Navigating Microservice Architectures

Jason Dudash
Open Source Solutions Architect and Builder
2019

github.com/dudash
dudash@redhat.com

DISTRIBUTED COMPUTING IS GREAT

mission critical systems are often distributed out of necessity

There are many “ilities” you can get by distributing your software

- Scalability
- Performance
- Reliability
- Resiliency
- Extensibility
- Availability

For example: **if you lose a rack of computers**
a second rack’s set of redundant services will take over control of that *chopper*

redhat.

BUT DISTRIBUTED COMPUTING IS HARD

Software

ROBOT KILL-CHOPPER GOES ROGUE above Washington DC!

'Software error' sends droid off military reservation

Air Force personnel inspect the wreckage of a Predator drone that crashed during a training flight in May 2013 near Creech Air Force Base in Nevada. (U.S. Air Force)

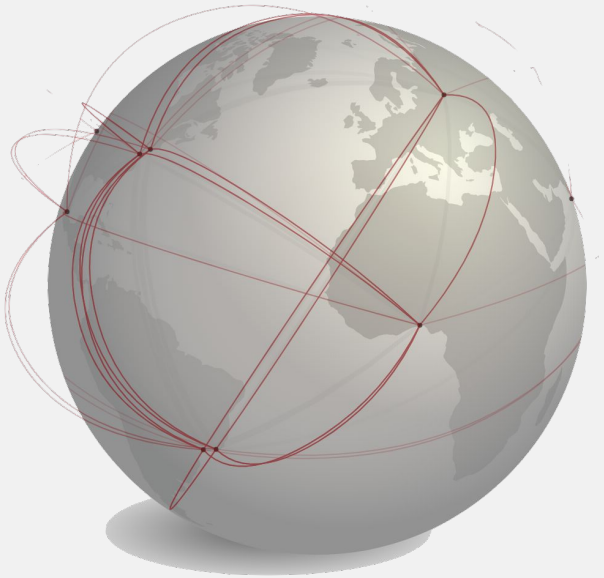
A Predator drone that crashed during a botched landing in April 2010 in Victorville, Calif. (U.S. Air Force)

WHY IS IT HARD?

it’s just groups of networked computers right?

The fallacies of distributed computing

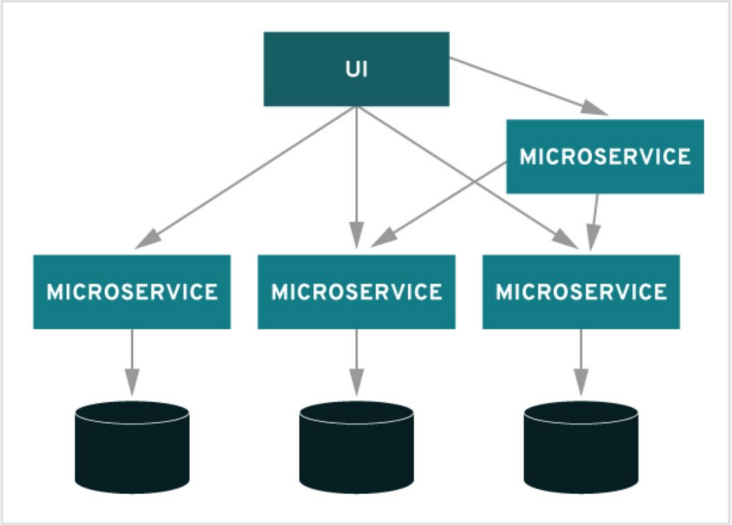
- The network is reliable.
- Latency is zero.
- Bandwidth is infinite.
- The network is secure.
- Topology doesn't change.
- There is one administrator.
- Transport cost is zero.
- The network is homogeneous.



WHAT ABOUT MICROSERVICES ?

microservices rely on distributed computing but additionally...

- Manage their own data (i.e. no central database)
- Do one thing and do it well
- What once might have been function call is now across the network
- Are polyglot - many different runtimes could be in the mix
- Can be built, versioned, and delivered independent of each other
- Might not be able to retain state (aka stateless)
- Typically provide an API which is used to communicate with it (e.g. REST)

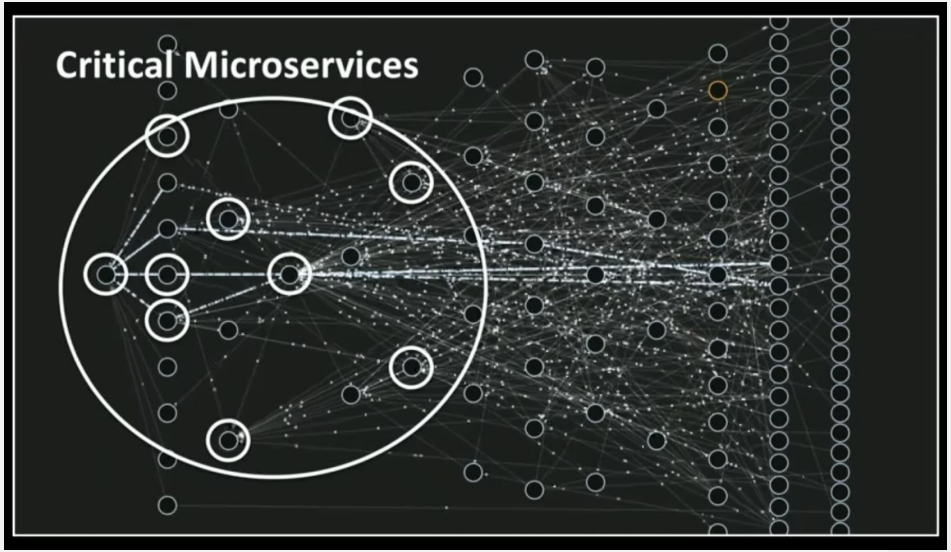


SCALE WILL MAKE IT HARDER

take Netflix for example...

This is where the challenges of distributed computing really start to affect your microservices

- What to do when services fail?
- How do I monitor health?
- How to collect and view logs?
- How to collect metrics and automate?
- How to scale for demand?
- How to deploy?
- How to ensure consistency?



PREREQUISITES (YOU MUST BE THIS TALL TO RIDE)

Some important considerations:

- Isolate the data
- Design to account for failures
- APIs are critical - monitor and manage them
- Plan to scale and do load testing
- Cut out the manual steps
- Always be releasing (CI/CD)
- Consider security for network and services
  - Access control, encryption, storage
- Storage as a service (dynamic provisioning)
- Deploy in containers

It’s a lot, but it’s OK to get there incrementally

SO WHY WOULD YOU WANT MICROSERVICES

don’t adopt it because it’s popular… do it for

AGILITY

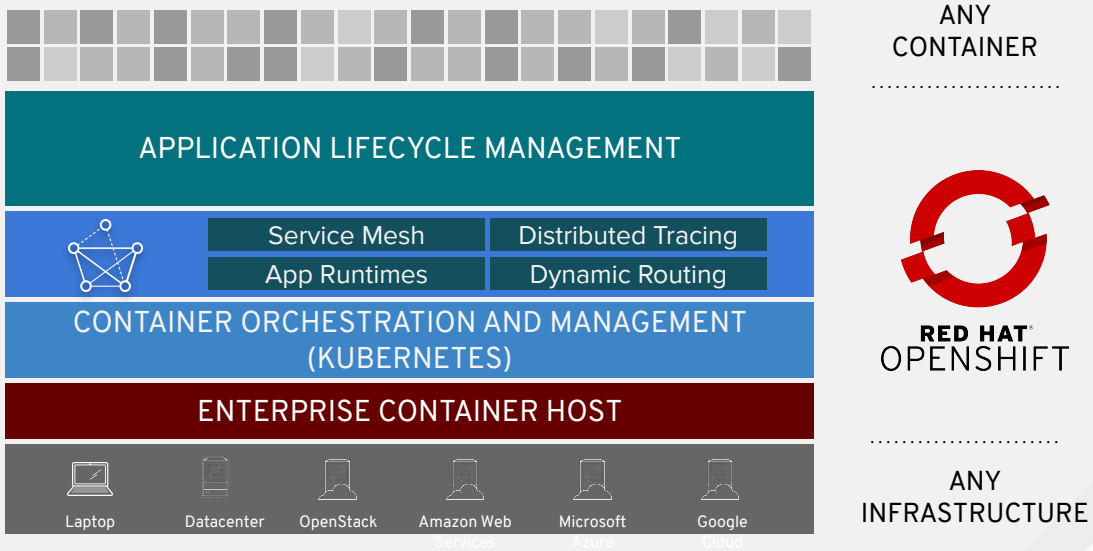
Code has **no business value** until it's deployed.

THEN HOW TO DEAL WITH THESE CHALLENGES?

take advantage of a platform built to scale

Leverage pre-built blocks of capability:

- Service discovery
- Load balancing
- Dynamic storage (multiple types)
- Log aggregation
- Continuous deployment
- Routing/DNS
- Multi-tenancy
- Distributed tracing
- Circuit breaking
- Traffic mirroring
- Mutual TLS, and more...



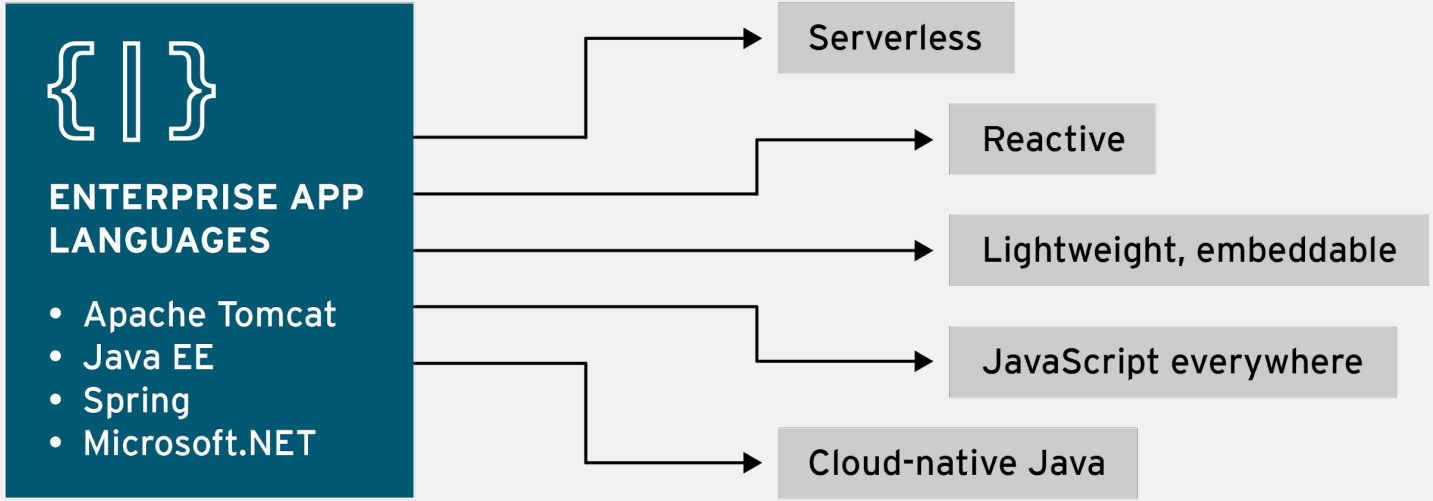
redhat

DEMO

let’s take a look at how a platform can help for **operational** aspects of microservices

MICROSERVICES LEAD TO MORE NEEDED BITS

Expanded use of languages, frameworks, and runtimes

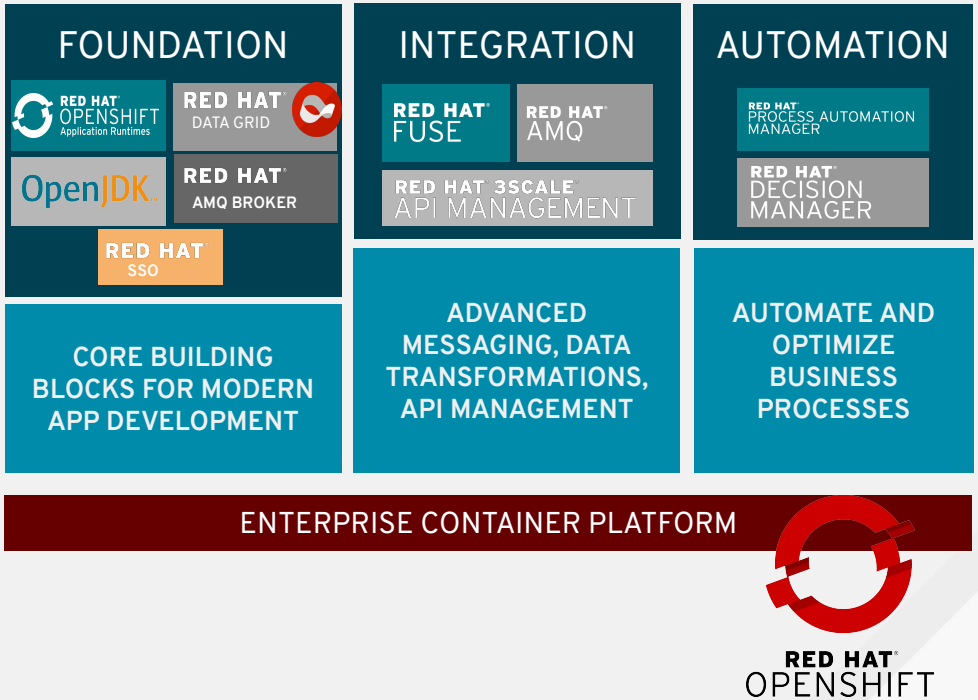


DON’T LEAVE DEVELOPERS OUT

Leverage core middleware and bundled with the platform and engineered to be ***container-native***

Leverage pre-built blocks of capability:

- Single-sign-on (service security)
- CI/CD pipelines
- Integration patterns
- IDE Tooling
- Application metrics
- API access control
- API rate limiting
- Cloud-native middleware



DEMO

let’s take a look at how a platform can help for **developing** of microservices

BACK TO THE “ILITIES”

If you’ve got the time, let’s do some more demos
and how a platform can satisfy non-functional requirements

INSTRUMENT, MONITOR, AND PUBLISH INTERFACES

Building APIs are more than just coding a data model in REST...

CONTROL

DEMO

INTEGRATION IS A REUSABLE PATTERN

Data will need to be transferred/transformed - don’t write the same code again and again.

REUSE

DEMO

SECURE FROM THE INSIDE AND ON OUTSIDE

mTLS, JWT, SAML, OAuth, OpenID Connect, Identity Brokering, User Federation...

SECURITY

DEMO

REMOVE THE BOTTLENECKS AROUND DATA

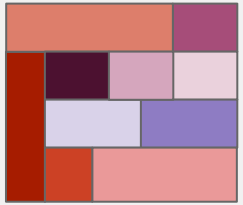
Session caching, database caching layer, NoSQL, fast distributed data plane...

RESPONSIVE

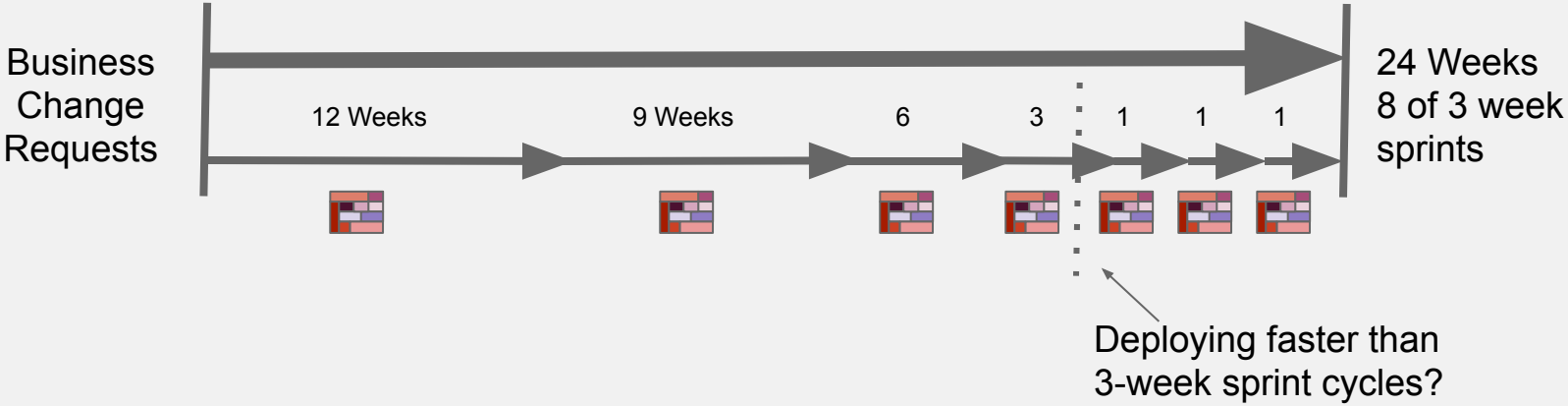
DEMO

CLOSING THOUGHT: *COULD I DO 10 RELEASES A DAY?*

technology and teams must be in place for this to be possible
(automation, containers, pipelines, blue/green, high trust)



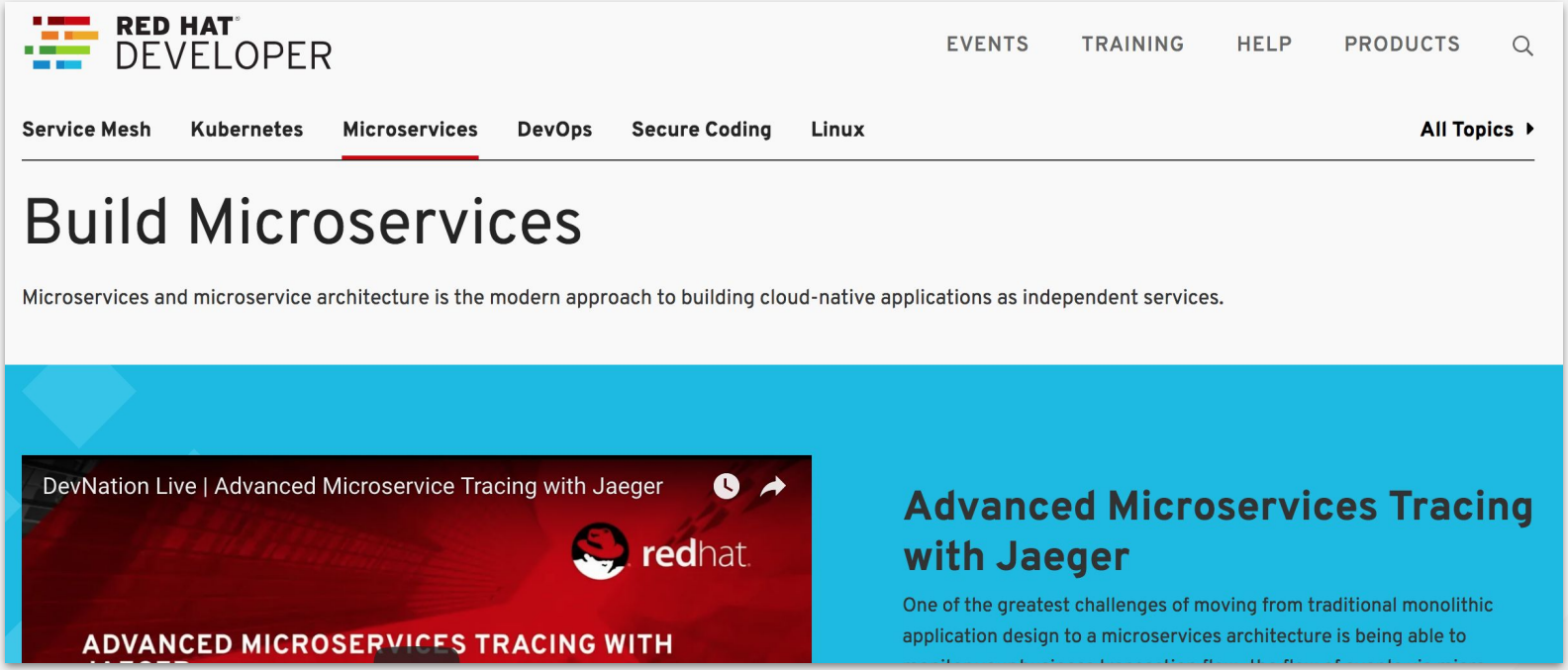
Monolithic System



Patches to your application as well as your “stack” are also deployments. Your stack consisting of the OS, JVM, runtime engine (e.g. Tomcat, JBoss EAP, Node.js), frameworks (e.g. Spring) all should be regularly patched via your CD Pipeline

IF YOU WANT TO DIG DEEPER

email me! ...or dig in on your own at developer.redhat.com



IF YOU DIDN’T GET THE LIVE DEMOS

here are a few videos you might like to watch

Service Mesh - Observability (Kiali)
https://vimeo.com/345028718

Service Mesh - Security (mTLS)
https://vimeo.com/343318228

Service Mesh - Reliability (Circuit Breaker & Bulkheads)
https://vimeo.com/343313938

OpenShift Autoscaling
https://vimeo.com/253515068

OpenShift Blue/Green Deployments
https://vimeo.com/272622087

OpenShift oc new-app
https://vimeo.com/287107850

TODO Middleware and Developer Videos

RHMI

odo / VSCode Plugin

CodeReady Workspaces

Camel-K


X0
B
A
-LIFE-
X0
X0
IT'S DANGEROUS TO GO
ALONE! TAKE THIS.

redhat.

SNAP THIS

THANK YOU

github.com/dudash

dudash@redhat.com

plus.google.com/+RedHat

facebook.com/redhatinc

linkedin.com/company/red-hat

twitter.com/RedHat

youtube.com/user/RedHatVideos

TITLE

subtitle

0-ILITY

DEMO NOTES

The self-service and automated aspects of a platform are important for doing of application level operations.

Let's look at health checks and service recovery

Self-service developer capabilities can greatly enhance productivity and testing scenarios

Let’s look at a front-end developer use case of rapid building, containerization, deployment, and routing

In the earlier case I showed, we would typically hook up API mgmt to the REST endpoints of those simple services. And we would then secure them with some combination of JWTs, single sign on, and authorization.

But that's not an exciting demo, so let's take a look at an emerging capability called a service mesh