

Self-service, automation, and other keys to modernizing application development

Jason Dudash
Principal Solutions Architect
Emerging Technology, Red Hat

June 2020



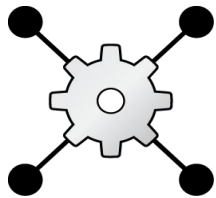
@dudash



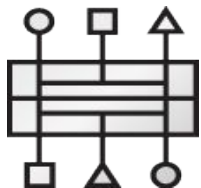
@dudashtweets

A DIGITAL EVOLUTION

Service Endpoints



Web Services

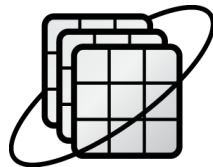


APIs

Architecture

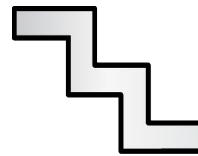


Monolith

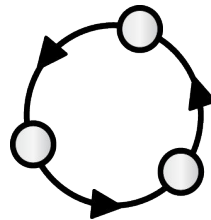


Microservices

Development Process



Waterfall

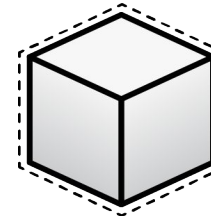


Agile + CI/CD

Deployment

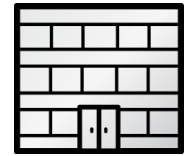


Server/VM



Container

Infrastructure



Data Center



Cloud

Why modernize?

WHAT DO AGENCIES **NEED**?

CHANGE FASTER

Increase the speed of change by **updating applications** to adapt to the markets and customers

DEVELOP FASTER

Increase the speed of **developing new applications** to address new business opportunities

DELIVER FASTER

Increase the speed of **app delivery** of existing and new applications to your customers

INNOVATE FASTER

Increase the speed of **innovation** across the organization to the pace that your business demands

Software has no business value until it's deployed

YOUR DIFFERENTIATION DEPENDS ON YOUR ABILITY TO DELIVER AND INNOVATE

Cloud-native
Applications



AI & Machine
Learning



Blockchain



Internet of
Things



Innovation
Culture



CLOUD-NATIVE APP DEV

A MODERN APPROACH TO BUILDING AND RUNNING APPLICATIONS



Empowers you to build and run scalable applications in dynamic environments

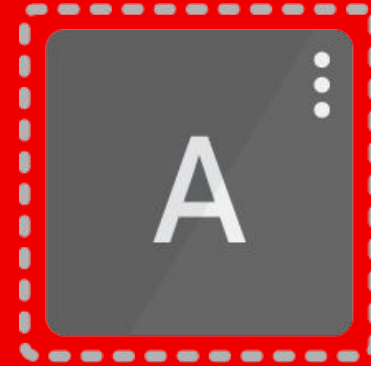
IT'S PART OF DIGITAL EVOLUTION

TRADITIONAL (LEGACY)

CLOUD-NATIVE (MODERN)

Server-centric	Container-centric
Scale up vertically	Scale out horizontally
Tightly coupled monolith	Loosely coupled and service-based
Infrastructure-dependent	Portable across infrastructure
Waterfall, semi-agile, and long delivery	Agile and continuous delivery
Local IDEs & developer tools	Cloud-based, intelligent tools
Siloed dev from ops, QA, and security teams	DevSecOps, NoOps, and collaboration

But we can't just
leave legacy
behind and go
cloud-native...



EXISTING APP

Valuable
Lots of investment
High complexity

A TYPICAL DIGITAL DARWINISM



Start small and start now

KEY STEP IN A SUCCESSFUL MODERNIZATION

AUTOMATION

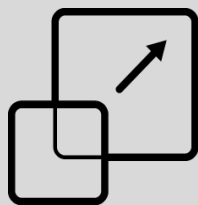
Think about what is time consuming for you today
Find your manual steps that are prone to failure

Stop and think:

Where have you accrued
technical debt?

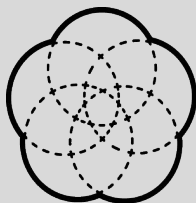


TYPICAL CONSEQUENCES OF EXCESS TECHNICAL DEBT



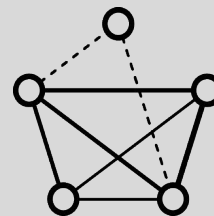
PLATFORM(S) STRUGGLE TO SCALE

Your architecture doesn't give you the agility you need to react to demand. It's also expensive in cost for underlying infra.



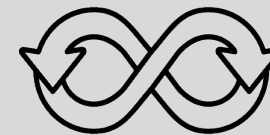
APPS & SERVICES ARE NOT COORDINATED ACROSS ENVIRONMENTS

You can't support rapid deployment & updates. Installation challenges.. Differences in envs results in hard to debug issues.



INSECURE APPS AND ENDPOINTS

You lack the security and analytics on exposed endpoints. No way to get metrics on usage. Apps have exploitable (or unknown) security CVEs



CUSTOMER EXPERIENCE IS INCONSISTENT

Due to difficulty in connecting systems to data. Inability to transform data. Challenges with adding features or adding new lines of business..

IS THERE ALWAYS ONE PERSON WHO SOLVES ALL THE PROBLEMS?

THE HERO

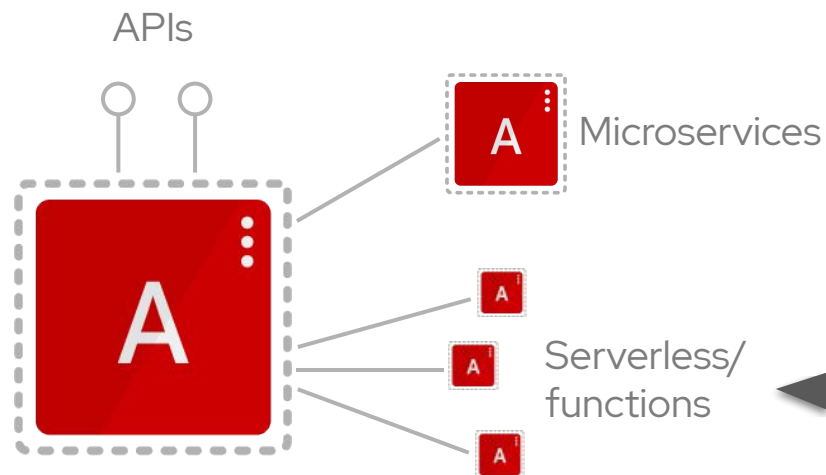
They're a single point of failure

Often they're only person who can accomplish key tasks

Paths for modernization

COMMON PATHS FOR MODERNIZATION

THESE ARE COMMON PATTERNS TO USE



HYBRID APP

Phased approach in migration

Lift and Shift

Leave the architecture alone but modernize the deployment platform. Can be used for performance increases by allowing for deploying to better hardware. Can be used to accelerate deployments and improve processes by leveraging platform automation. **Fast Monoliths.**

Refactor and Augment/Extend

Find parts of the architecture that are sources of pain - refactor. Build **new capability in microservices** with well-defined APIs. Wrap legacy software too brittle to change with **adapter layers**.

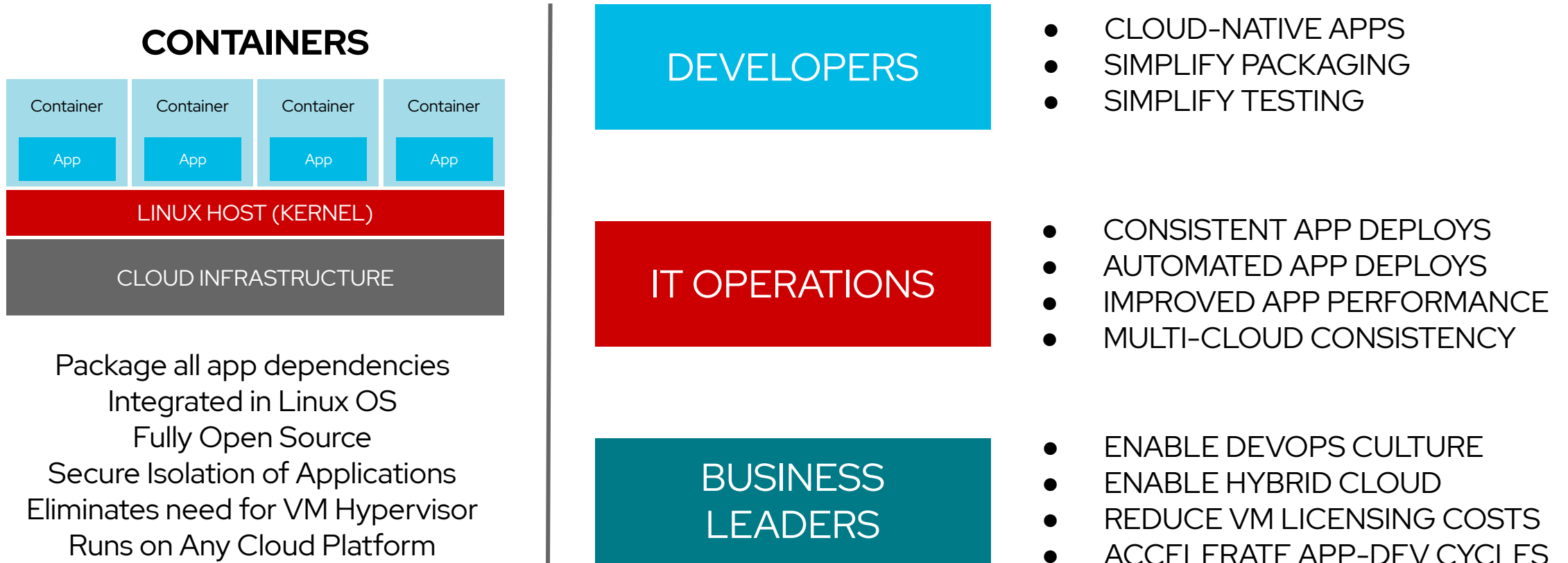
Rewrite/Replace

Create new functionality to **replace existing functionality**. Likely expensive and time-consuming. Typically only recommended when legacy vendors go away or a major skills gap forces it.

What does a modern
application architecture
look like?

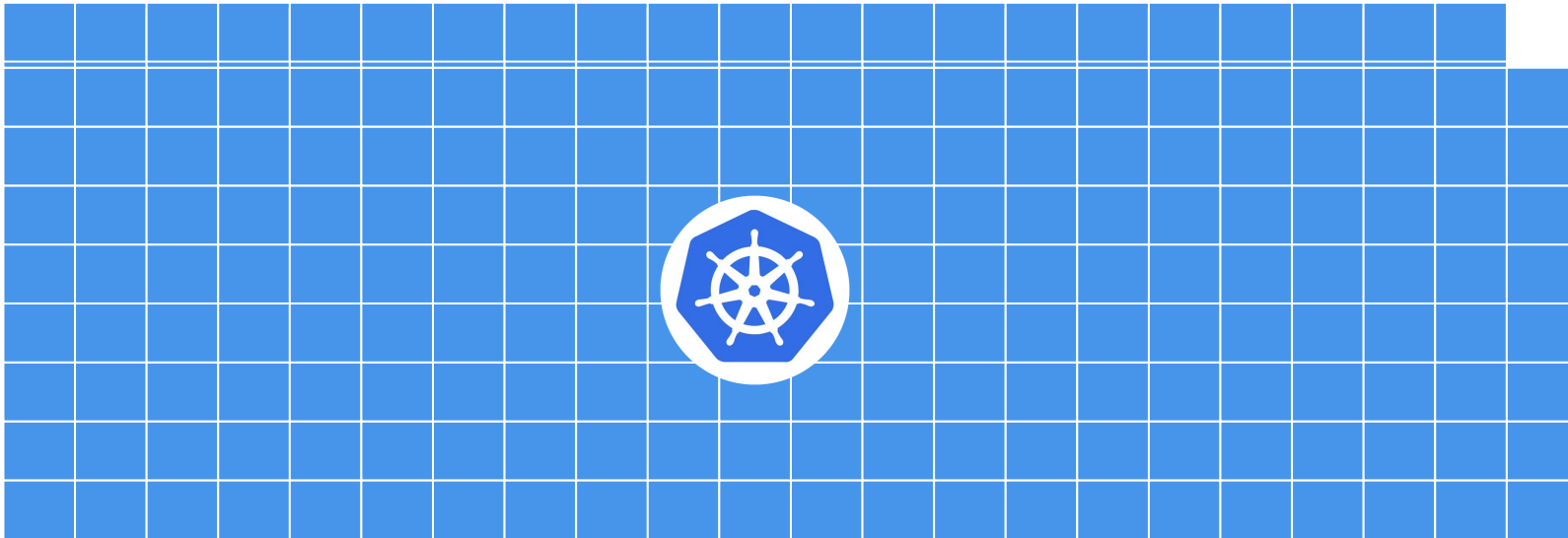
CONTAINERS UNDERPIN THE ARCHITECTURE

CONTAINER BENEFITS FOR MULTIPLE TEAMS



KUBERNETES GIVES IT SCALE

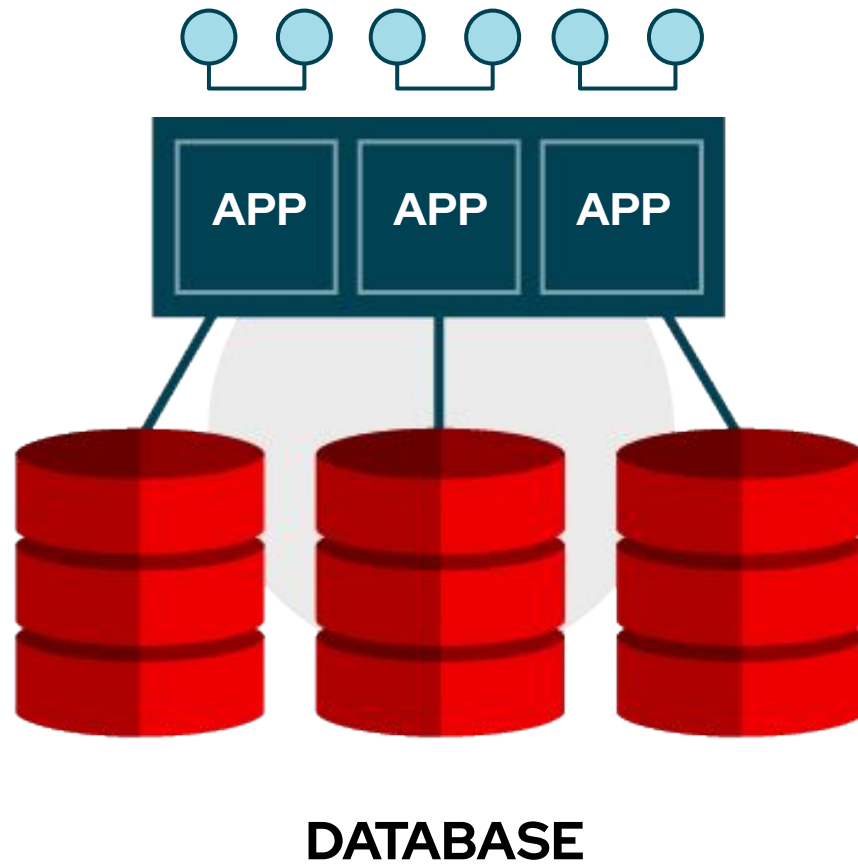
DEPLOYMENT AUTOMATION, AUTOSCALING, HEALTH CHECKING



By any objective measure, the industry has converged on Kubernetes as the container orchestration engine of choice

SERVICES WITH WELL DEFINED APIS

DEPLOYED IN CONTAINERS, OWNING THEIR OWN DATA



DESIGN APIS THAT ARE CONSISTENT AND REUSABLE BEFORE YOU CODE

API FIRST

Critical to managing internal reuse

Fosters innovation and enables good developer experiences

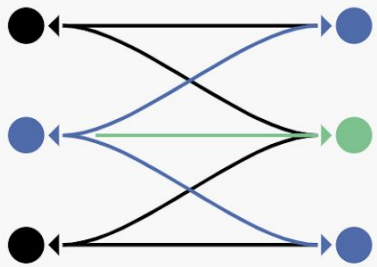
MICROSERVICES AREN'T A SILVER BULLET

MICROSERVICE EVERYTHING

Understand if they fit your team & use case
Sometimes it makes sense to keep your monolith
Microservices bring operational complexity

ISTIO MANAGES THE MICROSERVICES

ADDING OBSERVABILITY, SECURITY, AND CONTROL



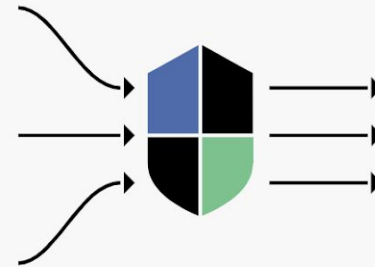
Connect

Intelligently control the flow of traffic and API calls between services, conduct a range of tests, and upgrade gradually with red/black deployments.



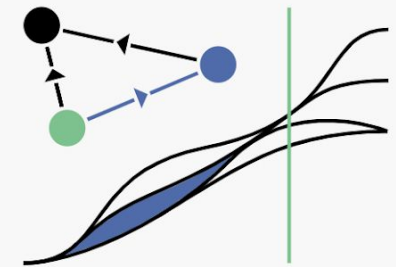
Secure

Automatically secure your services through managed authentication, authorization, and encryption of communication between services.



Control

Apply policies and ensure that they're enforced, and that resources are fairly distributed among consumers.



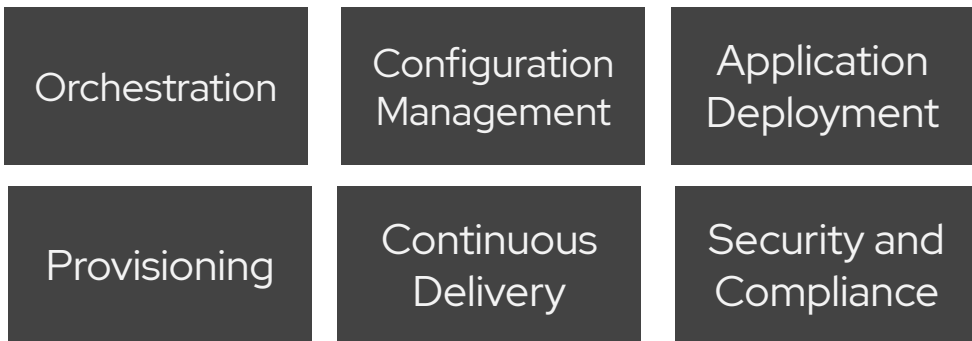
Observe

See what's happening with rich automatic tracing, monitoring, and logging of all your services.

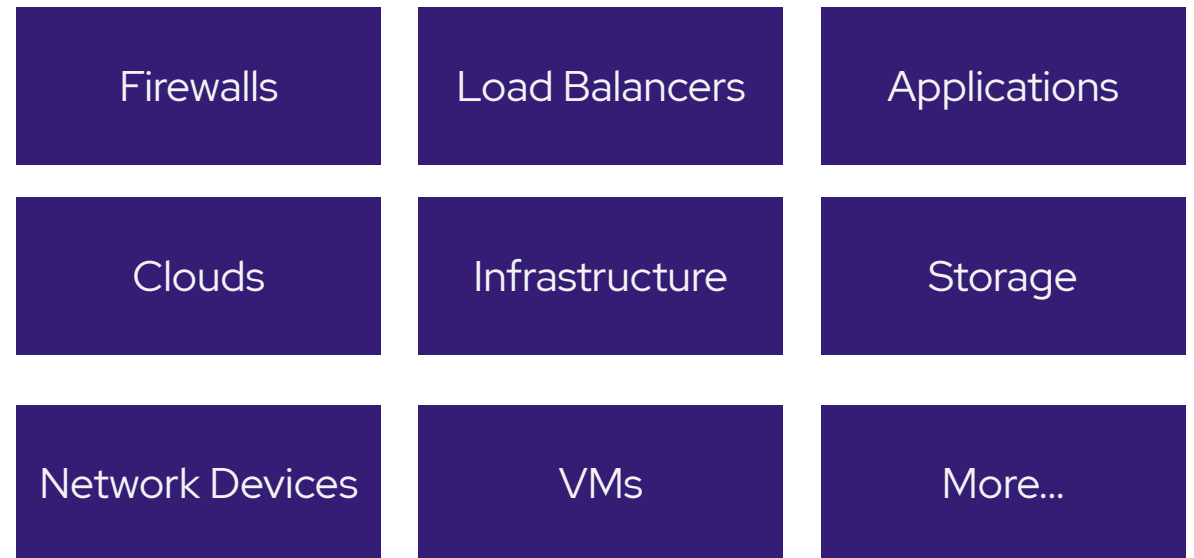
ANSIBLE AUTOMATES EVERYTHING ELSE

AUTOMATION BEYOND JUST THE CONTAINER PLATFORM

Do this...



On these...



And you can build operators to run Ansible in Kubernetes too!

OPEN SOURCE APPLICATION STACK

DEVELOPERS



IDE

CLI

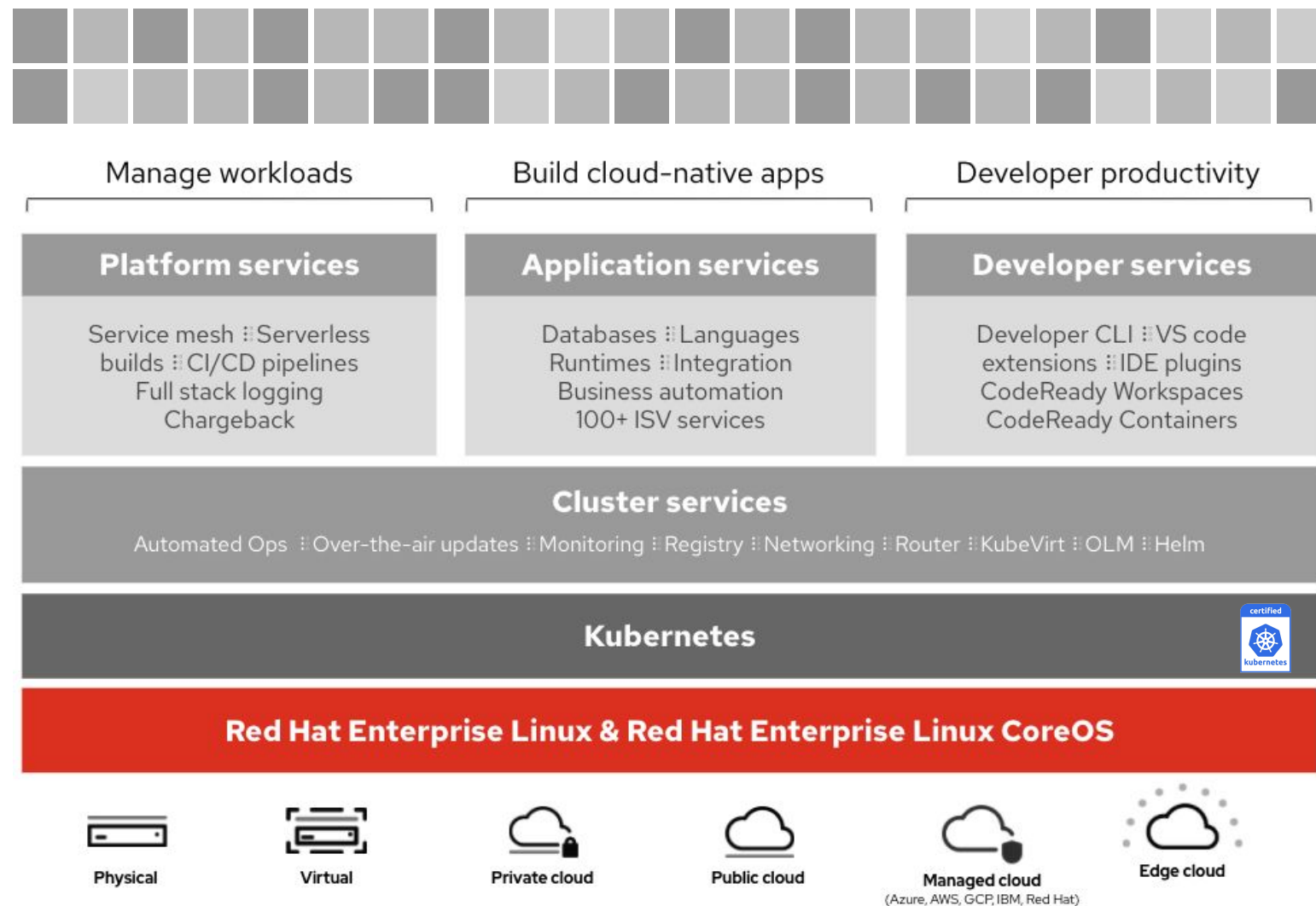
OPS
TEAMS

DASHBOARD

API

SECURITY

WEB UI



AN ESSENTIAL CHARACTERISTIC OF CLOUD COMPUTING
(NIST SP 800-145)

SELF SERVICE

Start doing this now, start small

Give scoped autonomy (with role based access control)

Application concerns

APPLICATION

- Choice of language/framework
- Self-service / productivity
- Low mem, fast startup
- Integrations framework



Microservices

SERVICE TO SERVICE

- Network resilience
- Service security
- Policy enforcement
- Metrics/Observability
- Load balancing



Infrastructure concerns

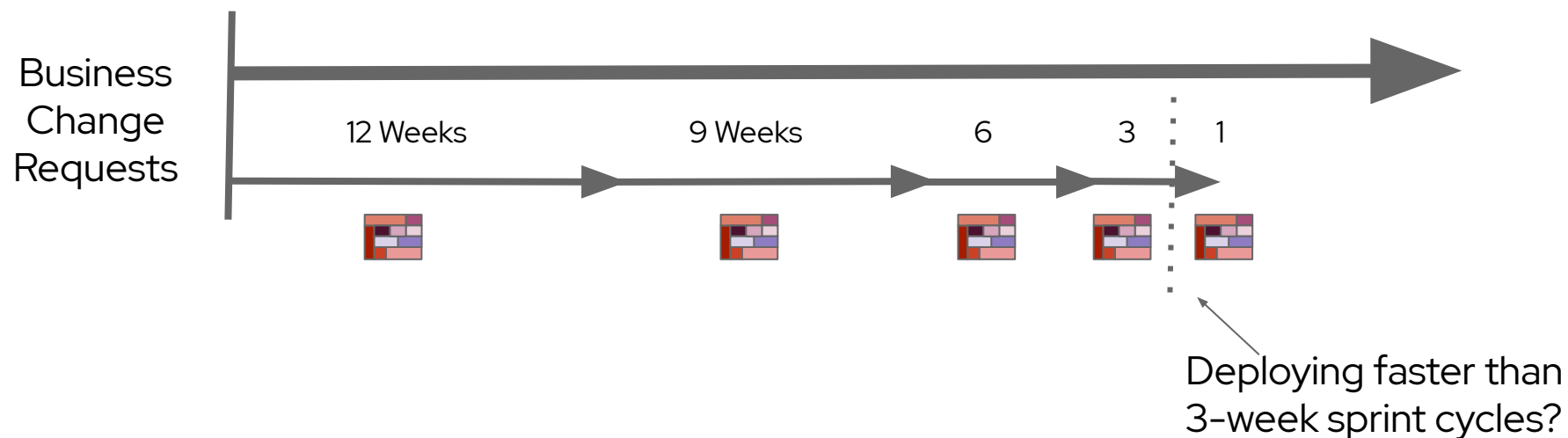
DEPLOYMENT PLATFORM

- Reliability
- Instance placement
- Scaling/autoscaling
- Resource usage
- Job scheduling
- Distributed Logging



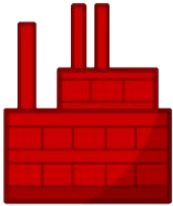
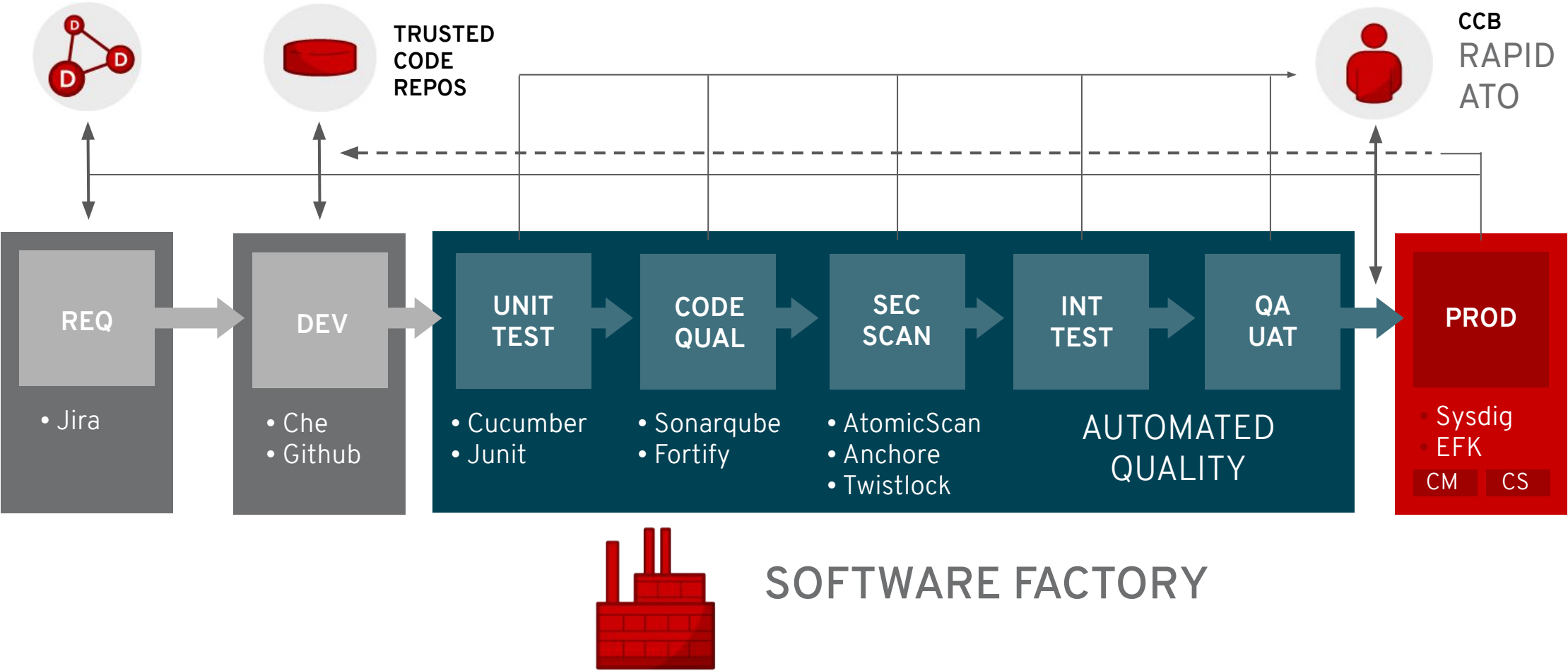
ABILITY TO DO MULTIPLE RELEASES A DAY

TECHNOLOGY AND TEAMS IN PLACE FOR A PROCESS ALLOWING RAPID DELIVERY
(AUTOMATION, CONTAINERS, PIPELINES, BLUE/GREEN, HIGH TRUST)



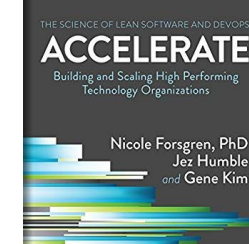
Patches to your application and your "stack" are deployments and should be regularly patched via your CD Pipeline




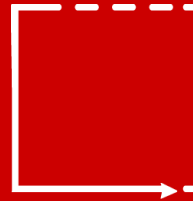
VIA A TRUSTED SOFTWARE FACTORY



SOFTWARE FACTORY

MEASURABLE VALUE IN MODERNIZATION




	 <p>↓ LEAD TIME FOR CHANGE</p>	 <p>↑ DEPLOYMENT FREQUENCY</p>	 <p>↓ MEAN TIME TO RECOVERY (MTTR)</p>	 <p>↓ CHANGE FAILURE RATE*</p>
	Measures of MARKET AGILITY		Measures of RELIABILITY	
WHAT	Time from code committed to deployed to a state of "done"	Proxy for batch size, how often does an app iterate	How long it takes systems to recover from failures	Percentage of deployments requiring rollback and/or fixes
WHY	Shorter is better. Enables faster feedback cycles and makes you better able to adjust to the marketplace	Indicator of batch size. Smaller batch size leads to more market agility	Critical to ensure that we aren't speeding up delivery at the expense of negative customer impacts	*Secondary indicator of stability but also representative of requirements analysis

Closing thought

Thank you

Red Hat is the world's leading provider of enterprise open source software solutions. Award-winning support, training, and consulting services make Red Hat a trusted adviser to the Fortune 500.

 [linkedin.com/company/red-hat](https://www.linkedin.com/company/red-hat)

 [youtube.com/user/RedHatVideos](https://www.youtube.com/user/RedHatVideos)

 [facebook.com/redhatinc](https://www.facebook.com/redhatinc)

 twitter.com/RedHat

RESOURCES / RESEARCH

1. Cloud Native - https://www.eclipse.org/community/eclipse_newsletter/2019/september/microprofile.php
2. Incremental Path - <https://developers.redhat.com/blog/2017/02/21/an-incremental-path-to-microservices/>
3. Gov't Modernize - <https://fcw.com/articles/2019/06/18/garland-pal-it-modernization-comment.aspx>
4. Accelerate - <https://www.amazon.com/Accelerate-Software-Performing-Technology-Organizations/dp/1942788339>
5. Whitepapers from Red Hat
 - a. <https://www.redhat.com/cms/managed-files/co-modernization-whitepaper-inc0460201-122016kata-v1-en.pdf>
 - b. <https://www.redhat.com/en/resources/application-delivery-container-platform-whitepaper>
 - c. <https://www.redhat.com/en/resources/modernize-public-sector-apps-whitepaper>
6. GAO Study - <https://www.gao.gov/assets/680/677454.pdf>
7. Thoughtworks tech radar - <https://assets.thoughtworks.com/assets/technology-radar-vol-22-en.pdf>
8. API First - <https://swagger.io/resources/articles/adopting-an-api-first-approach/>
9. OpenAPI Fuse - <https://developers.redhat.com/blog/2019/07/09/api-first-design-with-openapi-and-red-hat-fuse/>
10. Red Hat Digital Transformation - <https://www.redhat.com/en/solutions/digital-transformation>

QUOTES AND GOOD ONE LINERS

1. Software has no business value until it's deployed
2. "Change is too risky, let's just keep slowly going down the path to extinction" -- IT Dinosaurs
3. Modernization is not about adopting new technologies and practices, it is about what happens to the old ones
4. "Let's face it, all we are doing is writing tomorrow's legacy software today" -- Martin Fowler