# How Serverless Changes the IT Department

Paul Johnston
Opinionated Serverless Person
“Serverless all the things”

Medium/Twitter: @PaulDJohnston
Velocify Conf London 2018

# Paul Johnston

Experienced Interim CTO and Serverless Consultant

Environmentalist #ClimateChange bit.ly/2024wp

Twitter/Medium @PaulDJohnston

Ex-AWS Senior Developer Advocate for Serverless

Co-founder ServerlessDays (formerly JeffConf)

# Cloud, Data Centres and Energy

# Future of Cloud and Climate

Data Centres at least 2% of Global Carbon Emissions (bigger than aviation)

Growth of Cloud/Data Centres likely to grow by at least 5x in next 7 years

Efficiency is irrelevant due to Jevons Paradox - increased efficiency leads to increased demand

Not just climate, but energy security - energy price rises

Whitepaper written by Anne Currie and myself bit.ly/2024wp

# Ethics Whitepaper - The State of Data Centre Energy Use in 2018

| Cloud | Rating | Sustainable Servers? |
|---|---|---|
| Google | B+ | 100% with offsets |
| Azure | B | 100% with offsets and energy certificates |
| AWS | C | 100% with offsets in 5 regions, elsewhere unknown with estimates in <30-40% range |
| Oracle | C- | 100% with offsets in a few regions <30% overall |
| IBM | C- | ~50% overall |
| Alibaba | D- | Unknown but China a major market, and not known what energy is purchased |

# Background - Movivo

CTO of Movivo in 2015

One of the first Serverless startups

Android App + AWS Lambda

By early 2017 was in 20+ countries with over 500,000 MAU

AWS bill was ~$300/month (half data backup)

Team of 2 serverless and 2 android developers in total

# What *is* Serverless?

# Not FaaS*

*Not *just* FaaS

(although FaaS is important for most serverless implementations)

# Not -insert favourite FaaS platform here-*

*Again not *just* FaaS

# Serverless is Inherently

Event Driven

And

Asynchronous*

*Consequence of FaaS and Queues

# Definition of Serverless

> “A Serverless solution is one that costs you nothing to run if nobody is using it, excluding data storage.”

Me - September 2017

Serverless is
Economic
not Technological

So Serverless is about

Business

not

Technology*

*Consequence of CTO thinking

So why does this matter for an IT Department?

# 1) Organisations don’t like uncertainty

# 2) No such thing as a “standard IT Department”

# So how does serverless get in?

# Serverless entry point 1: Greenfield projects e.g. startups

# Serverless entry point 2:
# By stealth e.g. *that* lambda function

# Serverless will get in*

*it probably already is

In terms of actual resources...


Many many more resources

# Fewer lines of code…?

Fewer lines of executable code

So how do you manage it?

# Managing Serverless: Some Questions

How do you build a serverless team?

Do you manage tasks differently?

What are the key qualities of a serverless engineer?

# Managing Serverless: Putting it another way

What does it mean to use a Serverless approach

...for engineering

...and for the way you manage projects

...and teams?

# Is going serverless like herding cats?

# Answer?

“It’s almost exactly the same [as normal software development]” - Ben Kehoe

(Is this the end of my talk?)

# Managing Serverless Development...

...is not the same as managing normal software projects.

At least not completely the same (in my opinion)

There are some *special aspects*

# Because I’ve been a CTO...

Perspective is different

Pragmatist - keep it simple

Strategist - business directs tech

DON’T ASK ME TO CODE! (I can, but there are better people to do it)

# Best Practices for Development

# Best Practices

No-one is ever 100% using best practices (but that’s the aim)

These are some things you should be doing anyway…

# Best Practices (ok… Good Practices)

Daily Standups (can be virtual)

Pair Programming

Version Control

CI/CD (they are not the same thing!)

Testing

Code Reviews

etc… (not exhaustive)

# Serverless = Cloud 2.0

Serverless = Cloud 2.0

THE SECOND COMING OF CLOUD

# It’s about ~~No Maintenance~~ Automation

Avoiding maintenance comes out of Automation…

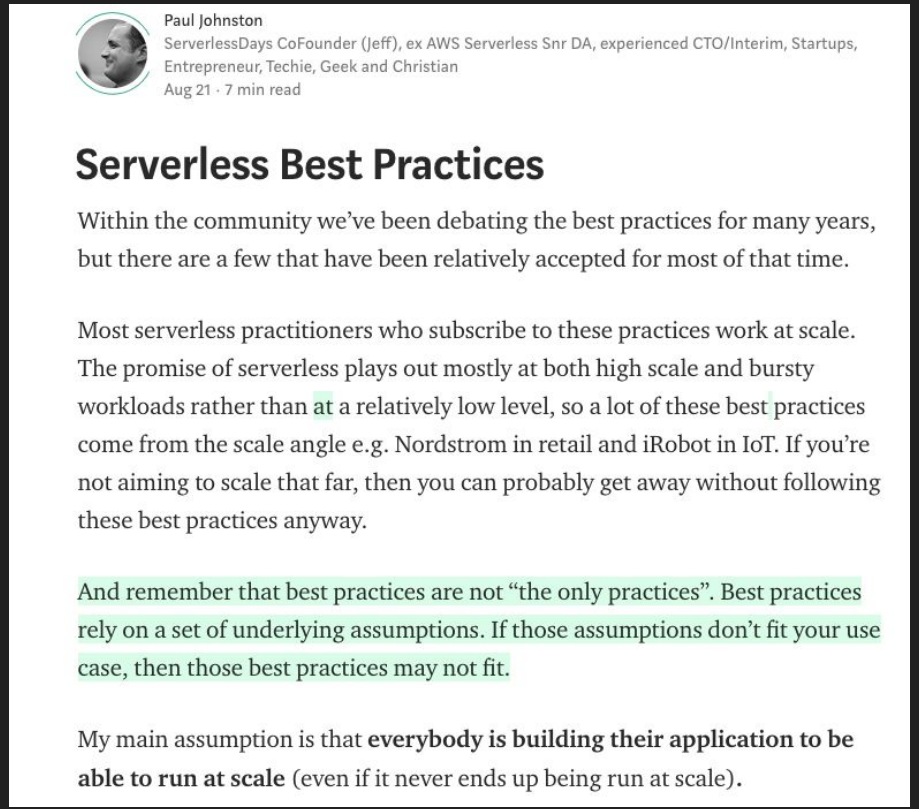
So Automate all the things

And remember to maintain your automation (because nobody else will)

# Serverless Best Practices

# Serverless Best Practices

Serverless Best Practices blog:
https://bit.ly/serverlessbestpractices `



Paul Johnston
ServerlessDays CoFounder (Jeff), ex AWS Serverless Snr DA, experienced CTO/Interim, Startups, Entrepreneur, Techie, Geek and Christian
Aug 21 · 7 min read

## Serverless Best Practices

Within the community we've been debating the best practices for many years, but there are a few that have been relatively accepted for most of that time.

Most serverless practitioners who subscribe to these practices work at scale. The promise of serverless plays out mostly at both high scale and bursty workloads rather than at a relatively low level, so a lot of these best practices come from the scale angle e.g. Nordstrom in retail and iRobot in IoT. If you're not aiming to scale that far, then you can probably get away without following these best practices anyway.

And remember that best practices are not "the only practices". Best practices rely on a set of underlying assumptions. If those assumptions don't fit your use case, then those best practices may not fit.

My main assumption is that **everybody is building their application to be able to run at scale** (even if it never ends up being run at scale).

# Serverless Best Practices

Each function should do only one thing

Functions don’t call other functions

Use as few libraries in your functions as possible (preferably zero)

Avoid using connections based services e.g. RDBMS

One function per route (if using HTTP)

Data flows not data lakes

Learn to use messages and queues (async FTW)

Just coding for scale is a mistake, you have to consider how it scales

# Serverless Solutions

What do these differences mean for

...Managing Teams?

...Individual Skillsets?

# FaaS

Each function is a high value unit

Functions != features

With FaaS/Serverless it’s Cloud Architecture not Software Architecture

# FaaS: Skillsets

Doer: Get it working

Tester: Just TDD/BDD please

Cloud First: Do as little as possible in code

Under Engineer-er: Don’t make it complicated, make it work - nobody cares if it’s perfect as the *quality impact* is far less important than in a monolith

# FaaS: Team

Kanban: Tasks and tiny code units work very well (Scrums + Sprints less so)

Discipline: Everyone on top of priorities

Pair Programming: This seems to work really well here. Tiny Code bases seems to enhance this

Feature Velocity: Good discipline/testing leads to high feature velocity (not code releases)

Principled Architecture: If you have clear architecture principles, you can move incredibly fast

# Events and Queues

Different Architecture for most coming from Web - not request-response

Asynchronous for most things by default

Distributed systems

Nuanced

# Events and Queues: Skillsets

Fast Learner: If new to this space

All Rounder: Cannot do FaaS without this. Not “someone else’s job”

Cloud First: Recognising what to use and when

Communicator: Explaining event driven systems is hard. Documenting/explaining is key

# Events and Queues: Team

Principled Architecture: Build up principles of how to use what type of queue/event and when (push, pull, volume etc)

Shared Knowledge: Communicate what you're doing individually, but sharing architecture understanding is key. Have a process (see Principled Architecture)

Constant Review: Often events and triggers already exist within a system. Recognise when to reuse and when to create new

# Services

Learn to use Services

Much less likely to “Roll your own” (containers, open source)

Already use many services so expand e.g. Analytics, Logging

# Services: Skillsets

Integrator: Learn to be an integrator of services. It’s key.

Pragmatist: Recognise that the service is probably built and run by somebody who knows more about this than you do

Cloud First: Recognising what to use and when

Under Engineer-er: Simply let others do the heavy lifting

# Services: Team

Pragmatic Approach: You can probably build 80% of services you use, but why would you?

Document: Why do you use the service? What is missing?

Support: Do not let one person be the point of contact for a service you use. Share the knowledge

Security: Learn best practices over api keys and shared secrets (outside the repo is rule #1)

# Infrastructure as Code

Most important aspect

Almost impossible to do Serverless at even moderate scale without it

Terraform, SAM, CloudFormation, Architect, stdlib, Serverless Framework, etc

# Infrastructure as Code: Skillsets

Automater: Every engineer should be an automater as a matter of principle (bash, scripts, CI/CD etc). Every release should automate more

All Rounder: Dev and Ops distinctions disappear. Everybody is everything. True (non-siloed) Engineering at that point

Doer: Role << Responsibility

Fast Learner: New services released almost daily. Need to be able to add in.

# Infrastructure as Code: Team

Automation: Most important automation.

Collaborative Deployment: Serverless tends towards much high number of deployments. Need great communication skills and understanding of canarying, blue/green deployment etc.

Feature Velocity: It’s only a feature when it’s deployed. Code != Features. Uncoupled Logic

Team Scale: You cannot achieve team scale without good Infrastructure as Code and CI/CD

# Impact: Hiring

Much broader skillset

More “Engineering” than “Developing”

Interviewing: Code Challenges? How do you interview? Primary skills are actually collaboration and ability to learn fast.

Onboarding: ...is hard - Nordstrom. May be easier to hire juniors and upskill than hire seniors and reskill.

# Impact: Upskilling

Much broader skillsets

Learning resources are fewer and relatively limited by use case - AWS, A Cloud Guru etc

Ingrained thinking may be problematic

More junior = better?

Senior leadership still needed but more as mentors (feature velocity)

# Impact: Remove Naysayers

Naysayers can be time drains

Remove or Repurpose

Velocity of team is such that this can significantly decrease productivity

# Impact: Roles << Responsibilities

Jobs will change over time

Within serverless engineering Dev and Ops merge, as does Sec and everything else

“Cloud Native Engineers”

# How does Serverless change the IT Department?

Serverless is not the future, it’s already here

Your IT Department will change

# How does Serverless change the IT Department?

Fewer 10x-ers, more mentors

Increases communication

Removes Silos - everyone is an Engineer (no Dev and Ops silos)

Enforces Infrastructure as Code

People have Responsibilities not Roles

Emphasis on Automation and Deployment

Much higher Feature Velocity

Decreased reliance on code perfection

Serverless Engineer

=

Cloud Native Engineer

(aka Pragmatic Hacker)

Serverless Team

=

Rapidly Collaborate +
Automate + Document

Serverless Team

=

Slack + Automate + Document

# How Serverless Changes the IT Department

Paul Johnston
“Serverless all the things”

Serverless Best Practices blog: https://bit.ly/serverlessbestpractices

Medium/Twitter: @PaulDJohnston
Velocify Conf London 2018