

Serverless Computing for the Inquiring Mind

Eoin Woods
Endava

Andy Longshaw
Coop Digital

licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/)

Agenda

- Introduction to Serverless Computing
- Exercise 1: Creating Some Functions
- Architectural Implications
- [Break]
- Exercise 2: Applying the Serverless Style
- Summary and Learning Points

Introduction

Serverless computing

Serverless computing is a cloud computing execution model in which the cloud provider dynamically manages the allocation of machine resources, and bills based on the actual amount of resources consumed by an application, rather than billing based on pre-purchased units of capacity. It is a form of utility computing.

Wikipedia - https://en.wikipedia.org/wiki/Serverless_computing

Huge recent interest in “serverless”



Popularity of “serverless” keyword
on Google Jan 2015 - Present



swardley

Follow

I like ducks, they're foel but not through choice. RT is not an endorsement but a sign that I find a particular ...
Nov 24, 2016 · 21 min read

Why the fuss about serverless?

To explain this, I'm going to have to recap on some old work with a particular focus on co-evolution.

Co-evolution

Let us take a hike back through time to the 80s/90s. Back in those days, computers were very much a product and the applications we built used architectural practices that were based upon the characteristics of a product, in particular mean time to recovery (MTTR)

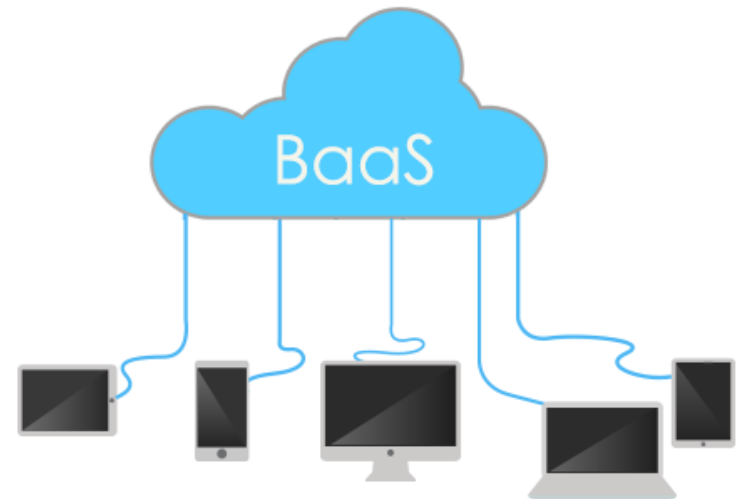
When a computer failed, we had to replace or fix it and this would take time. The MTTR was high and architectural practices had emerged to cope with this. We built machines using $N+1$ (i.e. redundant components such as multiple power supplies). We ran disaster recovery tests to try and ensure our resilience worked. We cared a lot about capacity planning and scaling of single machines (scale up). We cared an awful lot about things that could introduce errors and we had change control procedures designed to prevent this. We usually built test environments to try things out before we were

<https://hackernoon.com/why-the-fuss-about-serverless-4370b1596da0>

Serverless computing variants



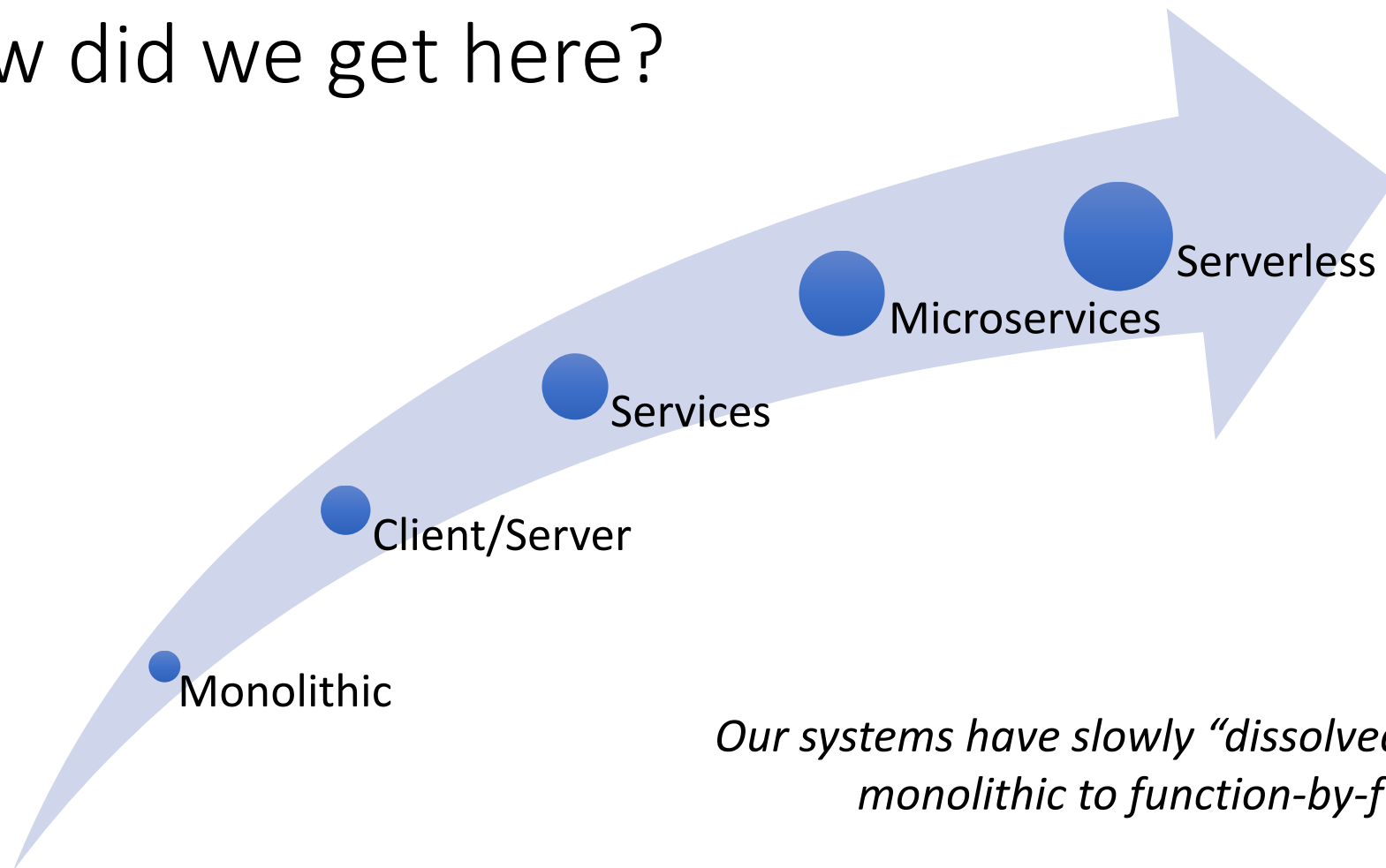
Function as a Service





Backend as a Service

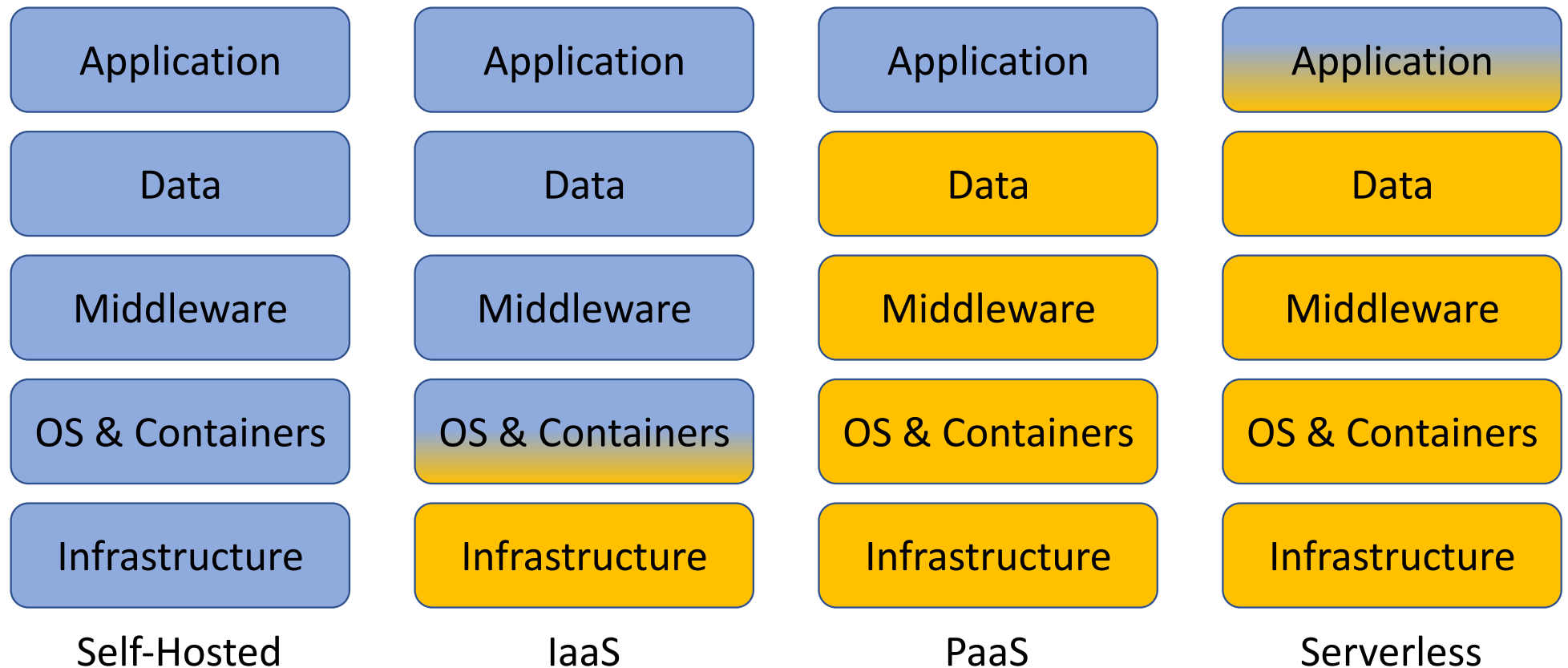
In this session we'll just look at Function as a Service

How did we get here?



What does it mean?

self-managed 
vendor service 



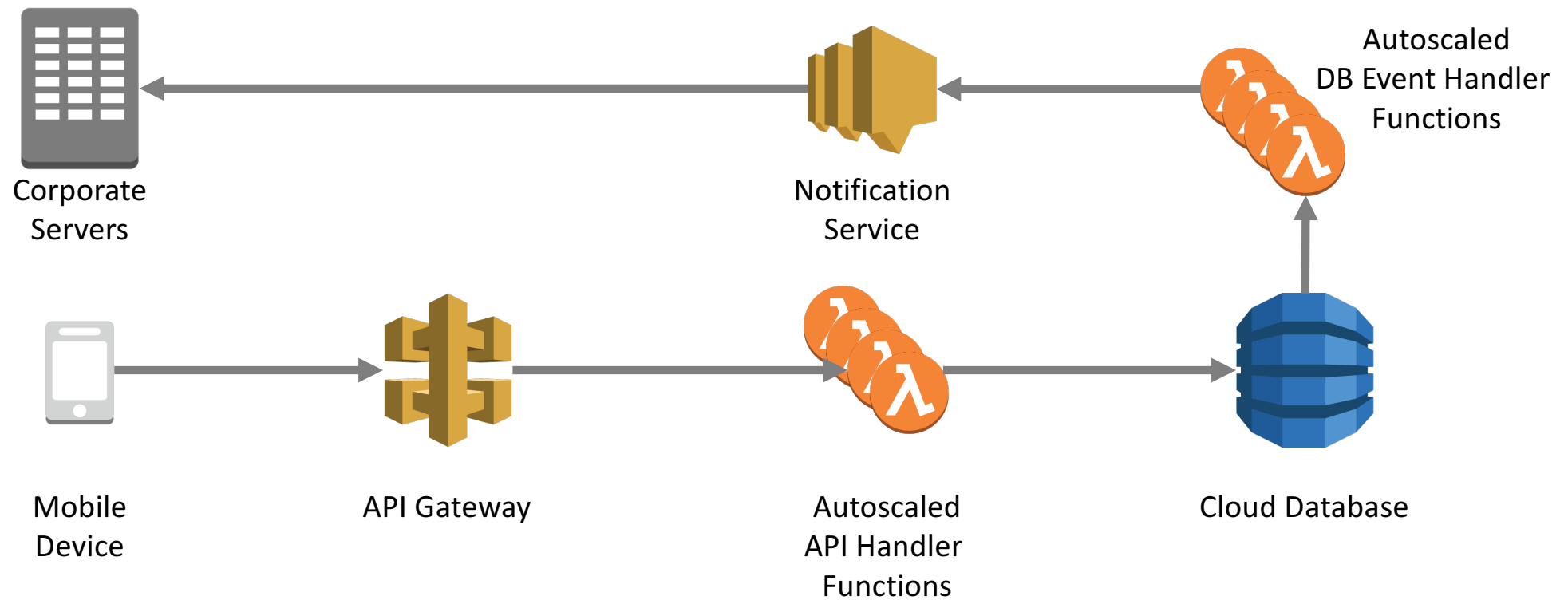
What does it mean?

- Software developed and deployed as individual functions
- Function is a first class object – cloud runs them for you
- Functions triggered by “events”
- Call cloud services to perform complex tasks or have side-effects
- Fixed set of languages
- Fixed programming model
- Cloud specific syntax, semantics
- Cloud specific services

What does it mean?

```
1  from __future__ import print_function
2
3  def calc_integers(event, context):
4      x = int(event['x'])
5      y = int(event['y'])
6      op = event['operator']
7
8      result = None
9      if op == '+':
10         result = x + y
11     elif op == '-':
12         result = x - y
13
14     print("{0} {1} {2} = {3}".format(x, op, y, result))
15     return result
16
```

Simple example



Implementations



AWS Lambda

- Node, Python, C#, Java
- API trigger or many AWS services
- Env vars for config

OpenWhisk

- Python, Java, Swift, Node, Docker
- IBM and 3rd party service triggers
- Code engine is Apache OpenWhisk



Azure Cloud Functions

- C#, F#, Node + Scripthost
- Extends Webjobs (inherits features e.g. config)
- API or many Azure service triggers



Google Cloud Functions

- Node only so far (beta service)
- Triggers on cloud storage, pub/sub message or API invocation
- Watch for variant which are Firebase only (db triggers)



Exercise 1 – 45 Minutes

Objective: develop a simple set of serverless functions on AWS

We have a set of instructions and a Github repository to assist

<https://github.com/andylongshaw/serverless>

Break – 15 Minutes

Architectural Implications

Architectural Implications

At the micro-level serverless seems pretty simple.
Just develop a procedure and deploy to the cloud...

... of course there's always a tradeoff somewhere ...

... what impact does this have on the overall design of
the system? (“the architecture”)

Architectural Implications

Cost

Complexity

Testing

Change

Emergent Structure

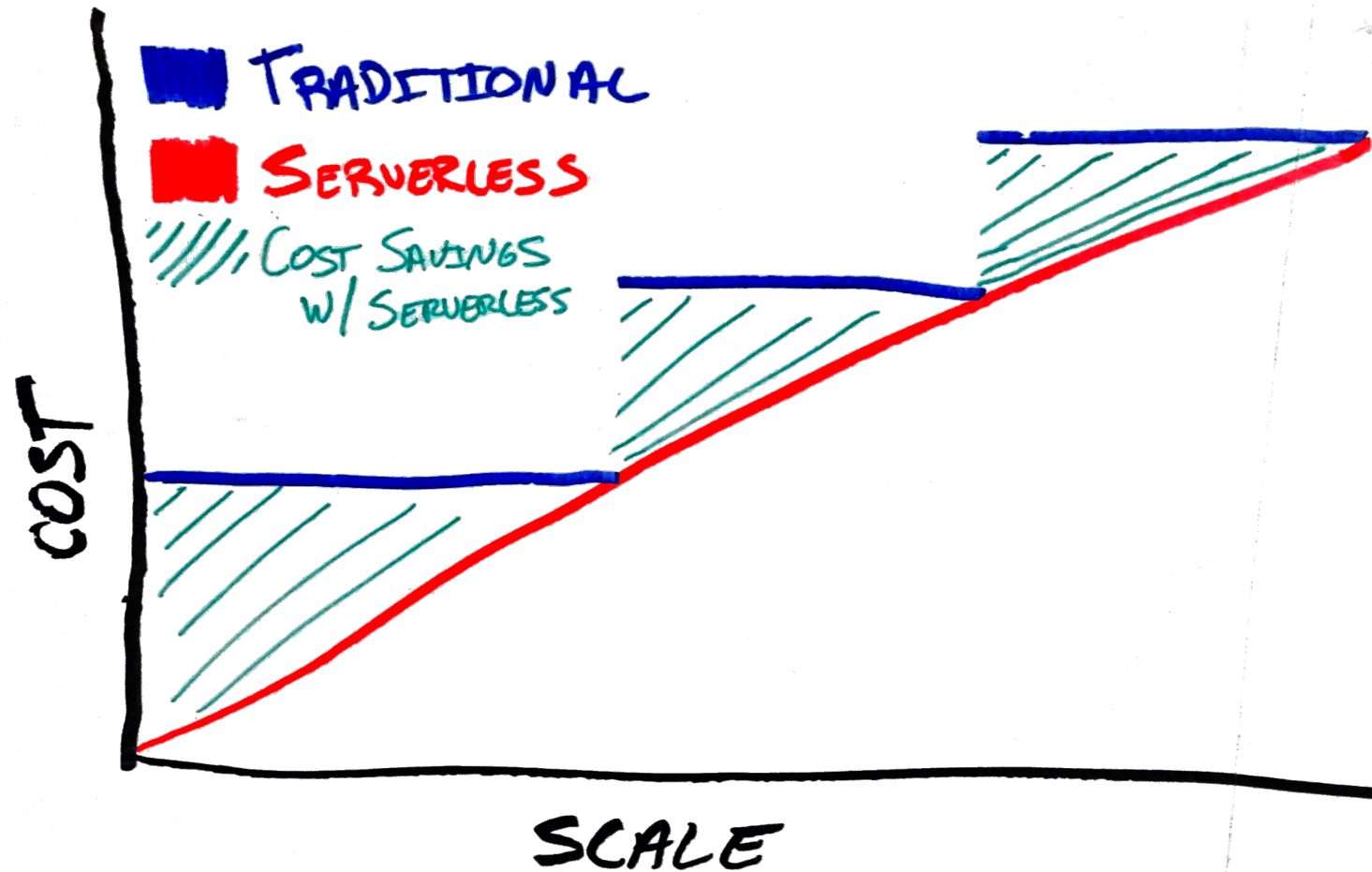
Operations

Vendor Dependency

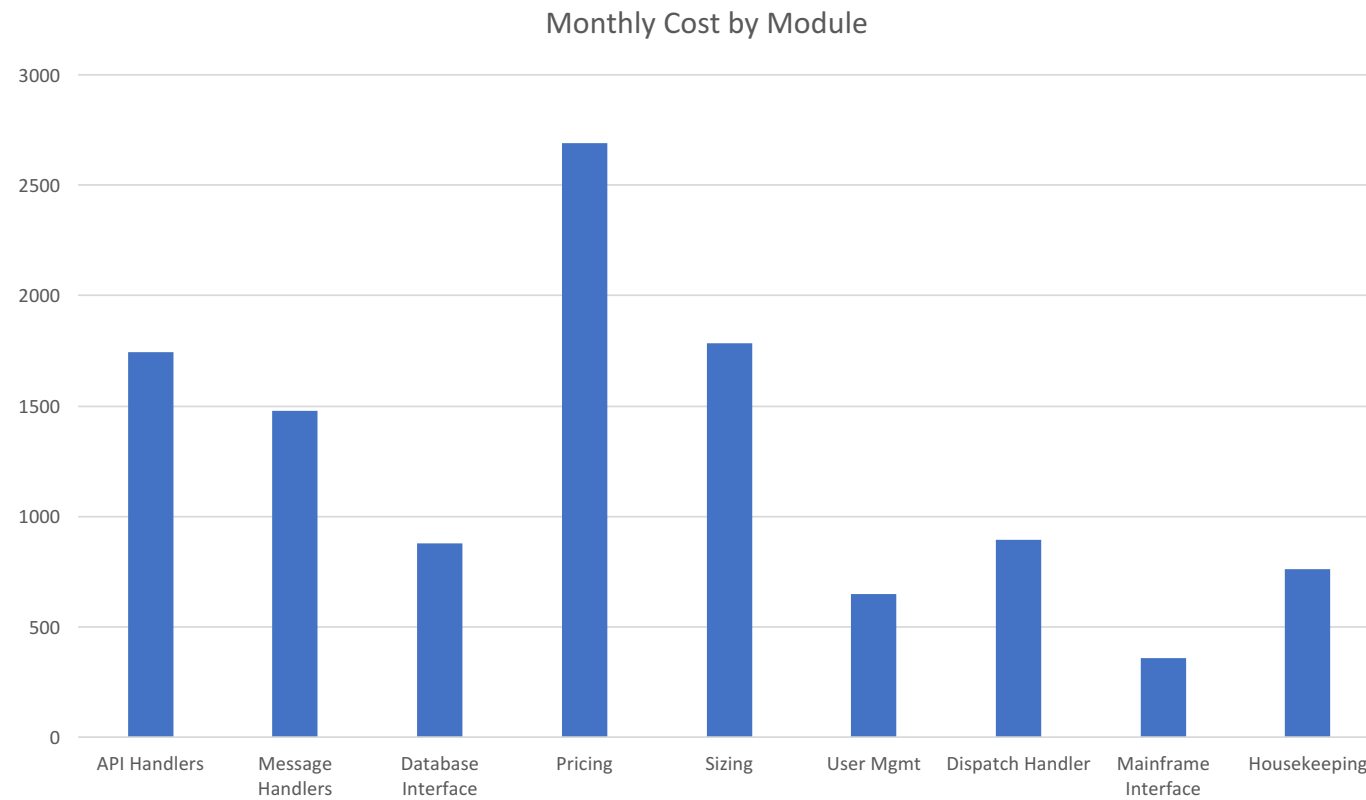
Programming Model

Scalability

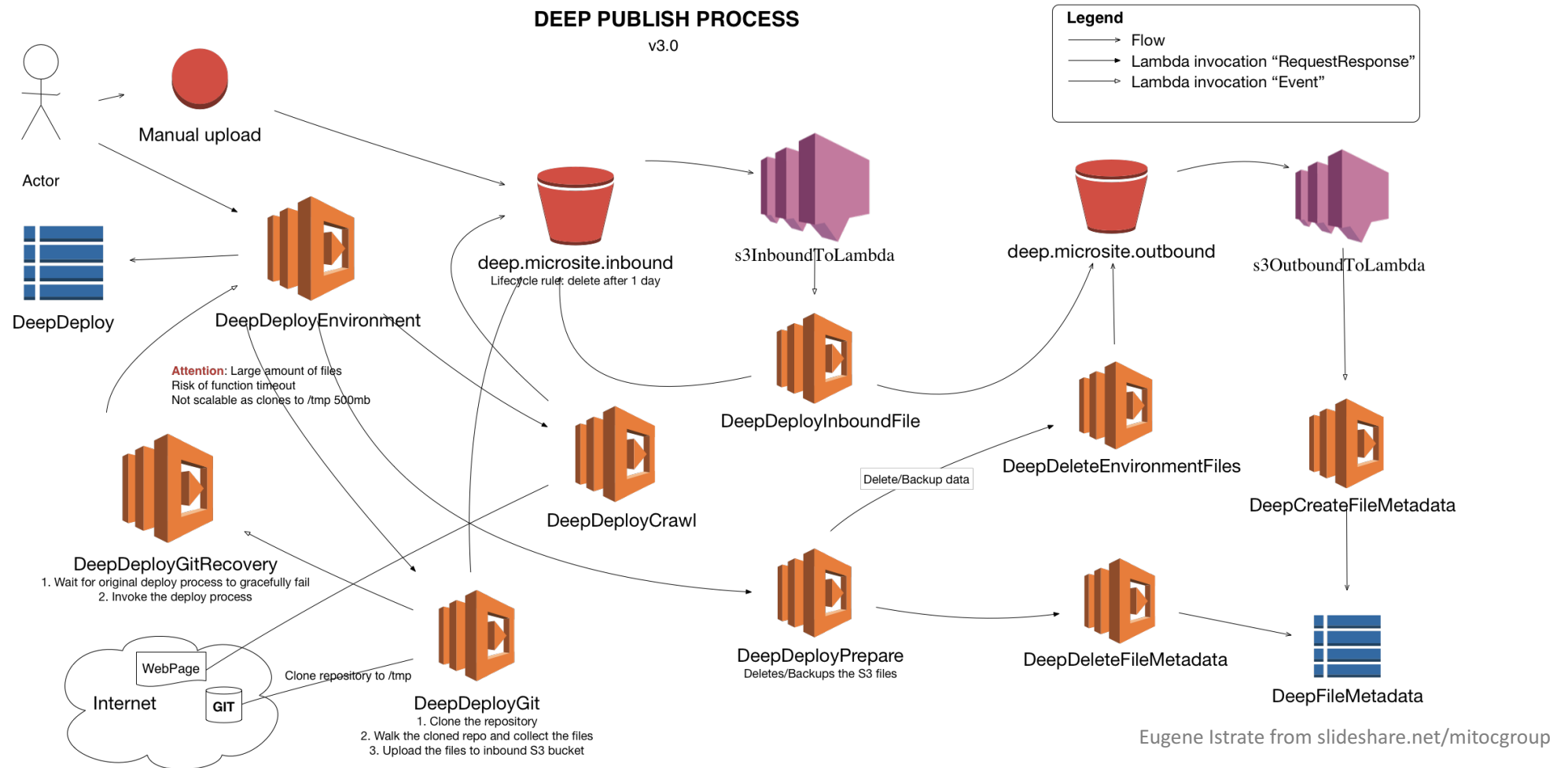
Cost



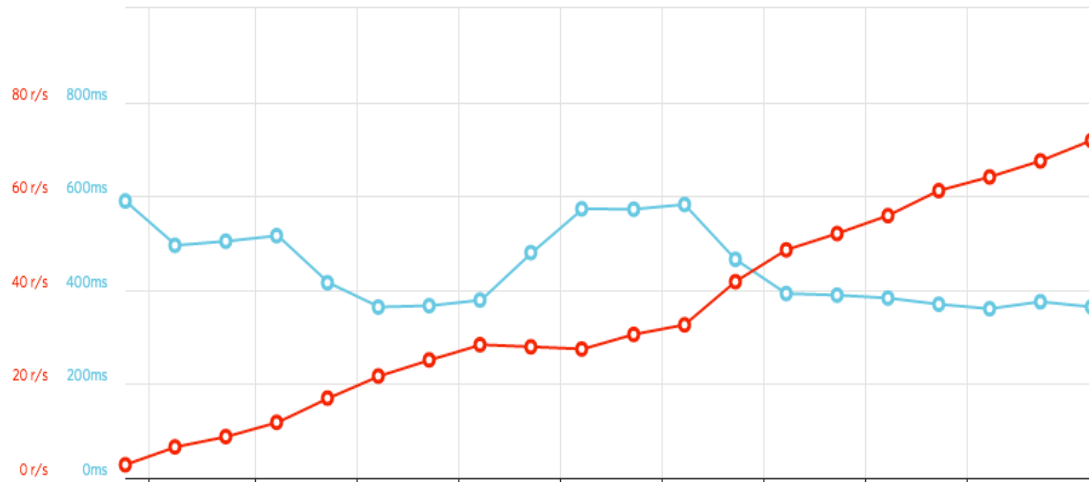
Cost Transparency



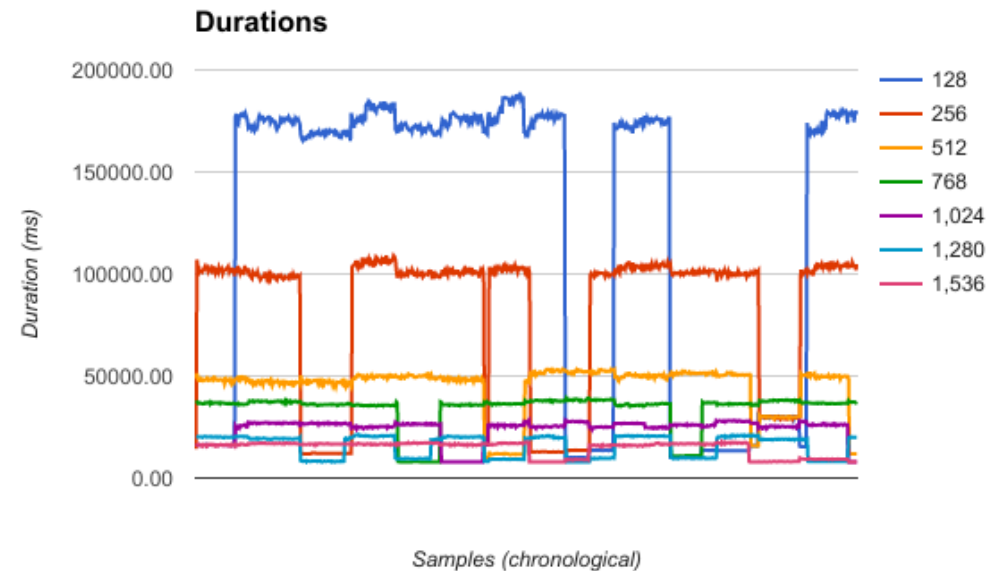
Complexity



Scalability (& Predictability)

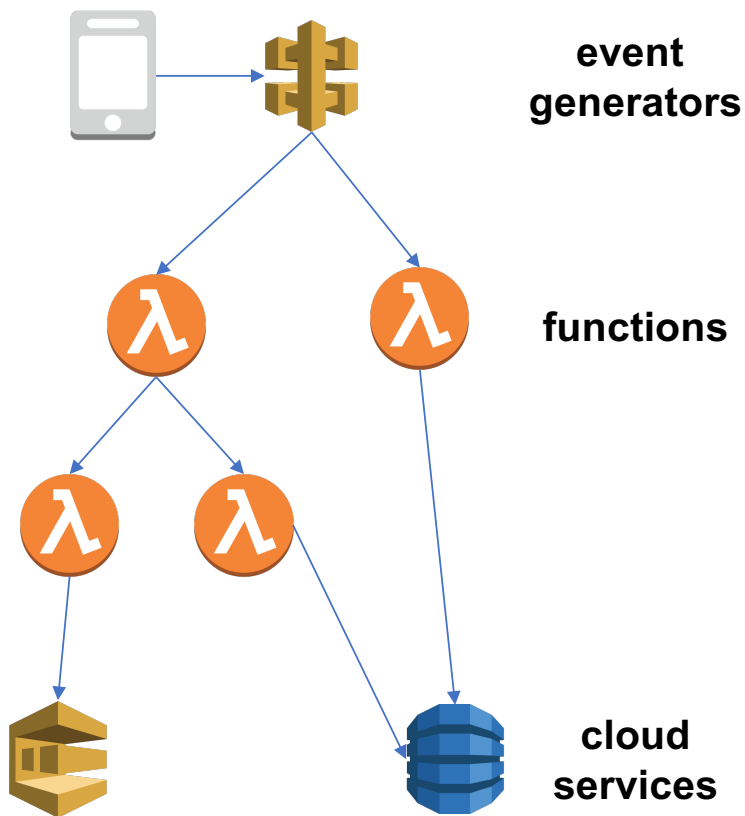


Alex Casalboni
<https://cloudacademy.com/blog/google-cloud-functions-serverless/>



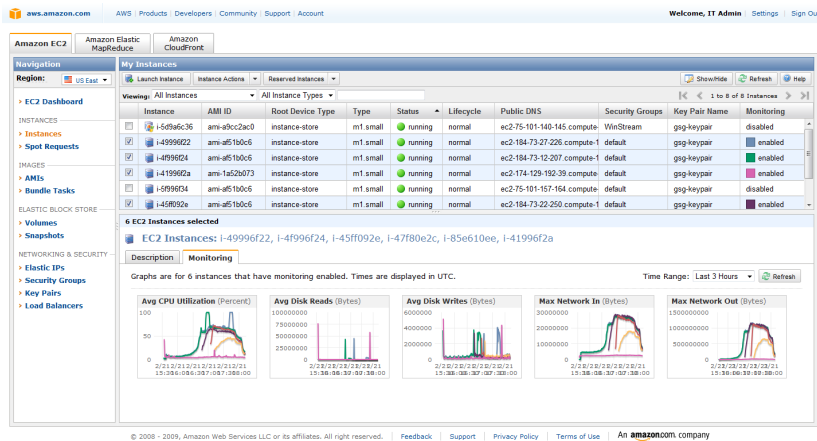
John Chapin
<https://blog.symphonia.io/the-occasional-chaos-of-aws-lambda-runtime-performance-880773620a7e>

Programming Model

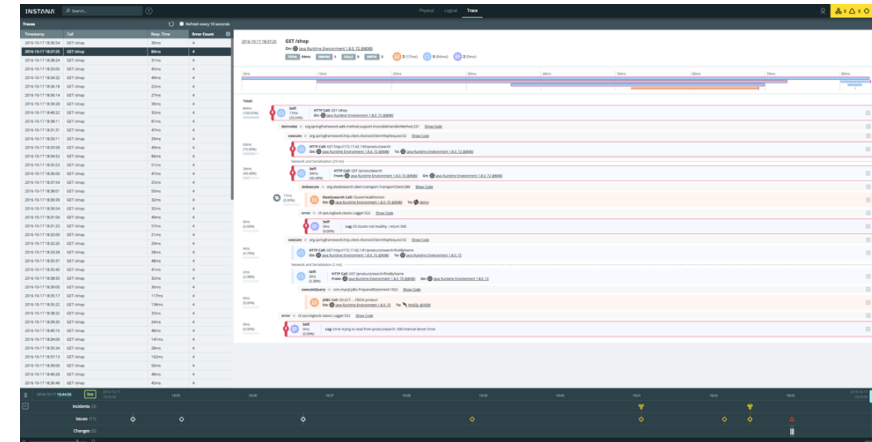


- Mandatory event-driven model
- Mandatory distributed thinking
- Often tied to proprietary cloud services for side effects or complex tasks
- Limited cloud-specific set of languages
- Local dev and test can be limited

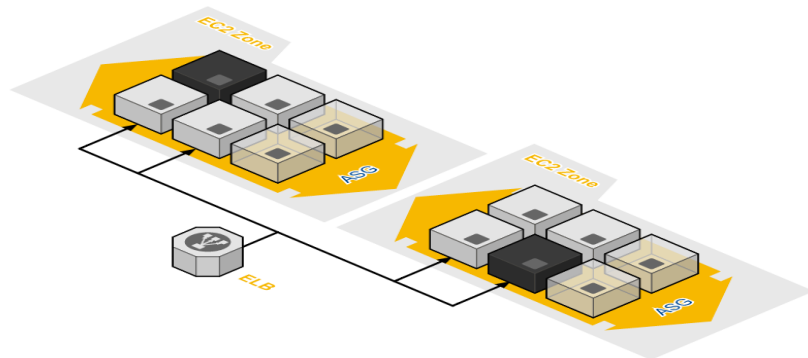
(Some) Operational Concerns



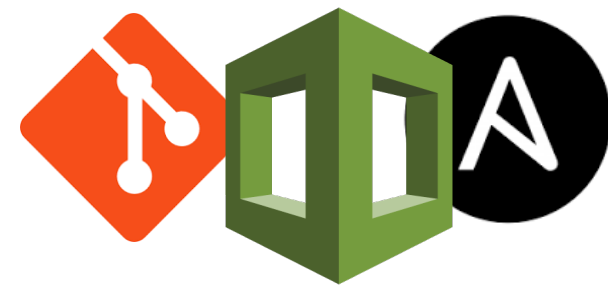
Monitoring and Visibility



Tracing and Fault Analysis



Failover and Recovery



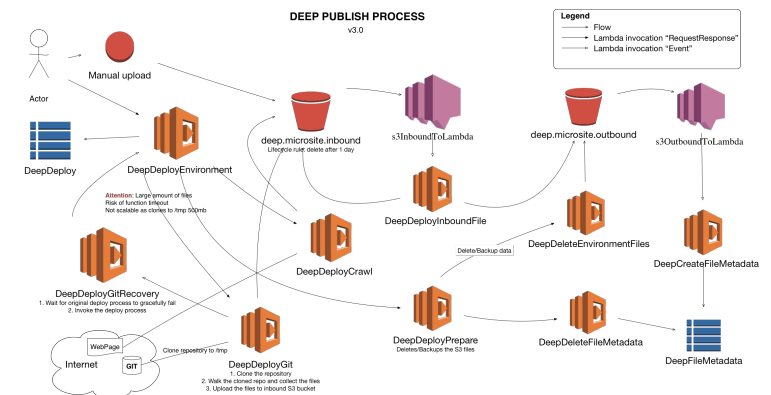
Configuration Management

Testing

```
Function name is: example
Executed example.example_handler
Estimated...
...execution clock time:      212ms (300ms billing bucket)
...execution peak RSS memory: 368M (385900544 bytes)
-----RESULT-----
value1
```

local unit testing (separate logic, emulators, mocks)

- Integration testing on cloud critical
- Needs very high degree of automation
- Fine grained change => flexibility in test sets



other testing needs the cloud

Architectural Implications

Quality	Implication
Cost	Incremental consumption; fine grained monitoring
Change	(very) fine grained change possible
Complexity	very loose coupling; many, many pieces
Emergent	architecture tends to emerge at runtime
Scalability	horizontal scalability automatically
Programming Model	programming model very constrained in terms of structure and languages; single execution; no “sessions”; fixed set of trigger types
Vendor Dependency	feature set depends on vendor; no practical portability at all (porting via rewrite); future cost model unknown
Operation	many small pieces to monitor and manage; can be high rate of change; fault analysis can be significantly more difficult
Testing	unit testing with mock services easy; larger scale testing complex

Exercise 2 – Applying Serverless – 30 minutes

- Consider a system you know well
 - We have an example if you need one
- Consider rebuilding its architecture to include serverless functions
 - Which parts of the system would benefit? Why?
 - Which parts would not benefit from serverless computing? Why?
 - What would the new system architecture be?
- What sorts of problem does it “fit”?
- Report back your thoughts and findings

Summary

Summary and Reflection

- Serverless is creating a lot of interest
- Unique economics, transparent infrastructure and potential for rapid change seem to be the main drivers
- Imposes a lot of constraints on developers and brings a load of new complexities in return, as well as cloud lock-in
- For cloud native applications it offers an interesting new option
- What would you use it for? And why?

Thank You

Eoin Woods

Endava

eoin.woods@endava.com

Andy Longshaw

Coop Digital

andy@blueskyline.com