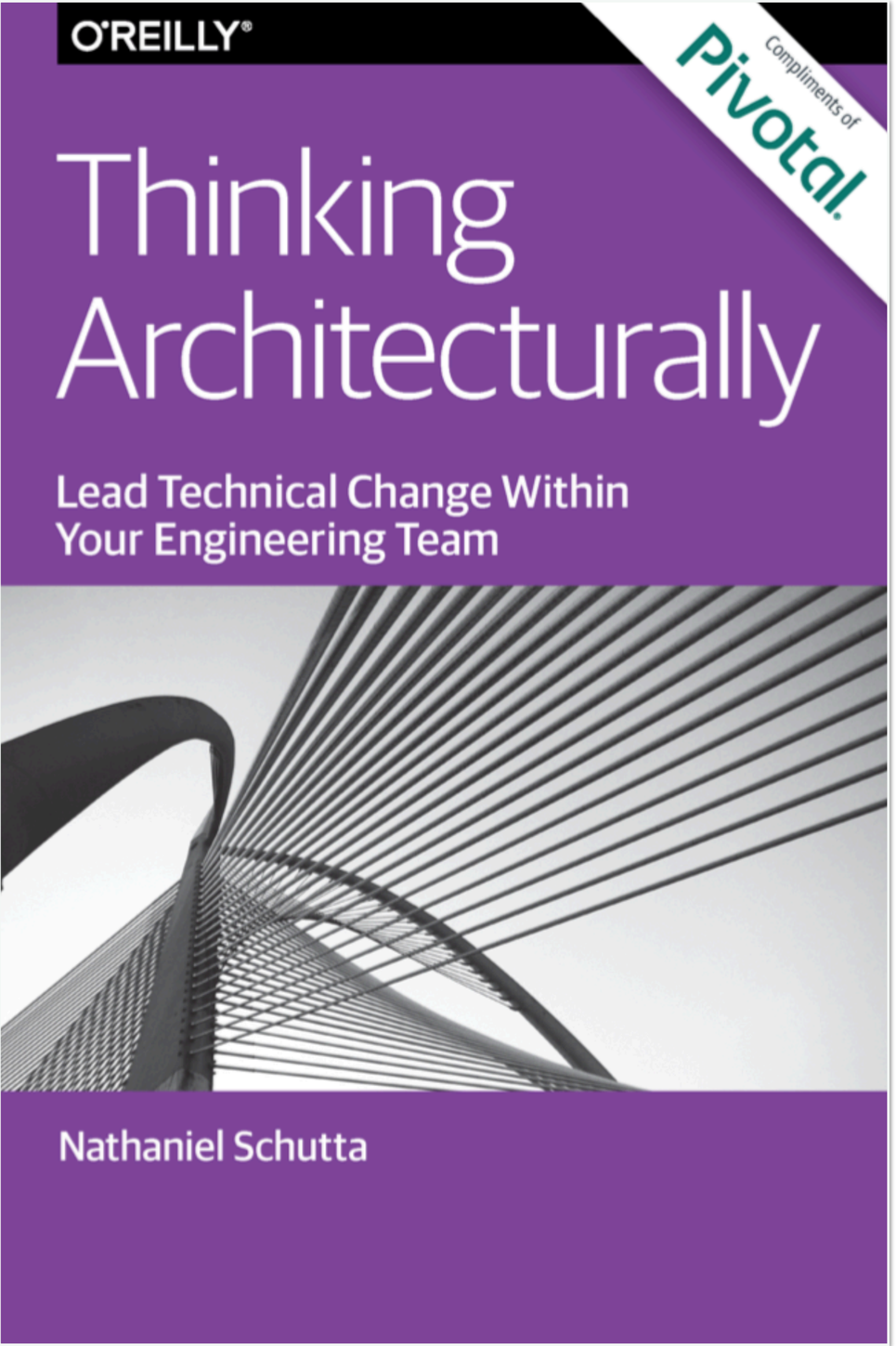

EVOLVING TO
CLOUD NATIVE
NATHANIEL SCHUTTA
@NTSCHUTTA
NTSCHUTTA.IO



https://tanzu.vmware.com/content/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.

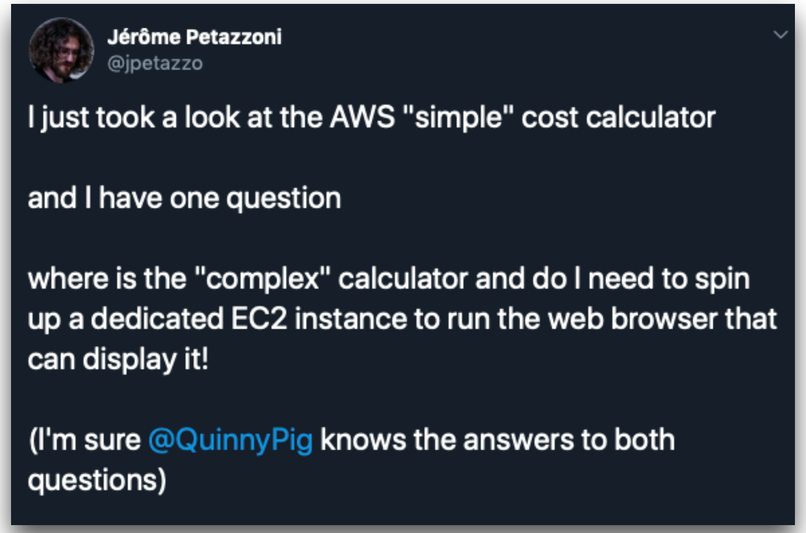
Scale up. Scale down. On demand.

Limitless compute.*

* Additional fees may apply.

Said fees can be…opaque.

https://mobile.twitter.com/whereistanya/status/1080864493108776961



https://mobile.twitter.com/jpetazzo/status/1227638126602080256

aws

SIMPLE MONTHLY CALCULATOR

Language: English

Need Help? Watch the Videos or Read How AWS Pricing Works or Contact Sales

Get Started with AWS: Learn more about our Free Tier or Sign Up for an AWS Account »

FREE USAGE TIER: New Customers get free usage tier for first 12 months

Reset All

Services | Estimate of your Monthly Bill ($ 0.00)

- Amazon EC2
- Amazon S3
- Amazon Route 53
- Amazon CloudFront
- Amazon RDS
- Amazon Elastic Load Balancing
- Amazon DynamoDB
- Amazon ElastiCache
- Amazon CloudWatch
- Amazon SES
- Amazon SNS
- Amazon Elastic Transcoder
- Amazon WorkSpaces
- Amazon WorkDocs
- AWS Directory Service
- Amazon Redshift
- Amazon Glacier
- Amazon SQS
- Amazon SWF
- Amazon Elastic MapReduce
- Amazon Kinesis Streams
- Amazon CloudSearch
- AWS Snowball
- AWS Direct Connect
- Amazon VPC
- Amazon SimpleDB

Choose region: US East (N. Virginia)

Inbound Data Transfer is Free and Outbound Data Transfer is 1 GB free per region per month

Clear Form

Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides resizable compute capacity in the cloud. It is designed to make web-scale computing easier for developers. Amazon Elastic Block Store (EBS) provides persistent storage to Amazon EC2 instances.

Compute: Amazon EC2 Instances:

	Description	Instances	Usage	Type	Billing Option	Monthly Cost
⊕	Add New Row					

Compute: Amazon EC2 Dedicated Hosts:

	Description	Number of Hosts	Usage	Type	Billing Option
⊕	Add New Row				

Storage: Amazon EBS Volumes:

	Description	Volumes	Volume Type	Storage	IOPS	Baseline Throughput	Snapshot Storage
⊕	Add New Row						

Compute: Amazon Elastic Graphics:

	Description	Number of Elastic Graphics	Usage	Elastic Graphics Size and Memory
⊕	Add New Row			

Additional T2/T3 Unlimited vCPU Hours per month:

For Linux, RHEL and SLES: 0

For Windows and Windows with SQL Web: 0

Elastic IP:*

(•) Enter values below ( ) Calculate

Total time the additional Elastic IPs are attached to running EC2 instances**: 0 Hours/Month

Total Non-attached time for all the Elastic IPs: 0 Hours/Month

Number of Elastic IP Remaps: 0 Per Month

Data Transfer:

Inter-Region Data Transfer Out: 0 GB/Month

Data Transfer Out: 0 GB/Month

Data Transfer In: 0 GB/Month

Common Customer Samples

- Free Website on AWS
- AWS Elastic Beanstalk Default
- Marketing Web Site
- Large Web Application (All On-Demand)
- Media Application
- European Web Application
- Disaster Recovery and Backup

https://mobile.twitter.com/paulbiggar/status/1228385370439467009

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”.

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?

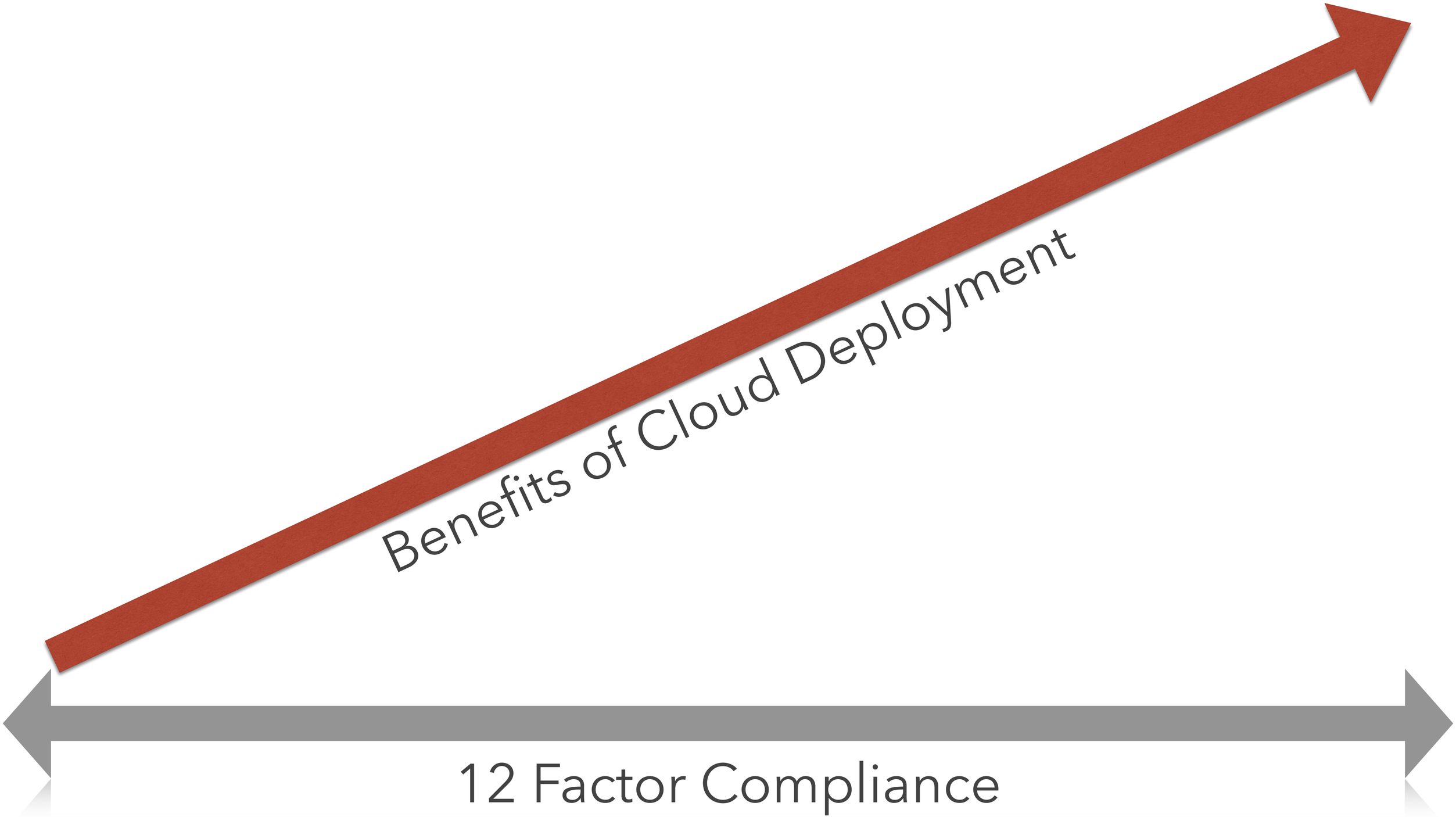
Sure.

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.


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.

youtube.com

YouTube

Search

SIGN IN

How Big Should Microservices Be?

2,361 views • May 3, 2020

121 1 SHARE SAVE

Sam Newman
719 subscribers

SUBSCRIBE

Another frequently asked question about microservices this week, and it's a doozy one. How big should microservices be?

Up next

AUTOPLAY

Principles Of Microservices by Sam Newman
Devoxx
213K views • 4 years ago

Introduction to Microservices, Docker, and Kubernetes
James Quigley
812K views • 2 years ago

Avoiding Microservice Megadisasters - Jimmy Bogard
NDC Conferences
315K views • 3 years ago

IRON SWING VS DRIVER SWING
Danny Maude
Recommended for you

Monolith To Microservices - Book Overview
Sam Newman
675 views • 2 months ago

Rory McIlroy shares tips to improve your drive | GOLFPAS...
Golf Channel
Recommended for you

GOTO 2019 • "Good Enough" Architecture • Stefan Tilkov
GOTO Conferences
121K views • 3 months ago

Why Do So Many Programmers Lose Hope?
Healthy Software Developer
115K views • 2 years ago

NOT ALL URETHANE GOLF BALL COVERS ARE THE SAME
Titleist
Recommended for you
New

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.

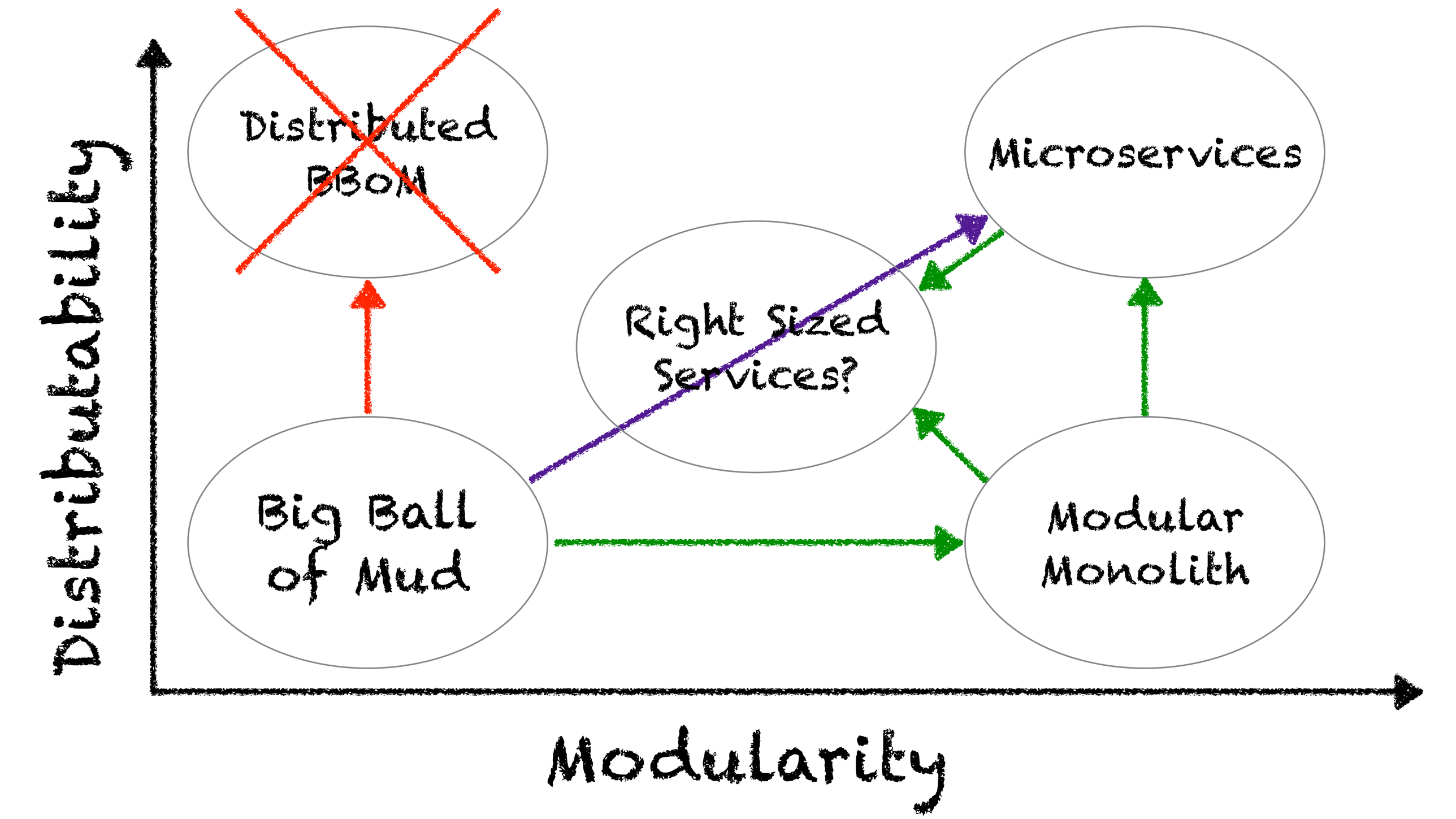
Please Microservice Responsibly.

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


Distributed BBoM
Microservices
Right Sized Services?
Big Ball of Mud
Modular Monolith
Distributability
Modularity

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.

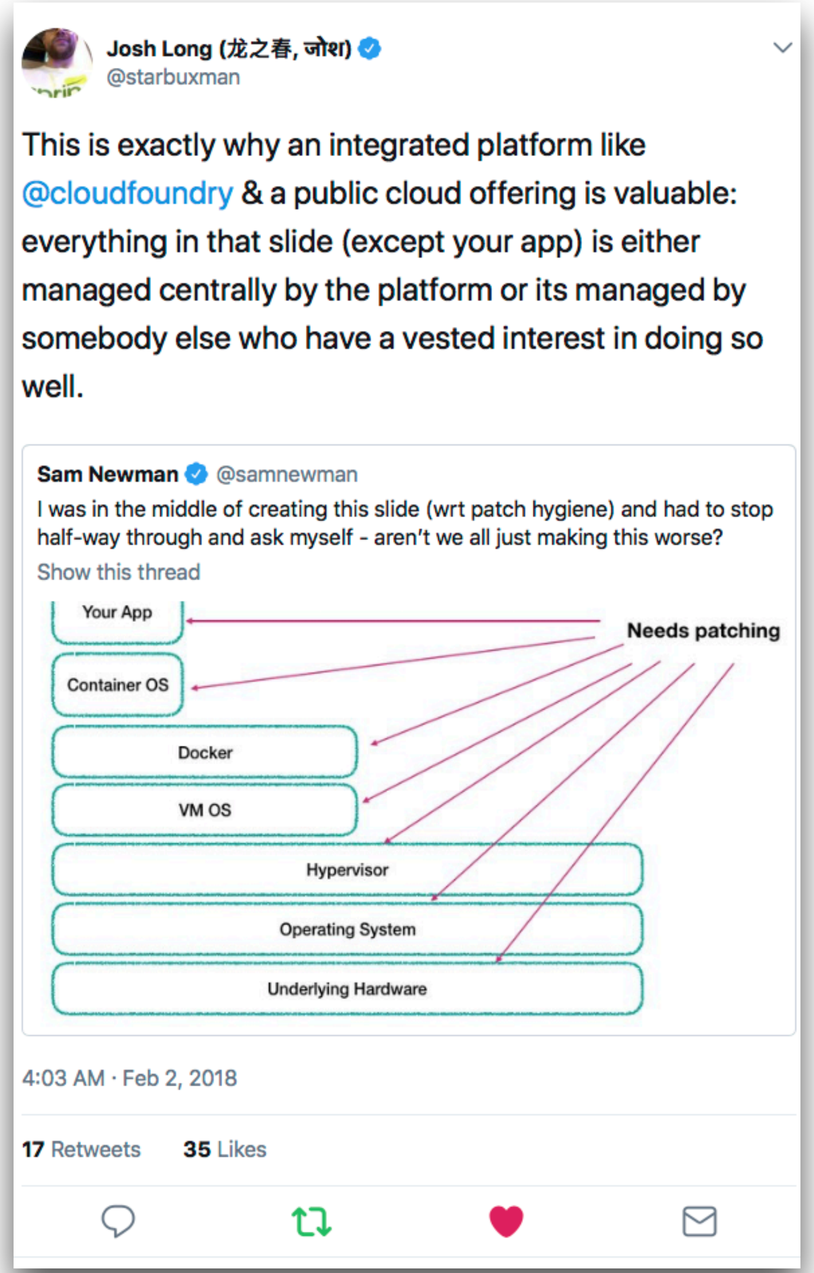
We are just (further)
abstracted away from them.

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

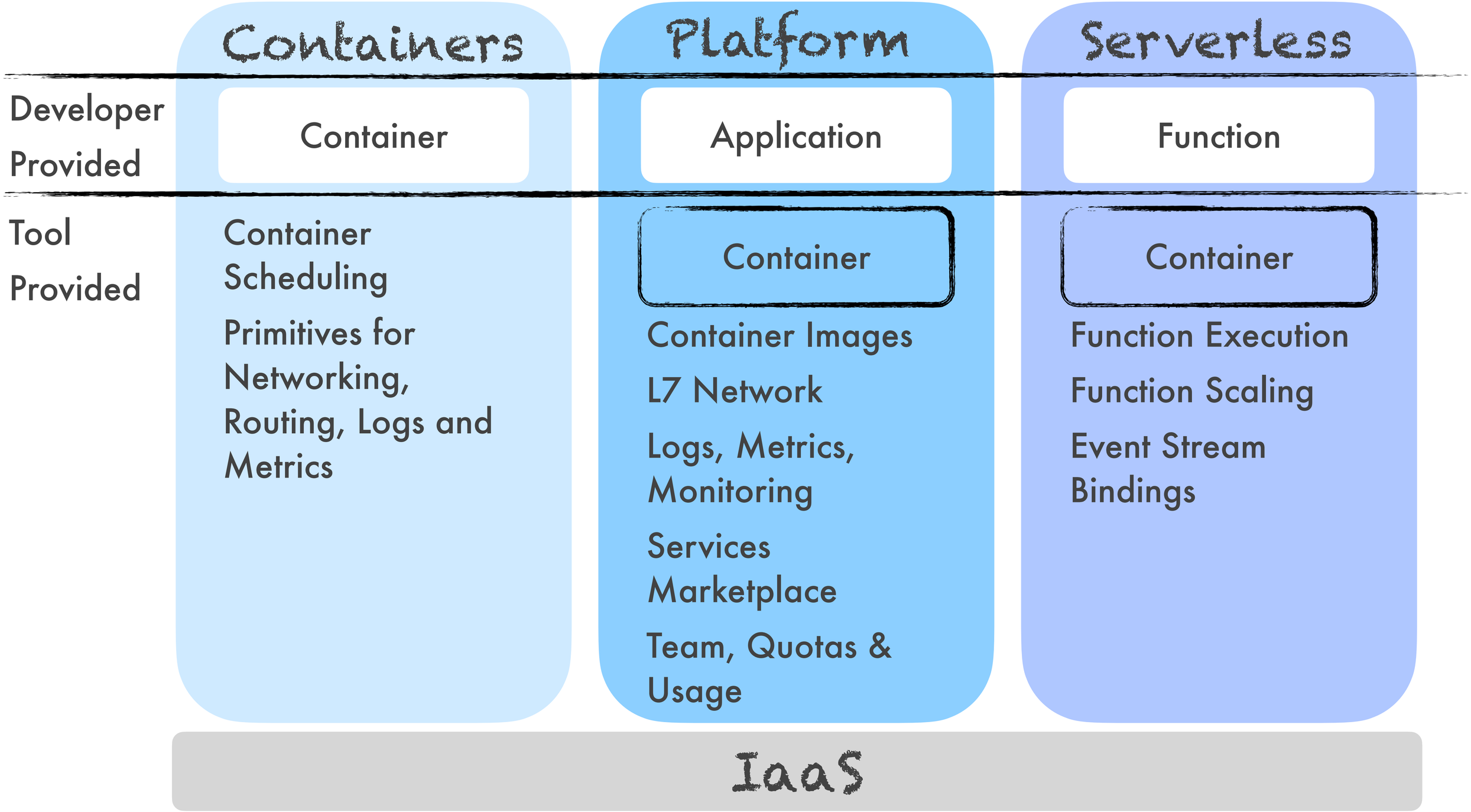
https://mobile.twitter.com/pczarkowski/status/1098978227169755136

In other words it is someone else’s problem.

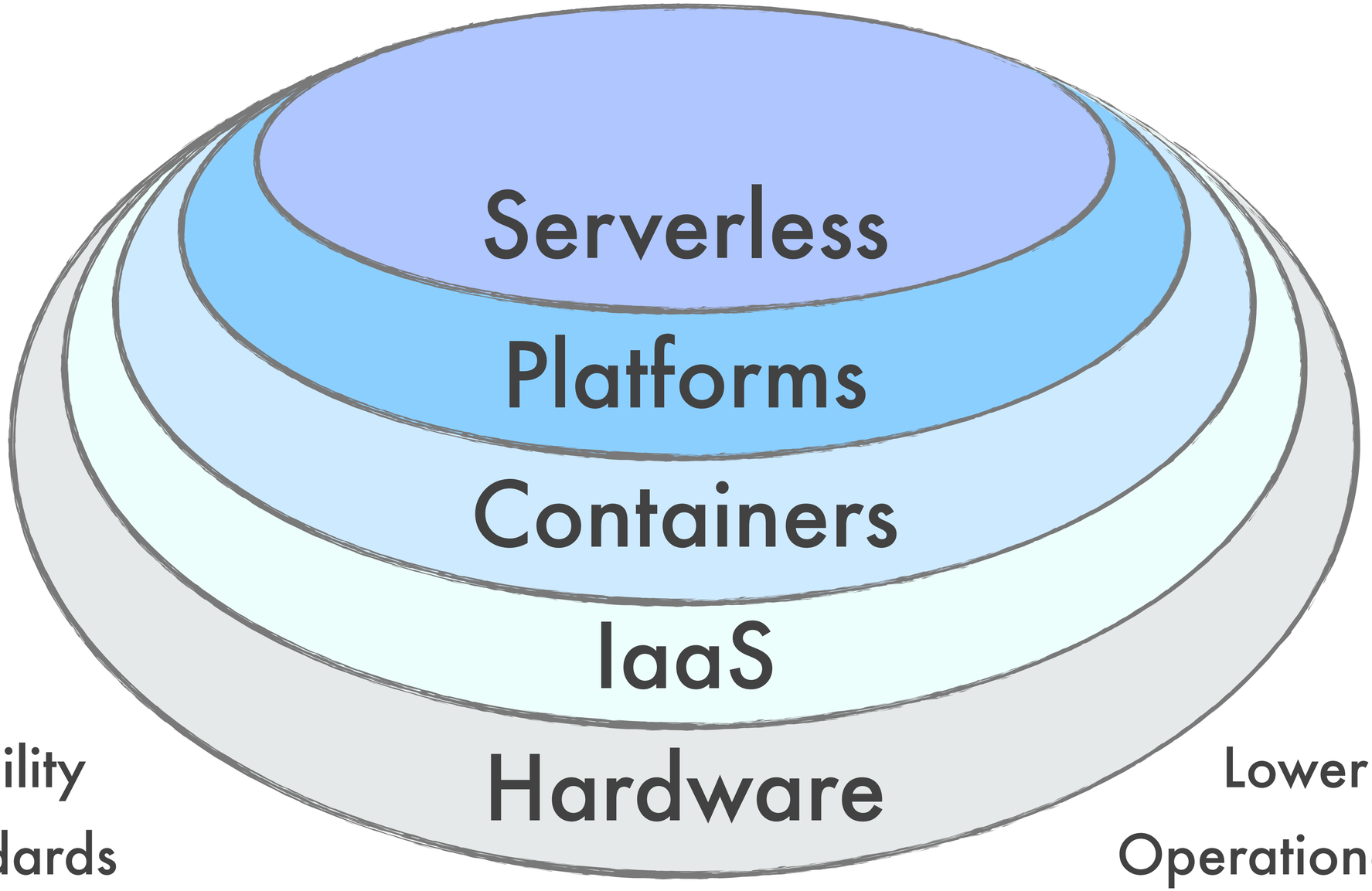
https://mobile.twitter.com/samnewman/status/952610105169793025



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.



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.

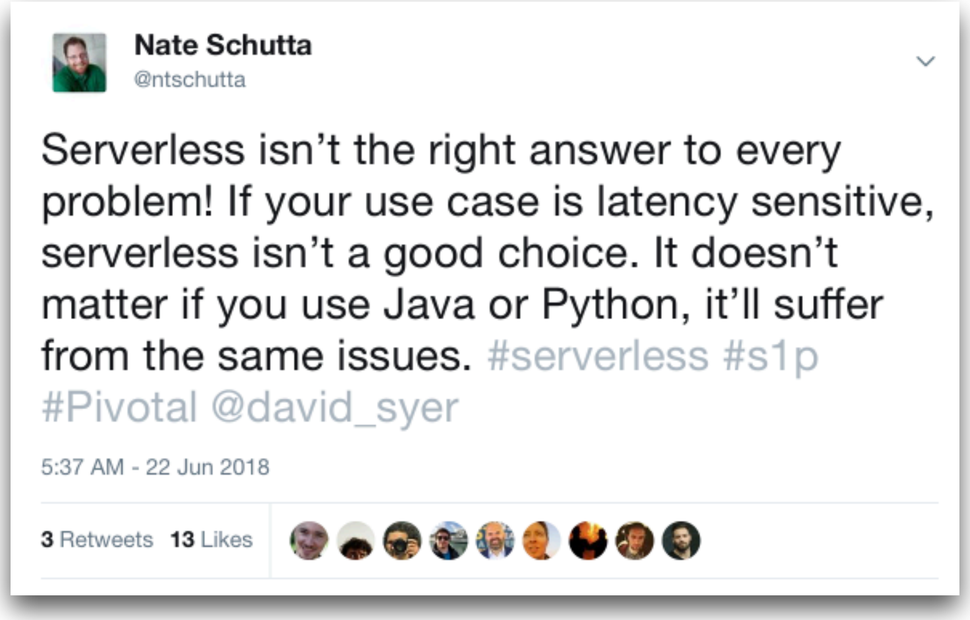
Serverless ops?

Again, less for us to worry about.

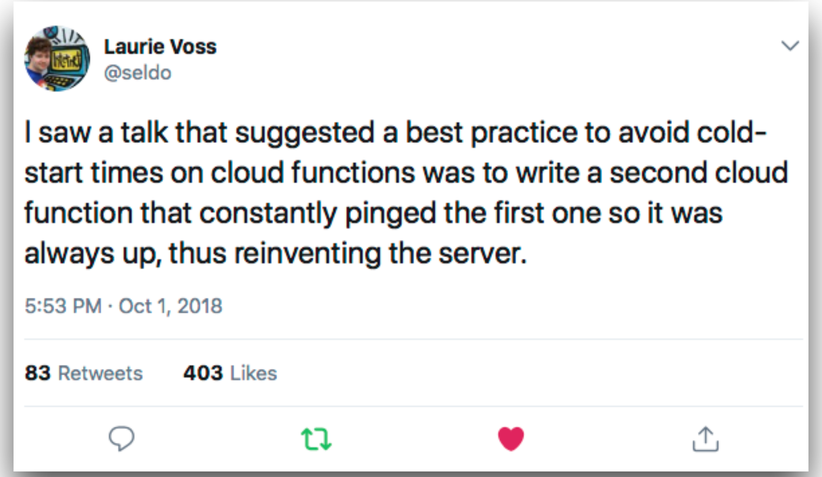
Rely on a platform.

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.

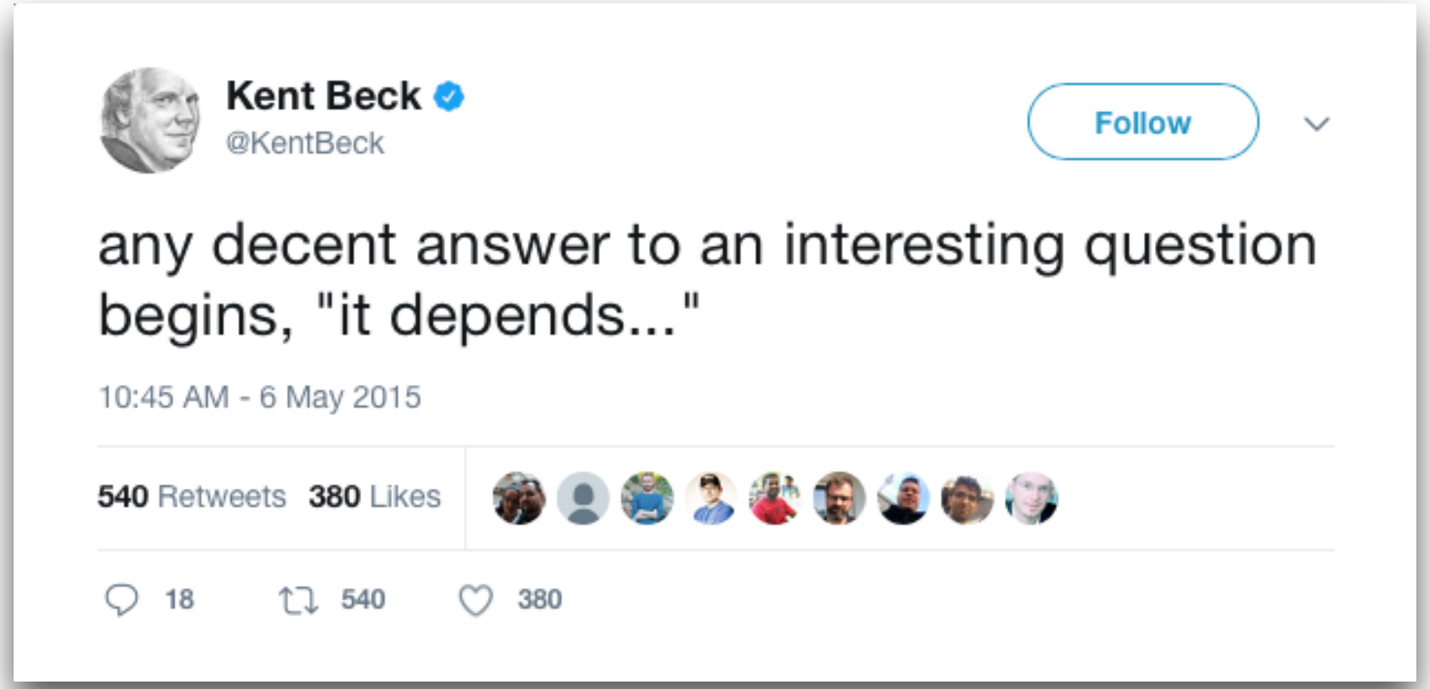
What is your desired end state?

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

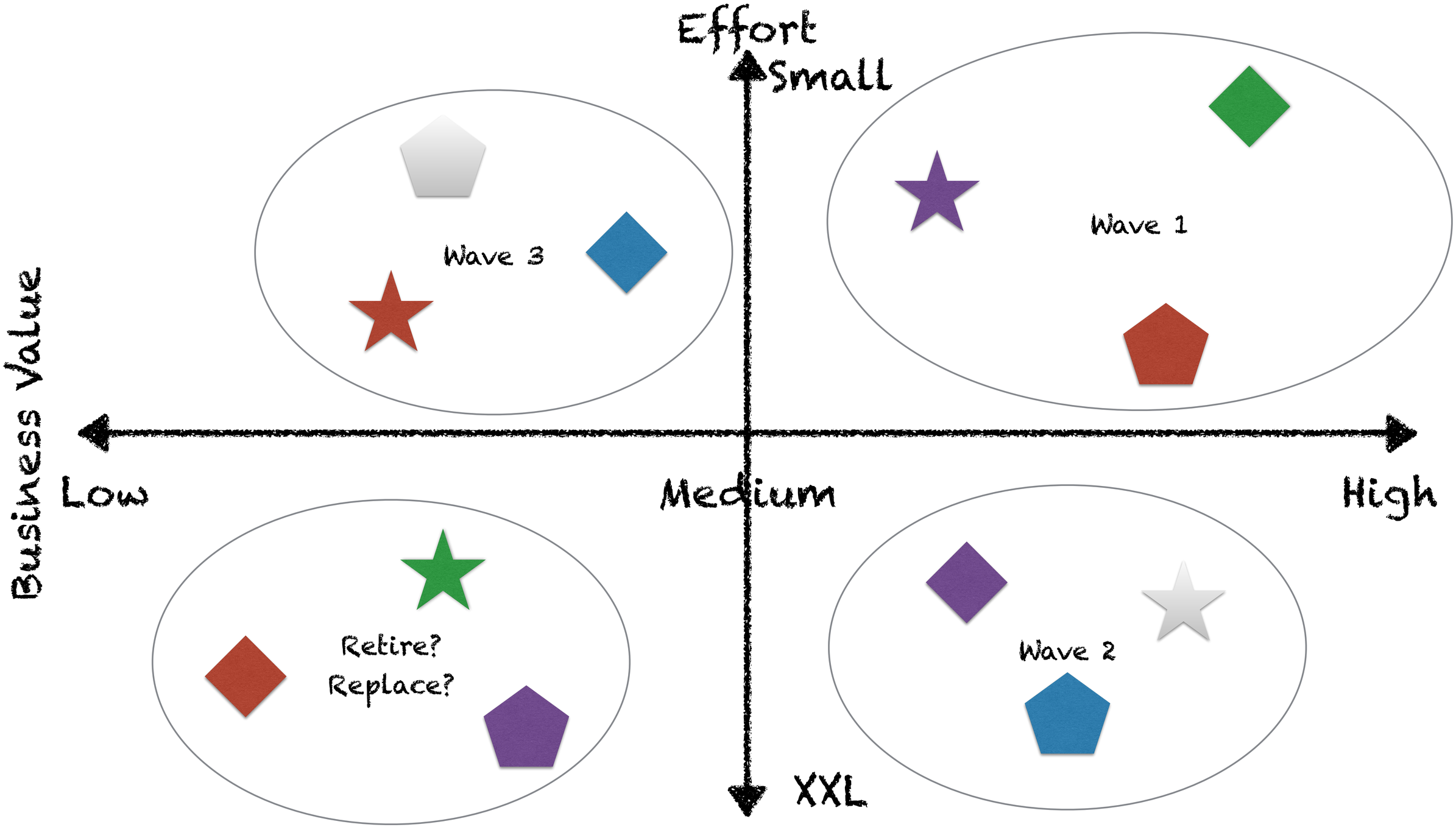
Consider staffing up a “lab” team with expertise.

Help teams migrate.

Should *pair* with the application area.

Need to grow the skills.

Create a roadmap.


Effort
Small
XXL
Business Value
Low
Medium
High
Wave 3
Wave 1
Retire?
Replace?
Wave 2

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.

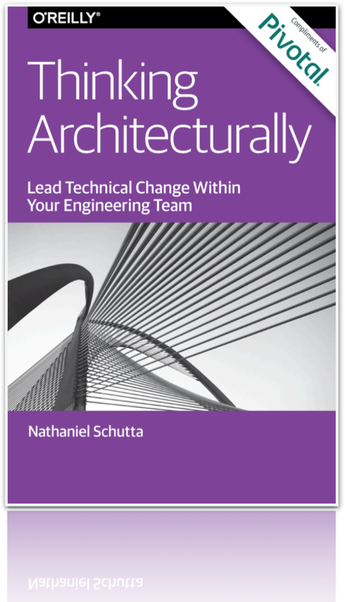
But have a rough idea of when.

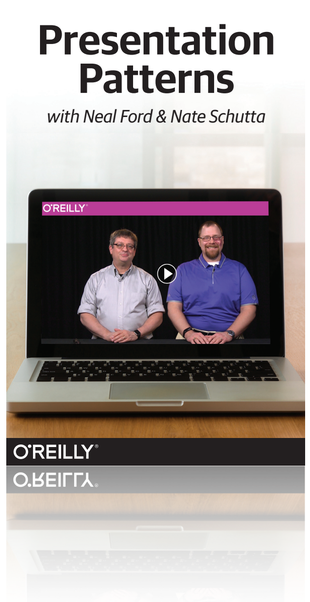
Does that satisfy your stakeholders?

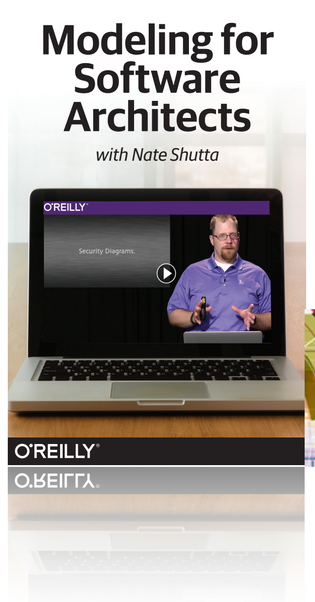
What can you do to accelerate the plan?

Good luck!

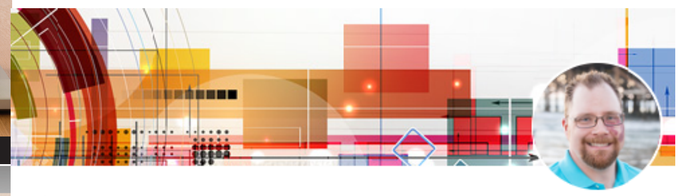
Thanks!







Nathaniel T. Schutta
@ntschutta
ntschutta.io



July 22 & 23, 2020
From developer to software architect
Presented by **Nathaniel Schutta**

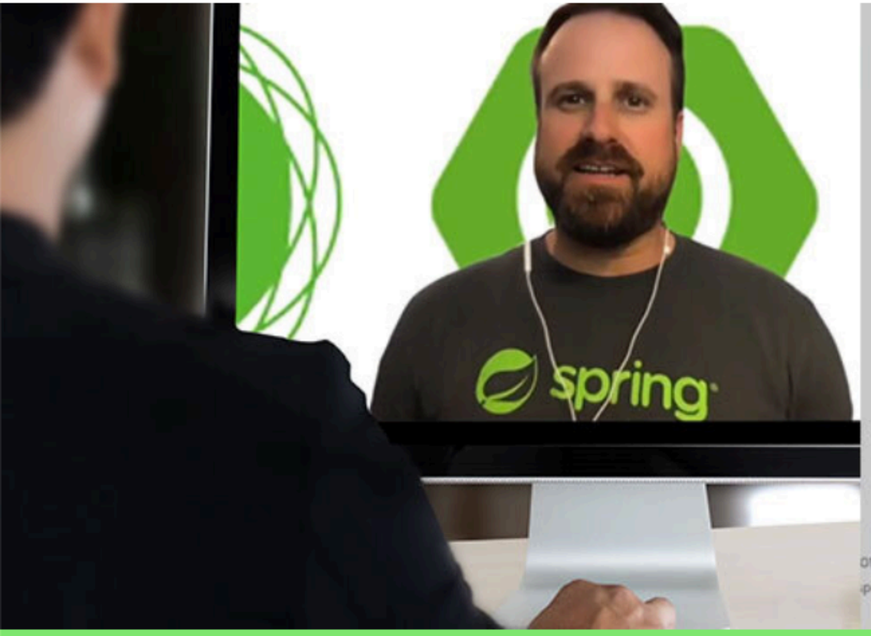
SpringOne TOUR

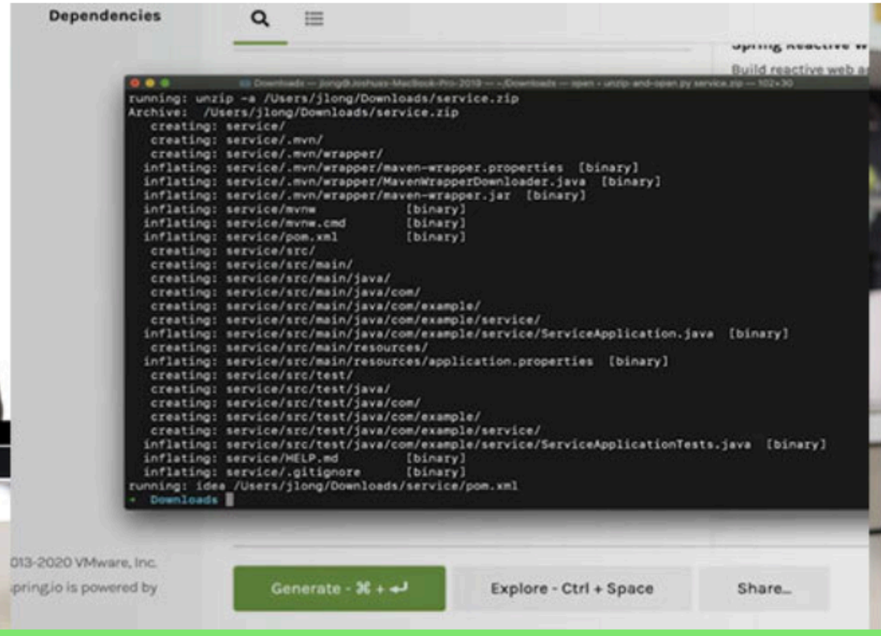
SpringOne Tour is going virtual

Even in an online format, SpringOne Tour still features the best **cloud native** Java content from **our annual developer conference**. Join us each month for a two-day, live event where your favorites from the cloud native community go in depth on a different topic, featuring a mix of presentations, interactive demos, and panel discussions.

FREE!

All events begin at 9 AM PDT





Topics

May 20–21	Jun 29–30	Jul 22–23