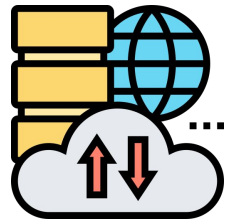
COEN 241
Introduction to Cloud Computing

Lecture 10 - OpenFaaS Deep Dive



Lecture 9 Recap

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

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

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 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

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.

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

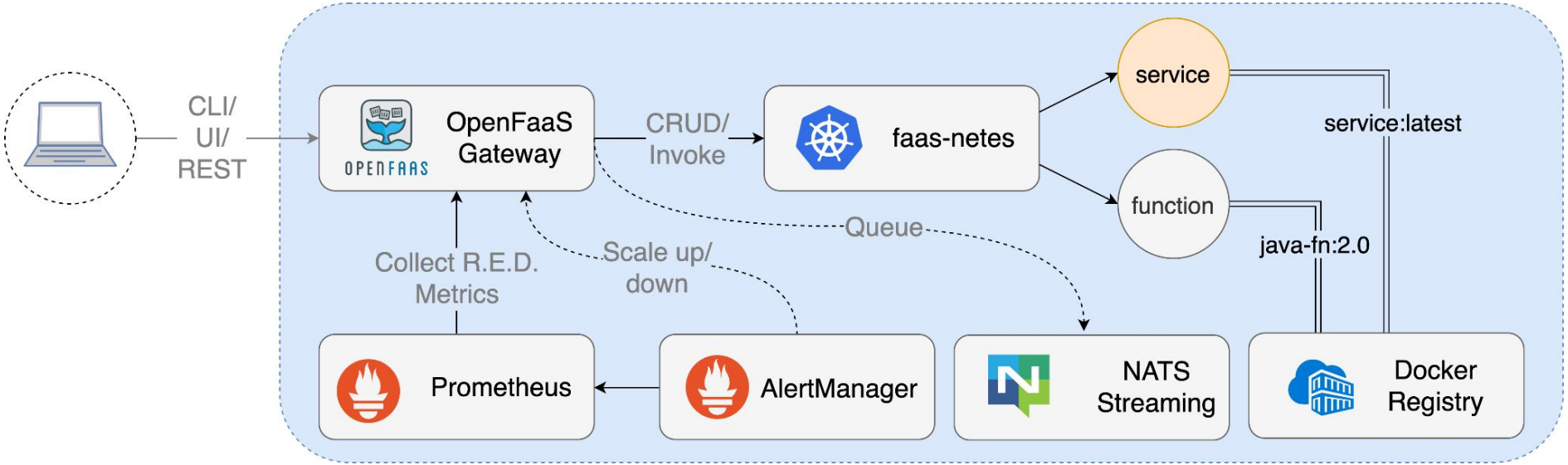
FaaS vs Containers

- 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

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



Agenda for Today

- OpenFaaS Demo
- Sustainable Data Center
- Midterm Preview
- Readings
  - Recommended: None
  - Optional:
    https://www.cncf.io/blog/2020/04/13/serverless-open-source-frameworks-openfaas-knative-more/
  - https://tech.fb.com/engineering/2020/01/hyperefficient-data-centers/
  - https://natick.research.microsoft.com/

OpenFaaS Demo

- 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
- Today's demo will be on faasd installation
  - Difference with kubernetes based OpenFaas will mostly be the installation and how you access each components (i.e., UI, Prometheus)

Creating/Using OpenFaaS Functions

- How to use existing OpenFaaS functions

```
$ faas-cli store list
$ faas-cli store deploy figlet
$ faas-cli store deploy hubstat
```

- How to write a new OpenFaas function

```
$ mkdir -p ~/functions && cd ~/functions
$ faas-cli new --lang python hello-python
```

- What does the output look like after the above command?

Creating/Using OpenFaaS Functions

- Edit the handler file

```
def handle(req):
    print("Received request: " + req)
```

- Edit the yaml file
  - gateway: We can specify a remote gateway, what the programming language is and where our handler is located within the filesystem.
  - functions: Defines the functions specs
  - lang: language of the function
  - handler: this is the folder / path to your source
  - image: Docker image name. If you are going to push to Docker Hub add the prefix of your Docker Hub account. i.e. sean/hello-python

```
provider:
  name: faas
  gateway: http://localhost:8080

functions:
  hello-python:
    lang: python3
    handler: ./hello-python
    image: hello-python
```

Creating/Using OpenFaaS Functions

- Build the function

```
$ sudo faas-cli build -f ./hello-python.yml
...

Successfully tagged hello-python:latest
Image: hello-python built.
```

- Check Docker Images

```
$ sudo docker images | grep hello-python
hello-python        latest       abcdefghij     one minute ago
```

Creating/Using OpenFaaS Functions for ARM64

- Build the function

```
$ sudo DOCKER_BUILDKIT=1 faas-cli up -f ./hello-python.yml  --no-cache
--build-arg platform=linux/arm64/v8
...

Successfully tagged hello-python:latest
Image: hello-python built.
```

- Check Docker Images

```
$ sudo docker images | grep hello-python
hello-python        latest       abcdefghij     one minute ago
```

Push the OpenFaas Function to Docker Hub

- Register for Docker Hub
- Log into Docker Hub
  - Must use the same username and password

```
$ sudo docker login
```

- All your images will now be deployed to your user name
  - E.g. <user_name>/<repostory_name>:<tag>

Push the OpenFaas Function to Docker Hub

- Push the function

```
$ sudo faas-cli push -f ./hello-python.yml
...

latest: digest:
sha256:97aac9eb09b60de93baac49a05c986c6c078c1f8c7e4df3f8d31e6b1d57d60c0 size:
4693
[0] < Pushing hello-python [<user_name>/hello-python:latest] done.
[0] Worker done.
```

- Check on Docker Hub

Deploy OpenFaaS Functions

- Deploy the function

```
$ sudo faas-cli deploy -f ./hello-python.yml
Deploying: hello-python.
No existing service to remove

Deployed. 200 OK.
URL: http://127.0.0.1:8080/function/hello-python
```

Creating/Using OpenFaaS Functions

- Test your function

```
$ curl localhost:8080/function/hello-python -d "I'm COEN 241"
Received request: I'm COEN 241
```

- Try the same thing with faas-cli

```
sudo faas-cli list
echo "I'm COEN 241" | faas-cli invoke hello-python
```

Looking into OpenFaaS Web UI

- Requires port forwarding if running faasd inside a VM
  - After forwarding the port 8080, you can access it via localhost:8080 in the browser of the host
  - Authenticate
    - `sudo cat /var/lib/faasd/secrets/basic-auth-password | faas-cli login --username admin --password-stdin`
  - Shown in Demo
- Prometheus needs extra work
  - Create a tunnel
    `$ ssh -L 9090:127.0.0.1:9090 -p8888 localhost`
  - Access via localhost:9090 in the browser
  - Shown in demo
- Port forwarding methods for other hypervisors are different!

Sustainable Data Center

Data Centers Use a LOT of Energy!

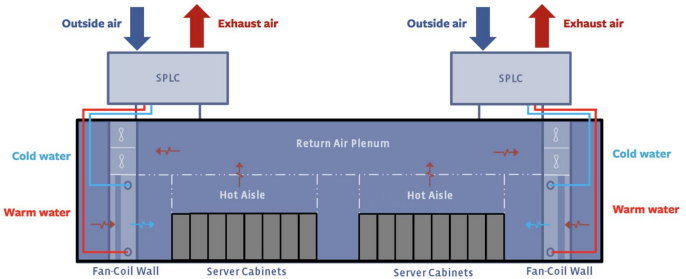
- 728 hyperscale data centers in operation globally
  - At least 5000 servers with around 10,000 square feet of physical space
  - Average data center size is 100,000 square feet
- Expect to be around 1200 hyperscale data centers by 2026
- Account for 1% of all global electricity usage

Sustainable Data Center

- Building energy efficient facilities
  - Air flow, cooling and HVAC
  - Uninterrupted Power Supply systems
- Utilizing energy conserving techniques
  - Reducing over provisioning / server consolidation
    - $500 energy per server
  - Energy efficient storage
    - Deduping
    - Selecting storage medium based on usage patterns
  - Reducing server power utilization
    - Energy saving mode
  - Error/Energy Tradeoff
- New hardware

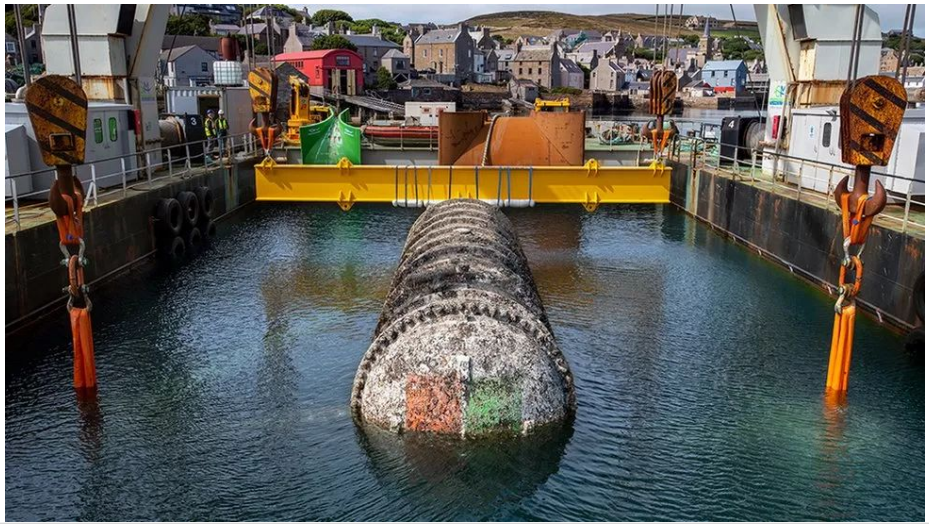
Sustainable Data Center

- Energy Efficient Facilities
  - Reducing PUE (Power Usage Effectiveness), lower the better
    - 1 means all power goes into computers only
    - 2 means for each computer, equal amount of energy goes into cooling
  - Currently at about 1.1 at Google, 1.57 elsewhere



Sustainable Data Center

- Underwater data center!
  - Lower failure rates, higher security



Sustainable Data Center

- Using Renewable Energy
  - Using Solar and other form of renewable energy
  - Many moving towards zero carbon by 2040

Energy Conserving Techniques

- Energy efficient storage

- Deduping
  - Reducing the amount of storage needed as much as 50-to-1

- Using Tapes
  - More durable and reliable than disks, and last much longer
  - A tape that has been stored correctly will last up to 30 years
  - Secure
  - Fast read, but slow writes

Energy Conserving Techniques

- Energy - accuracy tradeoff
  - Reduce accuracy in terms of energy efficiency
  - https://homes.cs.washington.edu/~luisceze/publications/Enerj-pldi2011.pdf
- Energy efficient language
  - Choice of language determines energy usage
- Energy efficient code
  - Profiling tools are available for energy usage

Table 4. Normalized global results for Energy, Time, and Memory

Total					
	Energy		Time		Mb
(c) C	1.00	(c) C	1.00	(c) Pascal	1.00
(c) Rust	1.03	(c) Rust	1.04	(c) Go	1.05
(c) C++	1.34	(c) C++	1.56	(c) C	1.17
(c) Ada	1.70	(c) Ada	1.85	(c) Fortran	1.24
(v) Java	1.98	(v) Java	1.89	(c) C++	1.34
(c) Pascal	2.14	(c) Chapel	2.14	(c) Ada	1.47
(c) Chapel	2.18	(c) Go	2.83	(c) Rust	1.54
(v) Lisp	2.27	(c) Pascal	3.02	(v) Lisp	1.92
(c) Ocaml	2.40	(c) Ocaml	3.09	(c) Haskell	2.45
(c) Fortran	2.52	(v) C#	3.14	(i) PHP	2.57
(c) Swift	2.79	(v) Lisp	3.40	(c) Swift	2.71
(c) Haskell	3.10	(c) Haskell	3.55	(i) Python	2.80
(v) C#	3.14	(c) Swift	4.20	(c) Ocaml	2.82
(c) Go	3.23	(c) Fortran	4.20	(v) C#	2.85
(i) Dart	3.83	(v) F#	6.30	(i) Hack	3.34
(v) F#	4.13	(i) JavaScript	6.52	(v) Racket	3.52
(i) JavaScript	4.45	(i) Dart	6.67	(i) Ruby	3.97
(v) Racket	7.91	(v) Racket	11.27	(c) Chapel	4.00
(i) TypeScript	21.50	(i) Hack	26.99	(v) F#	4.25
(i) Hack	24.02	(i) PHP	27.64	(i) JavaScript	4.59
(i) PHP	29.30	(v) Erlang	36.71	(i) TypeScript	4.69
(v) Erlang	42.23	(i) Jruby	43.44	(v) Java	6.01
(i) Lua	45.98	(i) TypeScript	46.20	(i) Perl	6.62
(i) Jruby	46.54	(i) Ruby	59.34	(i) Lua	6.72
(i) Ruby	69.91	(i) Perl	65.79	(v) Erlang	7.20
(i) Python	75.88	(i) Python	71.90	(i) Dart	8.64
(i) Perl	79.58	(i) Lua	82.91	(i) Jruby	19.84

Midterm Review

Midterm Overview

- 100 Minutes
- 50~60 questions
  - T/F
  - Multiple Choice
  - Matching
  - Multiple Answer
- Covers materials from Lecture 1 to Lecture 10

Midterm Concepts Covered

- Cloud Computing Models
  - IaaS, PaaS, SaaS
  - Public, Private, Hybrid Cloud
- Cloud Computing Definitions & Benefits
- Pros & Cons of Cloud Computing

Midterm Concepts Covered

- Virtualization
  - Emulation vs Virtualization vs Simulation
- System Virtualization
  - Types of Hypervisor
  - Xen Architecture
  - Need for System Virtualization
  - Paravirtualization
  - Vagrant
  - Memory Virtualization

Midterm Concepts Covered

- OS Virtualization
  - Container Technologies and Architectures. i.e., Docker, Kata architecture
    - AUFS
    - DockerFile
  - What enables OS Virtualizations
  - Namespaces, CGroups
  - Chroot
  - Container definition
- Orchestration
  - Kubernetes architecture and terms
  - Minikube
  - Infrastructure as Code

Midterm Concepts Covered

- Function as a Service
  - Definition
  - Pros and Cons
  - FaaS Architecture
    - Gateway
    - Where and How do Functions Run?
  - OpenFaaS

Example Questions

- Kata containers has more layers of abstraction than Docker containers
  - True or False?
- Cgroup is used to define parameters about the resource use of a set of processes
  - True or False?

Example Questions

- Match the correct virtualization techniques with implementations
  - System Virtualization
  - OS Virtualization
  - Application Virtualization

  - CoreOS
  - VirtualBox
  - JVM

Example Questions

- Select an Answer

- What is the definition of orchestration?
  a. Completing a single task or function without human intervention
  b. Automatic management of computer systems
  c. Act of provisioning and configuring cloud resources
  d. Starting Kubernetes

TODOs!

- HW 2
- Midterm!
- Get prepared for networking!

Agenda for Today

- OpenFaaS Demo
- Midterm Preview
- Sustainable Data Center
- Readings
  - Recommended: None
  - Optional: https://www.cncf.io/blog/2020/04/13/serverless-open-source-frameworks-openfaas-knative-more/

Questions?