Continuous Deployment with Kubernetes basics

Wojciech Barczyński
wbarczynski+trainings@gmail.com

START

- minimum deployment, service, configmap, and secrets
- learn-as-you-go

START

1. Structured logs
2. Liveness
3. Liveness and readiness
4. Prometheus metrics
5. Zero-downtime deployment

12FACTOR APPS

- Good to know what 12factorapp is.
- 12factor is not new, so you might find points obvious.

12FACTOR APPS

Few that are very relevant for Kubernetes:

1. III. Config - environment variable, injected by environment
2. VIII. Concurrency by process
3. IX. Disposability

12FACTOR APPS

4. X. Dev/Prod parity including time, tooling, and backing services
5. XI. Logs - push your logs to stdout and let log routers to take care of them

Continuous Deployment

- start at the
- see Continuous Deployment slides

Probes

- liveness - am I dead?
- readiness - can I serve requests?

slides on zero-downtime deployments

Probes

- deployment, statefulsets, daemonsets
- self-healing

Liveness Probe

```yaml
livenessProbe:
  httpGet:
    path: /model
    port: 8000
    httpHeaders:
      - name: X-Custom-Header
        value: Awesome
  initialDelaySeconds: 600
  periodSeconds: 5
  timeoutSeconds: 18
  successThreshold: 1
  failureThreshold: 3
```

k8s docs

Liveness Probe

- pod gets restarted
- too many restarts = CrashLoop

k8s docs

Liveness Probe

- I am dead
- My dependencies are dead → Readiness probe

Liveness Probe

- Good to detect that we do not respond
- We can actively set it up (less common)

Readiness Probe

```yaml
readinessProbe:
  exec:
    command:
    - cat
     - /tmp/healthy
  initialDelaySeconds: 5
  periodSeconds: 5
```

k8s docs

Readiness Probe

- temporary cannot server traffic

Cannot Serve Traffic

- loading or reloading ML model
- stop getting requests before shutdown
- our deps are dead*

[*] after retries, usually the app kills itself.

Probes best practices

- lightweight
- SHOULD NOT call external components

Pod lifecycle

Zero-downtime deployments

Your app should on stop

1. we get SIGTERM signal

Your app should on stop

1. we get SIGTERM signal
2. app MUST make readinessProbe fail

Your app should on stop

1. we get SIGTERM signal
2. app MUST make readinessProbe fail
3. k8s does not send more requests

Your app should on stop

1. we get SIGTERM signal
2. app MUST make readinessProbe fail
3. k8s does not send more requests
4. app shuts down gracefully

Your app should on stop

1. we get SIGTERM signal
2. app gives 500 on readinessProbe
3. k8s does not send more requests
4. app shuts down gracefully
5. kubernetes forces kill if 30s limit exceeded

GRACEFUL SHUTDOWN

```go
func (s *Service) prepareShutdown(h Server) {
	signal.Notify(s.Stop, os.Interrupt, syscall.SIGTERM)
	<-s.Stop
	s.StatusNotReady()
	shutdown(h)
}
```

from missy

Deployment Strategies

Demos and examples of the implementation:

zero-downtime deployments

Observability

Observability

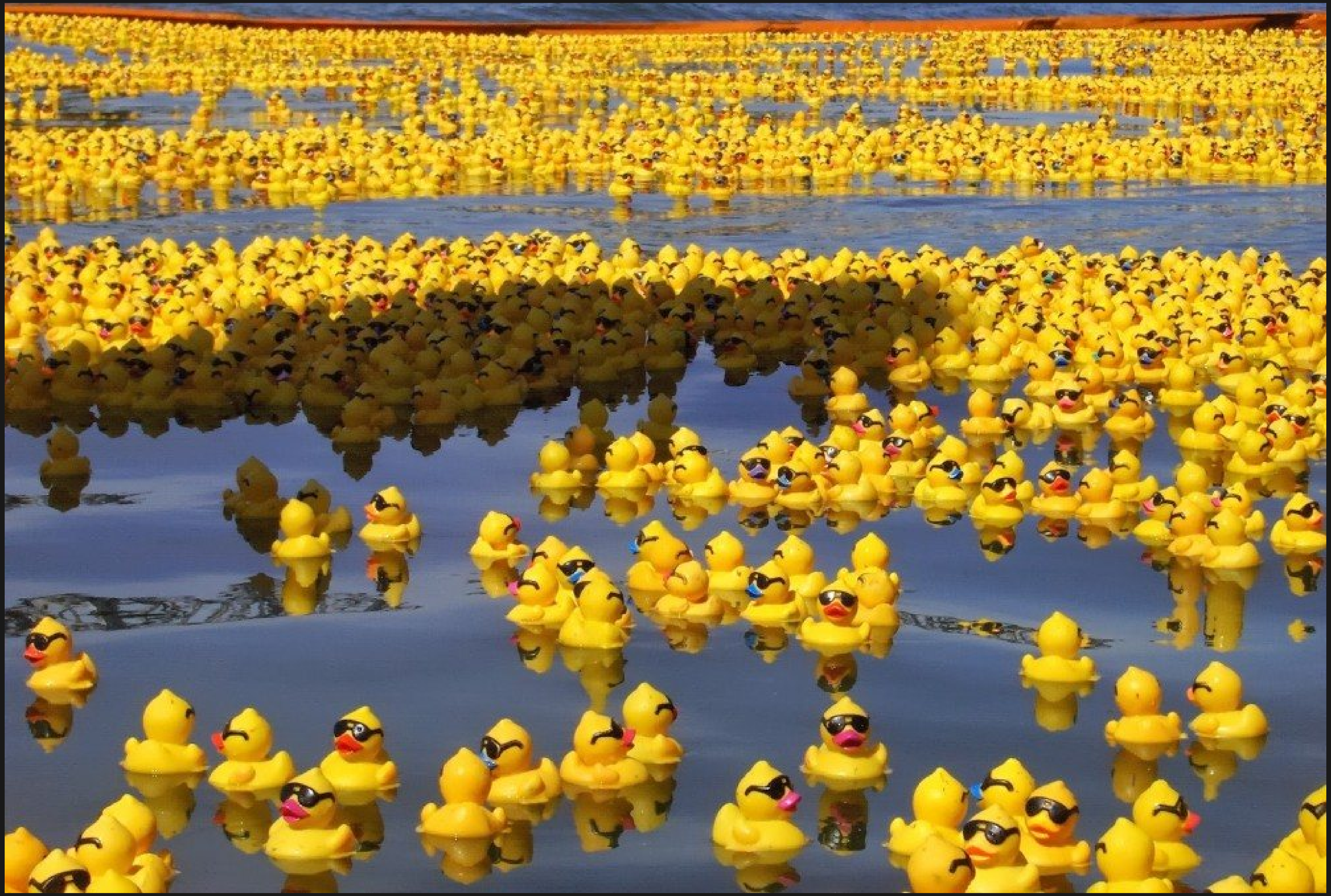
- logging / sentry
- monitoring
- tracking

WHY?

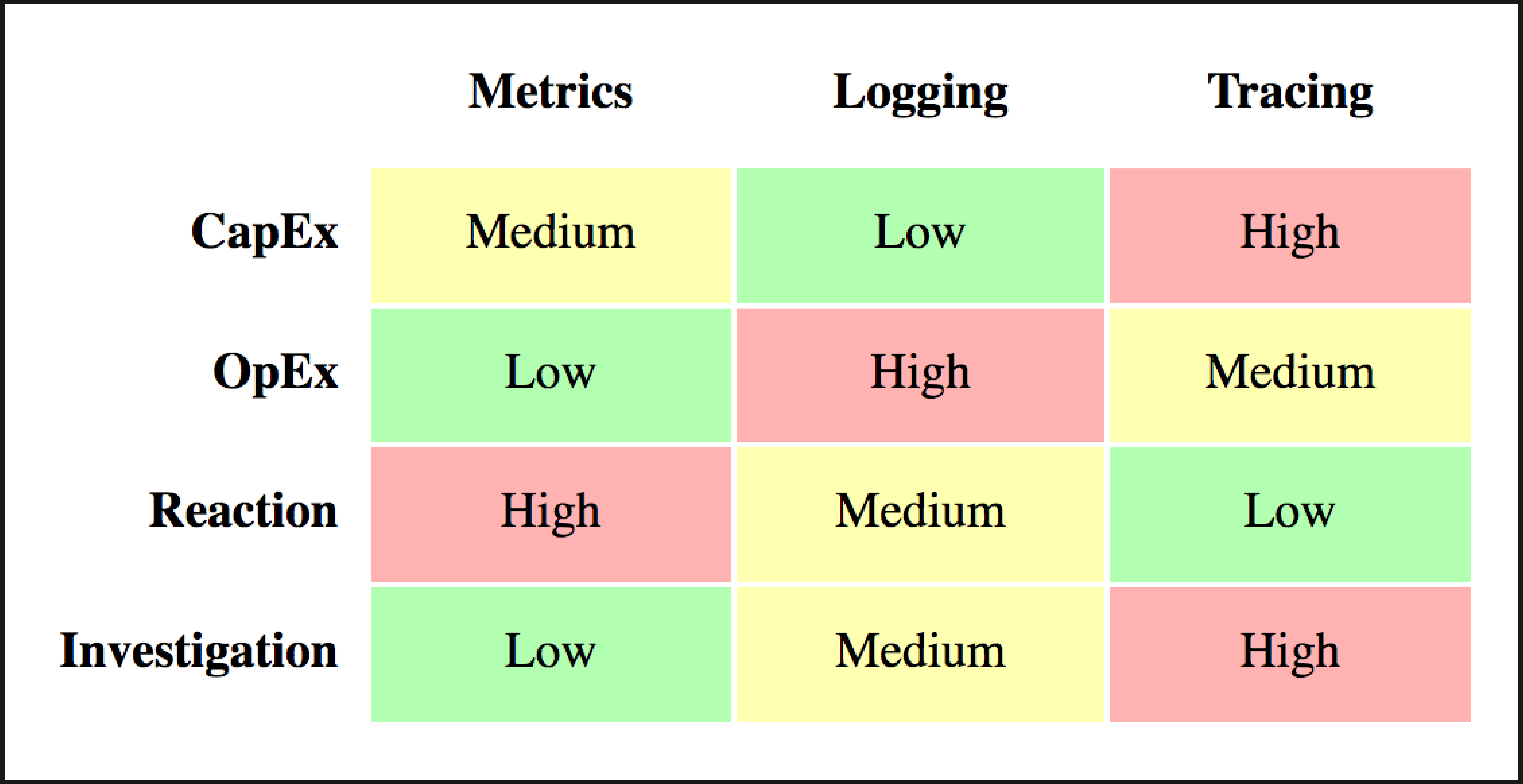
MONOLIT ;)

WHY?

MICROSERVICES ;)



Observability



	Metrics	Logging	Tracing
CapEx	Medium	Low	High
OpEx	Low	High	Medium
Reaction	High	Medium	Low
Investigation	Low	Medium	High

Go for Industrial Programming by Peter Bourgon

App Logs

Before we dive into observability.

App Logs

- structured
- support setting LOG_LEVEL[*] per env

[*] Deployment manifest.

Logs - Golang

- logrus
- uber-go/zap

Pass logger in the context.

Logs - Python

- structlog
- loguru

Fieds

What do you think?

- url, retries, error code
- ?

Centralized Logging

- ElasticSearch
- Fluentd
- Kibana

Worth following: Loki

Centralized Logging

- Debugging tool
- Post-mortem
- Finding the needle
- High TCO

Sentry

Monitoring

SaaS / Cloud

METRIC

Name	Label	Value
traefik_requests_total	code="200", method="GET"	3001

from slides

HOW TO FIND THE RIGHT METRIC?

- RED
- USE

HOW TO FIND THE RIGHT METRIC?

- Measure from the client experience
- Alert on the client experience

LOW LEVEL METRICS

- It is not onPremise :)
- Yes, if they correlate with experience [*]

[*] see saturation in 4 Golden signals

RED

Rate	How busy is your service?
Error	Errors
Duration	What is the latency of my service?

Tom Wilkie's guideline for instrumenting applications.

RED

- **Rate** - how many request per seconds handled
- **Error**
- **Duration** (distribution)

RED

- **Rate**
- **Error** - how many request per seconds handled we failed
- **Duration**

RED

- **Rate**
- **Error**
- **Duration** - how long the requests took

RED

- Follow Four Golden Signals by Google SREs [1]
- Focus on what matters for end-users

[1] Latency, Traffic, Errors, Saturation (src)

RED

Not recommended for:

- batch-oriented
- streaming services

USE

Utilization	the average time that the resource was busy servicing work
Saturation	extra work which it can't service, often queued
Errors	the count of error events

Documented by Berdan Gregg

USE

- **U**tilization: as a percent over a time interval: "one disk is running at 90% utilization".
- **S**aturation:
- **E**rrors:

USE

- **Utilization**:
- **Saturation**: as a queue length, e.g., "the CPUs have an average run queue length of four".
- **Errors**:

USE

- **utilization**:
- **saturation**:
- **errors**: scalar counts, e.g.,
  "this network interface drops packages".

USE

- **traditionaly** more instance oriented
- still useful in the microservices world

Monitoring with prometheus

- Slides: github.com/wojciech12/talk_monitoring_with_prometheus
- Prometheus naming conventions: docs

Questions?

Survey

Your feedback is important:

- TBA