

Functions and Platforms and Containers, Oh My!

Nathaniel Schutta
@ntschutta

Ah the Cloud!

So. Many. Options.

Microservices. Modular monoliths.

Container all the things?

What about serverless?

I hear that is *the* in thing now.

And that's before we even talk
about public vs. private...

Or cloud providers!

How do we make sense of it all?

What is *the* answer?

IaaS.

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.

It would take weeks...or months...to
get a new server instance.

What was the longest “request to ready” experience of your career?

Required us to make decisions
when we knew the least.

How much capacity will you need?



Take worst case...double it...add
some buffer. Then a bit more.

Just in case.

We have a six week (aka month)
lead time on all requests.

Lots of tickets.

And meetings.

And email.

And followup.

Often gave our servers pithy names.

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.

Prices dropped.

Servers were no longer the
constraining factor.

People costs eclipsed hardware costs.

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

Shared servers became a liability.

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



Introduce greater
automation to operations.

Chef, Puppet, bosh.

Infrastructure as code.

People can't do the same thing twice.

See golf.

We need consistency.

A script will do the exact same thing.

Every. Single. Time.

Lead times dropped.

Often as a reaction to
public cloud providers.

And shadow ops.

Aka team goes rogue.

Spins up an instance on AWS or GCP.

Takes seconds.

IaaS - provision server resources quickly (in many cases on premises).

Sometimes on demand self service.

In other instances, just the same process as before but faster.

How?

Virtualization.

Abstraction over the physical
compute resources.

Hypervisor manages the VMs.

Provides course grained components.

Useful if you need the flexibility.

But more for you to manage.

“With great power comes
great responsibility.”

— Uncle Ben

There ^{are}^{*} instances where we need
that level of flexibility.

But it shouldn't be your first thought.

Fundamentally, the underlying
layer of “the cloud”.

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 like Docker Hub.

hub.docker.com

Docker Store is the new place to discover public Docker content. [Check it out →](#)

Dashboard Explore Organizations Create ntschutta

Explore Official Repositories

| | | | |
|---|---------------|---------------|-------------------------|
|  alpine official | 3.4K STARS | 10M+ PULLS | DETAILS |
|  nginx official | 8.2K STARS | 10M+ PULLS | DETAILS |
|  httpd official | 1.6K STARS | 10M+ PULLS | DETAILS |
|  redis official | 4.9K STARS | 10M+ PULLS | DETAILS |
|  busybox official | 1.2K STARS | 10M+ PULLS | DETAILS |
|  ubuntu official | 7.4K STARS | 10M+ PULLS | DETAILS |
|  mongo official | 4.3K STARS | 10M+ PULLS | DETAILS |
|  memcached official | 968 STARS | 10M+ PULLS | DETAILS |

Docker Hub is community driven.

It isn't vetted. Or curated.

Anyone can push anything,
anyone can pull anything.

There is also the Docker Store.

Which is exactly what it sounds like.

Allows people to publish
and ~~sell~~ Docker images.

A screenshot of the Docker Store interface on a Mac OS X desktop. The search bar at the top contains the query "java". The main content area displays three container offerings:

- IBM Db2 Warehouse client container** (By IBM) - A container for migrating, operating, and maintaining Db2 Warehouse. It is Docker Certified and available for Container, Linux, IBM POWER, x86-64, Analytics, and Databases.
- OpenMapTiles Map Server** (By Klokan Technologies) - Provides OpenStreetMap Maps with API support for Leaflet, OpenLayers, WMS, WMTS, GIS, and Mapbox SDKs. It is available for Container, Linux, x86-64, and Application Services.
- CoScale agent** (By CoScale) - Production monitoring for Dockerized applications using anomaly detection. It is available for Container, Docker Certified, Linux, x86-64, DevOps Tools, Analytics, and Monitoring.

The sidebar on the left includes filters for Docker EE and Docker CE, and sections for Filters, TYPE (Store selected), DOCKER CERTIFIED (Docker Certified checked), and CATEGORIES (Analytics, Application Frameworks, Application Infrastructure, Application Services, Base Images, Databases, DevOps Tools, Featured Images, Messaging Services).

A screenshot of the Docker Store interface on a Mac OS X browser window. The URL in the address bar is `store.docker.com`. The search bar at the top contains the query `java`.

The main header features the **Docker store** logo, a search icon, and a user profile for **ntschutta**. Navigation links include **Explore**, **Publish**, and **Feedback**.

The Docker Store

Find Trusted and Enterprise Ready Containers, Plugins, and Docker Editions

Filter options include:

- TYPE**: Store, Community (Docker Hub)
- DOCKER CERTIFIED**: Docker Certified
- CATEGORIES**:
 - Analytics
 - Application Frameworks
 - Application Infrastructure
 - Application Services
 - Base Images
 - Databases
 - DevOps Tools
 - Featured Images
 - Messaging Services

Sort order: Most Popular

Search results for `java`:

- Oracle Java 8 SE (Server JRE) ⚡**
By Oracle
Oracle Java 8 SE (Server JRE)
Container, Docker Certified, Linux, x86-64, Programming Languages
- AppDynamics**
By AppDynamics (Cisco)
AppDynamics APM provides real-time, end-to-end management for the most complex and distributed applications.
Container, Linux, x86-64, Monitoring, Application Infrastructure, DevOps Tools, Analytics, Application Frameworks, Application Services, Base Images
- java**
By Docker
Java is a concurrent, class-based, and object-oriented programming language.
Container, Linux, x86-64, Base Images, Programming Languages

Screenshot of the Docker Store product page for Oracle Java 8 SE (Server JRE).

The page shows the following details:

- Product Name:** Oracle Java 8 SE (Server JRE)
- Provider:** Oracle
- Description:** Oracle Java 8 SE (Server JRE)
- Tags:** Container, Docker Certified, Linux, x86-64, Programming Languages
- Certification:** docker CERTIFIED
- Developer Tier:** Checked
- Terms of Service:** Link
- Proceed to Checkout:** Button

The page also includes sections for Description, Reviews, and Resources.

DESCRIPTION

Java Platform, Standard Edition (Java SE) lets you develop and deploy Java applications on desktops and servers, as well as in today's demanding embedded environments. Java offers the rich user interface, performance, versatility, portability, and security that today's applications require.

This Docker images provides the Server JRE, a runtime environment specifically targeted for deploying Java in server environments. The Server JRE includes tools for JVM monitoring and tools commonly required for server applications, but does not include browser integration (the Java plug-in).

REVIEWS

Average Rating: ★★★★☆ 9 Ratings

RESOURCES

EXPLORE

- Docker Editions
- Containers
- Plugins

ACCOUNT

- My Content
- Billing

PUBLISH

- Publisher Center

SUPPORT

- Feedback
- Documentation

SOCIAL

- Github
- Twitter
- Google+
- Facebook

Copyright © 2018 Docker Inc. All rights reserved.

Hub Cloud Legal Docker Swag

As you might have guessed, Docker Store content is vetted and curated.

Commercial entities.

Approved by Docker.

Images are supported by said entities.

And some cost money. Imagine that.

Some images are Docker Certified.

And note, some images come
from Docker directly.

store.docker.com

- Base Images
- Databases
- DevOps Tools
- Featured Images
- Messaging Services
- Monitoring
- Operating Systems
- Programming Languages
- Security
- Storage

OPERATING SYSTEMS

- Linux
- Windows

ARCHITECTURES

- IBM POWER
- IBM Z
- x86-64

 **java**
By Docker ↓ 10M+

Java is a concurrent, class-based, and object-oriented programming language.

Container Linux x86-64 Base Images Programming Languages

 **tomcat**
By Docker ↓ 10M+

Apache Tomcat is an open source implementation of the Java Servlet and JavaServer Pages technologies

Container Linux x86-64 IBM Z IBM POWER Application Infrastructure

 **tomee**
By Docker ↓ 5M+

Apache TomEE is an all-Apache Java EE certified stack where Apache Tomcat is top dog.

Container Linux IBM POWER x86-64 IBM Z Application Frameworks

 **node**
By Docker ↓ 10M+

Node.js is a JavaScript-based platform for server-side and networking applications.

Container Linux x86-64 IBM Z IBM POWER Application Infrastructure

 **groovy**
By Docker ↓ 10M+

Apache Groovy is a multi-faceted language for the Java platform.

Container Linux x86-64 Programming Languages

 **jetty**
By Docker ↓ 10M+

jetty provides a Web server and javax.servlet container.

Docker Hub and Docker Store are based on the Docker Public 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

SHARE

- [Share](#)
- [Tweet](#)
- [Email](#)
- [Print](#)

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



FROM THE WEB

Sponsored Links



Find Out In One Minute If You Pre-Qualify For A Citi Card

Citi



Why These 10 SUVs are the Cream of the Crop

Kelley Blue Book

by Taboola ▶

MORE FROM NBC NEWS



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.

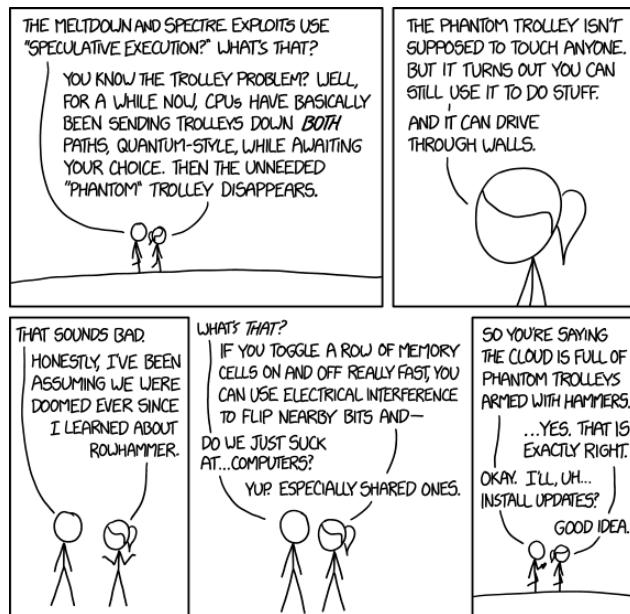
If your computer has a vulnerable processor and runs an unpatched operating system, it is not safe to work with



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.

One 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.

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](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.

Application Platforms.

Layer of abstraction on top of IaaS.

Developers focus on applications and data. Not middleware and runtime.

Concepts are similar across options.

In the case of Cloud Foundry...

BOSH sets up a pool of VMs.

Cloud Controller deploys and runs the apps on said VMs.

Router directs traffic
to the proper VMs.

Developers write the application.

Runtime is provided by the platform.

Before PaaS, we moved **code**
from one server to another...

Worked in dev...but not test.



Don't move the code,
move the runtime.

Coarser grained thing.

Worked in dev? That is the *exact*
same thing we will run in test.

Probably hasn't even changed VMs,
we just update the test route.

Eliminates an entire class of issues.

We have consistency!

Remember the 12 Factors?

X. Dev/Prod parity.

Environments should be consistent!

Shorten code to prod cycle.

Where does the runtime come from?

Buildpacks.

Push an app, CF figures out what kind
of app it is. Java? Go? .NET? etc.

Retrieve the buildpack, “deploy” your app to said buildpack.

That droplet gets uploaded and pushed to an available cell.

Buildpack has everything you
need to run that kind of app.

Moves the value line.

Less “undifferentiated heavy lifting”.

Onsi Fakhouri
@onsijoe

cf push haiku

here is my source code
run it on the cloud for me
i do not care how

4:18 PM · May 12, 2015

68 Retweets 105 Likes

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

What does that look like?

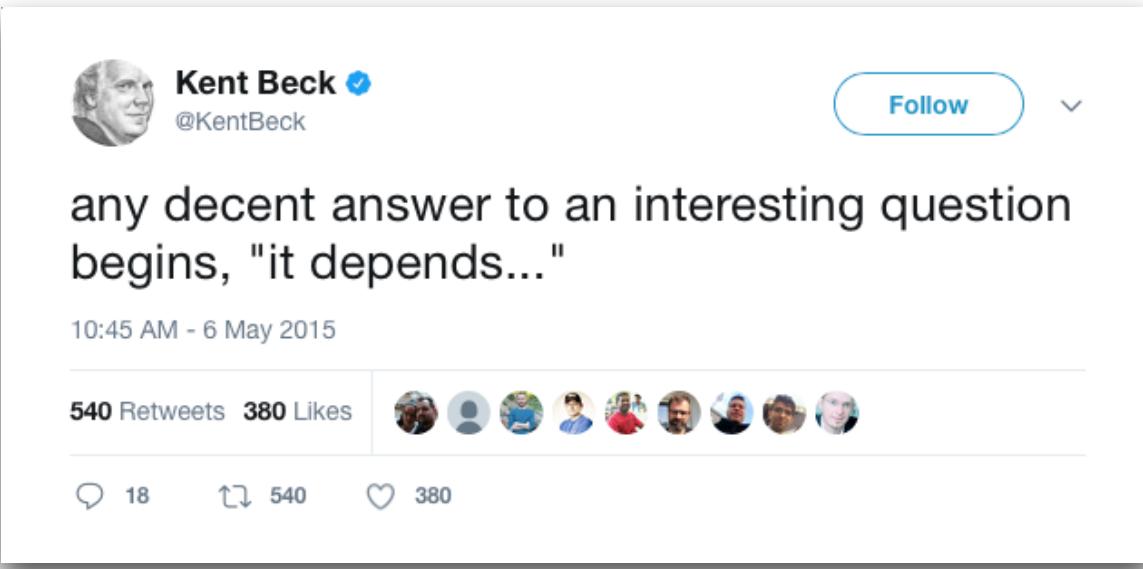
4. nschutta@Firethorn: ~/Sites/just-a-thought/public (zsh)
public (master) » c

~/Sites/just-a-thought/public

But what if I need something unique?

You *can* customize a buildpack.

Should you though?



A screenshot of a Twitter post from user Kent Beck (@KentBeck). The post features a profile picture of Kent Beck, his name, a blue verified checkmark, and his handle @KentBeck. To the right is a 'Follow' button and a dropdown menu icon. The tweet itself reads: "any decent answer to an interesting question begins, "it depends..."". Below the tweet is the timestamp "10:45 AM - 6 May 2015". A horizontal line follows, with "540 Retweets" and "380 Likes" on the left, and a row of small circular profile pictures on the right. At the bottom, there are engagement icons: a speech bubble with "18", a retweet icon with "540", and a heart icon with "380".

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

In some cases, absolutely.

But your first reaction should be
“Why do I think I need to do this?”

You build it, you own it.

Do you want to maintain
that for the next 3 years?

Don't forget to stay current...

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

Point of a platform - free you from
the undifferentiated heavy lifting.

What do our customers value?

The runtime?

That's a dial tone. They don't care.

We don't get cookies for
updating to OS version X+1.

Our customers want features.

Platforms allow us to spend more of
our time delivering value.

Less on plumbing.

Various options...

Cloud Foundry, Heroku.

Serverless.

Life moves pretty fast...

From IaaS to CaaS to PaaS...

Functions.

As a Service.

Now you're saying we
don't even need servers?

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...

In other words it is
someone else's problem.

Definitely suffers from the
shiny new thing curse.



And everything that entails.



There ^{are}* very good reasons
to utilize this approach!

Some relate specifically to a platform.

Perhaps you already have
data with a provider.

Using Amazon S3 for example.

You might want to tap into particular services offered by a provider.

Perhaps you want to utilize Google
Cloud Machine Learning Engine.

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”.

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.

| Every 3.0s: kubectl get functions,topics,pods,services,deploy Firethorn.local: Mon Jan 8 15:23:28 2018 | | | | | | |
|--|--------------|----------------|-------------|----------------|-----|---|
| NAME | AGE | READY | STATUS | RESTARTS | AGE | NAME |
| functions/echo-java | 32s | 1/1 | Running | 5 | 5h | po/demo-riff-function-controller-6975dbdc7d-kvfxm |
| topics/echo-java | 32s | 1/1 | Running | 11 | 5h | po/demo-riff-http-gateway-64fc56bd96-vmvnn |
| | | 1/1 | Running | 9 | 5h | po/demo-riff-kafka-c7f456685-p65bv |
| | | 1/1 | Running | 5 | 5h | po/demo-riff-topic-controller-58694bb5bf-n5fb5 |
| | | 1/1 | Running | 0 | 1h | po/demo-riff-zipkin-6c66b788bb-dhsjm |
| | | 1/1 | Running | 2 | 5h | po/demo-riff-zookeeper-6fd5c5bd54-fqvvk |
| <hr/> | | | | | | |
| NAME | TYPE | CLUSTER-IP | EXTERNAL-IP | PORT(S) | AGE | NAME |
| svc/demo-riff-function-controller | ClusterIP | 10.109.143.74 | <none> | 80/TCP | 5h | svc/demo-riff-function-controller |
| svc/demo-riff-http-gateway | NodePort | 10.101.136.125 | <none> | 80:31718/TCP | 5h | svc/demo-riff-http-gateway |
| svc/demo-riff-kafka | ClusterIP | 10.96.180.243 | <none> | 9092/TCP | 5h | svc/demo-riff-kafka |
| svc/demo-riff-zipkin | LoadBalancer | 10.98.172.131 | <pending> | 9411:31239/TCP | 1h | svc/demo-riff-zipkin |
| svc/demo-riff-zookeeper | ClusterIP | 10.110.26.126 | <none> | 2181/TCP | 5h | svc/demo-riff-zookeeper |
| svc/kubernetes | ClusterIP | 10.96.0.1 | <none> | 443/TCP | 38d | svc/kubernetes |
| <hr/> | | | | | | |
| NAME | DESIRED | CURRENT | UP-TO-DATE | AVAILABLE | AGE | NAME |
| deploy/demo-riff-function-controller | 1 | 1 | 1 | 1 | 5h | deploy/demo-riff-function-controller |

Functions aren't free however.

Well, ok, once you get
beyond the free tier at least...

Then some fractional cost there after.

First million (or two) requests are free*.

* Additional fees may apply.

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

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.

What use cases fit well with functions?

Rapidly evolving
business requirements...

Stateless workloads.

Infrequent (or sporadic) requests.

Variable scaling needs.

Asynchronous workloads.

Easy to parallelize.

IoT, machine learning, log
ingestion, batch processing.

“Conversational UIs” aka digital
assistants like Alexa.

CI/CD automation, chat
integration, stream processing...

Website back end services such as
logging, post handlers, auth.

Event processing, monitoring,
notifications, alerting.

Security scanning.

Very valuable tool.

It isn't a good fit for every workload.

But you knew that.

Spring Cloud Function.

A Spring take on functions!

Abstract away the runtime target.

Same code can be an endpoint, a
task, a stream processor...

Provide a unified programming
model across serverless providers.

Multicloud for the win!

As well as running locally or
on your PaaS of choice.

Bring the Spring Boot features you
know and love to serverless!

Focus on writing the business logic.

Utilizes core abstractions from
`java.util.function.`

Function, Consumer, Supplier.

Can optionally use Reactor's **Flux** as
parameters and return types.

Meaning you can use reactive
approaches like non-blocking IO.

Supports compiling String based
lambdas into function instances.

If this is just based on standard Java
classes, why should I use it?

Inversion of control.

Allows you to decouple the code
from the deployment profile.

Just add the right dependency.

Do you want to deploy your
function as a REST endpoint?

spring-cloud-function-web

Need that function to be a
stream processor instead?

spring-cloud-function-stream

Goal - remove the boilerplate.

Developers focus on the function.
Let the framework handle the rest.

Implement (and test) the code apart
from the target environment.

riff.

riff is for functions.

<https://projectriff.io>

FaaS for Kubernetes.

Executing functions in
response to events.

Developers focus on two things:

The function itself.

And the topic that triggers the function.

Event stream might be low frequency, might be continuous.

Or unpredictable.

Built on the experience of Spring
Cloud Streams and Data Flow.

Functions can be polyglot.

Java, JavaScript, shell, Go and Python.

Just a container running on K8s.

Scale up and down as needed.

riff function controller monitors
event broker for lag.

Once it sees a certain
amount of lag, will scale up.

By default, single instance scales down to zero after 10 seconds idle time.

Yes, it is configurable.

ICE - CAP for serverless.

Immediate. Consistent. Efficient.

Pick two.

What are your concerns? Resource utilization? Immutable containers?

Set of comprises.

If we keep containers running, we're
paying for idle resources.

If we inject code into a container,
we lose immutability.

If we launch containers on demand,
we have slow startup experience.

Many platforms use dynamic loading.

But the goal with riff is to give *you*
control over the tradeoffs.

Functions are packaged as containers.

Functions and topics are
just K8s resources.

In other words, managed via
kubectl just like a pod.

riff is thin.

Relies on Kubernetes Custom Resource Definition extension.

riff function controller monitors
event lag, scales as necessary.

Event broker is Kafka, pluggable event brokers are on the roadmap.

riff build - creates the Docker image.

riff apply - deploy the
function on Kubernetes.

Or `riff create` to init/build/apply.

Your code is added to a base
image specific to the invoker.

Just an instance of inversion of control. Don't call us, we'll call you.

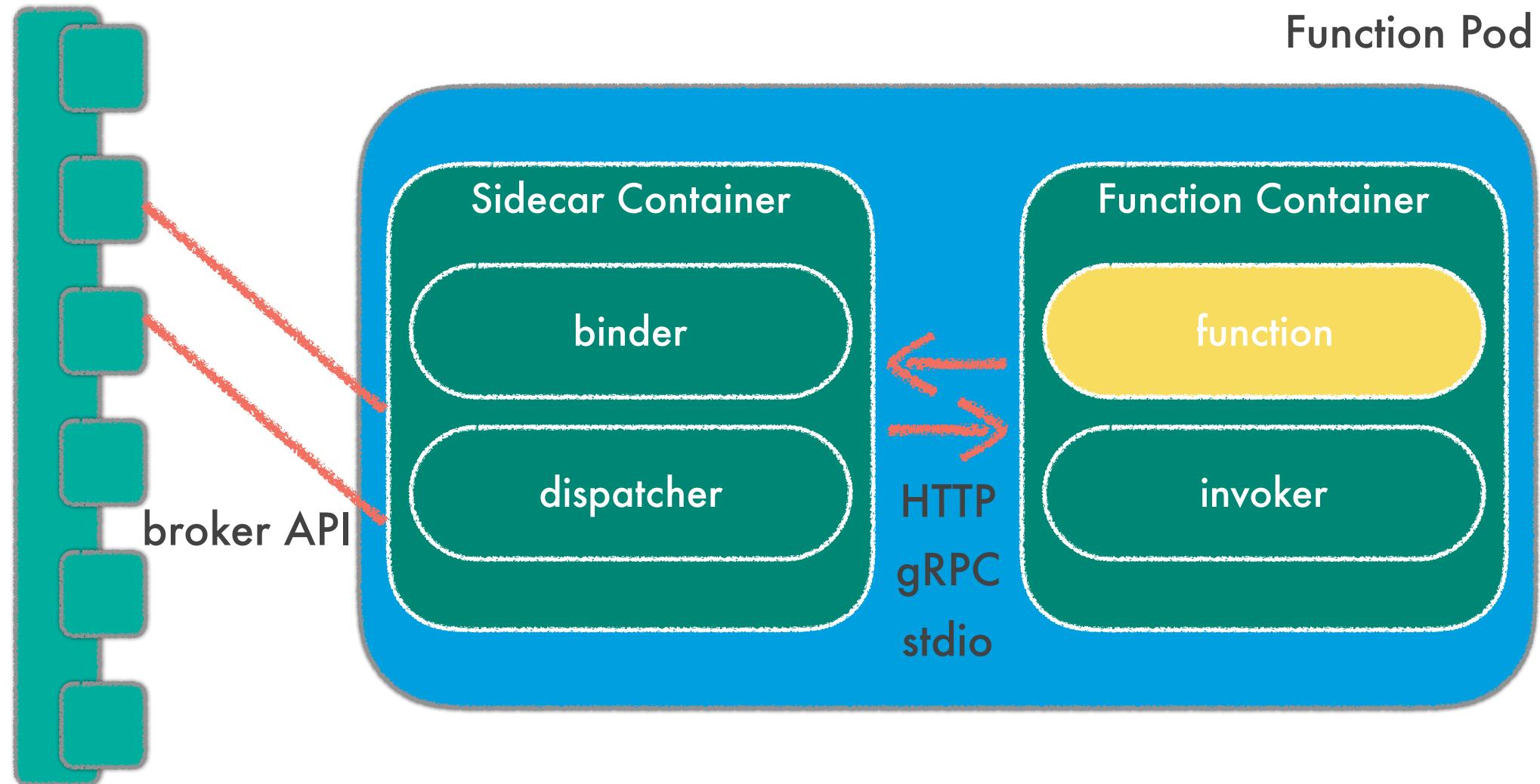
Sidecar contains the binding to the event broker. Aka proxy pattern.

Multiple implementation of the
function containers...

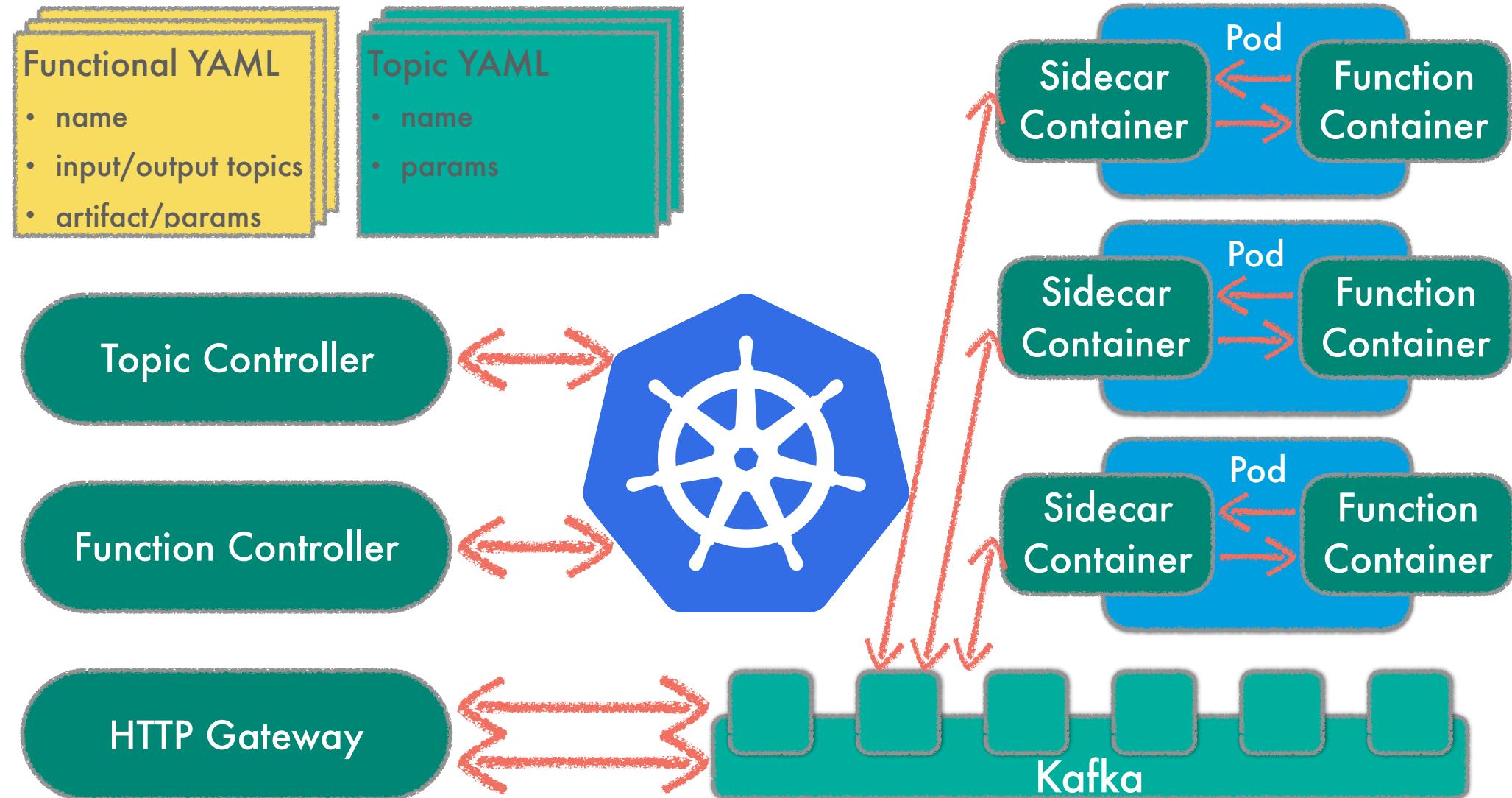
But only one implementation
of the sidecar.

Can communicate to the function
invoker over HTTP, stdio, gRPC...

Function Pod



Event Broker



Very new, evolving rapidly.

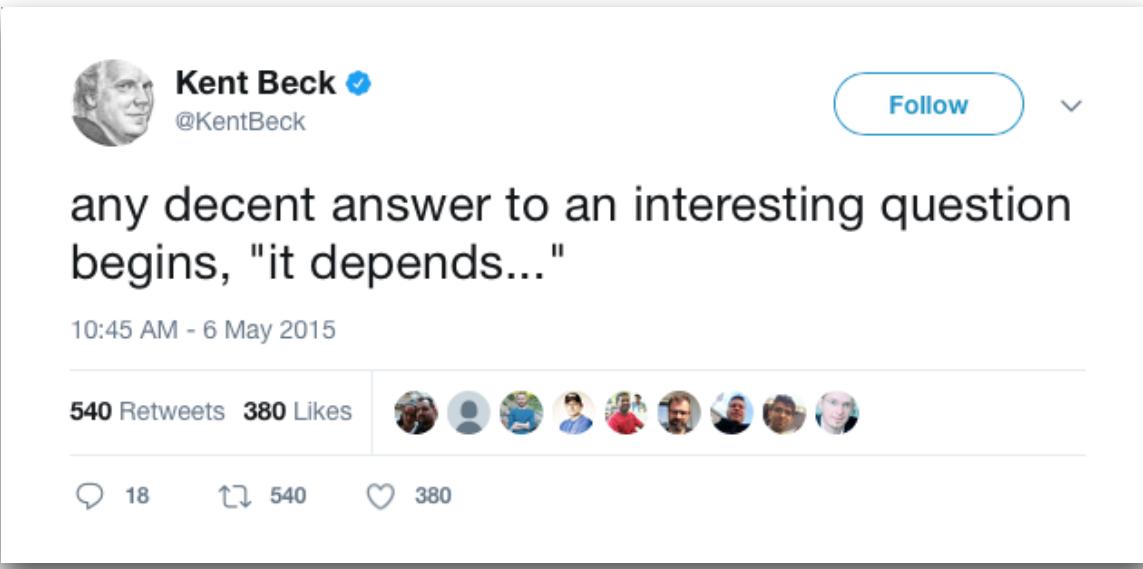
Agnostic layer for FaaS. Run it
on premise or public cloud.

All the public cloud providers have their own FaaS implementation.

Lambda, Google Cloud Functions,
Azure Functions.

Lots of options in this space...

Which approach should *you* choose?



A screenshot of a Twitter post from user Kent Beck (@KentBeck). The post features a profile picture of Kent Beck, his name, a blue verified checkmark, and his handle @KentBeck. To the right is a 'Follow' button and a dropdown menu icon. The tweet itself reads: "any decent answer to an interesting question begins, "it depends..."". Below the tweet is the timestamp "10:45 AM - 6 May 2015". A horizontal line follows, with "540 Retweets" and "380 Likes" on the left, and a row of small circular profile pictures on the right. At the bottom, there are engagement icons: a speech bubble with "18", a retweet icon with "540", and a heart icon with "380".

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

There is no **one** right answer here.

中

！

三

Not all workloads fit in each bucket.

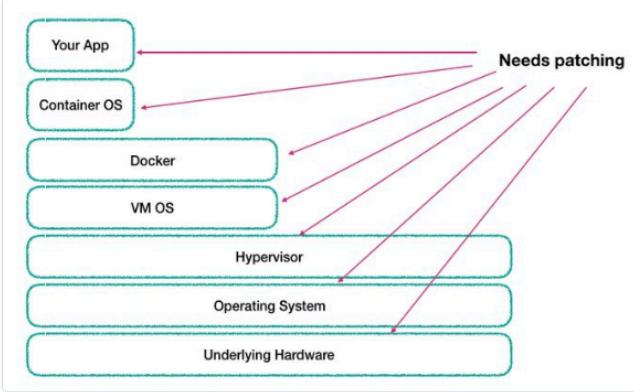
We need to pick the right runtime for each workload.

There are tradeoffs!

But you knew that.

 Sam Newman 
@samnewman

I was in the middle of creating this slide (wrt patch hygiene) and had to stop half-way through and ask myself - aren't we all just making this worse?



12:35 PM · Jan 14, 2018

1,071 Retweets 1,640 Likes

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

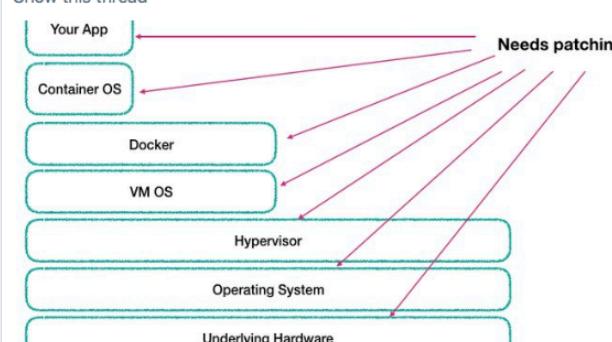
 Josh Long (龙之春, जोश) ✅
@starbuxman

This is exactly why an integrated platform like [@cloudfoundry](#) & a public cloud offering is valuable: everything in that slide (except your app) is either managed centrally by the platform or its managed by somebody else who have a vested interest in doing so well.

Sam Newman ✅ @samnewman

I was in the middle of creating this slide (wrt patch hygiene) and had to stop half-way through and ask myself - aren't we all just making this worse?

Show this thread



```
graph TD; YourApp[Your App] -- "Needs patching" --> ContainerOS[Container OS]; ContainerOS -- "Needs patching" --> Docker[Docker]; Docker -- "Needs patching" --> VMOS[VM OS]; VMOS -- "Needs patching" --> Hypervisor[Hypervisor]; Hypervisor -- "Needs patching" --> OS[Operating System]; OS -- "Needs patching" --> HW[Underlying Hardware]
```

4:03 AM · Feb 2, 2018

17 Retweets 35 Likes

Reply Share Like Message

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

Containers

Developer Provided

Container

Container Scheduling

Primitives for Networking, Routing, Logs and Metrics

Platform

Application

Container

Container Images

L7 Network
Logs, Metrics, Monitoring

Services
Marketplace

Team, Quotas & Usage

Serverless

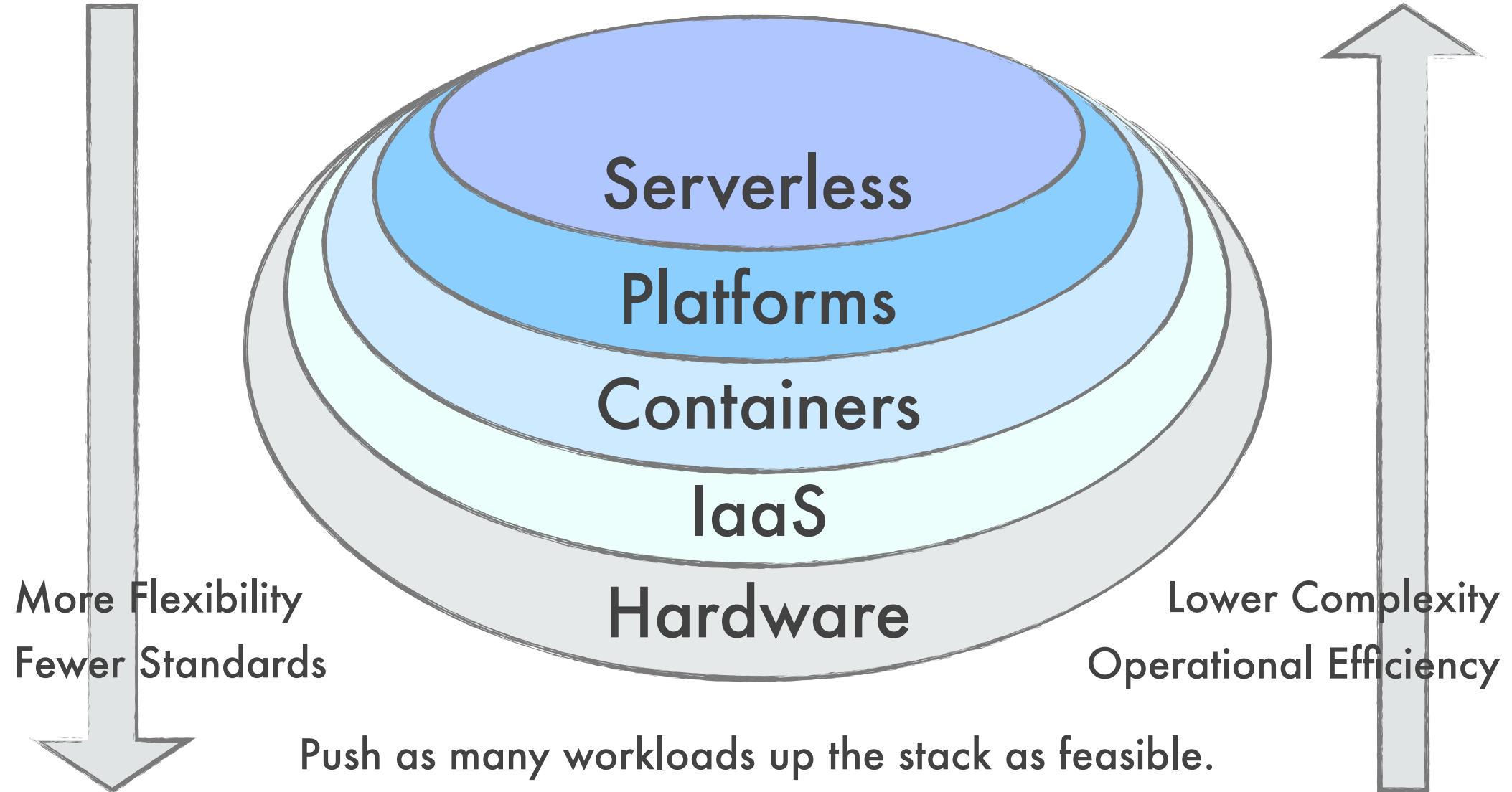
Function

Container

Function Execution
Function Scaling
Event Stream Bindings

IaaS

Different levels of abstraction.



With great flexibility comes
great responsibility.

Development complexity.

Do you want developers
thinking about OS patches?

Or just pushing apps?

The prevailing mythology is you should "containerize" everything and give it to a container orchestrator to run, but why?
They had one problem, "Run an app".
Now they have two, "Run a container that runs an app" and "maintain a container".

— Adam Wright

<https://news.ycombinator.com/item?id=16332184>

Operational efficiency.

How many operators does it take to
manage a bespoke environment?

Flexibility vs. standardization.

Guardrails aren't shackles.

They are designed to protect you.

How do you want to
allocate *your* resources?

Choose the option that fits the problem.

Push as many workloads as you can
up the abstraction hierarchy.

The Cloud is full of FUD.

And developers chasing
the new hotness.

Be strategic.

Hope is not a strategy!

But it is what rebellions are built on.

We need to be deliberate.

Understand the benefits - and the limitations - of these approaches.



Do the right thing.

Good luck!

Questions?

Thanks!

I'm a Software Architect, Now What?
with Nate Schutta

Presentation Patterns
with Neal Ford & Nate Schutta

Modeling for Software Architects
with Nate Schutta



Nathaniel T. Schutta
@ntschutta