Serverless/FaaS

One day intensive class

This is a lab heavy/intensive course

logistics

- Class Hours:
- Start time is 9:15am
- End time is 3:30pm
- Class times may vary slightly for specific classes
- Breaks mid-morning and afternoon (20 minutes)

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

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

- 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

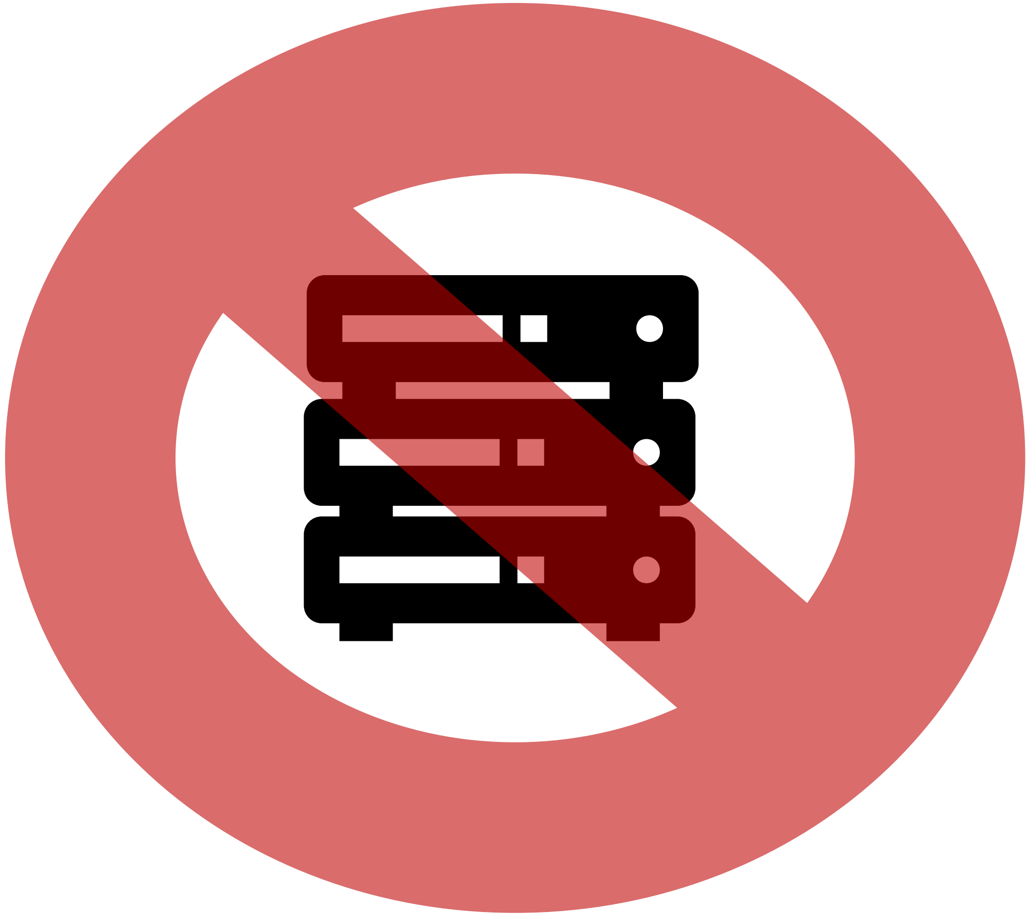
jordan@rin.ke

Expertise

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

Introductions

- Name
- Job Role
- 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 (please be specific)

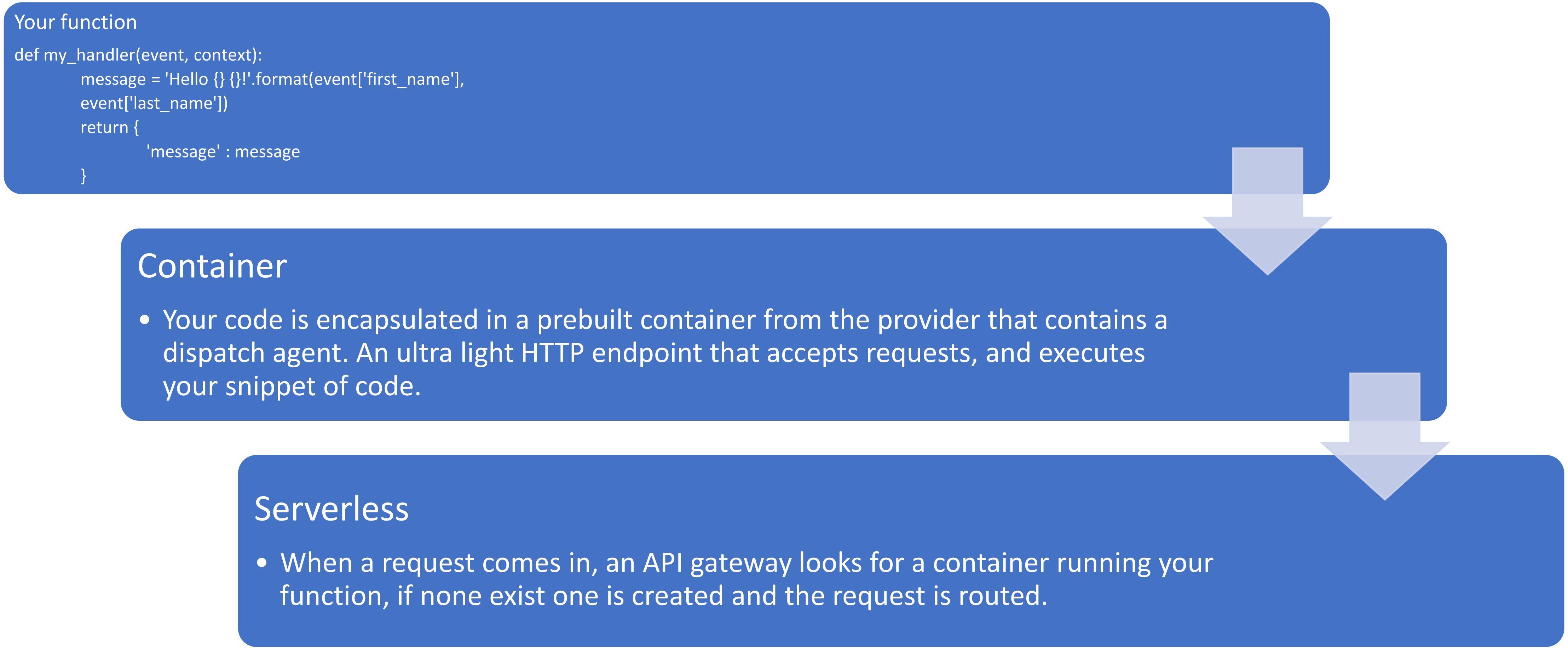


What is serverless/FaaS?

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
        }
```

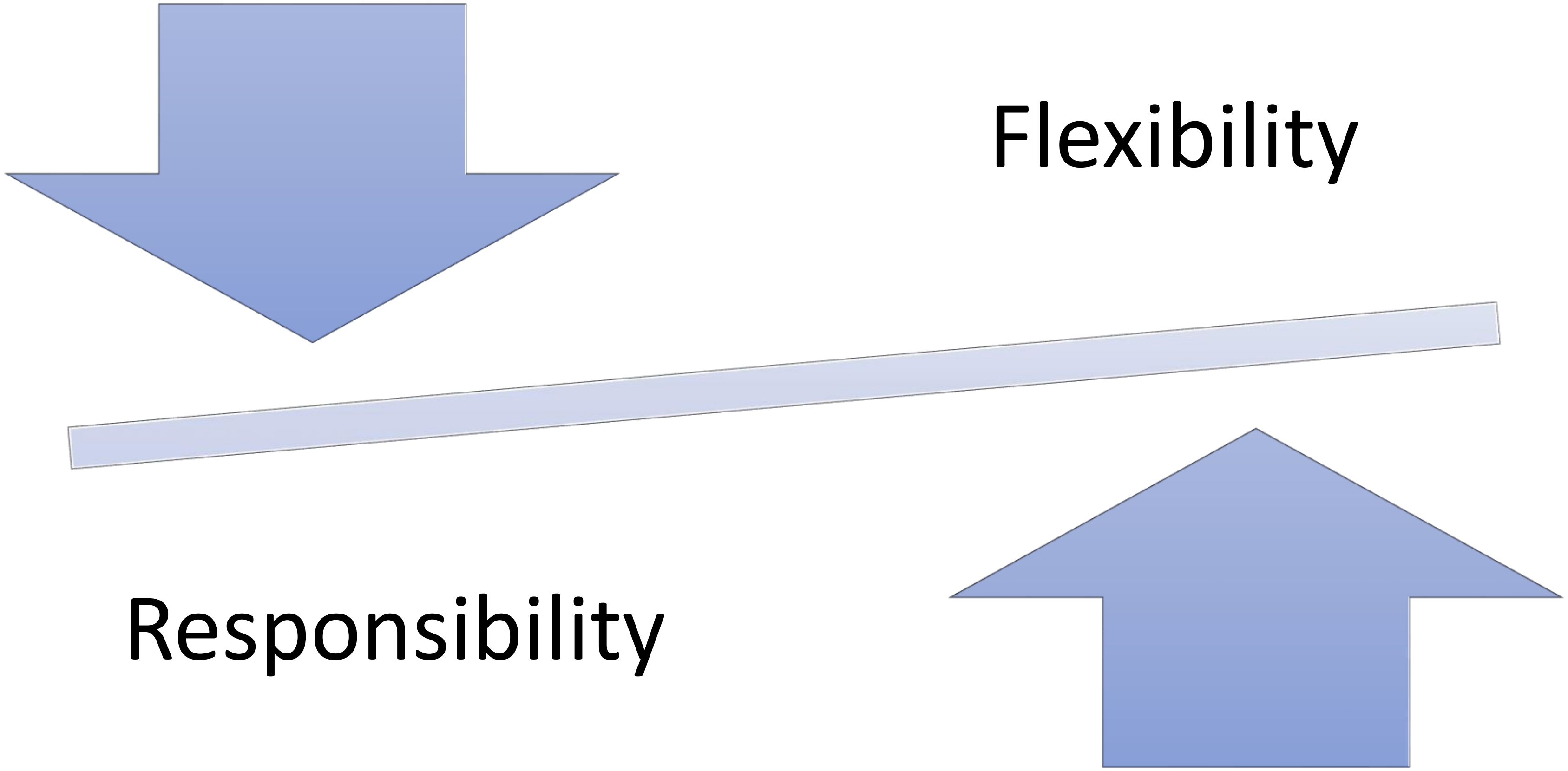
Container

- 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?



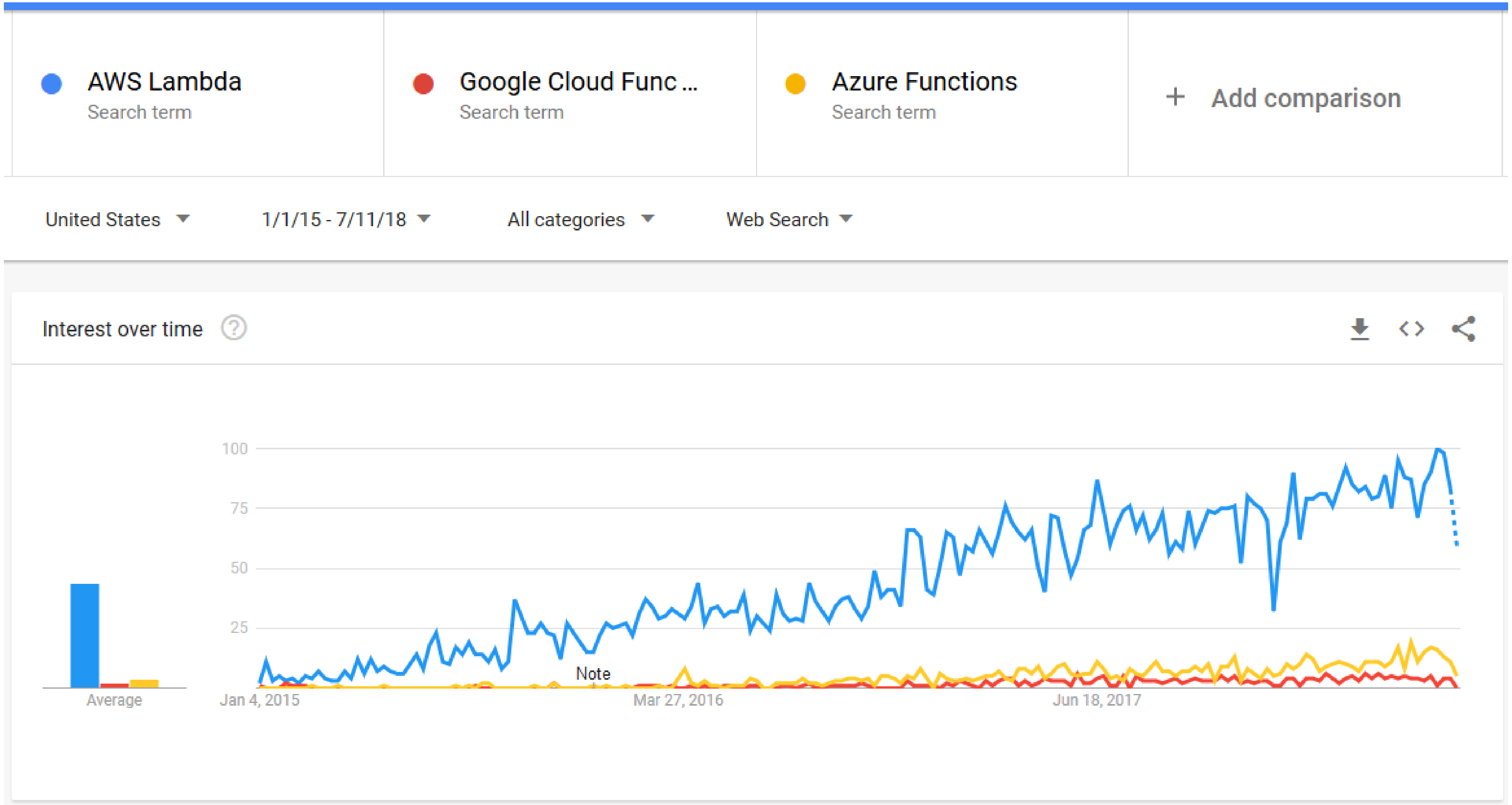
Flexibility

Responsibility

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

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

- AWS
- Azure
- Google Cloud
- CloudFlare
- OpenFaaS/Kubernetes (Self Hosted)

AWS Lambda

- Language support:
  - Node.js (JavaScript)
  - Python
  - Java (Java 8 compatible
  - C# (.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:
  - Javascript
- No custom containers

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/serverless

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

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.

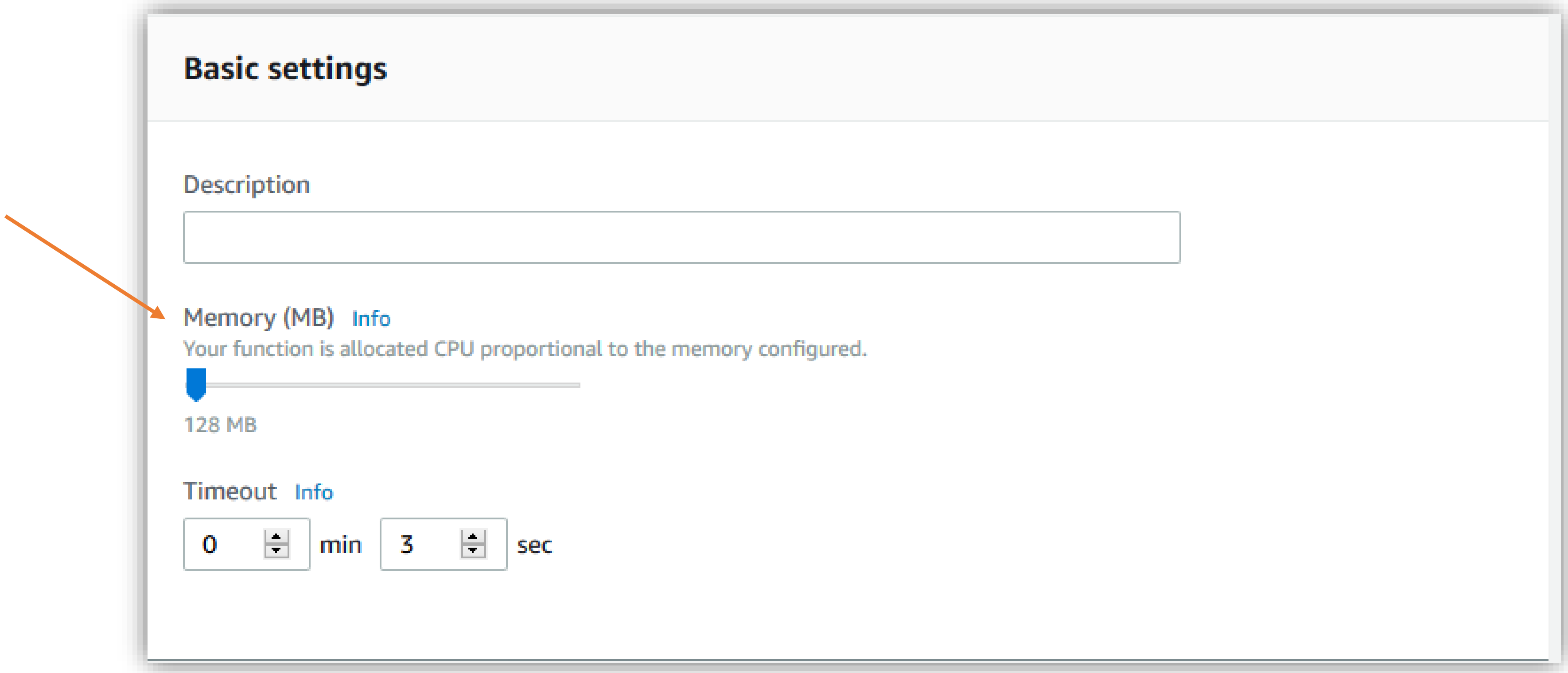
Cold start optimization

- 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

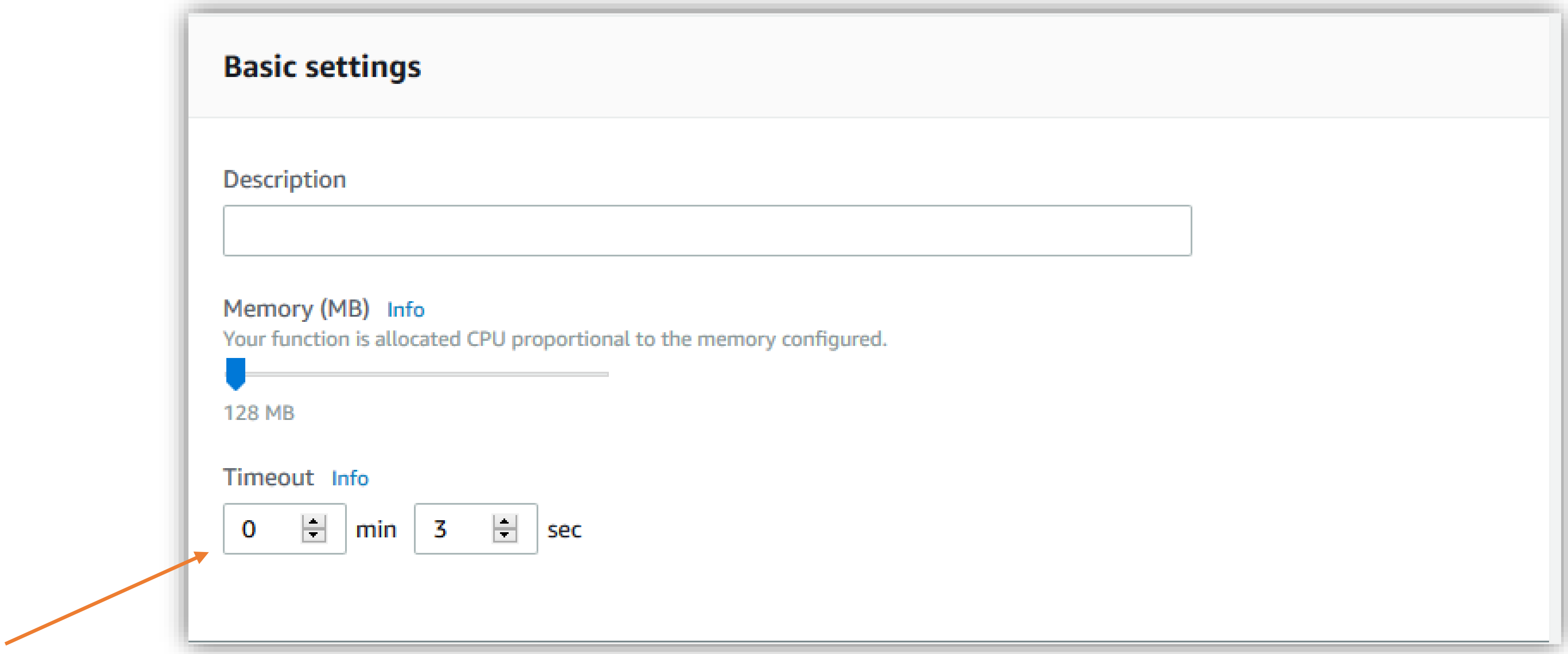
Proportional CPU

- 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

- You can use Lambda Step functions with a Task Timer to forever call itself ever 5 minutes to warm your function.
- Build in short circuit paths to prevent wasting cycles processing a warming call.
- You will have to 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

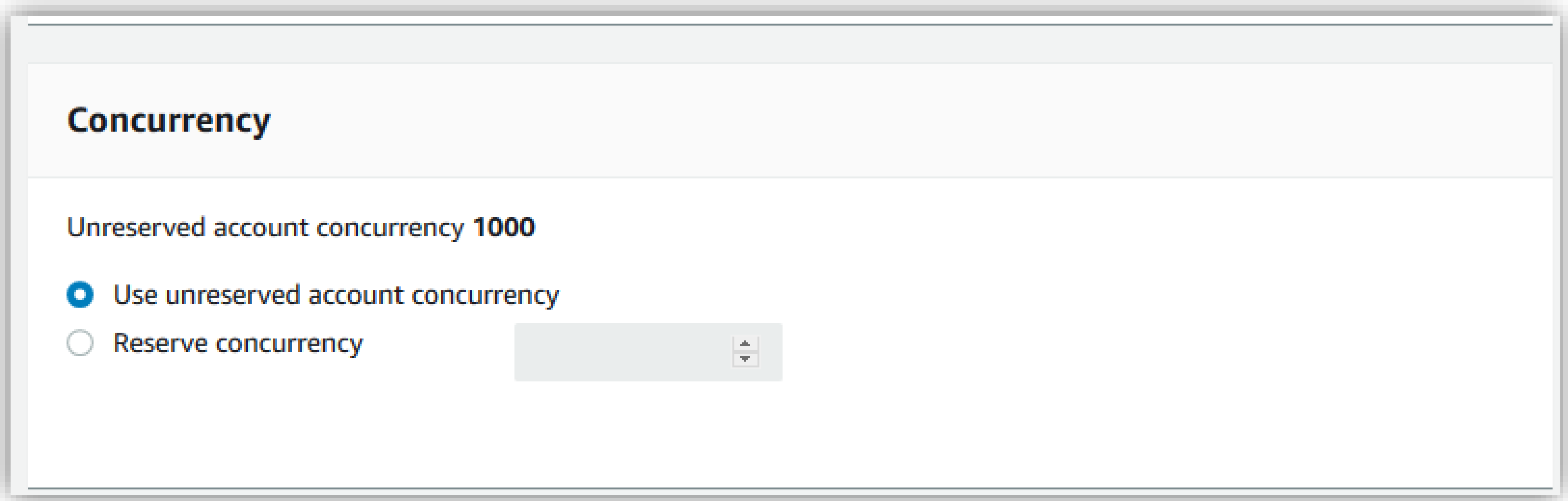
- The default function timeout is 3 seconds, this can be adjusted up to 300 seconds.

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

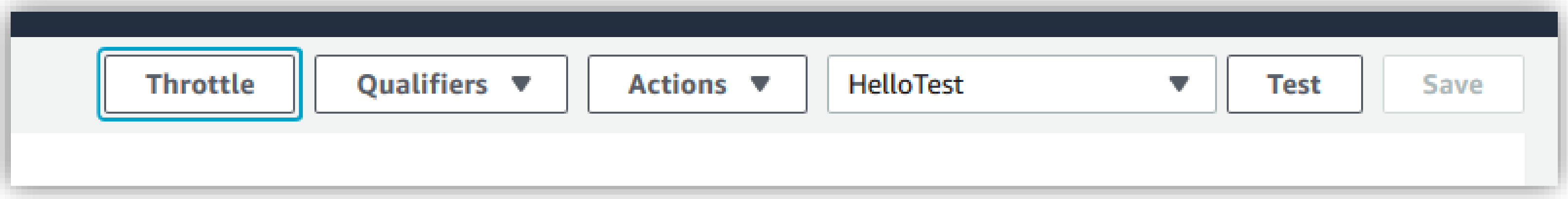
Unreserved account concurrency **1000**

● 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/serverless

Questions

Course Survey

- Before we head in to our final break, and final labs please take the training survey:

- **http://www.metricsthatmatter.com/student/evaluation.asp?k=16324&i=VC00431615**

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/serverless

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/serverless

Q&A / Have a nice day

- Any questions?
- Any lab issues?