Building High Scale Backend Systems

11 January 2023

Dineshkumar, Founder @ relyonmetrics

@devdineshkumar devdinu.github.io

About

Founder @ relyonmetrics

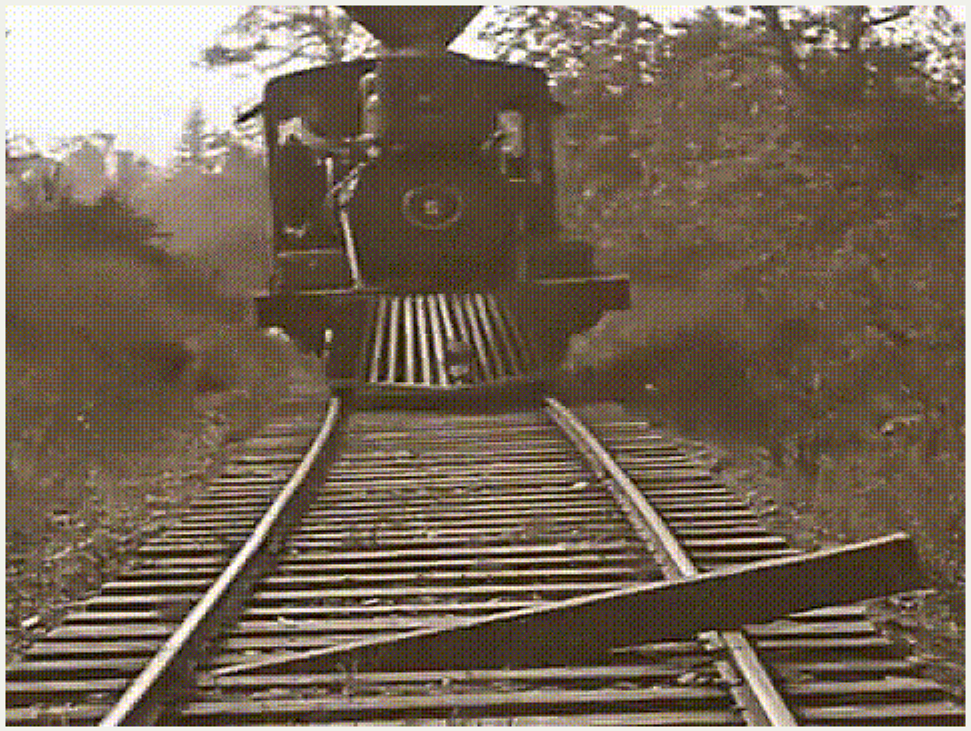
- Built backend for security platform @ Elastic, backend system for cmd.com
- Built event driven system for high scale @ Gojek
- Go, Kafka, distributed-systems enthusiast

Agenda

- Basics
- Production Readiness
- Service Level
- Observability
- System Level / Architecture
- Best Practices

Basics

Assume everything will fail! 🔥



Production Readiness

Service Level

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

Simple

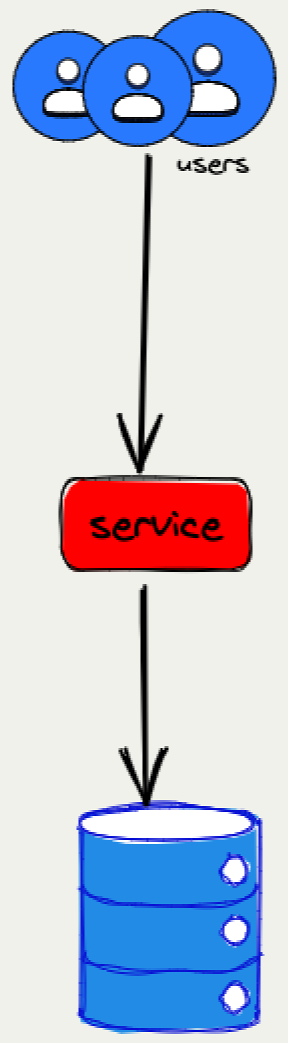
- Disable debug logs
- log id, latency and status codes to begin with
- Testing (ensure you add tests for every failures)

  `#development`

System

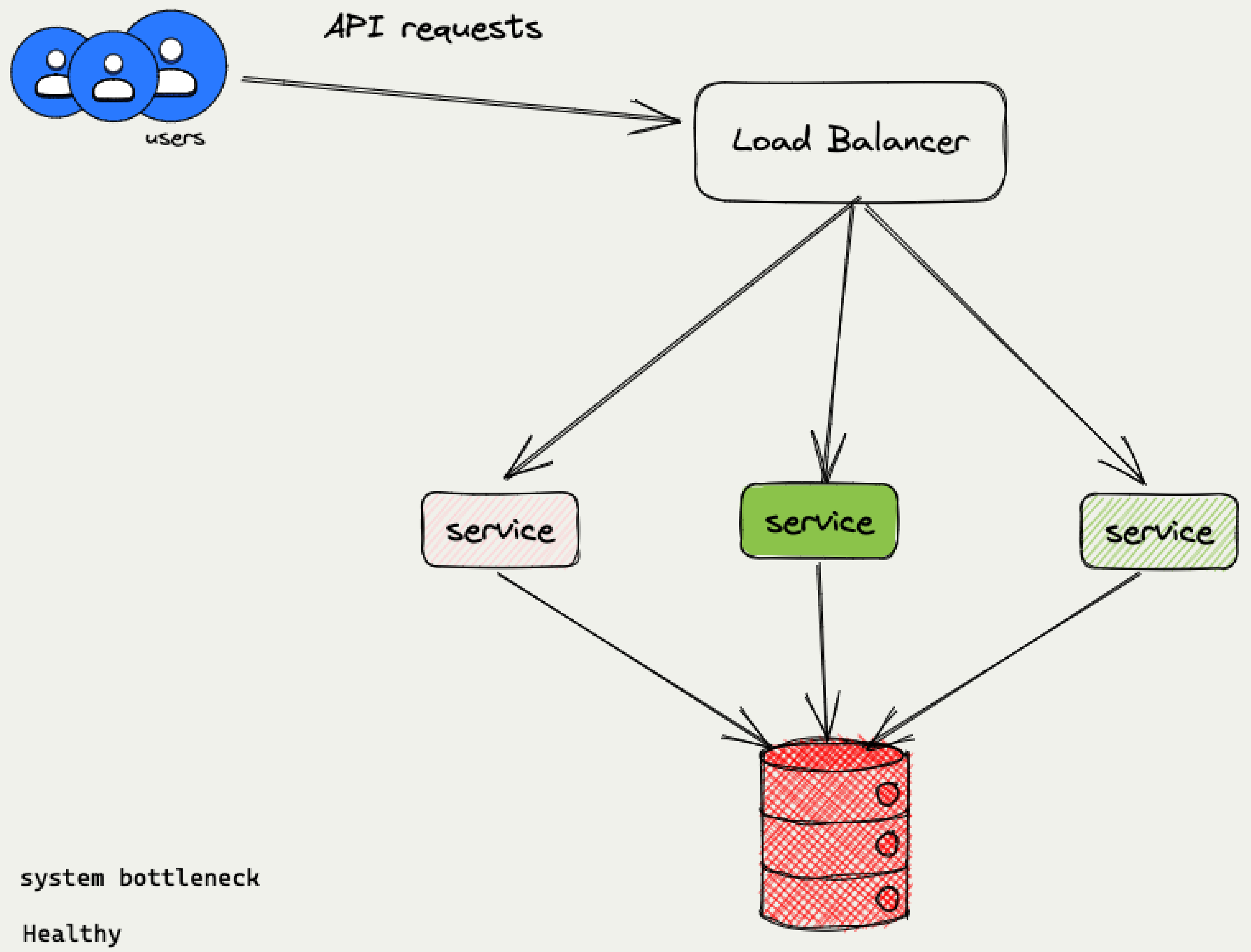
- Replication
- Sharding
- Load Balancer
- Metrics
- Event Driven Systems
- Architecture optimisation

Monolith service with DB



🟥 system bottleneck

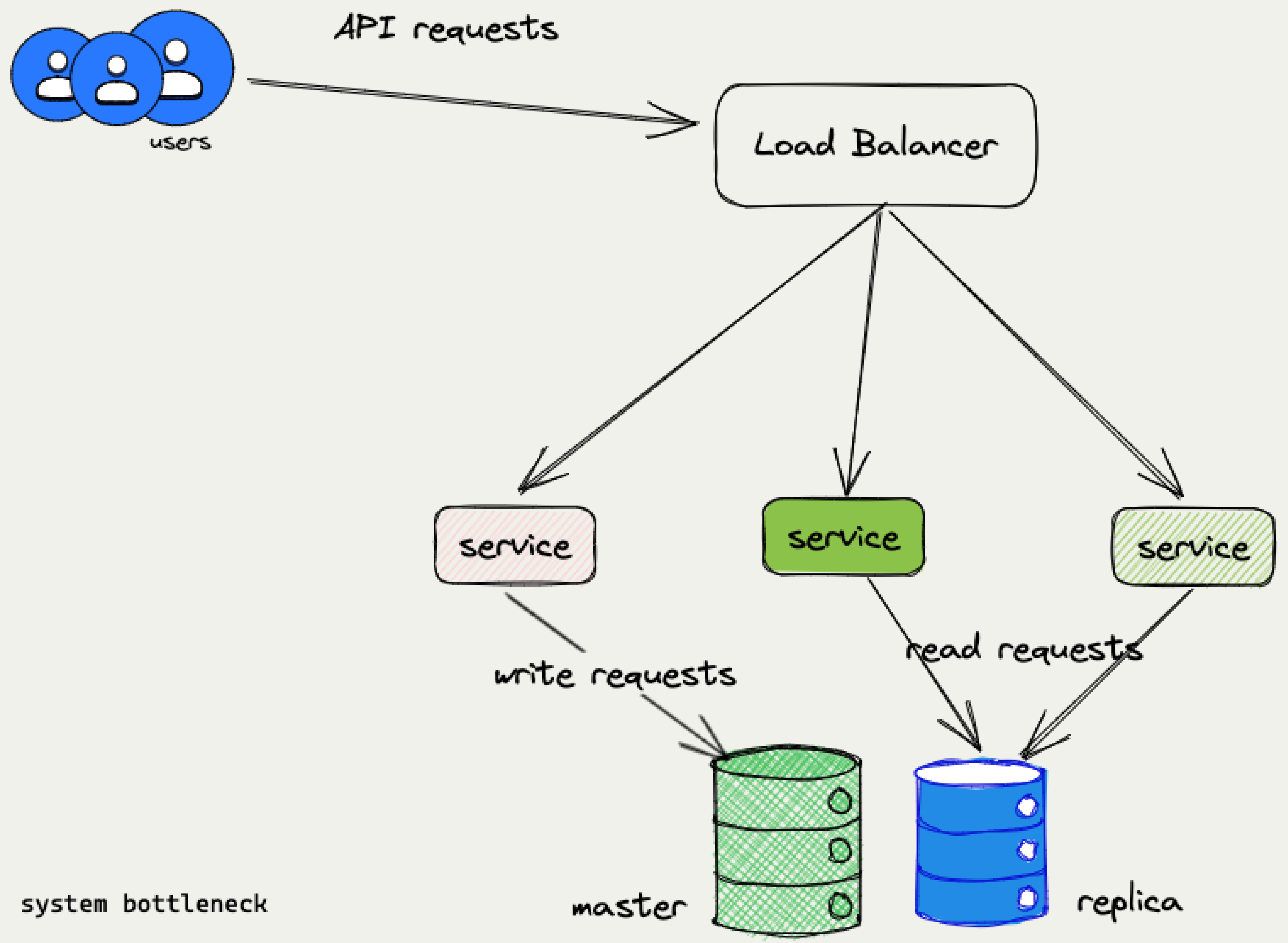
Database becoming a bottleneck



🟥 system bottleneck

🟩 Healthy

Simple service with DB replication



🟥 system bottleneck

🟩 Healthy

HowTo / Tools

few load balancing tools,

- Kubernetes
- HaProxy
- Nginx
- GLB / AWS LB

LoadBalancer

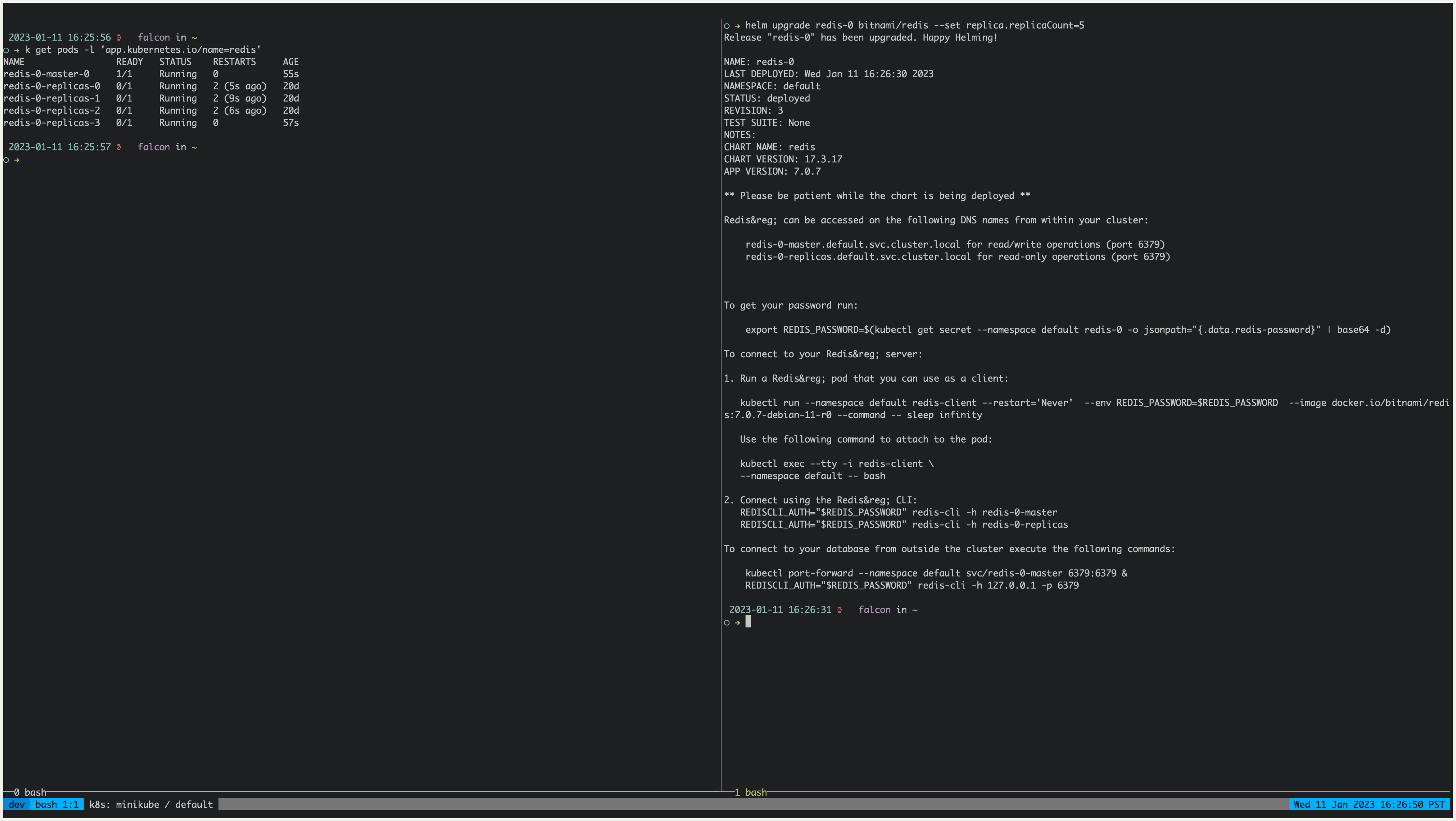
Sample load balancer config[1] for nginx with multiple backend instances as upstream

```
http {
    upstream myapp1 {
        server srv1.example.com;
        server srv2.example.com;
        server srv3.example.com;
    }

    server {
        listen 80;

        location / {
            proxy_pass http://myapp1;
        }
    }
}
```

1. Nginx Documentation for load balancing

redis cluster scaled up in k8s

```
helm upgrade redis-0 bitnami/redis --set replica.replicaCount=5
```



```
 2023-01-11 16:25:56    falcon in ~
○ → k get pods -l 'app.kubernetes.io/name=redis'
NAME                  READY   STATUS    RESTARTS     AGE
redis-0-master-0      1/1     Running   0            55s
redis-0-replicas-0    0/1     Running   2 (5s ago)   20d
redis-0-replicas-1    0/1     Running   2 (9s ago)   20d
redis-0-replicas-2    0/1     Running   2 (6s ago)   20d
redis-0-replicas-3    0/1     Running   0            57s

 2023-01-11 16:25:57    falcon in ~
○ →
```

```
○ → helm upgrade redis-0 bitnami/redis --set replica.replicaCount=5
Release "redis-0" has been upgraded. Happy Helming!

NAME: redis-0
LAST DEPLOYED: Wed Jan 11 16:26:30 2023
NAMESPACE: default
STATUS: deployed
REVISION: 3
TEST SUITE: None
NOTES:
CHART NAME: redis
CHART VERSION: 17.3.17
APP VERSION: 7.0.7

** Please be patient while the chart is being deployed **

Redis&reg; can be accessed on the following DNS names from within your cluster:

    redis-0-master.default.svc.cluster.local for read/write operations (port 6379)
    redis-0-replicas.default.svc.cluster.local for read-only operations (port 6379)



To get your password run:

    export REDIS_PASSWORD=$(kubectl get secret --namespace default redis-0 -o jsonpath="{.data.redis-password}" | base64 -d)

To connect to your Redis&reg; server:

1. Run a Redis&reg; pod that you can use as a client:

   kubectl run --namespace default redis-client --restart='Never'  --env REDIS_PASSWORD=$REDIS_PASSWORD  --image docker.io/bitnami/redi
s:7.0.7-debian-11-r0 --command -- sleep infinity

   Use the following command to attach to the pod:

   kubectl exec --tty -i redis-client \
   --namespace default -- bash

2. Connect using the Redis&reg; CLI:
   REDISCLI_AUTH="$REDIS_PASSWORD" redis-cli -h redis-0-master
   REDISCLI_AUTH="$REDIS_PASSWORD" redis-cli -h redis-0-replicas

To connect to your database from outside the cluster execute the following commands:

    kubectl port-forward --namespace default svc/redis-0-master 6379:6379 &
    REDISCLI_AUTH="$REDIS_PASSWORD" redis-cli -h 127.0.0.1 -p 6379

 2023-01-11 16:26:31    falcon in ~
○ →
```

```
0 bash                                  1 bash
dev  bash 1:1  k8s: minikube / default                Wed 11 Jan 2023 16:26:50 PST
```

Load Testing

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

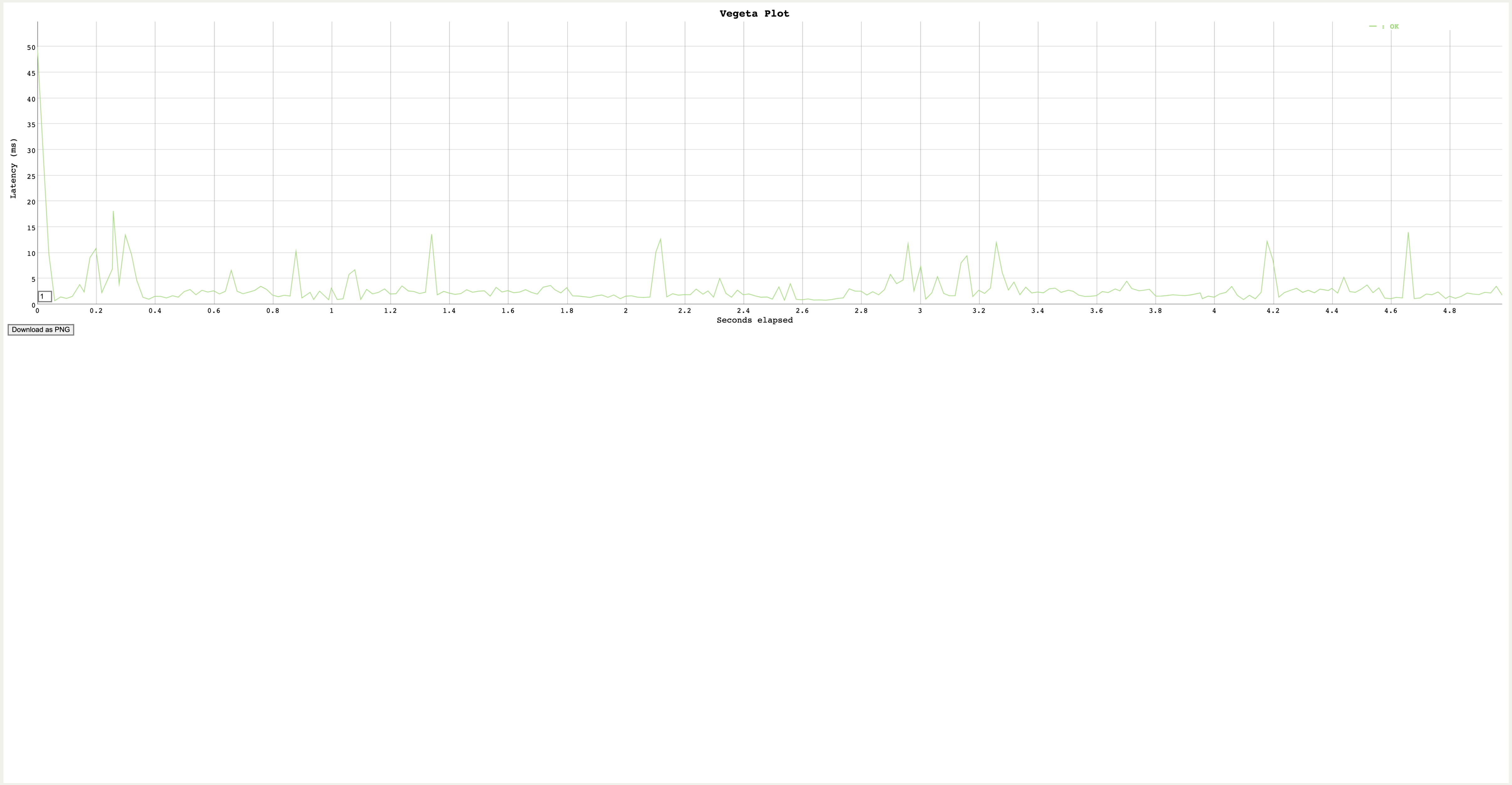
Tools: *wrk2*, *gatling*, *ab*, *vegeta*, *k6*, *...*

Demo

- Nginx service
- 2 replicas
- connected to postgres

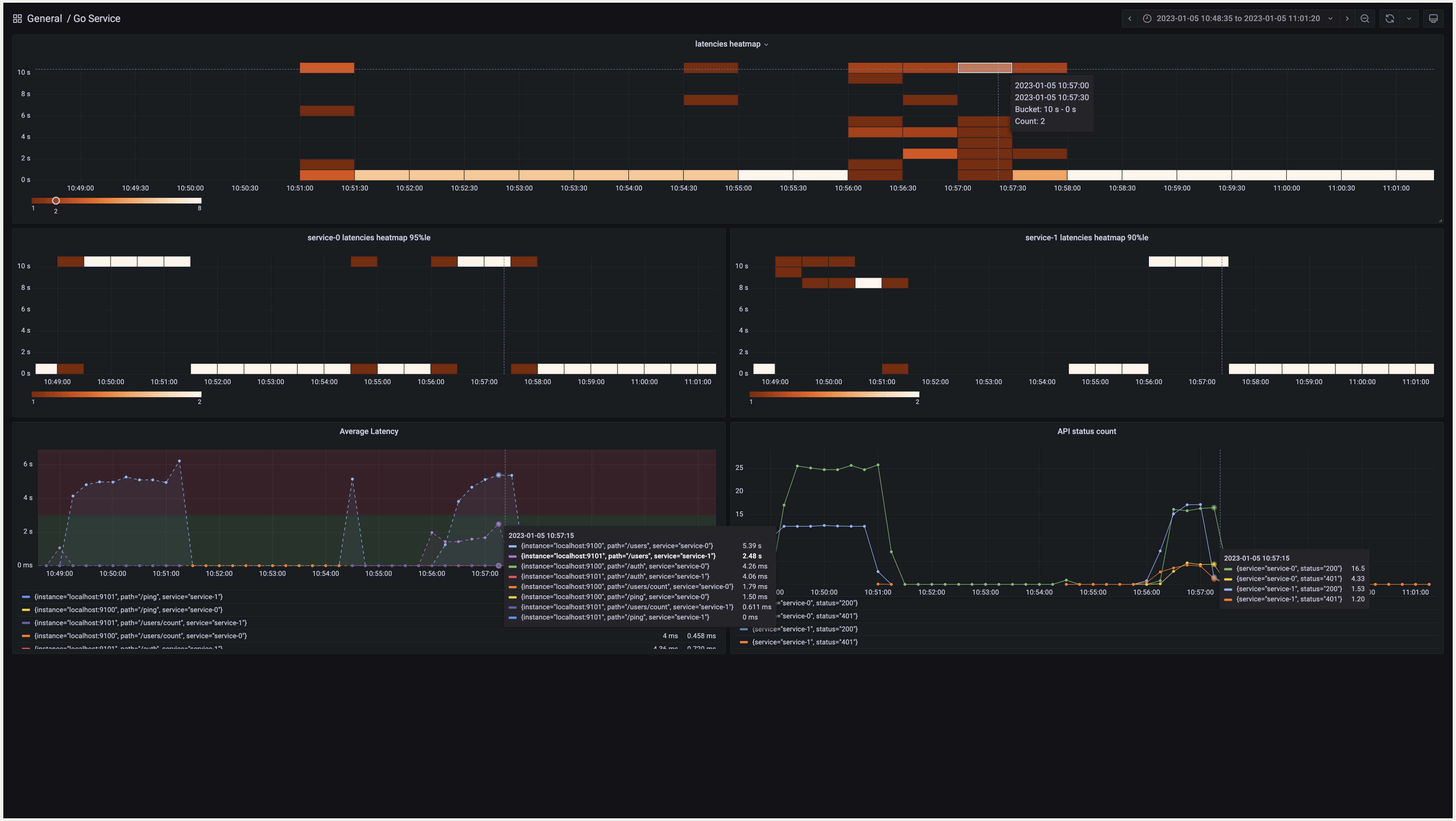
Sample Vegeta Report

```
vegeta attack -targets backend  -duration=5s  -timeout=300s \
| vegeta plot > output.html
```



Service metrics plotted as histogram

- service-0 has high latency



General / Go Service

2023-01-05 10:48:35 to 2023-01-05 11:01:20

latencies heatmap

2023-01-05 10:57:00
2023-01-05 10:57:30
Bucket: 10 s - 0 s
Count: 2

service-0 latencies heatmap 95%le

service-1 latencies heatmap 90%le

Average Latency

```
2023-01-05 10:57:15
{instance="localhost:9100", path="/users", service="service-0"}          5.39 s
{instance="localhost:9101", path="/users", service="service-1"}          2.48 s
{instance="localhost:9100", path="/auth", service="service-0"}           4.26 ms
{instance="localhost:9101", path="/auth", service="service-1"}           4.06 ms
{instance="localhost:9100", path="/users/count", service="service-0"}    1.79 ms
{instance="localhost:9100", path="/ping", service="service-0"}           1.50 ms
{instance="localhost:9101", path="/users/count", service="service-1"}    0.611 ms
{instance="localhost:9101", path="/ping", service="service-1"}           0 ms
```

```
{instance="localhost:9101", path="/ping", service="service-1"}
{instance="localhost:9100", path="/ping", service="service-0"}
{instance="localhost:9101", path="/users/count", service="service-1"}
{instance="localhost:9100", path="/users/count", service="service-0"}    4 ms    0.458 ms
```

API status count

```
2023-01-05 10:57:15
{service="service-0", status="200"}   16.5
{service="service-0", status="401"}   4.33
{service="service-1", status="200"}   1.53
{service="service-1", status="401"}   1.20
```

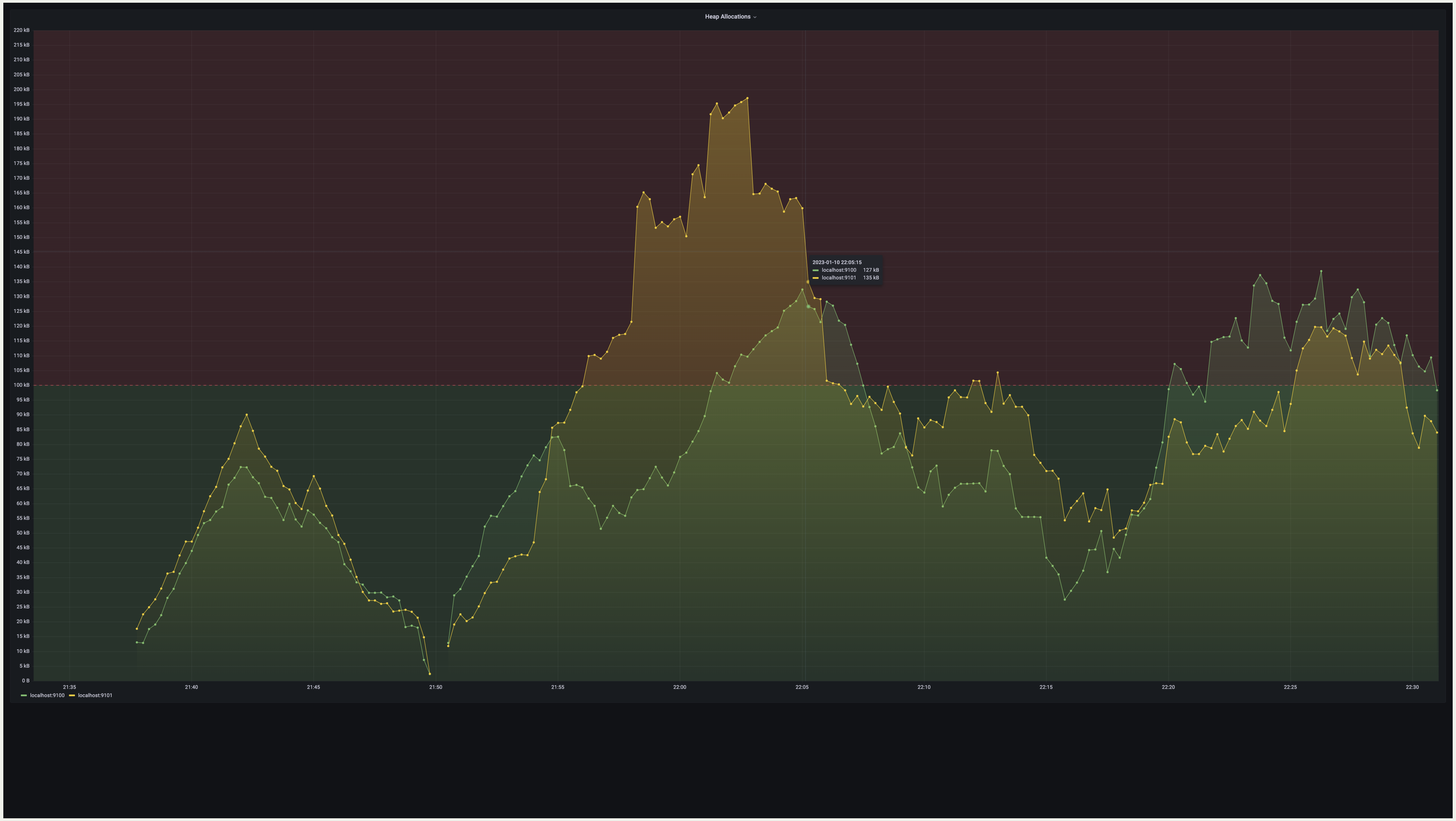
```
{service="service-0", status="200"}
{service="service-0", status="401"}
{service="service-1", status="200"}
{service="service-1", status="401"}
```

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

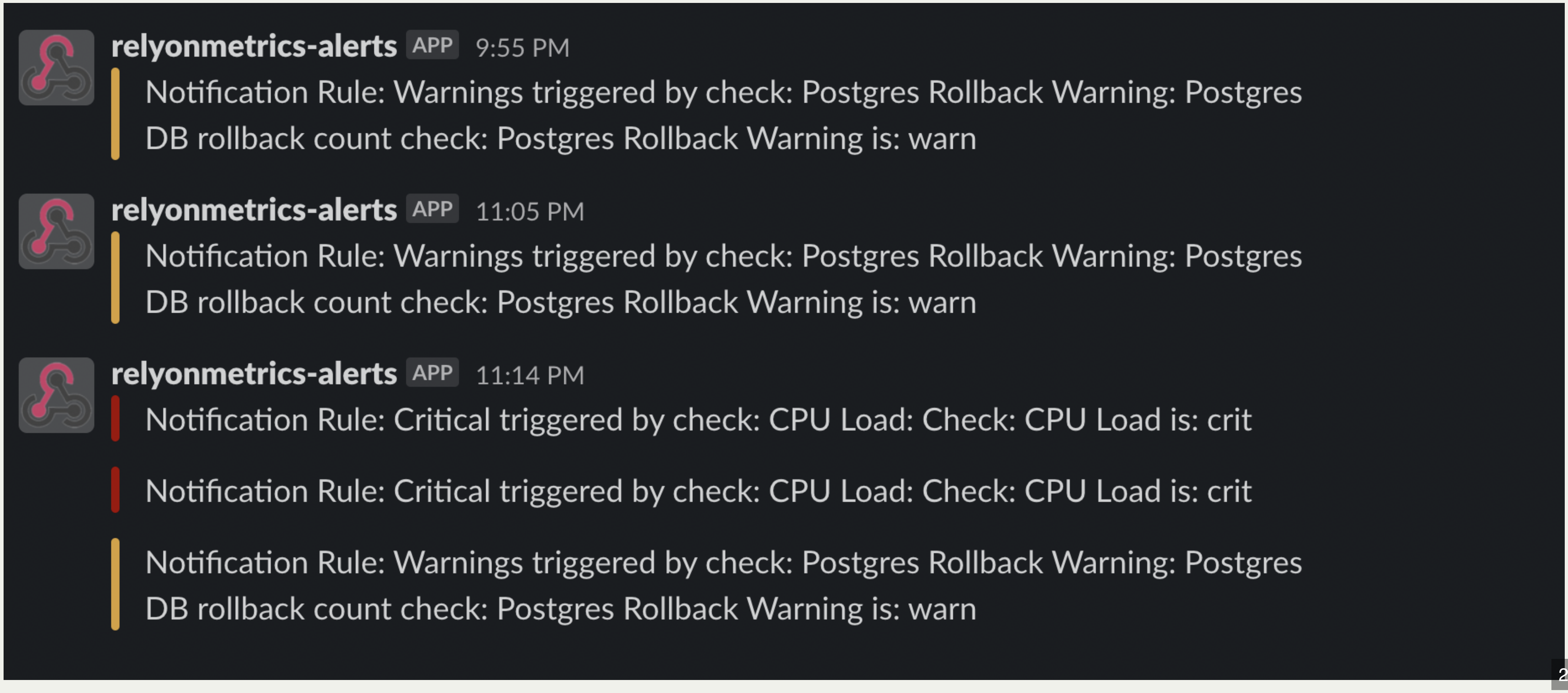
Track resources



Alerting

when metrics crosses a threshold or becomes anomaly

- Performance Metrics (latency)
- Rate change (throughput)
- Failures (HTTP)



relyonmetrics-alerts APP 9:55 PM

Notification Rule: Warnings triggered by check: Postgres Rollback Warning: Postgres DB rollback count check: Postgres Rollback Warning is: warn

relyonmetrics-alerts APP 11:05 PM

Notification Rule: Warnings triggered by check: Postgres Rollback Warning: Postgres DB rollback count check: Postgres Rollback Warning is: warn

relyonmetrics-alerts APP 11:14 PM

Notification Rule: Critical triggered by check: CPU Load: Check: CPU Load is: crit

Notification Rule: Critical triggered by check: CPU Load: Check: CPU Load is: crit

Notification Rule: Warnings triggered by check: Postgres Rollback Warning: Postgres DB rollback count check: Postgres Rollback Warning is: warn

Architecture

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

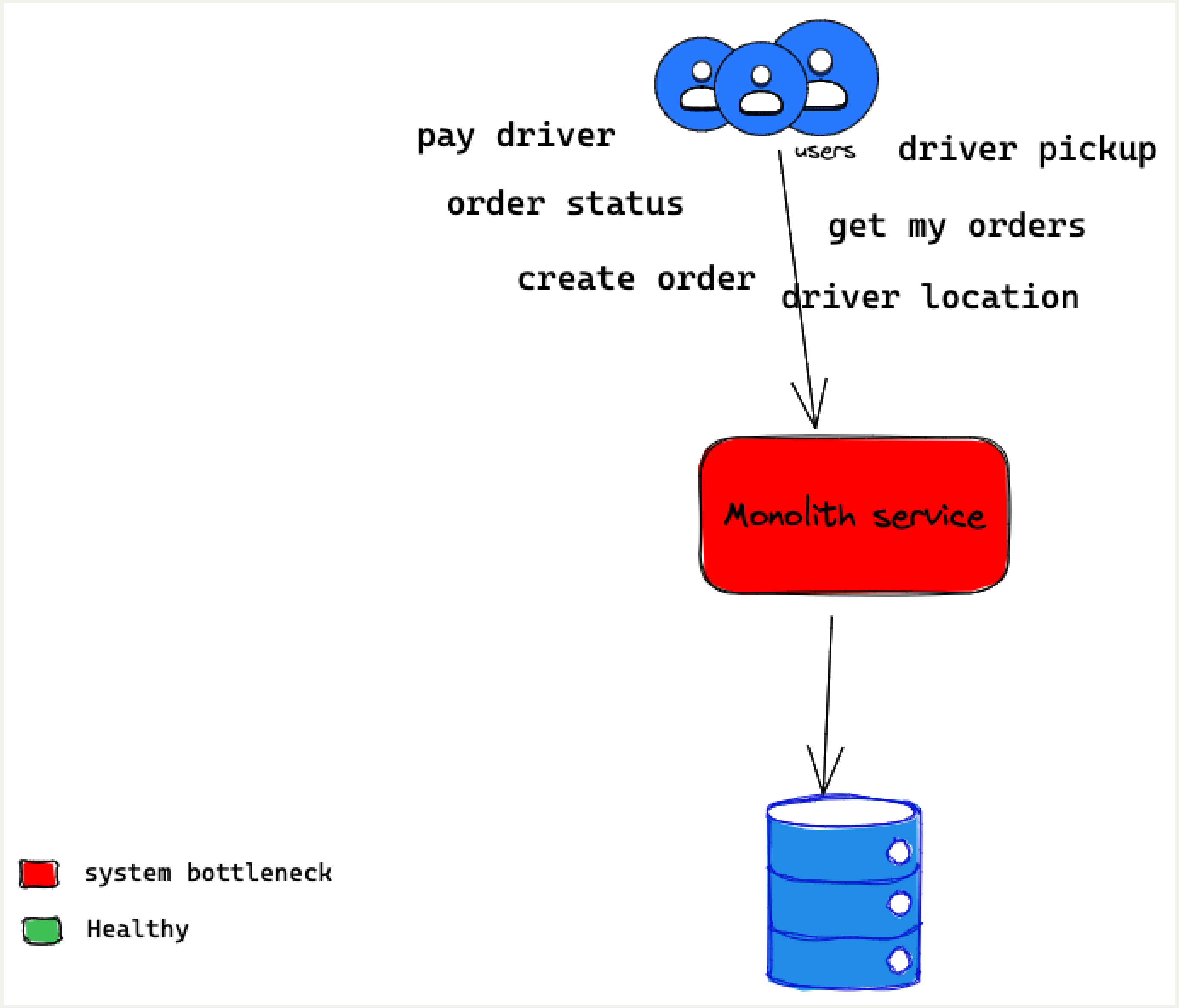
We have to rearchitecture or rebuild or remove complexity and extend architecture at times to scale our system further

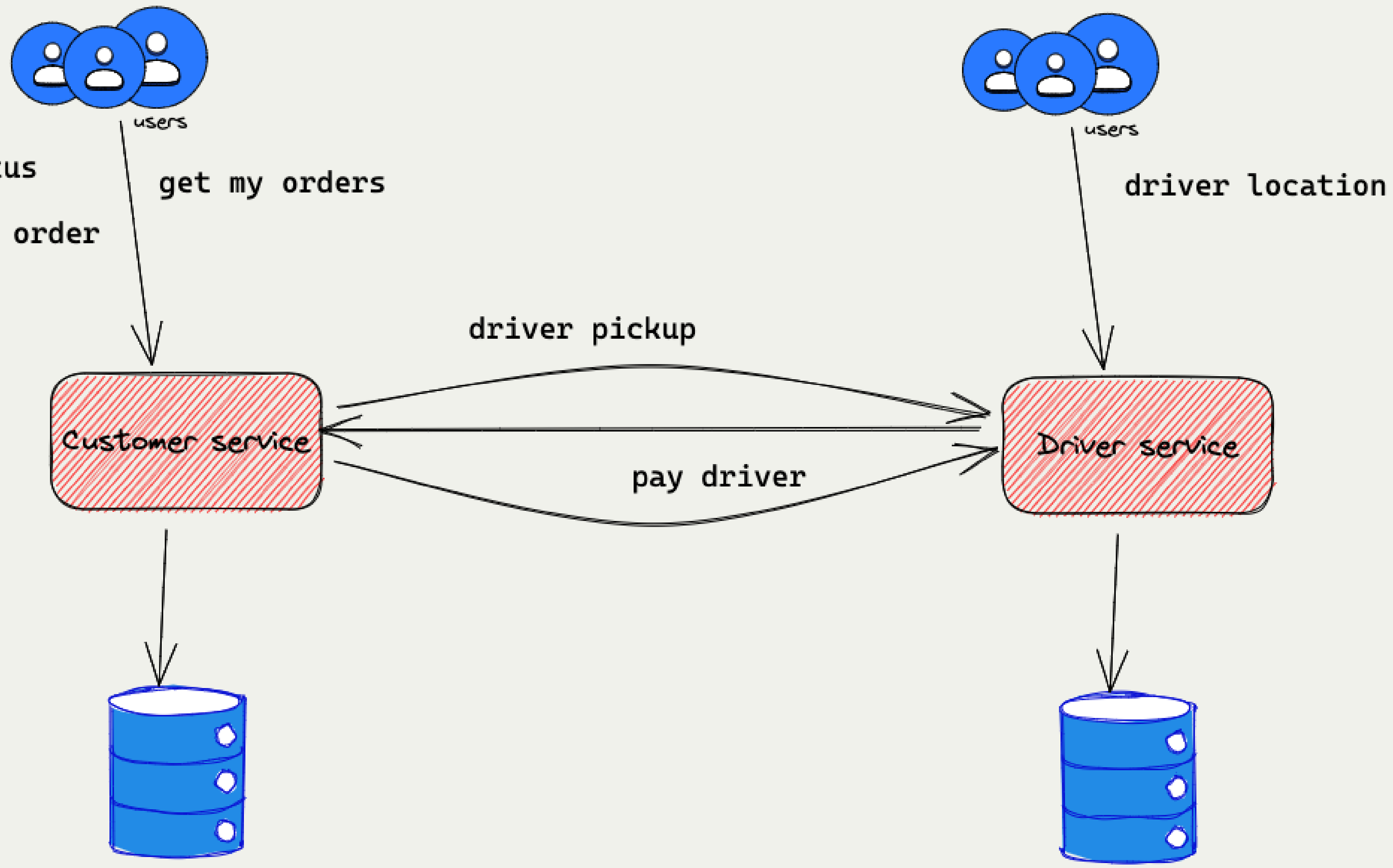
Adopting different stack (e.g, Go, rust as per need)

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

Monolith service

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





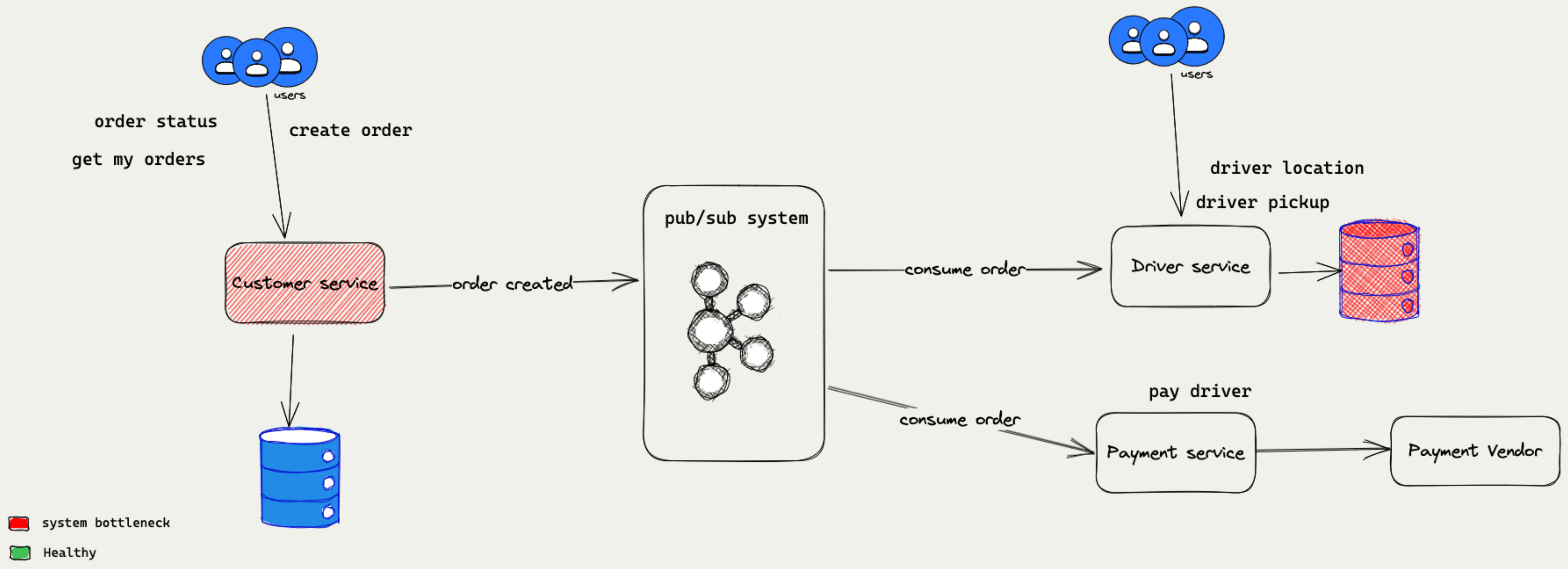
🟥 system bottleneck

🟩 Healthy

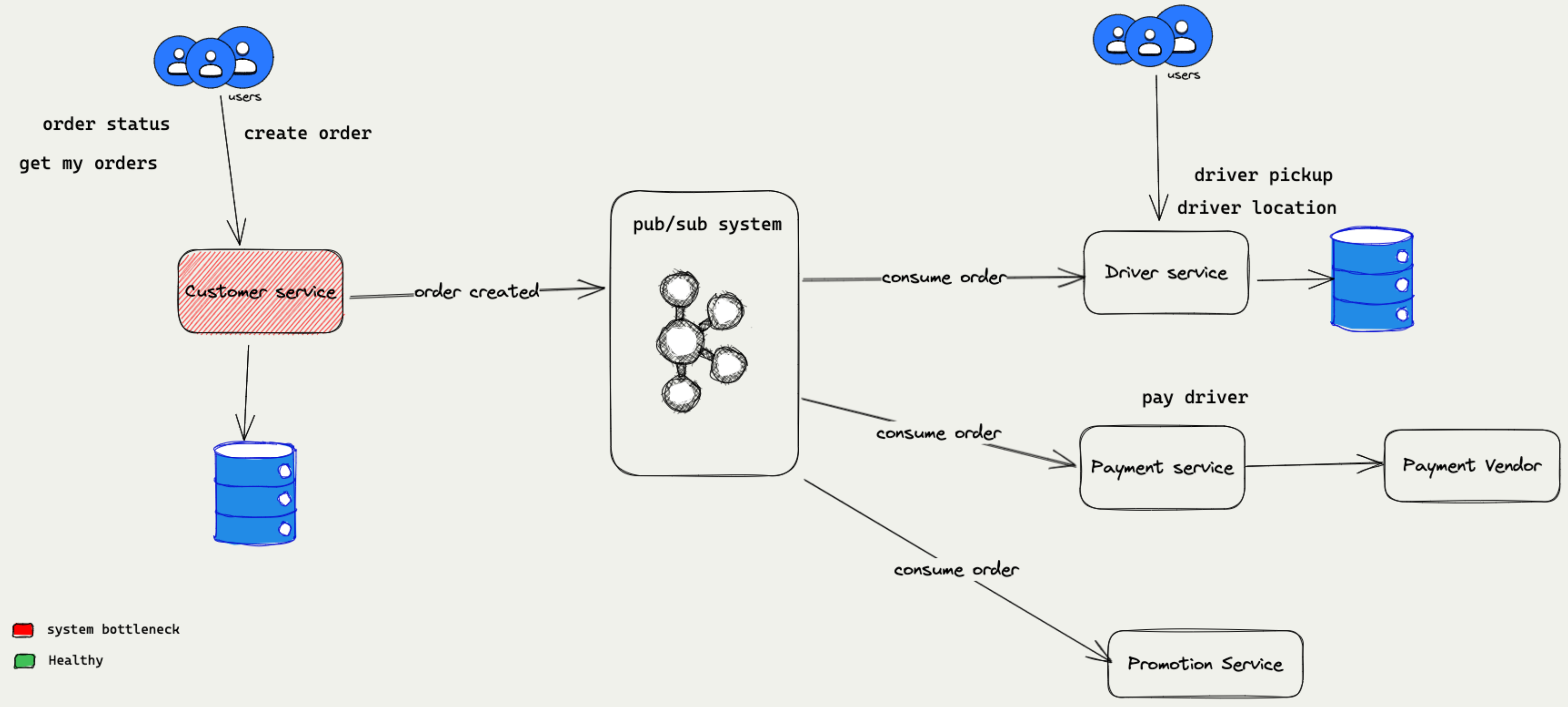
- Easy to scale independently (infrastructure)
- Teams can develop independently
- Data & complexity is isolated

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



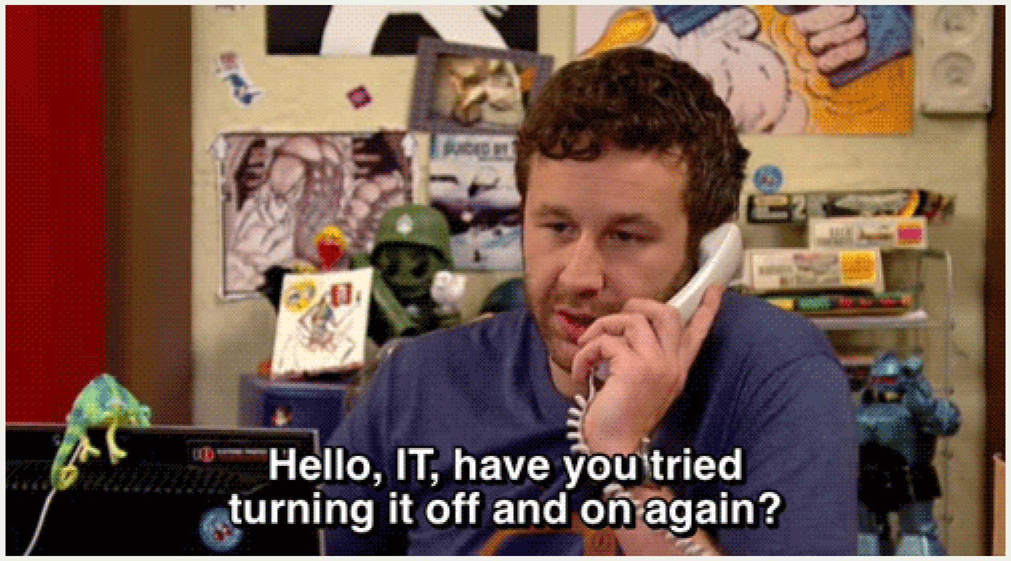
Leaks

resource leaks will be visible only during high load

Few of them are,

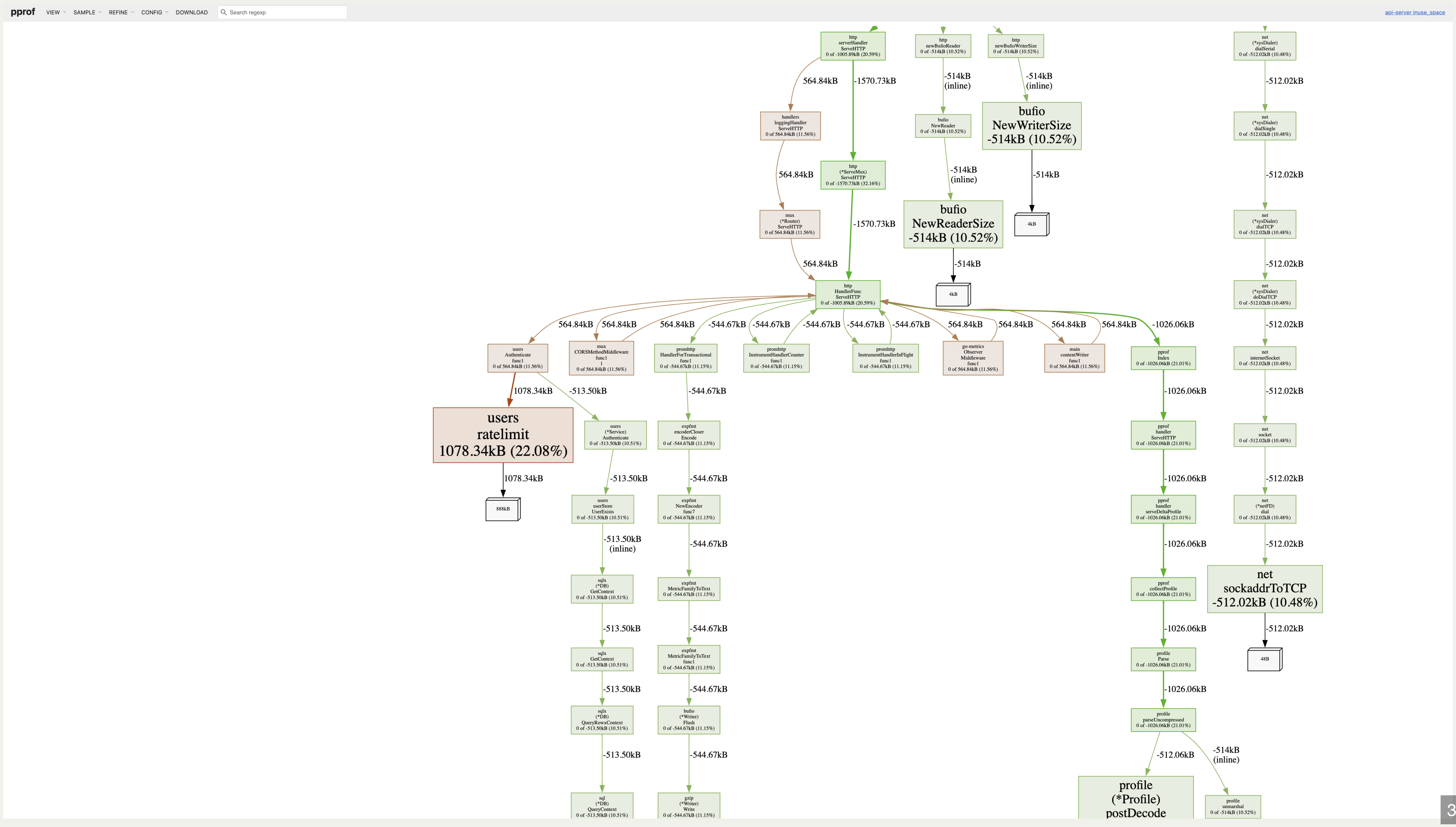
- Memory leaks
- goroutines / thread leaks
- file descriptor leaks

Solution 🔥



Profiling

```
curl http://localhost:9100/debug/pprof/heap?seconds=60 -o api-server-snapshot
go tool pprof -http=:3335  ./api-server-snapshot
```


pprof
VIEW
SAMPLE
REFINE
CONFIG
DOWNLOAD
Search regexp
api-server inuse_space
http serverHandler ServeHTTP 0 of -1005.89kB (20.59%)
http newBufioReader 0 of -514kB (10.52%)
http newBufioWriterSize 0 of -514kB (10.52%)
net (*sysDialer) dialSerial 0 of -512.02kB (10.48%)
handlers loggingHandler ServeHTTP 0 of 564.84kB (11.56%)
bufio NewReader 0 of -514kB (10.52%)
bufio NewWriterSize -514kB (10.52%)
bufio NewReaderSize -514kB (10.52%)
http (*ServeMux) ServeHTTP 0 of -1570.73kB (32.16%)
mux (*Router) ServeHTTP 0 of 564.84kB (11.56%)
http HandlerFunc ServeHTTP 0 of -1005.89kB (20.59%)
users Authenticate func1 0 of 564.84kB (11.56%)
mux CORSMethodMiddleware func1 1 0 of 564.84kB (11.56%)
promhttp HandlerForTransactional func1 0 of -544.67kB (11.15%)
promhttp InstrumentHandlerCounter func1 0 of -544.67kB (11.15%)
promhttp InstrumentHandlerInFlight func1 0 of -544.67kB (11.15%)
go-metrics Observer Middleware func1 0 of 564.84kB (11.56%)
main contentWriter func1 0 of 564.84kB (11.56%)
pprof Index 0 of -1026.06kB (21.01%)
users ratelimit 1078.34kB (22.08%)
users (*Service) Authenticate 0 of -513.50kB (10.51%)
expfmt encoderCloser Encode 0 of -544.67kB (11.15%)
pprof handler ServeHTTP 0 of -1026.06kB (21.01%)
users userStore UserExists 0 of -513.50kB (10.51%)
expfmt NewEncoder func7 0 of -544.67kB (11.15%)
pprof handler serveDeltaProfile 0 of -1026.06kB (21.01%)
sqlx (*DB) GetContext 0 of -513.50kB (10.51%)
expfmt MetricFamilyToText 0 of -544.67kB (11.15%)
pprof collectProfile 0 of -1026.06kB (21.01%)
sqlx GetContext 0 of -513.50kB (10.51%)
expfmt MetricFamilyToText func1 0 of -544.67kB (11.15%)
profile Parse 0 of -1026.06kB (21.01%)
sqlx (*DB) QueryRowxContext 0 of -513.50kB (10.51%)
bufio (*Writer) Flush 0 of -544.67kB (11.15%)
profile parseUncompressed 0 of -1026.06kB (21.01%)
sql (*DB) QueryContext 0 of -513.50kB (10.51%)
gzip (*Writer) Write 0 of -544.67kB (11.15%)
profile (*Profile) postDecode
profile unmarshal 0 of -514kB (10.52%)
net (*sysDialer) dialSingle 0 of -512.02kB (10.48%)
net (*sysDialer) dialTCP 0 of -512.02kB (10.48%)
net (*sysDialer) doDialTCP 0 of -512.02kB (10.48%)
net internetSocket 0 of -512.02kB (10.48%)
net socket 0 of -512.02kB (10.48%)
net (*netFD) dial 0 of -512.02kB (10.48%)
net sockaddrToTCP -512.02kB (10.48%)
888kB
4kB
4kB
48B
564.84kB
-1570.73kB
-514kB (inline)
-514kB
-512.02kB
-544.67kB
-1026.06kB
1078.34kB
-513.50kB
-513.50kB (inline)
-512.06kB

Advanced

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

Best Practices

- 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

Call for early adopters / design partners

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

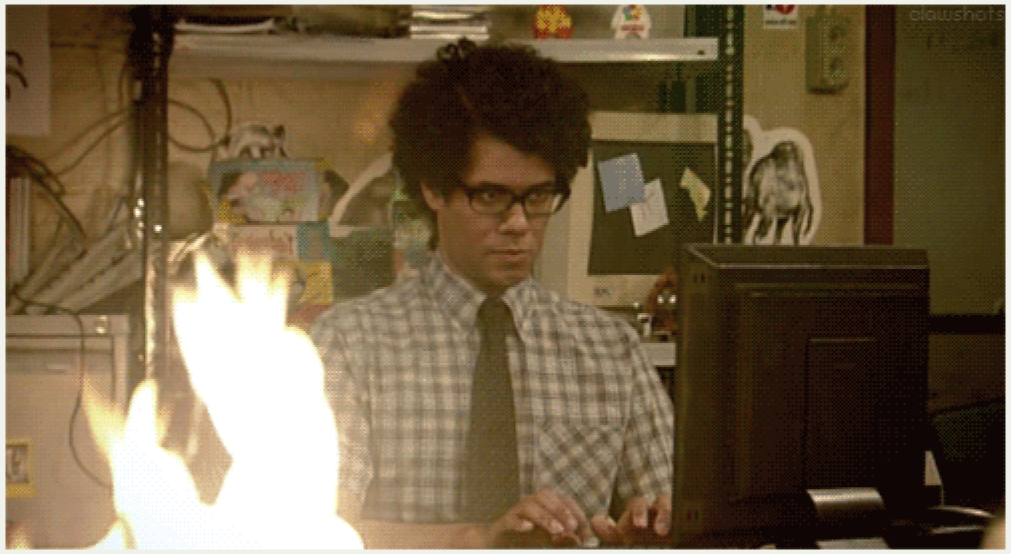
Aiming to help companies prevent downtime and reduce friction with adoption & burnout in devops

relyonmetrics.com

Reference

- Byzantine general's problem
- CAP Theorem
- scalescape/go-metrics
- go offiical pprof pprof
- pprof blog - julia evans

Thanks



Questions

@devdineshkumar

@devdinu

@relyonmetrics