

# Containerful Serverless

Roland Huß @ro14nd

Senior Principal Software Engineer, Red Hat  
OpenShift Serverless Architect



# Wait ... wat ?

---



# What is “Serverless” ?

“Serverless computing refers to the concept of building and running **applications** that **do not require server management**. It describes a finer-grained **deployment model** where applications, bundled as one or more functions are uploaded to a platform and then **executed, scaled, and billed** in response to the exact **demand** needed at the moment”

-- CNCF Definition, <https://www.cncf.io/blog/2018/02/14/cncf-takes-first-step-towards-serverless-computing/>

# Serverless vs. FaaS

**Serverless** is a **Deployment Model** that abstracts away the driving machine infrastructure.

- No server management required
- Executed, scaled and billed according to demand
- Defines a deployment packaging, but otherwise agnostic to the application

**FaaS** (Function-as-a-Service) is a **Programming Model** that mandates developing your application with fine grained function that match a given signature.

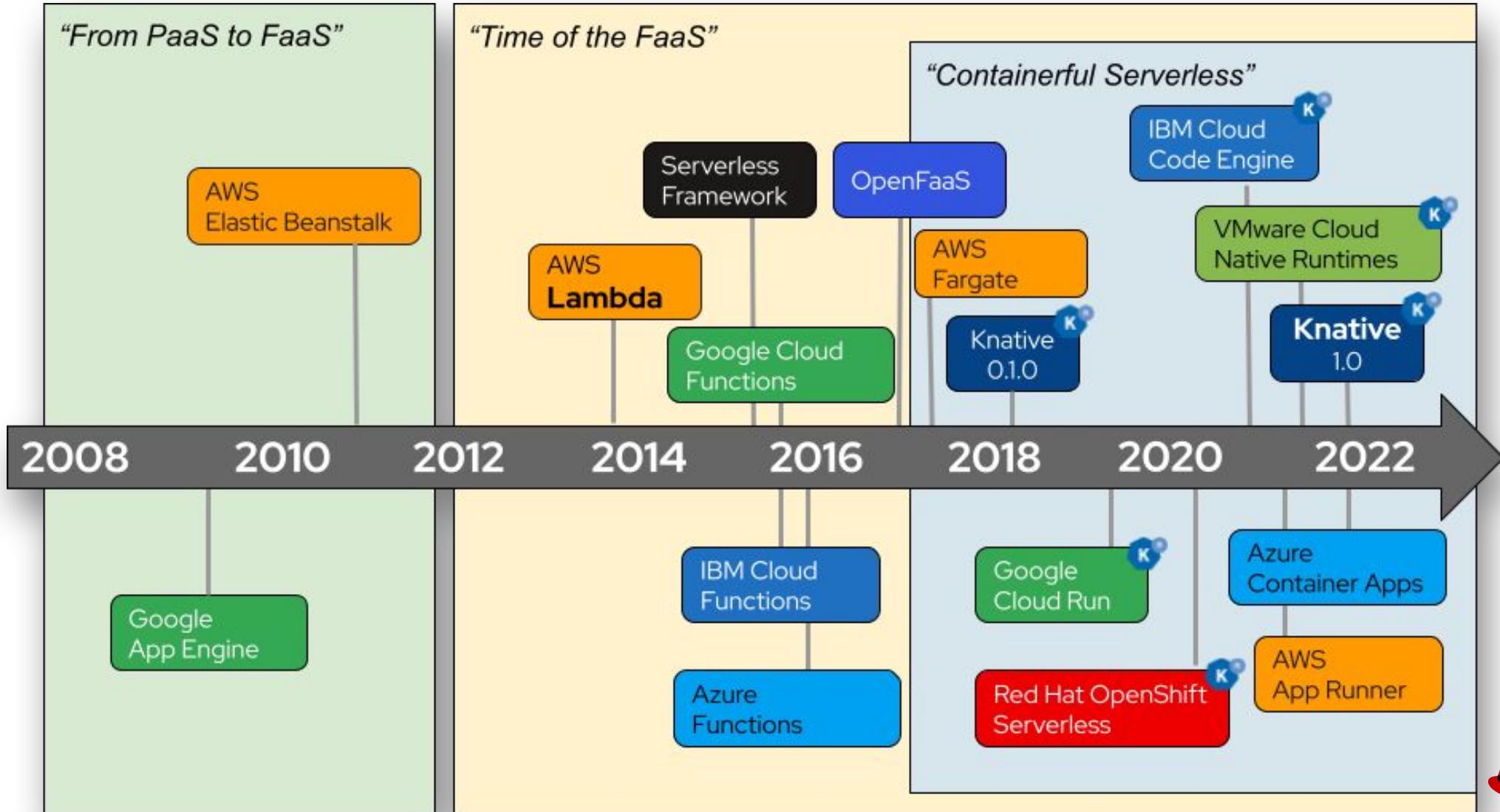
- Deployed as Serverless application
- Typically used as *glue code* to connect services

# Serverless Workloads

The ideal Serverless Workload is ...

- ... **stateless**
- ... **short-running**
- ... **HTTP**-based web services or **Event-Driven** Applications

# Serverless Timeline



# Serverless Packaging

- **Custom** (zip, tar, ...)
  - All Cloud Providers
- **OCI Container Images**
  - Docker, Podman, Kubernetes, ...
- **Virtual Machines**
- **MicroVMs**
  - Firecracker, Kata containers, gVisor, ...
- **Web Assembly Modules (WASM)**
  - Krustlets, WasmEdge

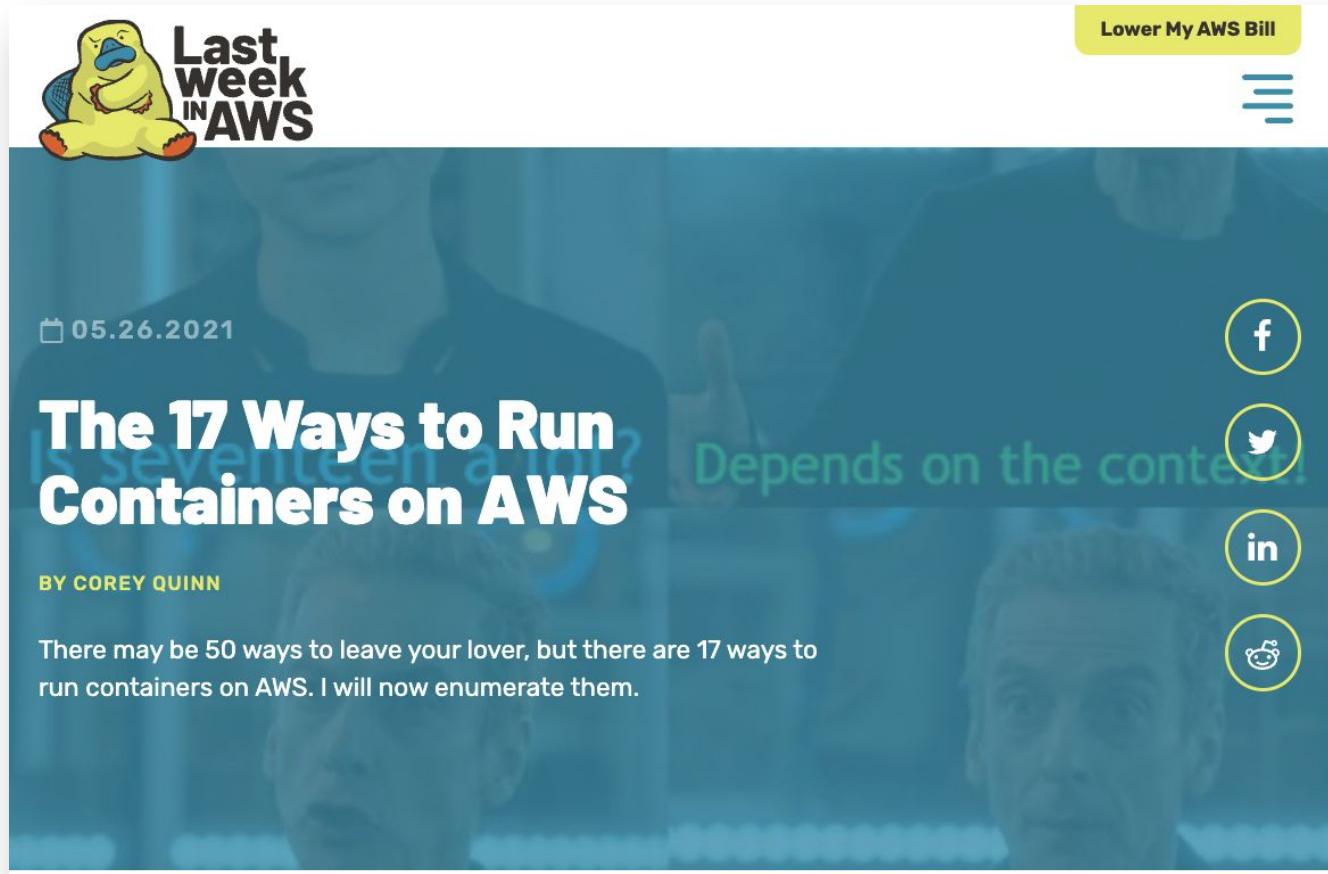
# Containerful Serverless

- **AWS:** Lambda, Fargate, App Runner
- **Microsoft Azure:** Container Instances, Container Apps
- **OpenFaaS**
- **Knative**
  - **Google** Cloud Run, **IBM** Code Engine, **VMware** Cloud Native Runtimes for Tanzu, **Red Hat** OpenShift Serverless

An aerial photograph showing a river flowing through a dense forest. The river curves through the landscape, creating a dark, winding path through the green trees. The forest appears to be a mix of coniferous and deciduous trees, with some yellow and orange foliage visible, suggesting it might be autumn. The overall scene is a natural, undisturbed environment.

AWS

# Containers on AWS

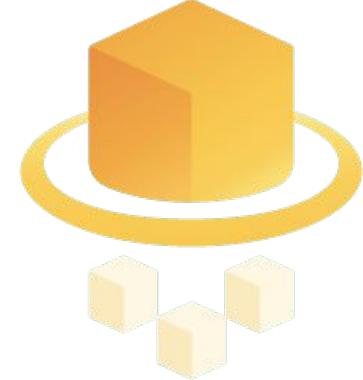


# AWS Lambda



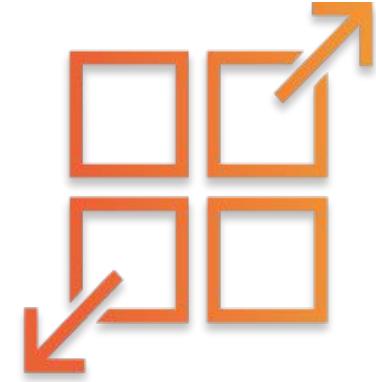
- Opinionated, **event-driven** computing platform
- Introduced **2014** with **Node.js** as runtime
- Main use case as '**glue code**' for connecting AWS services
- **Pay-as-you go** with 1 ms billing granularity
- **Container** packaging since 2019
  - up to 10 GB in size (instead of 50 MB zipped)
  - specific base images or clients neede to fullfill Lambda runtime contract

# AWS Fargate



- Introduced in 2017 as a computing **backend for ECS**
- Deploying containers without managing EC2 instances.
- Fargate can be used in EKS do deploy Kubernetes Pods on Fargate
  - “Infinite large Kubernetes Node”

# AWS App Runner



- Simplest and most opinionated way to deploy containers on AWS
- HTTP workloads
- Automatic Scaling based on traffic
- Automatic routing (public or private URL)
- No scale to zero

A wide-angle photograph of a tropical island. The foreground is filled with clear, turquoise-blue ocean water. In the middle ground, a dense forest of tall palm trees stretches across the island. A small, traditional wooden boat with a blue hull and white trim is anchored near the shore. The sky above is a bright, clear blue with a few wispy white clouds.

# Azure

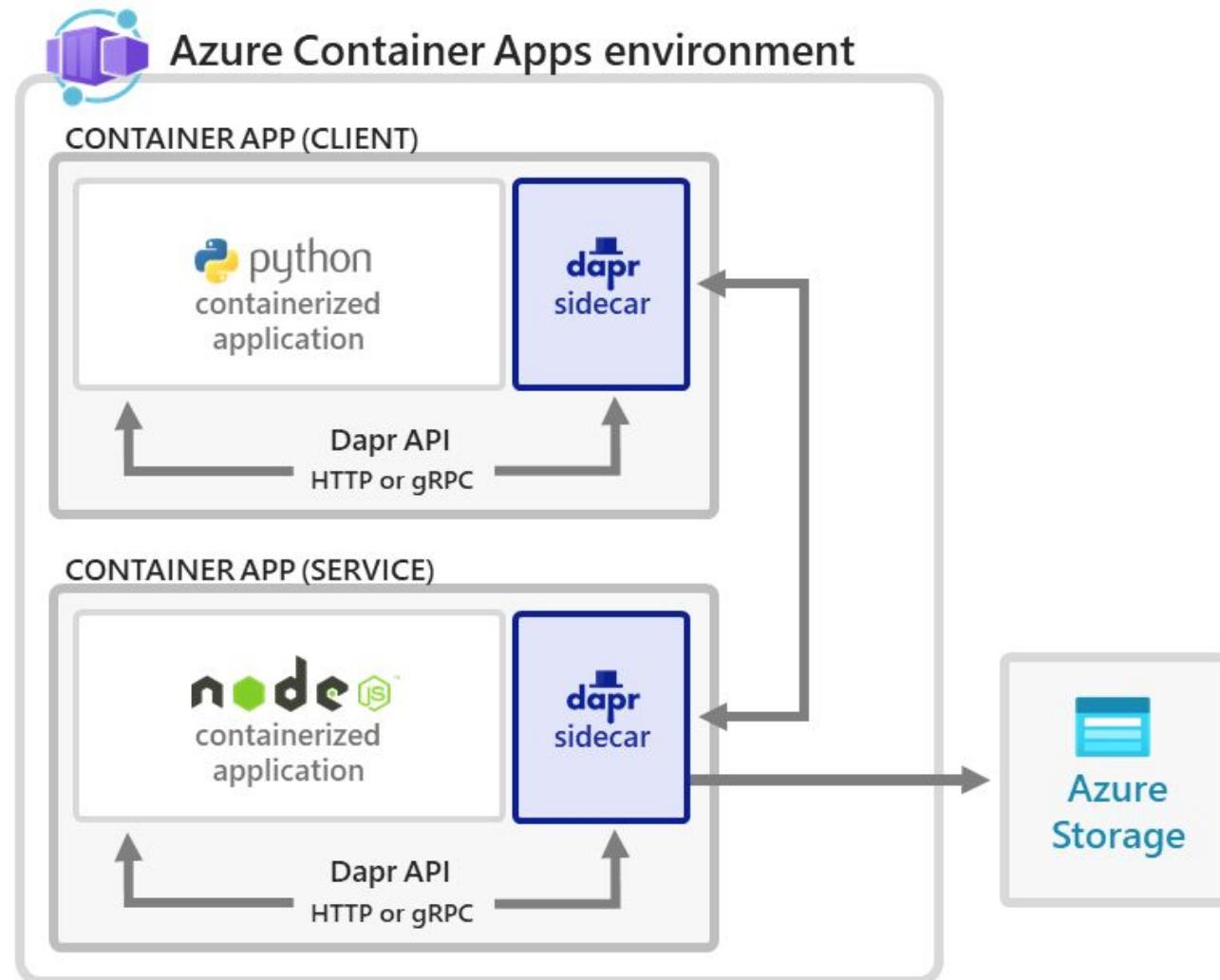
# Container Instances (ACI)



- Started in 2017
- Similar scope as AWS Fargate
- No scaling or load balancing
- Unopinionated approach to start containers
- Simpler to use than ECS / Fargate on AWS
- Can be added as virtual nodes to AKS

# Container Apps

- Started late 2021, in preview
- Multiple revisions with traffic split
- Autoscaling options: HTTP, CPU/Memory, KEDA
- Native Dapr integration

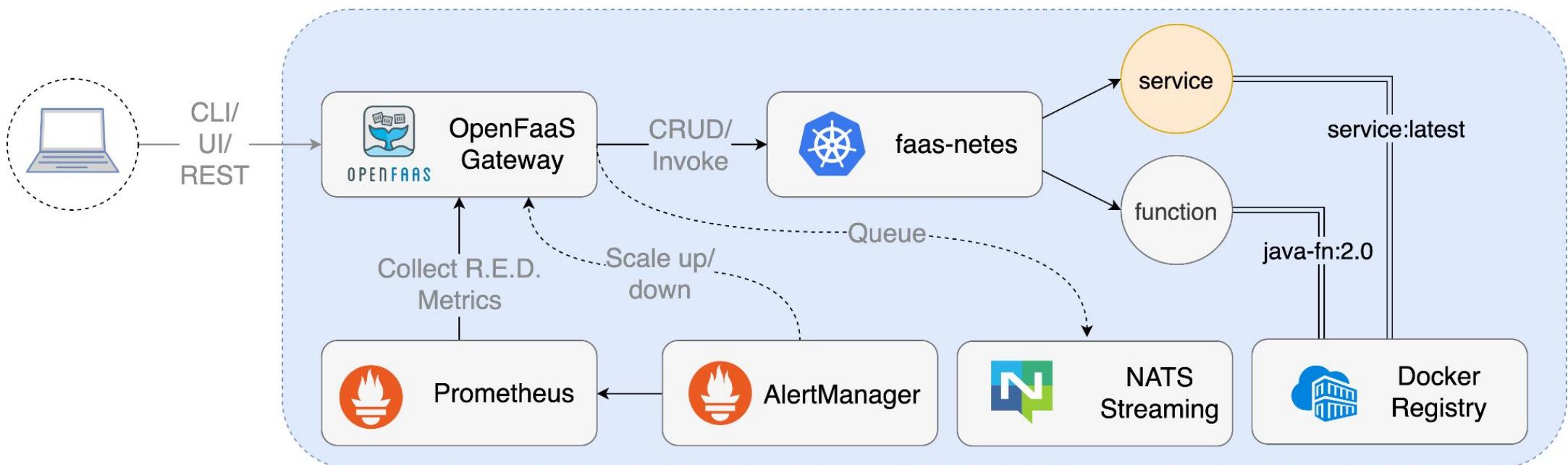


A large whale is captured mid-leap, its dark, textured body and long, curved tail fluke cutting through the deep blue ocean. A massive splash of white and light blue water erupts from the whale's movement. The background consists of a vast, calm sea under a clear, pale blue sky.

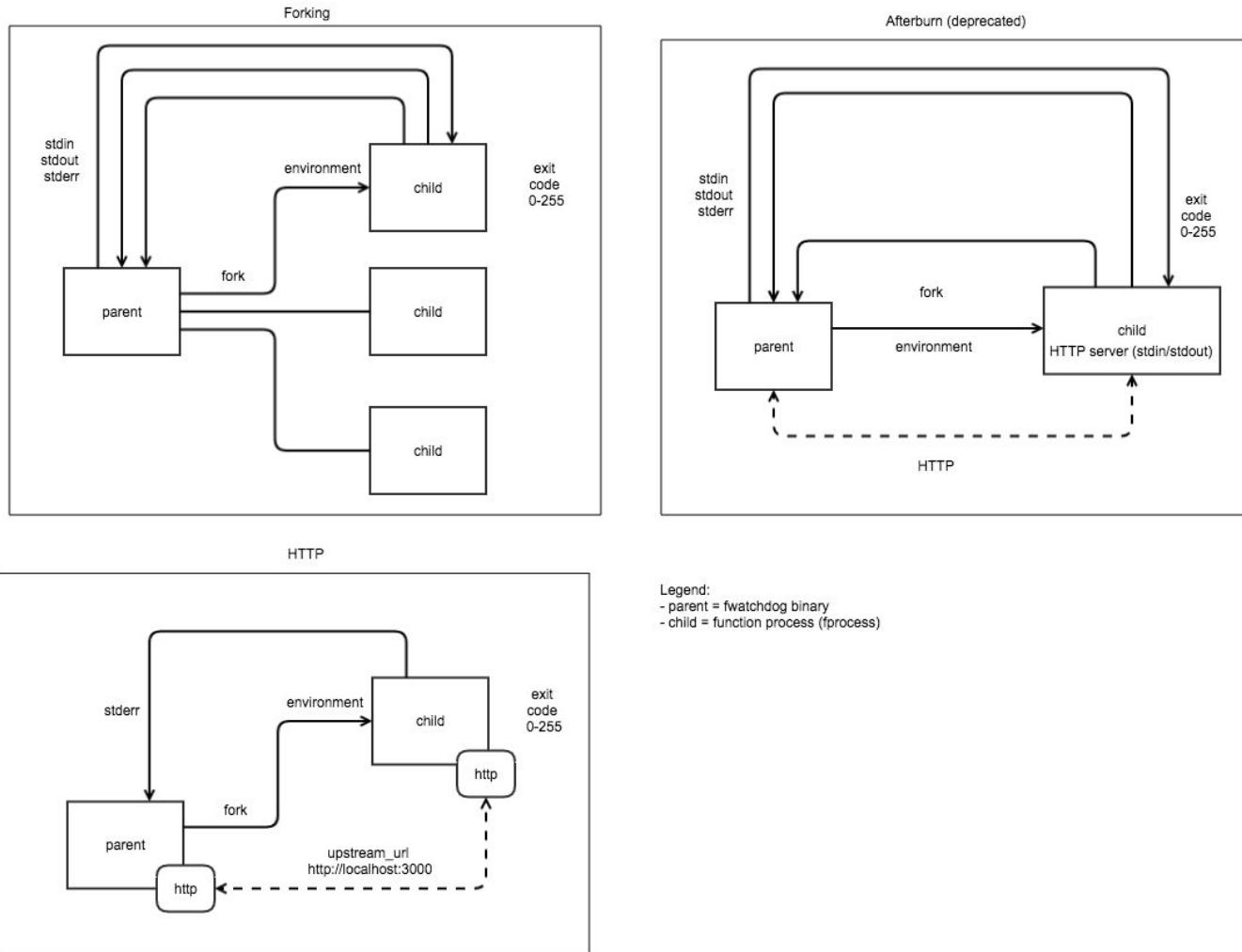
# OpenFaaS

# OpenFaaS

- Created by Alex Ellis in 2016
- Similar API as CGI and fast-CGI



# WatchDogs



# Open Core

- Exclusive Features of OpenFaaS Pro:
  - New Autoscaling engine including scale-to-zero
  - Retry of failed functions
  - New User Interface
  - Support for EDA application via triggers (Kafka, AWS SQS)
- No free trial (?)

A photograph of a traditional wooden longtail boat sailing on the ocean. The boat's hull is dark wood, and its deck is made of light-colored planks. A large wooden pole (steering oar) is visible at the stern. The boat is moving towards a group of green, craggy limestone islands under a blue sky with white clouds.

# Knative

**Kubernetes-based platform to  
deploy and manage modern  
serverless workloads.**

<https://knative.dev>

---

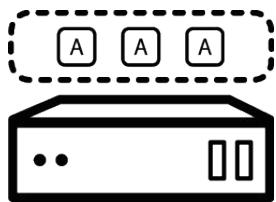
# Modern Event-driven Workloads with Knative

JAX, May 4th, 16:30 - 17:30  
Zagrebsaal A

# Components

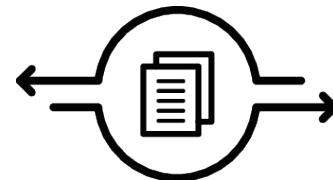
## Serving

A request-driven model that serves the container with your application and can "scale to zero".



## Eventing

Common infrastructure for consuming and producing events that will stimulate applications.



# Background Information

- Started as an **Open Source** Project mid-2018 by Google
- CNCF Incubating Project since March 2022
- Community driven with a lot of vendor backing
  - <https://github.com/knative>
  - <https://knative.dev>
  - Support by Google, Red Hat, IBM, VMware, Triggermesh, SAP and more
  - Organized in multiple Working Groups with weekly meetings
- Releases
  - Current: **v1.4**
  - 6 week release cadence

# Serving Concepts

- **Demand-based autoscaling**, including scale-to-zero
- **Separation** of code and configuration
- Simplified deployment model catered for **stateless applications**
- Rich **traffic split capabilities** to enable custom rollout strategies of new versions

# From Deployment to KService

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: random
spec:
  replicas: 1
  selector:
    matchLabels:
      app: random
  template:
    metadata:
      labels:
        app: random
    spec:
      containers:
        - image: rhuss/random
          name: random
      ports:
        - containerPort: 8080
```

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: random
spec:
  replicas: 1
  selector:
    matchLabels:
      app: random
  template:
    metadata:
      labels:
        app: random
    spec:
      containers:
        - image: rhuss/random
          name: random
      ports:
        - containerPort: 8080
```

No more K8s  
**Service** or  
**Ingress**  
required!

# Google Cloud Run



- Two variations:
  - Google Cloud Run → Knative API implemented on App Engine
  - Google Cloud Run For Anthos → Knative on Kubernetes
- Billed by 100ms
- Not all Knative features implemented
  - see <https://ahmet.im/blog/cloud-run-is-a-knative> (?)

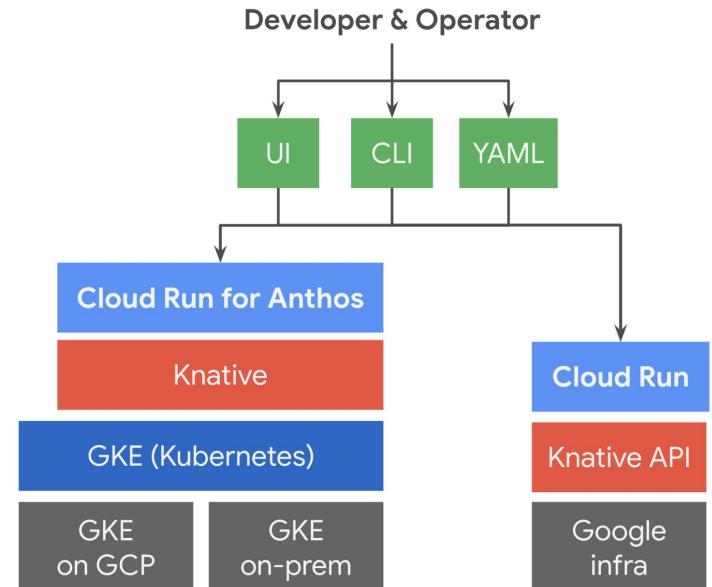
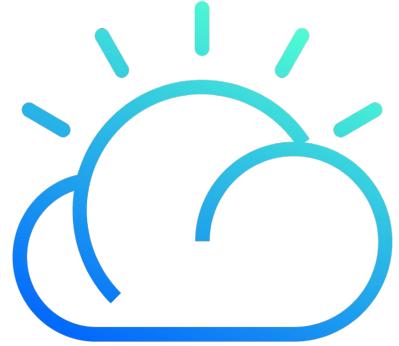
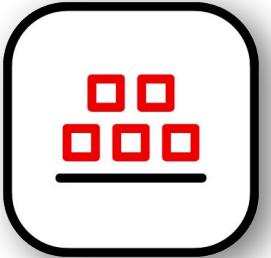


Diagram: @ahmetb

# IBM Cloud Code Engine

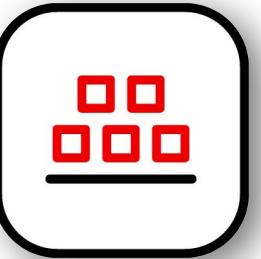


- Knative on the IBM Cloud
- Support for Batch Jobs
- In-Cluster container builds via Build Packs



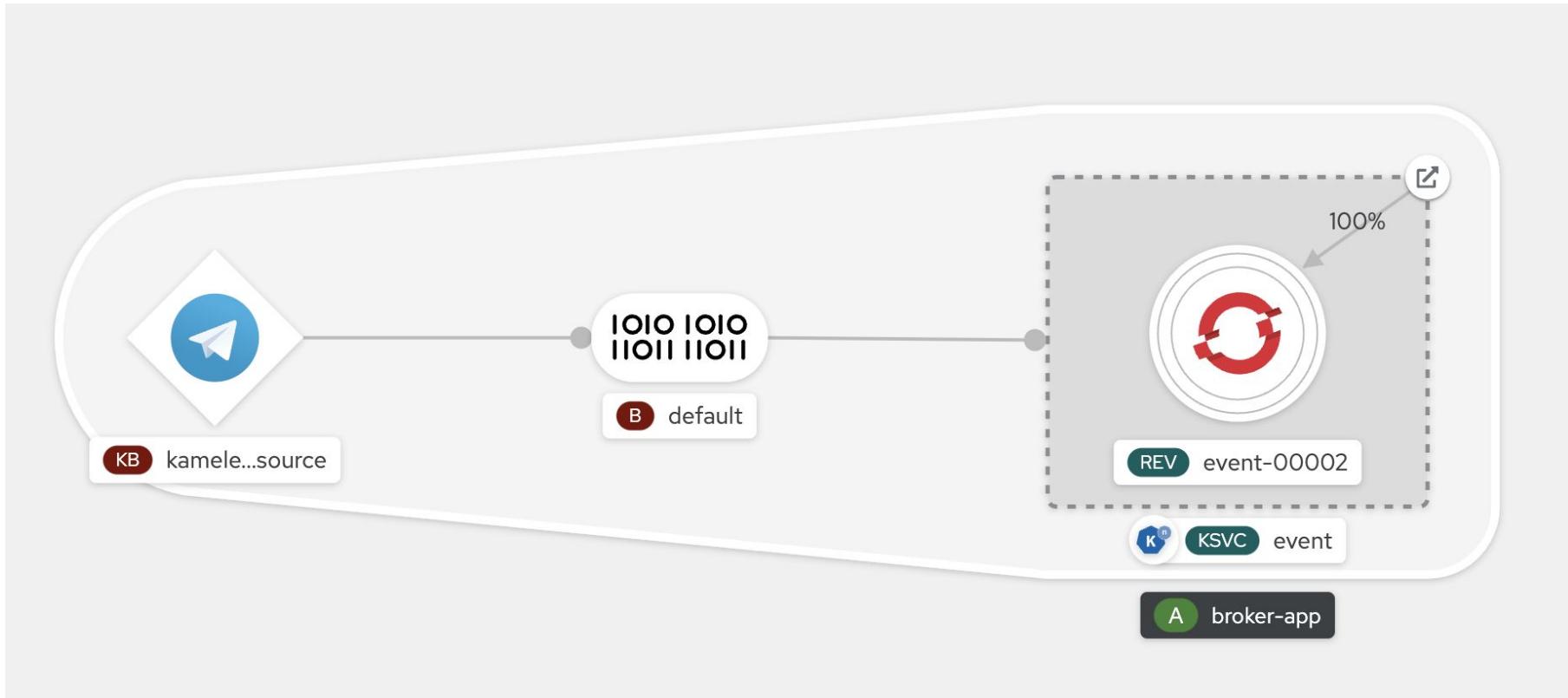
# Red Hat OpenShift Serverless

- Knative bundled with Red Hat OpenShift
- Runs everywhere where OpenShift runs
  - on-prem
  - AWS (ROSA)
  - Azure (ARO)
  - IBM (ROKS)
- Adds operator-based installation experience
- Kafka-backed Knative broker
- 50+ sources via Apache Camel connectors (Kamelets)



# Red Hat OpenShift Serverless

- Great visualization in the OpenShift Console



# Cloud Native Runtimes for VMware Tanzu



- Knative distribution on-top of Kubernetes
- Integral part of the Tanzu Application Platform (TAP)
- Eventing Broker is backed by RabbitMQ
- Includes TriggerMesh Sources for Amazon Web Services (SAWS)

# Challenges



# Coldstart

- Application startup times in Kubernetes:

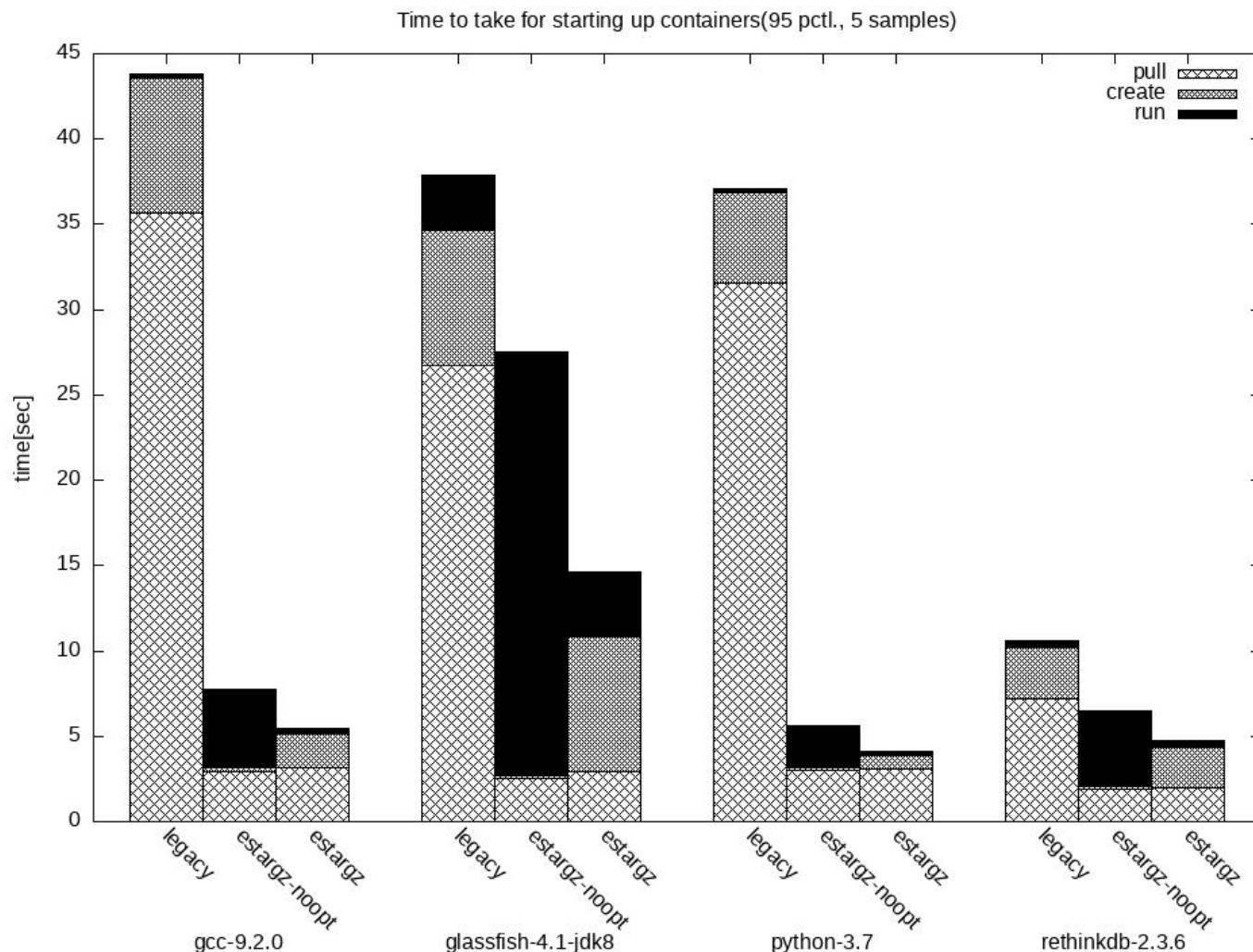
Pod Scheduling 0.1 - 10s	Image Pull 0- 10s	Container Startup <1s	Application Startup 0.01 - 5s	Pod Ready 0.1-1s
-----------------------------	----------------------	-----------------------------	----------------------------------	---------------------

- Scheduling time depends on cluster size and can take [10s for a 1000 nodes cluster](#)
- Application startup depends on the application stack
  - Fast for optimized frameworks like Quarkus or Node
  - JavaEE servers are no good fit
- Pod serves requests when readiness probes succeed
  - Kubelet has a 1s granularity for checking probes

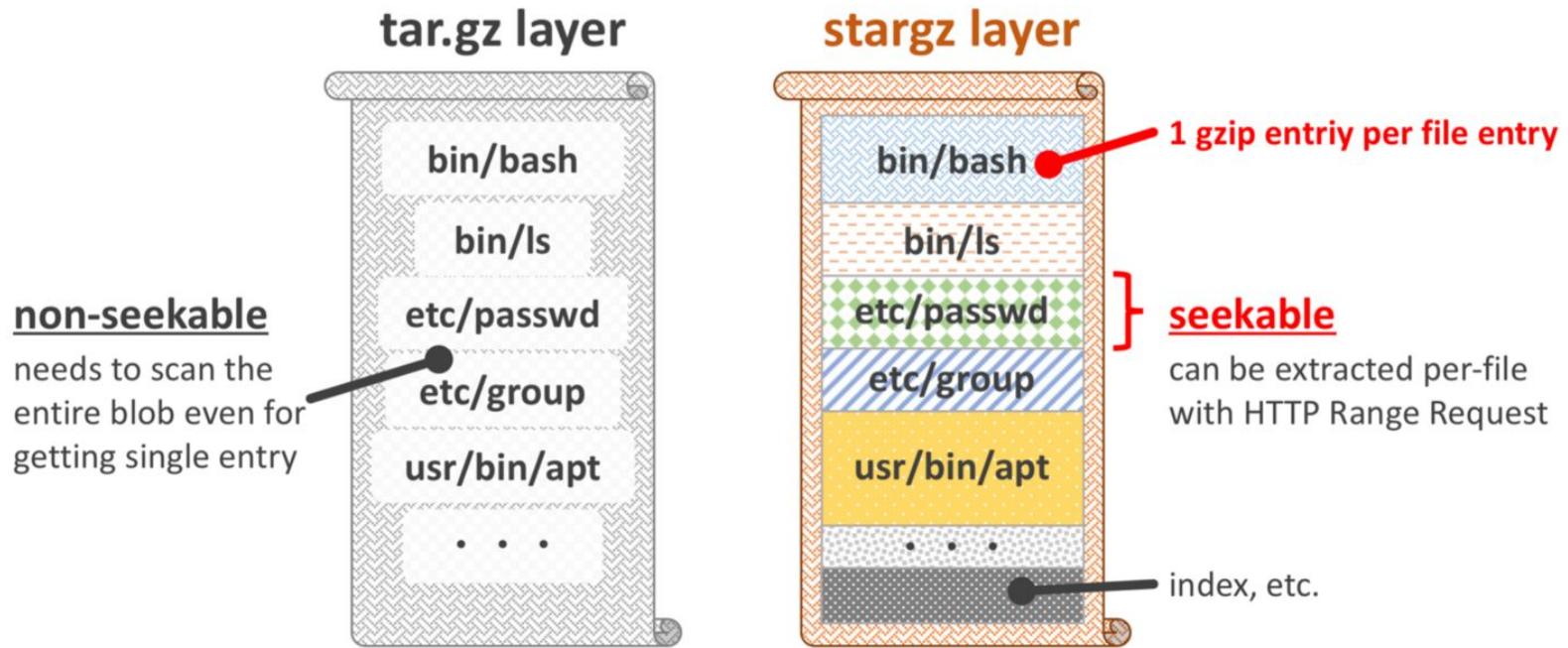
# Image Pull

- Application image needs to be pulled from a container image registry
  - Cached at the Container Runtime for future usage
  - Workarounds: Pre-warm with a Daemonset
- Lazy pulling with stargz snapshotter

# containerd/stargz-snapshotter

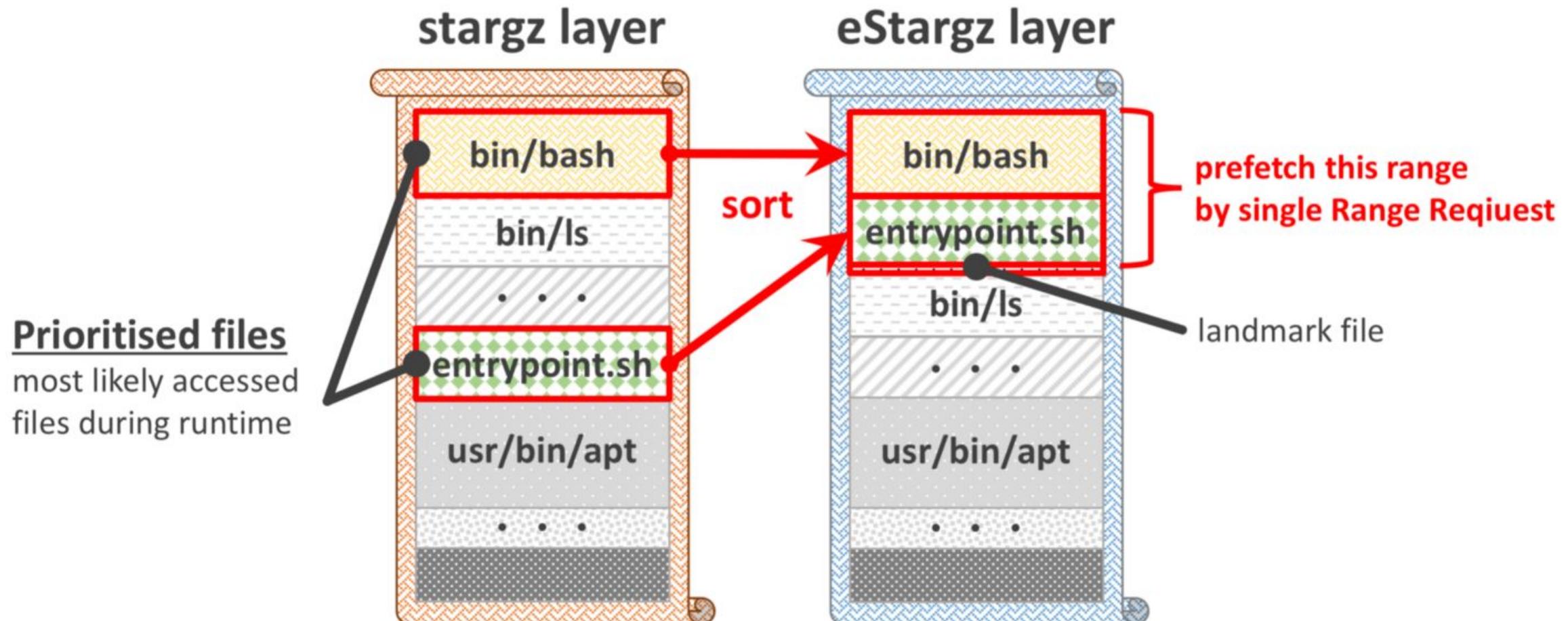


# Stargz (“seekable tar.gz”)



- tar: Non-indexed, gz: Non-Seekable
- Solution: Gzip tar-entries individually, concatenate and add gzipped-index file and tar footer

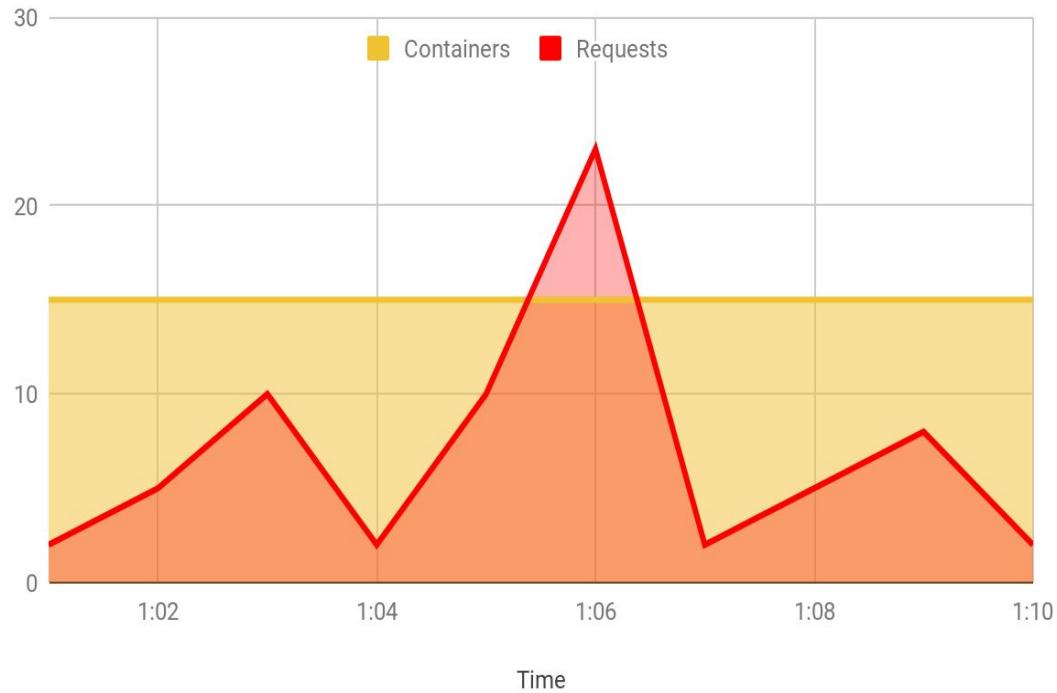
# eStargz Optimization



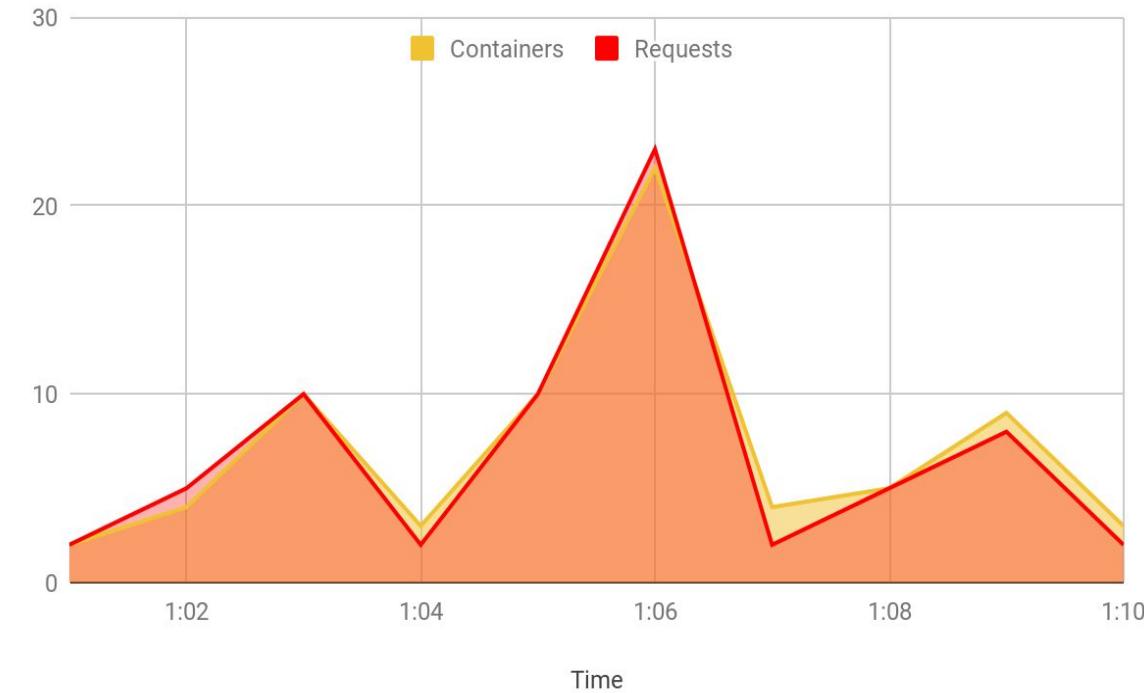
# Possible Startup Optimizations

- Pooling of network interfaces
- Move the Pod Lifecycle Event Generator (PLEG) to an event-driven model instead of 1s polling (requires change to the CRI)
- Precreated pool of “Pod hulls”
- More pooling in general
- See [Cold start latency improvements in Kubernetes and Knative](#) for more details

# Autoscaling



NOT Serverless

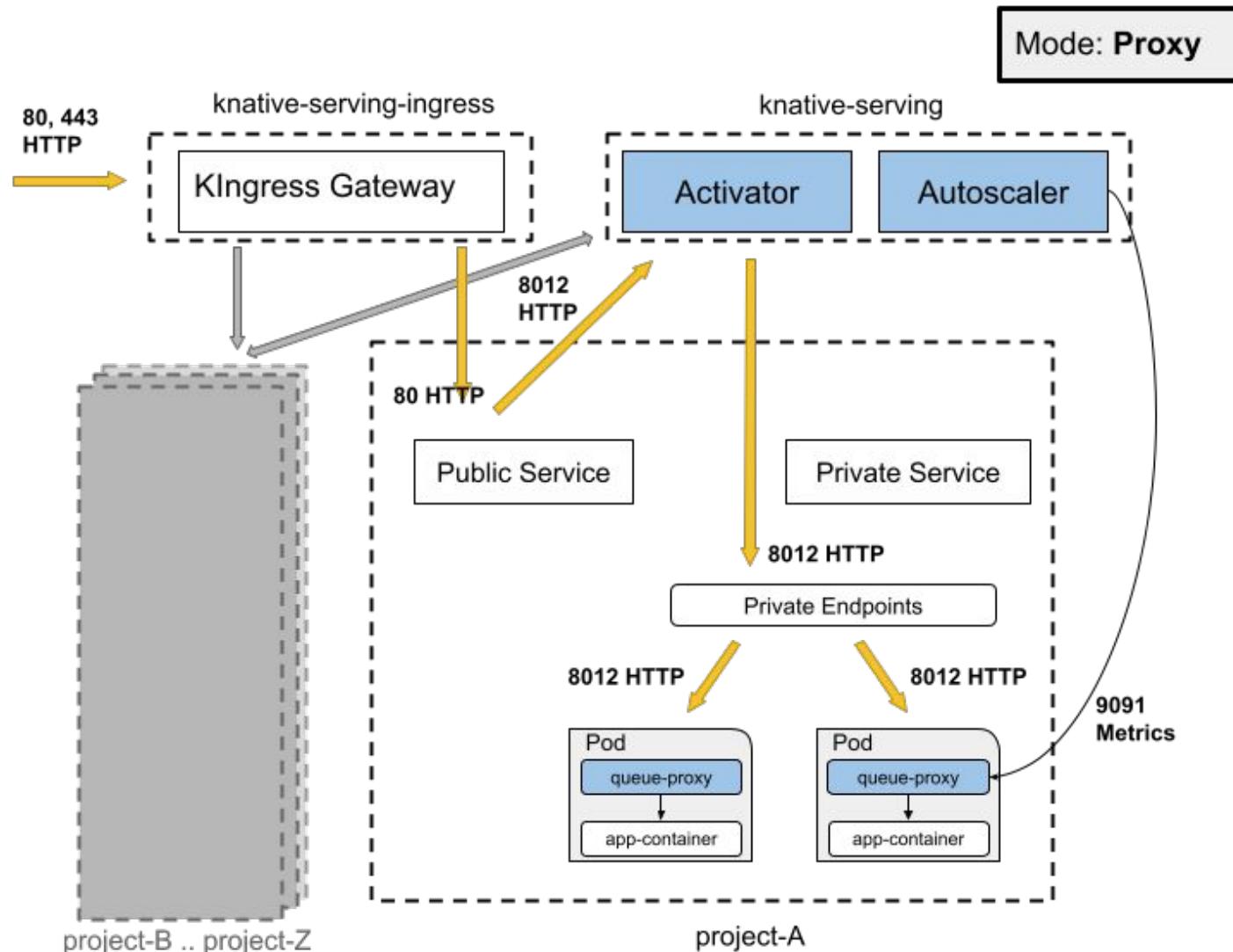


with Serverless

# Autoscaling Options

- Kubernetes Native:
  - Horizontal Pod Autoscaler (HPA)
  - Vertical Pod Autoscaler (VPA)
  - Cluster Autoscaler (CA)
- Knative Autoscaler
- KEDA

# Knative Autoscaling



# Pros & Cons



# Container Drawbacks

- Container Image needs to be **built**
  - Explicit via Docker/Buildah tooling
  - Automatically by language specific tooling
    - [Cloud Native Buildpacks](#), [Shipwright](#), [Eclipse JKube](#) ...
- **Coldstart** times
- Extra overhead because of **additional abstraction**

# Container Benefits

- **Standard** format
- **No vendor locking** per se
- No language restrictions
- Rich tooling landscape
- Hybrid scenarios (on-prem combined with cloud) easily possible
- Versioning
- Powerful orchestration platforms underneath (Kubernetes)

# The Future



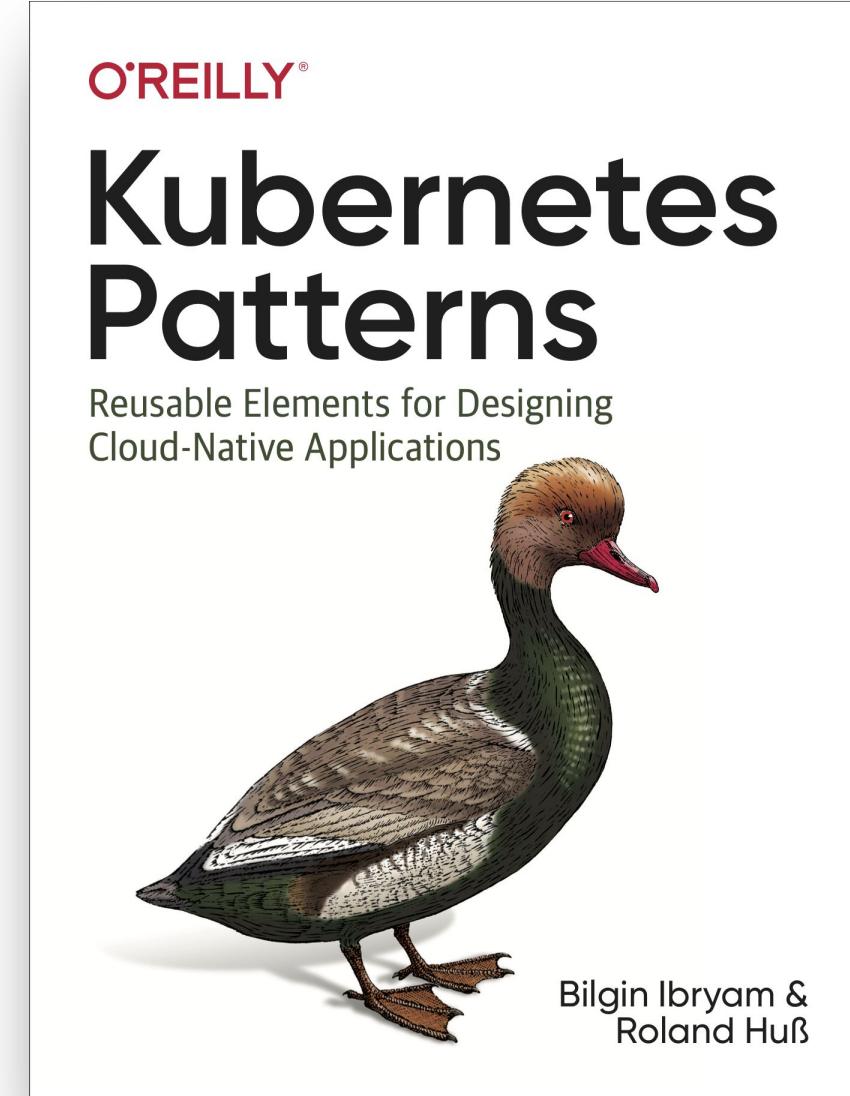
# What's (maybe) next ?

- **Web Assembly** (WASM) as a popular deployment format for serverless functions
  - e.g. like in [Krustlets](#)
- **Serverless Kubernetes Control-Plane**
  - so that optional features don't use resources
- **Sub-second cold start** times become a reality
- **Scale-to-Zero** as a **core** Kubernetes feature
- **Hybrid Serverless** with multi tenancy

# Thank you



@ro14nd



<https://k8spatterns.io>

# Picture Credits

<https://www.pexels.com/photo/boat-island-ocean-sea-218999/>

<https://unsplash.com/photos/UGMf30W28qc>

<https://pixabay.com/photos/hamburg-speicherstadt-channel-297671/>

<https://pixabay.com/photos/beer-machine-alcohol-brewery-1513436/>

<https://www.pexels.com/photo/people-art-abstract-blur-6077519/>

<https://www.pexels.com/photo/person-holding-a-glass-ball-1252893/>

<https://me.me/i/aws-lambda-is-just-glorified-cgi-bin-imgflip-com-change-my-mind-d0b715592ba34b08b79452ad02783ca2>

<https://unsplash.com/photos/vkQgb1lZZPQ>

<https://unsplash.com/photos/9JrBiphz0e0>

<https://www.pexels.com/photo/person-rock-climbing-3077882/>