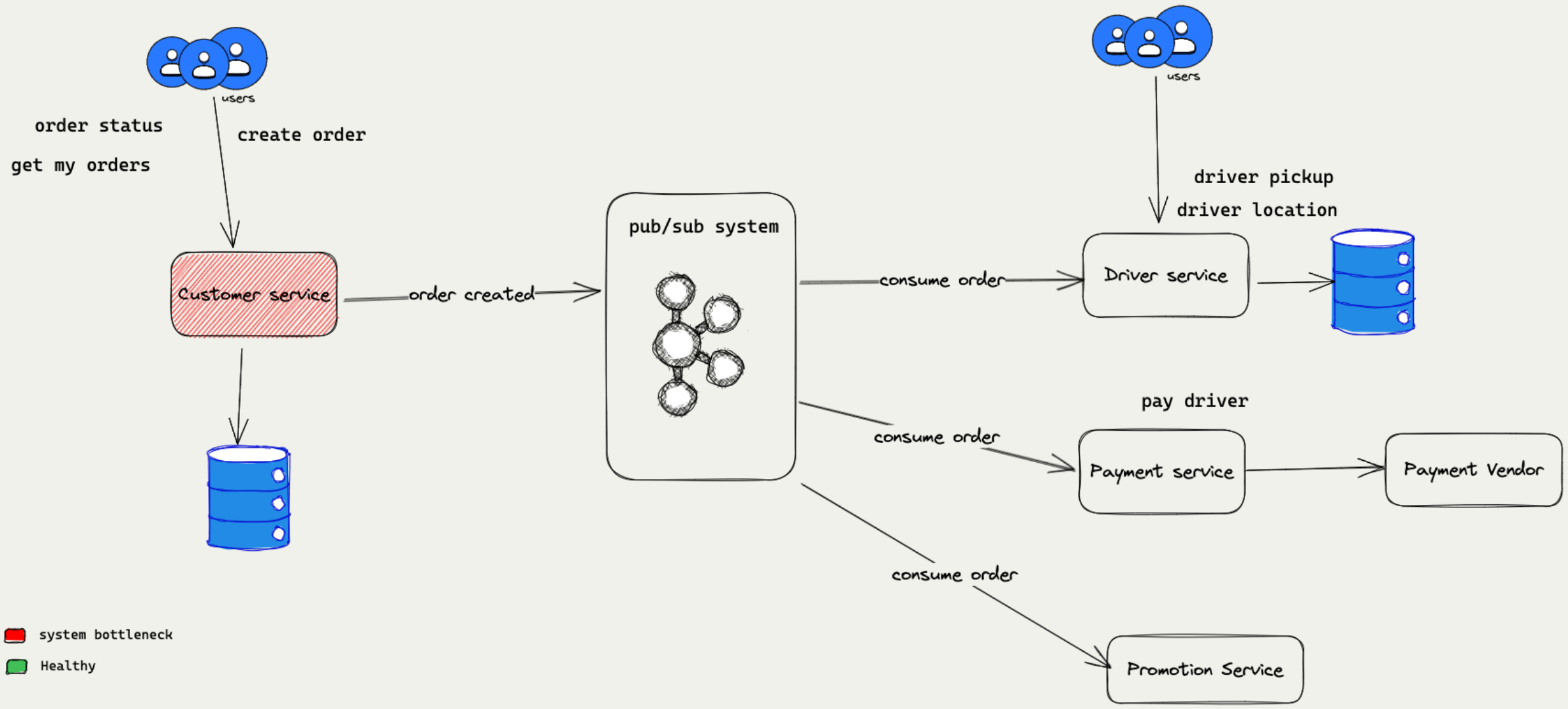
- Easy to scale independently (infrastructure)
- Teams can develop independently
- Data & complexity is isolated

Leveraging pub-system like kafka will enable us to extend features, independent functionalities etc.

Also keeping the system performant and stable.



more extensible and scaleable architecture for org



Advanced

- Don't go behind exactly once semantics
- Be idempotent
- Allow system wide partial failures
- Build Asynchronous systems

Best Practices

Microservices

- Sensible Timeouts
- Retries (assuming it'll fail) / leverage workers
  - Must be configured with retry limit
  - exponential backoffs
- Testing (TDD, Service Level, Contract)
- Backward compatibility
- Rollbacks
- IAC / Automation

Kafka

Consumers

- Consumer Group <= Partitions
- Prefer to avoid auto commit offsets
- Scale consumers based on dependencies (DB, resources etc)
- Use deadletter queue

Producers

- Reliable producer with configuration `acks=all`, `linger.ms`
- Producer - Batch / Streaming publish
- Use worker to produce events to deal with loss
- Prefer to avoid custom partitioning

Observability

- Monitoring
- Alerting
- Centralized logging
  - log properly with required information and additional metadata eg: status, method, order ...)
  - log stitching
- Tracing
  - As we scale systems with multiple components & dependencies it's required for finding component responsible for the latency/error

topic deems separte discussion or session

!

Call for early adopters / design partners

observability, reliable infrastructure *(kafka, postgres, redis, ...)*

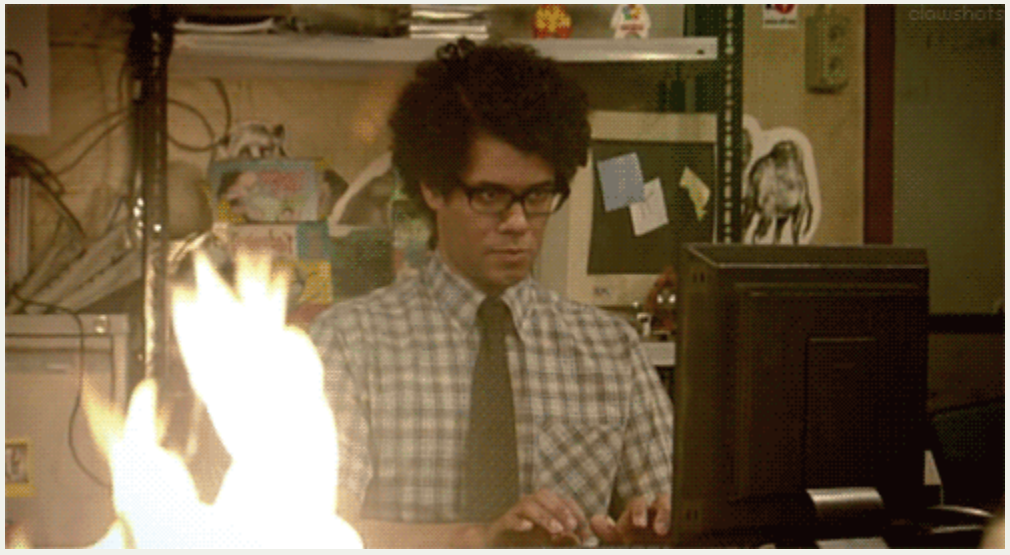
Aiming to help companies prevent downtime and reduce friction with adoption & burnout in devops

relyonmetrics.com

Reference

- Byzantine general's problem
- CAP Theorem
- Previous Talks - devdinu.github.com/talks.html
- Consumer config - enable.auto.commit
- Producer config
- Common pitfalls to avoid
- Netflix - chaos monkey

Thanks



Discuss / Questions

Feel free to DM / Setup a short call

Talks: *@devdineshkumar* Github: *@devdinu*

Youtube: *@relyonmetrics*