Serverless Computing
Overview

Yiying Zhang
Outline

• Overview and major offerings
• Example applications
• Limitations and discussion

Acknowledgement: Some of the slides are from Ali Ghodsi and Ion Stoica’s Berkeley lecture notes in 2018 and Tyler Harter’s HotCloud’16 OpenLambda talk
What is Serverless Computing?

- Computing without servers?
- Running applications without the need to manage servers?
- Running functions instead of containers/VMs?
- Infinite scaling?

- The truth: no clear, agreed definition, i.e., no one really knows
One Perspective: How Cloud and Virtualization Evolved
Classic Web Stack

RPCs

Application
Server
OS
Hardware

weak virtualization
1st Generation: Virtual Machines

RPCs

Application

Server

OS

Hardware

virtual H/W
2nd Generation: Containers

RPCs

Application

Server

OS

Application

Server

Hardware

virtual OS
3rd Generation: Serverless Computing

RPCs

Server and Runtime

Application

Application

OS

Hardware

virtual servers
A Related Topic: Microservice

- A software architecture that develops an application as a suite of small services, each of which can be deployed and scaled independently
- When one (micro)service is in large demand, can scale it up
- Different (micro)services can be written and managed by different teams
- Changing one (micro)service will not affect the others
A Related Topic: Microservice

A monolithic application puts all its functionality into a single process…

… and scales by replicating the monolith on multiple servers

A microservices architecture puts each element of functionality into a separate service…

… and scales by distributing these services across servers, replicating as needed.
Serverless means ...

- No server or container management
- High availability
- Flexible scaling
- No idle capacity
What is the essence of “Serverless Computing”? Or, what do people really like about it?

• Management-free

• No need to handle creation, failure, replication, etc.

• Autoscaling

• Spin up/down functions quickly based on load

• Only pay for what you use

“...I didn’t have to worry about building a platform and the concept of a server, capacity planning and all that “yak shaving” was far from my mind... However, these changes are not really the exciting parts. The killer, the gotcha is the billing by the function... This is like manna from heaven for someone trying to build a business. Certainly I have the investment in developing the code but with application being a variable operational cost then I can make a money printing machine which grows with users...”

source: https://hackernoon.com/why-the-fuss-about-serverless-4370b1596da0
What is Today’s Serverless Computing Like?

• Largely offered as Function as a Service (FaaS)

• Cloud users write functions and ship them

• Cloud provider runs and manages them

• Still runs on servers

• Have attractive features but also many limitations (more later this lecture)
First serverless app: BigQuery

Fully managed Data Warehouse
- “Arbitrarily” large data and queries
- Pay per bytes being processed
- No concept of cluster

Other similar systems
- AWS Athena
- Snowflake
- ...
AWS Lambda

- An event-driven, serverless computing FaaS platform introduced in 2014
- Functions can be written in Node.js, Python, Java, Go, Ruby, C#, PowerShell
- Each function allowed to take 128MB - 3GB memory and up to 15min
- Max 1000 concurrent functions
- Connected with many other AWS services
Lambda Function Triggering and Billing Model

• Run user handlers in response to events
  • web requests (RPC handlers)
  • database updates (triggers)
  • scheduled events (cron jobs)

• Pay per function invocation
  • No charge when no functions run (no triggering event)
  • Billed by duration of function, configured memory size, and # of functions
    • charge \textit{actual\_time} * \textit{memory\_cap}
Serverless Applications

Event source → Lambda function → Services (anything)

Changes in data state
Requests to endpoints
Changes in resource state

Node.js
Python
Java
C# (.NET Core & Core 2.0)
Go
Ruby
Powershell
BYR – Bring your own Runtime
Internal Execution Model

• Developers upload function code to a *handler store* (and associate it with a URL)

• Events trigger functions through RPC (to the URL)

• Load balancers handle RPC requests by starting *handlers* on *workers*
  
  • Calls to the same function are typically sent to the same worker(s)

• Handlers sandboxed in containers
  
  • AWS Lambda reuses the same container to execute multiple handlers when possible
Execution Model

Load Balancer

Load Balancer

user

RPC

handler store

H1 H2

workers

H2 sandbox

Python

Server

Python

Server

...
Execution Model

load balancers

user

RPC

Load Balancer

Load Balancer

handler store

H1

H2

workers

H2

sandbox

Python

Server

Python

Server

...
Azure Functions

• Debuted in 2016

• Support Python, C#, Java, JavaScript and PowerShell

• Three plans:
  • Consumption, Premium, and Dedicated
  • Dedicted: functions run on dedicated VMs and scaled manually (unlimited duration)
  • Max duration for consumption and premium: 10min and 60min
  • Max memory: 1.5GB - 14GB (depending on plans)
Google Cloud Functions

- Support Node.js, Python, and Go
- Memory size 128MB - 2GB
- Max function duration 9min
- Max number of functions per project: 1000
- Bill CPU and memory separately
Open-Source Projects

• OpenWhisk
  Originally developed by IBM and the core technology behind IBM Cloud Functions

• OpenLambda
  Research prototype developed at University of Wisconsin, written in Go and based on Linux containers
• Difficult and slow to manage states
  • Have to use (slow) cloud storage!
• No easy or fast way to communicate across functions
  • Have to go through cloud storage or other services
• Functions can only use limited resources
  • e.g., starting functions on “cold” machines can be slow
• Billing model does not fit all needs
Cold Start Time of Different Languages/Offerings

source: https://mikhail.io/serverless/coldstarts/big3/
Cold Start Time with Different Package Sizes

Comparison of cold start durations per deployment size (zipped)

source: https://mikhail.io/serverless/coldstarts/big3/
Good and Bad Use Cases

• Some good ones:
  • Parallel, independent, stateless tasks
  • Orchestrating functions
  • Function composition

• Some bad ones:
  • Stateful applications
  • Distributed applications and protocols
  • Applications that demand more resources (esp. memory)