

PRODUCTION READY MICROSERVICES AT SCALE

RAJEEV N B

GO-JEK

XIN CHÀO VIỆT NAM

RAJEEV N BHARSHETTY

- ▶ Product Engineer at GO-JEK
- ▶ Distributed Systems | Security | Data
- ▶ Building reliable and scalable systems
- ▶ **@rbharshetty** - Twitter
- ▶ **@rshetty** - Github

TRANSPORT, LOGISTICS, HYPERLOCAL DELIVERY AND PAYMENTS

- ▶ 18+ products
- ▶ 1 Million+ Drivers
- ▶ 500+ Microservices
- ▶ 15k+ Cores
- ▶ 2 Cloud Providers
- ▶ 6 Data centres
- ▶ 100 Million+ bookings a month



GO-JEK expands to 4 new markets - Thailand, Vietnam, Singapore & Philippines

AGENDA

- ▶ What are **Production Ready** Microservices ?
- ▶ Why do we need them ?
- ▶ How do we build them ?
- ▶ Future work
- ▶ Conclusion

MICROSERVICES

SMALL, AUTONOMOUS SERVICES THAT WORK TOGETHER

Sam Newman
(Building Microservices)

**PRODUCTION
READY**

WHAT DOES PRODUCTION READY MEAN ?

- ▶ Stable
- ▶ Reliable
- ▶ Scalable
- ▶ Performant
- ▶ Fault tolerant
- ▶ Monitored
- ▶ Prepared for Catastrophe
- ▶ Secure



BUILDING TRUST

WHY ?

Goal is to be **Available** to serve
our users

HOW ?

PRODUCTION READINESS CHECKLIST

WHY CHECKLIST ?

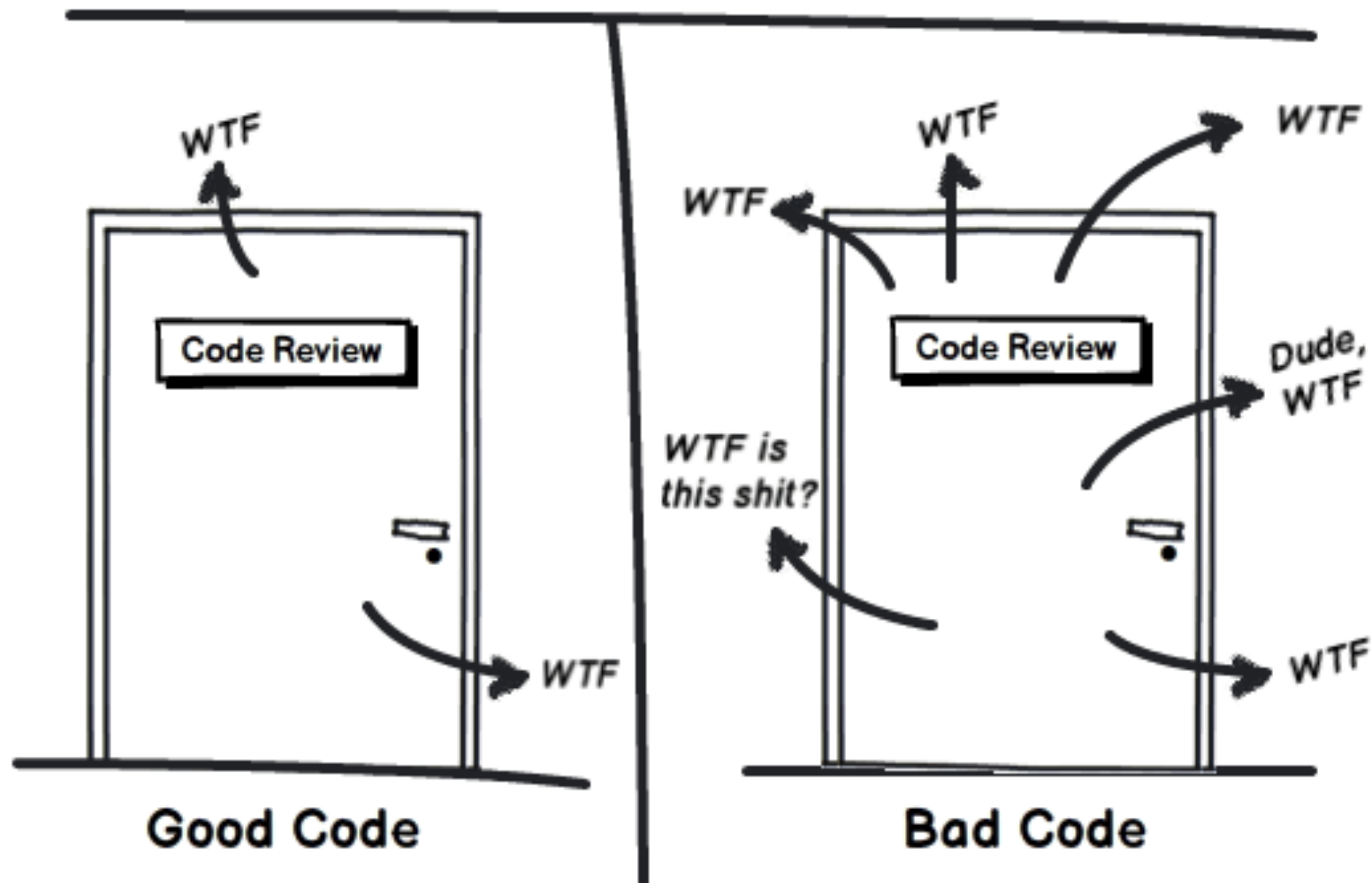
- ▶ **Organisation** Sprawl
- ▶ **Technical** Sprawl
- ▶ Increased **Development Velocity**
- ▶ Building **Trust** and **confidence**

LETS BEGIN ...

#1:

CODE QUALITY

Code Quality Measurement: WTFs/Minute



LINTING/FORMATTING

- ▶ Statically analyse code for lint and format errors
- ▶ Helps write **Idiomatic Go** code
- ▶ Helps Improve readability of the code
- ▶ Makes code **easier to change**
- ▶ Tools used: *"go fmt", "golint", "go vet"*

CODE SMELLS

- ▶ Statically analyse code for *Cyclomatic Complexity*, *DeadCode*, *Duplicated Code*, *ErrCheck* etc
- ▶ Keeps **code quality** in check (**Maintainability**)
- ▶ Helps maintain **Sanity** of the codebase (also of the people reading it)
- ▶ Tools: "**gometalinter**"
- ▶ **Meta Linter** (<https://github.com/alecthomas/gometalinter>)

```
hystrix/hystrix_client_test.go:513:12:warning: ineffectual assignment to err (ineffassign)
hystrix/hystrix_client.go:168::warning: declaration of "err" shadows declaration at hystrix/hystrix_client.go:
163 (vetshadow)
hystrix/hystrix_client.go:178:23:warning: error return value not checked (response.Body.Close()) (errcheck)
hystrix/hystrix_client.go:178::warning: Errors unhandled.,LOW,HIGH (gosec)
hystrix/hystrix_client.go:186::warning: Errors unhandled.,LOW,HIGH (gosec)
hystrix/hystrix_client_test.go:513:12:warning: this value of err is never used (SA4006) (megacheck)
→ heimdall git:(master) x
```

SECURE CODING

- ▶ Inspect source code for **Security** problems
- ▶ Find vulnerabilities like **SQL Injection, Hardcoded credentials** etc
- ▶ Help write secure code
- ▶ Tools: "**gosec**" (<https://github.com/securego/gosec>)
- ▶ Detects problems with various confidence levels

Available rules

- G101: Look for hard coded credentials
- G102: Bind to all interfaces
- G103: Audit the use of unsafe block
- G104: Audit errors not checked
- G105: Audit the use of math/big.Int.Exp
- G106: Audit the use of ssh.InsecureIgnoreHostKey
- G107: Url provided to HTTP request as taint input
- G201: SQL query construction using format string
- G202: SQL query construction using string concatenation
- G203: Use of unescaped data in HTML templates
- G204: Audit use of command execution
- G301: Poor file permissions used when creating a directory
- G302: Poor file permissions used with chmod
- G303: Creating tempfile using a predictable path
- G304: File path provided as taint input
- G305: File traversal when extracting zip archive
- G401: Detect the usage of DES, RC4, MD5 or SHA1
- G402: Look for bad TLS connection settings
- G403: Ensure minimum RSA key length of 2048 bits
- G404: Insecure random number source (rand)
- G501: Import blacklist: crypto/md5
- G502: Import blacklist: crypto/des
- G503: Import blacklist: crypto/rc4
- G504: Import blacklist: net/http/cgi
- G505: Import blacklist: crypto/sha1

```
[/Users/admin/work/go/src/github.com/gojektech/heimdall/httpclient/client.go:137] - G104: Errors unhandled. (Confidence: HIGH, Severity: LOW)  
> response.Body.Close()
```

```
[/Users/admin/work/go/src/github.com/gojektech/heimdall/httpclient/client.go:145] - G104: Errors unhandled. (Confidence: HIGH, Severity: LOW)  
> _, _ = bodyReader.Seek(0, 0)
```

```
[/Users/admin/work/go/src/github.com/gojektech/heimdall/hystrix/hystrix_client.go:178] - G104: Errors unhandled. (Confidence: HIGH, Severity: LOW)  
> response.Body.Close()
```

```
[/Users/admin/work/go/src/github.com/gojektech/heimdall/hystrix/hystrix_client.go:186] - G104: Errors unhandled. (Confidence: HIGH, Severity: LOW)  
> _, _ = bodyReader.Seek(0, 0)
```

Summary:
Files: 8
Lines: 787
Nosec: 0
Issues: 4

CLOSING COMMENTS ...

- ▶ Make it part of your development process (Makefile)
- ▶ Sample Project (<https://github.com/gojektech/heimdall>)
- ▶ Make it part of your **build pipeline** (**CI/CD**)
- ▶ Changes to the codebase becomes easier
- ▶ Helps maintain **Stability** and **Reliability** of a microservice

#2:

TESTING

**Wrote A Bunch Of
Code Without Testing**



Works

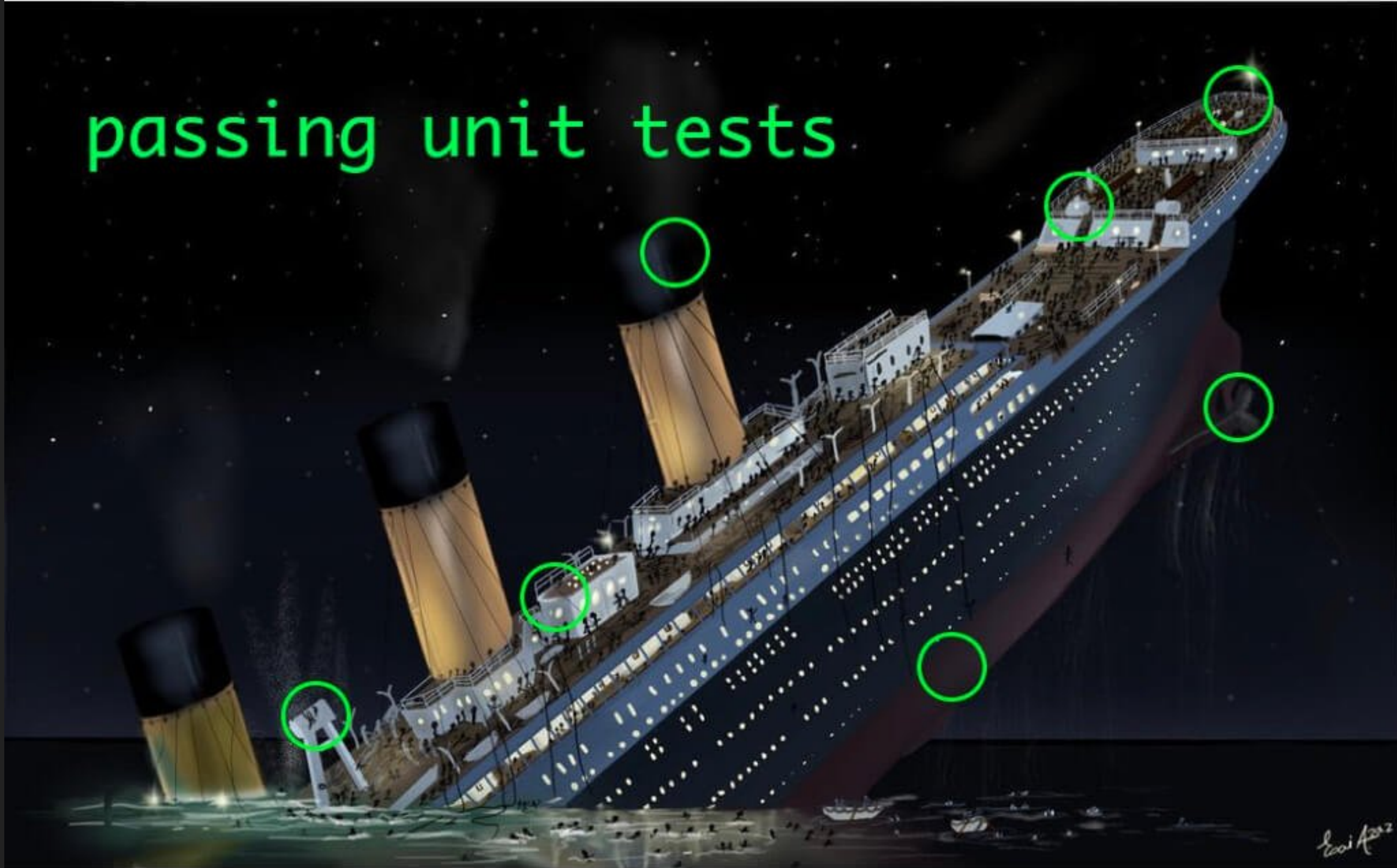
MemeBucket.com

UNIT TESTING

- ▶ Unit tests test a unit/function (Individual component)
- ▶ These are run very often during development
- ▶ Are very large in number (# of unit tests)
- ▶ Other practices include **TDD (Test Driven Development)**
- ▶ Tools: "in-built **Go testing framework**"
- ▶ Assertion library (<https://github.com/stretchr/testify>)

UNIT TESTS SUFFICIENT ?

passing unit tests





Guillaume Malette @gmalette

1h

Replying to @iamdevloper

Running integration tests:

- Ship still afloat after four watertight compartments flooded: PASS
- Band still plays after collision with iceberg: PASS

1 1 ...

INTEGRATION TESTING

- ▶ Integration tests help in testing multiple components working together
- ▶ Cover **End to end** flows
- ▶ Lesser in number than unit tests
- ▶ Helps in finding regressions in the codebase effectively
- ▶ Tools: "**godog**" (BDD)
- ▶ Link: (<https://github.com/DATA-DOOG/godog>)

Feature: customer legacy login

In order to access application resources

As a Customer

I need to login with my credentials

Scenario: Successful login

Given correct credentials for login

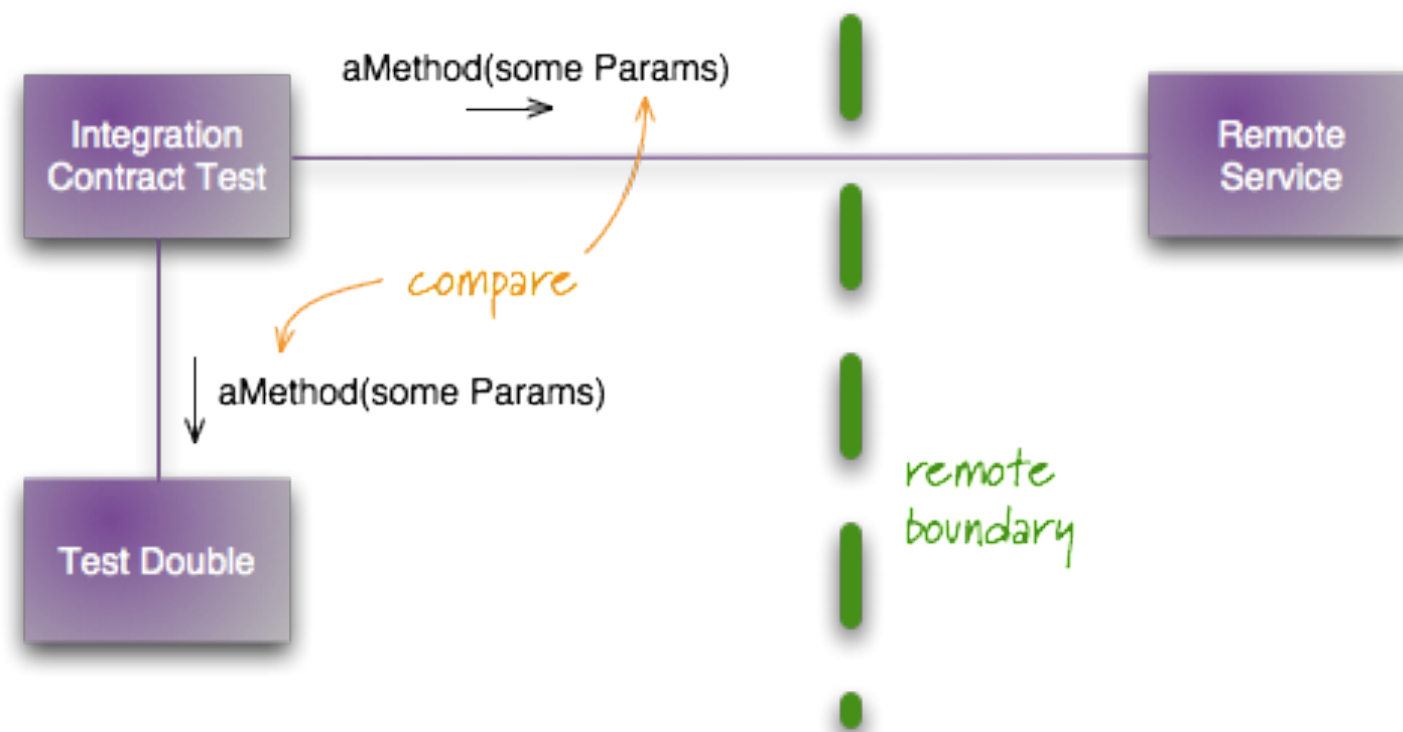
When customer logs in

Then the login response code should be 201

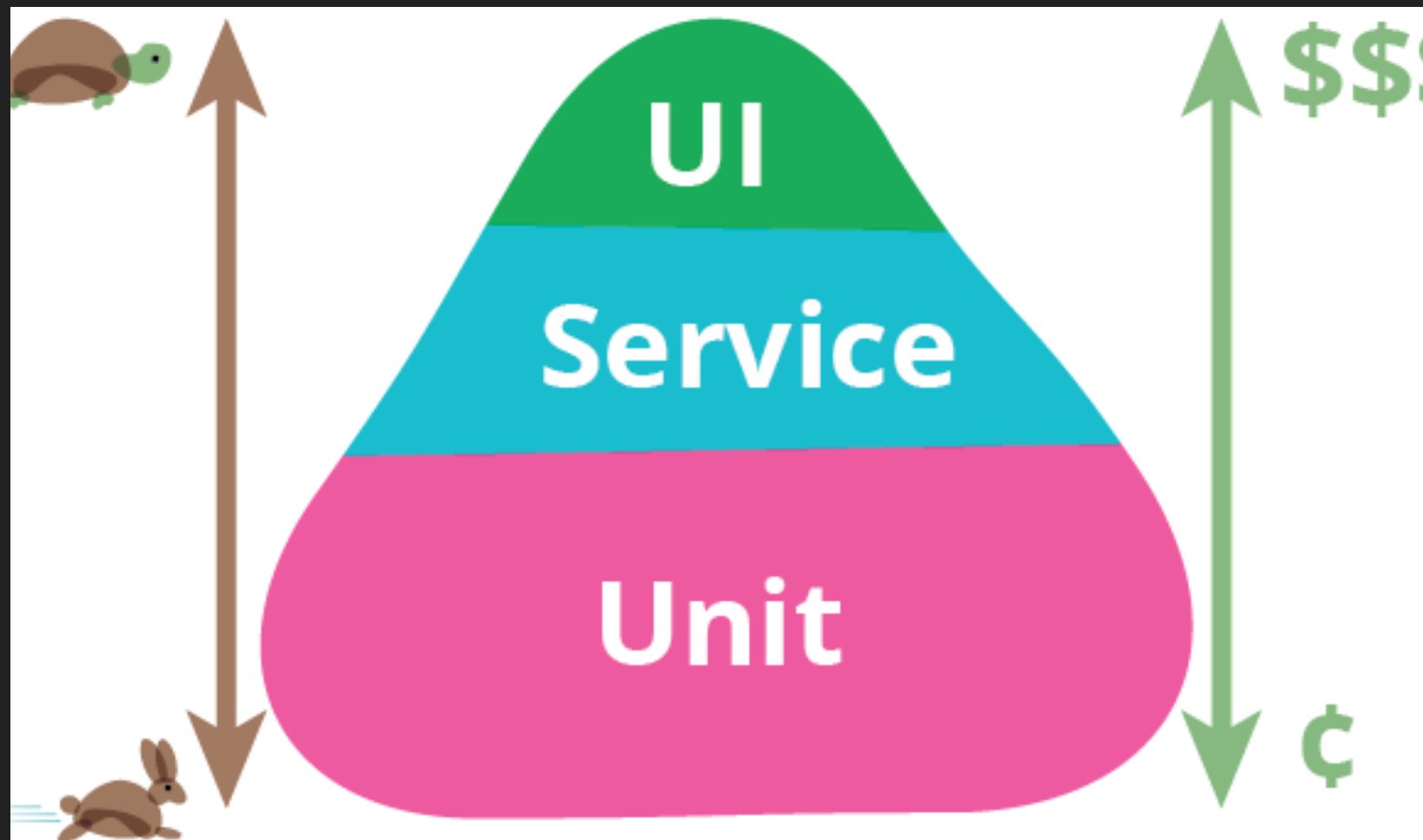
And the login response should have valid details

CONTRACT TESTING

- ▶ Test API contract between a Client(Consumer) and an API(Provider)
- ▶ Helps protect against any change in contracts of the provider
- ▶ Low frequency runs (Once a day)
- ▶ **Consumer Driven Contract Tests**
- ▶ Tools: "**pact-go**"



TEST PYRAMID



LOAD AND PERFORMANCE TESTING

- ▶ Test behaviour of app under various loads
- ▶ Helps find performance bottlenecks
- ▶ Also Memory Leaks
- ▶ Helps with **Capacity Planning** and finding **SLA**
- ▶ Sizing: **Step Load, Linear Load, Spiky load, Soak test**
- ▶ Tools: *"Gatling"*
- ▶ In go: *"Vegeta"* (<https://github.com/tsenart/vegeta>)

```
█
50 rps, 1 min, latency on both services
Requests      [total, rate]      3000, 50.01
Duration      [total, attack, wait] 1m0.217022968s, 59.983753s, 233.269968ms
Latencies     [mean, 50, 95, 99, max] 190.12369ms, 186.887664ms, 257.013489ms, 269.958324ms, 348.796101ms
Bytes In      [total, mean]      527506, 175.84
Bytes Out     [total, mean]      348000, 116.00
Success       [ratio]            99.57%
Status Codes  [code:count]       201:2987  500:13
```

CHAOS TESTING

- ▶ Break systems to understand service behaviour
- ▶ It is ***controlled failure injection***
- ▶ Helps in determining **Unknown Unknowns**
- ▶ It is a kind of **resiliency testing**
- ▶ Manual at GO-JEK
- ▶ Automated - **Netflix Simian Army**
- ▶ **Chaos Monkey, Janitor Monkey, Conformity Monkey**

CLOSING COMMENTS ...

- ▶ Integral part of **Development** process
- ▶ Integral part of your **CI/CD pipeline**
- ▶ Helps build **confidence** in microservice and the codebase
- ▶ Helps move faster (Increases Developer velocity)
- ▶ Helps with **Scalability, Performance, Stability and Reliability**

#3:

RESILIENCE

PATTERNS

TIMEOUTS

- ▶ Stop waiting for an answer after some time
- ▶ **DefaultHttpClient** has infinite timeout (Not suitable for production)
- ▶ Helps prevent/recover from failures when dependences fail/slow down
- ▶ Required at *Integration points*
- ▶ Philosophy of "*Fail Fast*"

HEIMDALL - TIMEOUTS

```
timeout := 100 * time.Millisecond
client := httpclient.NewClient(httpclient.WithHTTPTimeout(timeout))

_, err := client.Get("https://gojek.com/drivers", http.Header{})
```

RETRIES

- ▶ Retry request on failure
- ▶ *Eventually succeed*
- ▶ Could lead to bad experience for users (Latencies)
- ▶ Will help to recover from temporary network glitches
- ▶ **Idempotency is important** when retrying

HEIMDALL - RETRIES

```
backoffInterval := 2 * time.Millisecond
maximumJitterInterval := 5 * time.Millisecond

backoff := heimdall.NewConstantBackoff(backoffInterval,
maximumJitterInterval)
retrier := heimdall.NewRetrier(backoff)

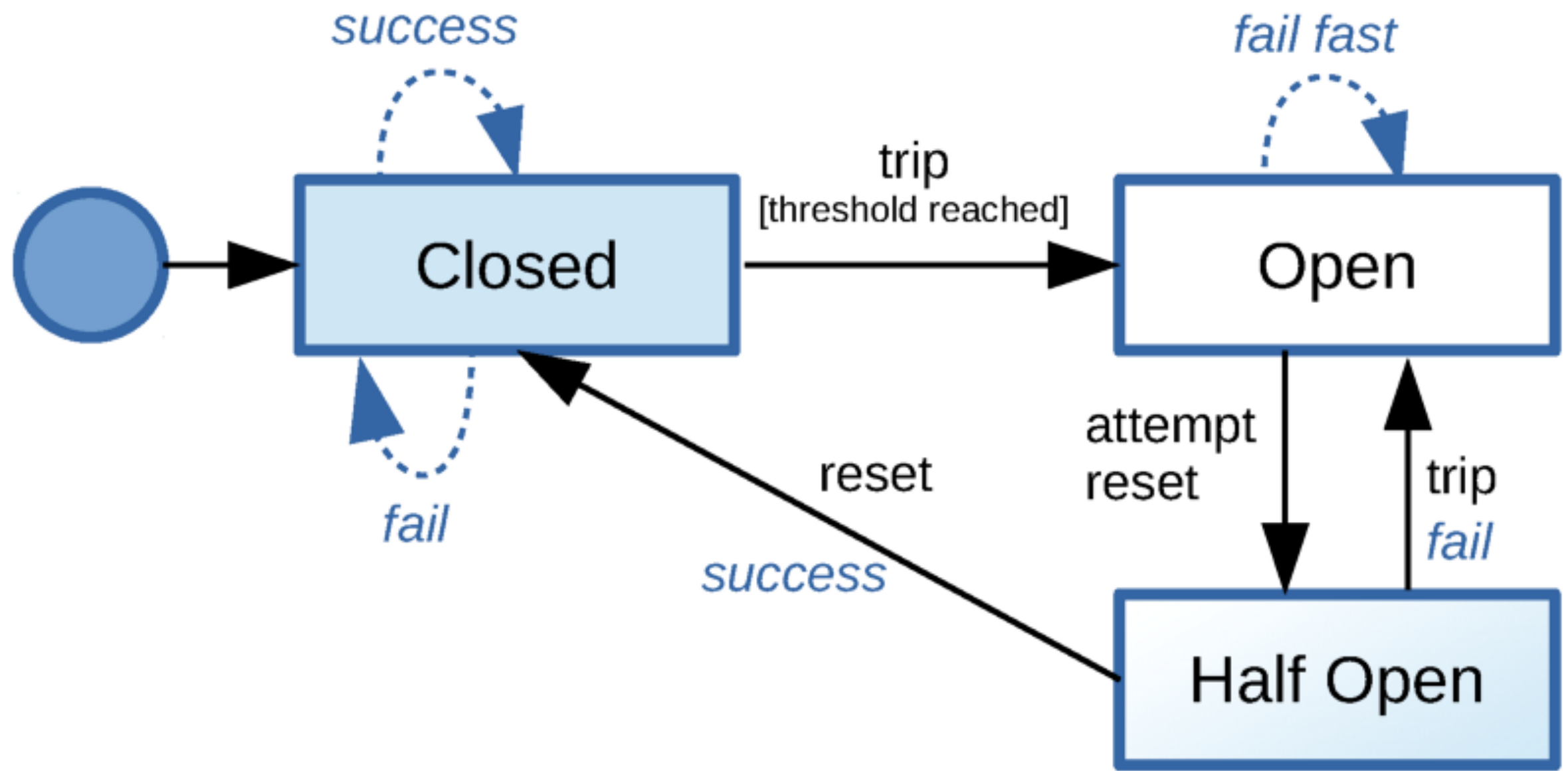
timeout := 1000 * time.Millisecond

client := httpclient.NewClient(
    httpclient.WithHTTPTimeout(timeout),
    httpclient.WithRetrier(retrier),
    httpclient.WithRetryCount(4),
)

client.Get("https://gojek.com/drivers", http.Header{})
```

CIRCUIT BREAKER

- ▶ Circumvent calls when calls to dependency on failure
- ▶ Once circumvented, fallback to fallback response
- ▶ ***Fail fast*** to protect dependent services
- ▶ ***Hystrix*** is a well known implementation
- ▶ ***Recovers back*** on healthy system behaviour



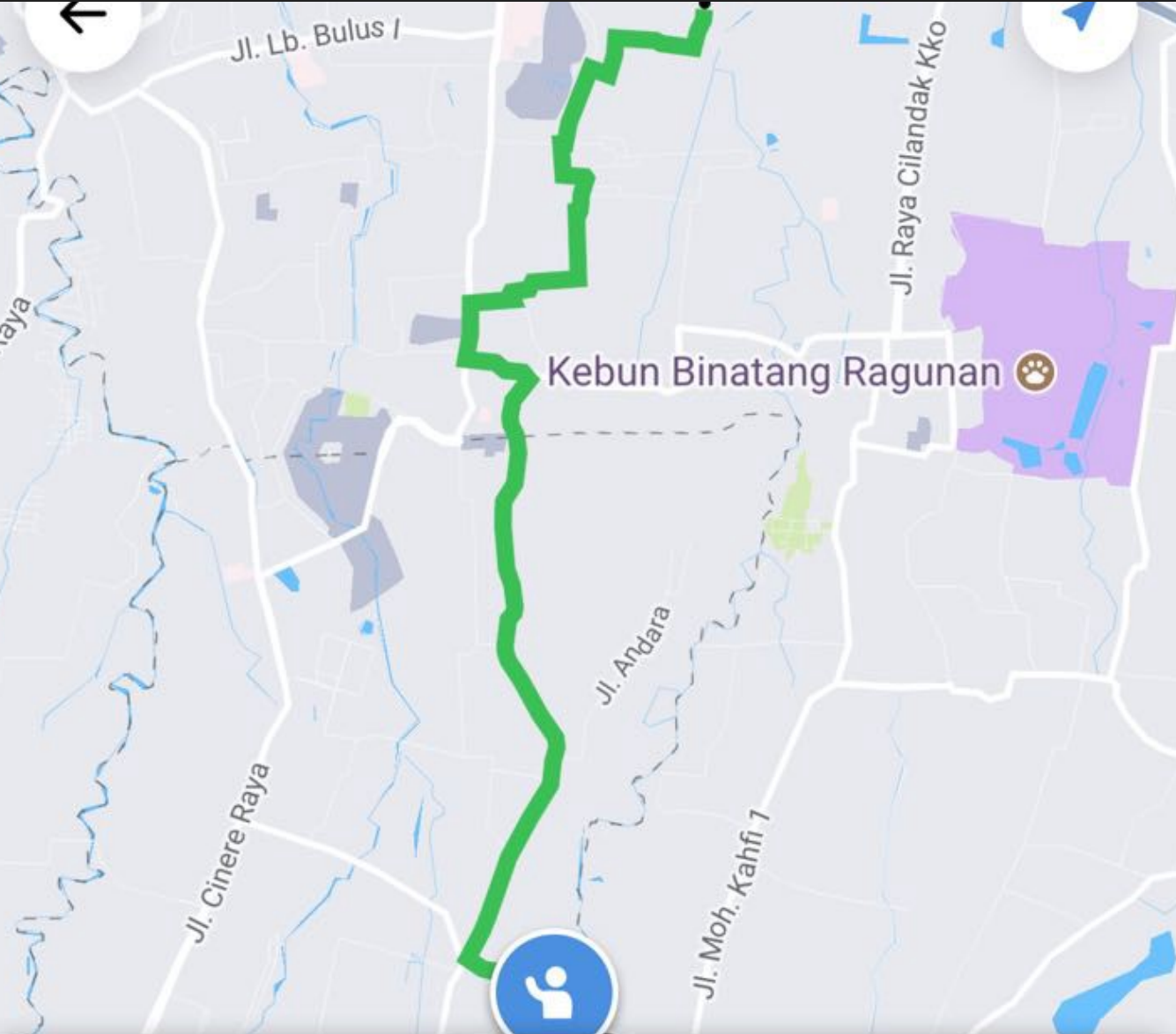
HEIMDALL - CIRCUIT BREAKER



```
fallbackFn := func(err error) error {
    _, err := http.Post("post_to_channel_two")
    return err
}


timeout := 10 * time.Millisecond



client := hystrix.NewClient(
    hystrix.WithHTTPTimeout(timeout),
    hystrix.WithCommandName("MyCommand"),
    hystrix.WithMaxConcurrentRequests(100),
    hystrix.WithErrorPercentThreshold(20),
    hystrix.WithSleepWindow(10),
    hystrix.WithRequestVolumeThreshold(10),
    hystrix.WithFallbackFunc(fallbackFn),
)



_, err := httpClient.Get("https://gojek.com/drivers", http.Header{})
```






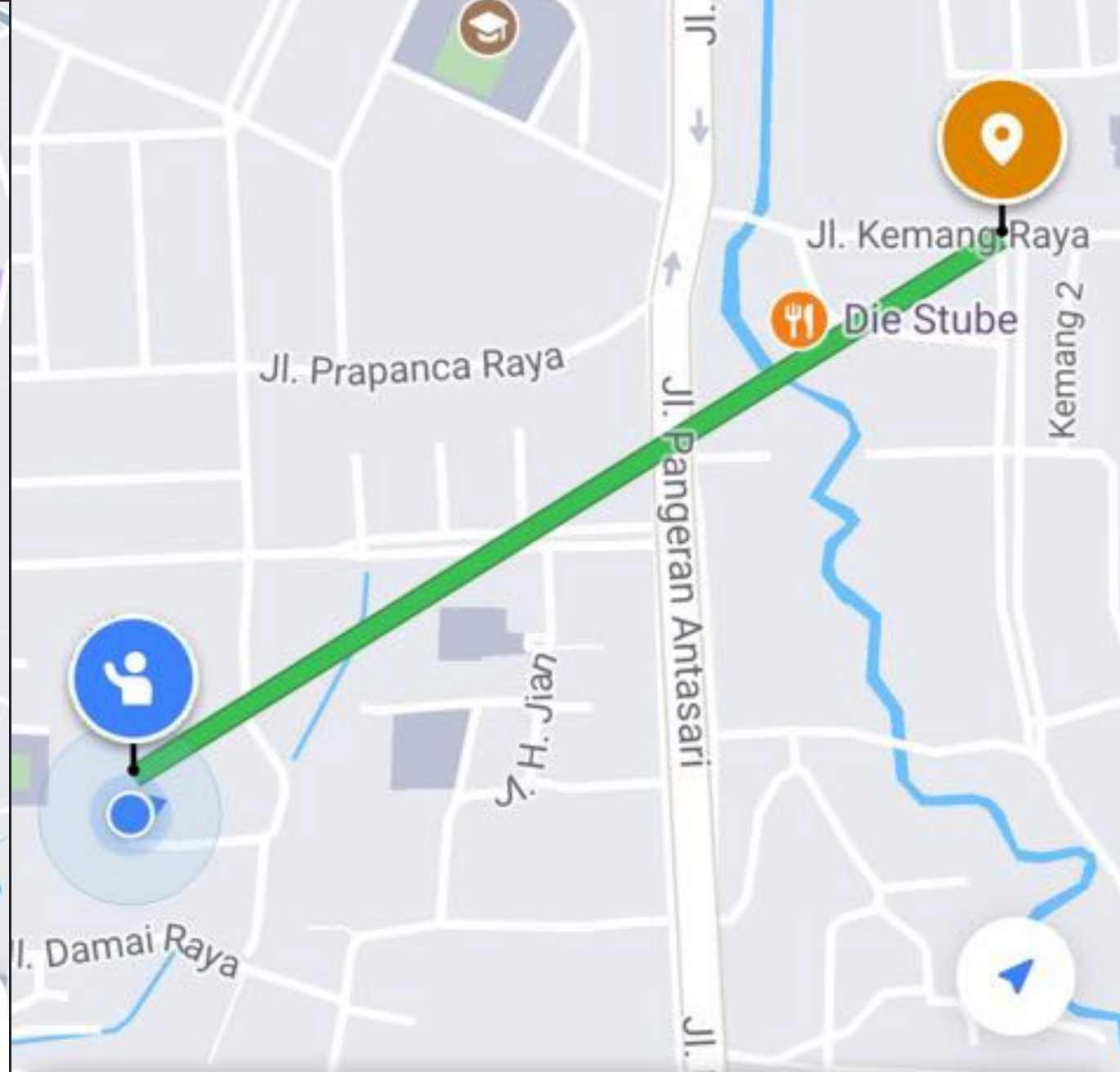
 **GO-CAR** 


1-4 PEOPLE 


 PICKUP LOCATION
Jl. Sawo No.36 



 DESTINATION • 8.1 km
Jl. Lap. Tembak No.25 



 **GO-PAY** **Rp39.000** 



 **GO-RIDE**

1 PERSON 

 PICKUP LOCATION
Essence Darmawangsa Apartment 

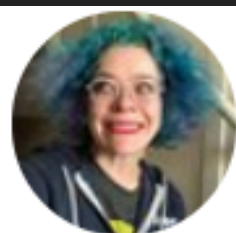
 DESTINATION • 0.7KM
grandkemang Jakarta 

CLOSING COMMENTS ...

- ▶ Protects systems and also its dependencies from failure
- ▶ **FailFast** to protect systems
- ▶ Needed at the integration points of your system
- ▶ **Think of Fallbacks** at your integration points (Graceful degradation)
- ▶ Helps with **Scalability, Stability and Reliability**

#4:

OBSERVABILITY



Charity Majors

@mipsytipsy

Follow



Replying to @mipsytipsy @mattklein123

Observability, otoh, is about being able to understand the inner workings of your software and systems by asking questions and observing the answers on the outside. Any question — no particular bias toward actionable alerts or problems.

OBSERVABILITY BROADLY INCLUDES:

MONITORING

LOGGING

~~DISTRIBUTED TRACING~~

MONITORING

**Monitoring is the act of checking
the behaviour and outputs of a
system and its components**

MONITORING

- ▶ It is all about looking out for **presence/absence of patterns**
- ▶ Is used to report **overall health** of the system
- ▶ Includes **Key Business and System Level Metrics**
- ▶ Think **USE** (Utilisation, Saturation and Errors)
- ▶ Think **RED** (Response times, errors, duration)

KEY METRICS

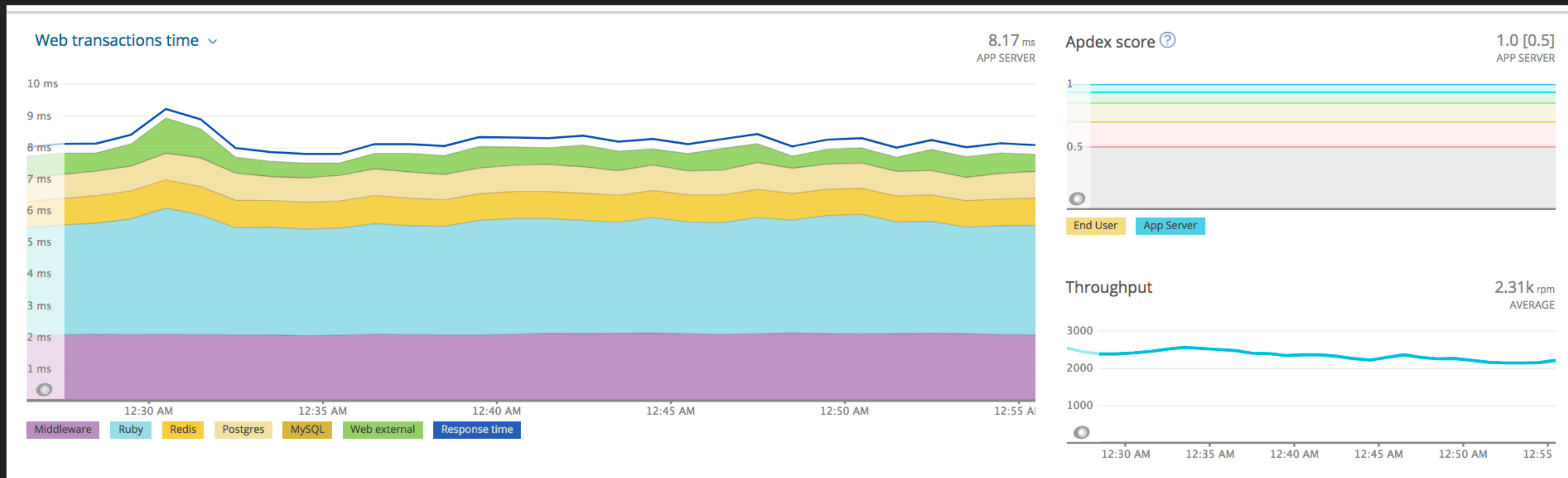
- ▶ Host and Infrastructure metrics
 - ▶ CPU
 - ▶ RAM
 - ▶ Threads
 - ▶ File Descriptors
 - ▶ Database connections

KEY METRICS

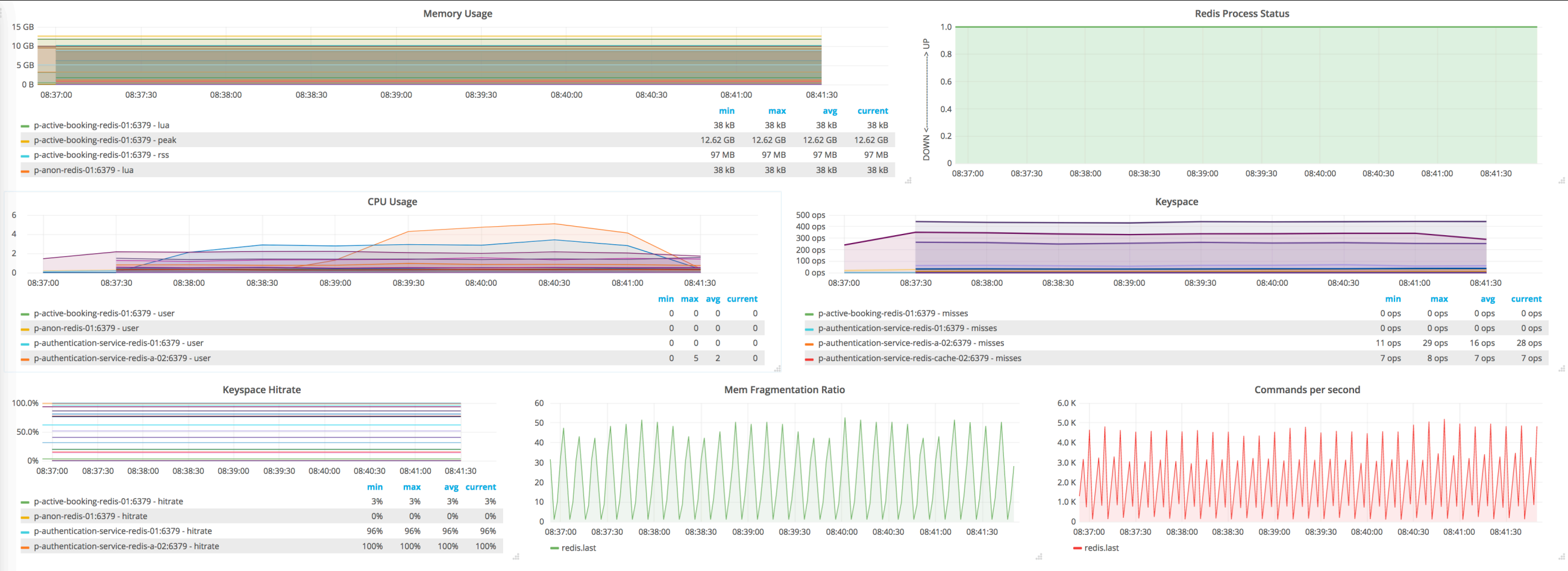
- ▶ Microservice Key Metrics
 - ▶ Availability
 - ▶ SLA
 - ▶ Latency
 - ▶ Success
 - ▶ Errors

DASHBOARD

- ▶ Reflect the state and **health of your system**
- ▶ Should capture key metrics
- ▶ **BusinessMetrics, UptimeMetrics, SystemMetrics**
- ▶ Grafana/NewRelic at GO-JEK



APM (APPLICATION PERFORMANCE MONITORING)



SYSTEM LEVEL METRICS (GRAFANA) – TICK STACK



Day

99.97%

▼ 0.02%

Week

99.94%

▼ 0.02%

Month

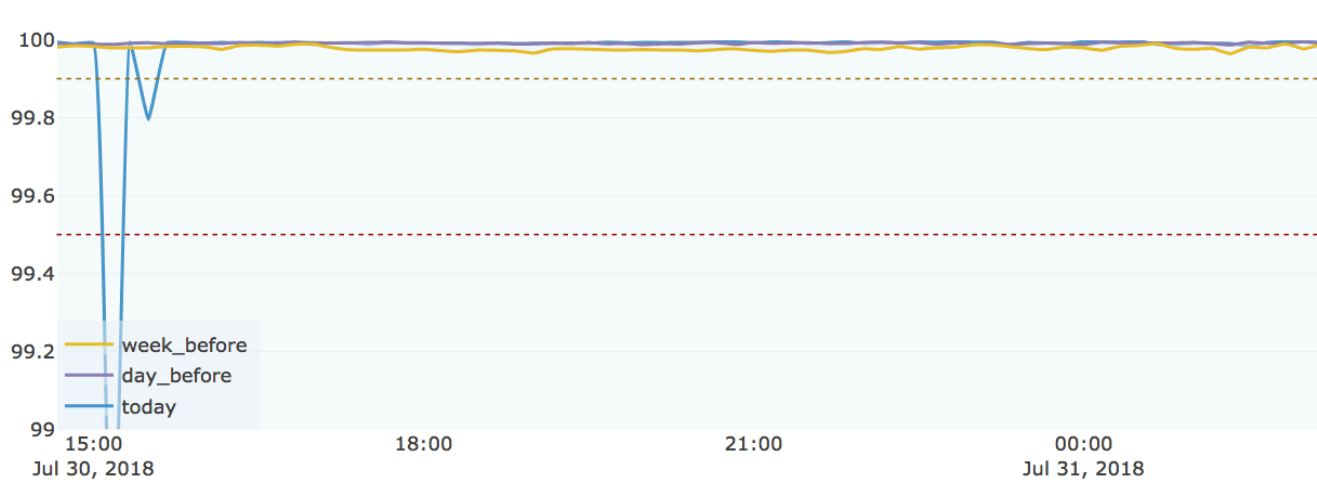
99.95%

▲ 0.07%

Quarter

99.94%

▲ 99.94%



Go-Food

99.94%

▼ 0.01%

■ ■ ■ ■ ■

Driver Platform

99.95%

▼ 0.03%

■ ■ ■ ■ ■

Transport

99.99%

—

■ ■ ■ ■ ■

Allocations

99.99%

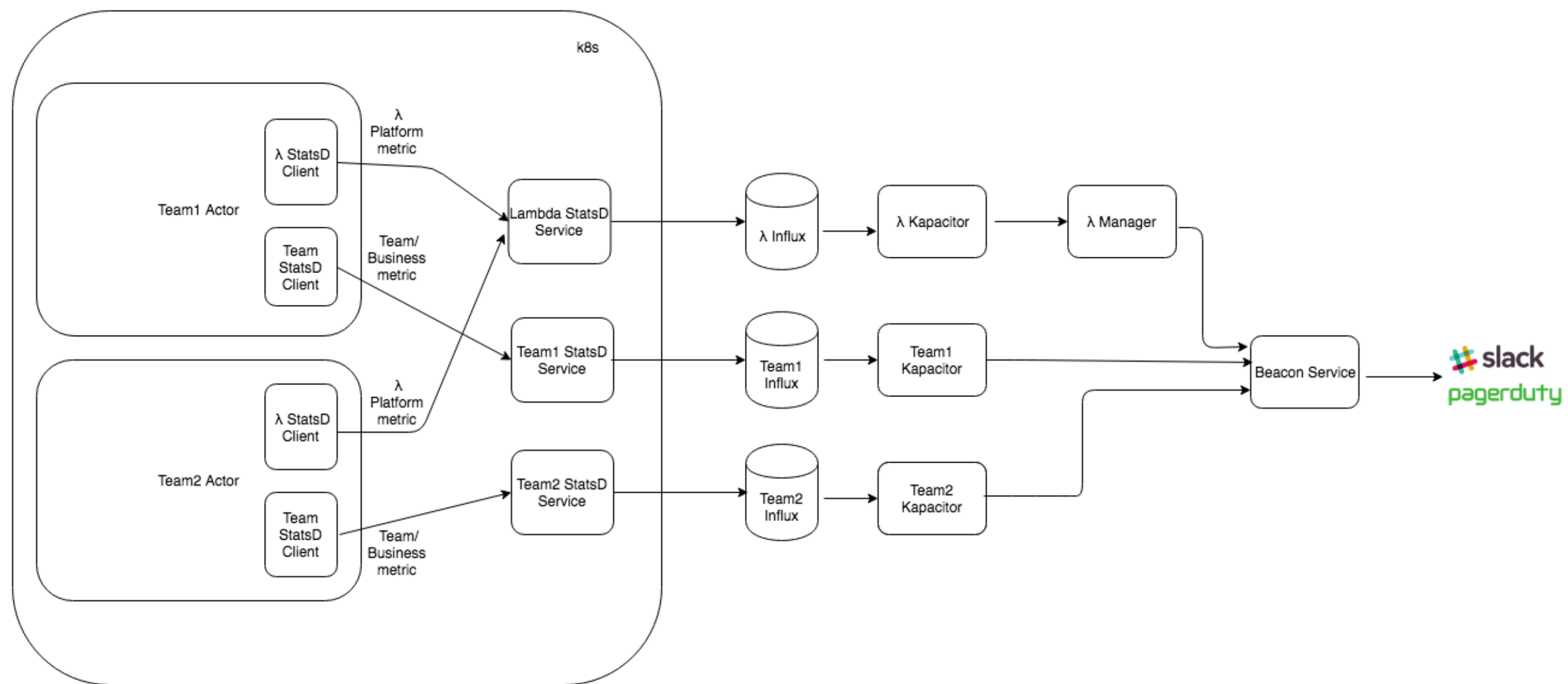
—

■ ■ ■ ■ ■

UPTIME MONITORING – INHOUSE

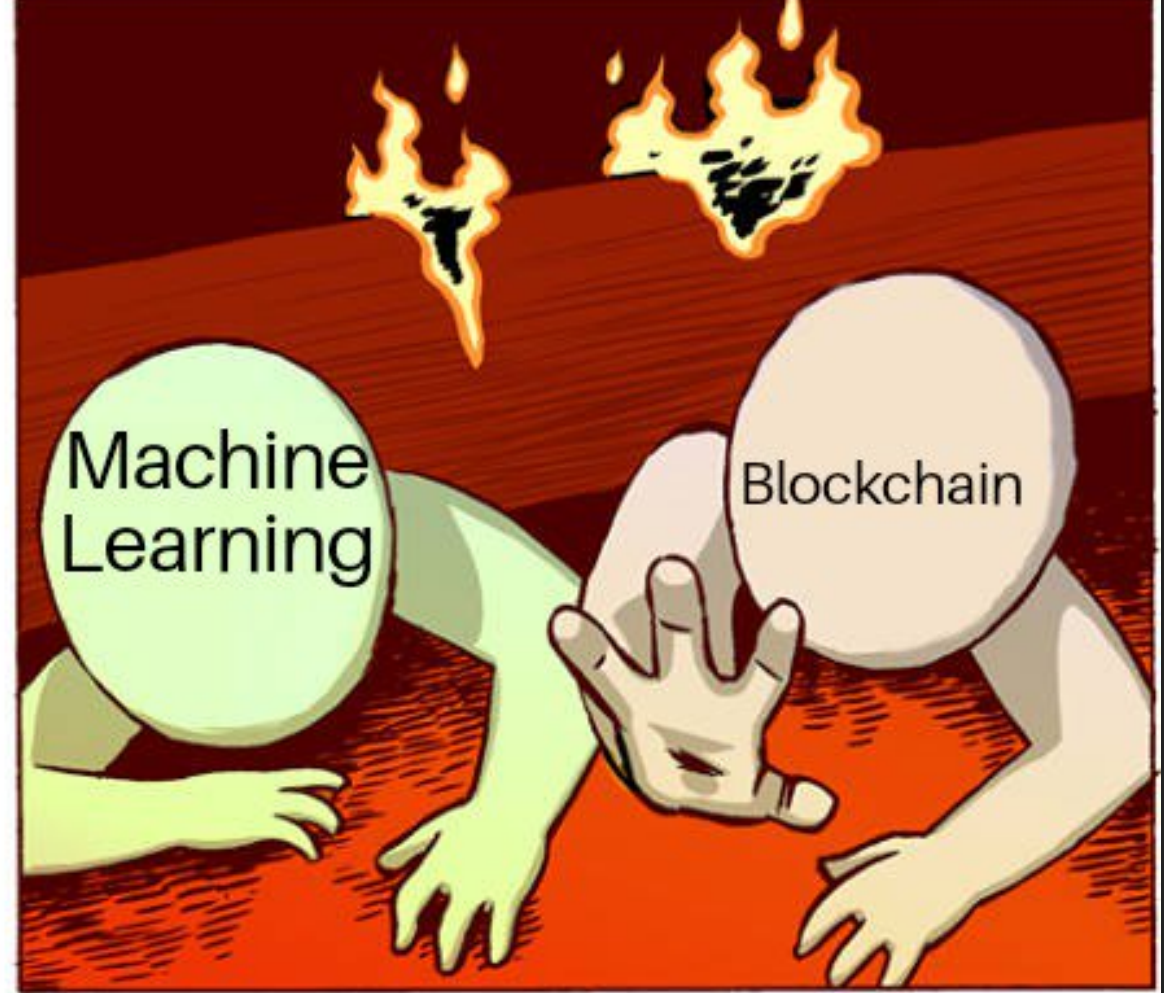
ALERTING

- ▶ Notifying team when an anomaly in key metrics is detected
- ▶ **Thresholds for various key metrics** is set for alerting
- ▶ Alerts should be **actionable**
- ▶ Help in alerting teams and recovering before a catastrophe
- ▶ **TICK** stack (Telegraf, Influx, Chronograph, Kapacitor)



StatsD Instrumentation of λ actors in k8s

LOGGING

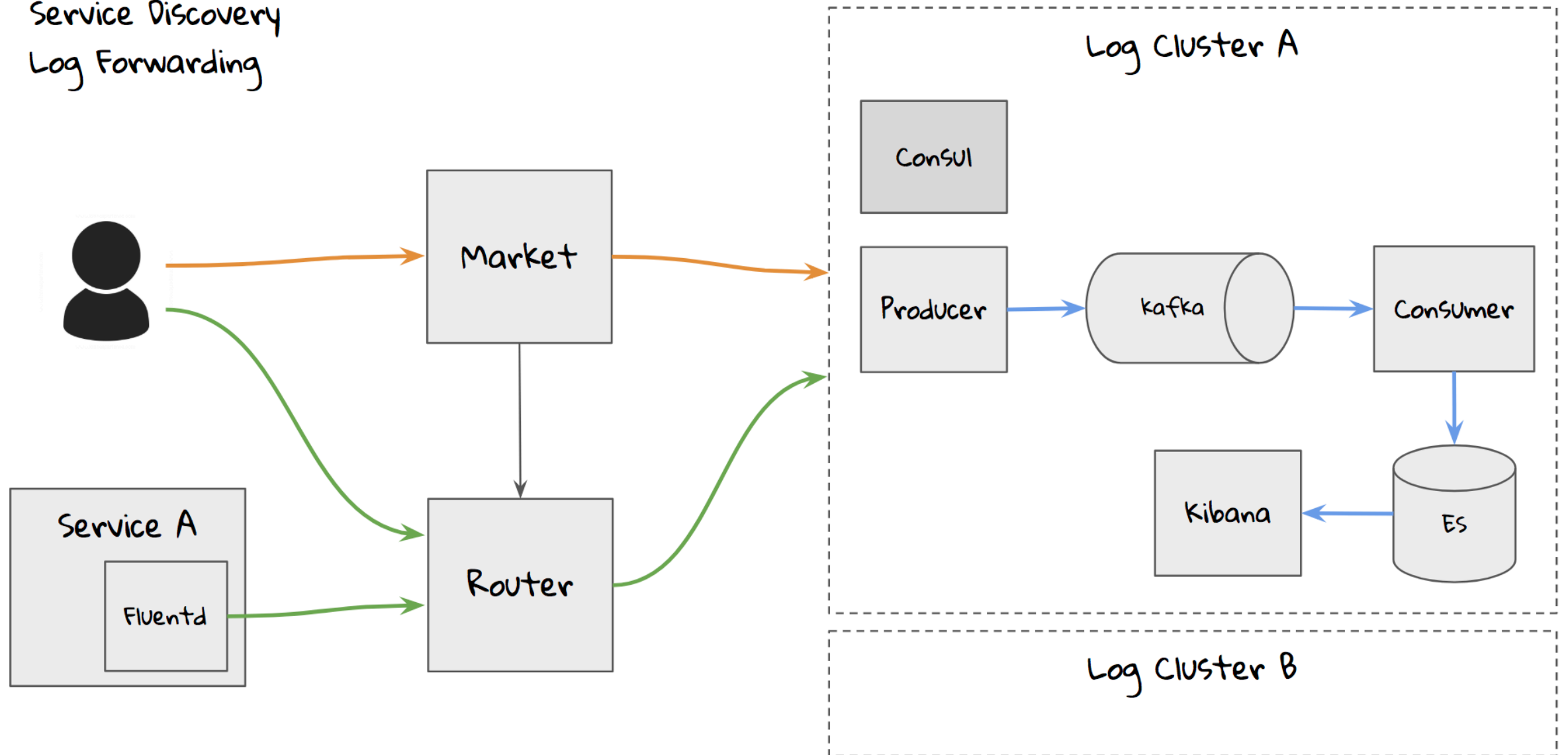


LOGGING

- ▶ Logging helps ***describe system state*** at any point in time
- ▶ Helps in debugging problems with the system
- ▶ Structured logging
- ▶ **Log centralisation, aggregation and analysis** is the key
- ▶ ***Barito-Log*** (github.com/BaritoLog)

Barito Overview

- Provisioning
- Service Discovery
- Log Forwarding



INSTRUMENTATION OF CODE

- ▶ Ability to measure various aspects of system behaviour
- ▶ Code needs to be written to monitor specific components
- ▶ It includes **Structured logging**, **StatsD metrics** (Counters, Gauges, Histograms) and **Error/Exception tracking (Sentry)**
- ▶ Should also be able to capture **Application key performance metrics (NewRelic)**

```
type StatsD struct {  
    client *statsd.Client  
}  
  
func (reporter *StatsD) Incr(key string, tags []string) {  
    if reporter.client != nil {  
        reporter.client.Incr(key, tags, 1)  
    }  
}  
  
reporter.Incr("customers.login.count", []string{"login"})
```

CLOSING COMMENTS ...

- ▶ Insights into **Service Health** and **Behaviour**
- ▶ Insights into **Service Performance**
- ▶ Helps providing **Debuggability in Production**
- ▶ Help bring visibility into system
- ▶ Knowing when things go wrong
- ▶ Helps with **Stability** of the system

#5:

DOCUMENTATION

API DOCS

- ▶ Document **API contracts** of the service
- ▶ Includes **Request, Response and Header** contracts
- ▶ Should provide enough information for integrating with your API
- ▶ Specified using **OpenAPI/Swagger** specification
- ▶ Tools: "**swagger**"

SWAGGER SPECIFICATION

Schemes

HTTPS

▼

default



POST	/login/request	Acquire new OTP for login purpose
POST	/token	Acquire new access token either from otp or from refresh token
DELETE	/token	Logout

ARCHITECTURAL DECISION RECORDS (ADR)

- ▶ Document to capture important architectural *decisions and design*
- ▶ Should be **versioned**
- ▶ It also captures **Context** and **Assumptions**
- ▶ Helps in understanding why some **key architectural decisions** were made
- ▶ Helps **alignment** across teams

SAMPLE ADR

1. Record architecture decisions

Date: 2018-03-06

Status

Accepted

Context

We need to record the architectural decisions made on this project.

Decision

We will use Architecture Decision Records, as described by Michael Nygard in this article: <http://thinkrelevance.com/blog/2011/11/15/documenting-architecture-decisions>

Consequences

See Michael Nygard's article, linked above. For a lightweight ADR toolset, see Nat Pryce's *adr-tools* at <https://github.com/npryce/adr-tools>.

RUNBOOKS

- ▶ Help **on-call developer** to have **step-by-step instruction** to **find, resolve and mitigate** the root cause of an alert
- ▶ **Host level** and **Infrastructure** alerts (General to org)
- ▶ **On-Call** Runbooks (Microservice specific)
- ▶ Helps in **reliability** of a service
- ▶ Helps in preventing **Catastrophe**

SAMPLE RUNBOOK

- ▶ **Booking History service** is down
 - ▶ Check the process status for the service (Dashboard)
 - ▶ Check the **throughput** on the APM dashboard
 - ▶ If no requests, log in to boxes and **restart the process**

CLOSING COMMENTS ...

- ▶ Brings **Clarity in Thought**
- ▶ Helps in **Incidence Response** during outages
- ▶ Helps in **alignment** of various stakeholders and the team
- ▶ Helps with **Catastrophe Preparedness, Stability and Reliability**

OTHERS

- ▶ Security
- ▶ Canarying (Test in production)
- ▶ CI/CD (Stable deployment process)
- ▶ Dependency checks
- ▶ HA on components (Postgres, Redis etc)

WRAPPING UP

- ▶ **Code quality**
- ▶ **Testing**
- ▶ **Resilience Patterns**
- ▶ **Observability**
- ▶ **Documentation**

TAKE AWAYS

- ▶ Code Health Metrics dashboard
- ▶ **Observability** driven Development
- ▶ Test Pyramid
- ▶ **Document** Everything (Trust me, it helps!)
- ▶ Think of **failures** always and protect against them

IN CONCLUSION...

**STANDARDISING
QUALITY
IS THE GOAL**

**AVAILABILITY
IS THE GOAL**

WHAT NEXT ?

- ▶ **Automating** the process of production readiness
- ▶ Defining a **production readiness score**
- ▶ Assigning every microservice a **readiness score**
- ▶ Help teams improve quality of microservices in Production
- ▶ **Measure** the impact and **improve**

REFERENCES

- Production Ready Microservices - Susan Fowler
- Google SRE Book
- Microservices Standardisation
- Resiliency in Distributed Systems
- TICK stack
- SLA



THANKS FOR
LISTENING

QUESTIONS?