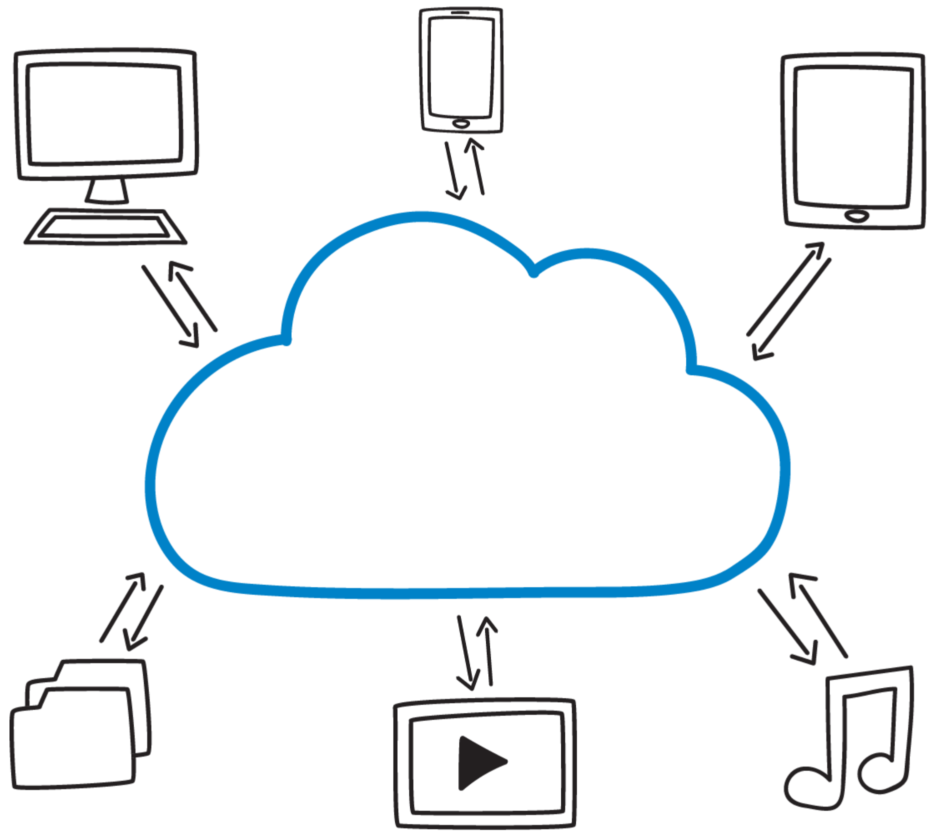
Serverless/FaaS

One day intensive class

This is a lab heavy/intensive course



logistics

- Class Hours:
- Start time is 8:30am
- End time is 4:30pm
- Class times may vary slightly for specific classes
- Breaks mid-morning and afternoon (20 minutes)

- Telecommunication:
- Turn off or set electronic devices to vibrate
- Reading or attending to devices can be distracting to other students

- Lunch:
- Lunch is 11:45am to 1:15pm
- Yes, 1 hour and 30 minutes
- Extra time for email, phone calls, or simply a walk.

- Miscellaneous
- Courseware
- Bathroom

Course Objectives

By the end of the course you will be able to:

- State the function and purpose of Serverless/FaaS
- Create a AWS Lambda function
- Connect your Lambda function to other AWS services
- Connect your Lambda function to an API Gateway
- Describe the benefits and trade offs of using FaaS

This is a lab heavy/intensive course

Agenda

- Welcome and Introductions
- Introduction to Serverless/FaaS
- Introduction to multiple FaaS providers
- Benefits and limitations of FaaS architecture
- Creating your first Lambda function
- Connect your Lambda function to an API gateway
- Connect your Lambda function to other AWS services
- Wrap-up

Jordan Rinke

Twitter: @jordanrinke

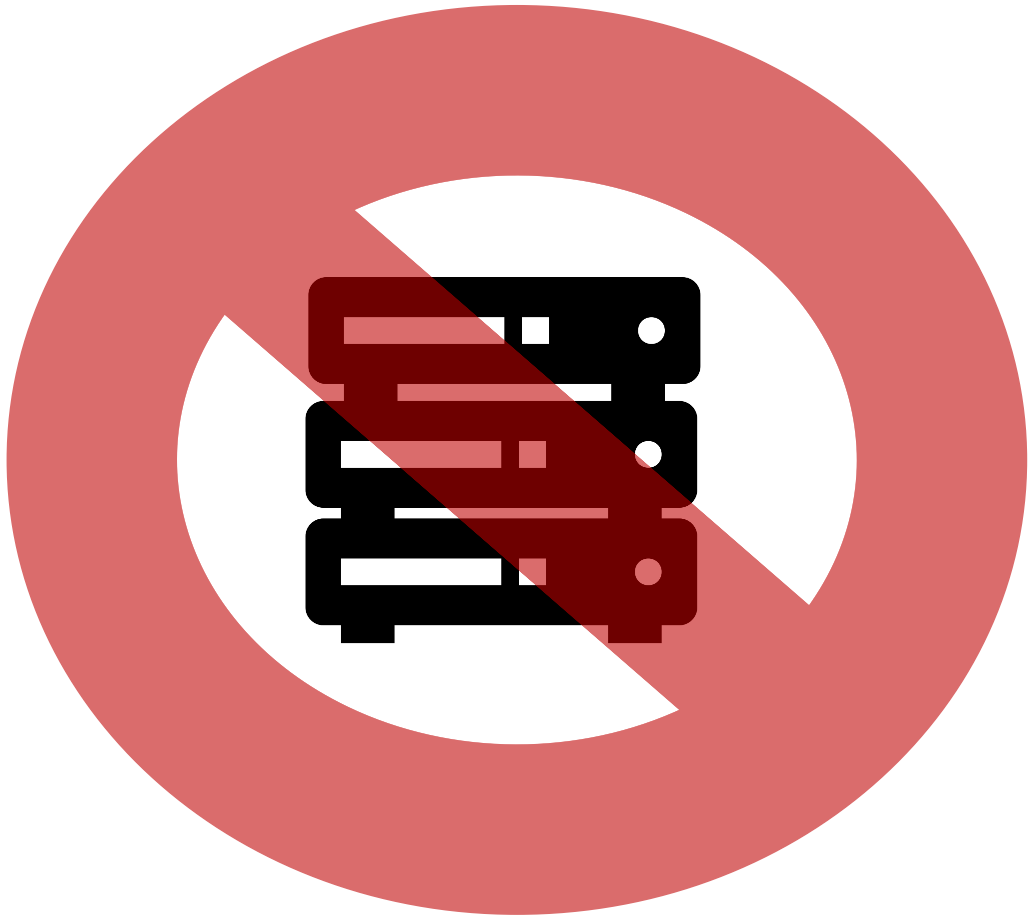
jordan@rin.ke

Expertise

- Cloud
- AWS/Azure/Google
- OpenStack
- CICD/Automation
  - Ansible/Chef/Puppet
  - Terraform/Jenkins
- Containers
  - Docker/Kubernetes
  - Microservices

Introductions

- Name
- Title, or Role if your title doesn't describe what you do
- Which statement best describes your Serverless/FaaS experience?
  a. I am ***currently working*** with Serverless on a project/initiative
  b. I ***expect to work*** with Serverless on a project/initiative in the future
  c. I am ***here to learn*** about Serverless outside of any specific work related project/initiative
- Expectations for course (why am I here & what do I want to learn)



What is serverless/FaaS?

Serverless = FaaS (Functions as a Service)

Traditional VM	**Containers**	**Serverless**
Function	Function	Function
Application	Application	Application
Container	Container	Container
Operating System	Operating System	Operating System
Virtual Hardware	Virtual Hardware	Virtual Hardware

How do you run just a function?

Your function

```
def my_handler(event, context):
        message = 'Hello {} {}!'.format(event['first_name'],
        event['last_name'])
        return {
                'message' : message
        }
```

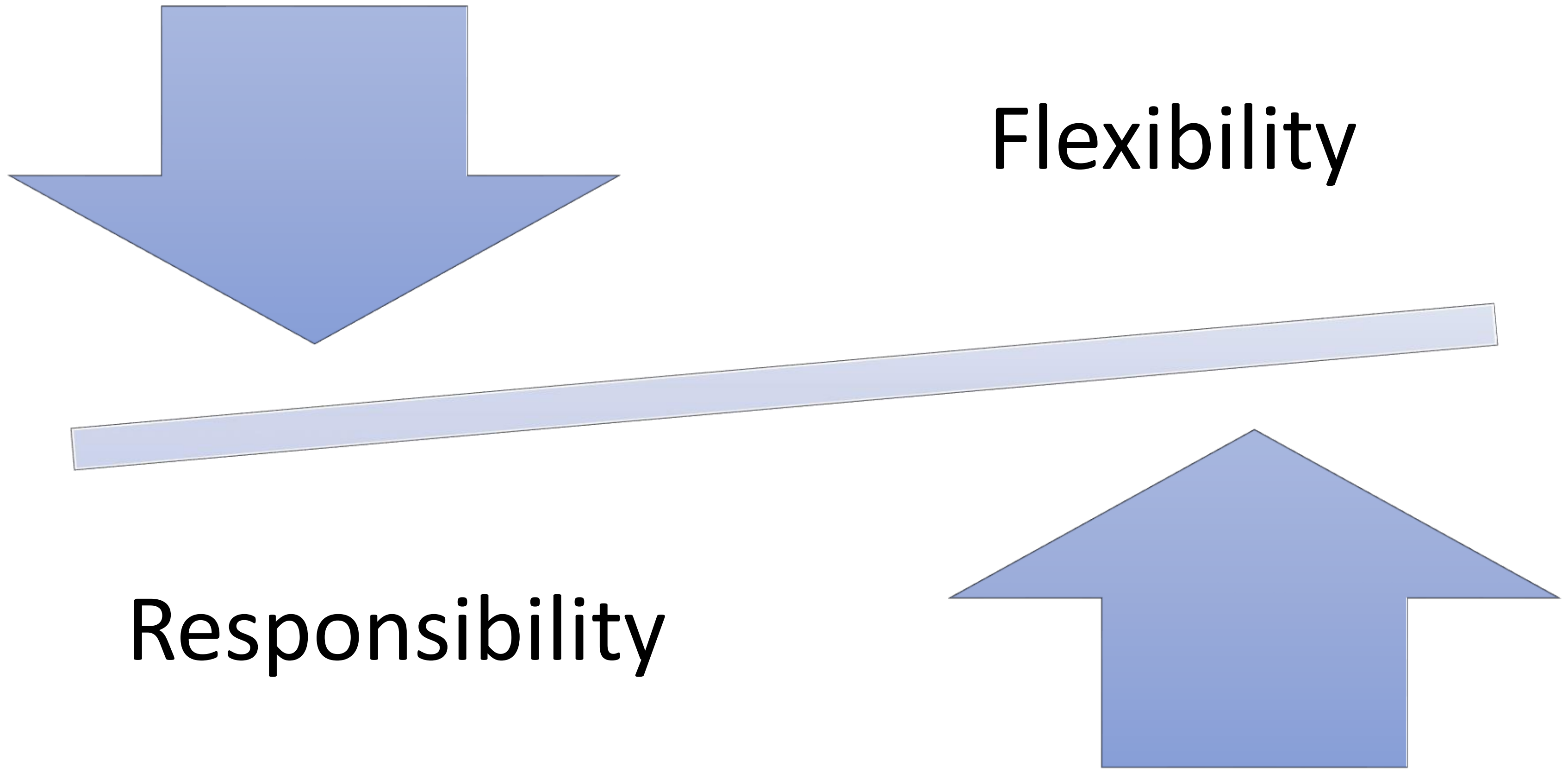
Container

- Your code is encapsulated in a prebuilt container from the provider that contains a dispatch agent. An ultra light HTTP endpoint that accepts requests, and executes your snippet of code.

Serverless

- When a request comes in, an API gateway looks for a container running your function, if none exist one is created and the request is routed.

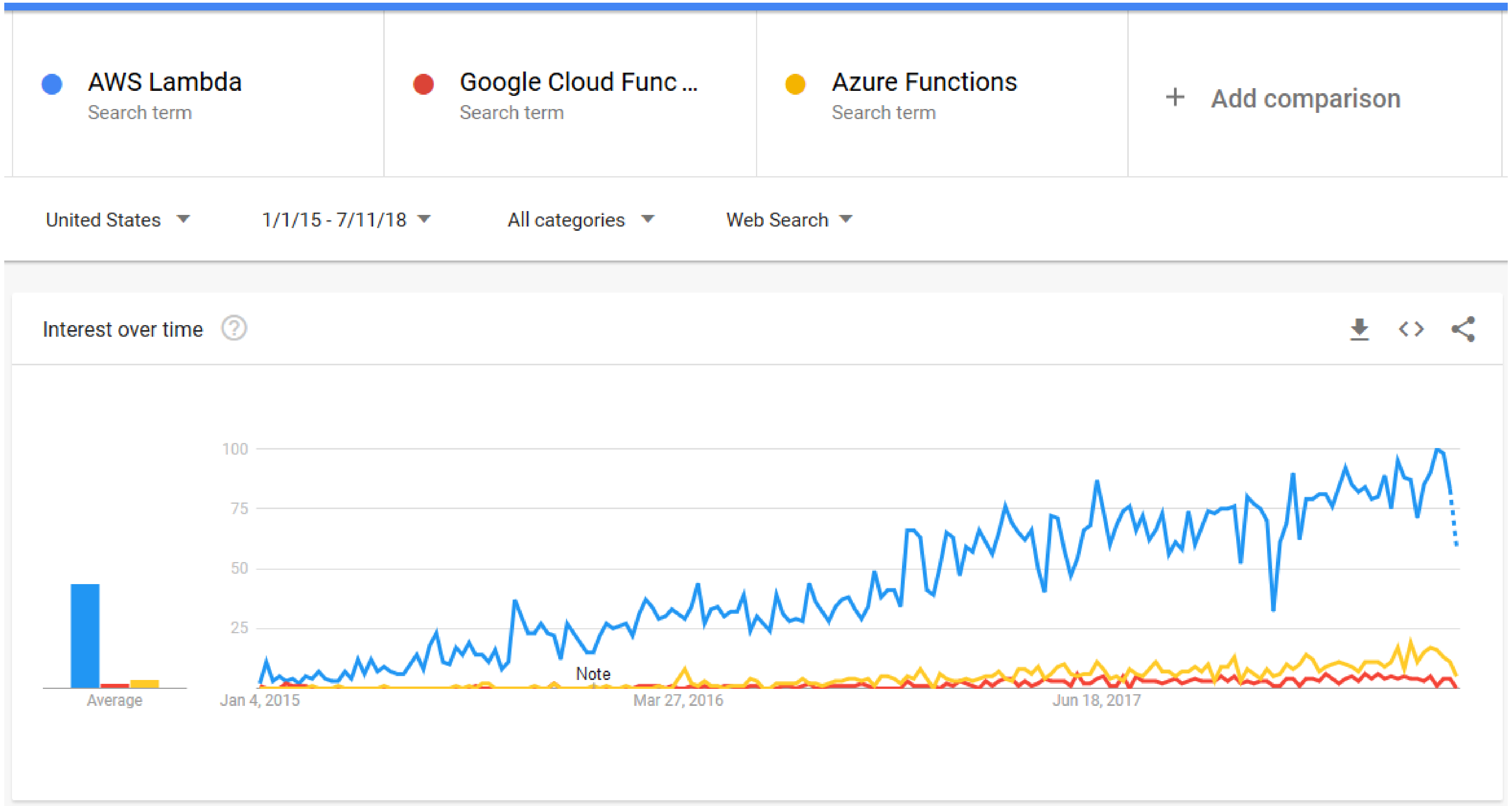
Serverless is just containers?



Where did it come from?

- AWS announced Lambda for technical preview Nov, 2014.
- Lambda was released for production April, 2015.
- Google Cloud announced a Lambda competitor named Cloud Functions April, 2016
- Azure announced a Lambda competitor named Functions Nov, 2016
- Initial OpenFaaS commits Dec, 2016
- Pivotal Cloud Foundry released PCF Functions in 2018
- Apache OpenWhisk 2016

Adoption/Interest



● AWS Lambda
Search term

● Google Cloud Func ...
Search term

● Azure Functions
Search term

+ Add comparison

United States ▾ 1/1/15 - 7/11/18 ▾ All categories ▾ Web Search ▾

Interest over time

Providers – Framework-as-a-Service

- AWS - Lambda
- Microsoft - Azure Functions
- Google – Cloud Functions
- CloudFlare
- OpenFaaS/Fn/Kubeless/Fission/Kubernetes (Self Hosted)
- Apache (IBM) - OpenWhisk

AWS Lambda

- Language support:
  - Node.js (JavaScript 4.3/6.10/8.10)
  - Python (2.7/3.6)
  - Java (Java 8 compatible)
  - C#/Powershell (1.0/2.0/2.1 .NET Core)
  - Go
- Has triggers for all major AWS services, such as running a Lambda function on DynamoDB change.
- No custom containers.

Azure Functions

- Language support:
  - C#
  - JavaScript
  - F#
  - Python
  - Batch
  - PHP
  - PowerShell
- Supports uploading custom containers to support any language.

Google Cloud Functions

- Language support:
  - Node.js javascript V6.10 and 8.10
  - Python 3.7.0
- No custom containers, runtime containers based on Ubuntu 18.0.4 for GCF execution environment

CloudFlare Workers

- Language support:
  - Javascript
- Specifically designed to run on CloudFlare CDN edge servers to improve page responsive logic.

OpenFaaS

- Language support:
  - All major languages are supported.
- Premade containers are available for most major languages
- Building custom containers is a common approach
- Containers are bootstrapped with a small Go HTTP service for dispatching to functions
- Self Hosted, Kubernetes native

Lab 1: Building your first Lambda function

- Log in to the AWS console, using the control panel create and test a hello world Lambda function
- Full lab details are found at https://github.com/scalable-af/labs/tree/master/serverless/aws

Questions

Too easy?

- That was too easy, why isn’t everything using this?
    - Design limitations
    - Speed
    - Cost

Design Limitations

- All functions are completely stateless.
- Functions may take many seconds to start.
- Functions have a limited duration run time.
- Functions can get expensive very quickly.

From here forward we will be focusing on Lambda specifically, different platforms have different but similar concerns – we will discuss Azure in the second half of the class

Stateless

- No data is maintained between function calls.
- All data must be consumed at function instantiation, and returned or sent to another location.
- Configuration can be passed in via Context and Environment Variables

Cold Starts vs Warm Starts

- Containers can take many seconds to start their first time – a “cold” start.
- Once a container is running subsequent requests are very fast.
  - However – containers are killed after roughly 30 minutes of no activity.
- VMs running containers are recycled ever 4 hours. You will experience cold starts at least every 4 hours.

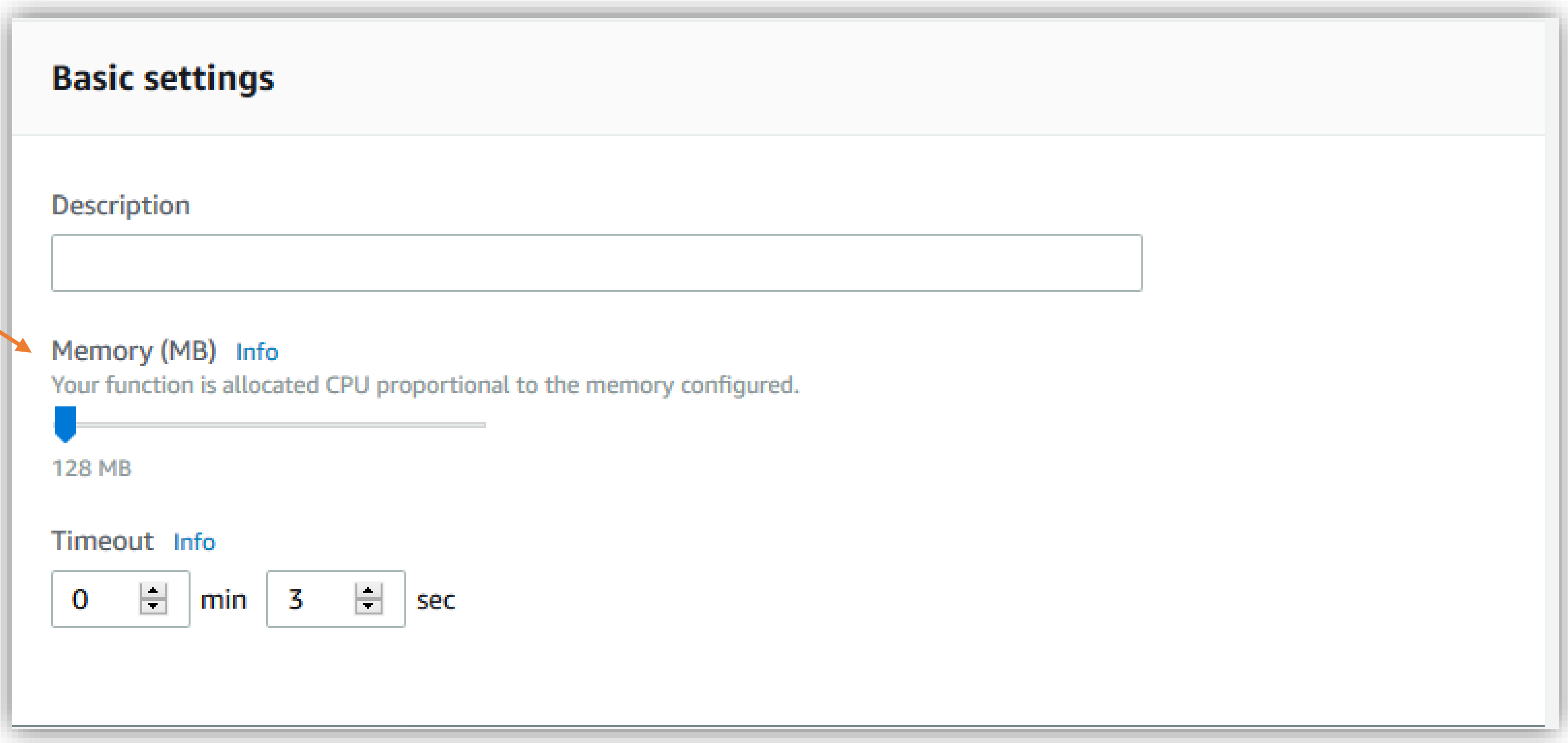
Cold start optimization

- The language you use dramatically impacts your cold start time
- Your configuration impacts your start time
    - Using a Lambda function in a VPC could lead to cold start times in the 10s of seconds range because it has to be attached to the private network
- Functions with more configured memory start faster.

Average cold start times

Language	128MB Mean Time(ms)	256MB Mean Time(ms)	512MB Mean Time(ms)	1024MB Mean Time(ms)	1536MB Mean Time(ms)
C#	4387	2234	1223	524	407
Java	3562	1979	999	539	339
Node	12	8	3	2	2
Python	1	0.8	0.4	0.4	0.4

Function sizing



Basic settings

Description

Memory (MB) Info

Your function is allocated CPU proportional to the memory configured.

128 MB

Timeout Info

0 min 3 sec

Over 1536MB the function gets access to a second vCPU

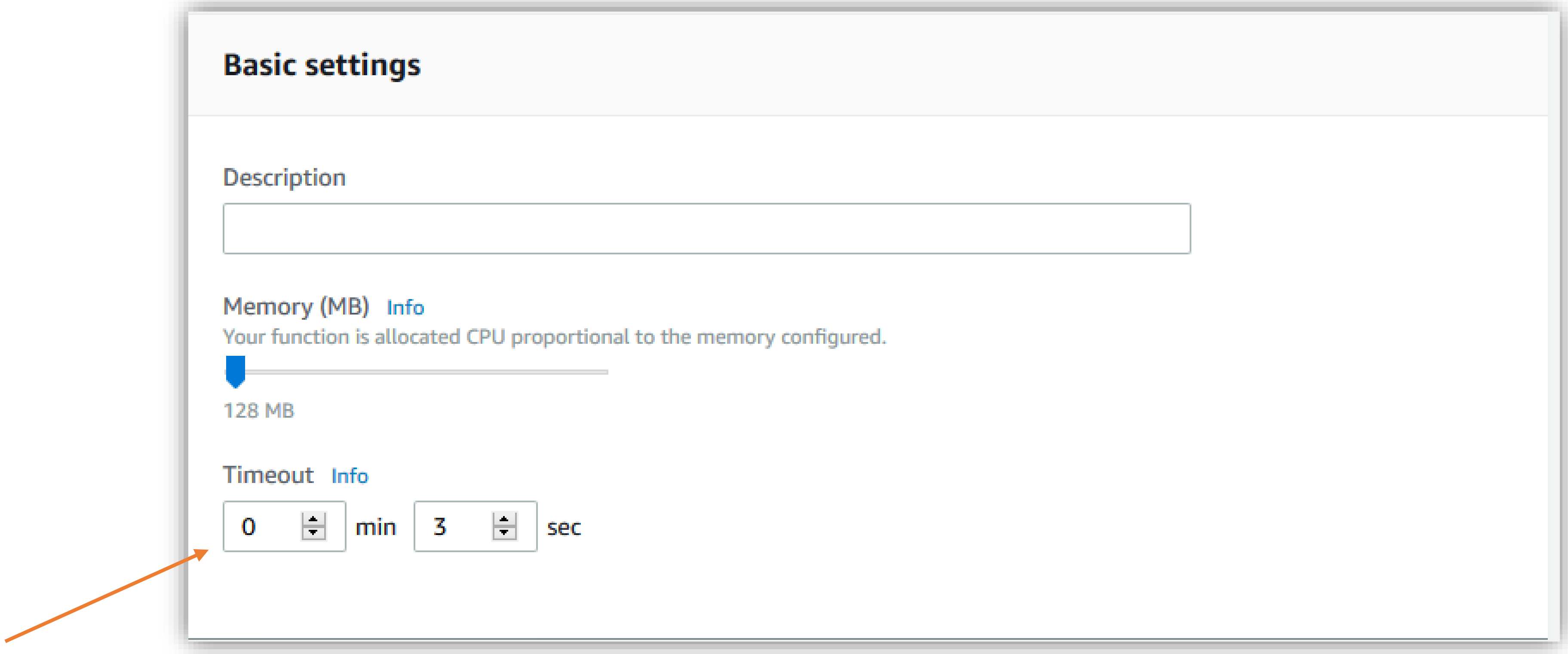
Proportional CPU

- Functions are given CPU shares based on their memory size.
- After 1536MB functions receive a second vCPU.
- Be sure to test function sizing, giving a function more memory may not increase speed over 1536MB if your function isn’t capable of running across multiple cores.
- It may be cheaper and be more performant to run a second instance with less memory, or break it into smaller functions

Preventing cold starts

- Use a Lambda Step functions with a Task Timer to forever call itself every 5 minutes to warm your function.
- Use a CloudWatch event timer to call a function on a regular schedule to warm your function.
- Build in short circuit paths to prevent wasting cycles processing a warming call.
- Run calls in parallel to fit your concurrency requirements otherwise users will still experience cold starts over a certain load.

Timeouts



Basic settings

Description

Memory (MB) Info

Your function is allocated CPU proportional to the memory configured.

128 MB

Timeout Info

0 min 3 sec

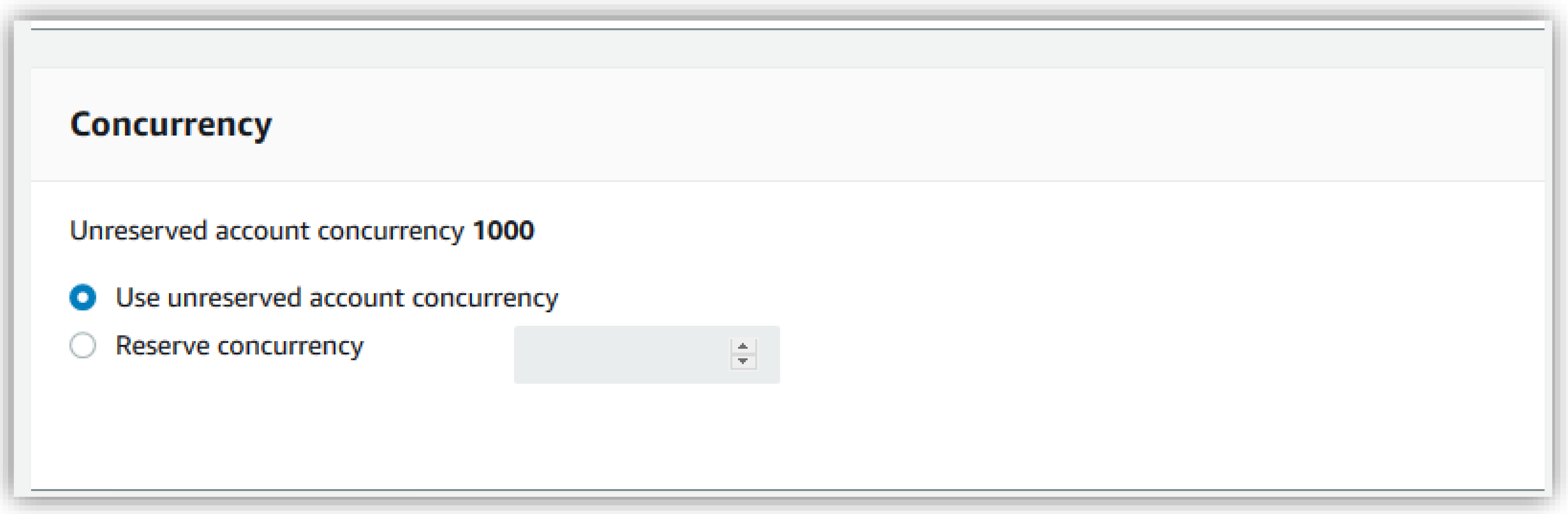
- The default function timeout is 3 seconds, this can be adjusted up to 300 seconds.

Billing

- Functions are billed in 100ms increments and are always rounded up.
- You are still billed if your function crashes or is terminated.
    - If you exceed your memory your function will be terminated, you will still be billed for the time up to the function crashing.
    - A crashing function with a calling application that retries can crash very very fast, and bill for 100ms every single time.

Questions

Concurrency and Scaling



Concurrency

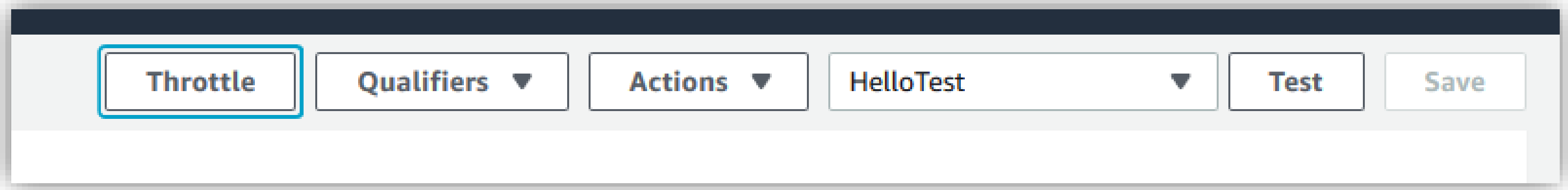
Unreserved account concurrency **1000**

[✓ Use unreserved account concurrency]

Reserve concurrency

- Account concurrency can be increased via a support ticket
- Reserved concurrency, reserves a portion of your available 1000 for this specific function.
  - This prevents one function, say an inbound function from using all of your capacity and starving the back end pipeline.

Throttling



- Clicking the throttle button will instantly turn your reservation to 0, in case of emergencies.
- Your function will also be throttled if you are using all of your concurrent executions (1000 by default).
- Throttle events are recorded in CloudWatch as throttle events, alarms can be configured for them.

Service Triggers

- Most AWS services have built in streams and triggers. They can be configured directly from the Lambda portal.

Lab 2: Connect your Lambda function to DynamoDB

- Create a DynamoDB table
- Configure Streams
- Attach the stream to your Lambda function
- Full lab details are found at https://github.com/scalable-af/labs/tree/master/serverless/aws

Questions

Logging

- By default all Lambda functions create a log stream in CloudWatch that log their execution time, and billed time.
- All built in logging packages work. Console.log() is all that is needed to output data to CloudWatch

API Gateway

- To access Lambda services externally you must configure an API Gateway.
- Lambda services must respond with JSON, and a valid status code
- API Gateway has a non configurable timeout limit of 30 seconds.

Lab 3: Connect your Lambda function to an API Gateway

- Create an API Gateway
- Configure and access your function
- Full lab details are found at https://github.com/scalable-af/labs/tree/master/serverless/aws

Questions

Canary Deployments

- Canary deployments are a pattern in which you can deploy new versions while limiting user impact.
- A new version is deployed and traffic is configured to go to a new version, over time that percentage is adjusted until the service is fully migrated.

Lab 4: Create a Canary deployment

- Create two versions of your Lambda function
- Configure the API Gateway to use a version of the function with 50/50 traffic splitting
- Full lab details are found at https://github.com/scalable-af/labs/tree/master/serverless/aws

Questions

Serverless Use Cases

Message bus/queue processing

- A common mobile pattern is using latency and cache tolerant message bus protocols instead of HTTP
- Serverless functions can watch a message queue and process events as they are received.
- Application pattern must be asynchronous with long polling or push. Persistent connections are not capable with serverless functions. *More on this later*

Web Backend

- The only standard supported protocol for serverless functions is HTTP, generally implemented as a RESTful API.
- React/Angular/Ember etc. can use routed serverless functions to implement all dynamic functionality.
- Static files must be hosted outside of serverless functions (A common pattern is to host static files in s3 buckets or another CDN solution.)
- This is specifically what CloudFlare is attempting to target, static hosted files at the CDN and light weight routing login serverless functions for better user experience.

Stream Processing

- AWS and Azure support connecting a serverless function to any stream such as kinesis or Azure event hub.
- Functions can scale out up to your concurrency limit to handle dynamic stream processing volume.
- Multiple functions can be chained together for proper stream processing patterns.
- Remember that billing is in 100ms increments, if your functions are averaging on the low side, combine them to eliminate your billing waste from rounding up.

File Processing

- All platforms with object storage have the ability to configure a serverless function to be triggered when a new object (file) is created.
- A common example is creating thumbnails from uploaded images, or compressing generated images before the are distributed to CDN endpoints.
- Another example is processing CSV files uploaded by users for insertion into a database.

Scheduled Tasks

- Use CloudWatch Events or Azure function timers to trigger serverless functions on a regular schedule.
- Examples are triggering database backups, or consolidating data. For instance averaging database values over the event interval into another database to have time reduced averages for long term business analytics.
- These can effectively take the place of a cron job in traditional infrastructure.

DevOps / Pipeline

- Serverless functions can be used to trigger build pipelines.
- Many ChatOps bots use serverless functions to process incoming chat events and to emit alerts to chat platforms.
- Serverless functions can trigger builds and deployments, a chain of serverless functions to built, test, and deploy traditional applications outside of the platform in question.

Alexa Skills

- Serverless functions are the most common backend for Alexa skills or any interactive user triggered event that does not require a visual user interface.

Questions

Serverless Frameworks

Serverless – The original FaaS framework.

- Supports Azure, GCP, AWS, IBM, Kubernetes
- Includes an Event Gateway abstraction for API routing and processing.
- Is provider agnostic – anything written with the serverless framework will work on any supported platform with minimal or no modifications.
- It is complex to implement and requires knowing the serverless framework specifically but abstracts and handles the native implementations at the platform level.
- Some elements such as the Event Gateway are not truly serverless.

AWS SAM

- An extension of CloudFormation to reduce the amount of boilerplate templating required to compose Lambda based applications.
- Allows you to run a local instance of the API Gateway, or deploy to a live API Gateway instance.
- Capable of running unit tests on lamda functions locally with a micro AWS environment specifically for running Lambda functions.
- AWS specific, designed for Lambda functions only. Well worth looking at if you are certain you will only be using AWS. The integration is very deep but extensive.

Questions

The Complete Serverless Web App

The Front End

- Static file hosting done on a CDN or object storage platform.
- Dynamic javascript based application, typically a single page application (SPA).
- Common frameworks are React, Angular, Ember, Backbone.
- Static files are compressed and loaded on the web edge as near to the end user as possible.
- All RESTful transactions are directed to serverless functions.

The Back End

- CRUD operations are received via HTTP and processed in real time, typically synchronously.
- Websockets are not supported as functions are designed to be short lived.
- FanOut and AWS IoT endpoints can enable websocket by providing a stateful layer that calls out to lambda functions while maintaining the socket connection.
- Client side libraries that implement message queues that don’t rely on sockets can be used to implement asynchronous functionality.

The Data

- Serverless functions interact with hosted data storage via key value stores or cloud hosted RDBMS.
- Replication between zones or geography are handled at the data layer enabling serverless applications to run in any zone where the data is replicated and accessible with no additional work.

HA/DR

- Serverless functions have no personality, using the API gateway or DNS GSLB with ECV one can host a serverless application in multiple regions, zones, or even providers with no additional infrastructure work.
- If the serverless functions exist in regions where hosted data solutions provide automated replication one can have geographically distributed real time failover with no infrastructure design or architecture investment.

Questions

What About Azure?

Azure functions mostly are the same as Lambda

- The specifics are different but you can generally accomplish the same things in Azure that you can accomplish with Lambda with functionally the same limitations.
- There are a number of key differences in terminology and minor implementation details.
- We will do the 4 previous AWS Lambda labs in Azure to demonstrate the specific differences.

Lab 1: Building your first Azure function

- Log in to the Azure portal, using the control panel create and test a hello world Azure function
- Full lab details are found at https://github.com/scalable-af/labs/tree/master/serverless/azure

Questions

Lab 2: Connect your Azure function to Cosmos DB

- Create a Cosmos DB
- Configure Triggers
- Attach the trigger to your Azure function
- Full lab details are found at https://github.com/scalable-af/labs/tree/master/serverless/azure

Questions

Lab 3: Create a Canary deployment

- Create two versions of your Azure function
- Configure the Azure API Management to use a version of the function with 50/50 traffic splitting using a random integer evaluation for “choose” context conditional routing. It isn’t nearly as easy as Lambda but it is … possible.
- Full lab details are found at https://github.com/scalable-af/labs/tree/master/serverless/azure

Questions

Q&A / Have a nice day

- Any questions?
- Any lab issues?