

ORACLE®

Cloud Native Labs



# Serverless Is the Abstraction We Deserve

---

Jesse Butler Cloud Developer Advocate, Oracle Cloud Infrastructure

 @jlb13

#OracleCloudNative  
[cloudnative.oracle.com](https://cloudnative.oracle.com)

“Serverless is the hero you deserve, but not the one you need right now.”

- Me, like a year ago

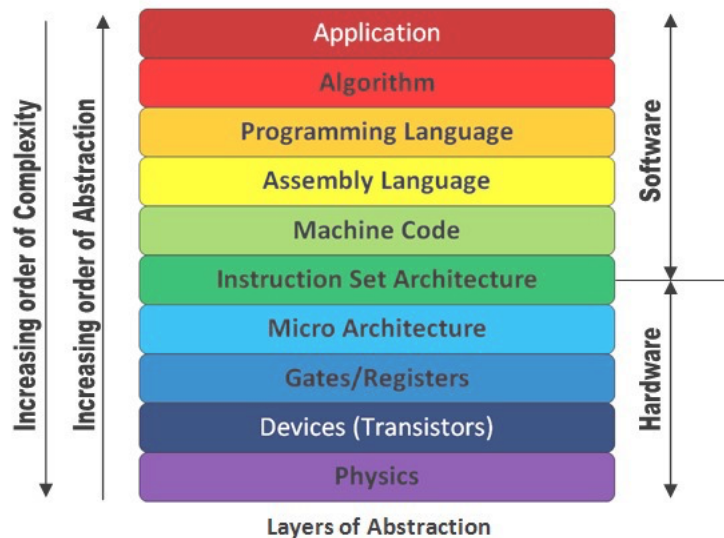
# Serverless is Just the Next Abstraction, Right?

- Why the friction? Is there fear?
- We might be unclear regarding what Serverless is really offering
- Especially true if we don't fully understand the liabilities within our current solutions



# Abstraction Enables Powerful Solutions

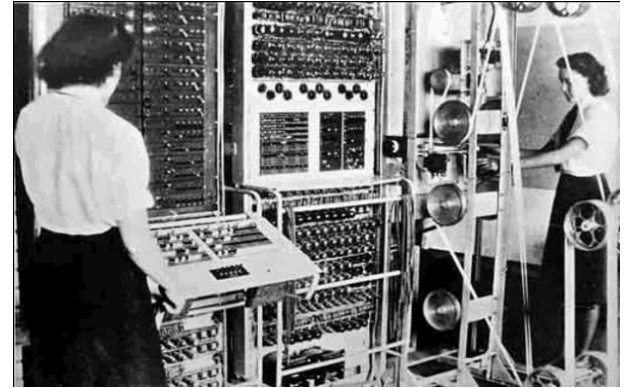
- Abstractions are good for computing
- Tubes led to transistors, ICs eventually led to multi-core CPUs
- Machine language led to assembly, C and the whole slew of higher level languages that we love more than C





# Abstraction Lowers the Barrier to Entry

- If we still needed soldering and wire-wrap skills to do computering, the world would be a very different place
- Word processing, music production, programming – everything benefits from abstraction



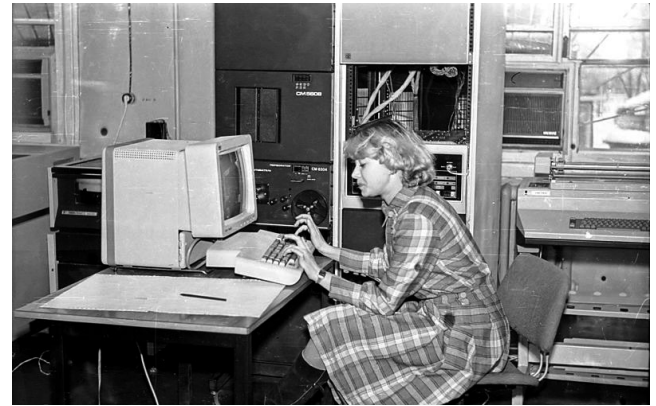
# Abstraction Lowers the Barrier to Entry

- If we still needed soldering and wire-wrap skills to do computering, the world would be a very different place
- Word processing, music production, programming – everything benefits from abstraction



# Abstraction Lowers the Barrier to Entry

- If we still needed soldering and wire-wrap skills to do computing, the world would be a very different place
- Word processing, music production, programming – everything benefits from abstraction



# Abstraction Lowers the Barrier to Entry

- If we still needed soldering and wire-wrap skills to do computering, the world would be a very different place
- Word processing, music production, programming – everything benefits from abstraction



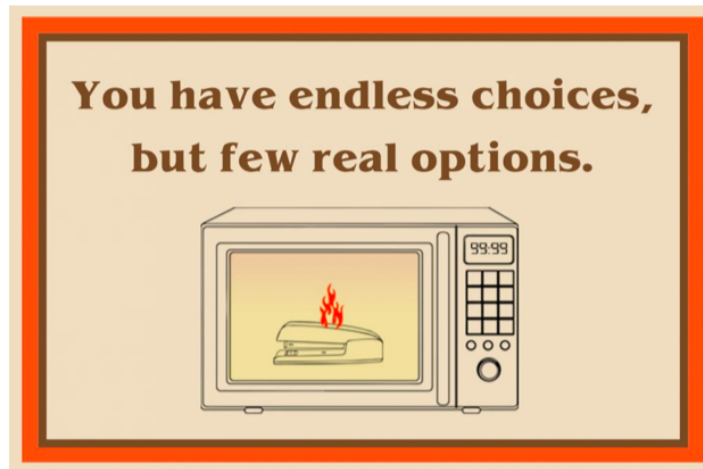
# Abstraction In Systems Can Feel Very Uncomfortable

- When systems and infrastructure are abstracted, things can be weird
- We might lose a lot of our well-worn solutions in this area, specifically
- We feel we are pulling dependencies in that we can't control
- We can start to understand some of the friction...



# Goals

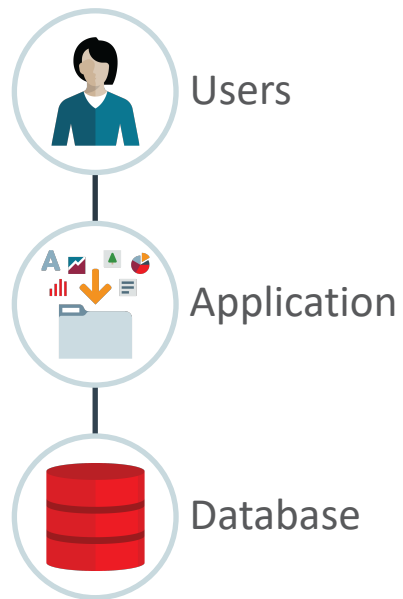
- Describe Serverless as both Architecture and Platform
- Share my opinion regarding the primary benefits of Serverless
- Break down some of the friction, and offer some guidance
- Hopefully help you get started!



# Monolithic Applications

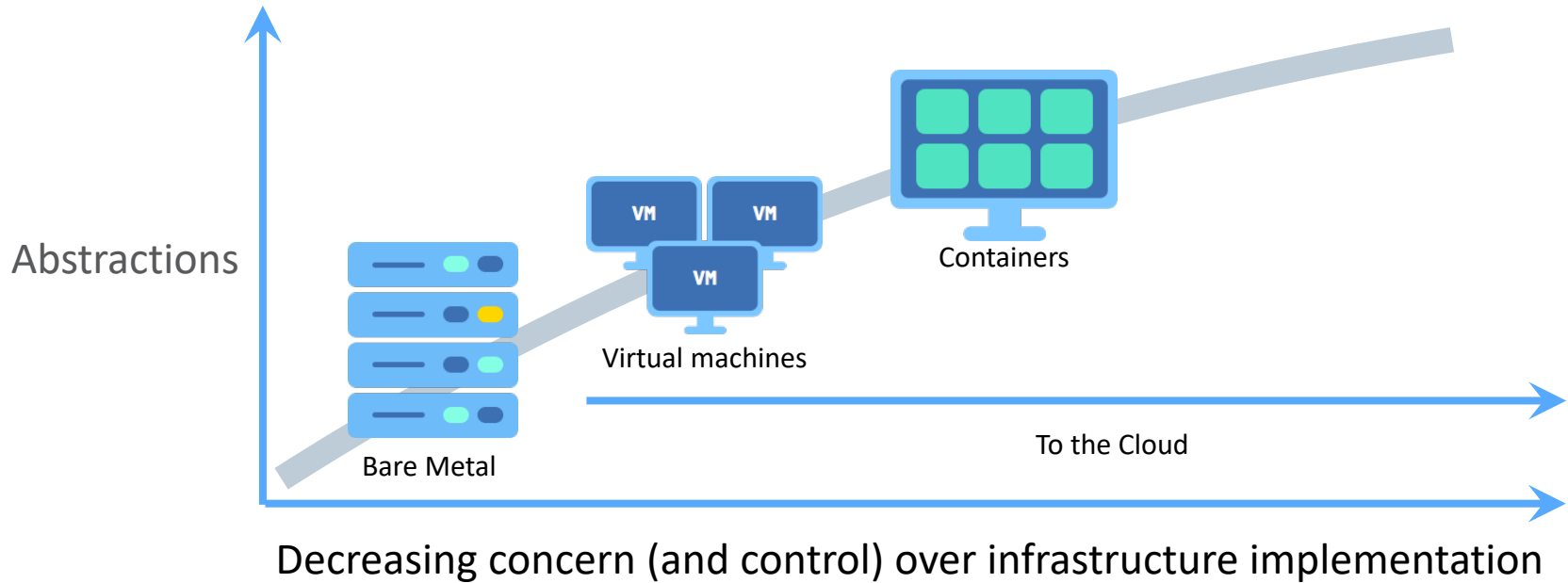


# Monolithic Applications



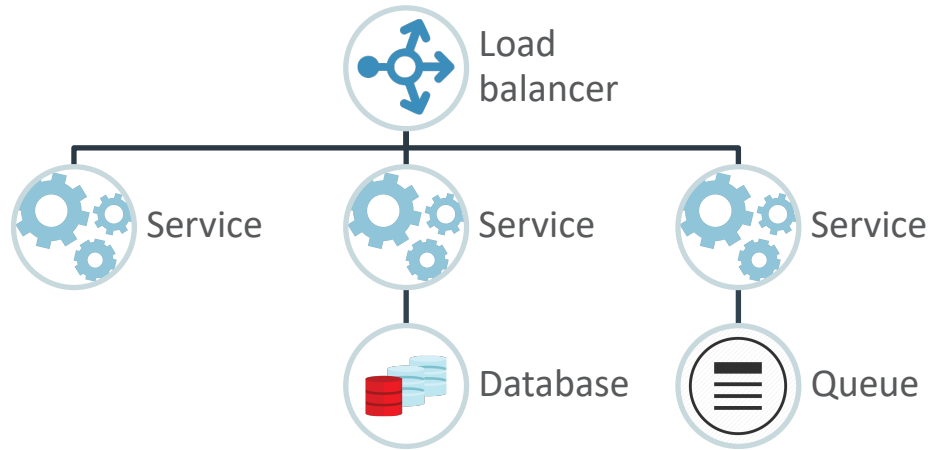


# Virtualization and Consolidation



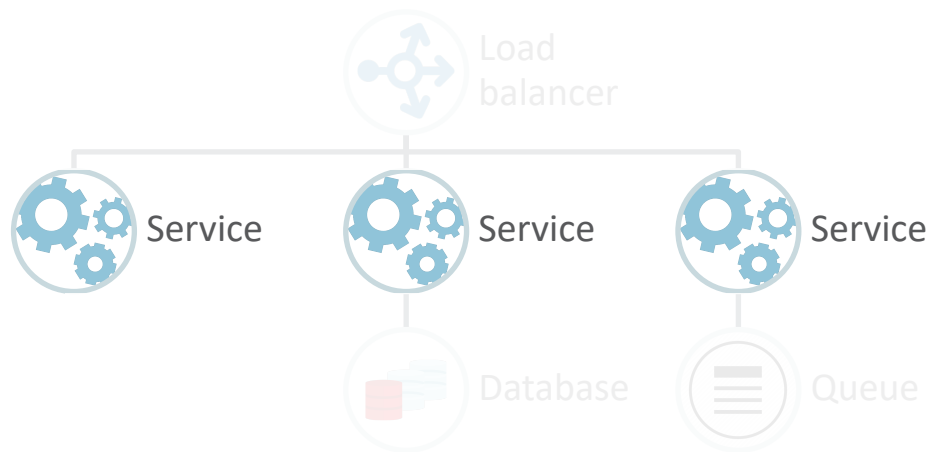
# Microservices

Deploying Code to Systems We Build in the Cloud with Containers and Kubernetes

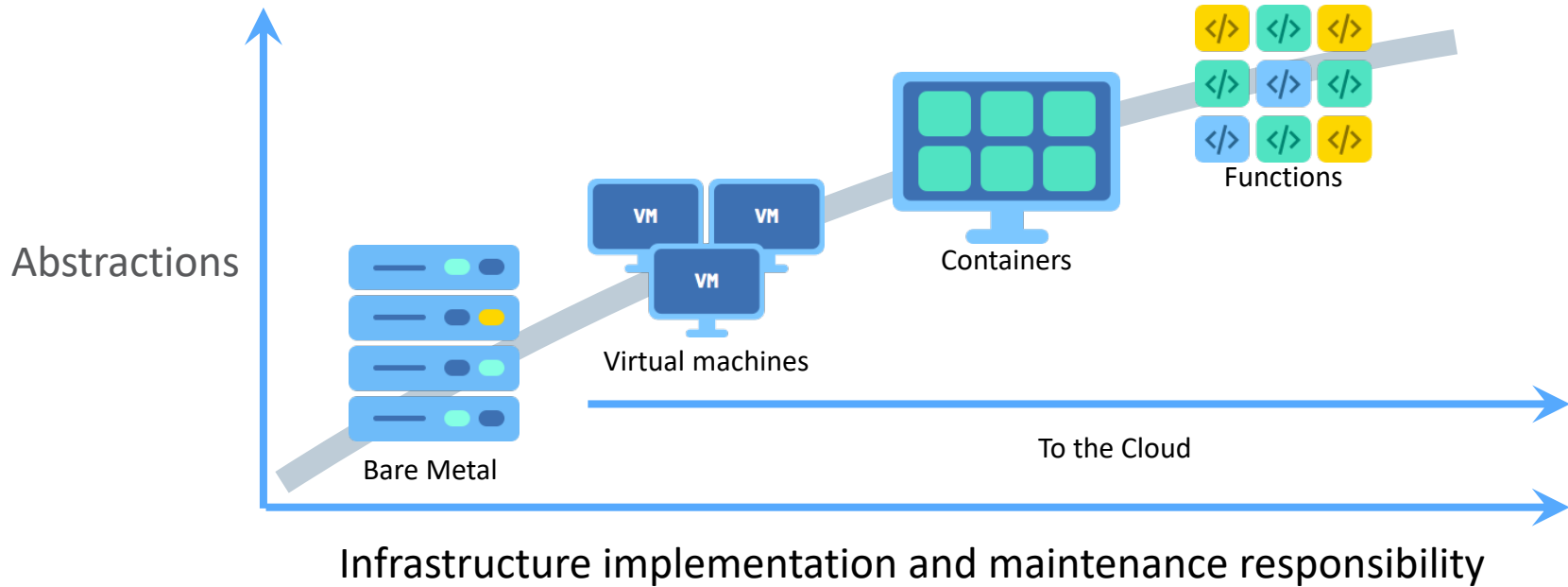


# Serverless

Deploying Code to Systems We Build in the Cloud with Containers and Kubernetes

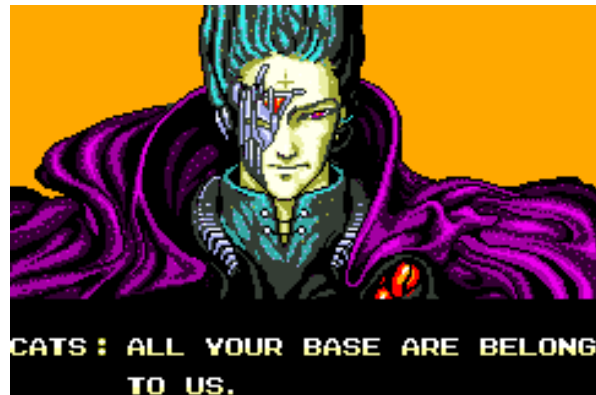


# Trend towards Serverless



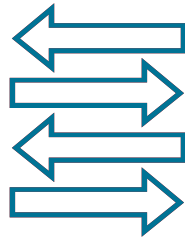
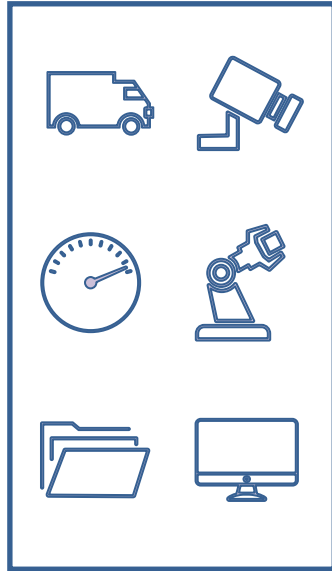
# What Can We Build With Serverless?

- Web and Mobile Backends
- Any other backend API implementations
- Real-time Processing of Files, Streams
- Batch Processing
- Gluing up SaaS things
- Kind of anything



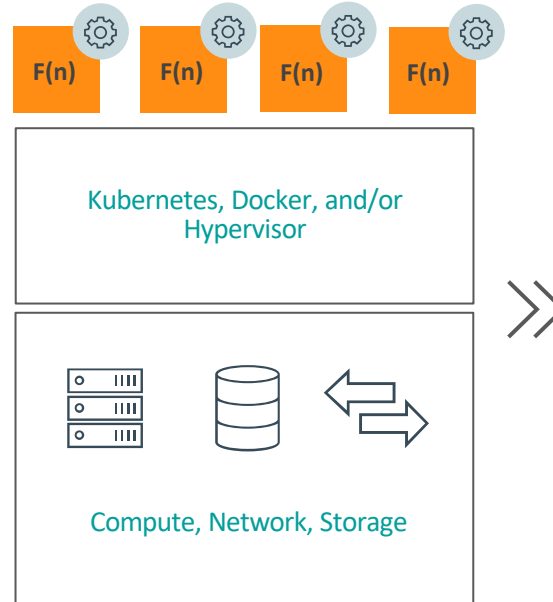
# Serverless as a Platform

## Event Sources



Triggers

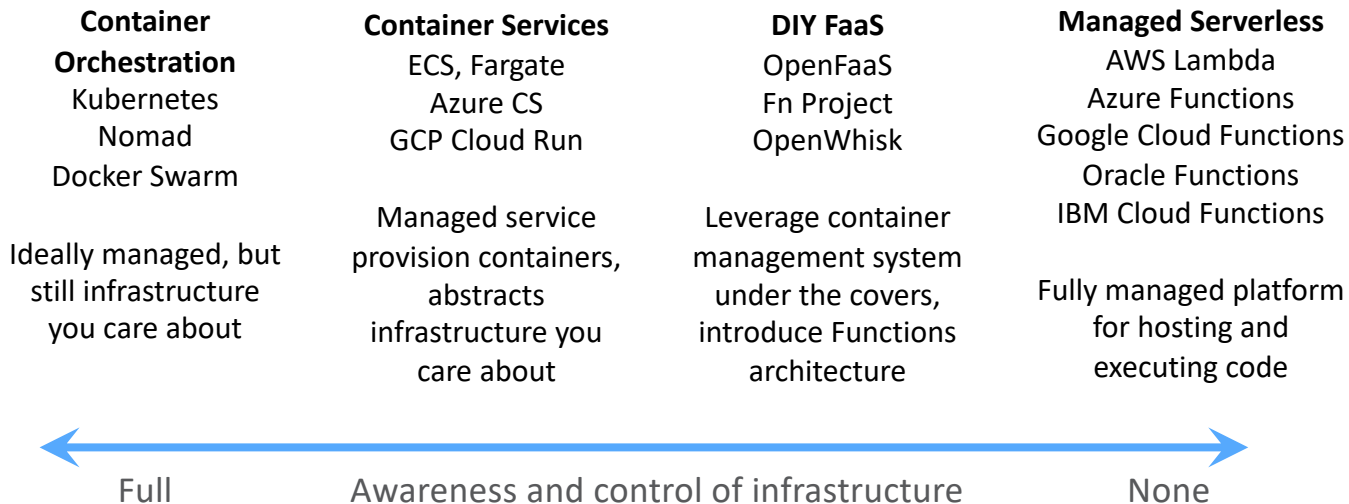
## Function Execution



## Backend Services

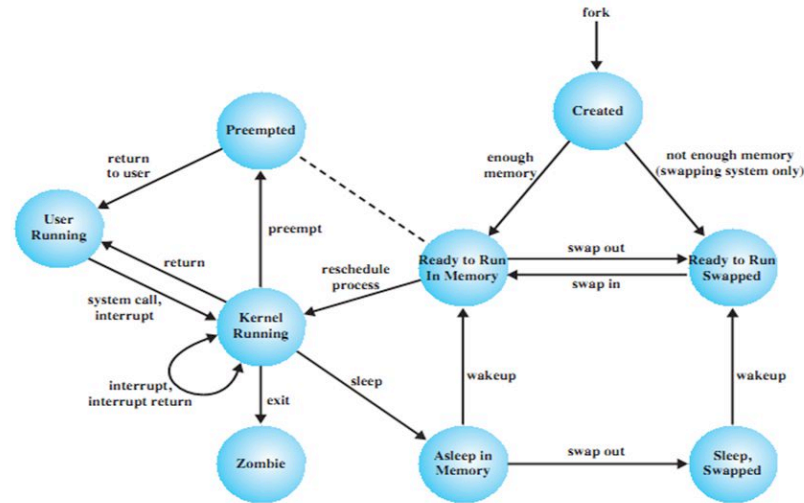


# Serverless is a Spectrum (Stay With Me)



# Containers are the new Process Model, Right?

## UNIX SVR4 States Process Model



40



# Containers are the new Process Model, Right?

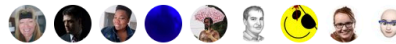


Replying to @dakami @jessfraz

The time has come,' the Captain said,  
To talk of many things:  
Of nodes — and pods — and etcd —  
Of ingresses — and kings —  
And why containers contain not —  
And whether pigs have wings.'

7:58 PM - 15 Apr 2019

16 Retweets 134 Likes



# Don't Conflate Requirements with Complexity Aversion



**Jesse Butler**

@jlb13



The notion that Serverless is less complex than running your own services at scale has a lot to do with the difference between a product and a platform. A lot of people happily use Linux, but with Debian or Fedora wrapped around it.

8:55 AM - 23 Apr 2019

1 Like

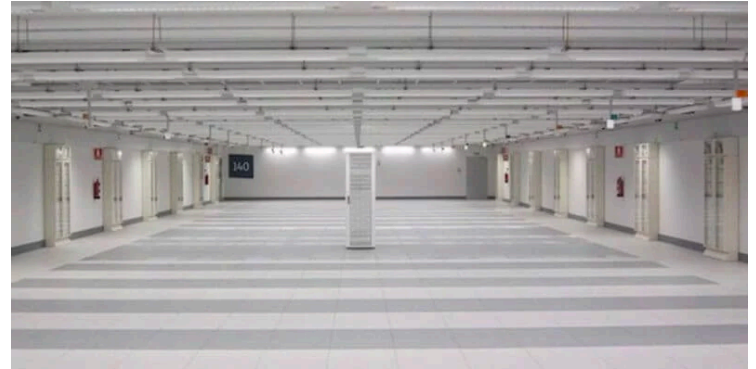


1

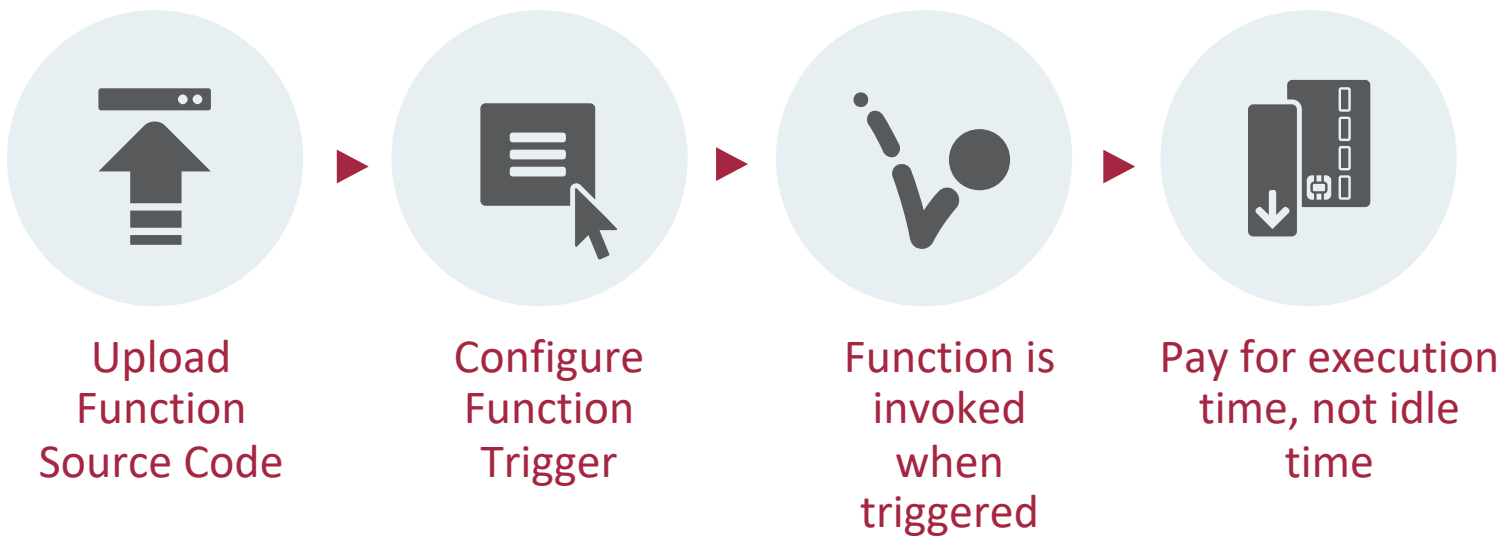


# What Is Serverless

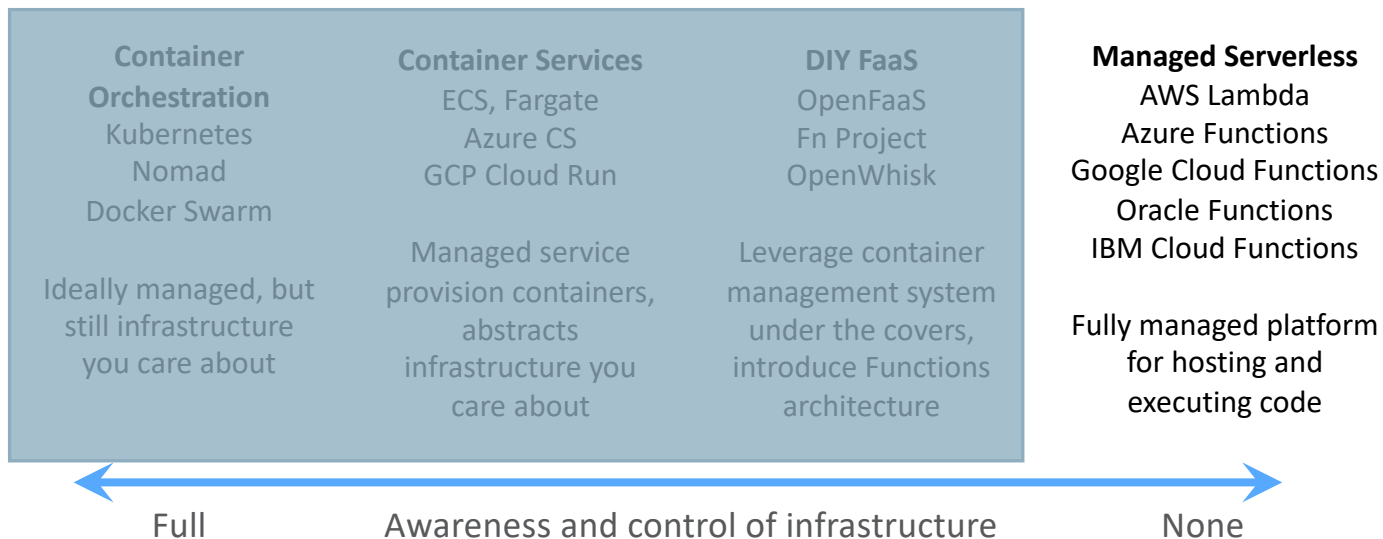
- Event-driven architecture
- Invisible infrastructure
- Automatic scaling on demand
- Granular billing for execution time
- Fault tolerant and highly available



# Serverless Deployment, the Duck Test



# Serverless is a Spectrum



# What Is Serverless, Distilled

## Serverless is a State of Mind

The point is focus—that is the why of serverless



Ben Kehoe

[Follow](#)

Mar 17 · 12 min read

**The point is focus**

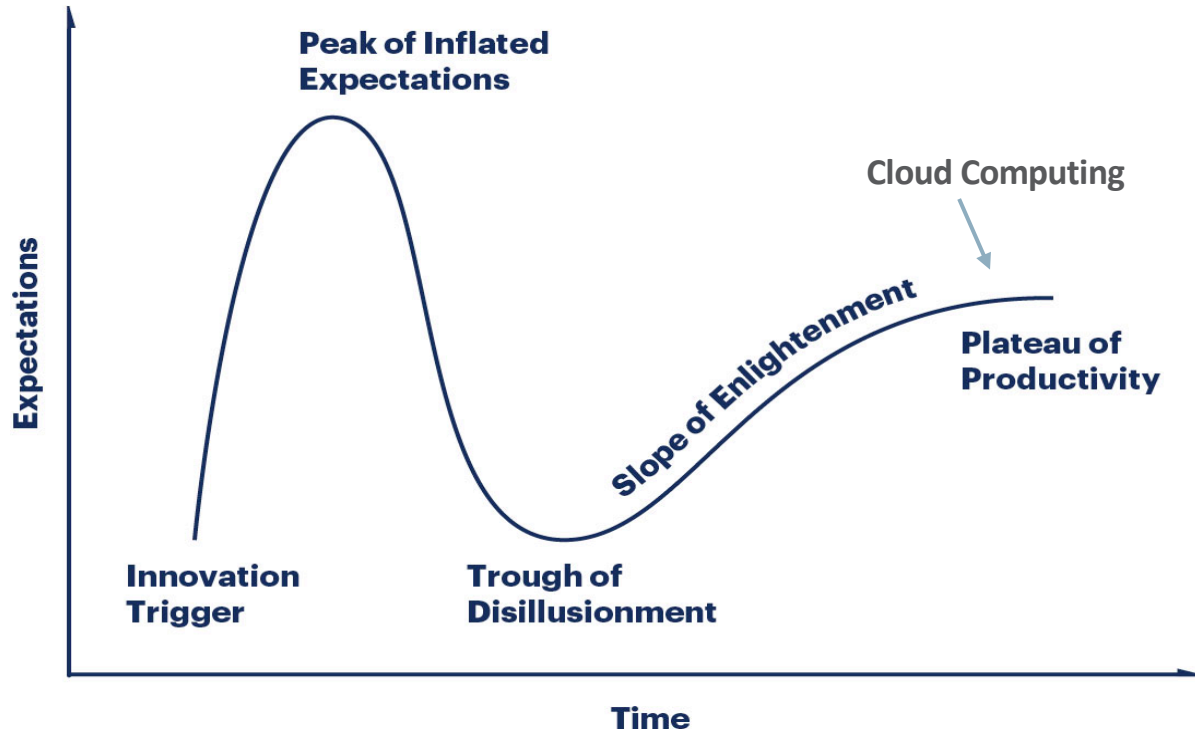
Serverless is a way to focus on business value.

# What Is Serverless Not

- It's not magic, it's a choice
- Brownfield: You need to break the monolith apart regardless
- Greenfield: You need a solid design and to truly understand the platform
- Nothing is free

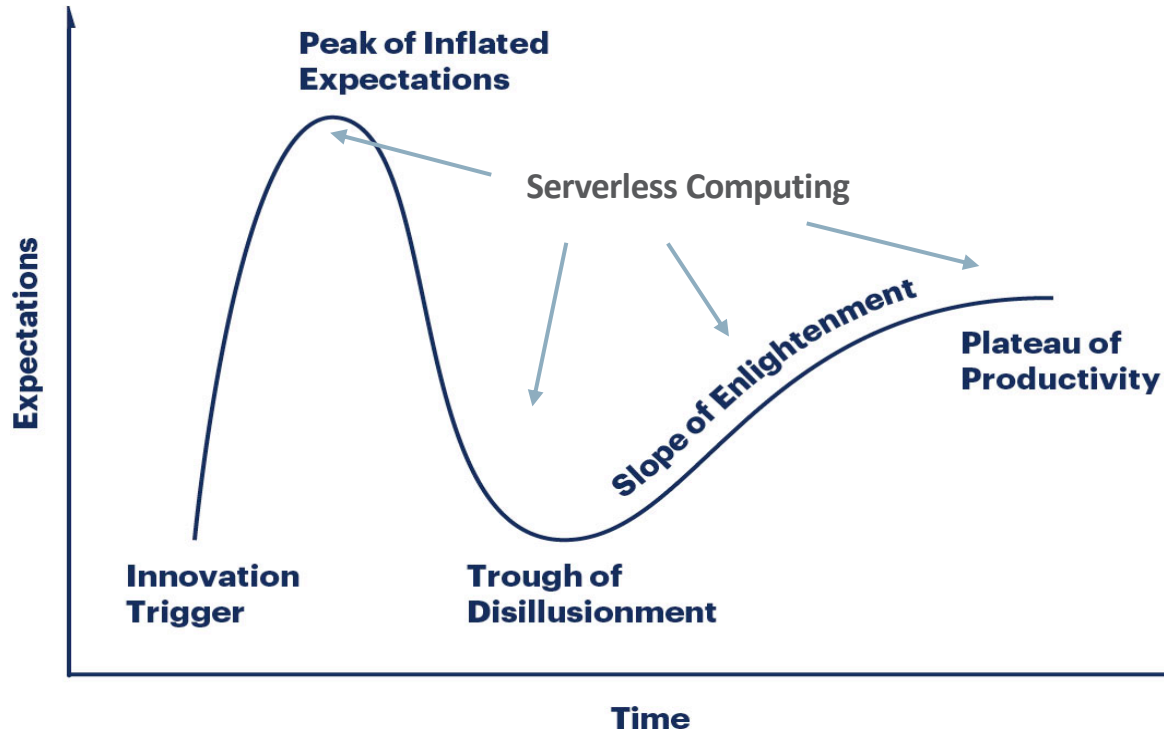


# Hype Cycle – Productive Adoption

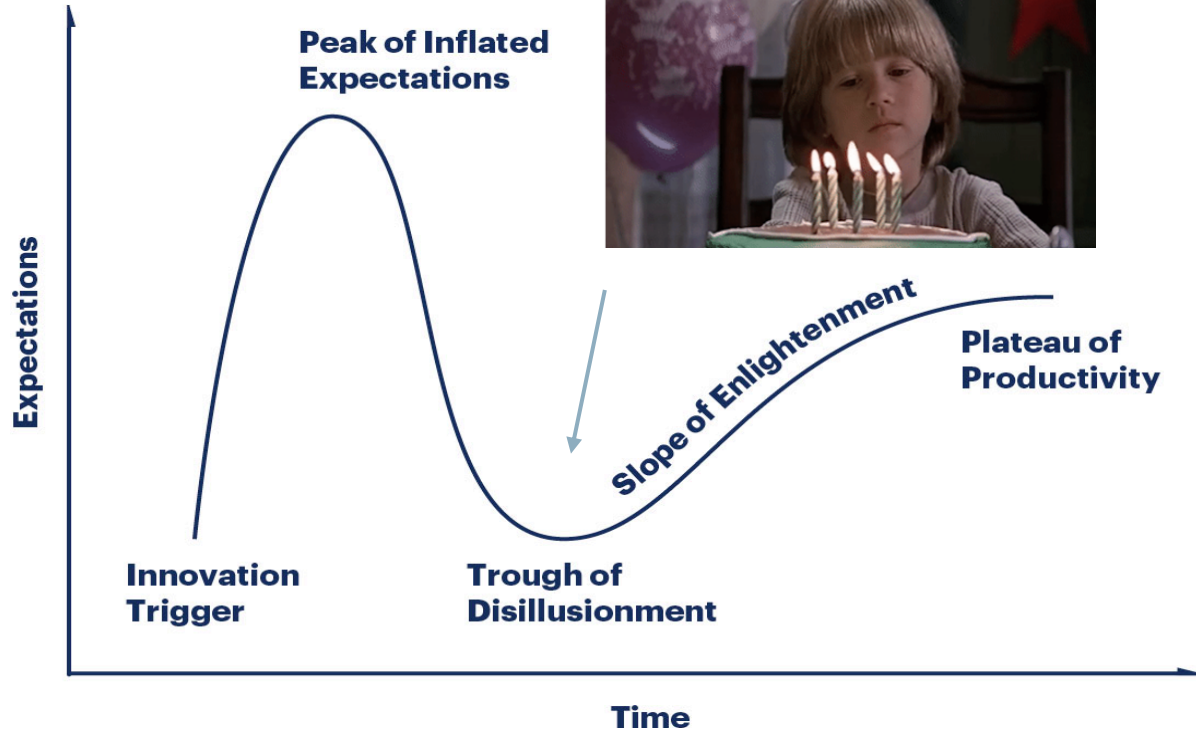




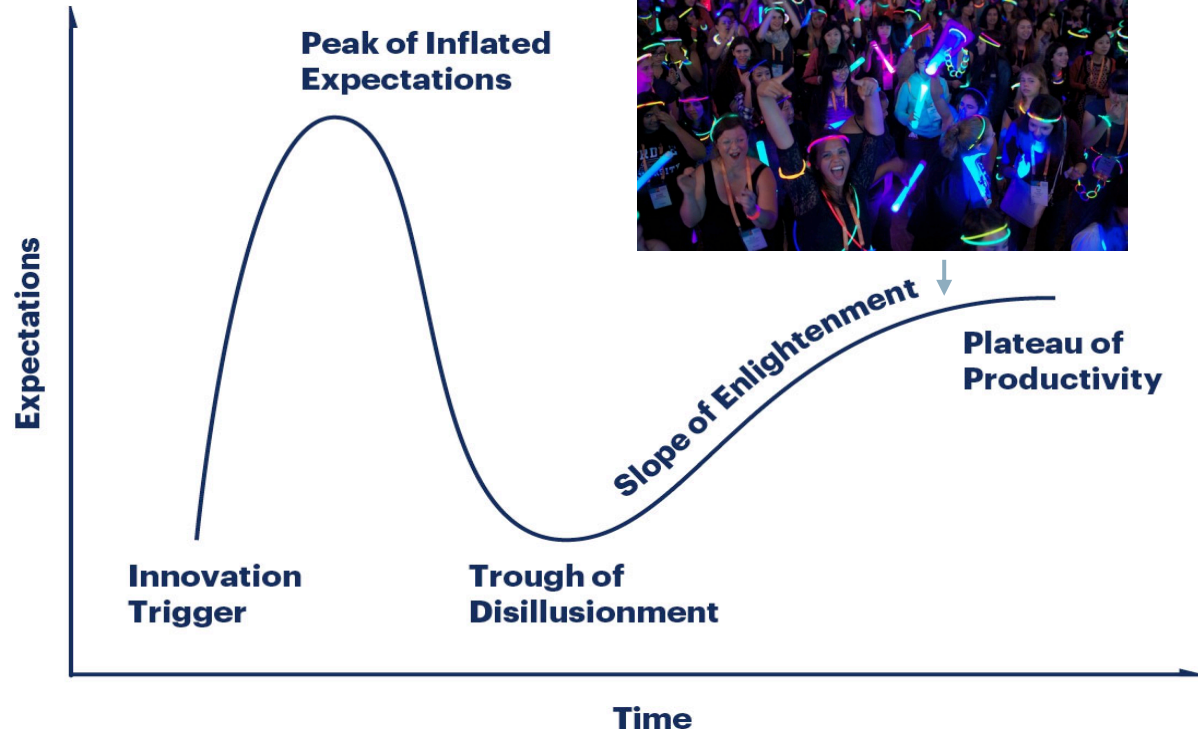
# Hype Cycle – Serverless in Waves



# Get Through This Quickly...



# ...And Get Here



# Serverless the Platform

- Serverless is both an architecture and a platform
- Our abstractions are converging
- Comparatively, it's a big change from cloud, or containers
- Pets vs Cattle isn't a thing with serverless, it's a "don't care"



# Letting Go

- There are trade-offs for the powerful abstractions of Serverless
- We have to let go of the control we feel we have
- Even harder if we've recently migrated to cloud on a different vector



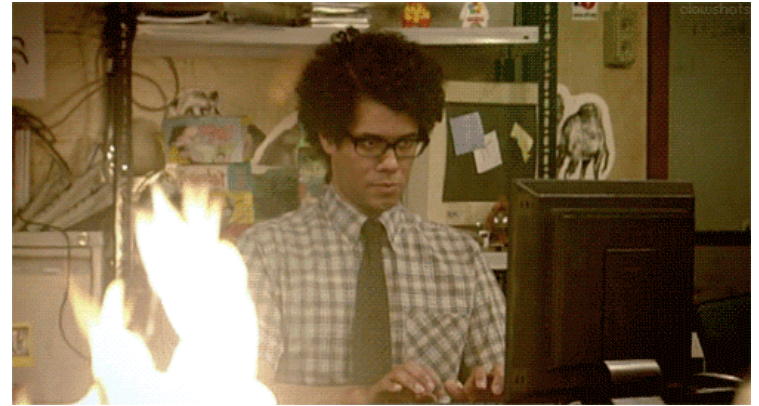
# Mother of all Dependencies

- As we deploy infrastructure, even if it's virtual, we feel in control
- We roll out solutions as needed, when needed, because we can
- To embrace Serverless fully, we lose some of this control and influence
- Or, like... all of it.



# Control From Necessity

- Control often comes out of necessity
- “Something is on fire and I need to find out what’s going on...”



# Standing in the Dumpster

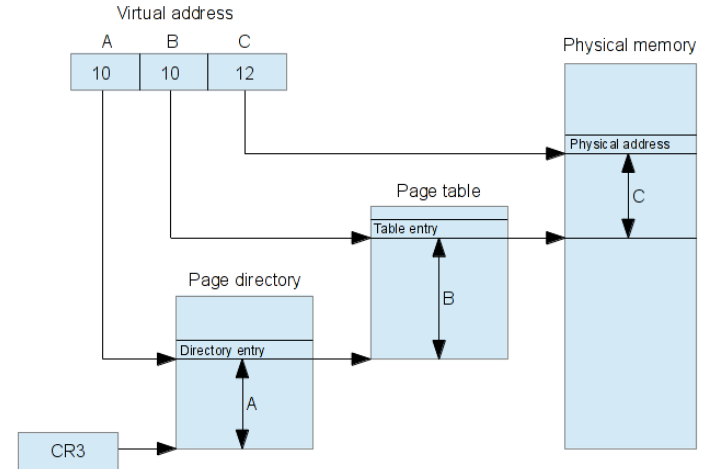
- Sometimes, we build really elegant solutions to these problems
- And sometimes, the fires are due to the systems themselves





# Acceptance of Abstraction within Context

- Often, acceptance teeters on context
- Everyone who writes code uses memory, we don't think too much about the virtual memory implementation
- We can accept some amount of abstraction as a good thing



# The Cloud is One Big Abstraction

- There is, in fact, a cloud. Many, really.
- You can offload a \*ton\* of concerns though, which is nice
- Cloud offers an opportunity to improve the way we build and ship software
- But nothing inherently enforces this



Well, lots of computers, and storage, and network, and managed services, and support, and capacity planning, and... yeah. This is silly.

# Let's Talk About Containers

- Containers are not complicated
- Docker solves many problems with containers elegantly
- But containers are ultimately an implementation detail



# Let's Talk About Kubernetes

- Kubernetes has great abstractions
- Kubernetes is an elegant, extensible solution for running containers at scale
- At the end of the day, it's a platform for building solutions



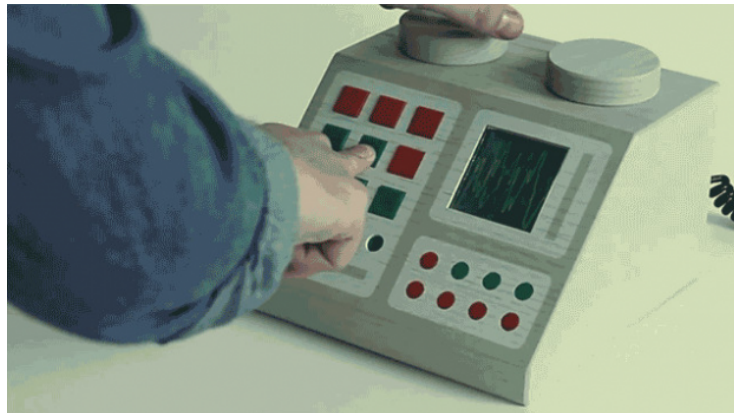
# So What's Wrong With This?

- Plot-twist: nothing, really
- Development teams \*can\* and \*do\* have great success adopting Kubernetes
- For development-focused teams, Serverless just makes sense



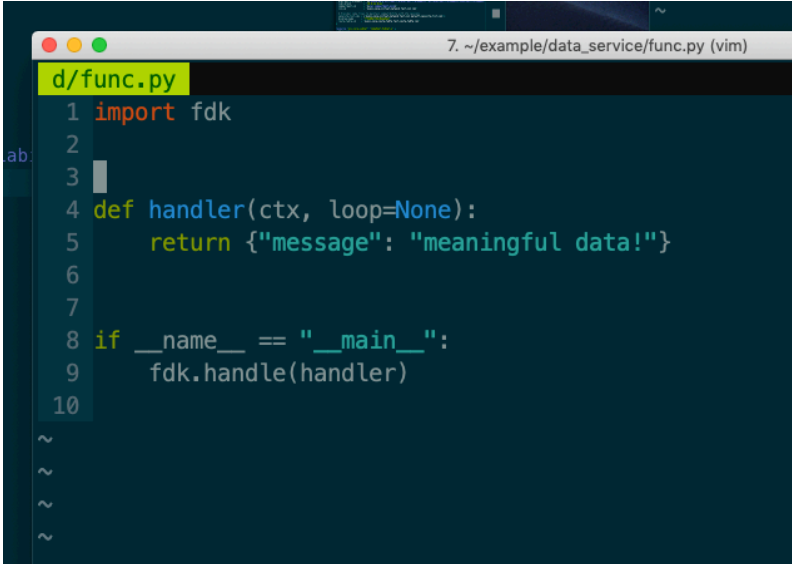
# Serverless as a Black Box

- It's about context
- Lean into the abstraction
- Serverless builds upon years of hard-learned systems lessons
- Write application code against a well-defined interface
- Complex systems concerns are pushed to the system



# On to Functions

- Ideally small, easy to reason about
- Implemented to work together
- Idempotency to maintain sanity
- All sounds familiar, surely...



```
7. ~/example/data_service/func.py (vim)
d/func.py
1 import fdk
2
3
4 def handler(ctx, loop=None):
5     return {"message": "meaningful data!"}
6
7
8 if __name__ == "__main__":
9     fdk.handle(handler)
10
~
~
~
~
```

# Single Responsibility Principal, 2003

“Every module, class, or function should have responsibility over a single part of the functionality provided by the software, and that responsibility should be entirely encapsulated by the class.”

- Robert C Martin (Uncle Bob), on SRP





# UNIX Philosophy, 1978

"Make each program do one thing well."

-Doug McIlroy, on the UNIX Philosophy



# Story Time

- I had to fix a bug once in a SCSI HBA driver
- This driver was implemented in 40k LOC
  - All in one source file
  - With very few functions
  - The few in place were thousands of lines long
  - It was hot garbage as a result
- First implemented about 15 years after Doug McIlroy's wonderful advice



# Serverless the Architecture

- Structures and components of a software system
- Practice of creating those components; how they are built
- Serverless the platform is an implementation detail of Serverless the architecture



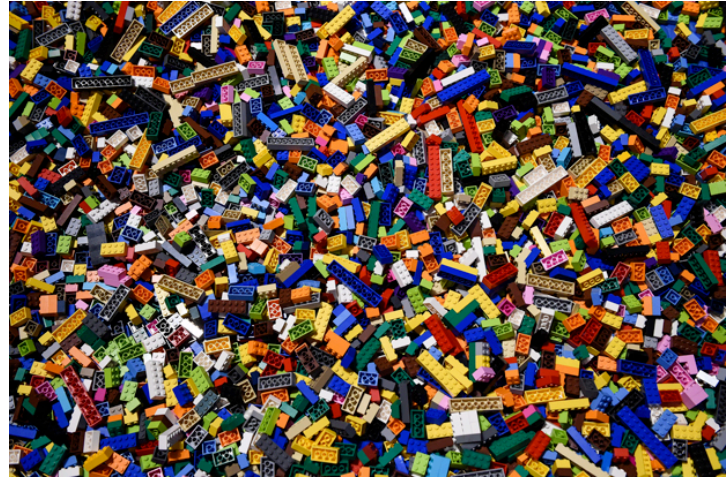
# Big Ideas

- Start small and compose solutions
- Don't own what you don't have to
- Plan ahead for observability
- Think ahead to migration cost



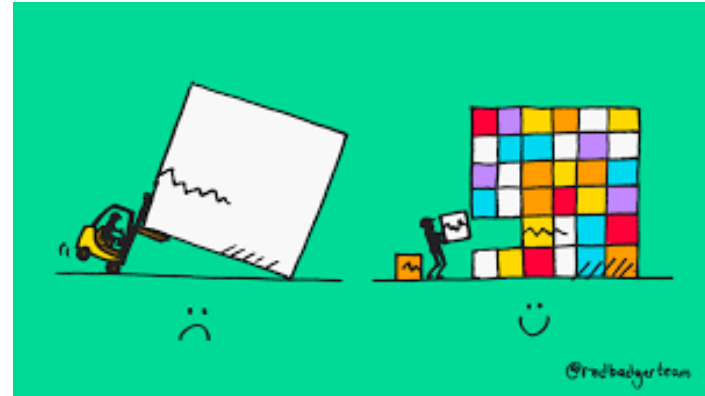
# Use Services and SaaS When Possible

- DBaaS
- Identity, Auth, Forms
- Storage Services
- Email, SMS
- Maps, GPS
- Media Streaming
- Chat and Chatbots



# Choose a Pattern that Helps you Minimize

- If a function does more than one discrete thing, break it up
- Better to proliferate than to decompensate
- Observability and triage become infinitely easier at the boundaries



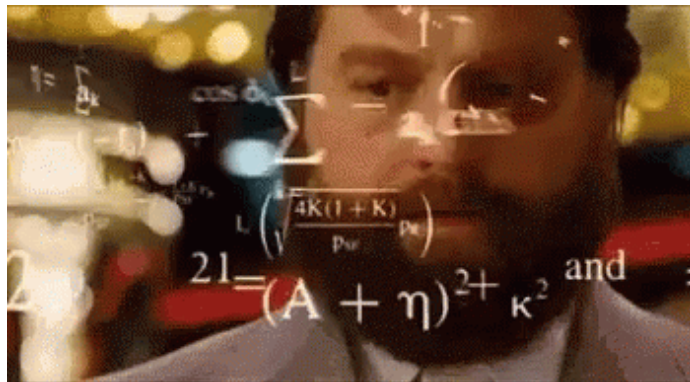
# Separate and Simplify

- Simplify your application estate
- Events can and probably should define application boundaries
- Share libraries between functions and applications, not execution context



# Platform and Practice Challenges

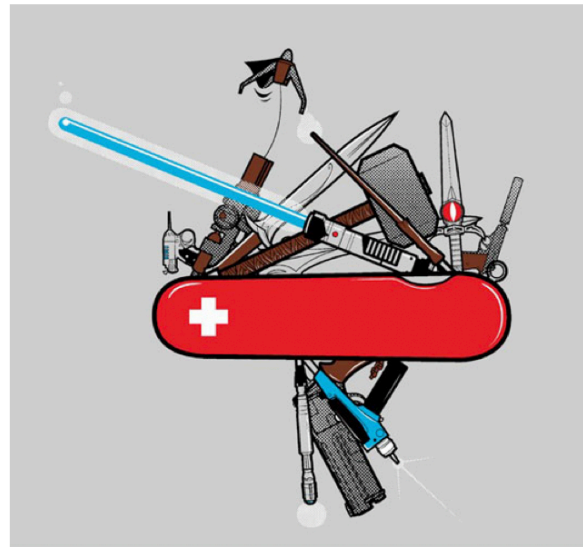
- How do we bring our current solutions into Serverless?
- CI/CD, Configuration Management, Observability are all table stakes
- This isn't as hard as it seems, if we embrace the abstraction





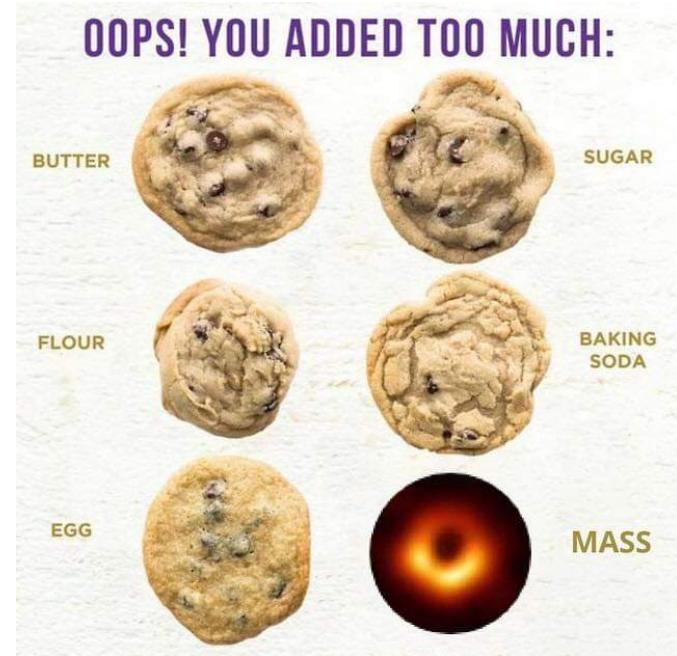
# Tooling and Pipelines

- Development workflow is largely unchanged. Do the things that work.
- As we've moved from bespoke compute environments, so too our automation has evolved
- Vendors fill in blanks, from managed services to gateway-type services



# Configuration Management

- We have established practices here, also largely unchanged
- Lean on managed services and aspects of the platform
- Do what you can at deployment to control your runtime environment



# Brownfield Adoption

- Cloud offers more than other people's computers and cables
- This is going to happen, regardless
- Far less moving parts with Serverless



# Security

- This becomes largely a vendor feature
- It's on you to not abuse the abstraction
- You don't need to know the details, but you still need to do the work



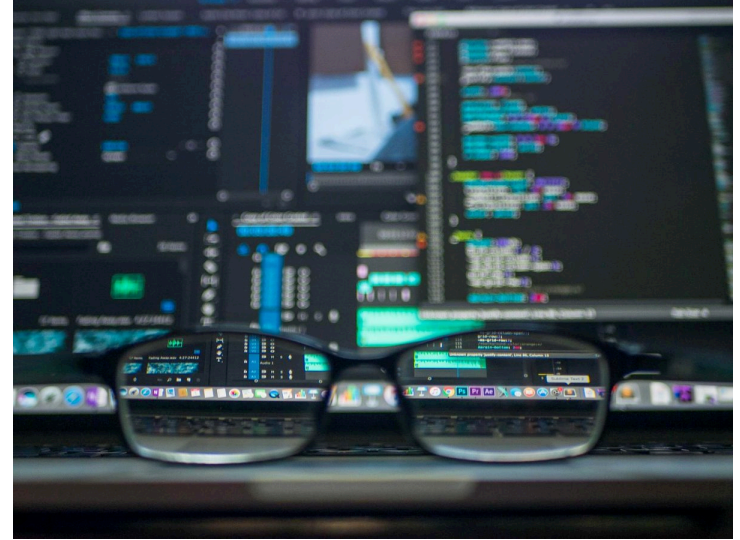
# Observability

- Metrics
  - Aggregate data regarding the behavior of a thing over time
- Tracing
  - Instrumentation which provides an instance of an action, traversing the entire stack
- Logging
  - Developer breadcrumbs we leave to give context for a certain code path



# Triaging Issues

- Monolith in a VM – log into the host, look at logs, run a debugger
- Containers in Kubernetes – Istio, Jaeger, Prometheus, Grafana, Falco
- Serverless is challenging. Logging is there, but that's not useful at scale
- Integrated solutions are needed



# Vendor L\*\*\*\_\*n

- A recent favorite punching bag...
- This is real, but is it a concern for you?
- Think migration, rather than lock-in



# Multi-cloud

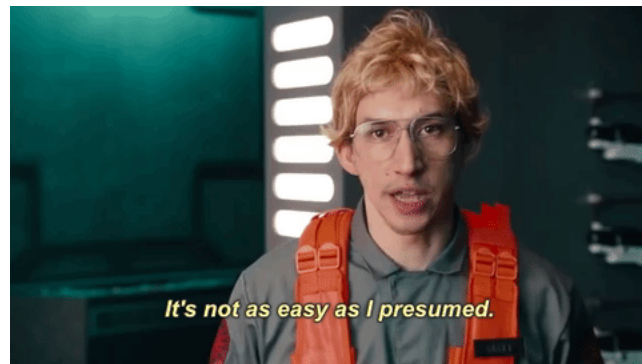
- Speaking of punching bags...
- Drop the notion of running an application across cloud platforms
- Think in terms of best-in-class options for a given application requirement
- Put as much wood behind one arrowhead as possible, within reason





# Is Serverless Simpler?

- Compared to what? Not usually.
- Serverless doesn't really mean less complexity. It may mean more.
- Serverless means an opportunity to focus your complexity where it matters
- Solving for business problems is almost always better than solving for infrastructure problems



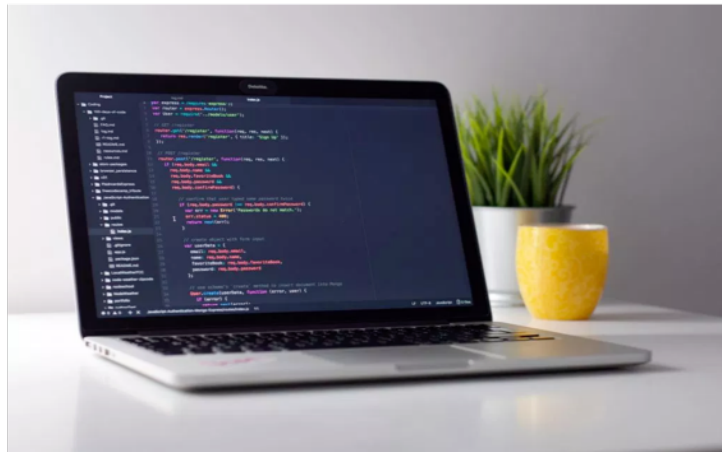
# Is Serverless Better?

- It can be! Let's not be religious...
- Less toil in deployment of systems is beneficial to focus
- OpEx reductions can be profound
- Tradeoff: we depend upon third parties to address issues as they arise
- Is this really a tradeoff?



# Final Thoughts

- Put Serverless on your radar
  - Greenfield
  - Brownfield migration
- Resist the urge to compare DevOps and Serverless. Apples to Apple Pie.
- It's not uncommon for a POC to roll into production
- Go build some stuff!





**ORACLE®**  
Cloud Native Labs

Thanks!

[cloud.oracle.com](https://cloud.oracle.com)

 @jlb13