lunar way | Cloud Native Aarhus

MAKE MONEY MATTER.

By Kasper Nissen (**@phennex**), DevOps Engineer **@thelunarway**

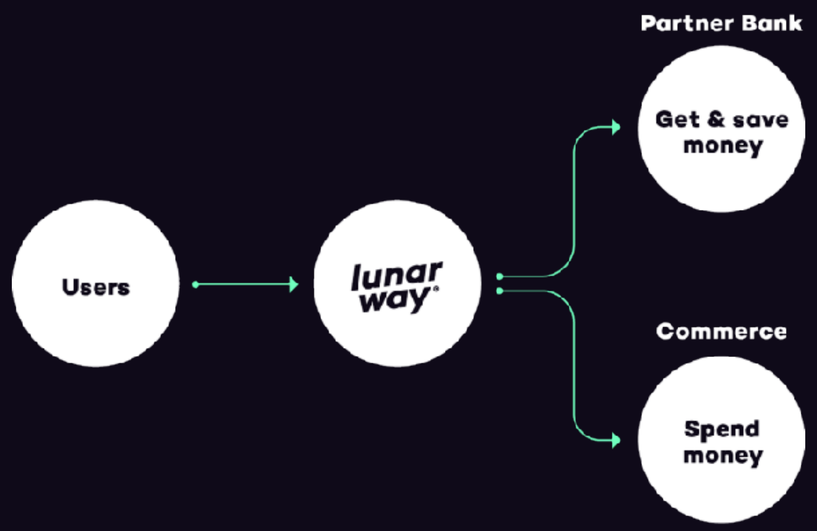
lunar way | Cloud Native Aarhus

MAKE MONEY MATTER.

By Kasper Nissen (**@phennex**), DevOps Engineer **@thelunarway**

Lunar Way

- The Partner model
  - Leverage the partner banks infrastructure
  - All money is in the partner bank
- Currently only in Denmark, will move to the nordics in the near future



Kasper Nissen

DevOps & Infrastructure Engineer @thelunarway

Experience

DevOps & Infrastructure Engineer @ LEGO (CITMABIS) (oursourced by IT Minds) for 5 months

Senior/Software Engineer @ IT Minds (~4 years part time)

Master thesis: KubeCloud - A Small-Scale Tangible Cloud Computing Environment.
Interview with Software Engineering Daily: bit.ly/2paZ5lg

Blogging about Cloud Native Tech @ www.kubecloud.io

M. Eng. Computer Technology from Aarhus University - Department of Engineering.

B. Eng. Information and Communication Technology from Aarhus University - School of Engineering

What do we have running?

19 services

215 containers in prod

1 rails monolith

3 kubernetes clusters

2 100 GB postgresql DB's

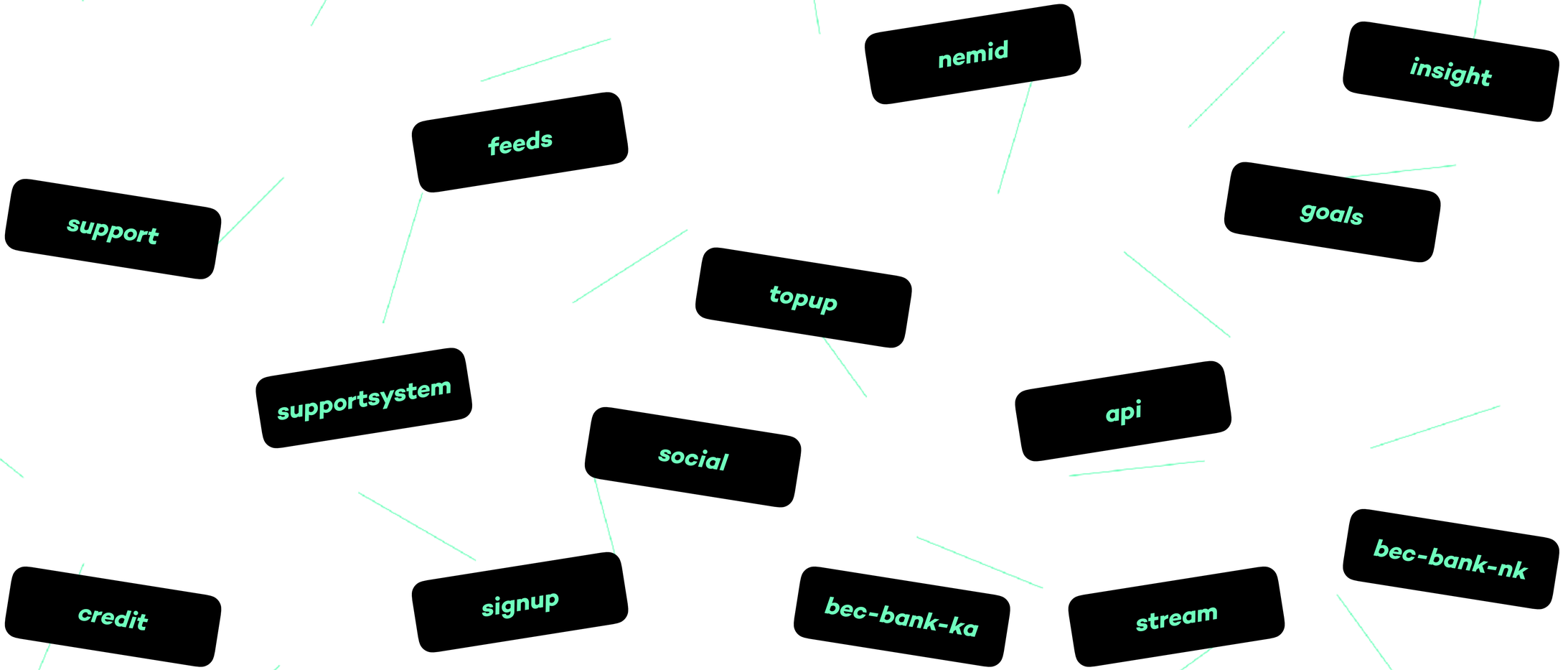
3 node rabbitmq cluster

3 AWS Accounts

13 infrastructure services

Where are we running?

Service overview



Service overview (with infrastructure)

Infrastructure

AWS

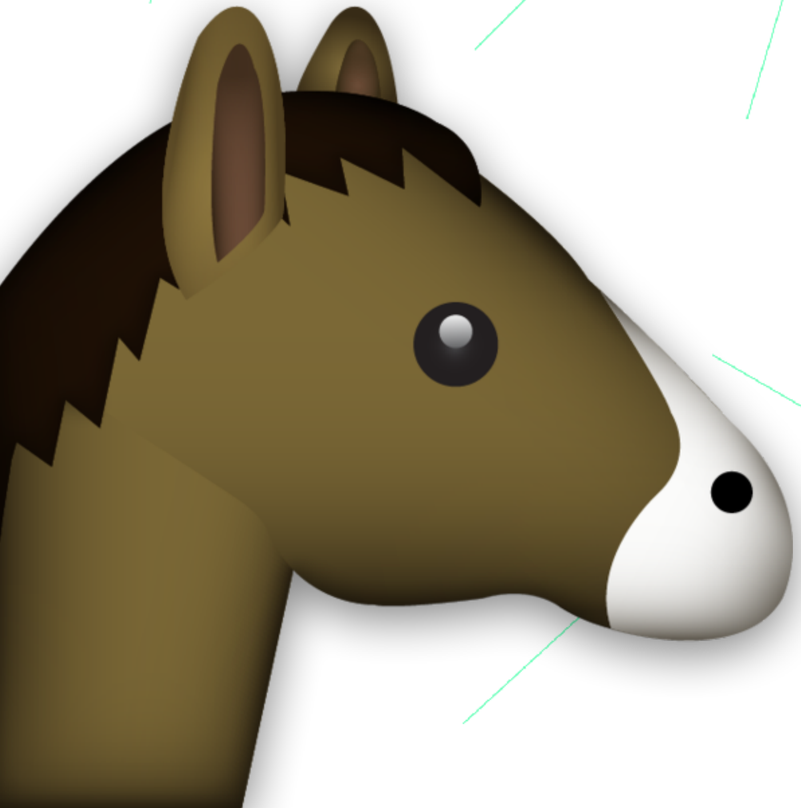
RabbitMQ Cluster

Kubernetes Cluster

Elasticsearch Cluster

PostgreSQL database

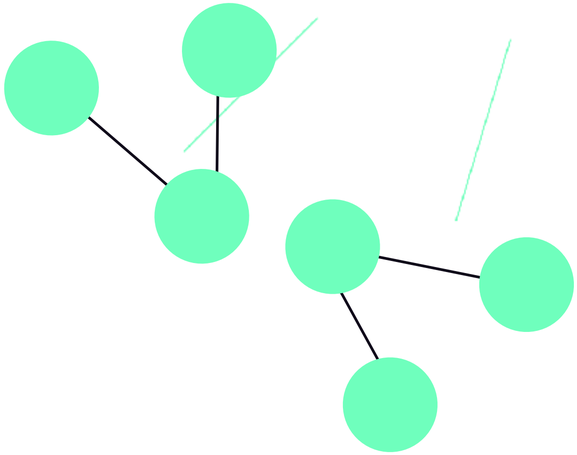
Where are we heading?



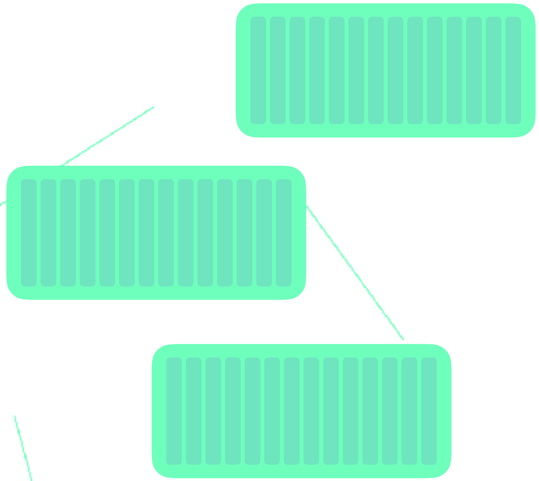
Horses vs Unicorns

- Gene Kim

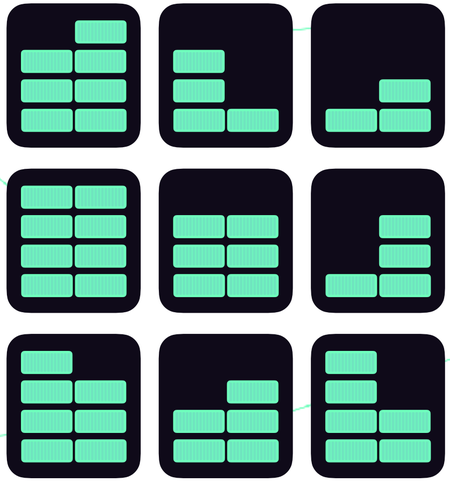
Cloud Native Utopia



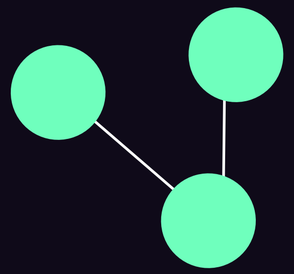
Microservice oriented



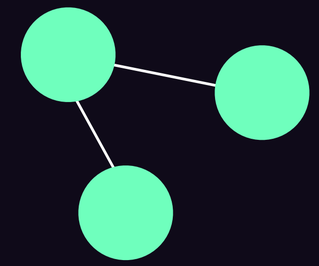
Container packaged



Dynamically scheduled



MICROSERVICES



Why ?

(our Microservice vision)

1.

- ***Development***
  - Freedom and autonomy
  - Best tool for the job
  - Speed

2.

- ***Architecture***
  - Fault tolerance
  - Flexibility
  - Coherence, decoupling, encapsulation

3.

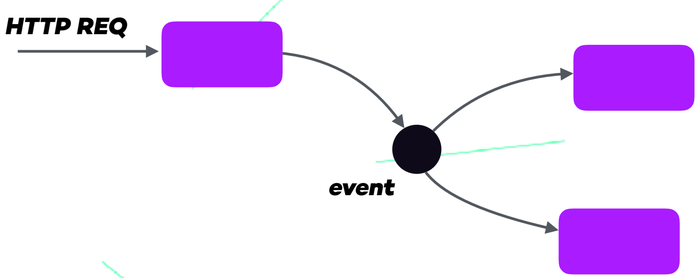
- ***Deployment***
  - Independence
  - Scalability
  - Speed
  - Resource utilisation

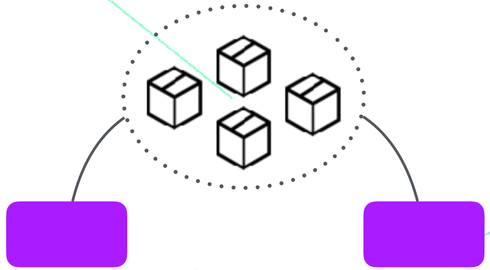
How are we building our services?

Asynchronous first
Decoupling in time and space allows for autonomy

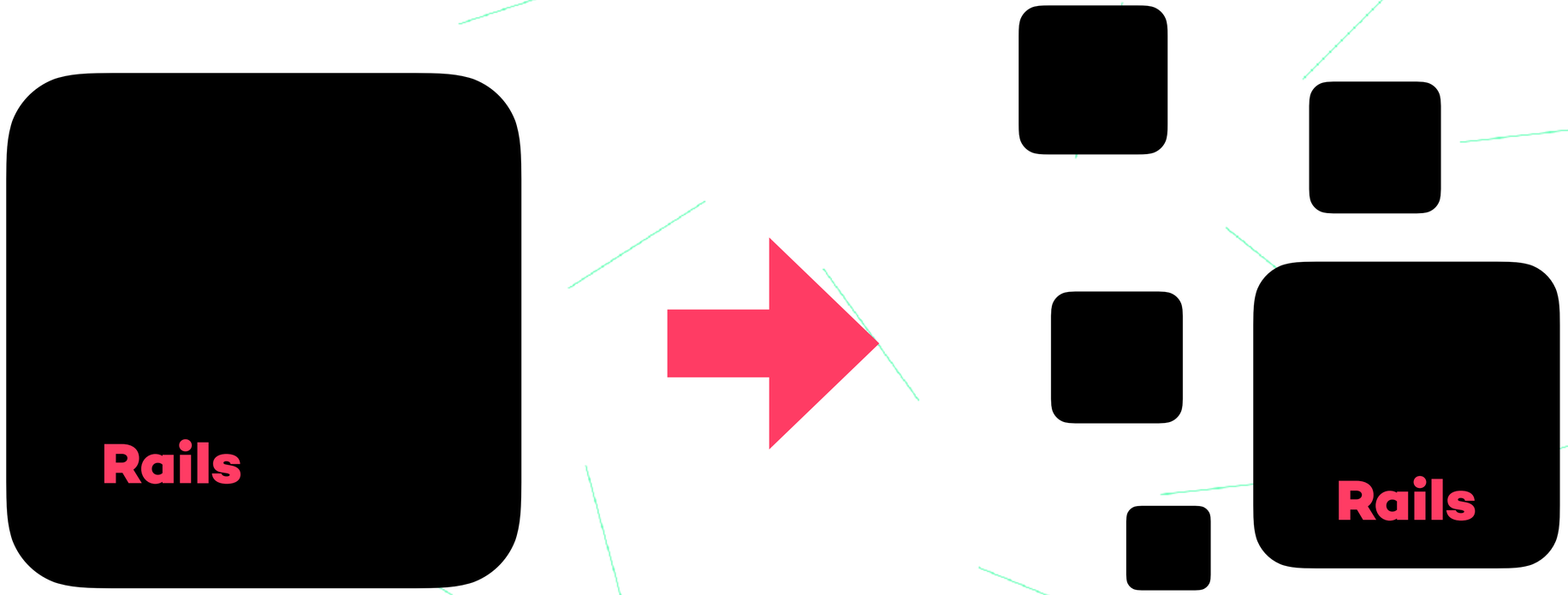
Shared dependencies
Common packages, such as logging, monitoring, communication

Each service has it's own repository
Containing source code, deployment spec, pipeline, etc.



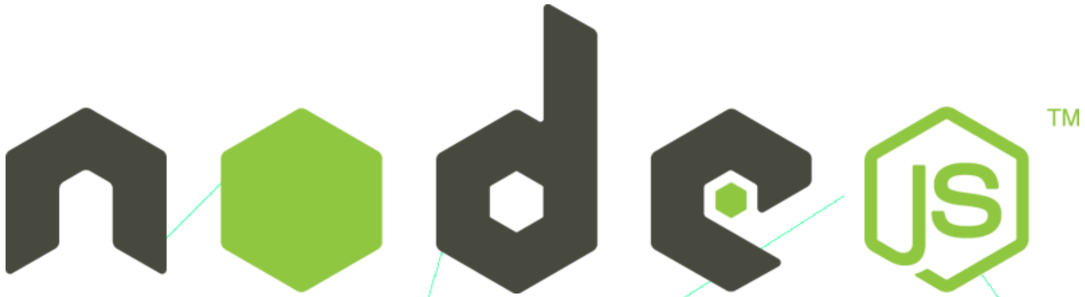


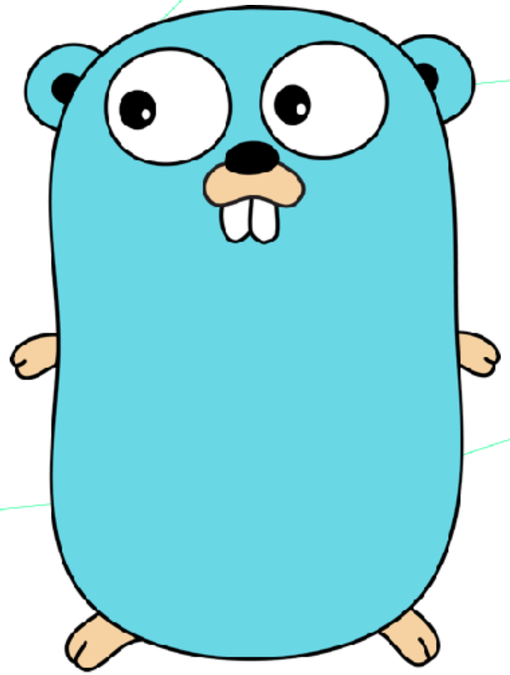
Killing our Rails monolith



... slowly strangling Rails

Languages

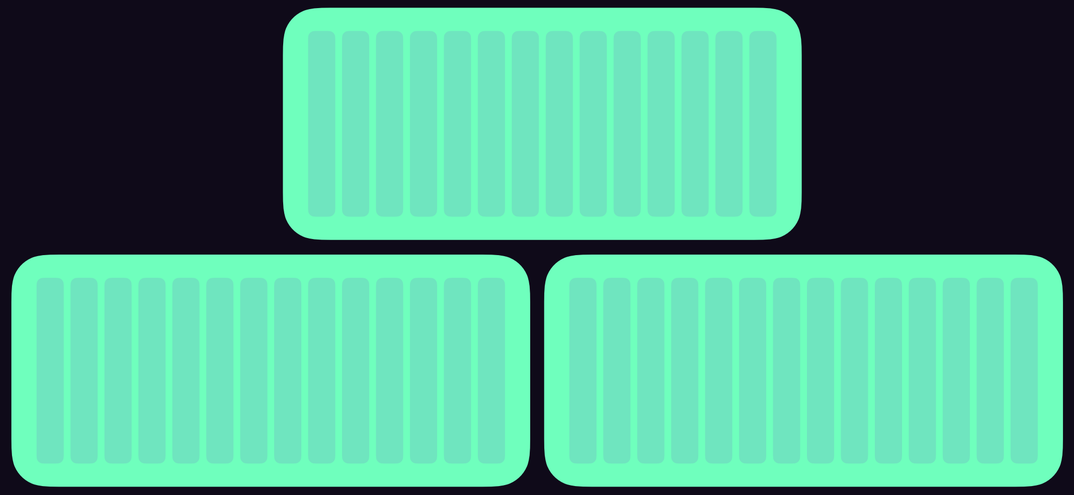




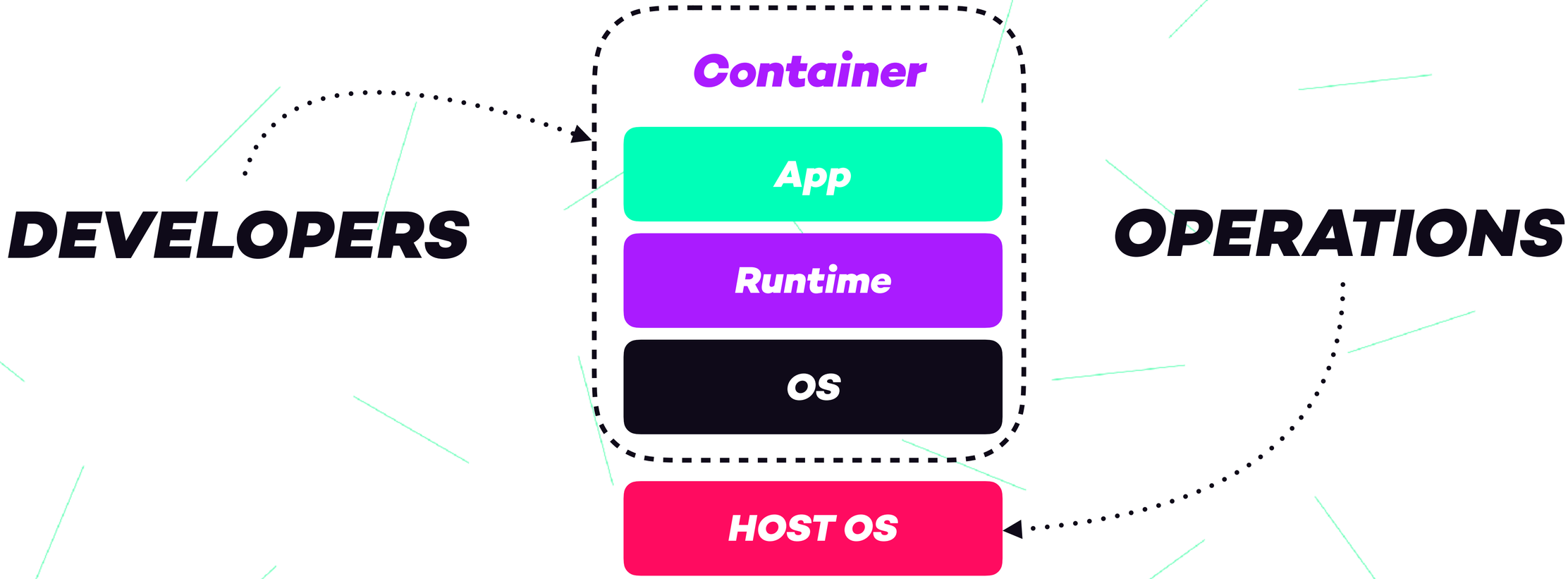
Challenges - so far

- **Development**
  - Tooling - e.g. Swagger for events?
  - Local env
  - Shared code
- **Deployment**
  - Automation
  - Versioning
- **Architecture**
  - Distributed monolith
  - Transparency
  - Complexity
    Cross cutting concerns
- **Operation**
  - Monitoring
  - Tracing events

CONTAINERS



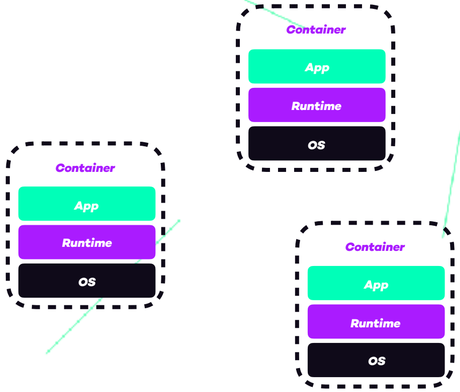
What?



Why?

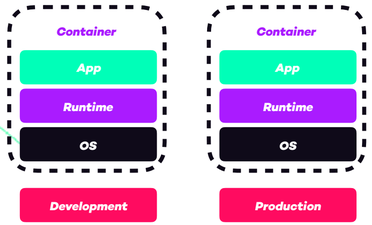
Isolation

Services are isolated and contained in their environment



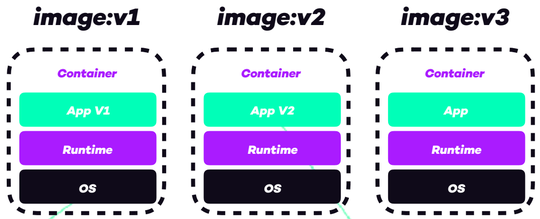
Consistency in portability

The container will run in the same way in local env as in prod



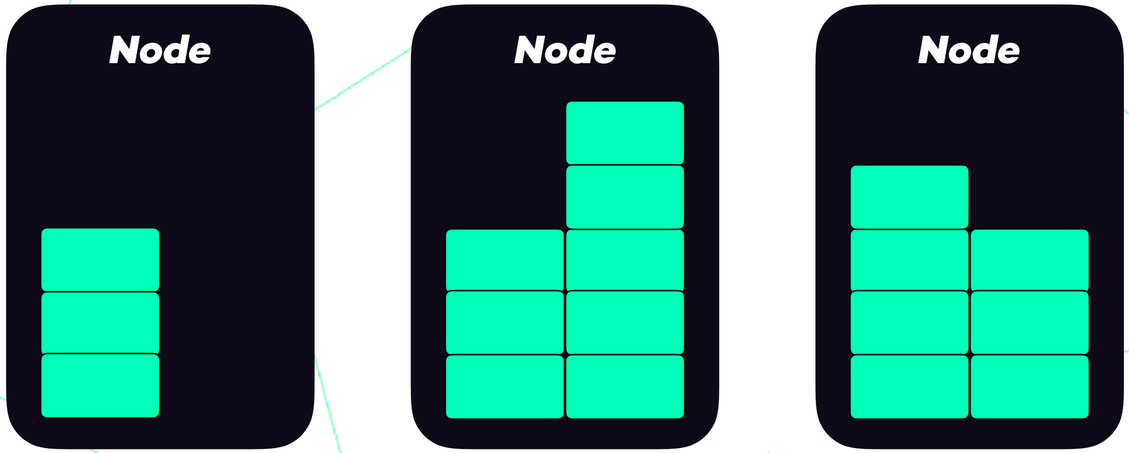
Versioning

Versioning a container is easy, rolling back and forth becomes easy



DYNAMICALLY SCHEDULED

What?



Why?

Scheduling
The scheduler will schedule your service on a node

Resource optimization
Scheduling allows for better packaging of services in hosts

Resiliency
If a node dies, the scheduler will reschedule on another node

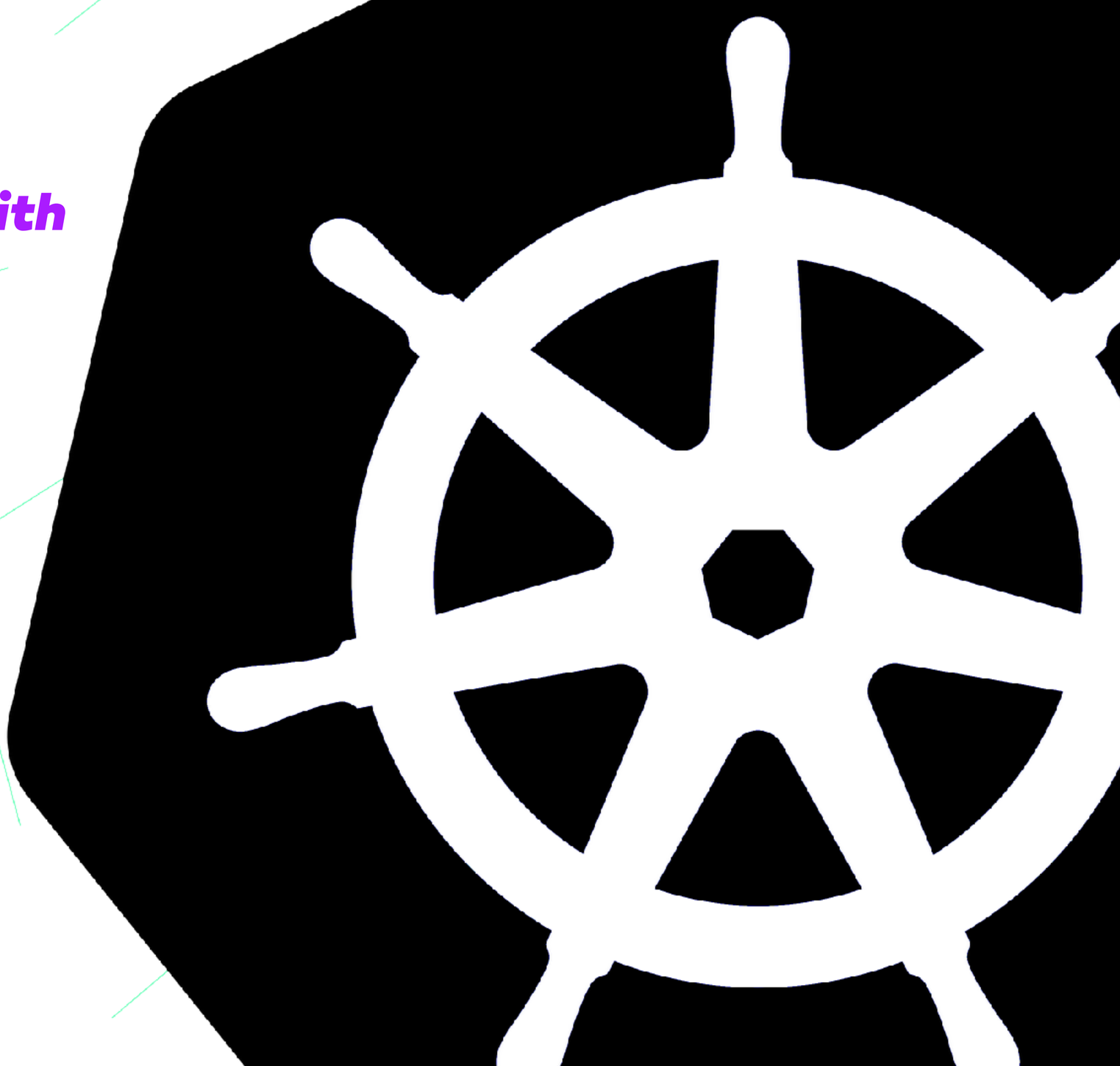
Scalability
Scaling a dynamically cluster is easy, just add more nodes

SO, HOW DO WE WORK TOWARDS BECOMING A UNICORN?

Currently used CNCF projects

Container Orchestration with

Kubernetes



Why Kubernetes?

Community
48k+ commits, 22+ GitHub stars, 1.1k contributors

Crossplatform
Multiple arcs, multiple cloud providers

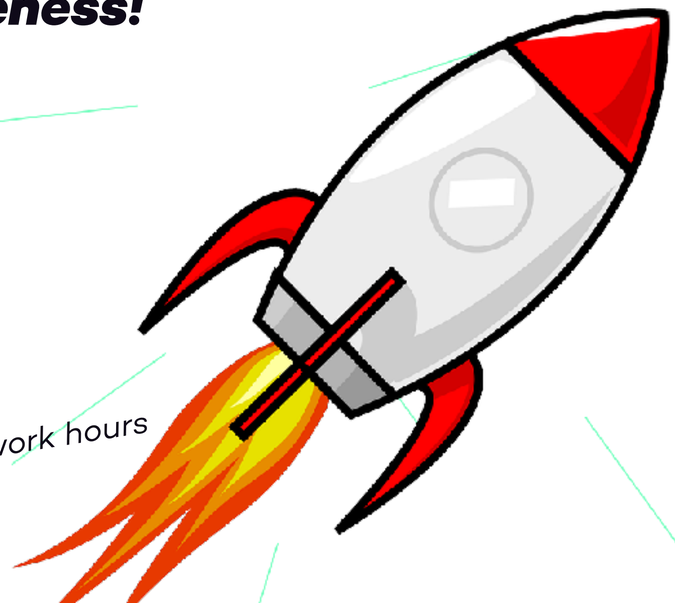
Resource optimization
Packing nodes to utilize available resources

Tooling
A lot of great tools

Scaling
Great integration with auto-scaling, both on node- and container-level

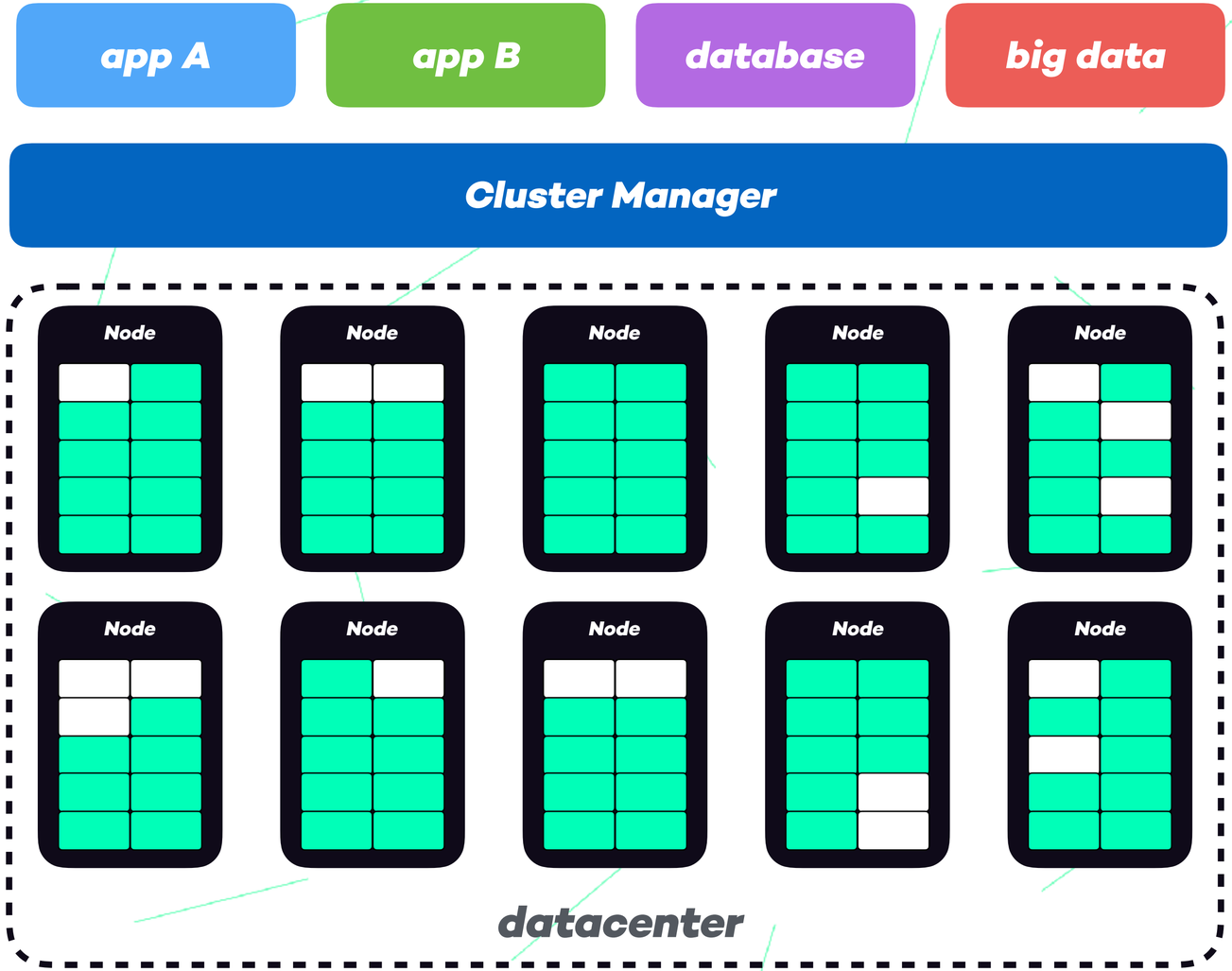
High availability
Automatic failover, redundancy

Pure Awesomeness!
It's just awesome!

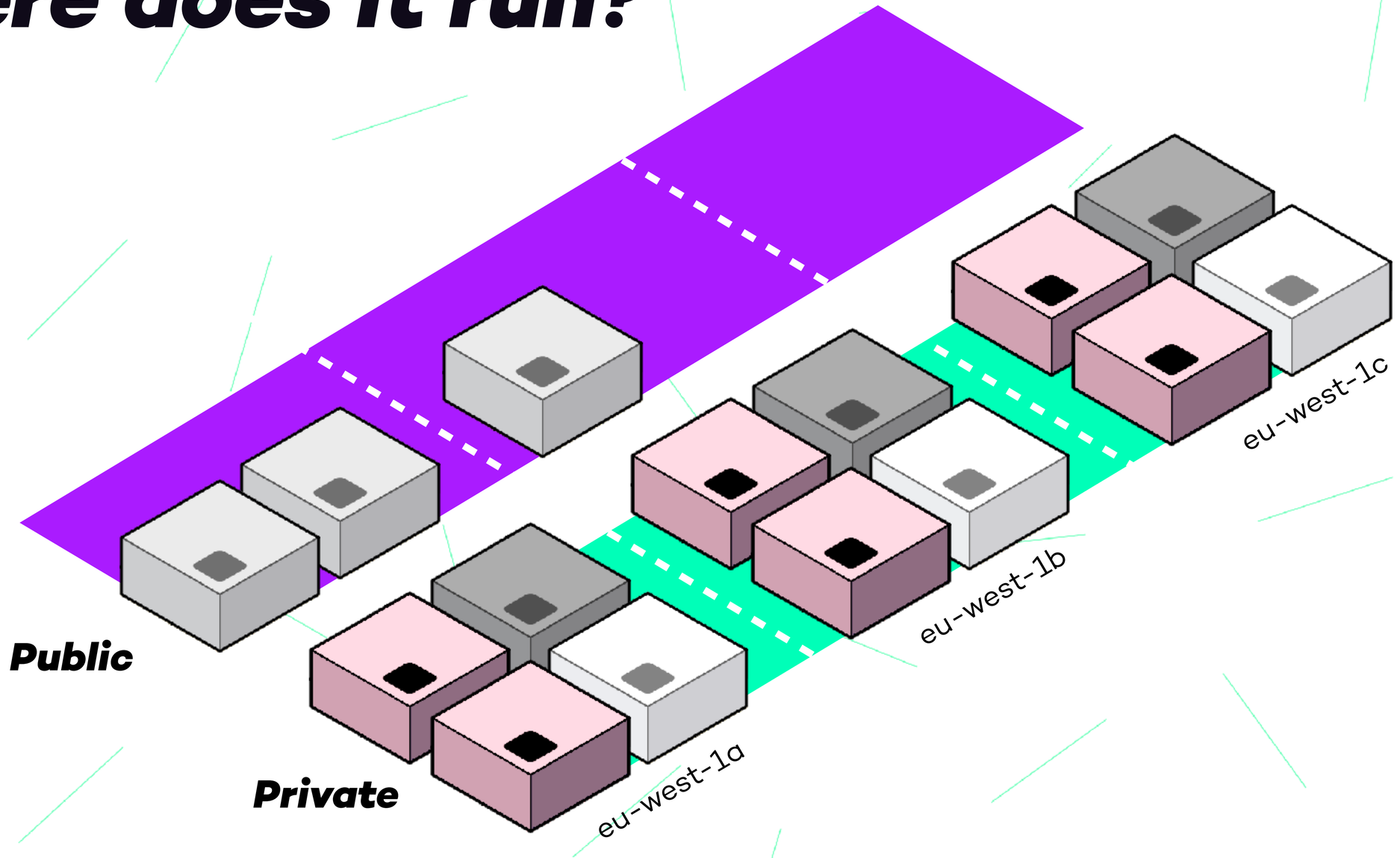


1.5 release:
Estimated 400 years of work hours

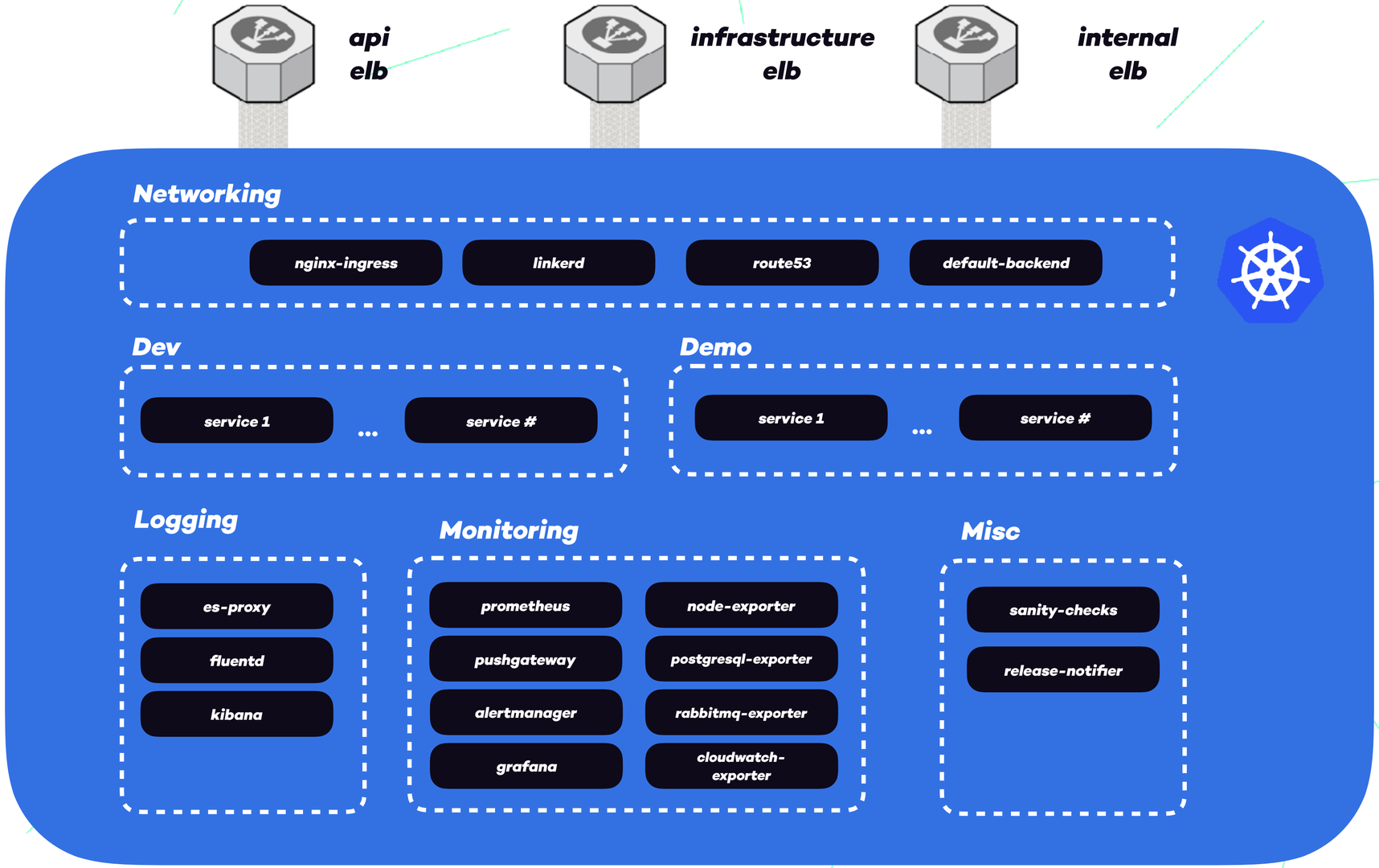
What does it do?



Where does it run?



Services running in kubernetes



What do we think of it?

Freedom
Squads can deploy and more or less implement how they see fit

Autonomous services
Squads can work independent of other squads

Continuous Delivery
Kubernetes allows us to deploy multiple times a day. It's easy to rollback in case something went wrong

Flexibility
We run many different type of workloads in the cluster. Gives us mobility to become cloud agnostic

Scalable infrastructure
Scaling the infrastructure is easy, both on node and container level

High availability
Kubernetes takes care of container failures, AWS Auto Scaling groups takes care of node failures

Easy maintenance
We are using Kubernetes Operations to help us spin up our clusters, and maintain them.

Log collection with

Fluentd

Why fluentd?

Simple and Easy
Provides a simple interface for specifying input and output. Works great with Kubernetes and containers.

Community
Big community around fluentd, validates our choice.

Small memory footprint
Do not require a lot of resources in the cluster

Proven reliability and performance
It's a fairly battle tested project

What does it do?

1.

```
<source>
  @type tail
  path /var/log/containers/*.log
  pos_file /var/log/containers/es-containers.log.pos
  tag kubernetes.application.*
  format json
  time_key event_time
  time_format %Y-%m-%dT%H:%M:%S.%3NZ
</source>
```

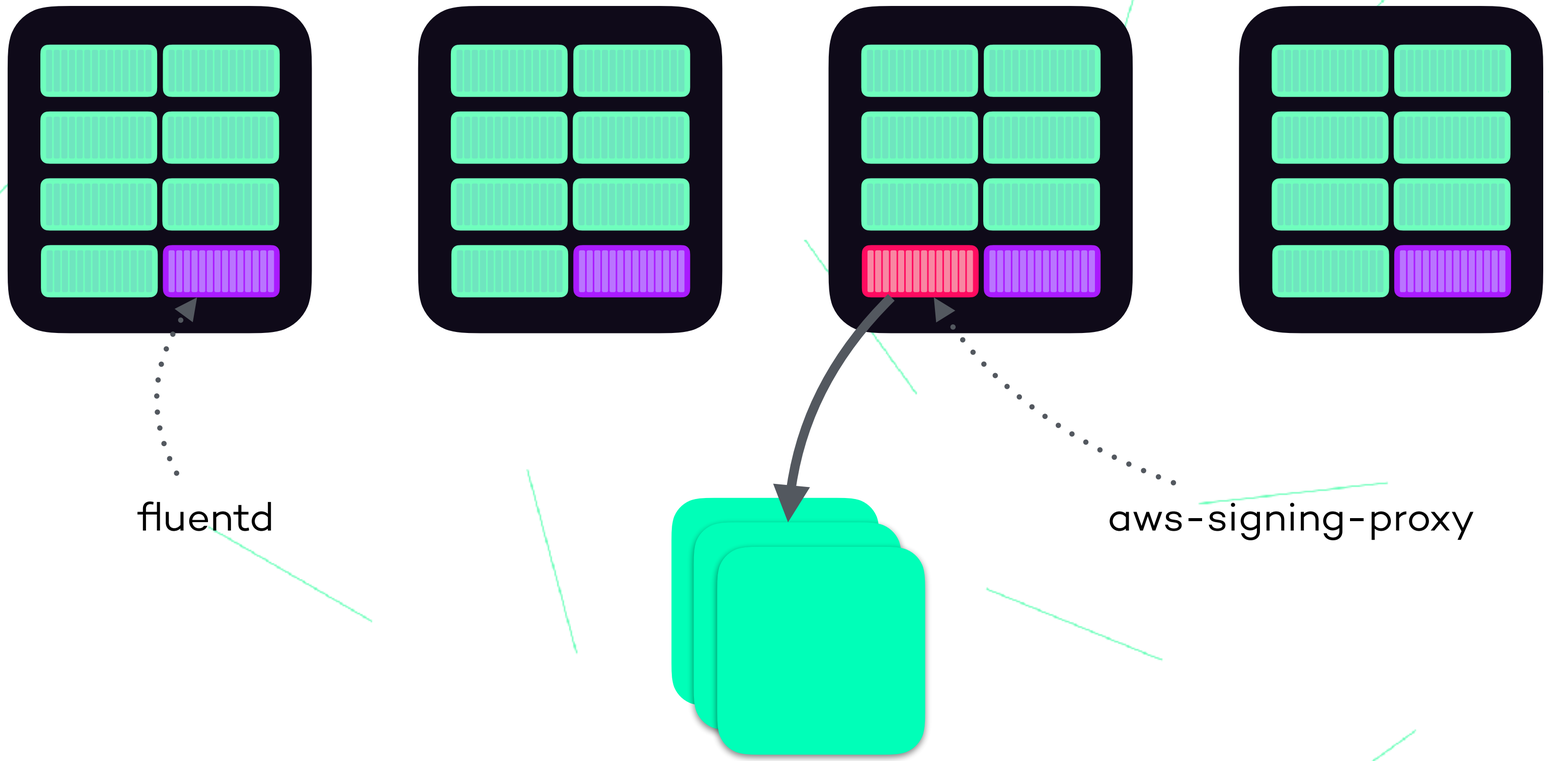
2.

```
<filter kubernetes.application.**>
  @type kubernetes_metadata
  merge_json_log true
  preserve_json_log false
</filter>
```

3.

```
<match kubernetes.application.**>
  type elasticsearch
  host es-proxy
  port 9200
  include_tag_key true
  logstash_format true
  logstash_prefix application-
  reload_on_failure true
</match>
```

Logging setup



What do we think of it?

Very easy to use
Set up is easy!

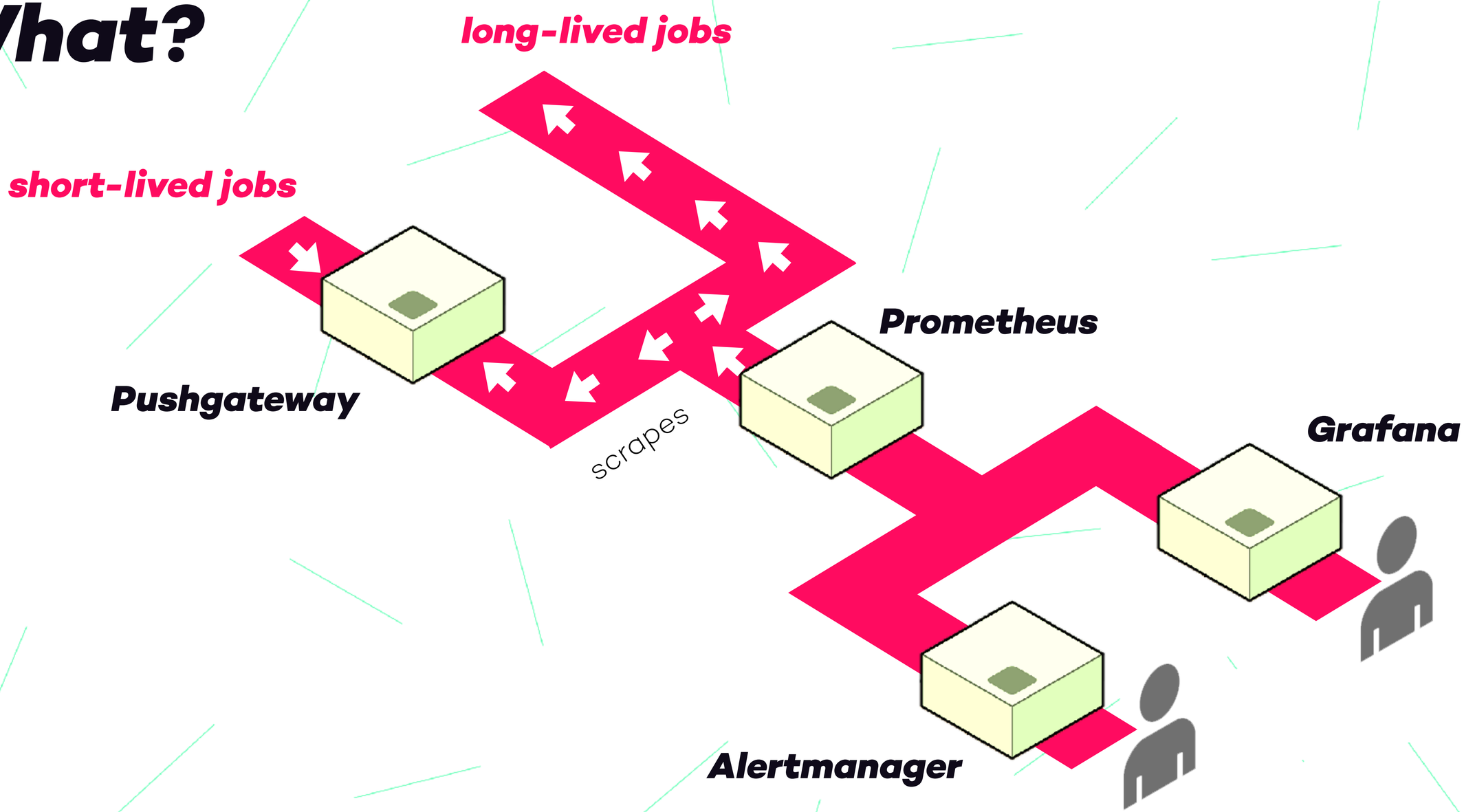
Works great with Kubernetes
Awesome plugin for adding Kubernetes metadata - making it easy to identify pods etc.

Run as a daemonset
Easy to run in every node of the cluster as a daemonset

Monitoring with

Prometheus

What?


long-lived jobs
short-lived jobs
Pushgateway
scrapes
Prometheus
Grafana
Alertmanager

What does it do?

Multi-dimensional data model
Time series identified by metric name and key/value pairs

Flexible query language
Comes with a builtin query language for al kinds of operations, sums, averages, increase, rate, etc.

Easy and simple
Easy to setup, and works great with Kubernetes service discovery

Alerting and great integration with Grafana
Prometheus has a builtin alerting system, and Grafana provides easy integration for making metrics visible in dashboards

Pull-based approach
Prometheus scrapes it’s targets at a regular interval

What metrics are we collecting?

Kubernetes specific metrics
Pods running, health of Kubernetes system components, etc.

RabbitMQ
Activity in queues, unacknowledged messages

Nodes
CPU, Memory

Traffic
Incoming traffic, upstream latency in cluster, etc.

Containers
CPU, Memory

Application Specific metrics
Relevant metrics, instrumented by the services owners

DEMO

What do we think of it?

Provides great insights

Provides valuable insights in the state of the cluster

Makes is easy to developers to instrument their services

We provide a simple package for instrumentation, making squads able to do their own monitoring. ***YOU BUILT IT, YOU RUN IT!***

Grafana integrations is sweet!

Grafana and Prometheus works well together, making Grafana the interface for building dashboards and alerts

Kubernetes <3 Prometheus

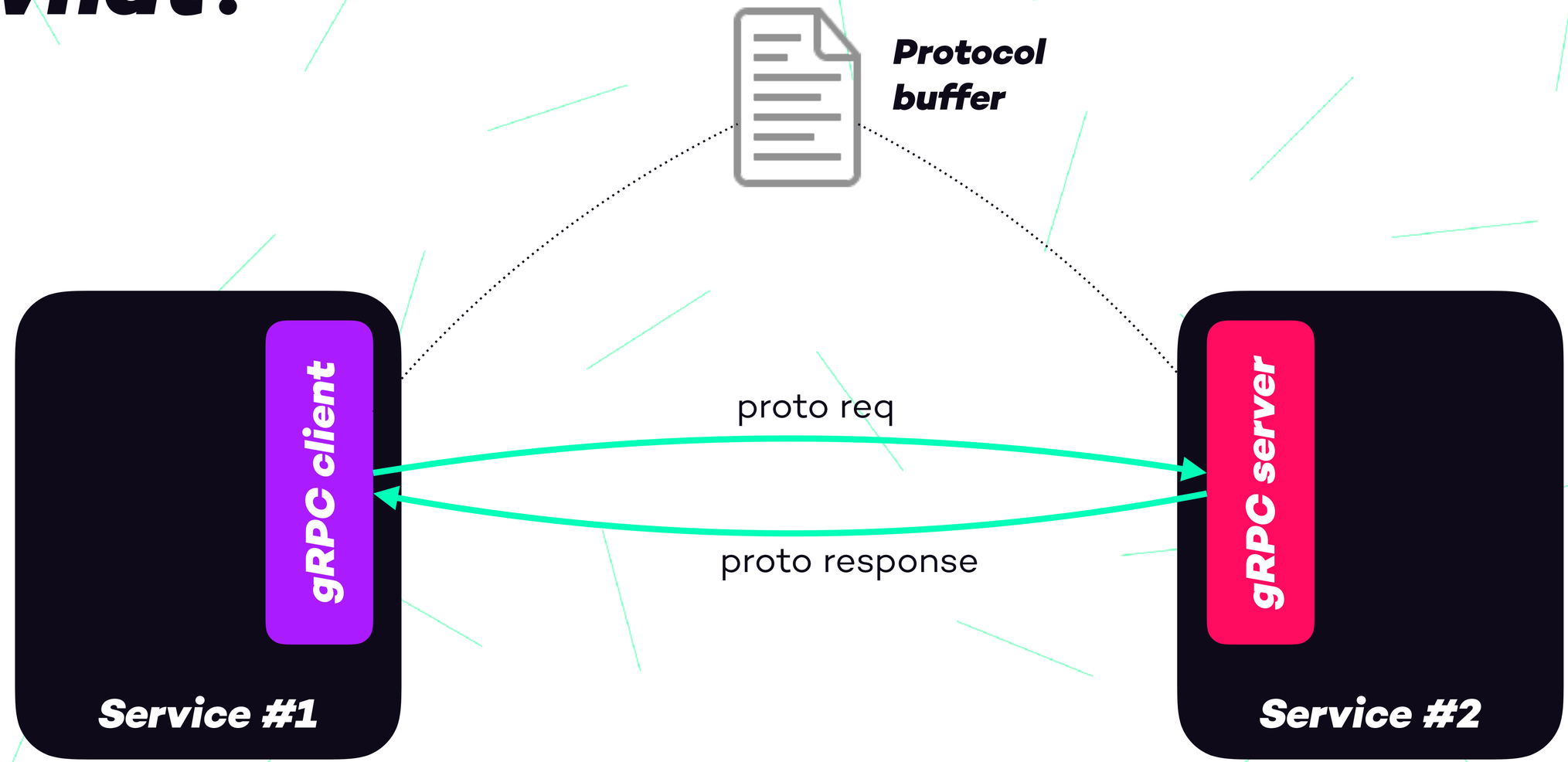
The only thing a service owner has to do in the cluster to make Prometheus scrape their services is to add:

```
annotations:
  prometheus.io/scrape: 'true'
```

Service to service communication with

gRPC

What?


Protocol buffer
gRPC client
proto req
proto response
gRPC server
Service #1
Service #2

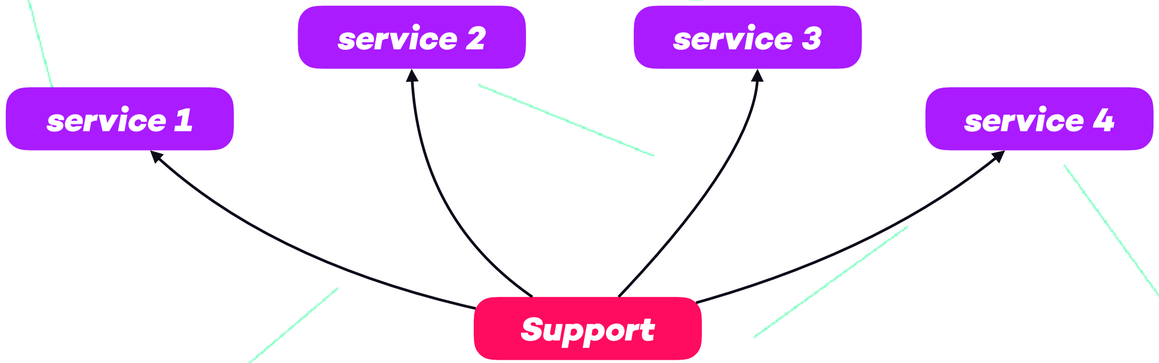
What do we use it for?

It will be our default choice for synchronous calls

Synchronous service to service communication will be aligned on gRPC

Internal support system will use it to fetch data from our services

Our internal support system needs information from different services on demand, the service will use gRPC to fetch the data



Why gRPC?

Simple service definition using Protocol buffers
Simpler request handling, no need for serialization and deserialization

Binary protocol
Less overhead in communication.

Works across multiple languages and platforms
gRPC has a widespread support for multiple languages, making it a perfect fit in our current polyglot architecture

Works great with go and the rest of the ecosystem
Docker, Kubernetes uses gRPC as long with Go. It’s a natural extension for service to service communication. Based on many years of Google experience!

Back to the

UNICORN...

Are we there yet?

YES! and no..

We deploy multiple times a day
Deployment is autonomous, squads can deploy to production as they please.

We can easily scale to larger demands, if necessary
Scaling our infrastructure is easy

We can tolerate AZ failures to some extend
Our services are spread across availability zones

HOWEVER, doing microservices are complex!
We still need to implement better tracing, using the CNCF project ***OpenTracing*** and Zipkin
We need more insights and smarter routing in our service to service communication, we will be using ***linkerd***.

Last thing

DO YOU HAVE ANY QUESTIONS?

Thank you for listening!

That was it for me!

If you wanna know more, send me a message in the Cloud Native DK Slack Community.
Remember to sign up at: https://cloudnative-dk.herokuapp.com/

Catch me on Twitter ***@phennex***

I will be speaking again:

- CoDe:U - Continuous Delivery Users Århus (June 20th in INCUBA, Åbogade 15, Aarhus N)
  - Link: https://www.meetup.com/CoDe-U-AROS/events/239847862/

- GOTOCon Copenhagen - October 1st
  - Link: https://gotocph.com/2017/sessions/237