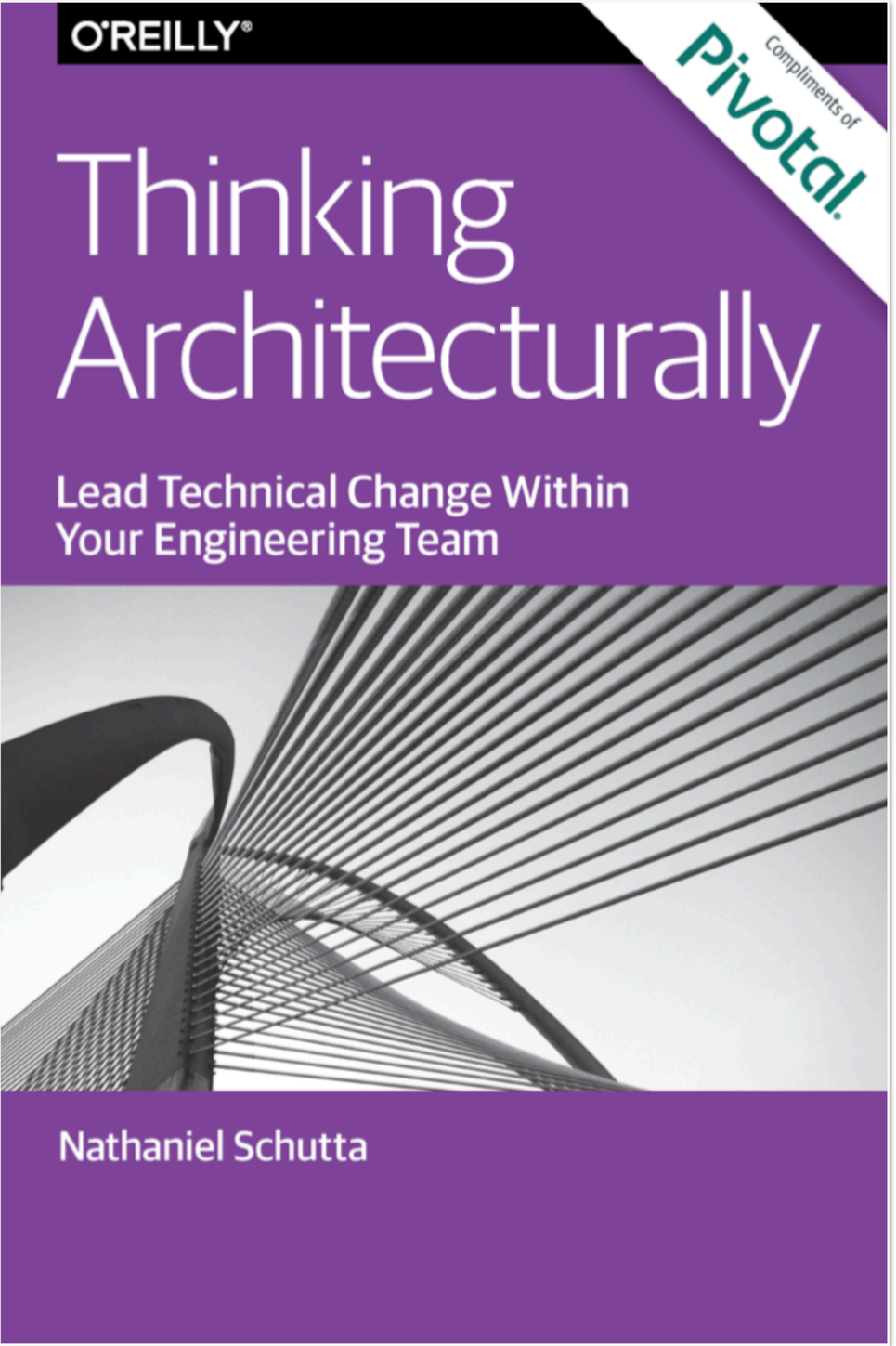

EVOLVING TO
CLOUD NATIVE
NATHANIEL SCHUTTA
@NTSCHUTTA
NTSCHUTTA.IO



https://content.pivotal.io/ebooks/thinking-architecturally

Ah “the cloud!”

So. Many. Options.

Microservices. Modular monoliths.

Container all the things?

What about serverless?

Functions. As a Service.

Did someone say Polycloud?

https://www.thoughtworks.com/radar/techniques/polycloud

How do we make sense of all this?!?

There are real engineering issues to overcome.

Many believe in magic sparkle ponies...

How do we avoid pitfalls?

And a strong case of resume driven design?

WHAT IS CLOUD NATIVE?

https://mobile.twitter.com/as_w/status/1090763452241534976

Applications designed to take advantage of cloud computing.

Fundamentally about how we create and deploy applications.

Cloud computing gives us some very interesting abilities.

Scale up. Scale down. On demand.

Limitless compute.*

* Additional fees may apply.

Cloud native isn’t just an architectural pattern.

Combination of practices, techniques, technologies.

Agile development.

Continuous delivery.

Automation.

Containers.

Microservices.

Functions.

Changes our culture.

DevOps.

Infrastructure is a different game today isn’t it?

We’ve seen this massive shift.

Servers used to be home grown.

Bespoke. Artisanal.

Spent days hand crafting them.

Treated them like pets…

Did whatever it took to keep them healthy and happy.

Servers were a heavily constrained resource.

They were really expensive!

Had to get our money’s worth…

Thus was born app servers.

Put as many apps as possible on a server.

Maximize the return on investment.

But that has some unintended side effects.

Shared resources.

One application’s bug could take down multiple apps.

Coordinating changes hurts.

“Your app can’t get this feature until all other apps are ready.”

Currency === 18 months of freezes, testing, frustration.

Organizations ignored currency issues…pain wasn’t “worth it”.

“Fear is the path to the dark side. Fear leads to anger. Anger leads to hate. Hate leads to suffering.”

–Yoda

#YodaOps

Move code from one server to another...

Worked in dev…but not test.

Why?!?

The environments are the same…right?

“Patches were applied in a different order…”

Can I change careers?

Things started to change.

Servers became commodities.

Linux and Intel chips replaced custom OS on specialized silicon.

https://mobile.twitter.com/linux/status/936877536780283905?lang=en

Prices dropped.

Servers were no longer the constraining factor.

People costs eclipsed hardware costs.

Heroku, AWS, Google App Egine, Cloud Foundry, Azure.

Shared servers became a liability.

Treat them like cattle…when they get sick, get a new one.


15

New abstractions.

Containers and PaaS changed the game.

Package the app up with everything it needs.

Move *that* to a different environment.

Works in dev? You’re testing the exact same thing in test.

So. Much. Win.

Your app needs a spiffy new library? Go ahead!

It doesn’t impact any other app because you are isolated.

Moves the value line.

Less “undifferentiated heavy lifting”.

https://mobile.twitter.com/onsijoe/status/598235841635360768?lang=en

Changes development.

Always be changing.

Run experiments. A/B testing.

Respond to business changes.

Deliver in days not months.

https://mobile.twitter.com/ntschutta/status/938109379995353088

Speed matters.

Disruption impacts every business.

Your industry is not immune.

Amazon Prime customers can order from Whole Foods.

Some insurance companies view Google as a competitor.

We’re all technology companies today.


12 FACTORS

Twelve Factor App.

https://12factor.net

Characteristics shared by successful apps.

At least at Heroku.

I. One codebase in version control, multiple deploys.

Version control isn’t controversial. Right?!?

Sharing code? It better be in a library then…

II. Explicitly define your dependencies.

Do not rely on something just “being there” on the server.

If you need it, declare it.

III. Configuration must be separate from the code.

The things that vary from environment to environment.

Could you open source that app right now?

IV. Backing services are just attached resources.

Should be trivial to swap out a local database for a test db.

In other words, loose coupling.

V. Build, release, run.

Deployment pipeline anyone?

Build the executable…

Deploy the executable with the proper configuration…

Launch the executable in a given environment.

VI. Stateless - share nothing.

https://mobile.twitter.com/stuarthalloway/status/1134806008528809985

State must be stored via some kind of backing service.

In other words, you cannot rely on the filesystem or memory.

Recovery. Scaling.

VII. Export services via port binding.

App exports a port, listens for incoming requests.

Localhost for development,
load balancer for public facing.

VIII. Scale via process.

In other words, scale horizontally.

IX. Start up fast, shut down gracefully.

Processes aren’t pets,
they are disposable.

Processes can be started (or stopped) quickly and easily.

Ideally, start up is seconds.

Also can handle
unexpected terminations!

X. Dev/Prod parity.

From commit to production should be hours…maybe days.

Definitely not weeks.

Developers should be involved in deploys and prod ops.

Regions should be identical. Or as close as possible to identical.

Backing services should be the same in dev and prod.

Using one DB in dev and another in prod invites pain.

XI. Logs as event streams.

Don’t write logs to the filesystem!

It won’t be there later…

Write to stdout.

Stream can be routed any number of places.

And then consumed via a wide variety of tools.

XII. Admin tasks run as one off processes.

Database migrations for instance.

REPL for the win.

Run in an identical environment to the long running processes.

Your legacy apps will violate some factors.

Maybe all 12!

In general...

II. Explicitly define your dependencies.

Probably one of the harder ones to satisfy.

Do we really need this library?

“It works, don’t touch it.”

III. Configuration must be separate from the code.

Many an app has hardcoded credentials.

Hardcoded database connections.

VI. Stateless - share nothing.

Also can be challenging.

Many apps were designed around a specific flow.

Page 2 left debris for Page 3!

“Just stash that in session”.

IX. Start up fast, shut down gracefully.

Many apps take way too long to start up…

Impacts health checks.

X. Dev/Prod parity.

Environments should be consistent!

Shorten code to prod cycle.

“It worked in test…”

Do your applications have to be fully 12 factor compliant?

Nope.

Is it a good goal?

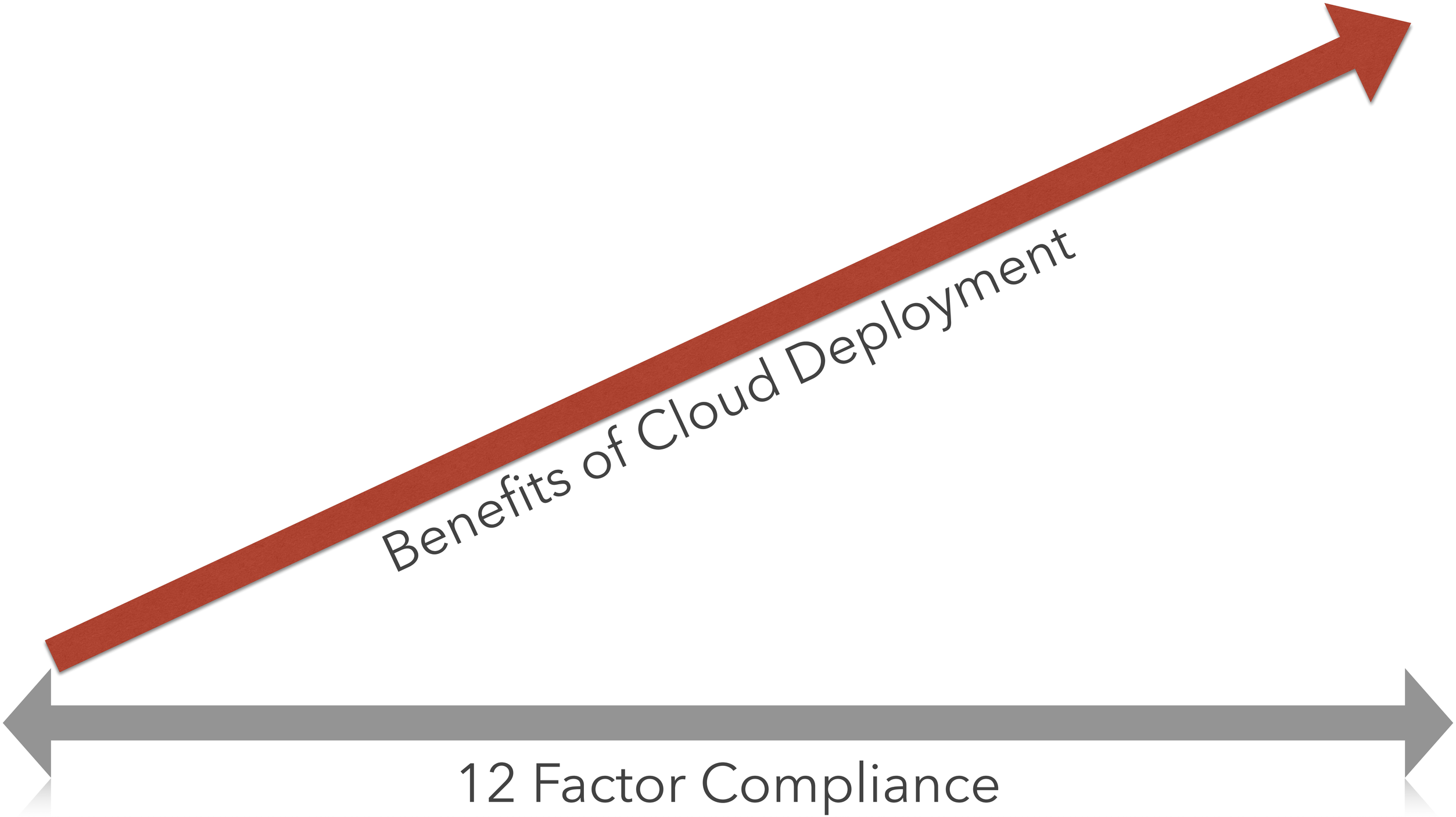
Sure.

But be pragmatic.

Certain attributes lessen the advantages of cloud.

Long startup time hurts elastic scaling & self healing.

Think of it as a continuum.


Benefits of Cloud Deployment
12 Factor Compliance

Developers also talk about 15 factor apps.

aka Beyond the Twelve-Factor App.

https://content.pivotal.io/blog/beyond-the-twelve-factor-app

However you define it…

To maximize what
the cloud gives us…

Applications need to be designed properly.

Legacy applications will fall short.

Opportunistically refactor!

Building greenfield?

Go cloud native!

Don’t build legacy.


CONTAINERS

Docker.

https://www.docker.com

Docker is, well a box.

Put whatever you want in it.

Includes everything needed to run.

Your code, system libraries, executables, settings, etc.

Your code is isolated
from the environment.

How often does your staging app server actually match production?

Instead of copying the code from one environment to another...

Just define the entire environment.

Similar to virtual machines.

Containers virtualize at the OS level, not the hardware level.

You have to tell Docker what you need.

Create a Docker image which is often stored in a registry.

Filters

Docker Certified

- [ ] Docker Certified

Images

- [ ] Verified Publisher
  Docker Certified And Verified Publisher Content
- [ ] Official Images
  Official Images Published By Docker

Categories

- [ ] Analytics
- [ ] Application Frameworks
- [ ] Application Infrastructure
- [ ] Application Services
- [ ] Base Images
- [ ] Databases
- [ ] DevOps Tools
- [ ] Featured Images
- [ ] Messaging Services
- [ ] Monitoring
- [ ] Operating Systems
- [ ] Programming Languages
- [ ] Security
- [ ] Storage

Operating Systems

- [ ] Linux

1 - 25 of 2,016,820 available images.

Most Popular

VERIFIED PUBLISHER

Oracle Database Enterprise Edition DOCKER CERTIFIED

By Oracle • Updated a year ago

Oracle Database 12c Enterprise Edition

Container | Docker Certified | Linux | x86-64 | Databases

VERIFIED PUBLISHER

Oracle Java 8 SE (Server JRE) DOCKER CERTIFIED

By Oracle • Updated 2 months ago

Oracle Java 8 SE (Server JRE)

Container | Docker Certified | Linux | x86-64 | Programming Languages

VERIFIED PUBLISHER

MySQL Server Enterprise Edition DOCKER CERTIFIED

By Oracle • Updated 3 months ago

The world's most popular open source database system

Container | Docker Certified | Linux | x86-64 | Databases

VERIFIED PUBLISHER

Oracle WebLogic Server DOCKER CERTIFIED

By Oracle • Updated a month ago

Oracle WebLogic Server

Container | Docker Certified | Linux | x86-64 | Application Frameworks | Application Infrastructure

Docker Hub is community driven.

It isn’t vetted. Or curated.

Anyone can push anything,
anyone can pull anything.

Not entirely the wild west though.

There are Verified Publisher images.

Filters (1) Clear All

Docker Certified

- [ ] Docker Certified

Images

- [x] Verified Publisher
  Docker Certified And Verified Publisher Content
- [ ] Official Images
  Official Images Published By Docker

Categories

- [ ] Analytics
- [ ] Application Frameworks
- [ ] Application Infrastructure
- [ ] Application Services
- [ ] Base Images
- [ ] Databases
- [ ] DevOps Tools
- [ ] Featured Images
- [ ] Messaging Services
- [ ] Monitoring
- [ ] Operating Systems
- [ ] Programming Languages
- [ ] Security
- [ ] Storage

Operating Systems

- [ ] Linux
- [ ] Windows

Architectures

- [ ] ARM
- [ ] ARM 64

1 - 25 of 206 available images.

Most Popular

Publisher Content

VERIFIED PUBLISHER

Oracle Database Enterprise Edition DOCKER CERTIFIED

By Oracle • Updated a year ago

Oracle Database 12c Enterprise Edition

Container | Docker Certified | Linux | x86-64 | Databases

VERIFIED PUBLISHER

Oracle Java 8 SE (Server JRE) DOCKER CERTIFIED

By Oracle • Updated 2 months ago

Oracle Java 8 SE (Server JRE)

Container | Docker Certified | Linux | x86-64 | Programming Languages

VERIFIED PUBLISHER

MySQL Server Enterprise Edition DOCKER CERTIFIED

By Oracle • Updated 3 months ago

The world's most popular open source database system

Container | Docker Certified | Linux | x86-64 | Databases

VERIFIED PUBLISHER

Oracle WebLogic Server DOCKER CERTIFIED

By Oracle • Updated a month ago

Oracle WebLogic Server

Container | Docker Certified | Linux | x86-64 | Application Frameworks | Application Infrastructure

VERIFIED PUBLISHER

Db2 Developer-C Edition DOCKER CERTIFIED

By IBM • Updated 3 months ago

Full feature, free version for non-production environments. Ideal for developers.

Commercial entities.

Approved by Docker.

Images vetted by Docker.

Images are supported
by said entities.

And some cost money. Imagine that.

Some images are Docker Certified.

Filters (1) Clear All

Docker Certified

- [x] Docker Certified

Images

- [ ] Verified Publisher
  Docker Certified And Verified Publisher Content
- [ ] Official Images
  Official Images Published By Docker

Categories

- [ ] Analytics
- [ ] Application Frameworks
- [ ] Application Infrastructure
- [ ] Application Services
- [ ] Base Images
- [ ] Databases
- [ ] DevOps Tools
- [ ] Featured Images
- [ ] Messaging Services
- [ ] Monitoring
- [ ] Operating Systems
- [ ] Programming Languages
- [ ] Security
- [ ] Storage

Operating Systems

- [ ] Linux
- [ ] Windows

Architectures

- [ ] ARM
- [ ] ARM 64

1 - 25 of 41 available images.

Most Popular

× Docker Certified

VERIFIED PUBLISHER

Oracle Database Enterprise Edition DOCKER CERTIFIED

By Oracle • Updated a year ago

Oracle Database 12c Enterprise Edition

Container | Docker Certified | Linux | x86-64 | Databases

VERIFIED PUBLISHER

Oracle Java 8 SE (Server JRE) DOCKER CERTIFIED

By Oracle • Updated 2 months ago

Oracle Java 8 SE (Server JRE)

Container | Docker Certified | Linux | x86-64 | Programming Languages

VERIFIED PUBLISHER

MySQL Server Enterprise Edition DOCKER CERTIFIED

By Oracle • Updated 3 months ago

The world's most popular open source database system

Container | Docker Certified | Linux | x86-64 | Databases

VERIFIED PUBLISHER

Oracle WebLogic Server DOCKER CERTIFIED

By Oracle • Updated a month ago

Oracle WebLogic Server

Container | Docker Certified | Linux | x86-64 | Application Frameworks | Application Infrastructure

VERIFIED PUBLISHER

Db2 Developer-C Edition DOCKER CERTIFIED

By IBM • Updated 3 months ago

Full feature, free version for non-production environments. Ideal for developers.

Some overlap with the Verified Publisher tag…

Those images have passed certain tests, follow best practices.

Also subject to a vulnerability scan.

There are also Official Images.

Filters (1) Clear All

Docker Certified

- [ ] Docker Certified

Images

- [ ] Verified Publisher
  Docker Certified And Verified Publisher Content
- [x] Official Images
  Official Images Published By Docker

Categories

- [ ] Analytics
- [ ] Application Frameworks
- [ ] Application Infrastructure
- [ ] Application Services
- [ ] Base Images
- [ ] Databases
- [ ] DevOps Tools
- [ ] Featured Images
- [ ] Messaging Services
- [ ] Monitoring
- [ ] Operating Systems
- [ ] Programming Languages
- [ ] Security
- [ ] Storage

Operating Systems

- [ ] Linux
- [ ] Windows

Architectures

- [ ] ARM
- [ ] ARM 64

1 - 25 of 151 available images.

Most Popular

Official Image

OFFICIAL IMAGE

couchbase

Updated 29 minutes ago

10M+ Downloads 374 Stars

Couchbase Server is a NoSQL document database with a distributed architecture.

Container Linux x86-64 Storage Application Frameworks

OFFICIAL IMAGE

redis

Updated 30 minutes ago

10M+ Downloads 6.6K Stars

Redis is an open source key-value store that functions as a data structure server.

Container Linux Windows ARM 64 ARM x86-64 386 IBM Z PowerPC 64 LE Databases

OFFICIAL IMAGE

mongo

Updated 30 minutes ago

10M+ Downloads 5.6K Stars

MongoDB document databases provide high availability and easy scalability.

Container Windows Linux ARM 64 x86-64 Databases

OFFICIAL IMAGE

ubuntu

Updated 30 minutes ago

10M+ Downloads 9.2K Stars

Ubuntu is a Debian-based Linux operating system based on free software.

Container Linux ARM x86-64 IBM Z PowerPC 64 LE 386 ARM 64 Base Images Operating Systems

OFFICIAL IMAGE

alpine

Updated 30 minutes ago

10M+ Downloads 5.0K Stars

A minimal Docker image based on Alpine Linux with a complete package index and only 5 MB in size!

Published by Docker.

Curated, documented, tested.

Follow best practices,
vulnerability scanned.

Docker Hub is based on the Docker Public Registry.

Not the only option.

Many companies will have their own internal registry.

What do you mean by “image”?

An image is a template or recipe for building a container.

Images often customize existing images.

Parent image acts as a starting point for further customization.

But again, *you* are responsible for what goes in the box!

“With great power comes great responsibility.”

–Uncle Ben

Don’t forget to stay current…

BUSINESS

SEP 14 2017, 3:21 PM ET

Equifax Hackers Exploited Months-Old Flaw

by BEN POPKEN

Equifax announced late Wednesday that the source of the hole in its defenses that enabled hackers to plunder its databases was a massive server bug first revealed in March.

For the rest of the IT world, fixing that flaw was a "hair on fire moment," a security expert said, as companies raced to install patches and secure their servers. But at Equifax, criminals were able to pilfer data from mid-May to July, when the credit bureau says it finally stopped the intrusion.

Equifax, Software Company Blame Each Other for Security Breach 1:52

"We know that criminals exploited a U.S. website application vulnerability," Equifax said in an update on its website Wednesday night. "The vulnerability was Apache Struts CVE-2017-5638." Equifax said it was working with a leading cybersecurity firm, reported to be Mandiant, to investigate the breach. Mandiant declined an NBC News request for comment.

Related: The One Move to Make After Equifax Breach

The Apache Software Foundation, which oversees the Apache Struts project, said in a press release Thursday that a software update to patch the flaw was

Most of the Fortune 100 still use flawed software that led to the Equifax breach

Zack Whittaker
@zackwhittaker / 1 week ago

Almost two years after Equifax's massive hack, the majority of Fortune 100 companies still aren't learning the lessons of using vulnerable software.

In the last six months of 2018, two-thirds of the Fortune 100 companies downloaded a vulnerable version of Apache Struts, the same vulnerable server software that was used by hackers to steal the personal data on close to 150 million consumers, according to data shared by Sonatype, an open-source automation firm.

That's despite almost two years' worth of patched Struts versions being released since the attack.

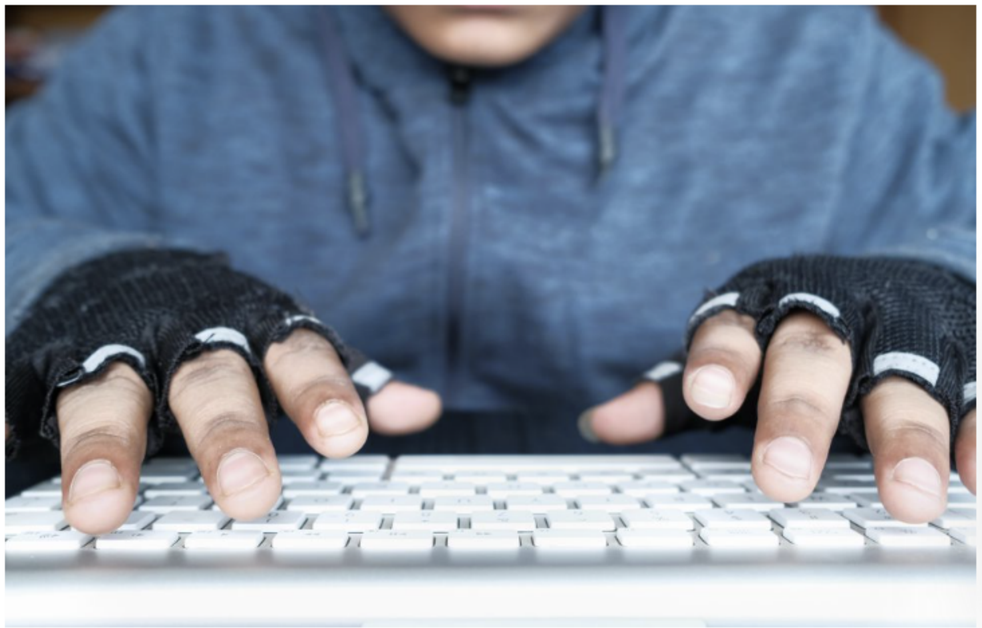
Sonatype wouldn't name the Fortune 100 firms that had downloaded the

SECURITY / LEER EN ESPAÑOL

Exactis said to have exposed 340 million records, more than Equifax breach

We hadn't heard of the firm either, but it had data on hundreds of millions of Americans and businesses and leaked it, according to Wired.

BY ABRAR AL-HEETI / JUNE 28, 2018 10:14 AM PDT



Worst hacks of the year 00:00 / 03:24

Technology

Marriott hack hits 500 million Starwood guests

30 November 2018 Share

Sheraton is one of Marriott's brands

The records of 500 million customers of the hotel group Marriott International have been involved in a data breach.

The hotel chain said the guest reservation database of its Starwood division had been compromised by an unauthorised party.

It said an internal investigation found an attacker had been able to access the

Top Stories

Tabloid's owner defends Jeff Bezos report
AMI, owner of a US magazine accused of blackmail by Amazon's founder, says it acted in good faith.
40 minutes ago

What US ruling may mean for Roe v Wade
2 hours ago

Russia probe chief grilled by lawmakers
24 minutes ago

Features

Meltdown and Spectre

Vulnerabilities in modern computers leak passwords and sensitive data.

Meltdown and Spectre exploit critical vulnerabilities in modern processors. These hardware vulnerabilities allow programs to steal data which is currently processed on the computer. While programs are typically not permitted to read data from other programs, a malicious program can exploit Meltdown and Spectre to get hold of secrets stored in the memory of other running programs. This might include your passwords stored in a password manager or browser, your personal photos, emails, instant messages and even business-critical documents.

Meltdown and Spectre work on personal computers, mobile devices, and in the cloud. Depending on the cloud provider's infrastructure, it might be possible to steal data from other customers.

Meltdown

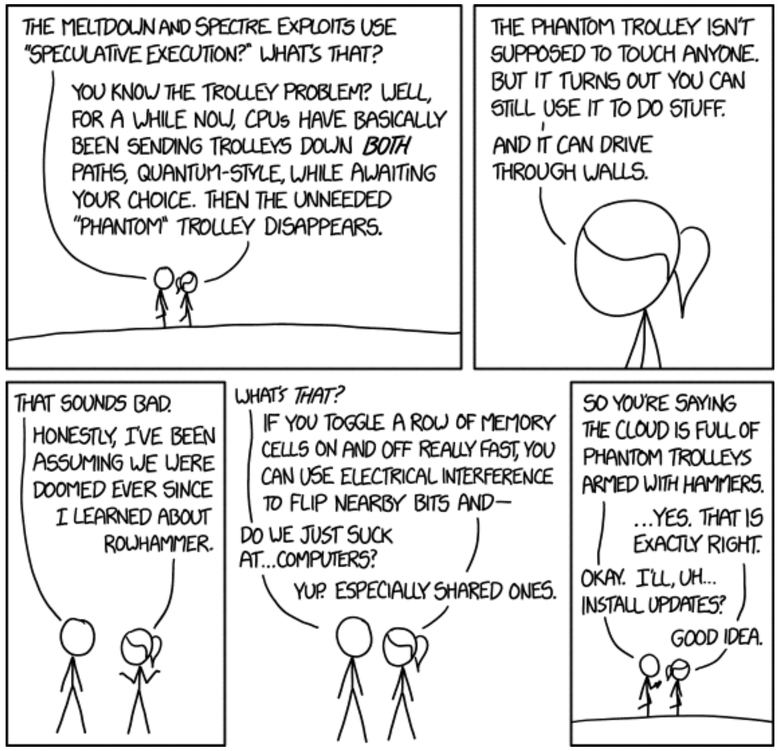
Meltdown breaks the most fundamental isolation between user applications and the operating system. This attack allows a program to access the memory, and thus also the secrets, of other programs and the operating system.

If your computer has a vulnerable processor and runs an unpatched operating system, it is not safe to work with

Spectre

Spectre breaks the isolation between different applications. It allows an attacker to trick error-free programs, which follow best practices, into leaking their secrets. In fact, the safety checks of said best practices actually increase the attack surface and may make applications more susceptible to Spectre

Spectre is harder to exploit than Meltdown, but it is also



Randall Munroe

https://xkcd.com/1938/

Of course today, you can’t talk about containers without mentioning…

Kubernetes.

https://kubernetes.io

K8s - deploy, scale & manage containers.

Built from lessons learned at Google.

They have experience.

Has many of the features you would expect.

Self-healing. Scaling. Service discovery. Load balancing.

Automated rollouts and rollbacks.

kubectl

kube control, kube CTL, kube cuddle, kube cuttle….

K8s command line interface.

You describe your desired state, Kubernetes makes it so.

(insert your favorite Captain Picard reference here).

Number of abstractions: pod, service, volume, namespace…

Pod houses one (or more) containers.

Also storage, unique IPs.

Pod is fundamental unit of deployment.

An instance of an application.

Typically Docker containers but K8s supports others.

1 container per pod is common, but some containers are tightly coupled.

Pods can contain multiple containers that then share resources.

“Helper” containers are often referred to as sidecars.

All the public cloud vendors have a k8s option.

Google Kubernetes Engine,
Amazon EKS, Amazon Fargate…

Kubernetes on Azure, PKS.

No shortage of options.

https://mobile.twitter.com/kelseyhightower/status/1099726594808168453

https://twitter.com/danveloper/status/1114874890170109952

https://mobile.twitter.com/agmsbush/status/1092990028148498432

https://twitter.com/krisnova/status/1114007653011689472

https://twitter.com/rettori/status/1110166982408785920

https://mobile.twitter.com/kelseyhightower/status/1102581558207176711

https://twitter.com/kelseyhightower/status/1105135408369684481

Do *not* underestimate the challenge of running k8s.

It is not simple!

http://searchitoperations.techtarget.com/news/450431623/Kubernetes-release-adds-maturity-in-19-but-devils-in-the-details

But it gives us another set of primitives to work with.


MICROSERVICES

Reaction to monoliths and heavy weight services.

As well as cloud environments.

Monoliths hurt.

Developer productivity takes a hit.

Hard to get your head wrapped around a huge code base.

Long ramp up times
for new developers.

Small change results in building and deploying everything.

Scaling means scaling the entire application!

Not just the part that needs more capacity.

Hard to evolve.

We’re all familiar with the second law of thermodynamics…

Otherwise known as a teenagers bedroom.

The universe really wants to be disordered.

Software is not immune
from these forces!

Modularity tends to break down over time.

Over time, takes longer to add new functionality.

Frustration has given birth to a “new” architectural style.

Enter the microservice.

No “one” definition.

In the eye of the beholder…

https://mobile.twitter.com/littleidea/status/500005289241108480

Anything that can be rewritten two weeks or less.

Think in terms of characteristics.

Suite of small, focussed services.

Do one thing, do it well.

Linux like - pipe simple things together to get complex results.

Independently deployable.

Independently scalable.

Evolve at different rates.

Freedom to choose the right tech for the job.

Built around business capabilities.

High cohesion, low coupling…

Applied to services.

It is just another approach. An architectural style. A pattern.

Despite what some developers may have said.

Use them wisely.

Please Microservice Responsibly.

https://content.pivotal.io/blog/should-that-be-a-microservice-keep-these-six-factors-in-mind

“If you can't build a monolith, what makes you think microservices are the answer?”

–Simon Brown

http://www.codingthearchitecture.com/2014/07/06/distributed_big_balls_of_mud.html

Sometimes the right answer is a modular monolith…

https://www.youtube.com/watch?v=kbKxmEeuvc4


SERVERLESS

From IaaS to CaaS to PaaS…

What about serverless?

Functions.

As a Service.

I hear that is *the* in thing now.

But we just refactored to cloud native microservices…

Don’t throw that code away just yet!

Fair to say FaaS is a subset of serverless.

Though many use the terms interchangeably.

First things first. There are still servers.

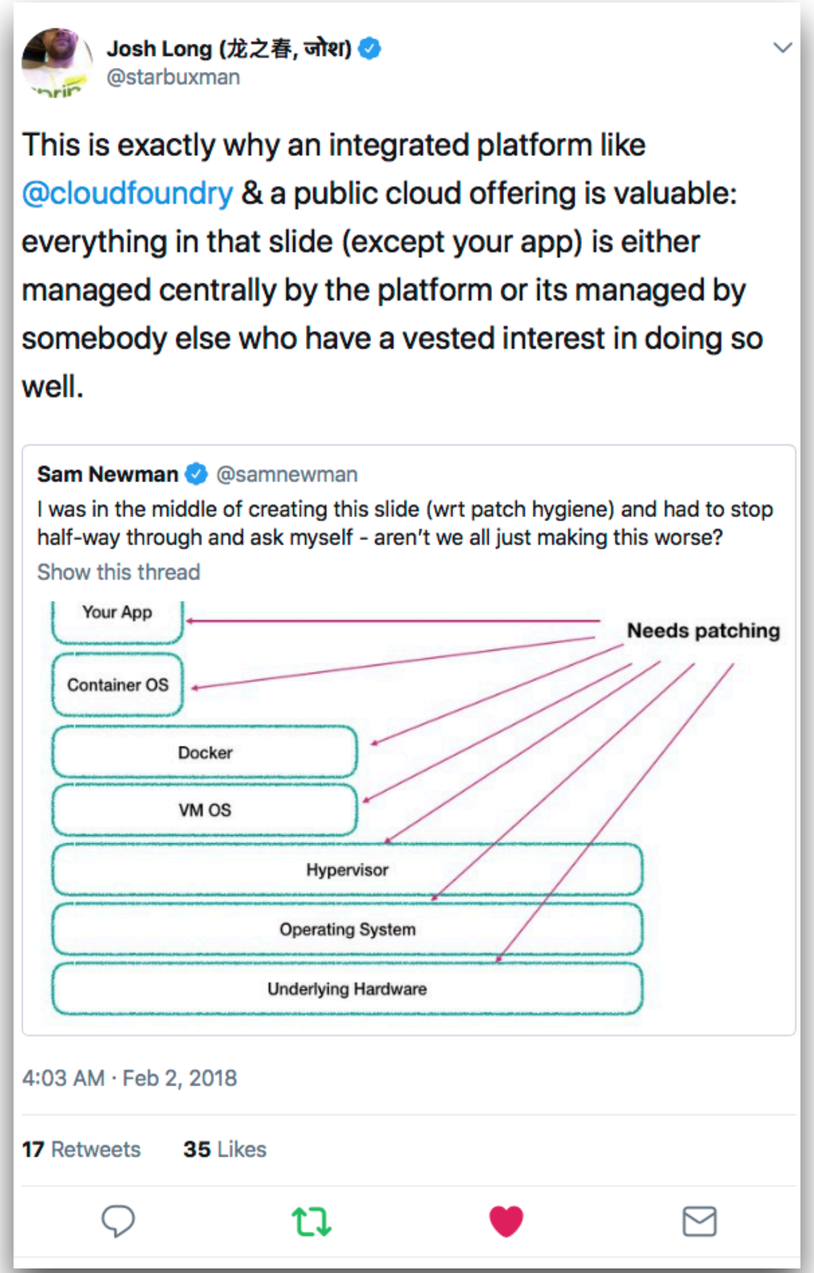
We are just (further)
abstracted away from them.

We don’t have to spend time provisioning, updating, scaling…

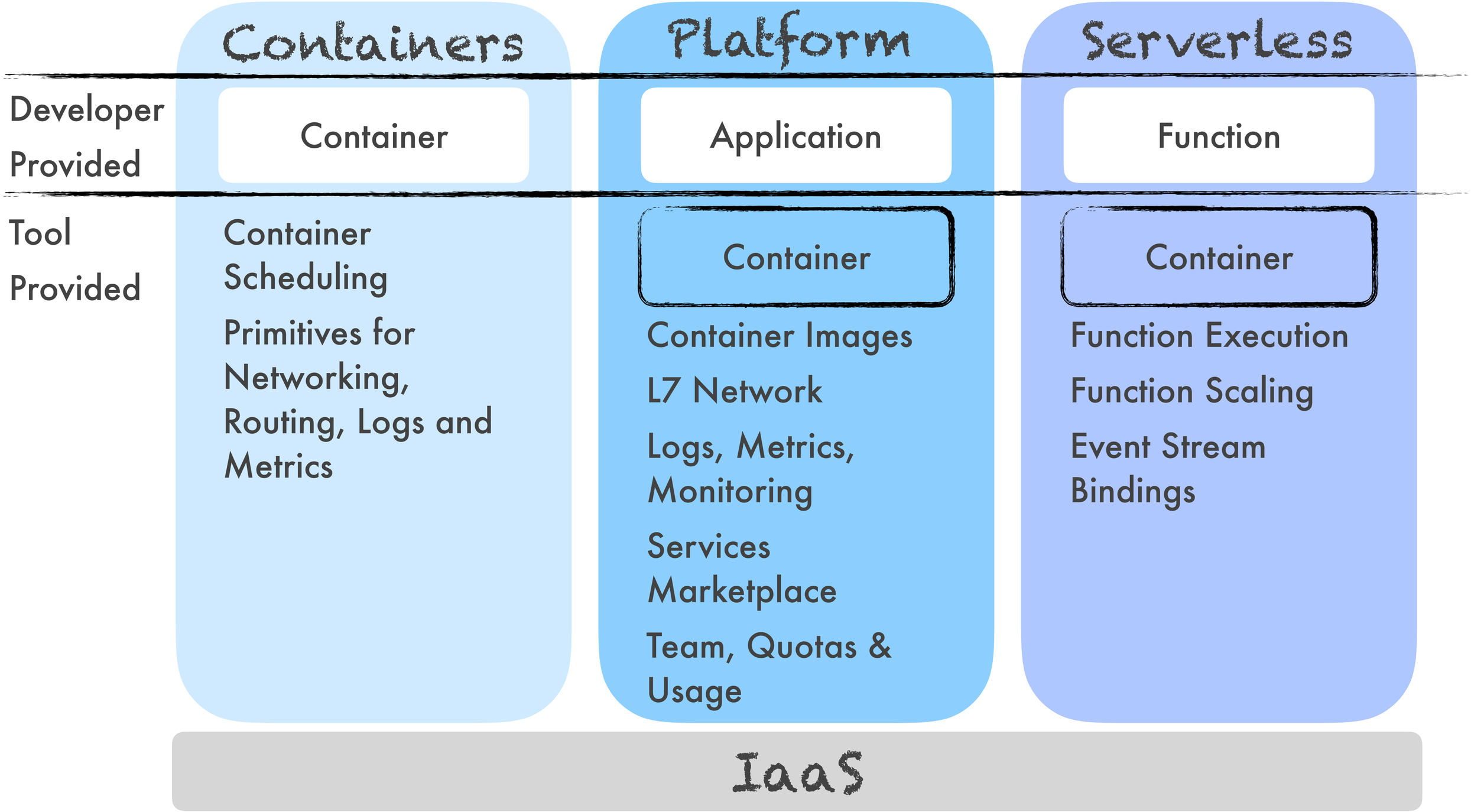
https://mobile.twitter.com/pczarkowski/status/1098978227169755136

In other words it is someone else’s problem.

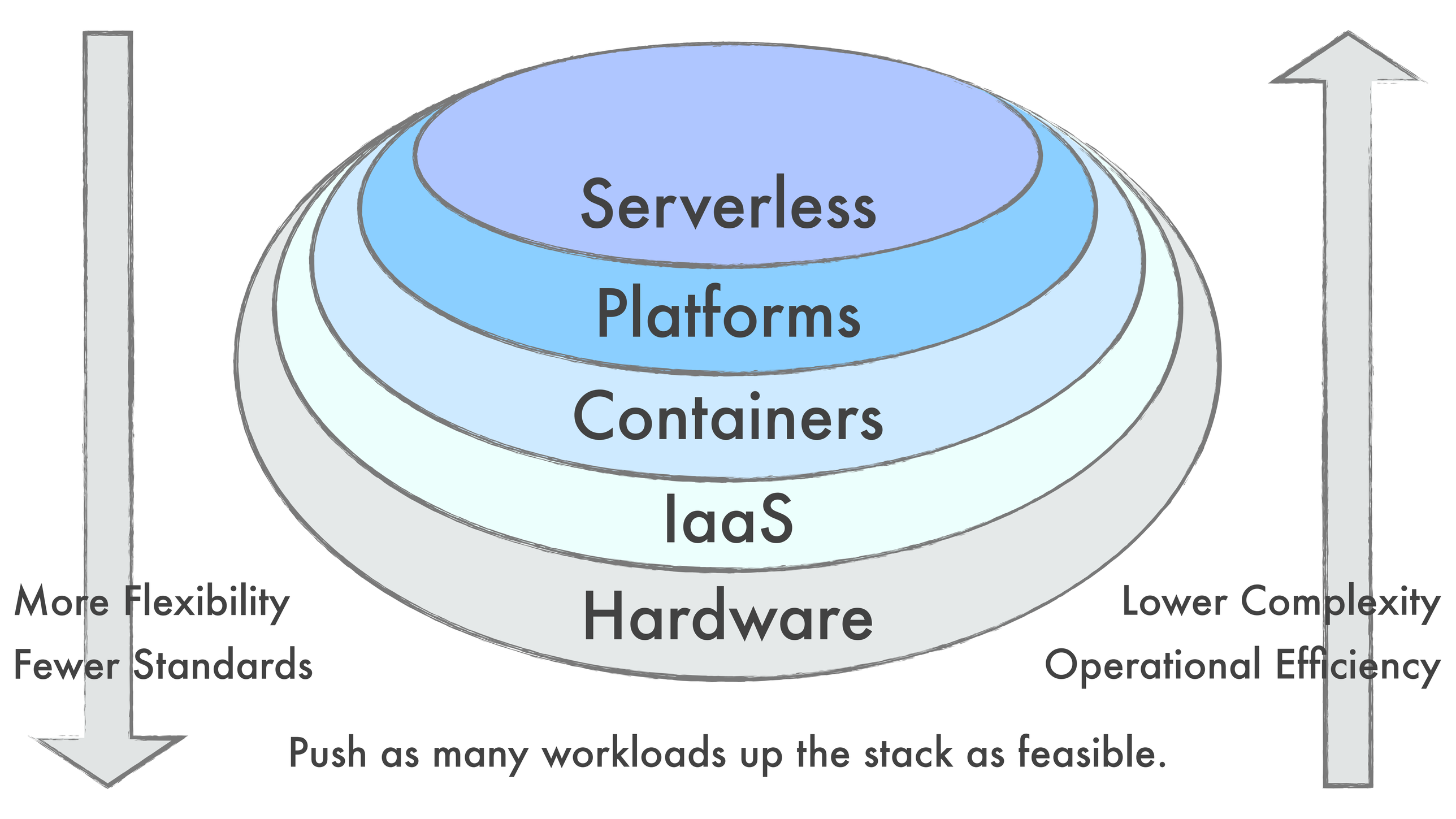
https://mobile.twitter.com/samnewman/status/952610105169793025



https://mobile.twitter.com/starbuxman/status/959366771462496256


Containers
Platform
Serverless
Developer Provided
Container
Application
Function
Tool Provided
Container Scheduling
Primitives for Networking, Routing, Logs and Metrics
Container
Container Images
L7 Network
Logs, Metrics, Monitoring
Services Marketplace
Team, Quotas & Usage
Container
Function Execution
Function Scaling
Event Stream Bindings
IaaS

Different levels of abstraction.


Serverless
Platforms
Containers
IaaS
Hardware
More Flexibility
Fewer Standards
Lower Complexity
Operational Efficiency
Push as many workloads up the stack as feasible.

Veritable plethora of options.

AWS Lambda, Azure Functions, Google Cloud Functions…

riff, OpenWhisk, Kubeless, Knative…

Definitely suffers from the shiny new thing curse.

And everything that entails.

There *are* very good reasons to utilize this approach!

But it isn’t just a new a way to cloud.

There are serious efficiency gains to be had with this approach!

Development efficiencies.

Functions push us further up the abstraction curve.

Allows us to focus on implementation not infrastructure.

Do you know what OS your application is running on?

Do you care?

What *should* you care about?

Where is the “value line” for you?

We want to get out of the business of “undifferentiated heavy lifting”.


"Good job configuring servers this year."
–No CEO Ever…

Focus on business problems,
not plumbing problems.

Resource efficiencies.

Function hasn’t been called recently?

Terminate the container.

Request comes in? Instance springs into existence.

First million (or two) requests are free*.

* Additional fees may apply.

For example: data transfer fees or other services you leverage.

Functions aren’t free however.

A fractional cost per request.

Charged based on # of requests, run duration & resource allocation.

Can be challenging to determine just how much it will cost…

But for certain workloads,
it is very cost effective.

Operational efficiencies.

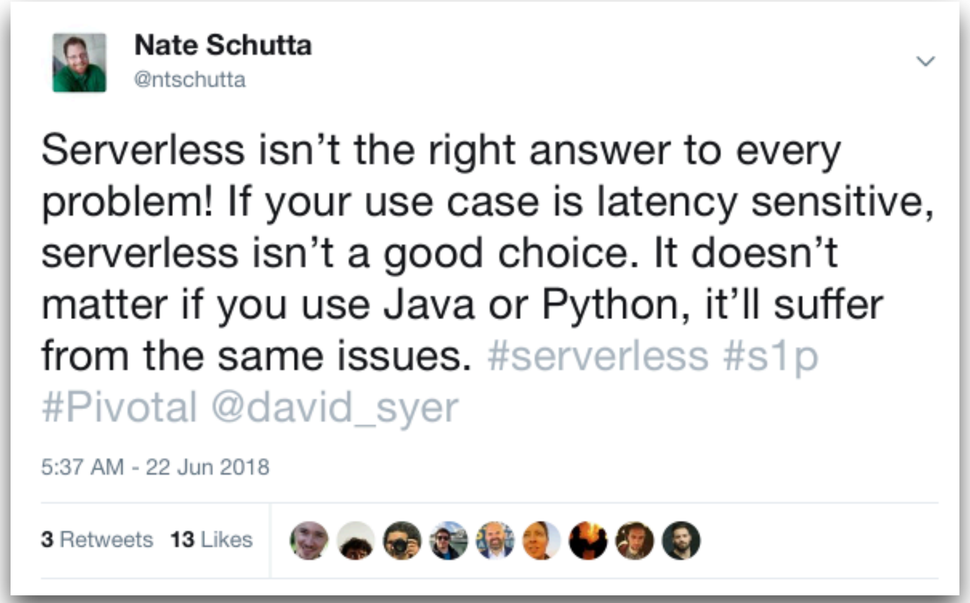
Serverless ops?

Again, less for us to worry about.

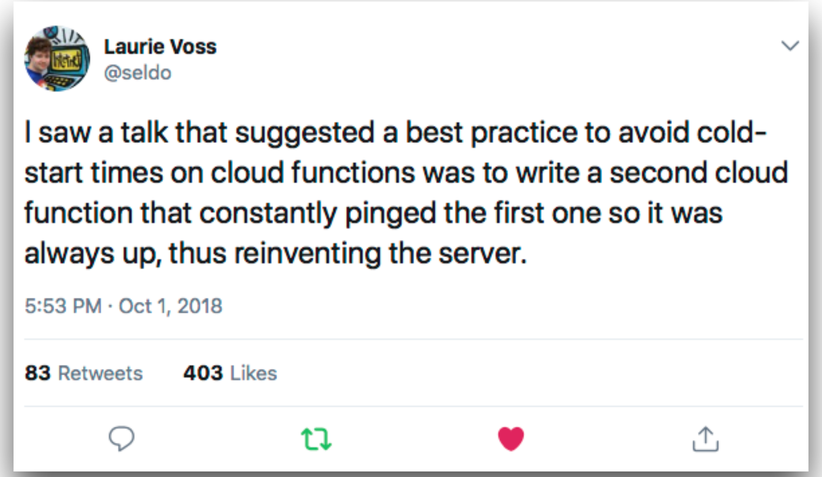
Rely on a platform.

Very valuable tool.

It isn’t a good fit for every workload.



https://twitter.com/ntschutta/status/1010109588832702464



https://mobile.twitter.com/seldo/status/1046895968329728000

But you knew that.


PLAN THE JOURNEY

Before you start, figure out where you are.

You need to assess the applications.

Some will be great candidates, others…

We need to understand a few things about our applications.

Technical characteristics.

What is the tech stack?

What version are we on?

How many users?

How many transactions per second (estimate)?

What components do we use?

What 3rd party things does the application use?

What are the data integrations?

What is the data access technology?

Do we use any internal frameworks?

Are there any batch jobs?

Do we have CI/CD? What are we using to build the apps?

What do we have for test coverage?

We need to assess the refactoring effort.

Look for certain red flags.

Vendor dependencies.

Writing to the file system.

Reading from the file system.

Long startup times.

Long shut down times.

Non-HTTP protocols.

Hard coded configuration.

Container based shared state.

Distributed transactions.

Evaluate your applications for 12 Factors compatibility.

Again, it is a sliding scale.

How far out of alignment is the app?

This effort requires application expertise.

And it will take time.

At least a few hours per.

Consider building a little application for all the data.

Excel is not an application platform. Cough.

Unless you have a small portfolio…

Assessments will bucket your applications.

Low/Medium/High.

Or red/yellow/green.

Whatever works!

“Cutoffs” likely arbitrary.

Sanity check it.

What is the business value of the application?

Consider the life cycle of the application.

Is it strategic?

Is it something we're going to invest in?

Or is it going to be retired soon?

Retirement isn’t a hard no though.

When matters. A lot.

“When I started, this app was marked sunset…”

That was 25 years ago. Still running.

If it is going away in a few months…not worth it.

Going to be around for a few years, probably is.

Now we need to do some planning.

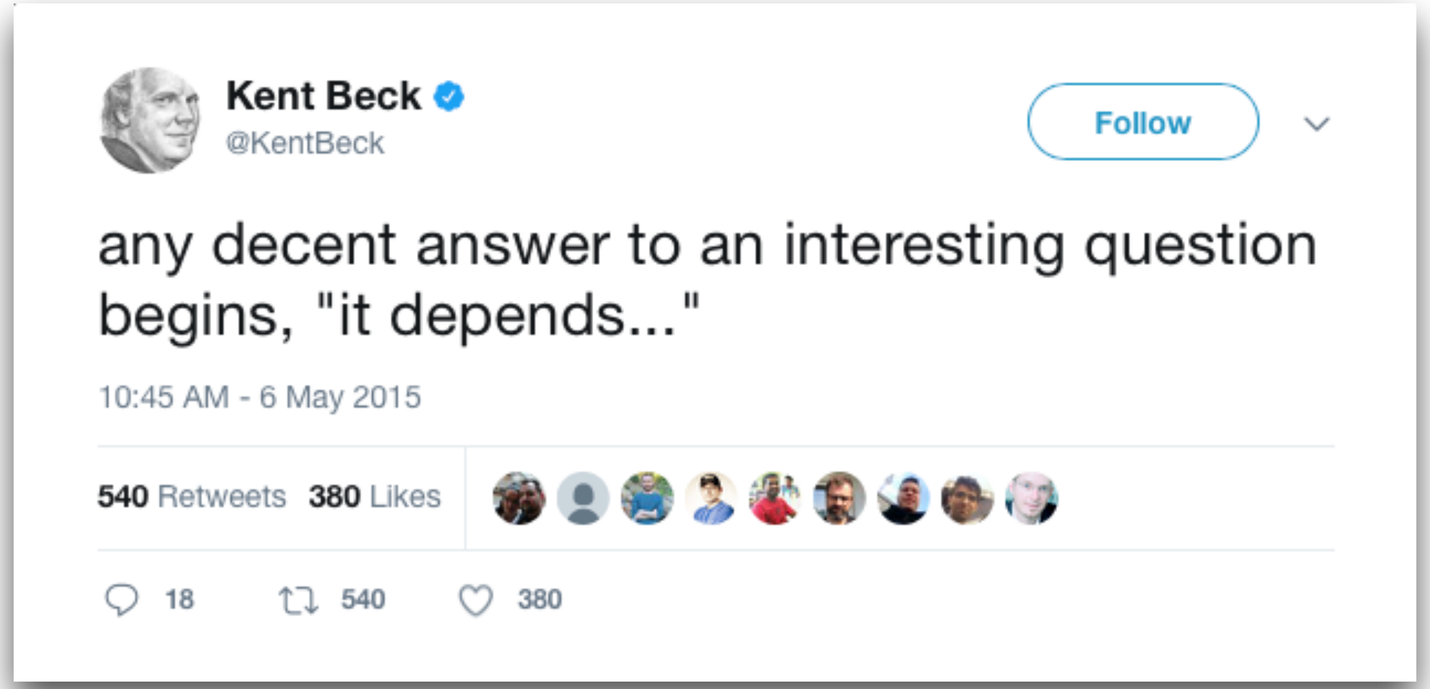
What is your desired end state?

Cloud native? Just get it running on cloud?

Legacy apps will require refactoring.

How long does it take to forklift an application?

https://blog.pivotal.io/pivotal-cloud-foundry/features/the-forklifted-application



https://twitter.com/KentBeck/status/596007846887628801

Strongly recommend
a pilot app or two.

Get a feel for it.

Usually a few weeks or so.

Ask the experts.

https://pivotal.io/application-transformation

Consider staffing up a “lab” team with expertise.

Help teams migrate.

Should *pair* with the application area.

Need to grow the skills.

Create a roadmap.

When can the applications move?

It doesn’t have to be
the entire application!

Some deployable units will be easy, others hard.

Move what you can.

Again, the terminology can be challenging…

Look to opportunistically migrate what you can.

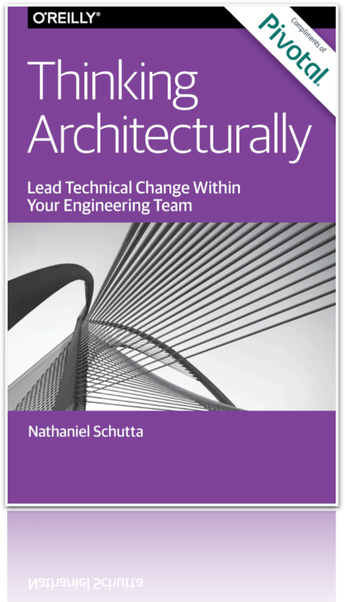
But have a rough idea of when.

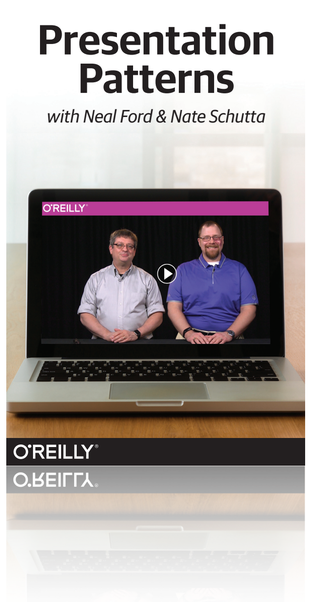
Does that satisfy your stakeholders?

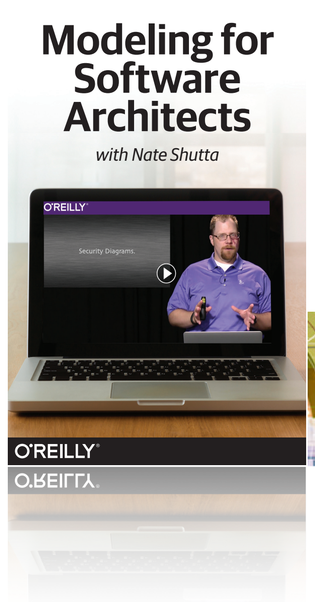
What can you do to accelerate the plan?

Good luck!

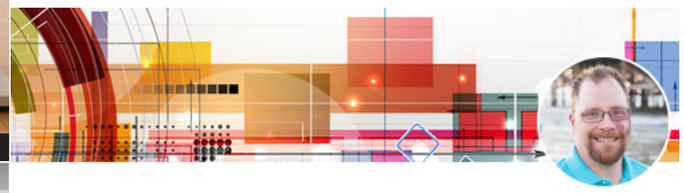
Thanks!







Nathaniel T. Schutta
@ntschutta
ntschutta.io



March 2 & 3, 2020

From developer to software architect

Presented by **Nathaniel Schutta**

SpringOne TOUR by Pivotal

Cloud-Native Java From the Source

The SpringOne Tour brings the best Cloud-Native Java content from our flagship conference directly to you. In 2 days, you'll learn about both traditional monolithic and modern, Cloud-Native Java from the source. Experience valuable facetime with expert Pivotal speakers in both traditional presentation and informal Pivotal Conversations about modern Application Development, DevOps, CI/CD, Cloud and more.