Electric Cloud

Software (and Software Pipeline) Architecture Matters

Anders Wallgren | CTO, Electric Cloud
@anders_wallgren

Agenda

I. Some context
II. Why software architecture is important
III. Loosely coupled architectures (microservices)
IV. Some patterns to help with microservices
V. Why software pipeline architecture is also important
VI. Resources
VII. Q&A?

Some Context

Every business is a software business

"Software is eating the world"
Marc Andreessen, The Wall Street Journal, August 2011

FINSERV
building solutions to deliver better service

RETAIL
building platforms for online sales and support

AUTOMOTIVE
building services for the connected car

FEDERAL
delivering on time and with higher quality

HEALTHCARE
building solutions to improve care

TELECOM
building embedded and online services

Software Is Eating My Bank

https://www.cbinsights.com/research/disrupting-banking-fintech-startups/

Electric Cloud

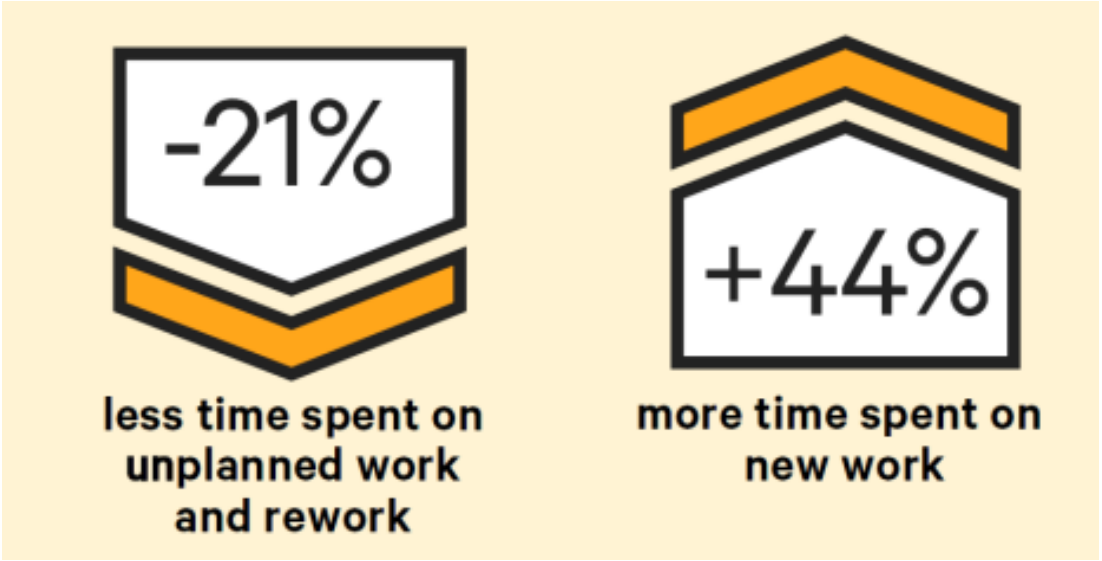
“Software is now the primary driver of innovation & disruption.”

What Is The One Question That Predicts Software Team Performance With Startling Accuracy?

“To what degree do we fear doing deployments?”

Source: Puppet Labs 2015 State Of DevOps: https://puppetlabs.com/2015-devops-report

High-Performing IT Organizations Do It More Often

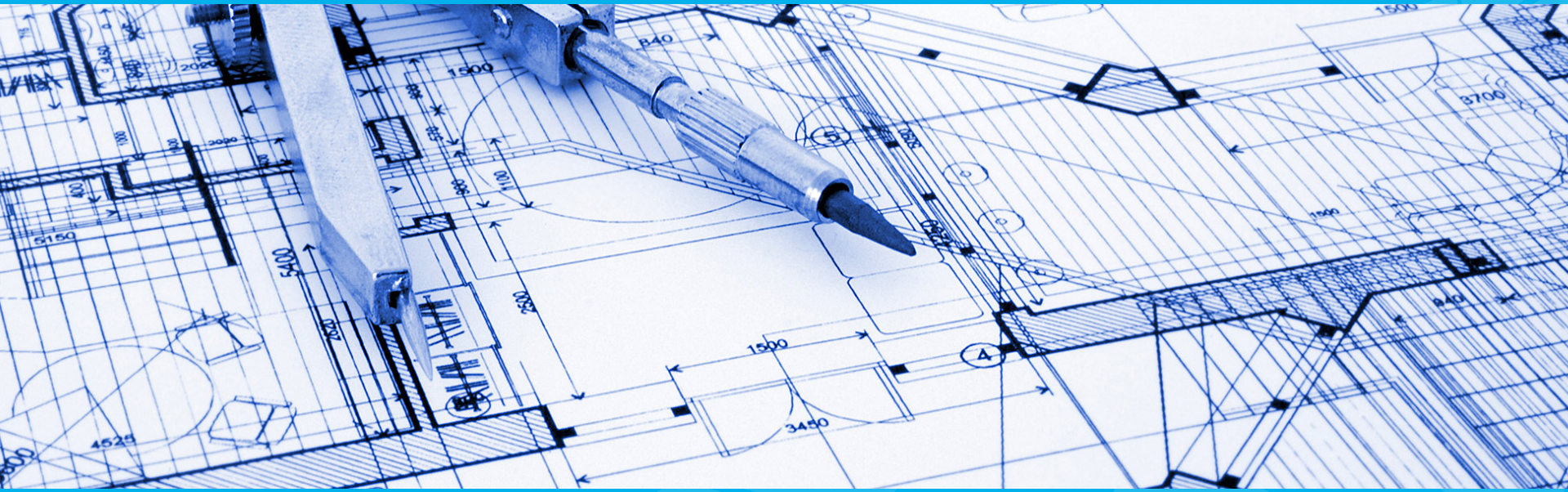


2017: 5x lower change failure rate, 96x faster recovery from failures.

From: IT Revolution and Puppet Labs' 2016/2017 State of DevOps

Electric Cloud

But I was told this talk was about architecture?

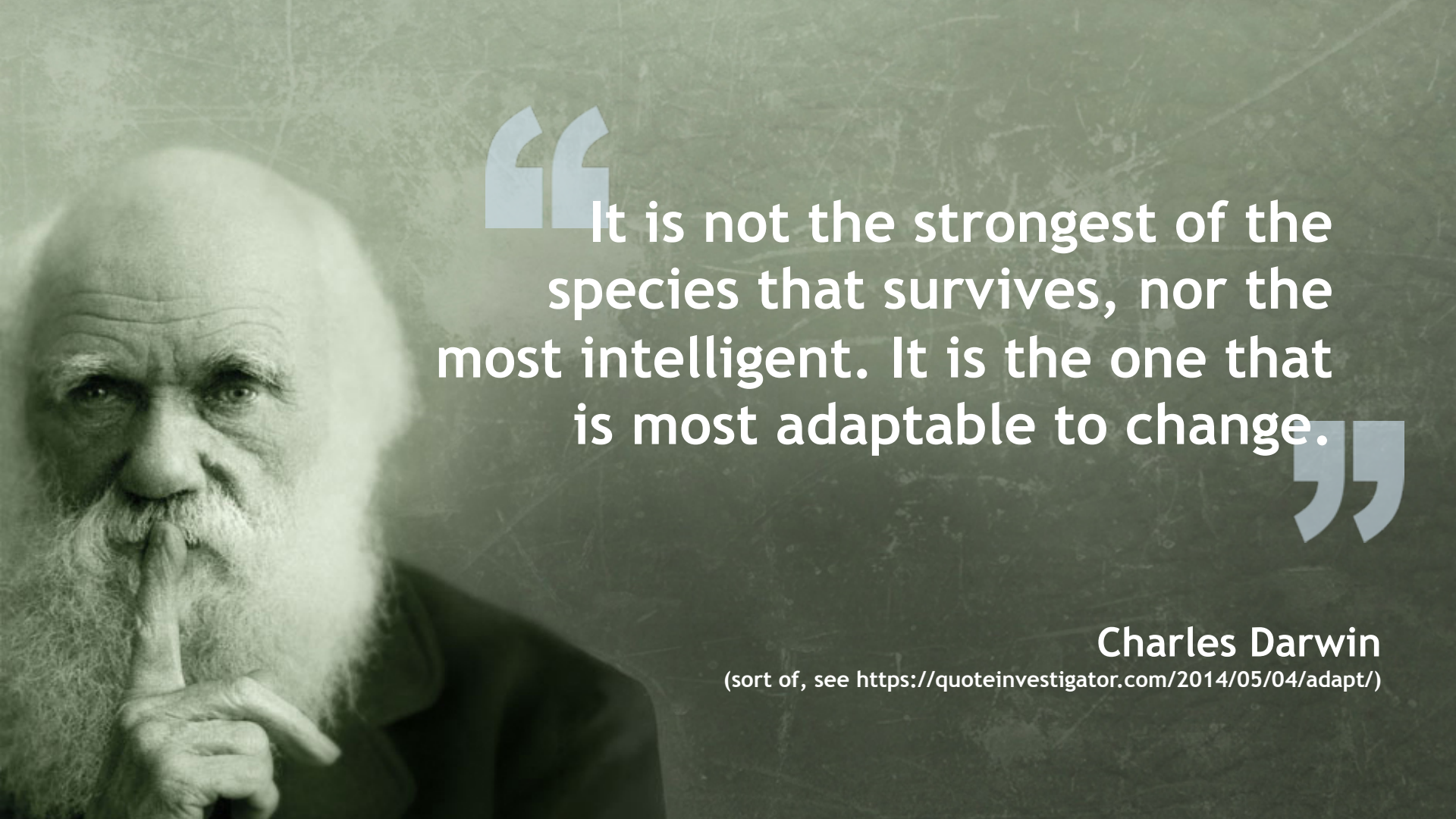


Why Software Architecture is Important

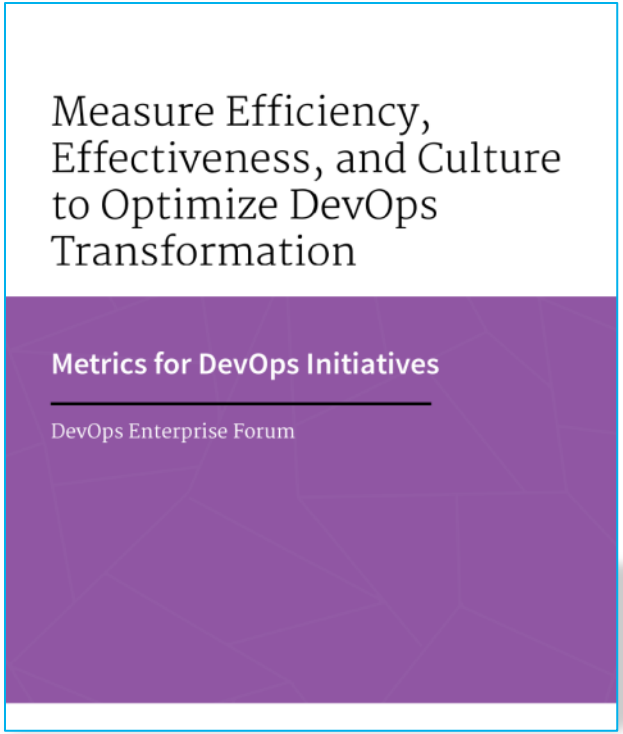
Architecture is about the important stuff.

Whatever that is.

Martin Fowler's definition


“It is not the strongest of the species that survives, nor the most intelligent. It is the one that is most adaptable to change.”
Charles Darwin
(sort of, see https://quoteinvestigator.com/2014/05/04/adapt/)

> Software's most important quality is its adaptiveness and ease of change.



Electric Cloud

Architectural outcomes: can my team…

- …make large scale changes to the design of its system without the permission of someone outside the team, or depending on other teams?
- …complete its work without fine-grained communication and coordination with people outside the team?
- …deploy and release its product or service on demand, independently of other services the product or service depends upon?
- …do most of its testing on demand, without requiring an integrated test environment?
- …perform deployments during normal business hours with negligible downtime?

puppet DORA DEVOPS RESEARCH & ASSESSMENT

Loosely coupled architectures and teams are the strongest predictor of continuous delivery.

puppet DORA DEVOPS RESEARCH & ASSESSMENT

What Is The Cost of Not Adapting and Scaling?

$100K Hr
Infrastructure Failure

$0.5 - $1M Hr
Critical App Failure

$1.25 - $2.5B Yr
F1000 Downtime Cost

"DevOps & the Cost of Downtime:
Fortune 1000 Best Practice Metrics Quantified"

Electric Cloud

Loosely Coupled Architectures

(mostly Microservices, but other stuff, too)

Monolithic Architectures

The good, the bad, the ugly…

Pros

- Can be easier to develop
- Can be easier to test
- Can be easier to deploy

Cons

- Easier to produce spaghetti code
- Harder to integrate new technologies
- Harder to learn and understand the code
- **You have to scale everything to scale anything**
- **Can’t deploy anything until you deploy everything**

Microservices Architecture

A suite of services, each focused on doing one thing well

- Independently developed
- Independently deployable
- Exposes an API
- Runs in its own process

Sounds loosely coupled to me!

"Gather together those things that change for the same reason, and separate those things that change for different reasons."

– Robert Martin

What's cool about Microservices?

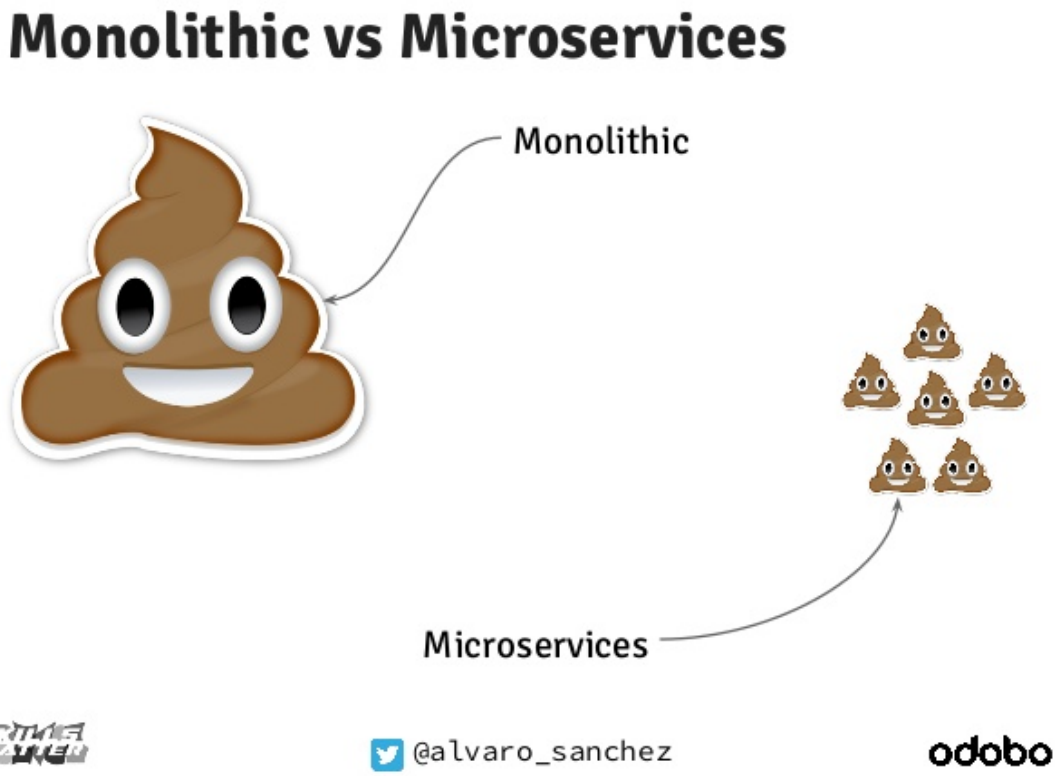
Divide and conquer complex distributed applications

Independently developed and deployed, ideal use case for containers

Freedom to choose the right technology (new or old) for each service

Smaller more autonomous teams are more productive

Microservices are not a silver bullet!



Get Loose! Some Best Practices For Decomposing The Monolith

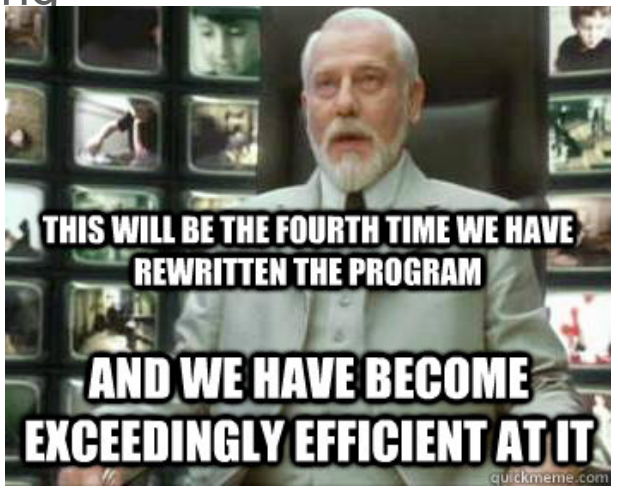
Decomposing your monolith

- Identify what can be migrated to microservices - look for seams – areas of code that are independent, focused around a single business capability
- Use the strangler pattern liberally
- Domain-Driven Design (Domain Contexts, Aggregates) are useful
- You'll ~~probably~~ get it wrong the first time
- Be Agile in your efforts – don't boil the ocean
- Benchmark against the monolith early
- Don't ignore organizational structure (Conway's Law)
- Dependency analysis tools help, but are no panacea

"You don't decompose monoliths, you *erode* them" - Mirco Hering

Decomposing your monolith's state

- State ~~may~~ will be your largest source of coupling
- Understand your schema (if you have one)
  - Foreign key constraints
  - Shared mutable data
  - Transactional boundaries
- Is eventual consistency OK?
  - Avoid distributed transactions
  - Messaging between services
- Split data before you split code?
- Do you need an RDBMS at all or can you use other stores?
- Look into Saga, Event Sourcing, CQRS, but beware of complexity

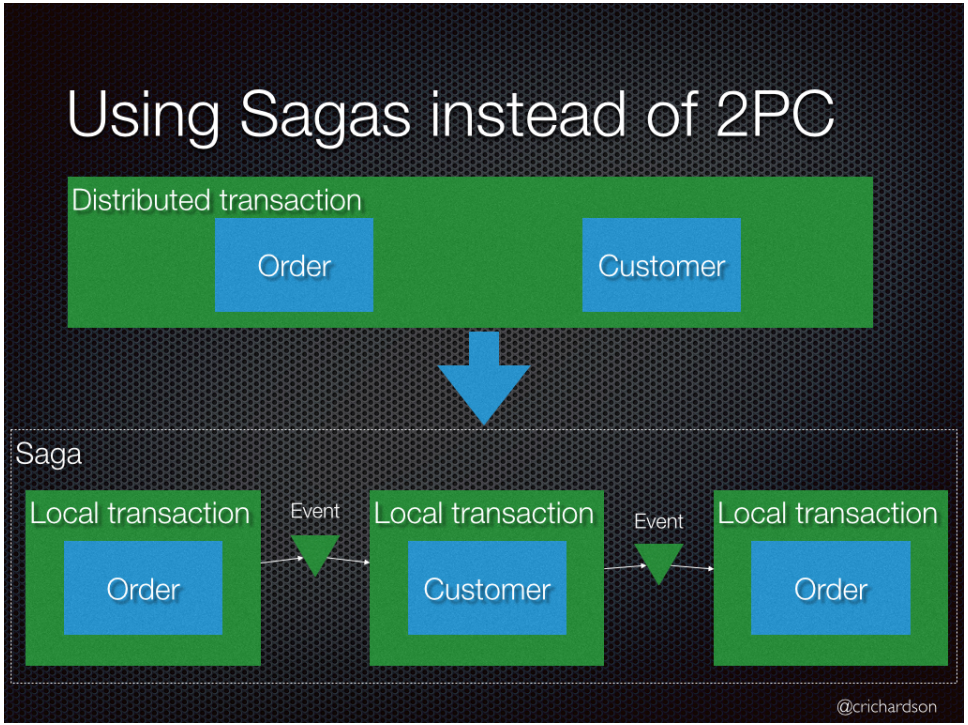


“Size doesn’t matter” - Adrian Cockcroft

- If you can’t update a service without changing other things, it’s too large (or just wrong)
- The smaller the services, the more potential decoupling benefit
- The smaller the services, the more moving parts you will have
- You should be able to (re-)rewrite one service in a “small” number of weeks
- Don’t get too hung up on size: it’s probably better to have small number of macro-services than a single monolith

Patterns To Assist in De-Coupling

Saga Pattern: Maintain data consistency across services



Problem:
Two services, each owns a business entity, we need a consistent update without 2PC

Solution:
Implement business logic that spans multiple services using service-local transactions + events

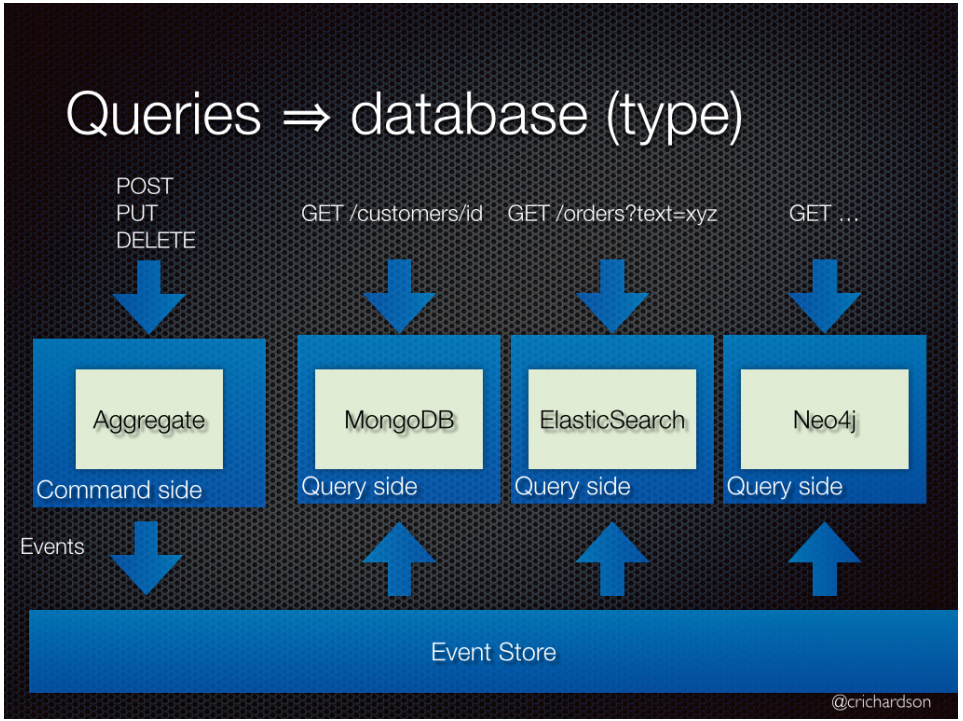
Each transaction publishes an event, which triggers the next transaction in the saga

Challenges:

- More complex, different programming model
- You may need to program compensating transactions to “undo”
- You need to be able to commit a transaction AND publish an event atomically (without 2PC)

http://microservices.io/patterns/data/saga.html

Command Query Responsibility Segregation



Problem:
Two services, each owns a business entity, we need to query across both, without the benefit of JOIN

Solution:
Treat the "CUD" and "R" of CRUD as separate concerns:

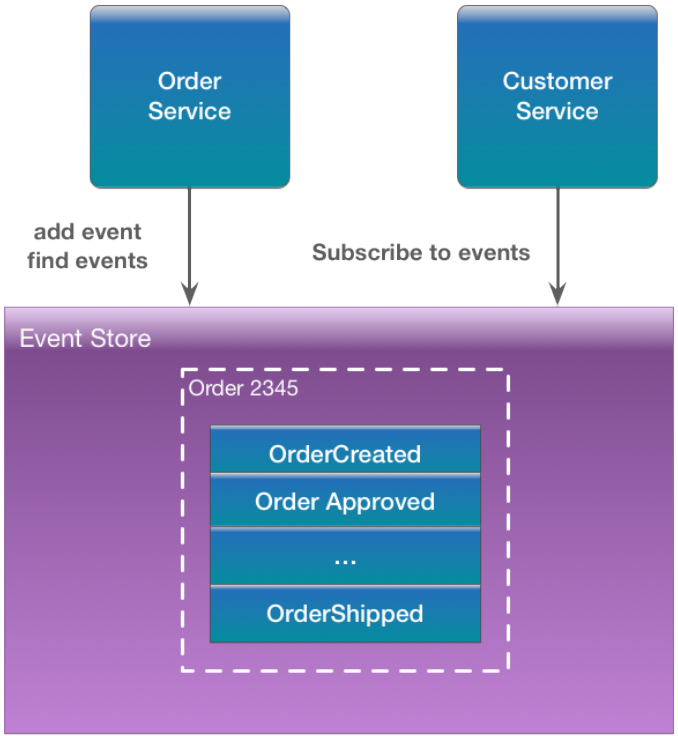
- Command-side
  - Handles create/update/delete
  - Emits events when data changes
- Query-side
  - Handles reads against data
  - Kept up to date by subscribing to data change events

Challenges:
- Complexity
- You need to be OK with eventual consistency, because lag
- Reporting databases may be a simpler approach

http://microservices.io/patterns/data/cqrs.html

Electric Cloud

Event Sourcing



Problem:

Atomically publish events when business data changes, without 2PC

Solution:

Persist business entities as a sequence of mutating events instead of (or in addition to) as a “row in the table”

Benefits:

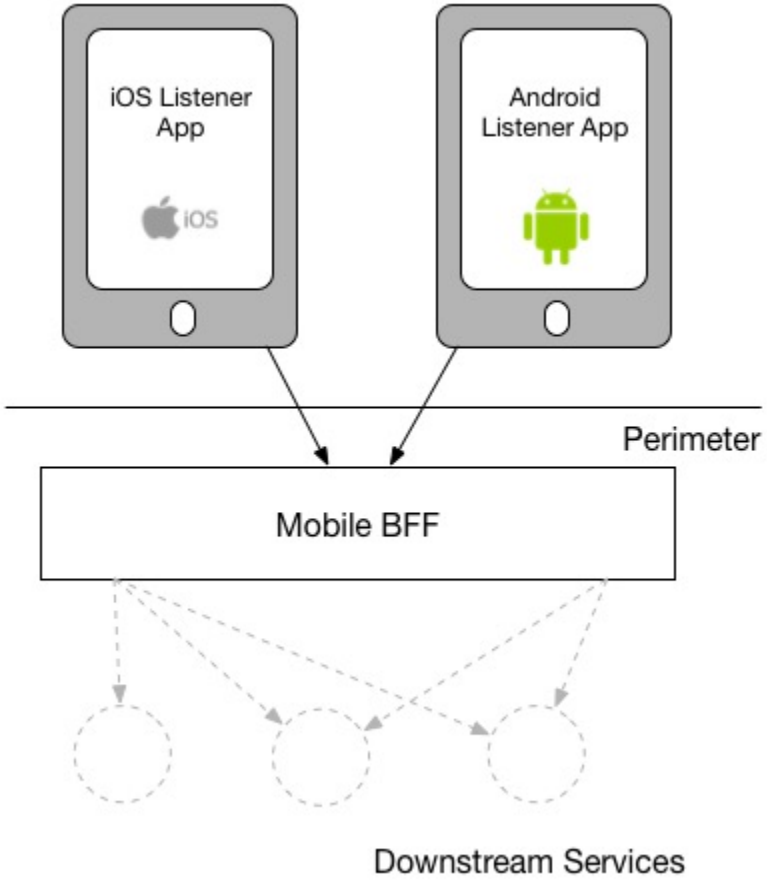
- Audit log for free (hibernate-envers)
- You can query the state of the system at a point in time

Challenges:

- Complex and probably not something you build an entire system out of
- Queries are hideous, so this leads you in the direction of CQRS

http://microservices.io/patterns/data/event-sourcing.html

BFF: Backends For Frontends

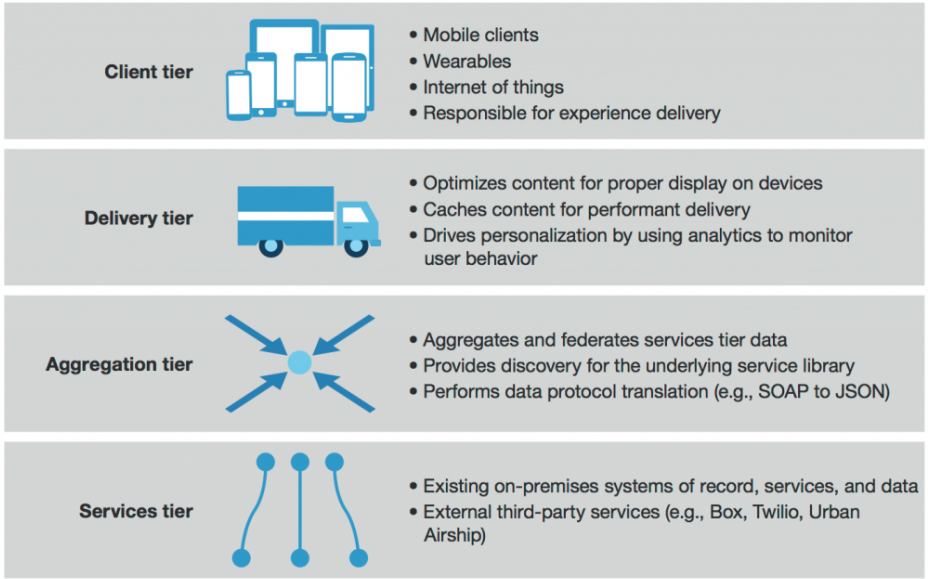


- A variation on API gateways
- Tailored APIs for mobile devices, IoT, whatever
- “One backend per user-experience”
- “Impedance matching”
- Aggregate APIs to downstream services
- Use as strangler pattern to transition to new APIs

http://samnewman.io/patterns/architectural/bff/

Not three tiers, but four!

> Mobile's Continuous Delivery Requires A Loosely Coupled Architecture



https://go.forrester.com/blogs/13-11-20-mobile_needs_a_four_tier_engagement_platform/

Electric Cloud

Serverless

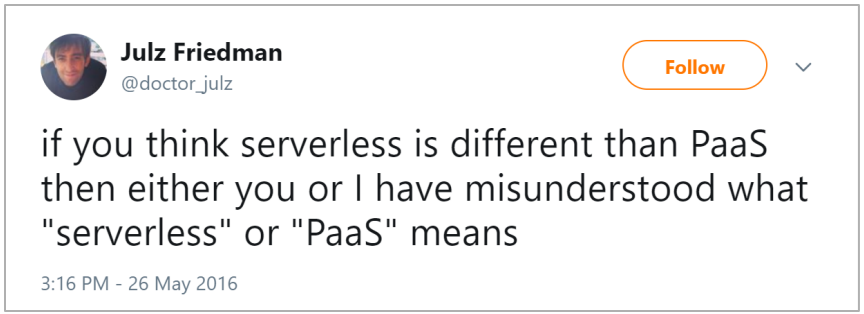
Backend-as-a-Service (BaaS)

- App depends largely or exclusively on third-party backend services
- Rich eco-system of services
- Why roll your own?

Function-as-a-Service (FaaS)

- AWS Lambda, Google Cloud Functions, Azure Functions, etc.
- Run backend code without managing server applications
- Code is triggered by events (S3 changes, messages, schedules)
- Easy horizontal scaling
- Very different programming model

How is “Serverless” not just PaaS?

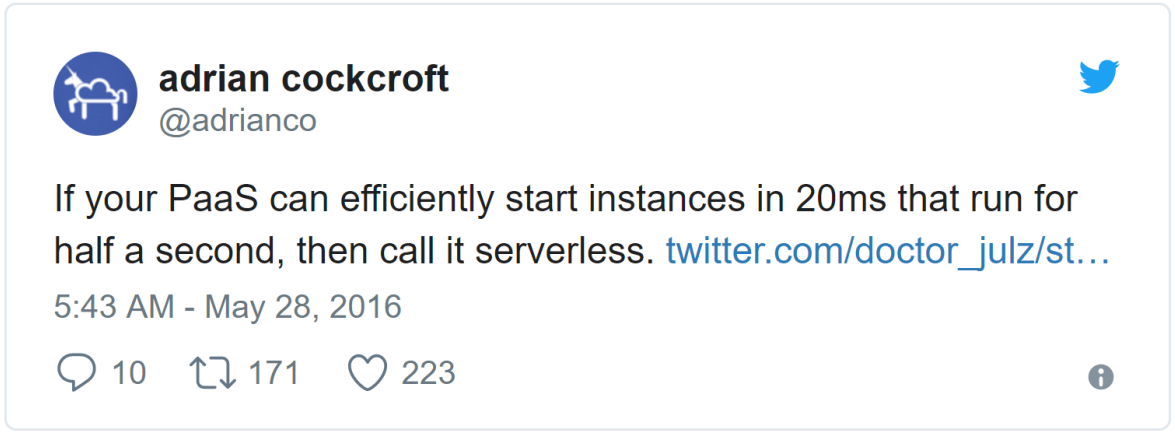


Julz Friedman
@doctor_julz

Follow

if you think serverless is different than PaaS then either you or I have misunderstood what "serverless" or "PaaS" means

3:16 PM - 26 May 2016



adrian cockcroft
@adrianco

If your PaaS can efficiently start instances in 20ms that run for half a second, then call it serverless. twitter.com/doctor_julz/st…

5:43 AM - May 28, 2016

10 171 223

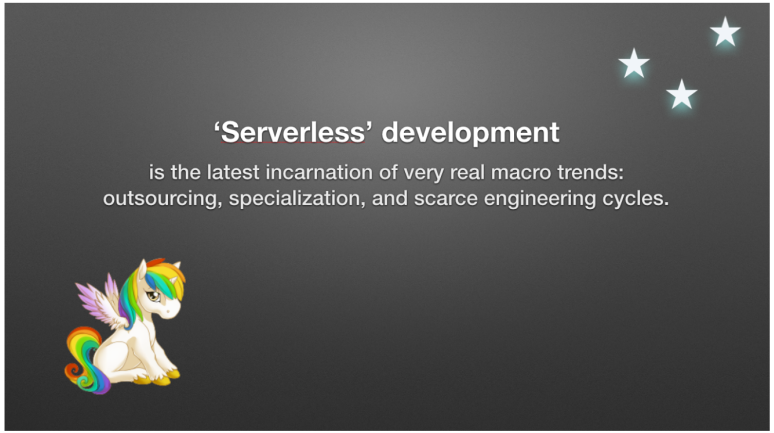
Serverless? Yes, but...

Charity Majors
@mipsytipsy

Follow

I mean how hard can it be to just glue together APIs that other people have written and support and scale? 🤔 #serverless #NoDevs

12:48 PM - 23 May 2016



https://charity.wtf/2016/05/31/operational-best-practices-serverless/

> Figure out what your core differentiators are, and own the shit out of those
> – Charity Majors

Caitie McCaffrey
@caitie

Follow

Real question how does state get persisted with #serverless?

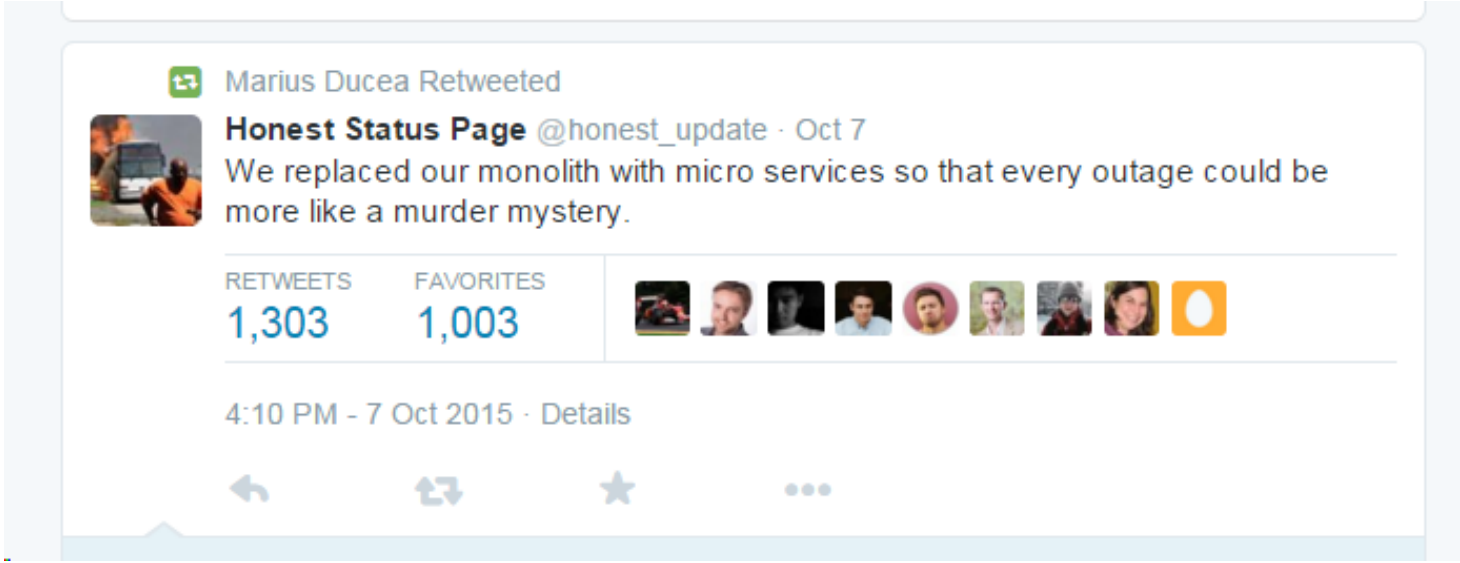
I understand scale out of stateless servers, but who stores the state?

12:38 PM - 26 May 2016

Electric Cloud

A Few Other Things That Might Not Appear To Be Software Architecture, But Really, They Are

The Importance of Monitoring



APPDYNAMICS

dynatrace

sysdig

Twistlock

Electric Cloud

Monitoring Best Practices

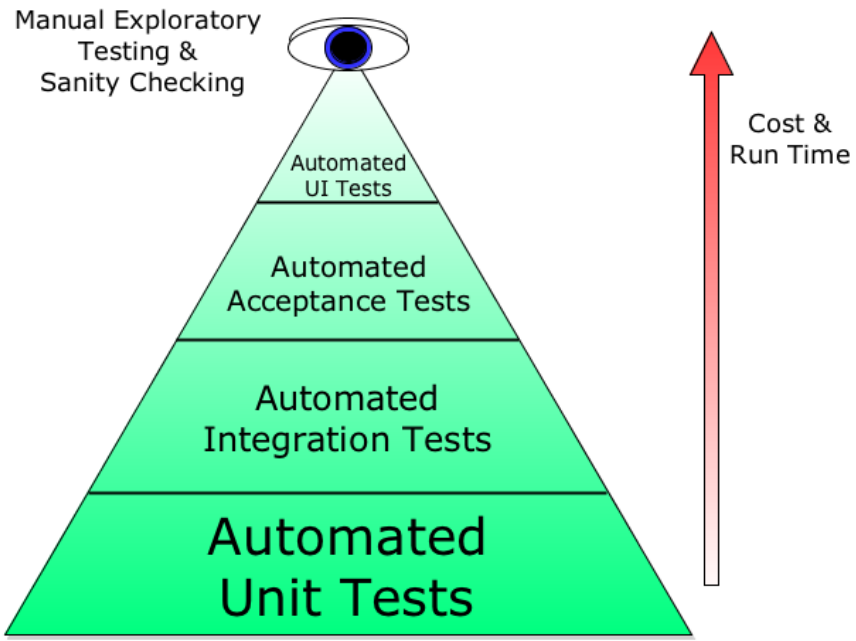
- All services should log and emit monitoring data in a consistent fashion (even if using different stacks)
- Log early, log often
- Aggregate monitoring and log data into a single place
- Use techniques like correlation ids to track requests through the system
  - “So then requestId 0xf00dfee8 in the log on ms-app-642-prod becomes messageId 1125f34c-e34e-11e2-a70f-5c260a4fa0c9 on ms-route-669-prod?”
- Monitor latency and response times between services
- Understand what a well-behaving service looks like, so you can tell when it goes wonky
- Monitor the host (CPU, memory, etc)

https://neo4j.com/blog/managing-microservices-neo4j/

Electric Cloud

Architect for Automated Testing

- Spend the time where you need but no more than necessary
- Use service virtualization / stubbing / mocking
- Consider using separate pipelines for end-to-end tests
- Performance testing at the service level is more important than in a monolith
- As always, **flaky tests are the devil**



(I know they're not an "architecture" but)

What About Containers?

How do Containers help with Agile/DevOps/CD?

Containers help increase velocity, quality, and repeatability of software delivery by providing a uniform means of application distribution (the container image) that includes not only the application artifact(s), but all its dependencies and environment as well.

docker

rkt

Why we love containers, specifically

- Faster startup & shutdown
  - Not booting the whole OS every time
- Smaller than VMs
  - Usually…beware of image bloat
- Portable – build once, run anywhere
  - Great for distributing build environments to devs
- Better environment fidelity throughout the pipeline
  - No need for ops to use the Magic 8 Ball as much
- Great match for microservices
- Separation of concerns
  - Applications are decoupled from the infrastructure they run on
- Higher density
  - Better resource utilization at scale
- Orchestration platforms provide scaling, resiliency
- License savings (one license per host, multiple containers on the host)

Container Orchestration can do much heavy lifting

- Container lifecycle management
- Auto scaling
- Self-healing
- Networking, routing, load balancing, ingress control
- Service binding/discovery
- Namespaces (including DNS)
- QOS, node affinity/anti-affinity
- Storage (persistent and otherwise)

Azure
Container Service

Why Software Pipeline Architecture Matters

or

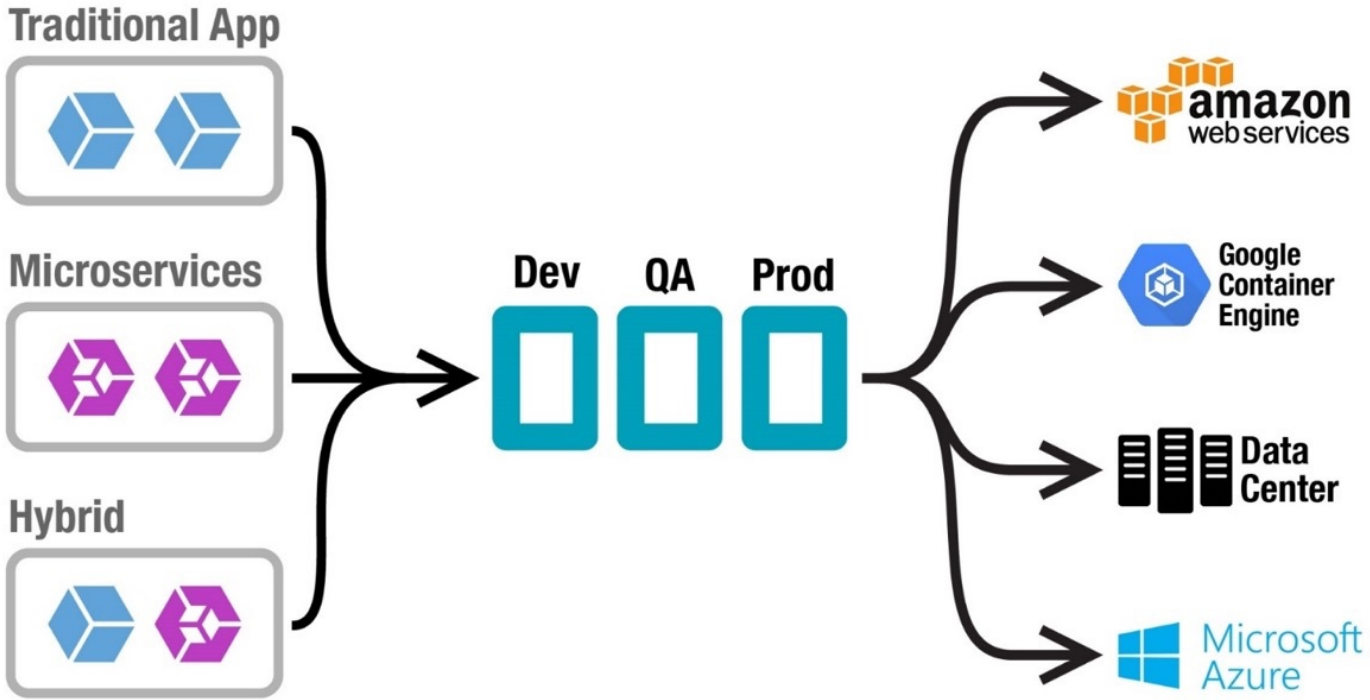
"If a tree falls in the woods, and nobody is around to hear it, and it hits a mime, does anyone care?" - Gary Larson

Software Delivery has changed…

Large App → Few Releases

Small & Modular App(s) → Many Releases

Many architectures, many pipelines, many destinations

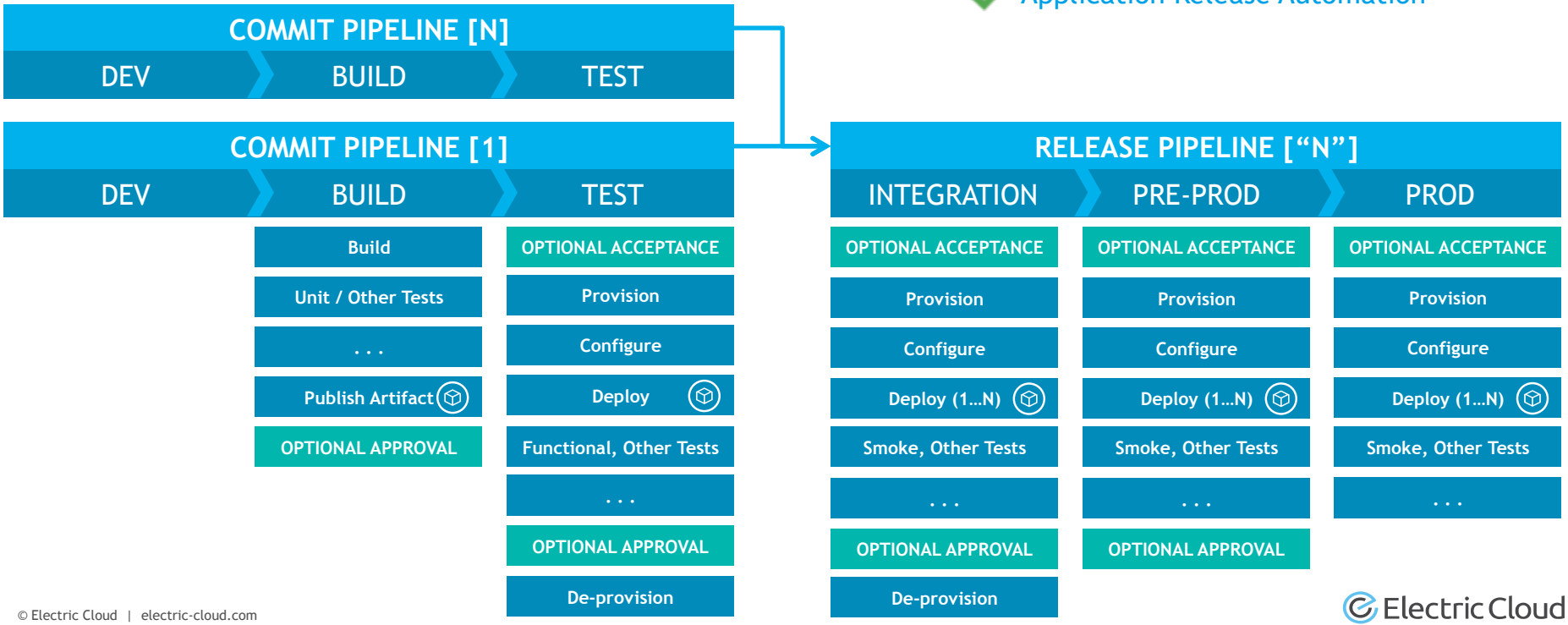


"Bi-modal IT is a massive reductionist oversimplification" – Jez Humble

Electric Cloud

Different Flavors of Pipelines

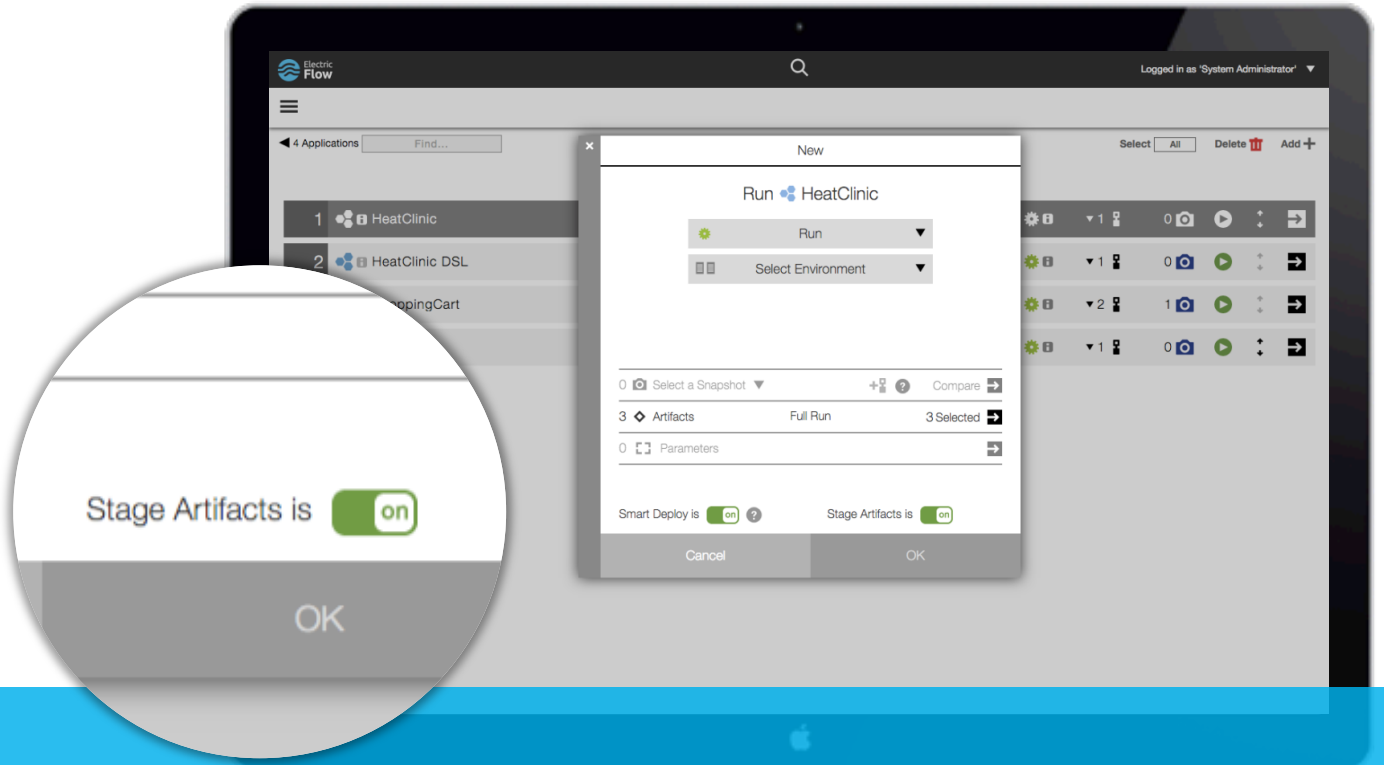
- Pipeline Orchestration
- Build/Test Automation
- Cloud Resource Management
- Deployment Automation
- Application Release Automation



Electric Cloud

Some Software Pipeline Best Practices

1 Artifact Repo



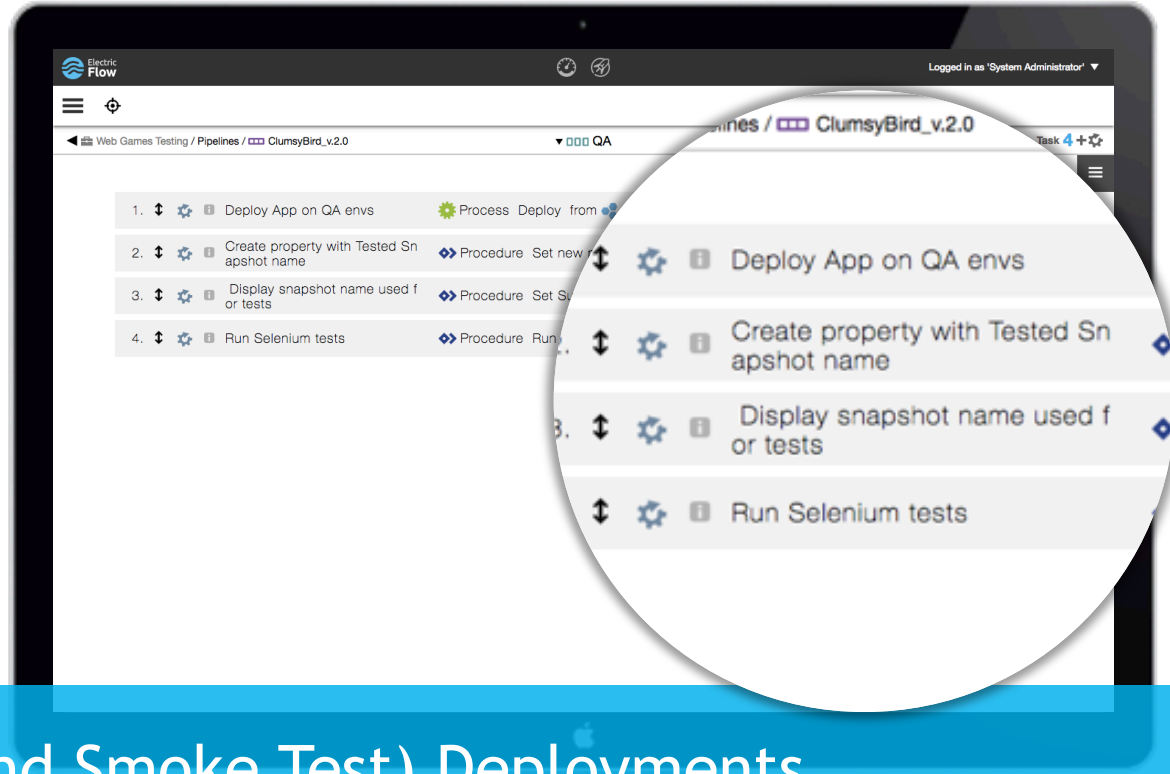
Have an artifact repository

Version everything (applications, infrastructure and middleware)

2

Artifact Repo

Automate Deployments



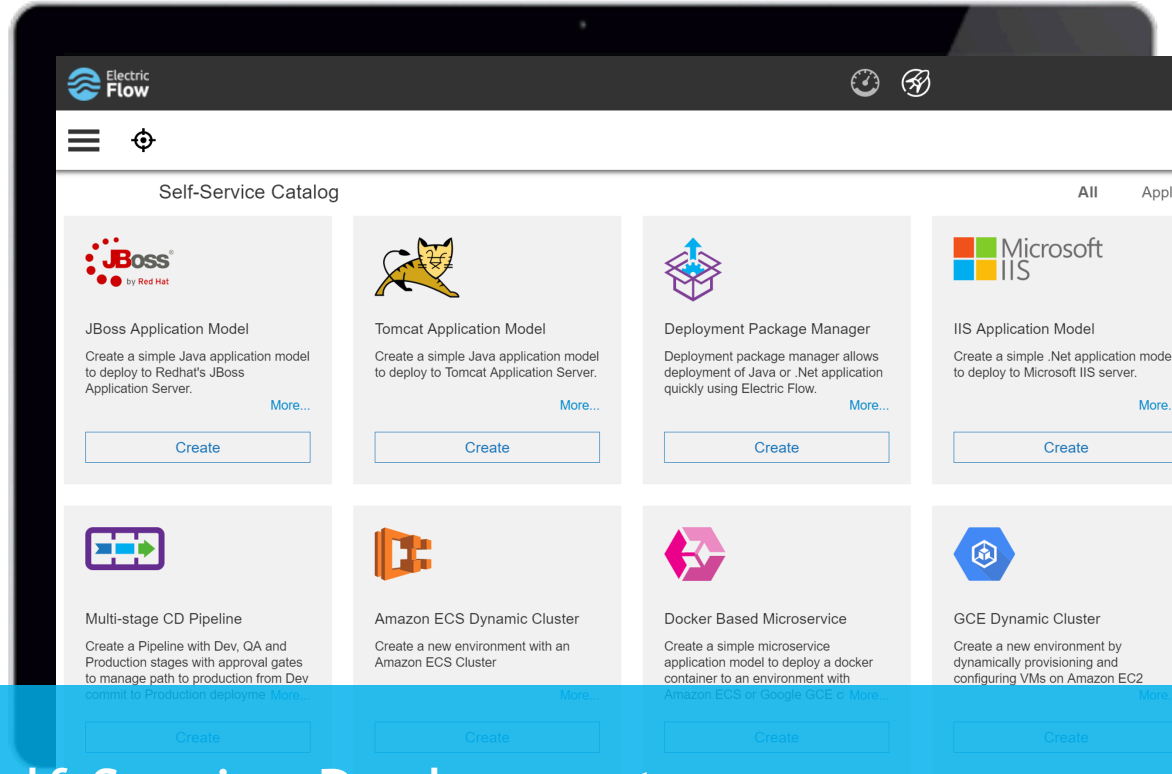
Automate (and Smoke Test) Deployments

"The most powerful tool we have as developers is automation" – Scott Hanselman

From: The DevOps Handbook, Gene Kim, Jez Humble, Patrick Debois, John Willis

3

Artifact Repo
Automate Deployments
Self-Service Deploys

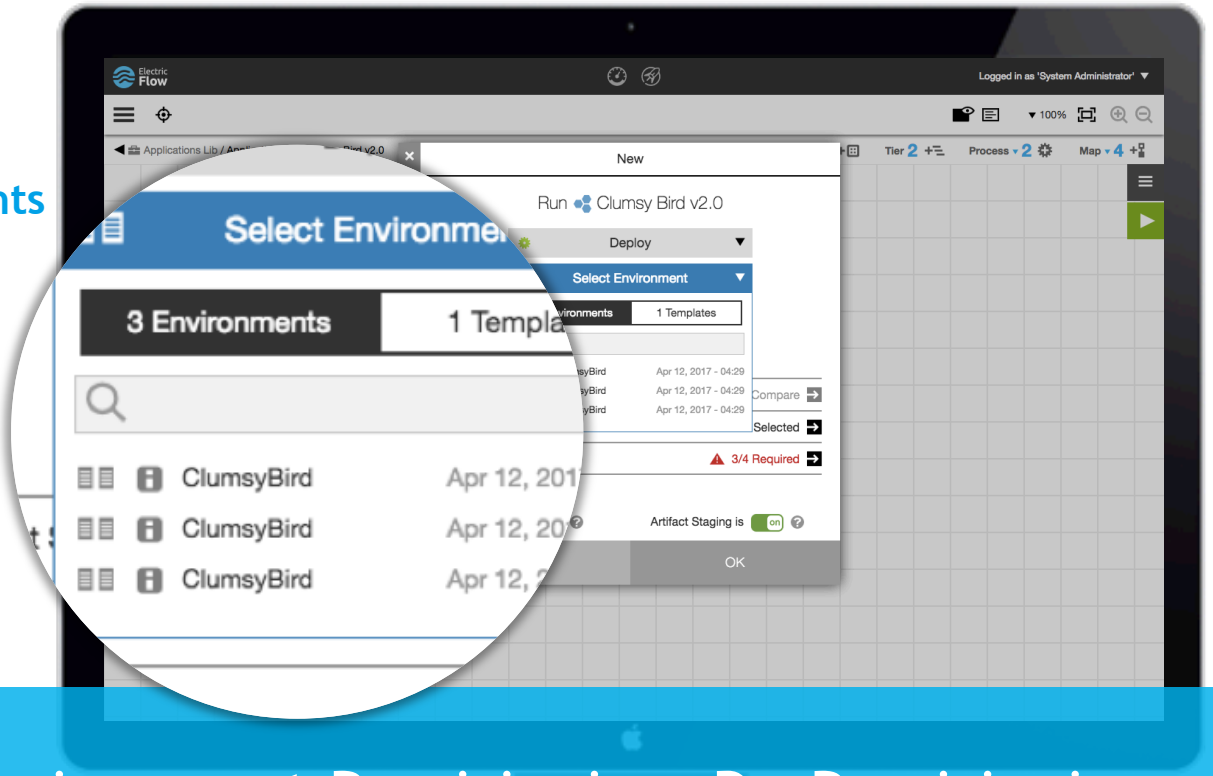


Enable Self-Service Deployments

Allow teams to predictably and efficiently deliver application versions, on-demand.

4

Artifact Repo
Automate Deployments
Self-Service Deploys
Self-Service Environments

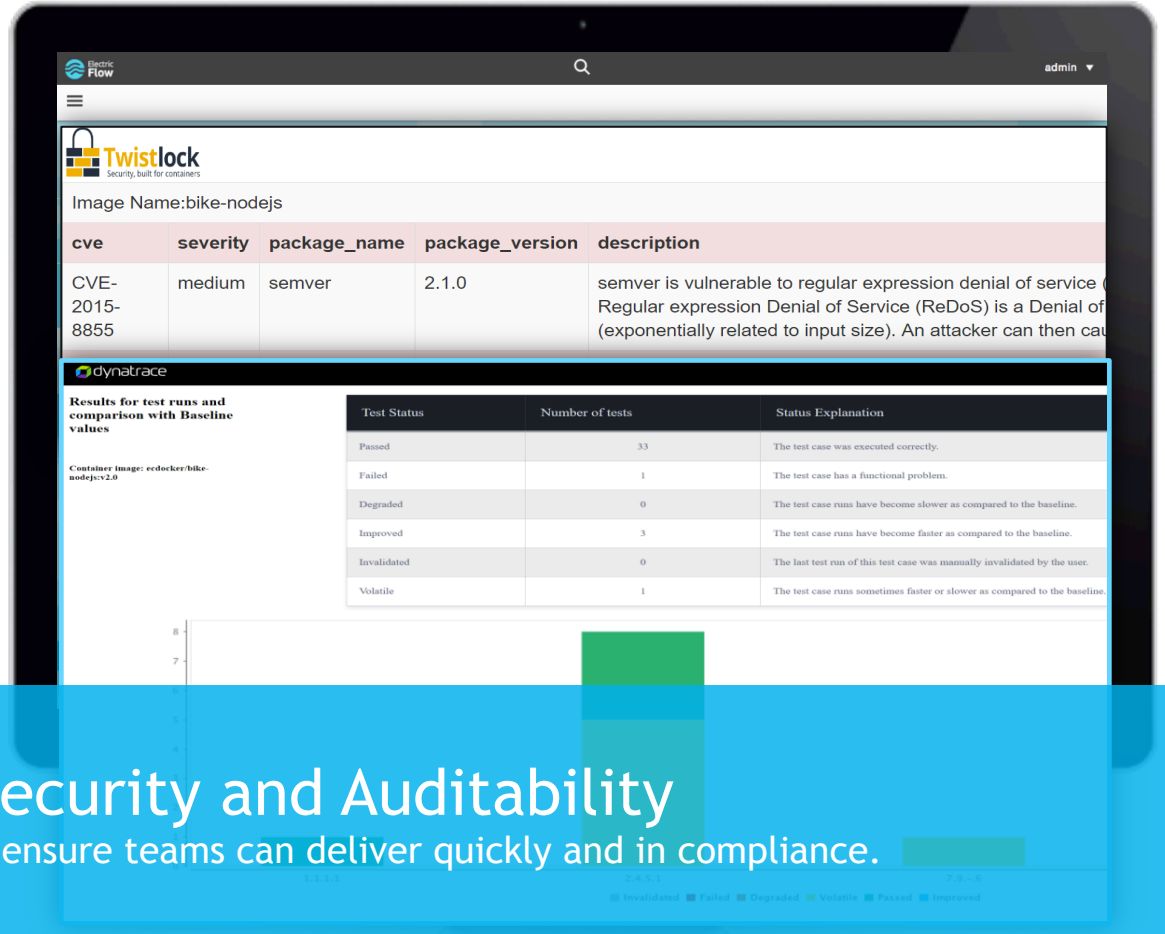


Enable Self-Service Environment Provisioning, De-Provisioning

Spin up production-like environments anywhere in the cycle. Tear them down when finished.

5

Artifact Repo
Automate Deployments
Self-Service Deploys
Self-Service Environments
Security, Auditability



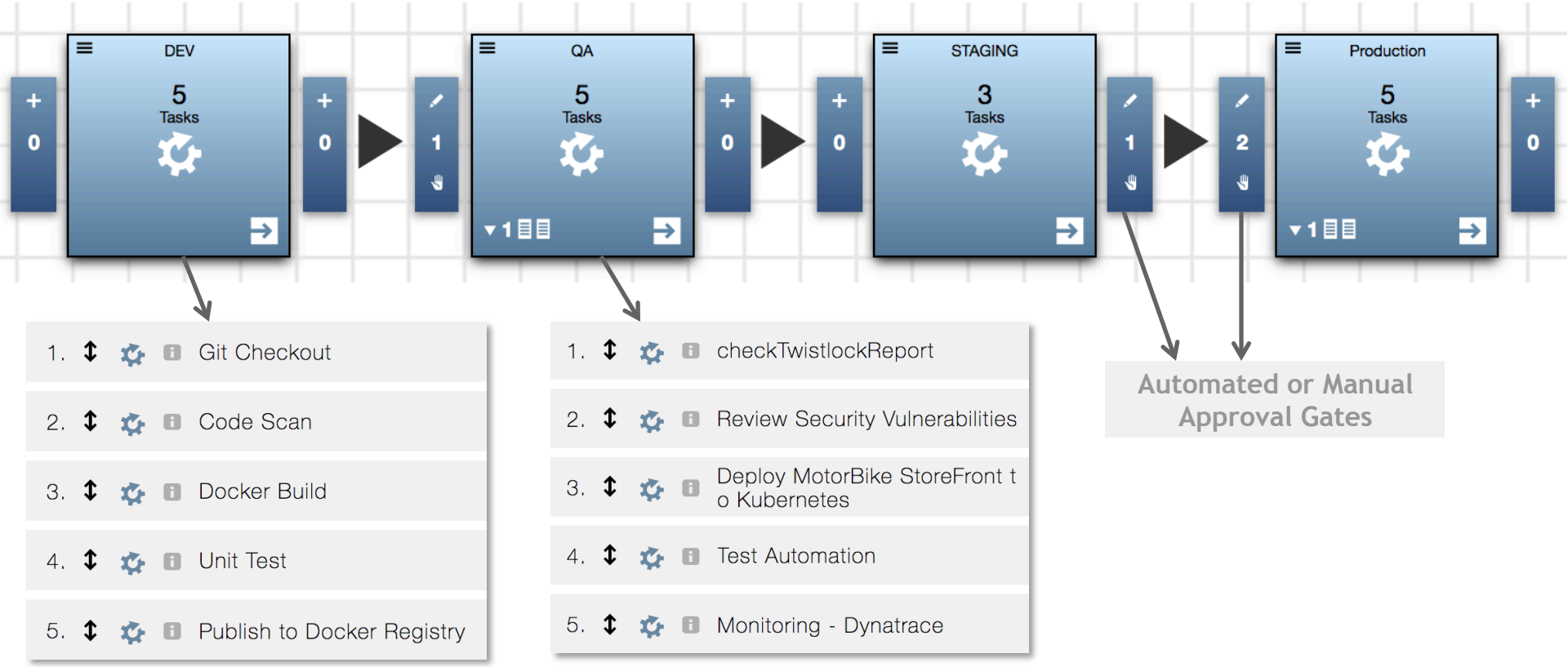
Ensure Security and Auditability

Approvals and permissions ensure teams can deliver quickly and in compliance.

Best Practices for Software Pipelines

- Automate all the things – no more manual handoffs
- Your software pipeline needs to be as available as your app (HA, DR, etc.)
- Don’t tie your software pipeline architecture to your software architecture
- Are your tests automated? Really automated?
- Reduce onboarding time, waiting, and complexity with self-service automation/ChatOps approaches
- Provide a real-time view of all the pipelines’ statuses and any dependencies or exceptions
- Plug the pipeline into monitoring so that alerts can trigger automatic processes such as rolling back a service, switching between blue/green deployments, scaling and so on (and shift-left monitoring)
- Allow for automatic and manual (if you have to…) approval gates into and out of pipeline stages
- Create reusable models/processes/automation for your various pipelines

Simple Container Release Pipeline



Electric Cloud

Best Practices for Microservices Pipelines

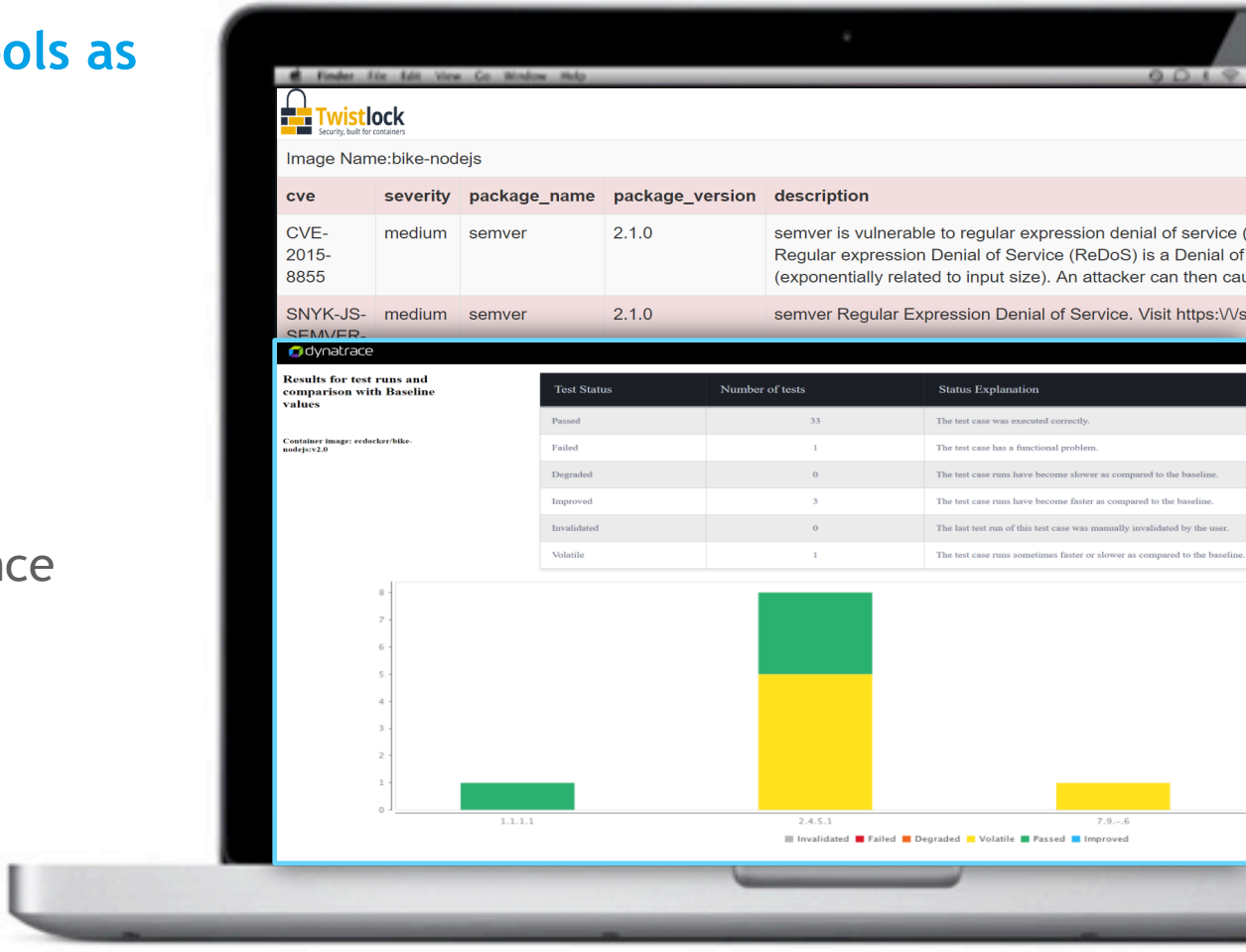
- One repository per micro-service, if you're using them
- Independent CI and Deployment pipelines per service
- "Automate all the things": plug in your toolchain to orchestrate the entire pipeline (CI, testing, configuration, infrastructure provisioning, deployments, application release processes, and production feedback loops.)
- Your pipeline must be tools/environment agnostic to support each team's workflow and tool chain
- Test automation tools and service virtualization are critical

Best Practices for Container-deployed Apps

- Track artifacts/images through the pipeline (who checked-in the code, what tests were run, pass/fail results, on which environment it was deployed, which configuration was used, who approved it and so on)
- Bake in compliance into the pipeline by binding certain security checks and acceptance tests

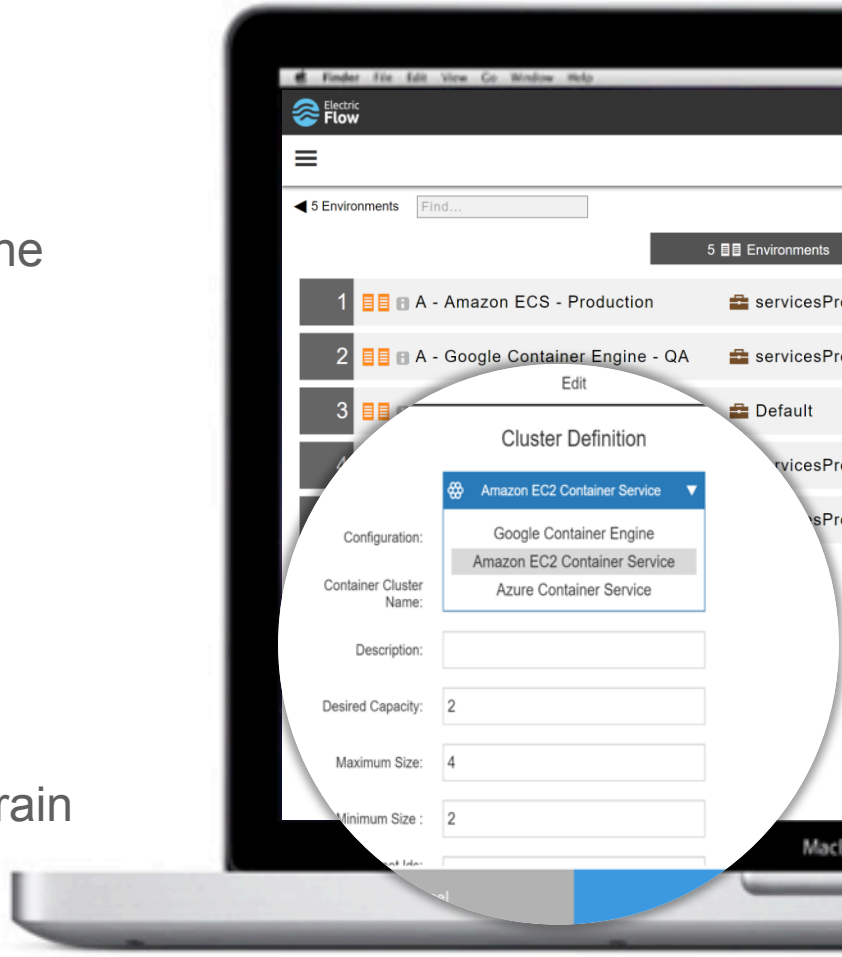
Orchestrate all your tools as part of your software pipeline

- CI
- Testing tools
- Vulnerability tracking
- Shift-Left Monitoring
- Approvals and compliance
- Deployments



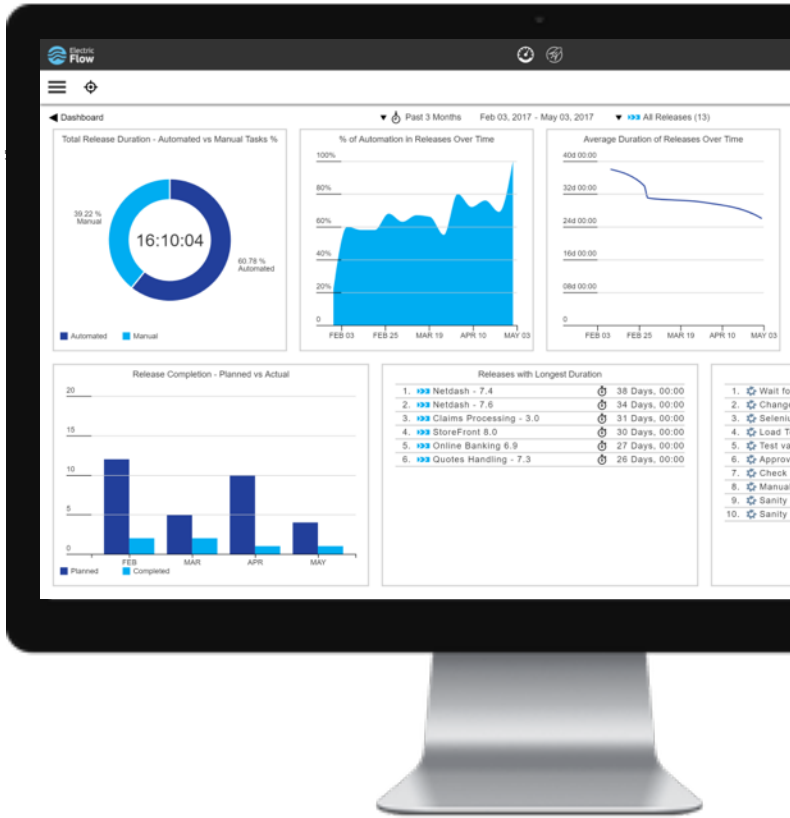
Environments and Deployments

- Automate all the things!
- Ensure fidelity of environments throughout the pipeline
- One service per host
  - Minimize the impact of one service on others
  - Minimize the impact of a host outage
- Use VMs/containers to make life easier
  - Containers map very well to microservices
  - “Immutable servers”
- PaaS solutions can help, but can also constrain



What about our Release Pipeline?

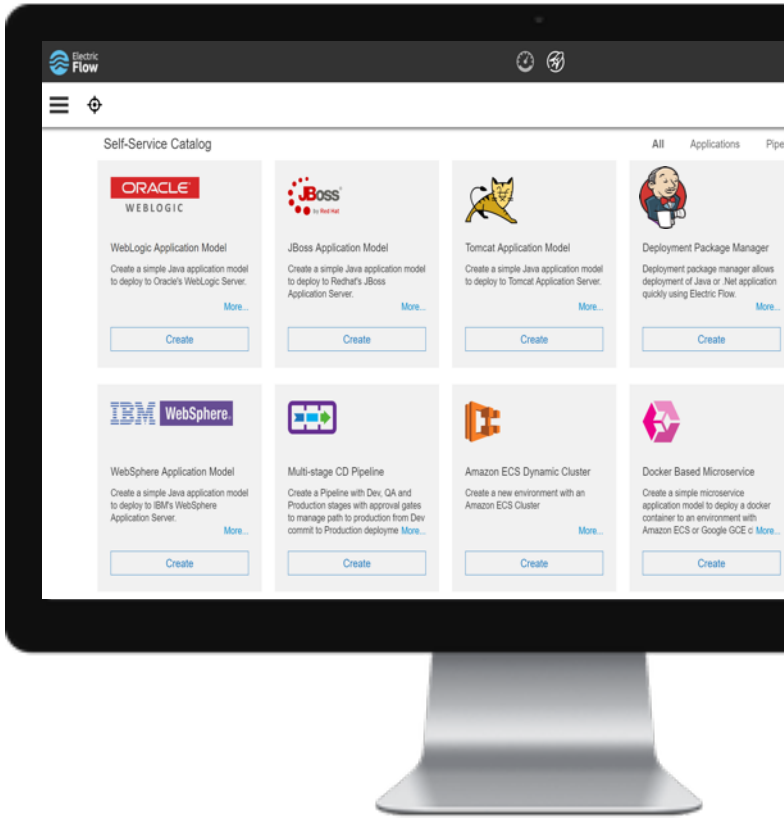
- Automate **and Orchestrate** all the things!
  - Ideally, one platform for all software delivery (CD, data center, cloud, container)
- Govern the promotion of all the things!
  - Understand pipeline and release statuses and any dependencies or exceptions
- Make pipeline tools/environment agnostic
  - Support each team's workflow and tool chain
- One repository per service
  - Independent CI and Deployment pipelines per service



Electric Cloud

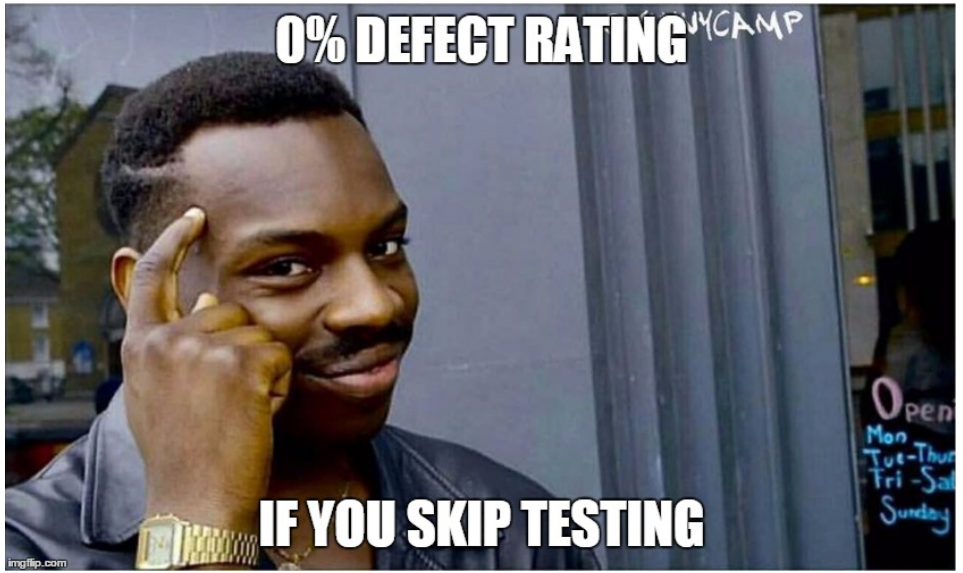
What about our Release Pipeline (cont'd)?

- Enable self-service automation
  - Standardize and operationalize to reduce onboarding time, waiting, complexity
- Monitor early and often
  - Set up alerting; trigger automatic processes (service rollback, scaling, etc)
- Track artifacts through the pipeline
  - Who checked-in the code, test coverage and quality, environments deployed to, configurations used, approvers, etc)
- Assure security compliance
  - Include security checks, acceptance tests
  - Allow for both automatic and manual approval gates



Electric Cloud

Don’t have “emergency change processes”



Electric Cloud

Resources

- Migrating to Microservices at Netflix
  https://www.infoq.com/presentations/migration-cloud-native
- How we ended up with microservices
  http://philcalcado.com/2015/09/08/how_we_ended_up_with_microservices.html
- Confusion in the Land of the Serverless
  https://www.infoq.com/presentations/serverless-issues
- How to Make the Leap: Building Cloud-Ready Applications into the Architecture
  https://www.infoq.com/articles/cloud-ready-applications
- Backend for Frontend
  http://samnewman.io/patterns/architectural/bff
- WTF IS OPERATIONS? #SERVERLESS
  https://charity.wtf/2016/05/31/wtf-is-operations-serverless
  https://charity.wtf/2016/05/31/operational-best-practices-serverless
- Saga: How to implement complex business transactions without two phase commit
  https://blog.bernd-ruecker.com/saga-how-to-implement-complex-business-transactions-without-two-phase-commit-e00aa41a1b1b
- Developing Transactional Microservices Using Aggregates, Event Sourcing and CQRS
  https://www.infoq.com/articles/microservices-aggregates-events-cqrs-part-1-richardson
  Mobile Needs A Four-Tier Engagement Platform
  https://go.forrester.com/blogs/13-11-20-mobile_needs_a_four_tier_engagement_platform
- How to Make the Leap: Building Cloud-Ready Applications into the Architecture
  https://www.infoq.com/articles/cloud-ready-applications
- What Is "Cloud-Native" Data and Why Does It Matter?
  https://www.infoq.com/articles/cloud-native-data
- Low-risk Monolith to Microservice Evolution
  http://blog.christianposta.com/microservices/low-risk-monolith-to-microservice-evolution


Questions?

Thank You!

Anders Wallgren | CTO, Electric Cloud
@anders_wallgren

“What are Containers?” in one slide

- Container Image
  - Binary artifact that includes an application/service and everything required to run that app/service
  - Images are built from a descriptor file (e.g. dockerfile)
  - Primary purpose is to run a single service
- Container Runtime
  - Execution environment for containers
  - Each container gets its own process space, network stack, and file system
  - Underlying OS resources are shared among containers
- Container Registry
  - Repository for container images – can be public or private
- Container Orchestration Platforms
  - Registry + Runtime + Lots of other features

How do Containers help with Agile/DevOps/CD?

Containers help increase velocity, quality, and repeatability of software delivery by providing a uniform means of application distribution (the container image) that includes not only the application artifact(s), but all its dependencies and environment as well.

docker

rkt

Why we love containers, specifically

- Faster startup & shutdown
  - Not booting the whole OS every time
- Smaller than VMs
  - Usually…beware of image bloat
- Portable – build once, run anywhere
  - Great for distributing build environments to devs
- Better environment fidelity throughout the pipeline
  - No need for ops to use the Magic 8 Ball as much
- Great match for microservices
- Separation of concerns
  - Applications are decoupled from the infrastructure they run on
- Higher density
  - Better resource utilization at scale
- Orchestration platforms provide scaling, resiliency
- License savings (one license per host, multiple containers on the host)

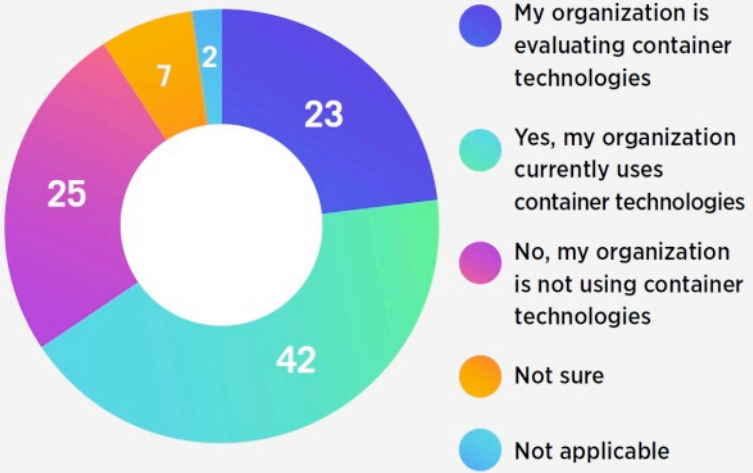
But I heard containers aren't secure?

- Container security has come a long way, quickly
- Applications deployed in containers are arguably *more* secure than applications deployed on the bare OS/VMs
- Containers present a smaller attack surface and more isolation in case the application itself is compromised
- Images are built programmatically, so less snowflakes and environment drift
- Environments can be secured earlier in the software pipeline since the environment is part of the container
- Easier to provide immutable configurations
- Software pipeline can (and should) be configured to inspect containers for security issues

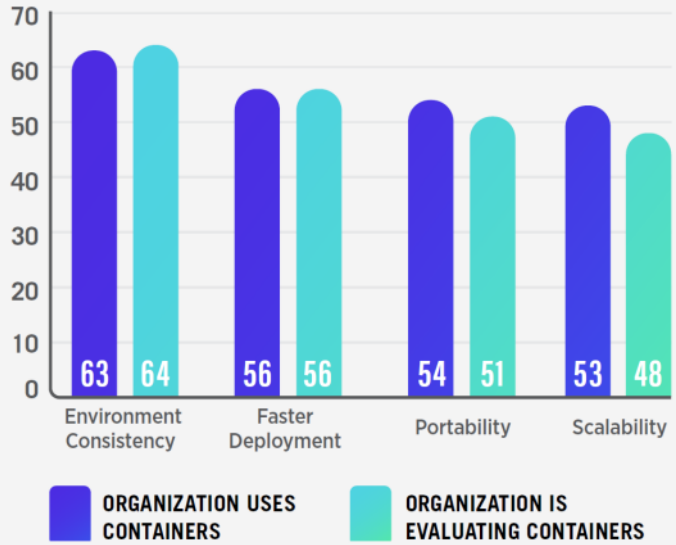
Really?
That all sounds too good to be true!

Teams are realizing the benefits

Does your organization currently use container technologies?



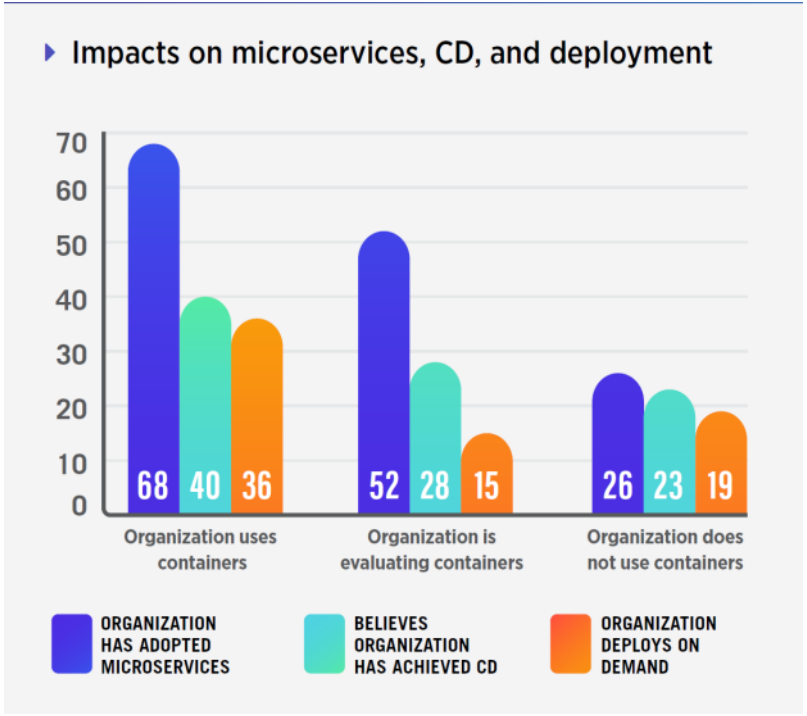
Benefits to the organization expected/received from containers



https://dzone.com/guides/orchestrating-and-deploying-containers

Electric Cloud

But there are challenges



https://dzone.com/guides/orchestrating-and-deploying-containers

Electric Cloud


A brief diversion into microservices
(or: why application architecture matters)
downloaded from pickywallpapers.com

Microservices Architecture

A suite of services, each focused on doing one thing well

- Independently developed
- Independently deployable
- Exposes an API
- Runs in its own process

Loosely coupled architectures are the strongest predictor of continuous delivery

"Gather together those things that change for the same reason, and separate those things that change for different reasons."

– Robert Martin

What's cool about Microservices?

Divide and conquer complex distributed applications

Loose coupling

Makes it easier to adopt new technologies

Smaller more autonomous teams are more productive – better resource utilization

Monolithic Apps

The good, the bad, the ugly…

Pros

- Can be easier to develop
- Can be easier to test
- Can be easier to deploy

Cons

- Easier to produce spaghetti code
- Can be harder to integrate new technologies
- Can be harder to learn and understand the code
- **You have to scale everything to scale anything**
- **Can’t deploy anything until you deploy everything**

Why Microservices in Containers? (The PB&J Part)

- 2002: One service per metal box
  - “I remember my first dual-core box, too!”
  - “Why is that 32-core server idle all the time? Can I have it?”
- 2007: Hypervisor + 1 VM + Multiple services in that VM
  - “Yeah, can’t run ServiceA and ServiceB side by side, conflicting versions of…”
  - “Yeah, we did that until ServiceC filled up /tmp and took down ServiceD"
  - “Yeah, we tend to run ServiceE by itself once we’re past QA”
- 2012: Hypervisor + Multiple VMs + 1 Service in each VM
  - “Yeah, each VM OS has a copy of that in memory, so…”
- 2013: Containers: run multiple services in isolation without the OS overhead

Enterprise Microservices & Container Needs

- Heterogeneous Architectures
  - Traditional Monolith
  - Microservice Only
  - Hybrid
- End-to-end Pipelines
  - Model Containers & Microservices
  - Container Pipeline Orchestration

And now back to containers…

Containers: Bet You Can’t Run Just One…

If you just want to run a couple of containers on your laptop, that’s easy.

But if you want to operationalize containers for production use and enterprise scale, you’re going to need container orchestration.

Container Orchestration

Container orchestration platforms typically provide a container runtime, registry, plus features that help you manage containers at scale

What Does Container Orchestration Help With?

- Container lifecycle management
- Auto scaling
- Self-healing
- Networking, routing, load balancing, ingress control
- Service binding/discovery
- Namespaces (including DNS)
- QOS, node affinity/anti-affinity
- Storage (persistent and otherwise)

Azure
Container Service

Electric Cloud

Software Pipeline Best Practices for Container Delivery

Electric Cloud

Software Delivery has changed…

Large App → Few Releases

Small & Modular App(s) → Many Releases

Best Practices for CD Pipelines of Container-based Apps

- Your Automated Software Pipeline Is Your Friend™
  - Ideally, one platform handles all your software delivery
  - Are your tests automated? Really automated?
  - How's your test coverage?
- Self-service automation/ChatOps approaches
  - Reduce onboarding time, waiting, complexity
- Your solution should provide a real-time view of all the pipelines' statuses and any dependencies or exceptions.
- Make sure your deployment pipeline plugs into your monitoring so that alerts can trigger automatic processes such as rolling back a service, switching between blue/green deployments, scaling and so on.

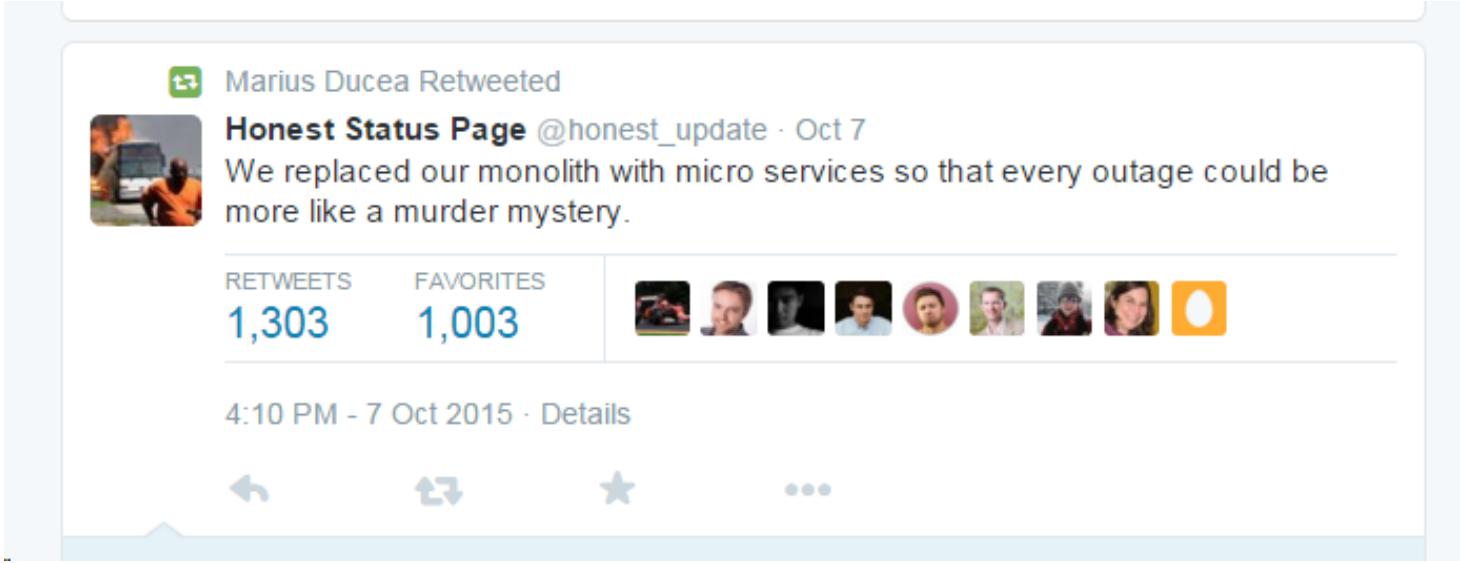
Best Practices for CD Pipelines of Container-based Apps

- One repository per micro-service, if you're using them
- Independent CI and Deployment pipelines per container
- "Automate all the things": plug in your toolchain to orchestrate the entire pipeline (CI, testing, configuration, infrastructure provisioning, deployments, application release processes, and production feedback loops.)
- Your pipeline must be tools/environment agnostic to support each team's workflow and tool chain
- Test automation tools and service virtualization are critical

Best Practices for CD Pipelines of Container-based Apps

- Track artifacts/images through the pipeline (who checked-in the code, what tests were run, pass/fail results, on which environment it was deployed, which configuration was used, who approved it and so on)
- Bake in compliance into the pipeline by binding certain security checks and acceptance tests
- Allow for both automatic and manual approval gates into and out of pipeline stages
- Create reusable models/processes/automation for your various pipelines

The Importance of Monitoring



APPDYNAMICS

dynatrace

sysdig

Twistlock

Electric Cloud

https://neo4j.com/blog/managing-microservices-neo4j/

Electric Cloud

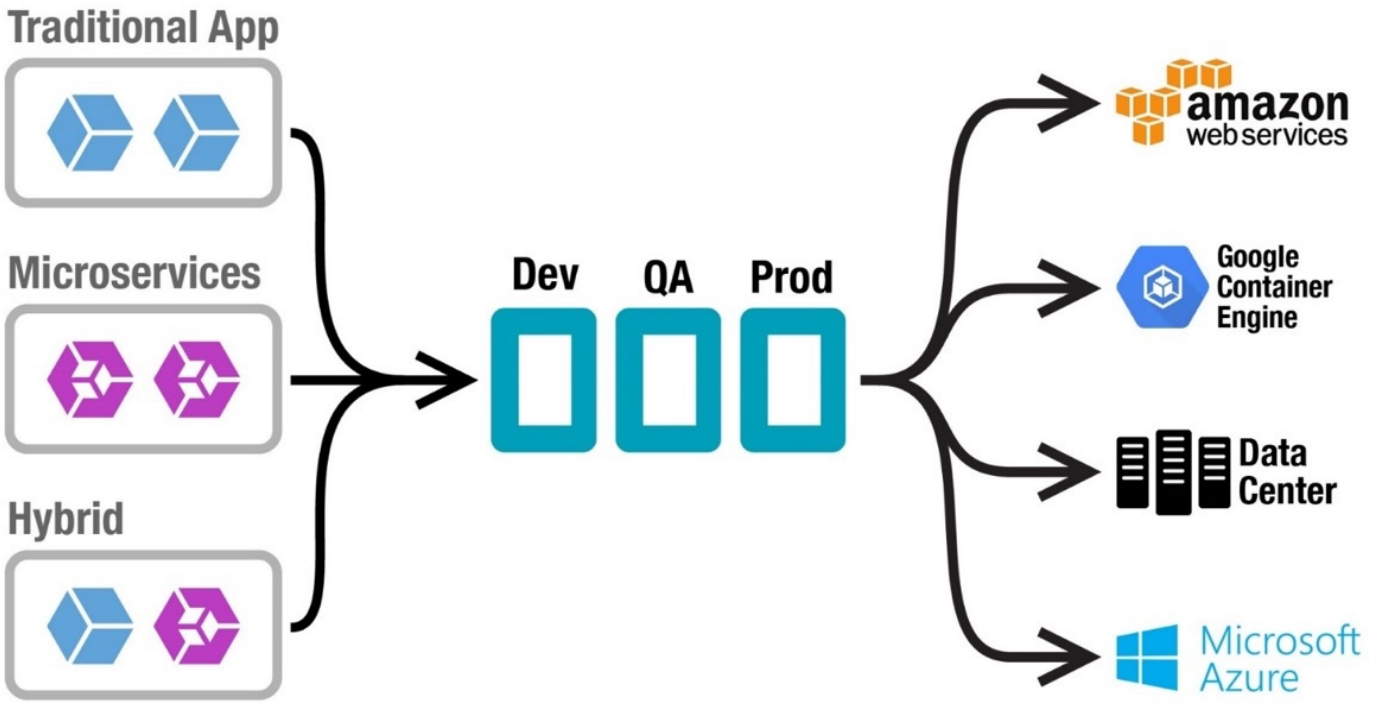
Monitoring Best Practices

- All services should log and emit monitoring data in a consistent fashion (even if using different stacks)
- Monitor latency and response times between services
- Monitor the host (CPU, memory, etc)
- Aggregate monitoring and log data into a single place
- Log early, log often
- Understand what a well-behaving service looks like, so you can tell when it goes wonky
- Use techniques like correlation ids to track requests through the system
  - “So then requestId 0xf00dfee8 in the log on ms-app-642-prod becomes messageId 1125f34c-e34e-11e2-a70f-5c260a4fa0c9 on ms-route-669-prod?”

Using Containers & Microservices in the Software Pipeline

Electric Cloud

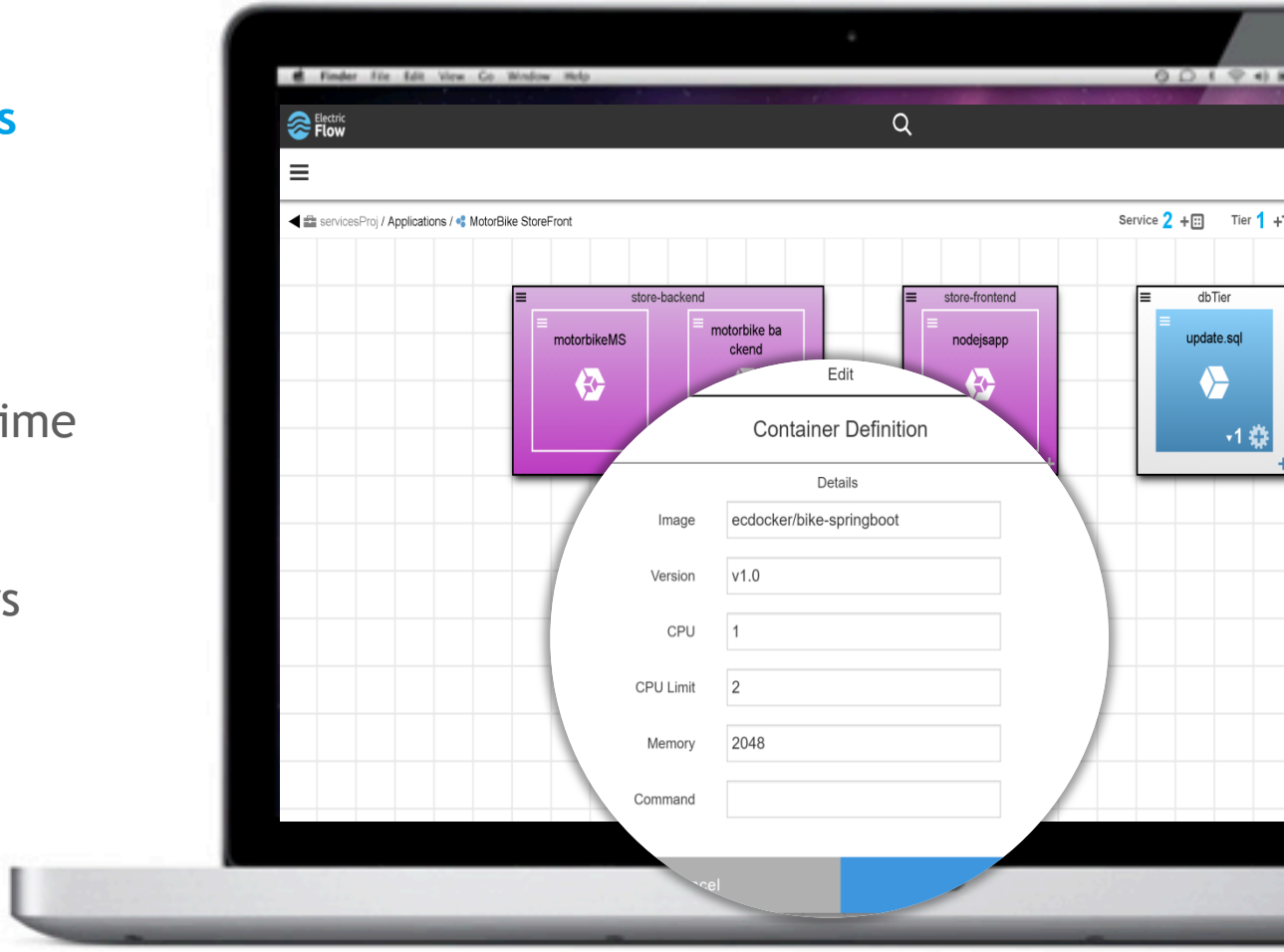
End-to-End Container Delivery Management



Electric Cloud

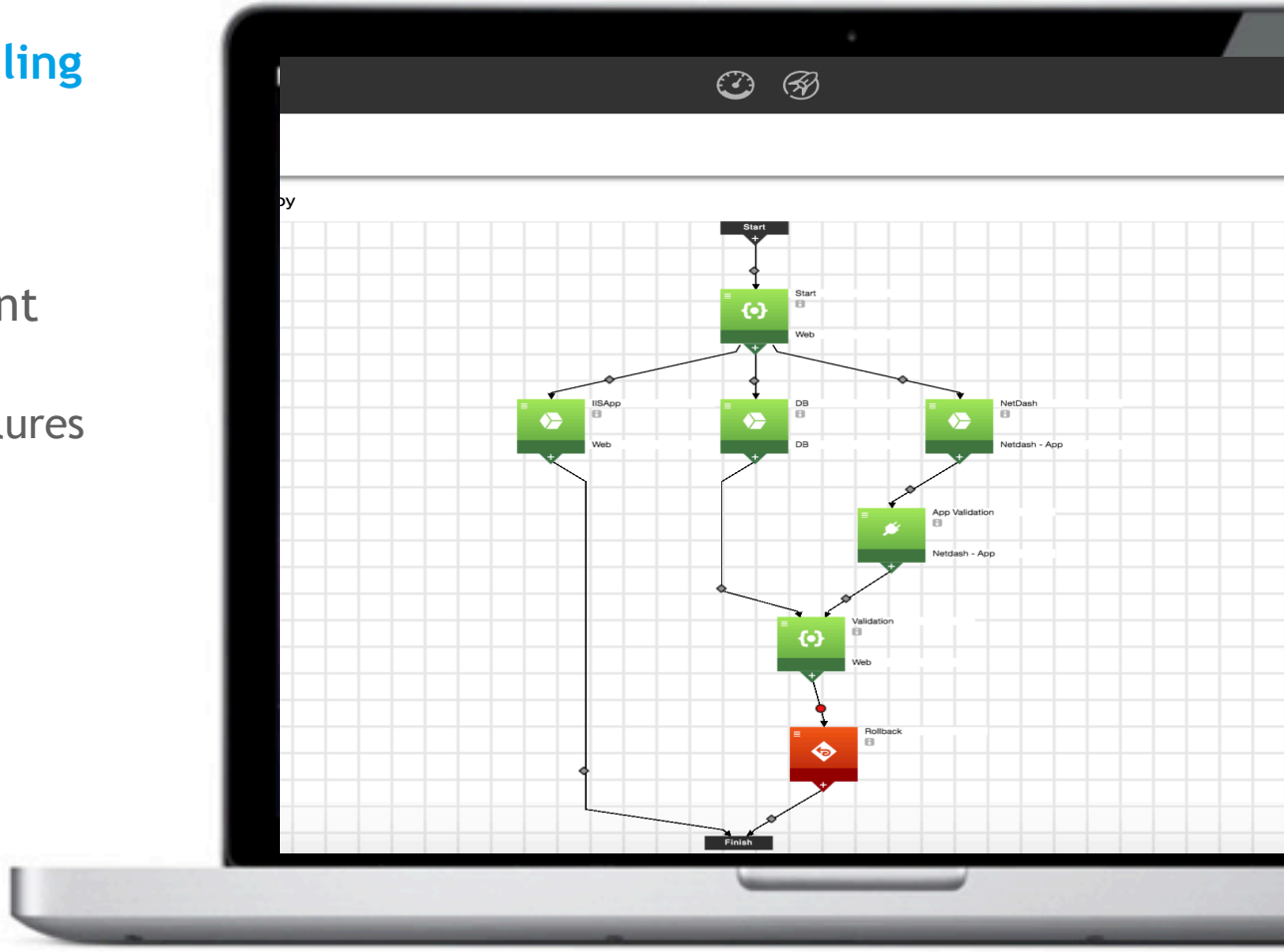
Model your Container- & Service-based applications

- Model Any type of application: Monolithic, Microservices or Hybrid
- Application definition is independent of the runtime environment
- Deploy individual microservices/containers independently



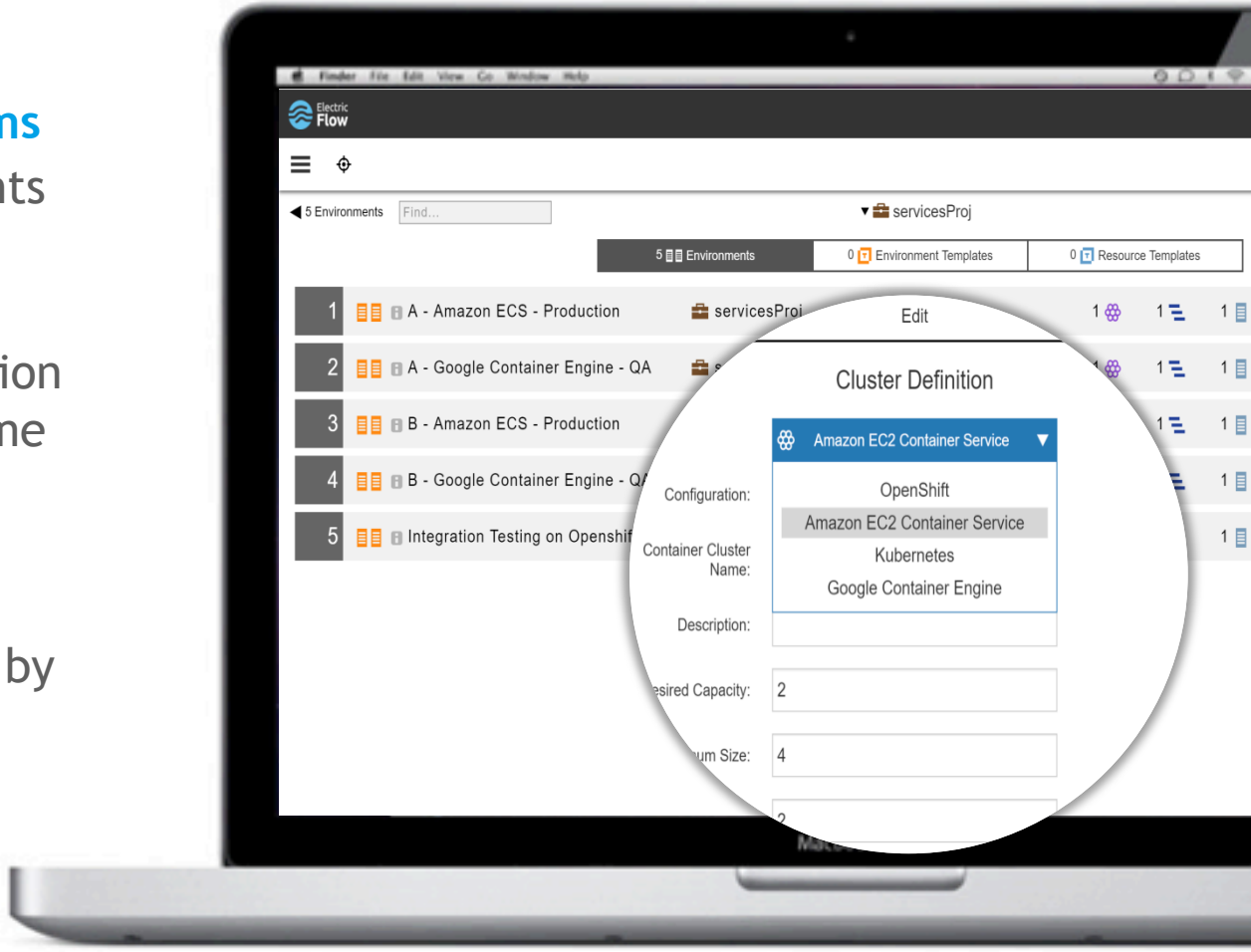
Flexibility and Error Handling in Deployment Processes

- Conditional execution
- Error handling and complexity in deployment process
- Automated Rollback on Failures



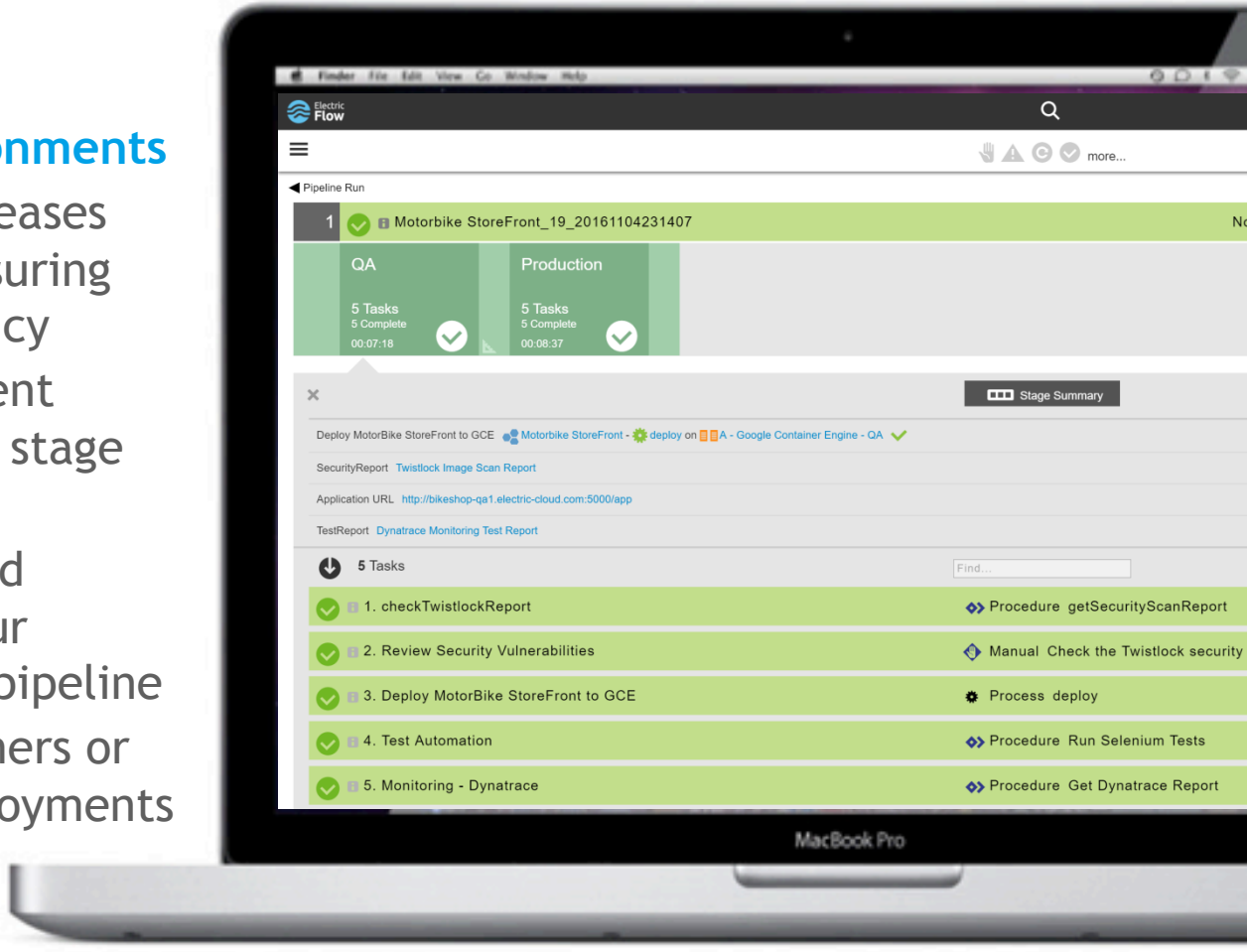
Model-driven approach for container runtime platforms

- Model hybrid environments
- Avoid lock-in to a single platform
- Model the dynamic creation of clusters on your runtime platform
- Enable scaling policies, networking and other infrastructure supported by the runtime platform



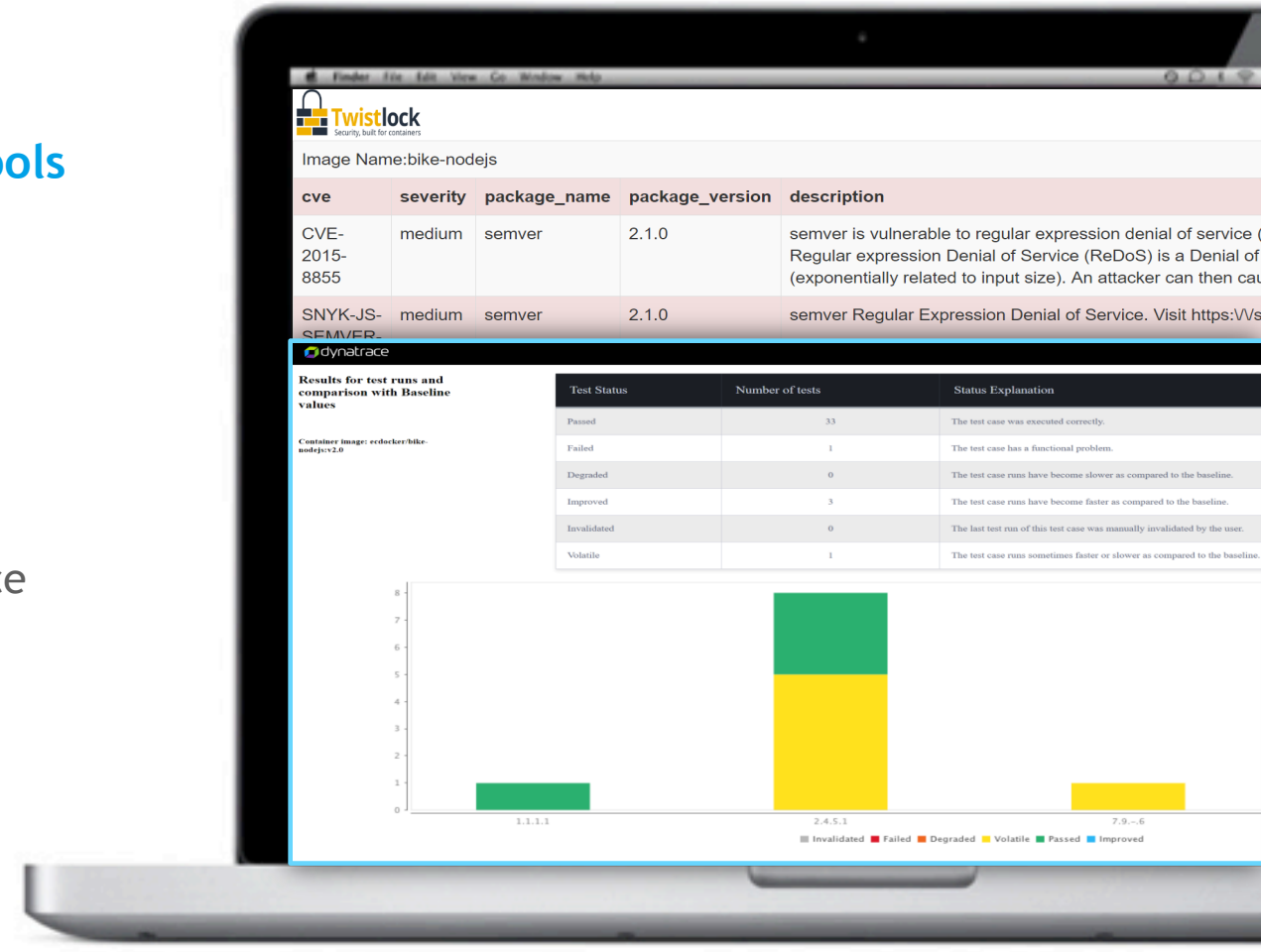
DevOps Release pipeline orchestrating container deployments across environments

- Coordinate container releases across environments, ensuring repeatability & consistency
- Orchestrate all deployment activities as part of each stage of the pipeline.
- Incorporate approvals and compliance checks in your automated, data-driven pipeline
- Deploy individual containers or coordinate multiple deployments in one pipeline

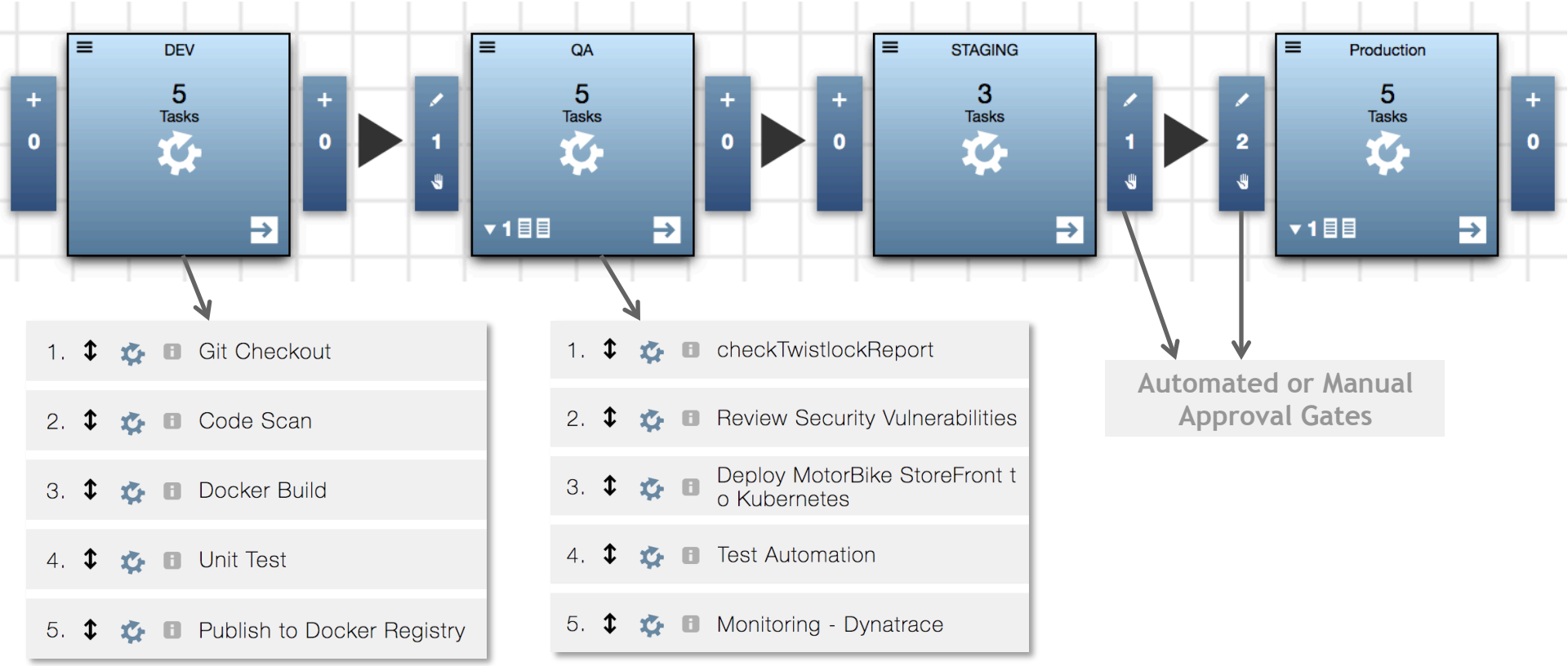


Orchestrate all of your tools processes as part of your DevOps Release Pipeline

- Shift-Left Monitoring
- Vulnerability tracking
- Testing tools
  - Including Test Data Management and Service Virtualization
- ITSM approvals and compliance
- Approvals and Gates



Container Release Pipeline

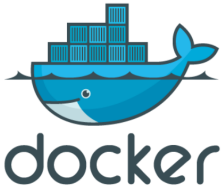


Electric Cloud


Questions?

Thank You!

Anders Wallgren | CTO, Electric Cloud
@anders_wallgren



Azure
Container Service

Electric Cloud