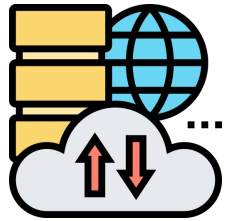
COEN 241
Introduction to Cloud Computing

Lecture 9 - FaaS/Serverless



Lecture 8 Recap

- Kubernetes
  - Concepts
  - Architecture
- Minikube
- Readings
  - Recommended: None
  - Optional:
    - https://kubernetes.io/docs/tutorials/kubernetes-basics/
    - https://labs.play-with-k8s.com/
    - https://minikube.sigs.k8s.io/docs/handbook/
    - https://circleci.com/blog/what-is-yaml-a-beginner-s-guide/

What is Kubernetes?

- An open-source container-orchestration system for automating computer application deployment, scaling, and management.
- It is a Greek word meaning helmsman or pilot.
- Often called K8s, which is derived by replacing the eight letters of “ubernete” with the digit 8.
- Project emerged from Google in 2014; v1.0 released 2015
  - Related to Borg—Google’s (secret) internal container scheduler
  - K8s is implemented in Go, in contrast to Borg’s C++

Why is Kubernetes Useful?

- Provides an easy way to scale your containers
- Avoids cloud vendor lock-in
- Allows easy multi-cloud adoption
- Better management of application
- Works with many different container technologies
- It is a BIG piece of software with lot of moving parts
  - We will give a high-level overview of it

Some Kubernetes Terms

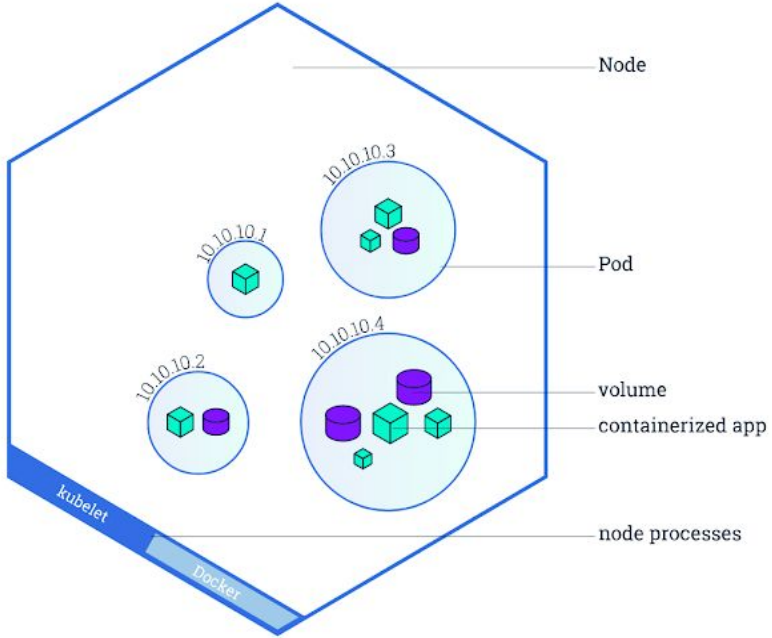
- **Pods:** tightly coupled set of containers; smallest unit of scheduling
- **Node:** A Pod always runs on a Node. A Node is a worker machine in Kubernetes and may be either a virtual or a physical machine,
- **Services:** An abstraction which defines a logical set of Pods and a policy by which to access them
  - Sometimes this pattern is called a micro-service
- Volumes: persistent storage; can share between containers
- Namespaces: multiple virtual cluster in a same physical cluster

Kubernetes Pods

- Pods are the smallest deployable units of computing that you can create and manage in Kubernetes
- Group of one or more containers with shared storage and network resources, and a specification for how to run the containers
  - A group of Docker containers with shared namespaces and file system volumes
- A Pod's contents are always co-located and co-scheduled, and run in a shared context

Kubernetes Nodes

- A set of worker machines that run containerized applications
- Every cluster has at least one worker node
- When you deploy K8s, you get a **cluster**
- `kubectl get nodes`



Kubernetes Services

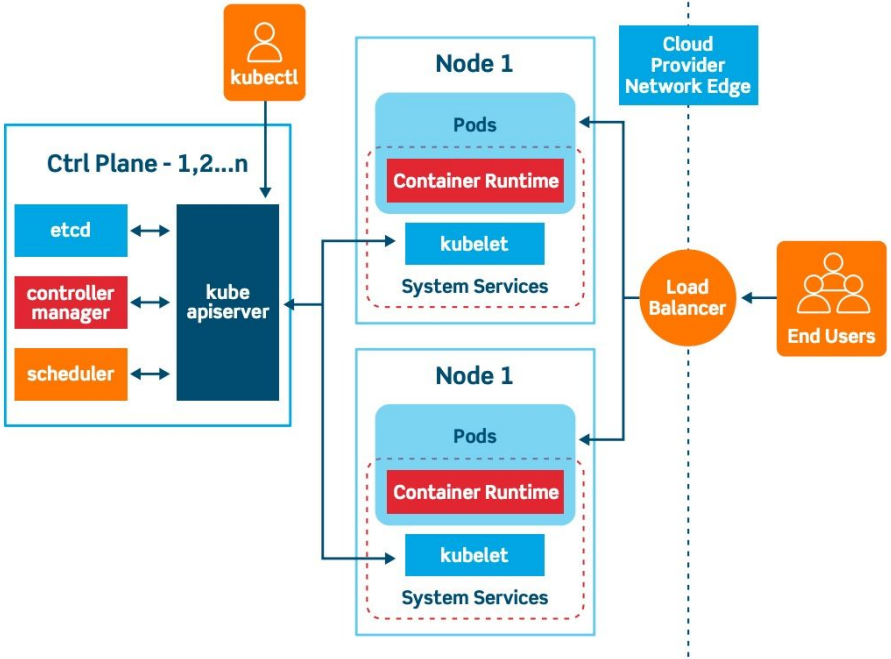
- A **logical abstraction** for a deployed group of pods in a cluster
  - which all perform the same function
- Kubernetes way of configuring a proxy to forward traffic to a set of pods
- Since pods are ephemeral, a service enables a group of pods to be assigned a name and unique IP address (clusterIP)
- As long as the service is running on that IP address, it will not change.
- Services also define policies for the access to Pods

Stateless vs Stateful Applications

- Stateless applications scale easily: just start more pods
  - E.g., web servers presenting read-only workloads
- Stateful apps are more difficult, e.g.:
  - Databases having primary and secondary instances
  - Machine Learning Applications
- Kubernetes StatefulSet controllers enables stateful applications
  - Provides durable storage
  - Persistent ID for each Pod
  - https://kubernetes.io/docs/tutorials/stateful-application/basic-stateful-set/

Kubernetes Architecture Overview

- Kubernetes cluster consists of:
- Master Node(s) (control plane)
  - With a distributed storage (etcd)
- A number of cluster nodes



Minikube: A local/test setup

- https://minikube.sigs.k8s.io/docs/start/
- Running both master and worker processes on the same node with a minimum requirement:
  - 2 CPUs or more, 2GB of free memory, 20GB of free disk space
- Use **kubectl** to interact with the Minikube cluster
- **Kubectl:** A CLI for Kubenetes cluster to talk to the master
  - Not only Minikube, but any type of Kubernete cluster
  - https://kubernetes.io/docs/reference/kubectl/overview/

Agenda for Today

- Function as a Service
- OpenFaaS
- Readings
  - Recommended: CCSA Chapter 4
  - Optional:
    - https://aws.amazon.com/solutions/case-studies/netflix-and-aws-lambda/

Function as a Service (FaaS)

What is Function as a Service?

- A cloud computing platform that allows customers to run and manage applications **without the complexity of building and maintaining the infrastructure** needed to run the application.

- Started in 2010 by a start up called PiCloud
  - https://techcrunch.com/2010/07/19/picloud-launches-serverless-computing-platform-to-the-public/
  - Acquired by Dropbox in 2013

- AWS Lambda is the first public cloud to offer FaaS in 2014
  - Followed by Google Cloud Functions, Microsoft Azure Functions, IBM/Apache's OpenWhisk in 2016 and Oracle Cloud Fn in 2017

What is Function as a Service?

- Developers write independent code scripts known as “functions” and upload those functions to the FaaS
  - That’s why we call it “Function”-as-a-Service
- Once uploaded, the functions can be then triggered by some event or run on a schedule

FaaS Example (in Python)

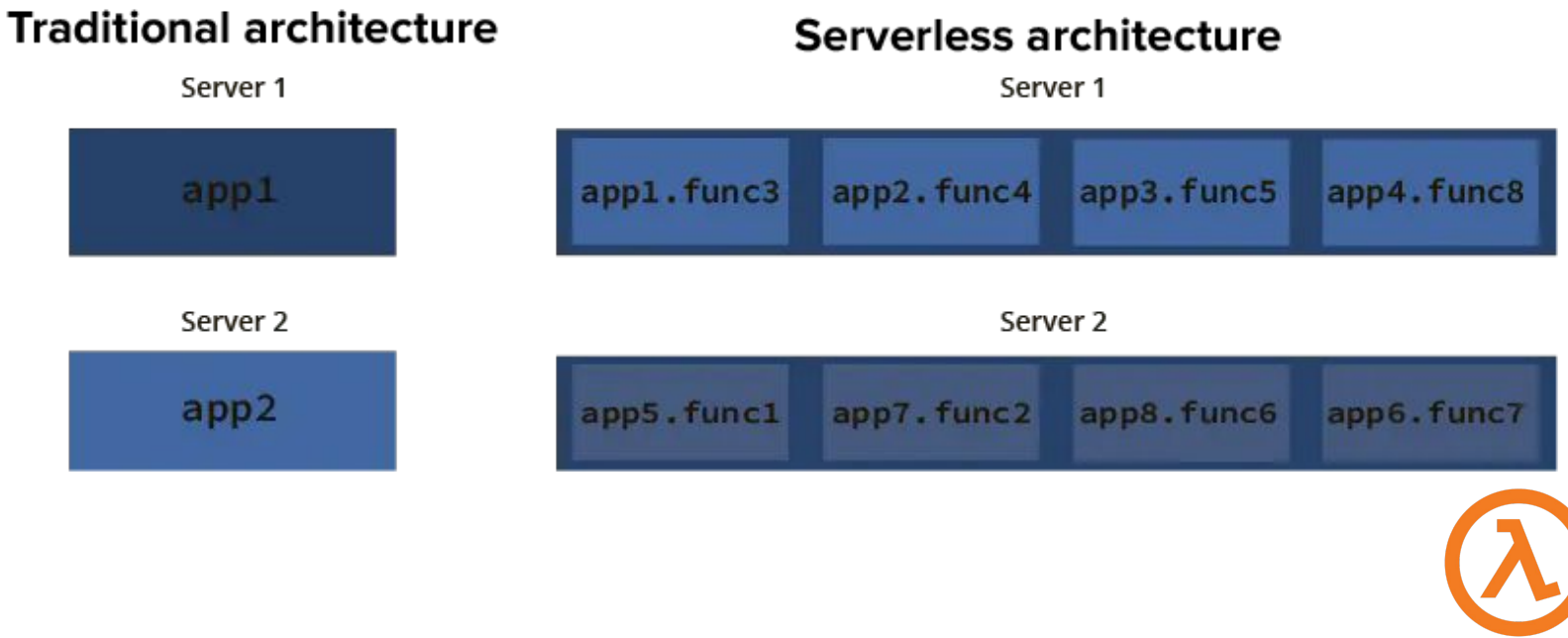
```
import os
import json

def lambda_handler(event, context):
    return {
        "statusCode": 200,
        "headers": {
            "Content-Type": "application/json"
        },
        "body": json.dumps({
            "message ": “Hellow World”
        })
    }
```

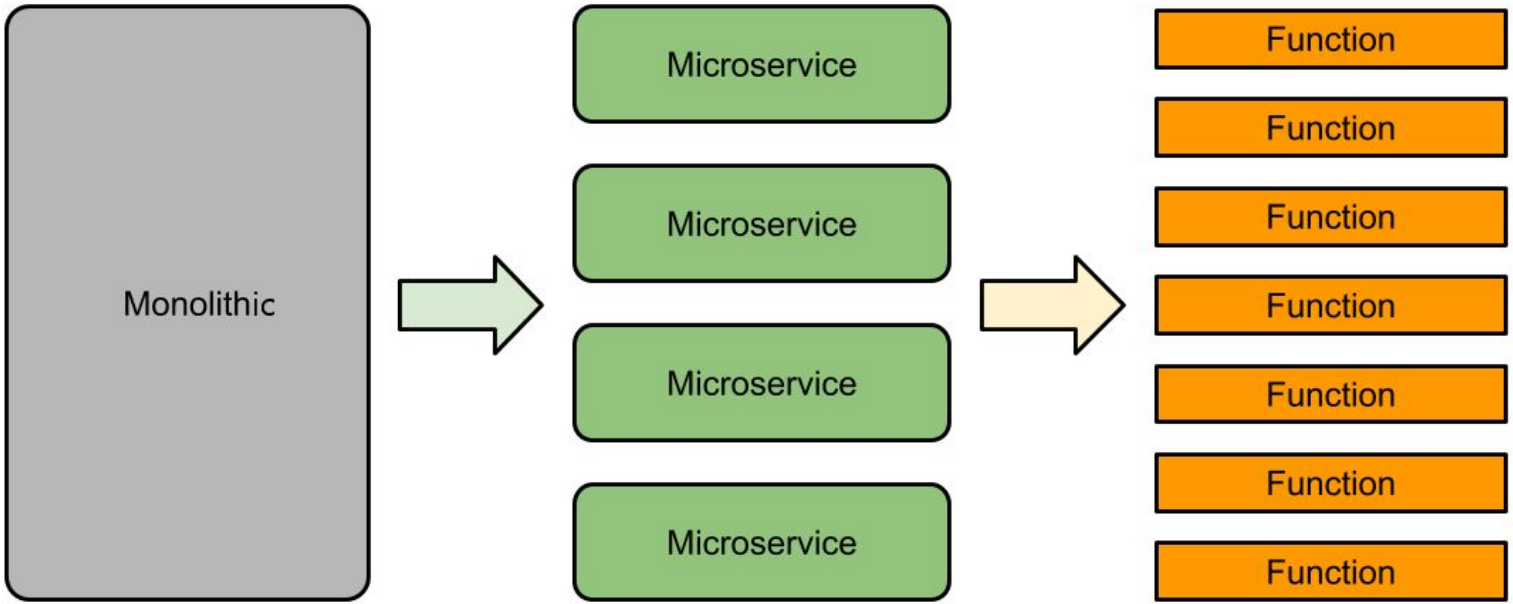
Serverless vs FaaS

- Serverless is a **cloud computing execution model** that abstracts away much of the complexity associated with infrastructure management from developers
- FaaS is a common form of serverless computing
  - i.e., Trend away from needing to considering servers at all
- FaaS is a platform providing serverless architecture deployment, orchestration and management

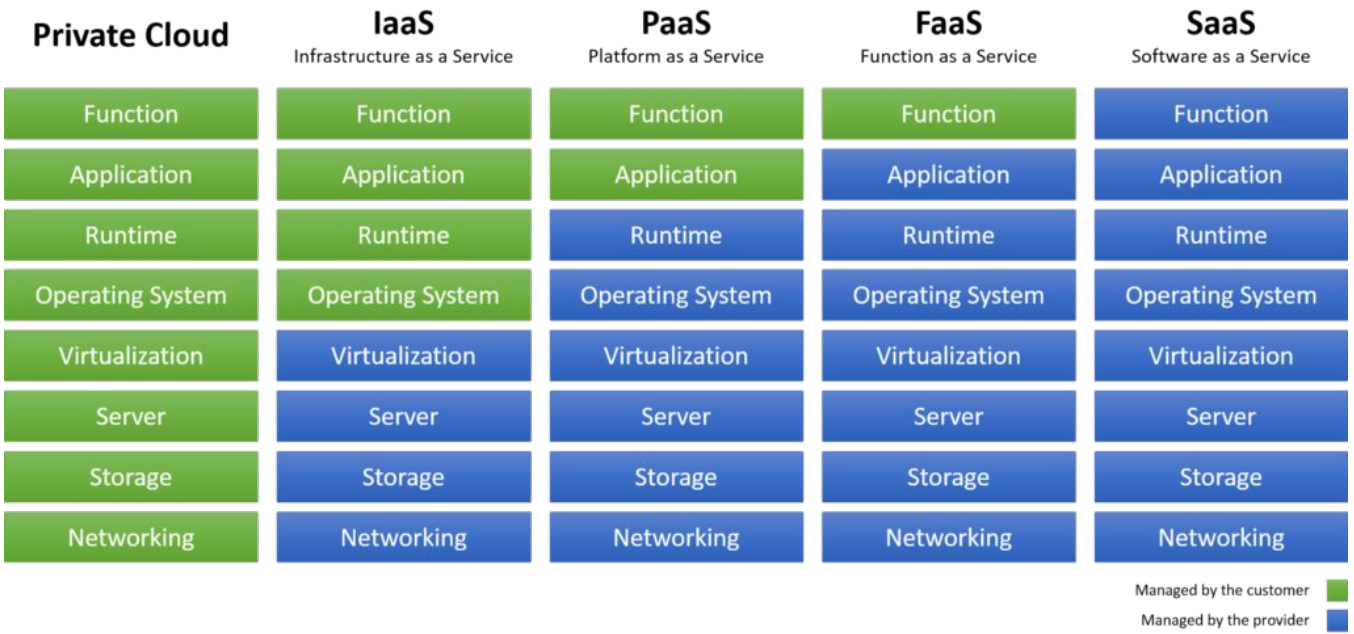
Function as a Service vs Others



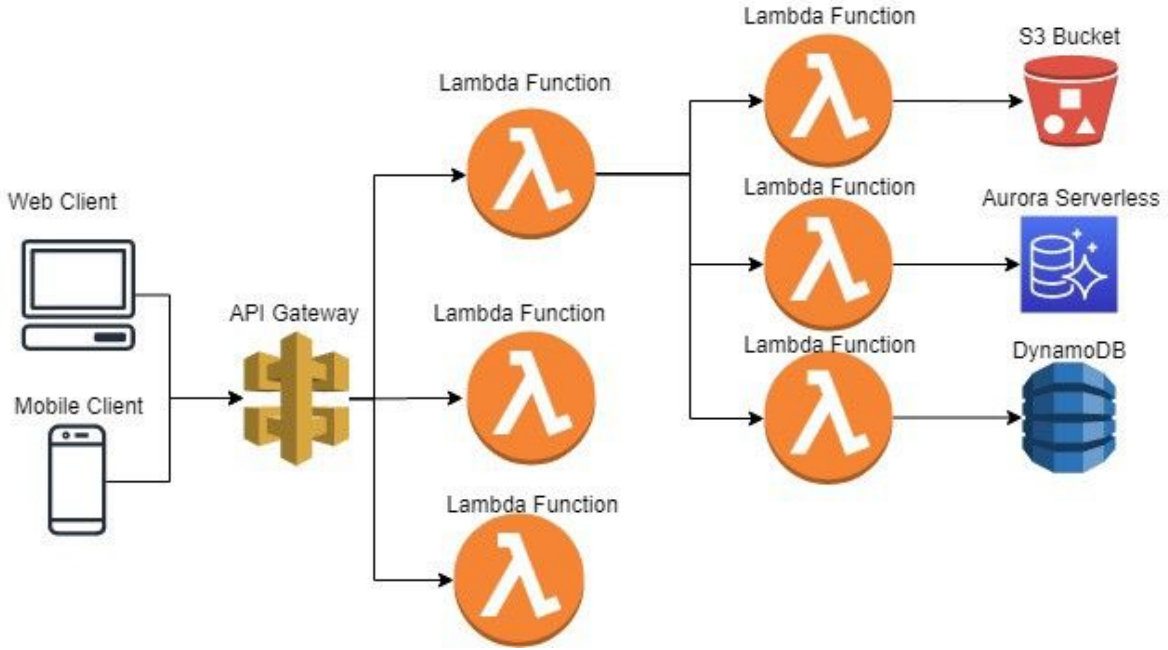
Function as a Service vs Others



Function as a Service vs Others



Function as a Service Architecture



Function as a Service Architecture

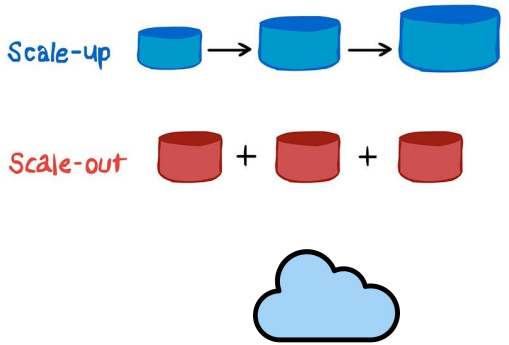
- Functions in FaaS are “stateless”, which means they cannot save and share session states between two different runs of the same function or different functions
  - Must use an external means of storing states
- Function can call other serverless functions as part of an application.
  - Need to communicate with other functions using asynchronous messaging

FaaS Use Cases

- Many use cases
  - Web apps / Backends
  - Chatbots
  - Scheduled tasks
  - IT Automation
  - **Distributed stream processing systems (DSPS)**
  - **Event-driven programming**
  - **Internet of Things (IoT)**
  - **Edge Computing**
- Involving **short**, **self-contained** tasks with low memory requirement

Distributed Stream Processing Systems (DSPS)

- Traditional DBs vs DSPSs
  - Databases: Runs queries when instructed to do so
  - DSPS: Computing is triggered by new tuples appearing
- Stream processing systems define a data flow graph
  - Data Source -> Operators -> Sinks
  - Operators: nodes that transform input streams to outputs
- Many high-quality, “scale-out”, open source DSPSs:
  - Apache Storm, Apache Spark Streaming, Apache Flink, …
  - Scale-out: Adding-more resources
  - Scale-up: Adding power to given resource



Event-Driven Programming

- In IaaS, VM is yours, so your code is always running
- In PaaS, still usually a sense that your code is active
  - PaaS auto-scales the server instances that run your code
- In FaaS, your code operates in a **reactive style**
  - Reactive programming typically relies on callbacks
  - Some sort of shared event dispatcher issues callbacks
  - FaaS may have wide variance in function execution latency
    - Why?

FaaS and Internet of Things

- IoT embodies ambition of all devices being networked
  - Devices including cars, streetlights, toasters, wireless sensors, etc.
- For sensor networks, want to offload data processing
  - To extend lifetime of battery-powered devices
  - Sensor data will be disseminated periodically, often at random intervals
- FaaS facilitates data transformations
  - Provides reliable endpoints for IoT devices to interact with on demand

FaaS and Edge Computing

- Edge computing sits in between cloud and IoT
  - Often consists of full-size computing devices, widely distributed
  - ... but not at cluster scale (so not scale-out edge computing)
- Example: Content Delivery Networks (CDN)
  - Geographically distributed servers that speed up the delivery of web content by bringing it closer to where users are
- CloudFront CDN runs AWS Lambda functions
  - E.g., personalize web content within any AWS Region's DC
- Assist in short tasks without traveling all the way to the DCs

Advantages of Function as a Service

- Increased developer productivity and faster development time
  - Server infrastructure management is handled by someone else
- Easy to scale and horizontal scaling is managed by the platform
- Only pay for or consume resources when necessary and as needed
  - Never pay for idle resources
- Applications can be written in almost any programming language
- Built in availability and fault tolerance

Disadvantages of Function as a Service

- Decreased transparency. Hard to understand the entire system.
  - Potentially tough to debug
- Auto-scaling of function calls often means auto-scaling of cost.
  - Makes it tough to estimate cost
- Can be tough to keep track of many functions.
- Stateful applications are harder to build
- Hard to pack dependencies!

Disadvantages of Function as a Service

- **Time and Resource Limitations**
  - AWS
  - Azure
  - Google
- Cold start Lag
- Vendor Lock-In
- Different Programming Paradigm, requiring time to migrate

FaaS vs Containers

- User's Applications
  - FaaS: User creates a program code , specifies needed dependencies and uploads it. The provider then provisions the computing environment
  - Container: User creates a container definition file that specifies apps and configurations. The container is initiated from the image created by the user.
- Uptime
  - FaaS: Only when the code is running
  - Container: When the container is running
- Payment for the resources
  - FaaS: Only when the code is running
  - Container: A server is needed to run the container

FaaS vs Containers

- Runtime Limitations
  - FaaS: Limited
  - Containers: Unlimited
- Hosting specificity
  - FaaS: Better suited for microservices
  - Containers: Suited for any type of services
- Programming languages support
  - FaaS: Limited
  - Containers: Unlimited

AWS Lambda

- AWS Lambda (2014) was first successful FaaS
- AWS Lambda supports many programming languages:
  - Python, Java, Node.js, Go, Ruby, and C# (.Net core)
  - Even Linux executables!
- Aims for millisecond startup latency
  - Caching containers will likely mean significant speed-up from recent use
  - Warm-boot

AWS Lambda Pricing

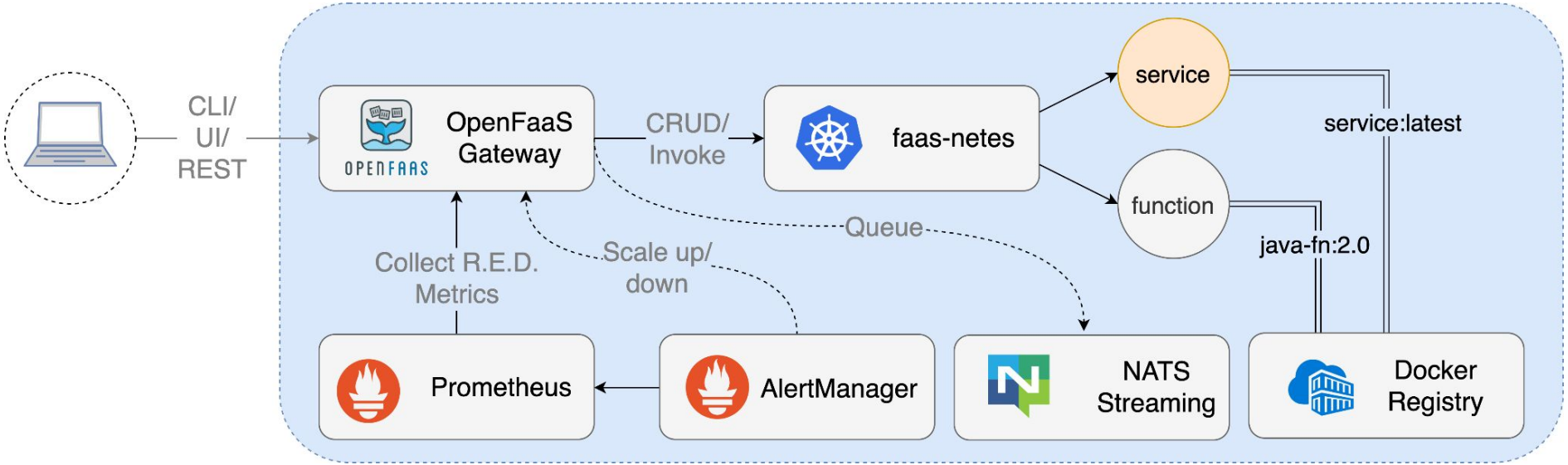
- Pricing based on **number of requests** and their **duration**
  - Request cost is $0.20 per million per month (US East)
  - … but the first million requests per month are free
- Duration cost is $0.0000166667 per **GB-second**
  - So involves both time and allocated memory you've chosen
  - … but the first 400,000 GB-seconds per month are free
  - Memory allocation can be as low as 128 MB
  - (So the free tier will go a long way, for small-scale applications)
- Welcome to try it out!

OpenFaaS

What is OpenFaaS?

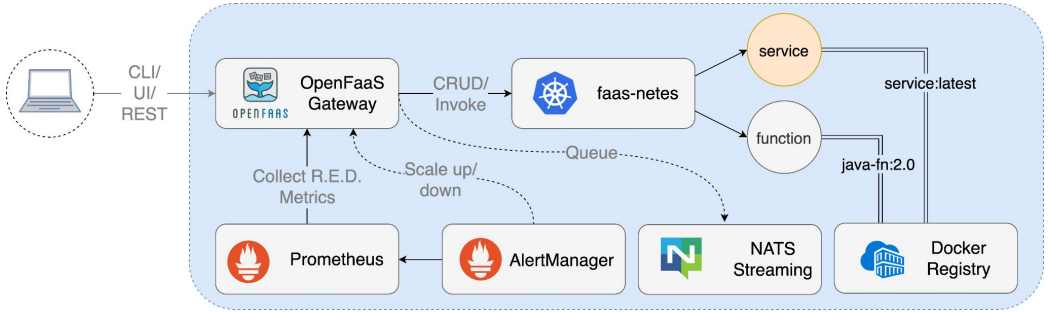
- An open-source framework for building and running serverless applications on your own infrastructure
  - Public cloud providers do not disclose a lot about their infrastructure
- Originated from the serverless framework in Docker Swarm and now supports other kinds of infrastructure backends, such as **Kubernetes**
- **Functions in OpenFaaS are containers**
  - Any program written in any language can be packed as a function by leveraging the container technologies of Docker.

OpenFaaS Infrastructure



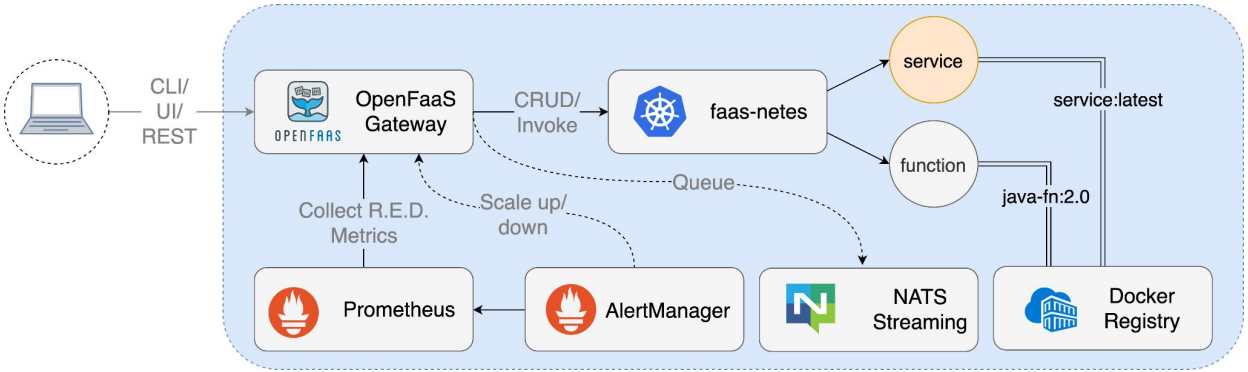
OpenFaaS Infrastructure

- OpenFaaS consists of
  - An API gateway
  - CLI
  - Function watchdog
  - Prometheus
  - Kubernetes (or faasd) orchestration engines
  - Containers for Running the functions



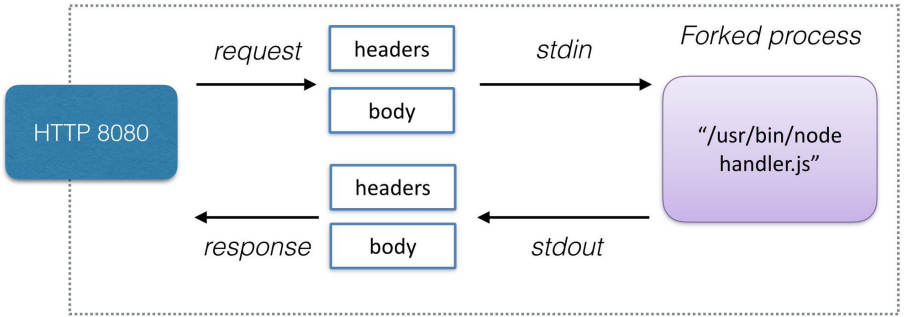
OpenFaaS Infrastructure

- CLI: Either curl or faas-cli to connect to the API Gateway
- API Gateway: Receives user requests via the CLI or from web and invokes the functions via faas-netes (Kubernetes)



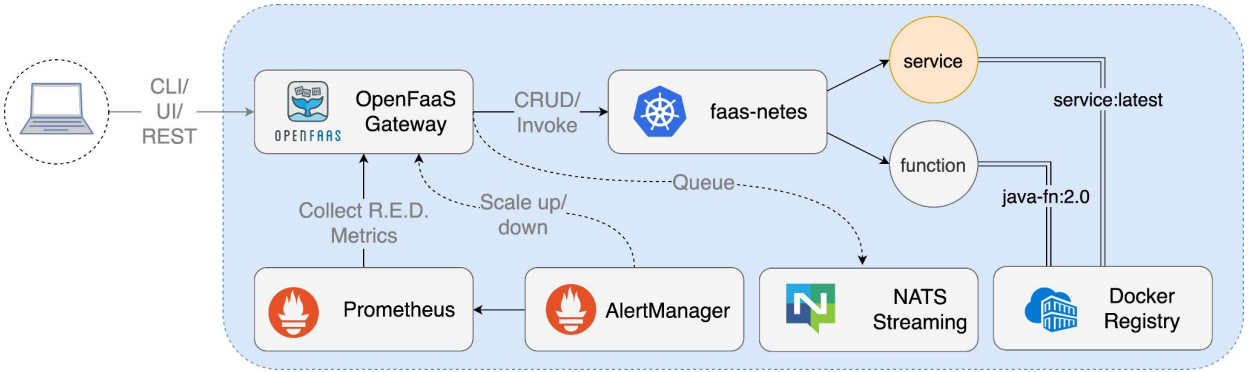
OpenFaaS Infrastructure

- Function Watchdog: Responsible for starting and monitoring functions
  - Any binary can become a function through the use of watchdog.
  - Requirement (now deprecated)
    - Accept input via the standard input (stdin) and print out the result to the standard output (stdout).
    - of-watchdog allows alternative to STDIO for communication between the watchdog and the function.



OpenFaaS Infrastructure

- Prometheus: Separate monitoring and time-series database
  - https://prometheus.io/
- AlertManager: Reads usage (requests per second) metrics from Prometheus and fires alerts to the API Gateway for auto-scaling



OpenFaaS Installation

- How to install OpenFaaS?
  - https://docs.openfaas.com/cli/install/
  - PLONK(F) Stack
    - Prometheus, **Linux,** OpenFaaS, NATs, faasd
  - https://docs.openfaas.com/deployment/
  - https://github.com/openfaas/faasd/
- DO NOT USE Native Mac or Windows
  - Use a Linux subsystem or a VM on QEMU or Virtualbox
- We will use multipass (also will be used for HW 2)
  - https://multipass.run/

TODOs!

- Kubernetes Quiz!
- HW 2
- Study for Midterms!

Agenda for Today

- Function as a Service
- OpenFaaS
- Readings
  - Recommended: CCSA Chapter 4
  - Optional:
    - https://aws.amazon.com/solutions/case-studies/netflix-and-aws-lambda/

Questions?