MANICODE

SECURE CODING EDUCATION

Zero to DevSecOps:
Security in a DevOps World

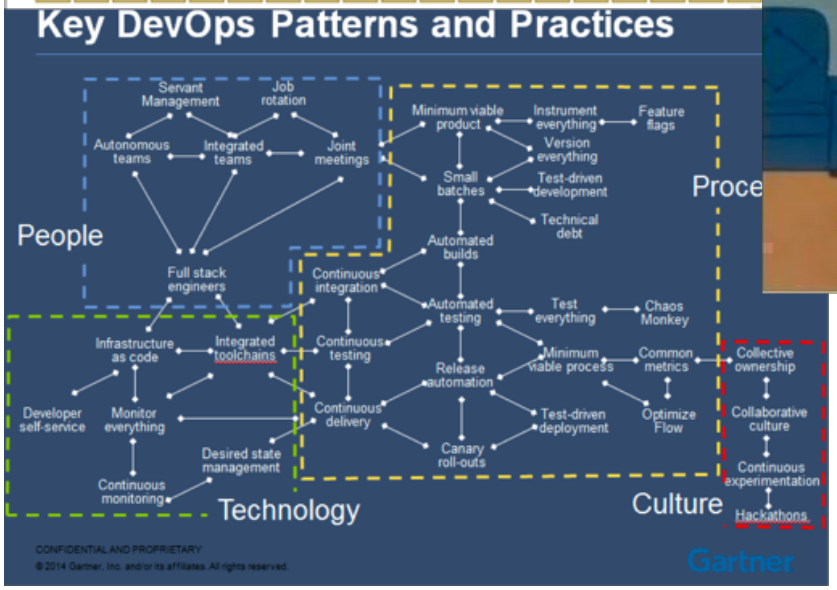
A little background dirt…

@jimmesta

- 10 years of penetration testing, teaching, and building security programs
- OWASP AppSec California organizer and Santa Barbara chapter founder
- Conference speaker
- Been on both sides of the InfoSec fence
- Loves Clouds

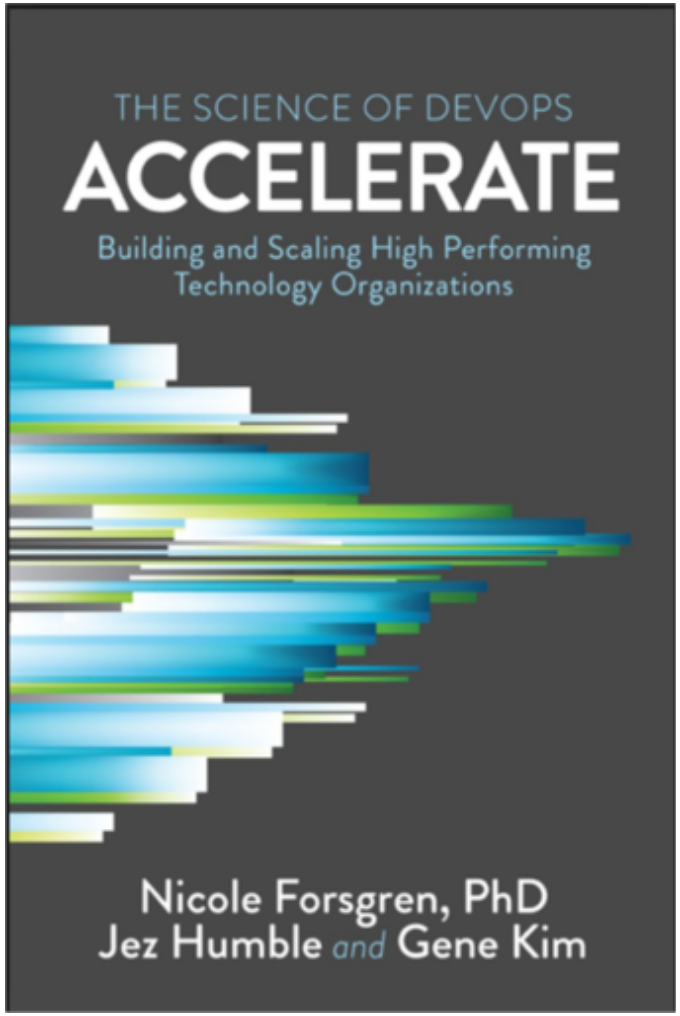
We Have a “Situation”

The Situation



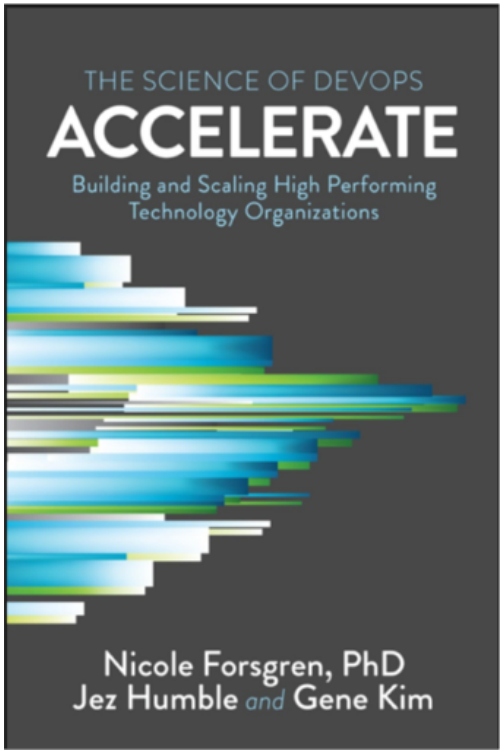
The (Actual) Current State of Affairs

"Our research has uncovered 24 key capabilities that drive improvements in software delivery performance in a statistically significant way."



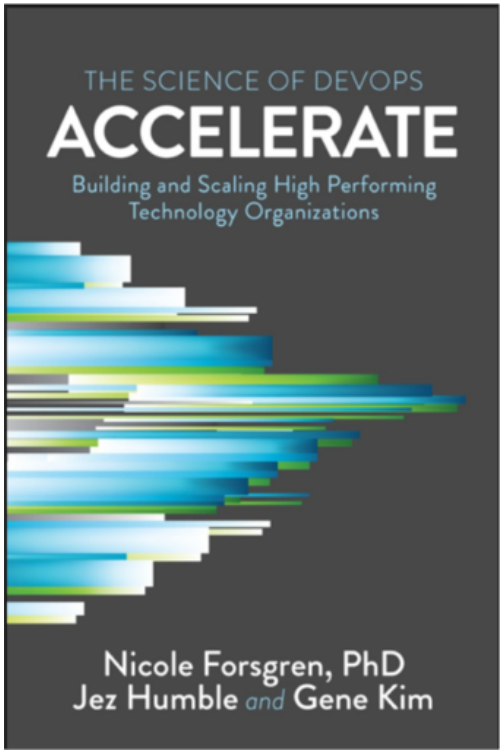
Continuous Delivery Capabilities

- Version Control
- Deployment Automation
- Continuous Integration
- Trunk-Based Development
- Test Automation
- Test Data Management
- ***Shift Left on Security***
- Continuous Delivery



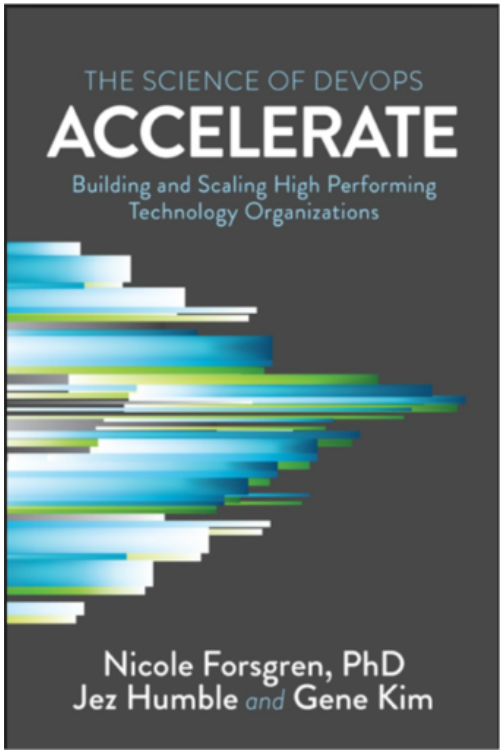
Architecture Capabilities

- Loosely Coupled Architecture
- Empowered Teams
- Customer Feedback
- Working in Small Batches
- Team Experimentation



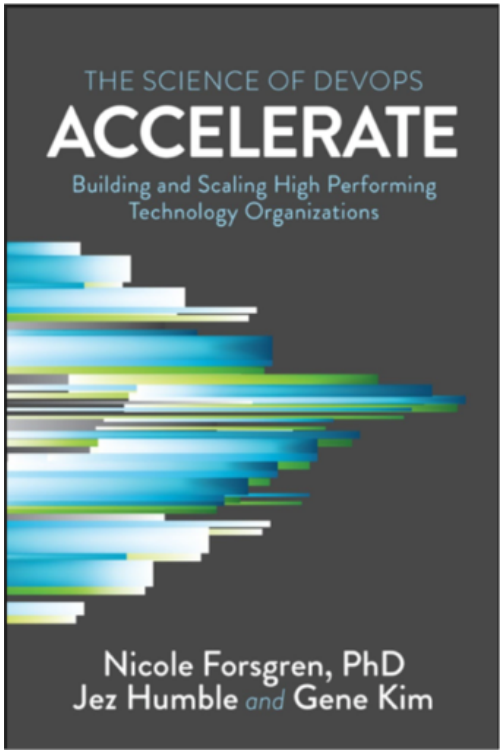
Lean Management and Monitoring Capabilities

- Change Approval Process
- Monitoring
- Proactive Notification
- WIP Limits
- Visualizing Work



Cultural Capabilities

- Supporting Learning
- Collaboration Among Teams
- Job Satisfaction
- Transformational Leadership



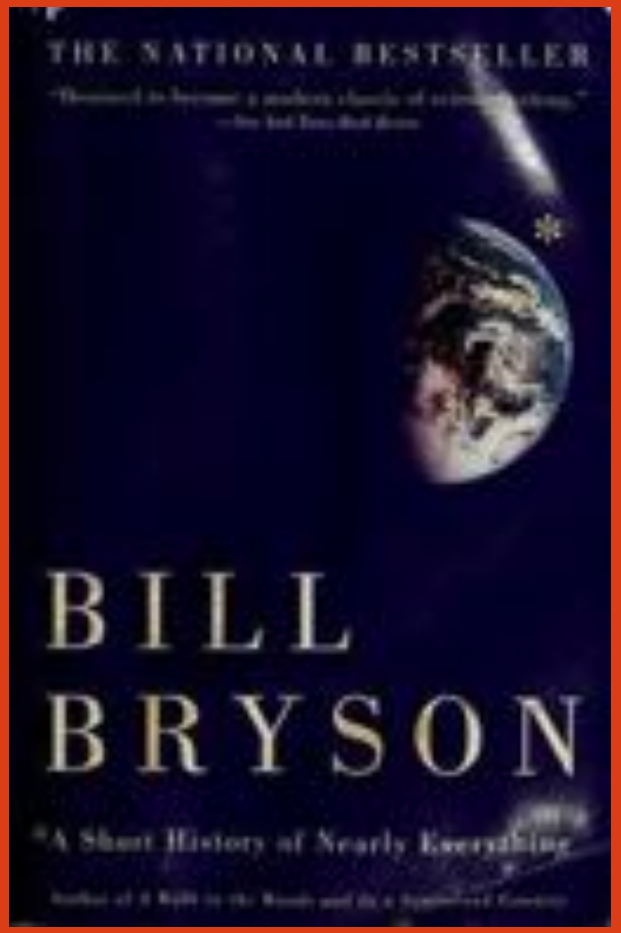
High Performers vs. Low Performers

- 46x more frequent code deployments
- 440x faster lead time from commit to deploy
- 170x faster mean time to recover from downtime
- 5x lower change failure rate

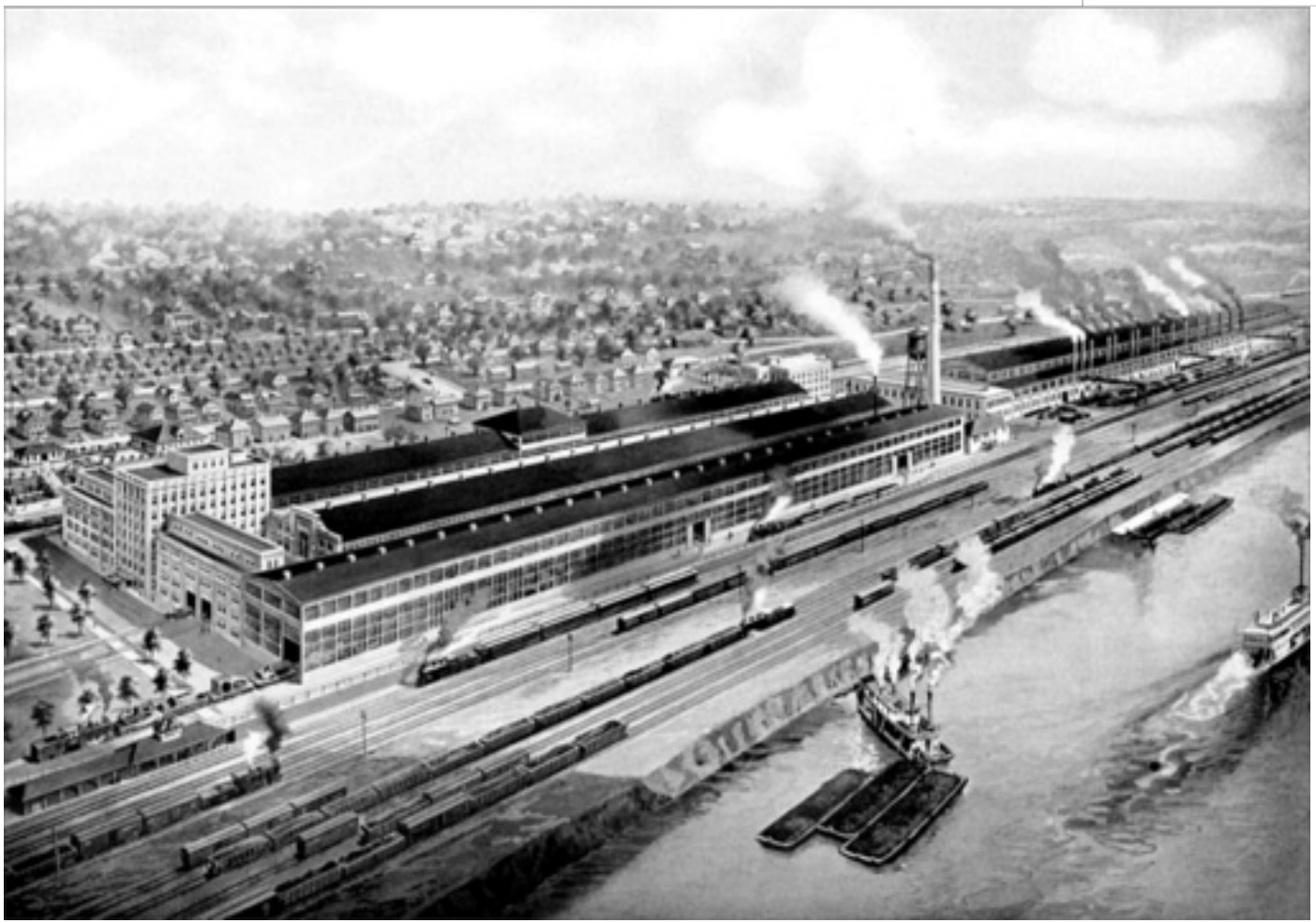
High Performing Security Teams

"High-performing teams were more likely to incorporate information security into the delivery process. Their infosec personnel provided feedback at every step of the software delivery lifecycle, from design through demos to helping out with test automation. However, **they did so in a way that did not slow down the development process**…"

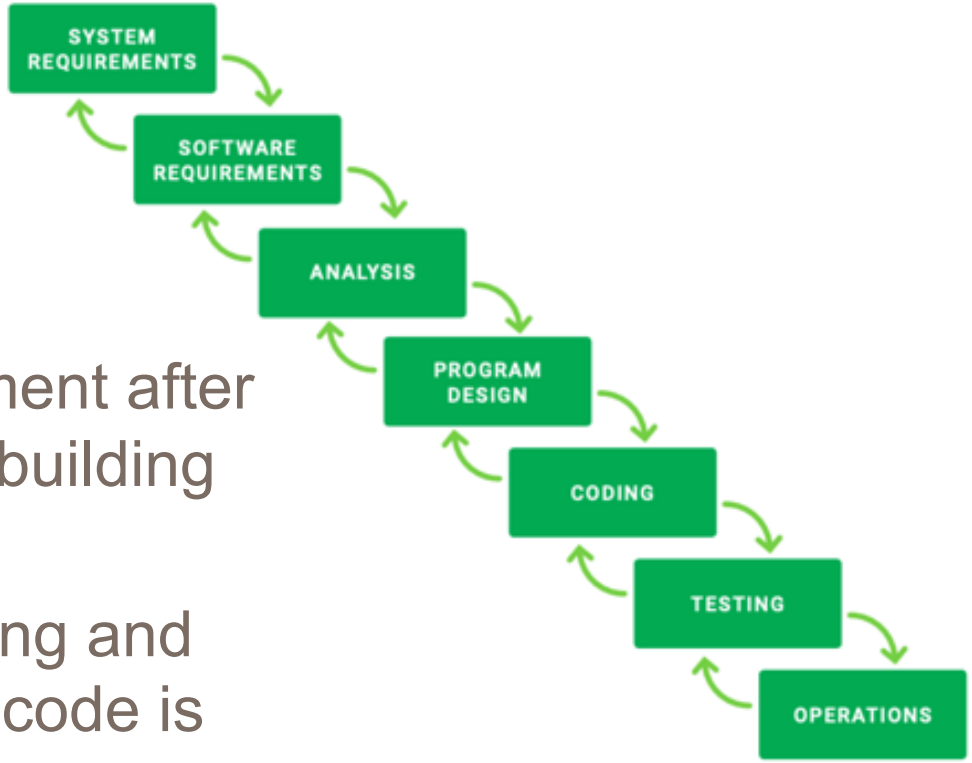
A Brief History of the SDLC



Part 1: The Waterfall Era



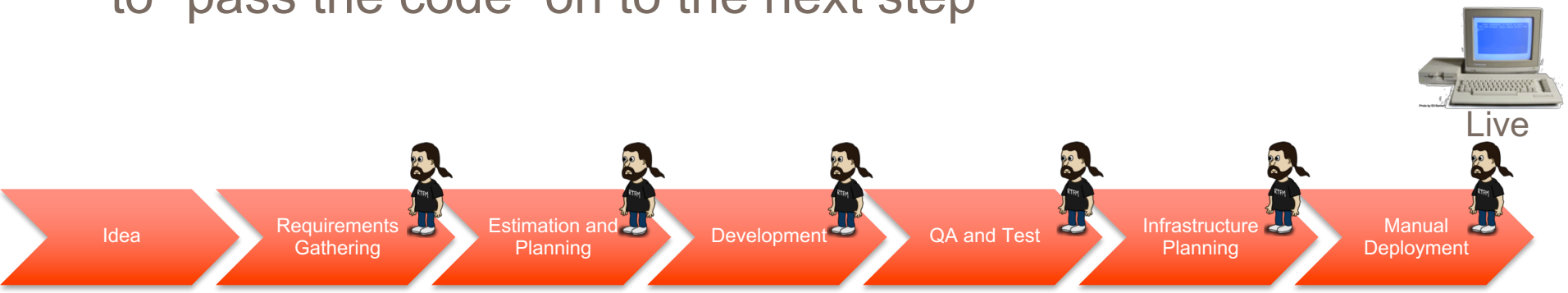
Part 1: The Waterfall Era



- Modeled software development after what we knew and learned building hardware
- Months (or years!) of planning and preparation before a line of code is written
- All good stories have to start *somewhere*

Traditional SDLC AKA “Waterfall”

- **Optimizes for risk management.** Assuming the cost of a mistake is high and tolerance for risk is low
- Critical services still benefit from certain ”waterfall” methodologies
- Linear progression when deploying software
- Relies heavily on human intervention and interaction to “pass the code” on to the next step



Part 2: The Agile Enlightenment

Putnam McDowell, left, and Chester Engineers President Al Baily

Alive and well

Mestek is a new 'chapter' in Mesta story

By William H. Wylie
The Pittsburgh Press

MESTEK INC., once the mighty Mesta Machine Co., is alive and apparently well after emerging earlier this year from a bankruptcy ordeal that lasted nearly two years.

Things are going so much better that Putnam B. McDowell, who steered Mesta through the tricky Chapter 11 maze, said, "Now I can sit down and have a drink with some of those lawyers and we laugh. . . . It's like war stories."

But the Mesta bankruptcy was no laughing matter during the grueling days of 1983 and '84 when the fate of the once "Cadillac" of mill machinery builders was being litigated in Federal Bankruptcy Court here.

Asked if he ever had any doubts about getting out of Chapter 11 — less than 10 percent of the companies that file make it — he replied, "About every third day for a year something disastrous seemed about to happen . . . But I never lost my basic faith that somehow we could work it out."

Thousands of employees and retirees were hurt financially by Mesta's collapse. Jobs were lost and some pension benefits were reduced by the government's Pension Benefit Guaranty Corp., which took over the fund.

The West Homestead plant, which housed one of the world's largest foundries, and the New Castle facility were sold, sounding Mesta's last hurrah as a manufacturer.

After distribution of $25.1 million in cash, more than 1 million shares of common and preferred stock and warrants to purchase common stock, notes totaling $4.7 million and deferred payments of $1.5 million, creditors received about 30 cents on the dollar.

Mestek is a mere shadow of its former self, with approximately 220 employees, total assets of $10.7 million and estimated annual revenues of $15 million to $18 million.

That contrasts sharply with the 3,000 who worked for Mesta during its heyday, assets of $74 million and annual sales as high as $120 million.

If it weren't for two Mesta subsidiaries and a joint venture with one of Victor Posner's companies — none of the subsidiaries was involved in the bankruptcy — there might not be a Mestek. The holding company's principal sources of income are The Chester Engineers Inc., a Coraopolis-based engineering firm, and MCS Inc., a Monroeville computer company.

Mestek's 49 percent interest in Mesta Engineering Co., which is owned jointly with Pennsylvania Engineering Corp.,

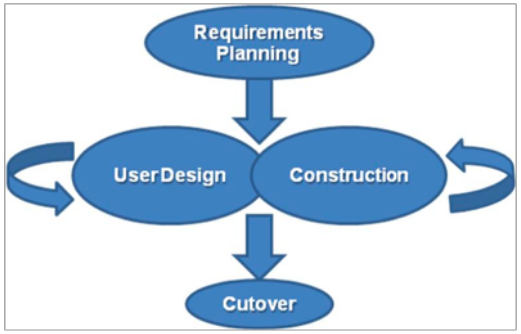
Please see Mestek, C5

Part 2: The Agile Enlightenment

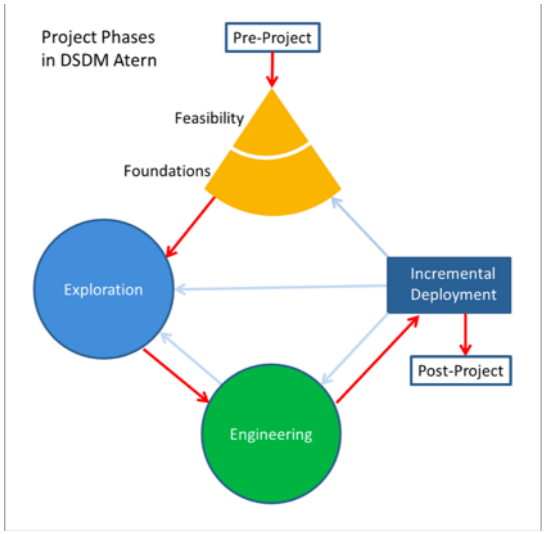
- Realization that software differs from hardware
- Competition emerges and first-to-market matters
- 90's was all about experimentations in effective software deployment
- Sprints, daily standups, retrospectives emerge
- Manual testing, QA, and deployment

Part 2: The Agile Enlightenment

Extreme Programming



Rapid Application Development (RAD) Model

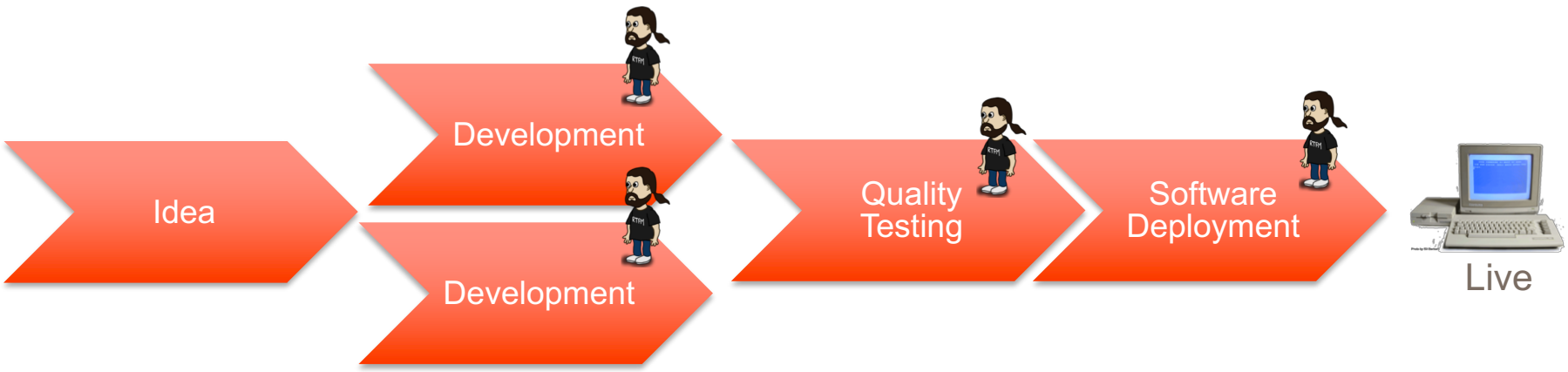


Dynamic Systems Development Method

Scrum

Agile / Scrum / Extreme

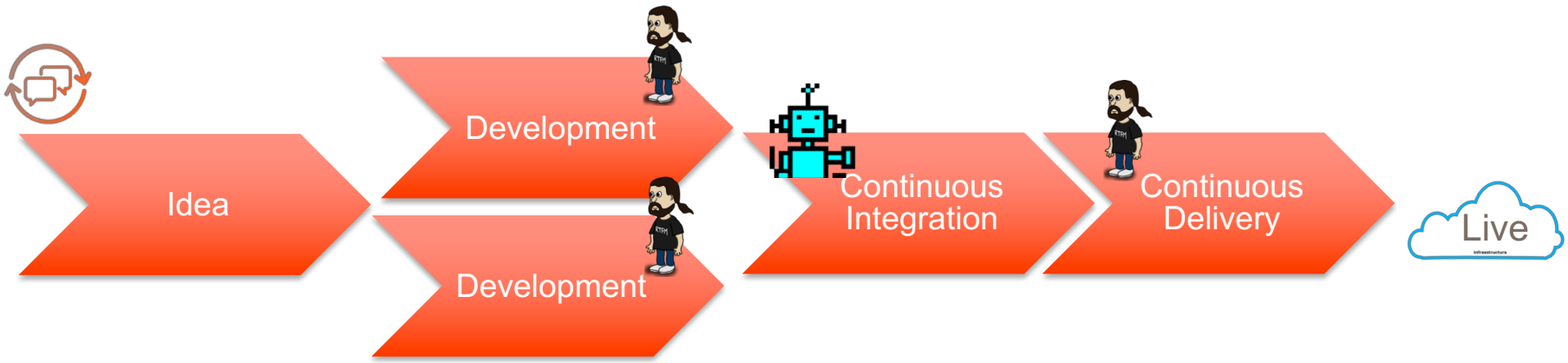
- Begin optimizing for speed and agility
- **Incremental changes**
- Beginning of TDD, timeboxing, stories, pair-programming, etc.
- We begin *thinking* about and measuring the effectiveness of our SDLC



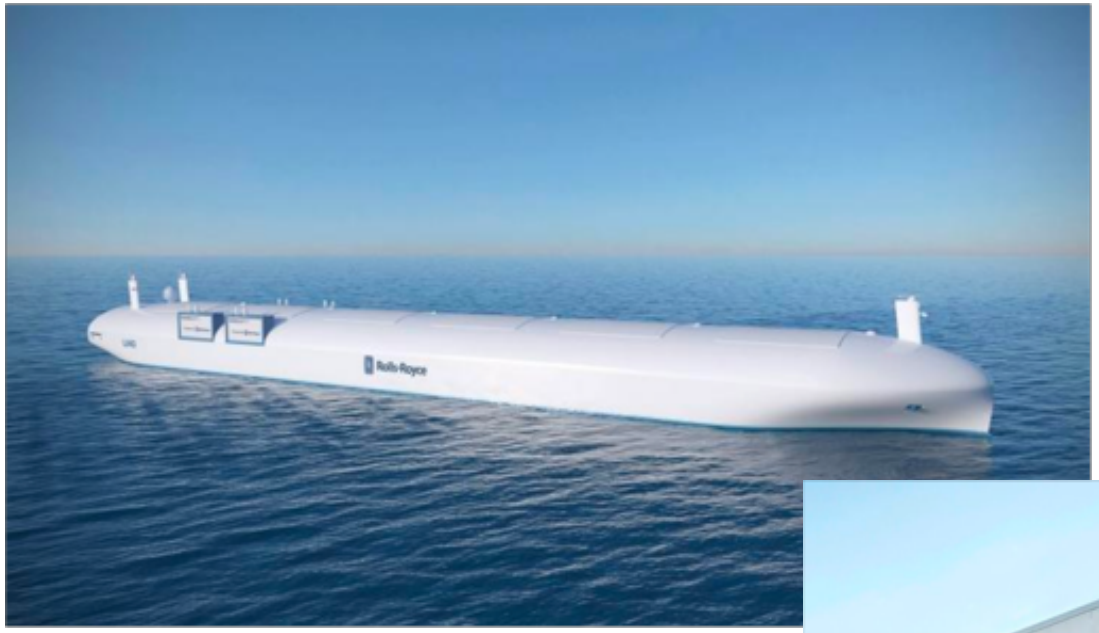
Part 3: Invasion of the Robots

Continuous Integration and Delivery

- Optimizes for speed and agility. Assuming the cost of a mistake is low and tolerance for risk is high
- Parallel and incremental changes
- Automation and upfront work makes this possible
- **Self-testing code and early days of automated QA**

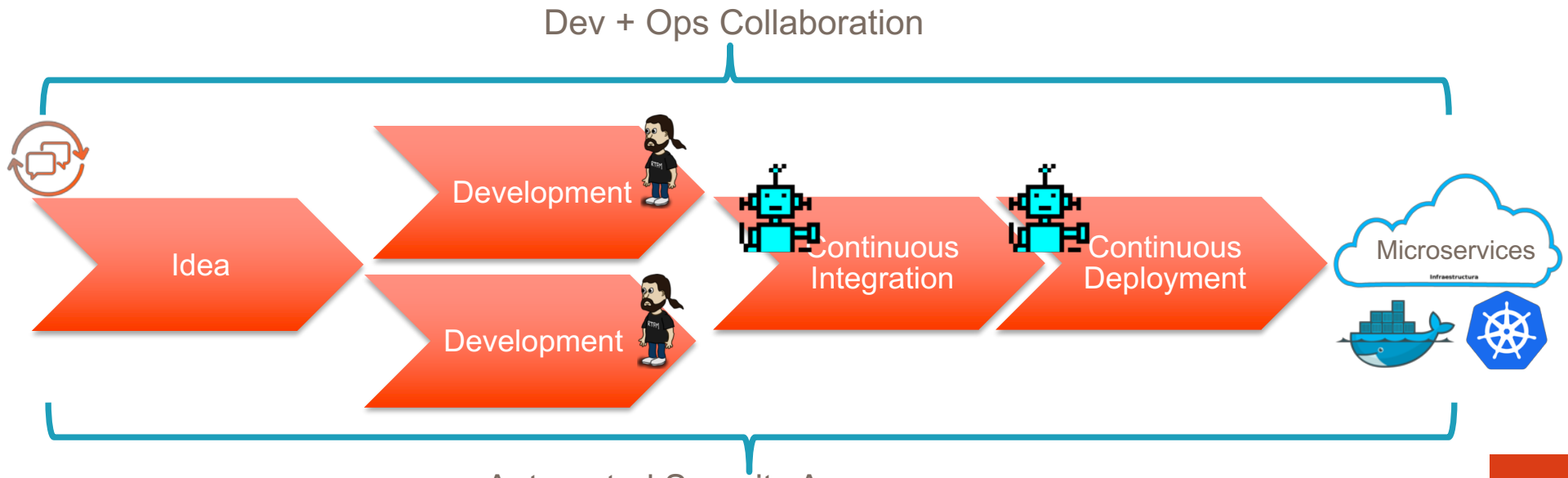


Part 4: The Current State of Affairs



DevSecOps

- Cultural shift towards end-to-end ownership of code
- Zero-downtime, automated deployments
- Emergence of containers, serverless, and zero-downtime deployments
- ”Everything-as-Code” is the new standard
- **Security is no longer a blocker or silo**

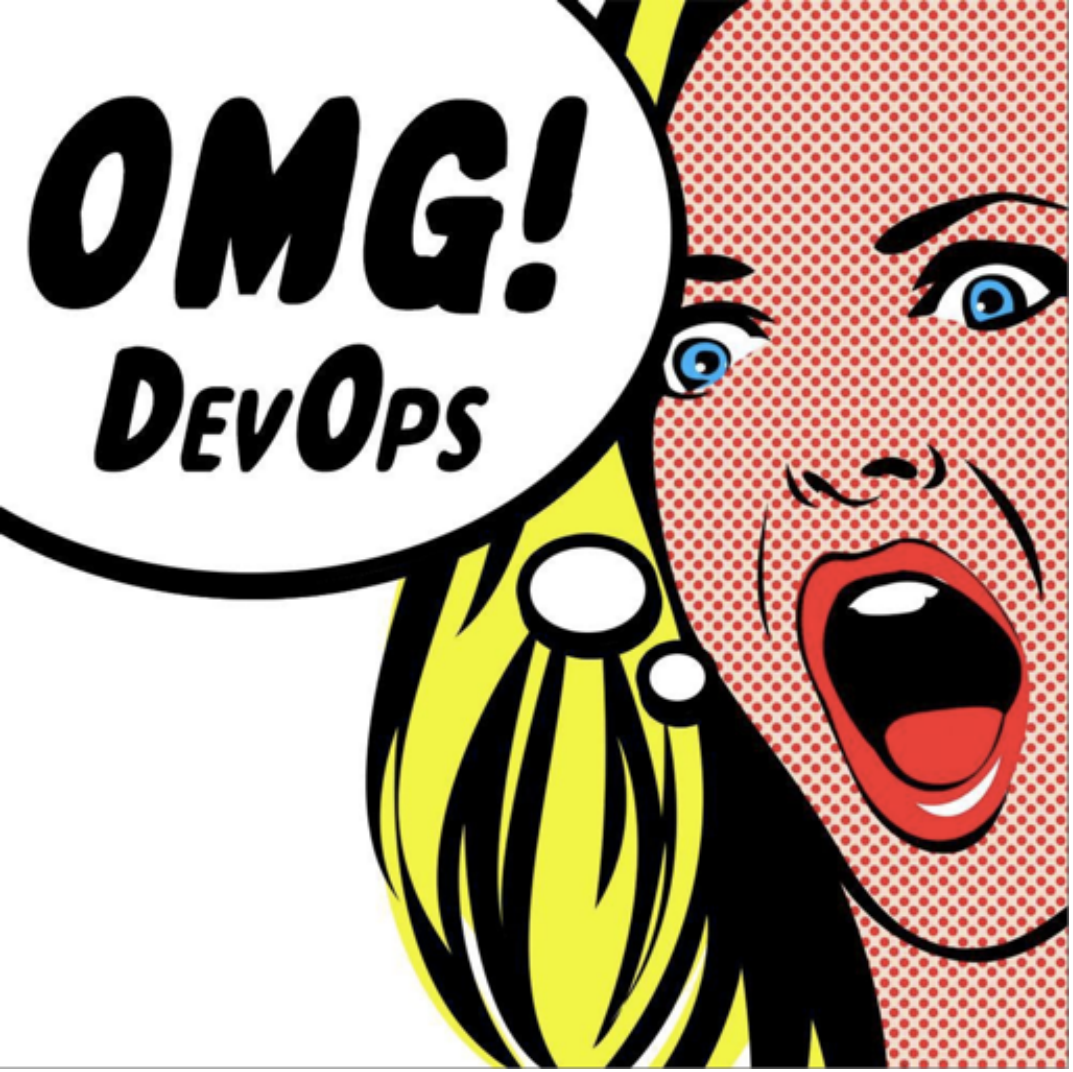


DevSecOps Advantages

- Add customer value
- Puts security in everyone’s job description
- Eliminate “black box” security teams and tools
- Ability to measure security effectiveness
- Reduce attack surface and vulnerabilities
- Increase recovery speed
- Save $$$
- Secure by default mentality

The Rest is History…

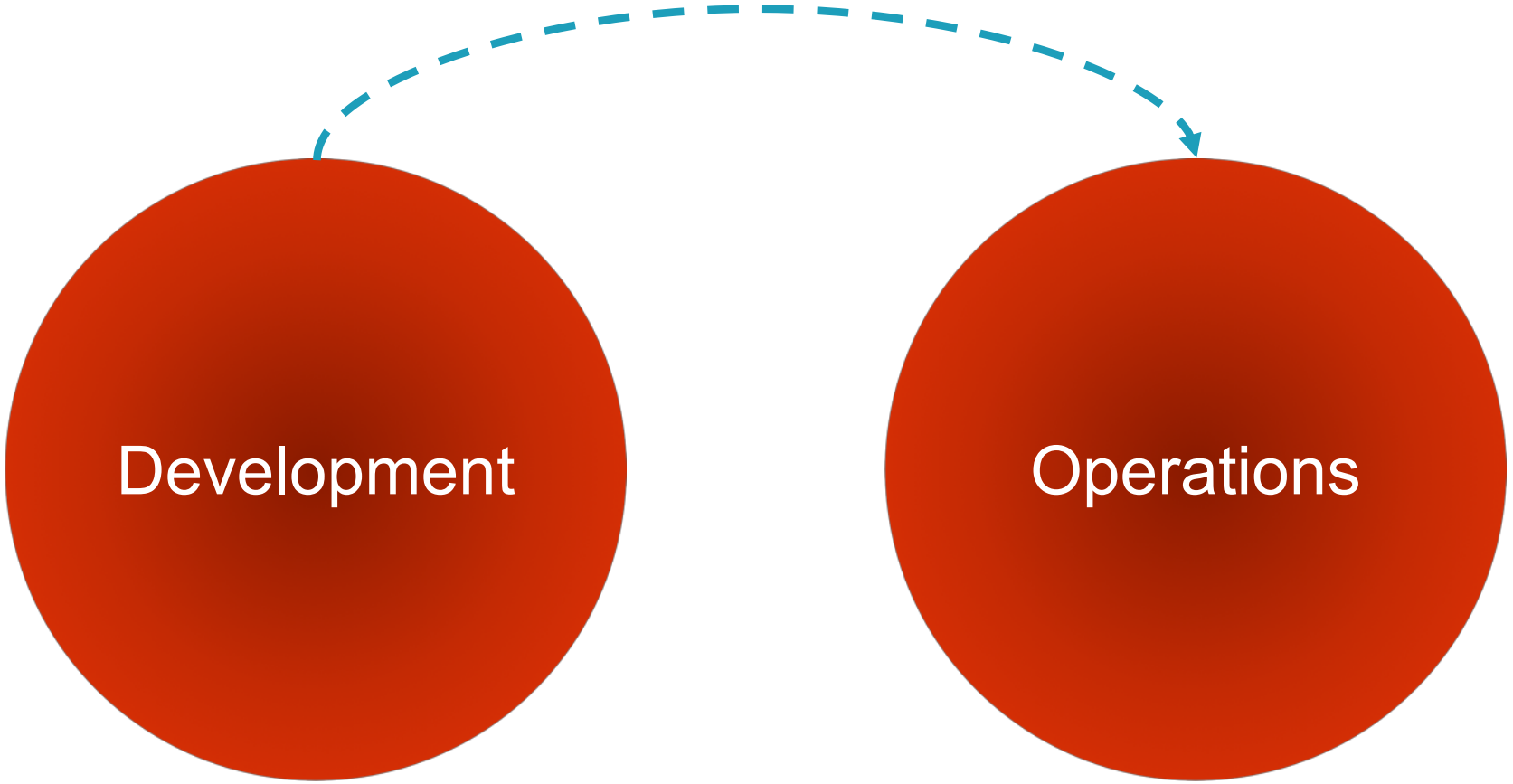
Introduction to DevOps



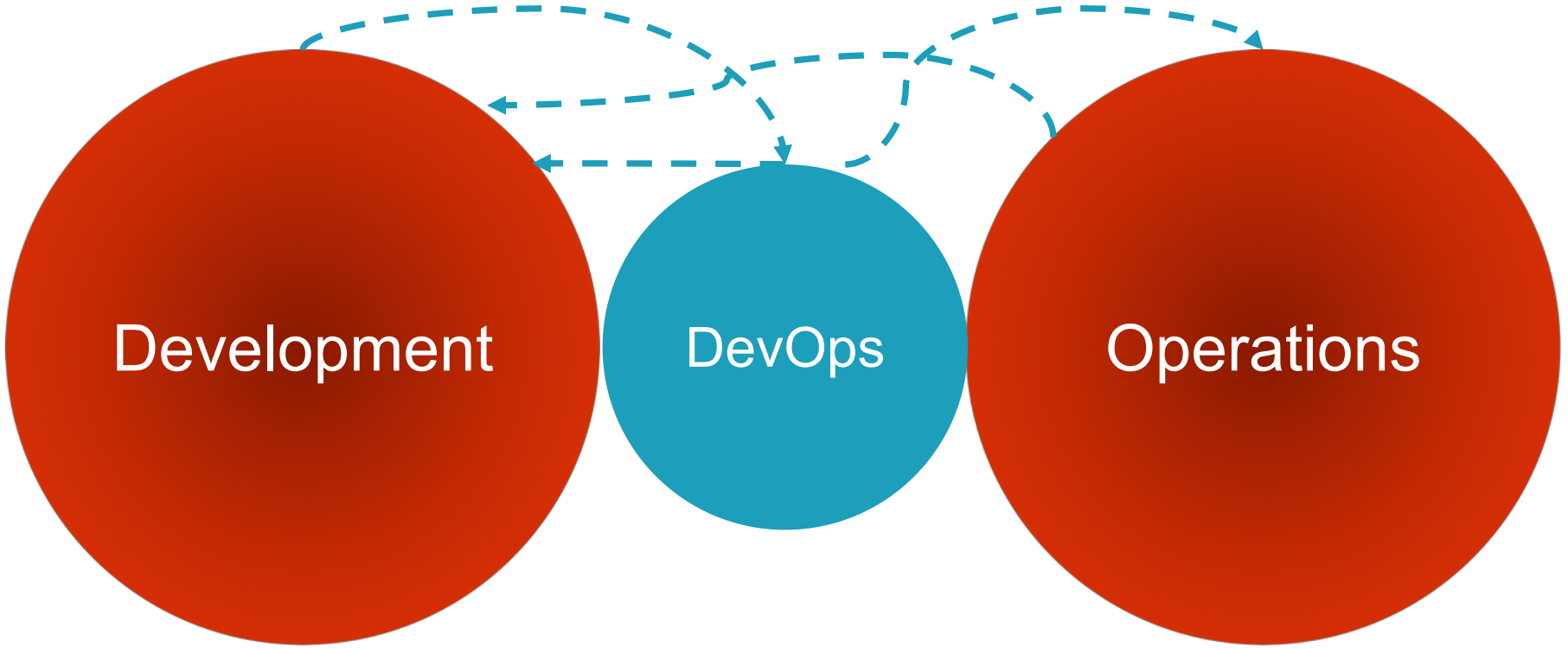
What *is* DevOps?

DevOps Anti-Patterns

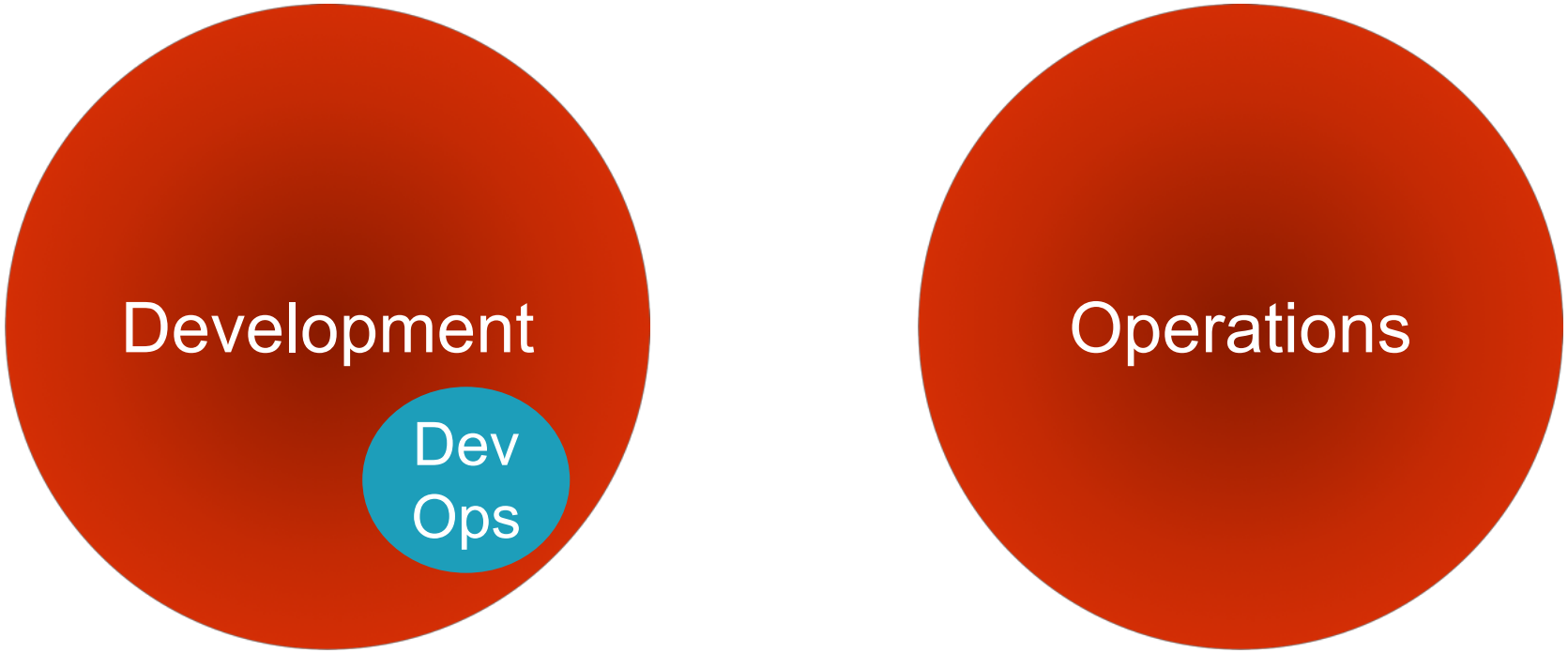
Anti-Pattern: “Throw it Over the Wall”



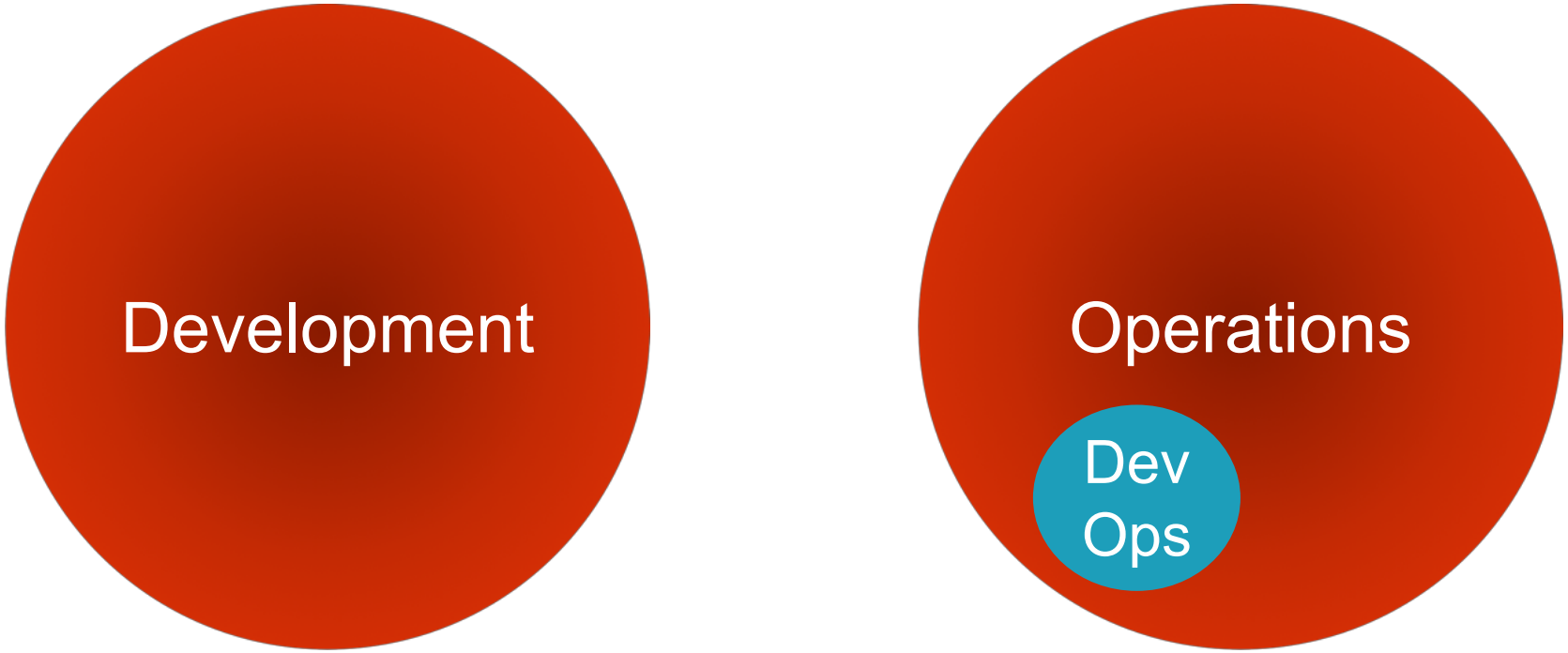
Anti-Pattern: “DevOps Team Silo”



Anti-Pattern: “NoOps” Approach

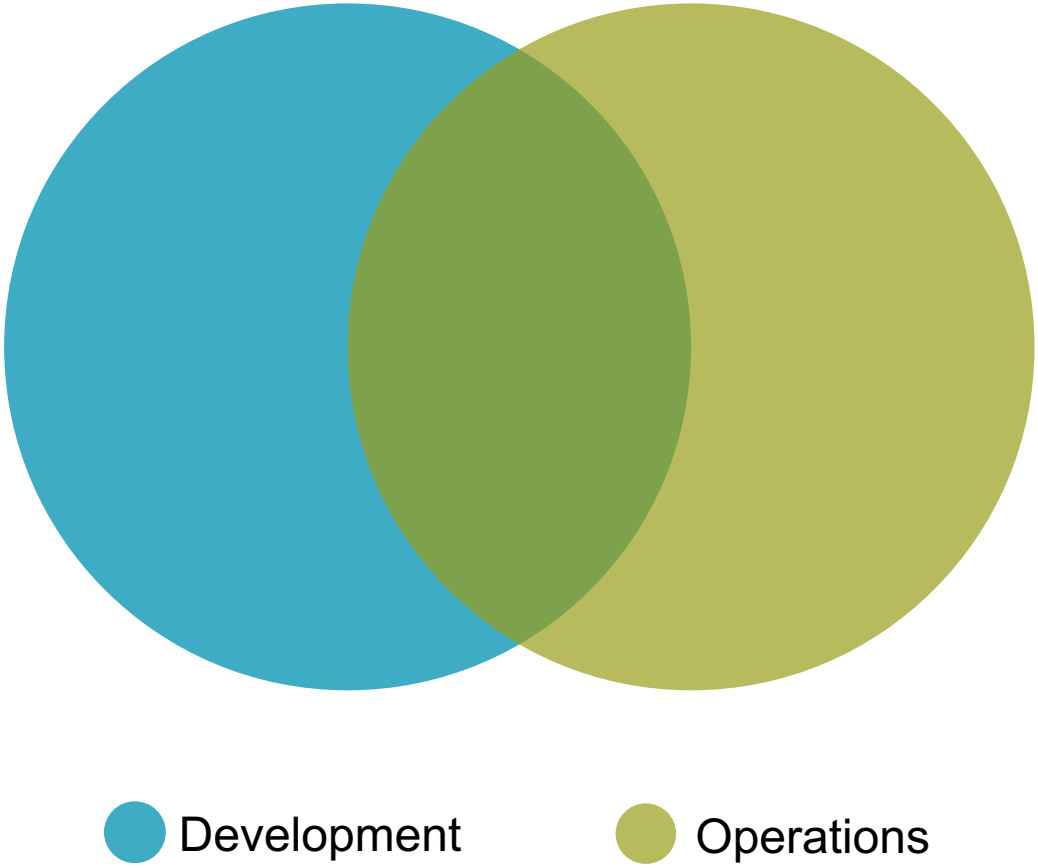


Anti-Pattern: “Ops Will Handle it”

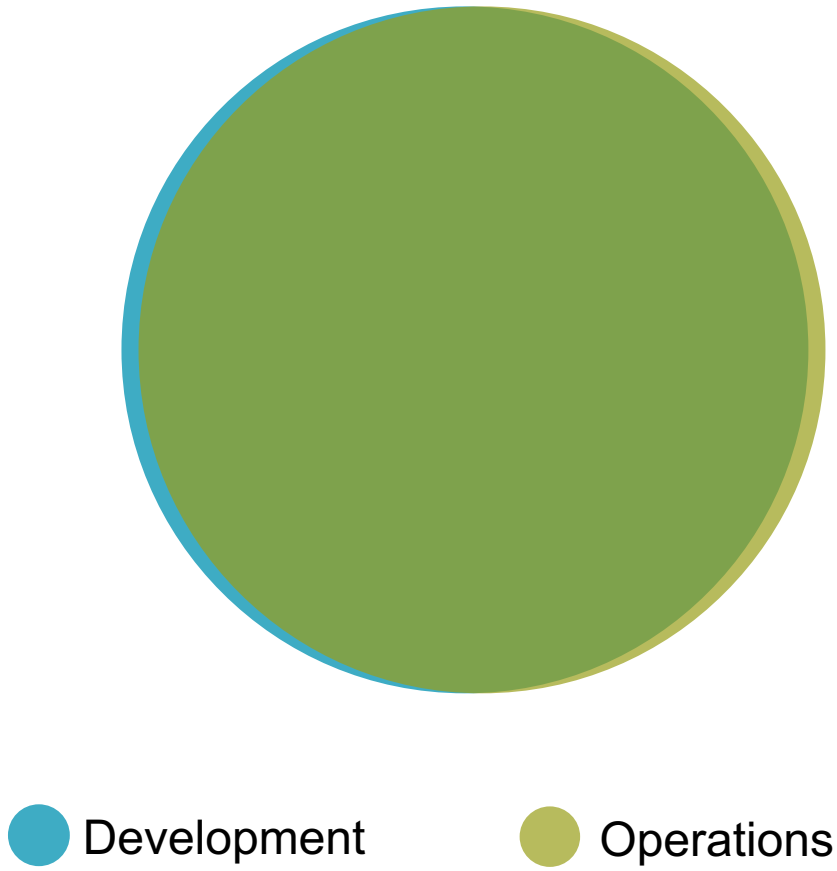


Anti-Pattern: “Ops Will Handle it”

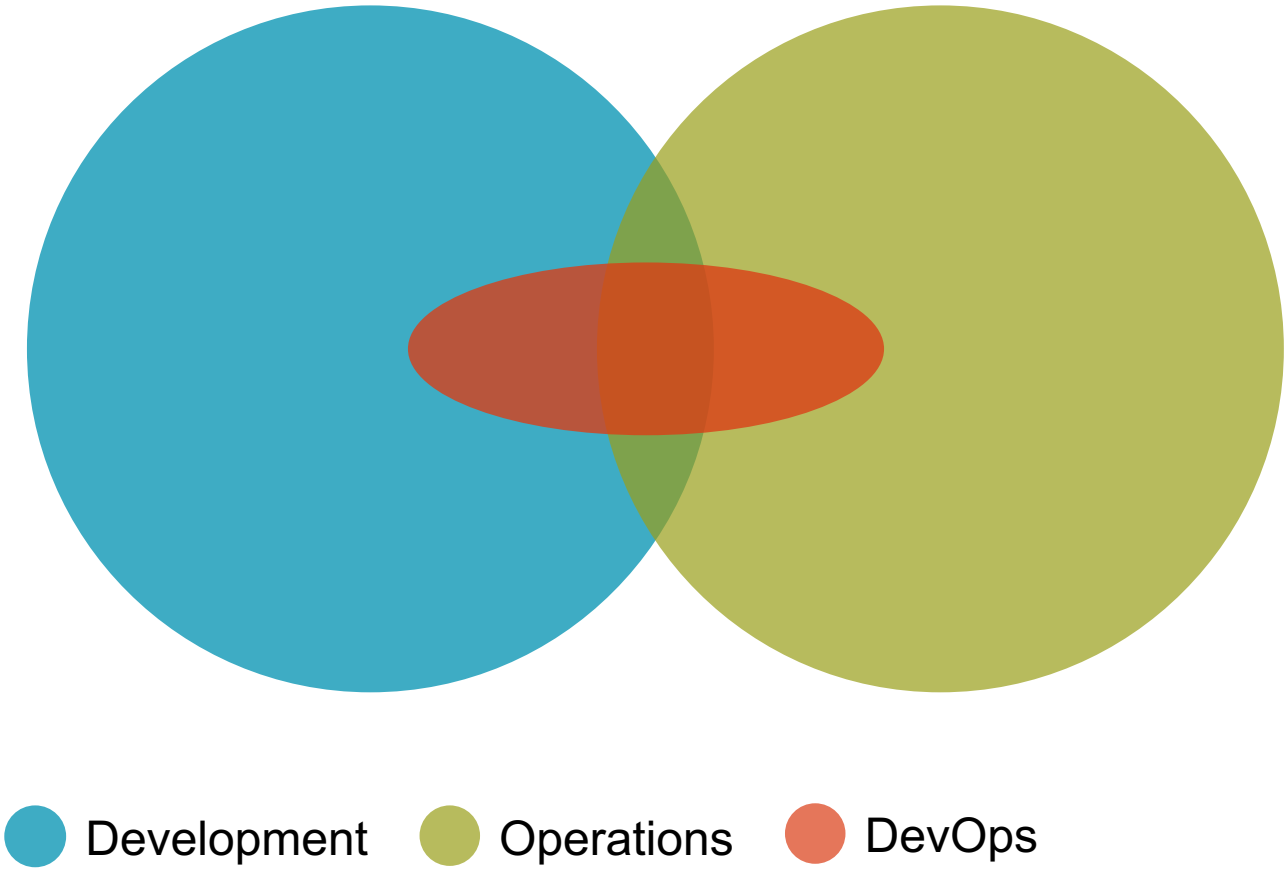
Development and Operations Collaboration



Dev and Ops Fully Shared Responsibilities



DevOps-as-a-Service



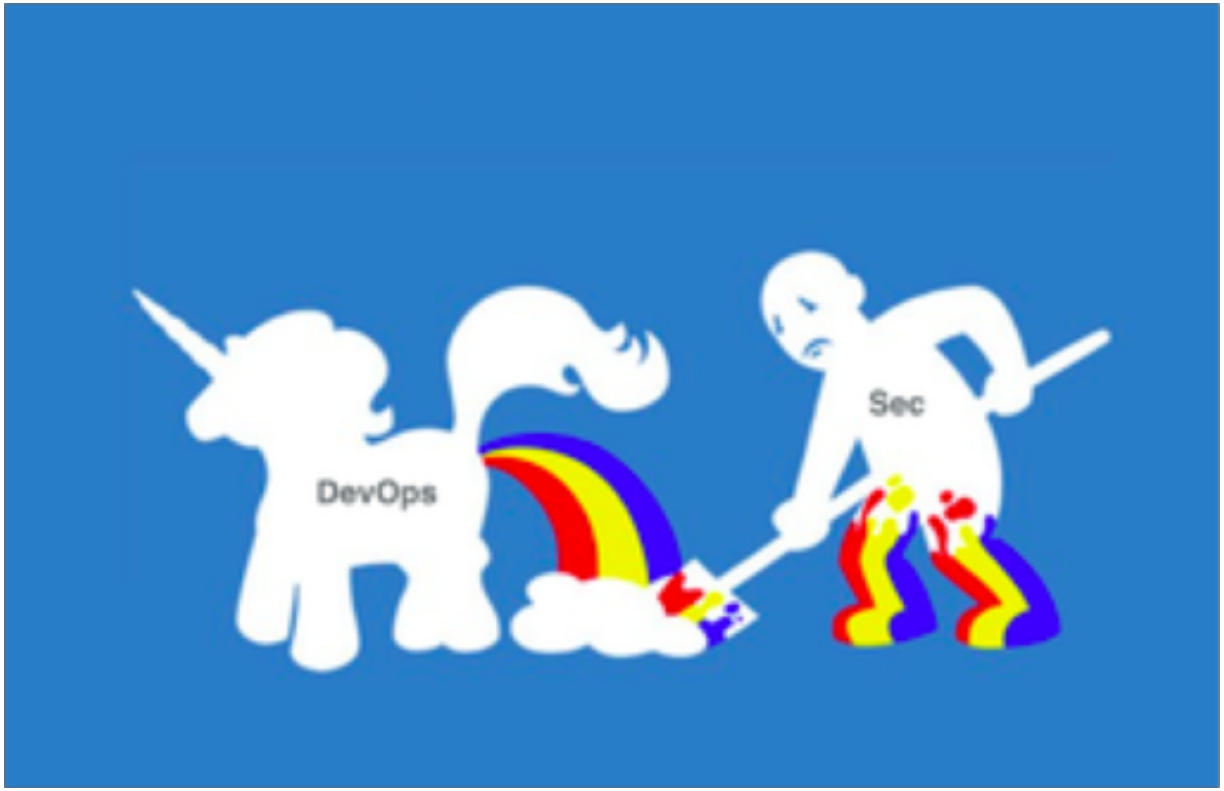
Process

- Collaboration Across Teams
- Cross-Training
- Support Business Agility
- Breaking Down Silos
- Automation of Repetitive Tasks
- “Everything” as Code
- Test, Measure, and Monitor

People

Tools

We want to turn this…



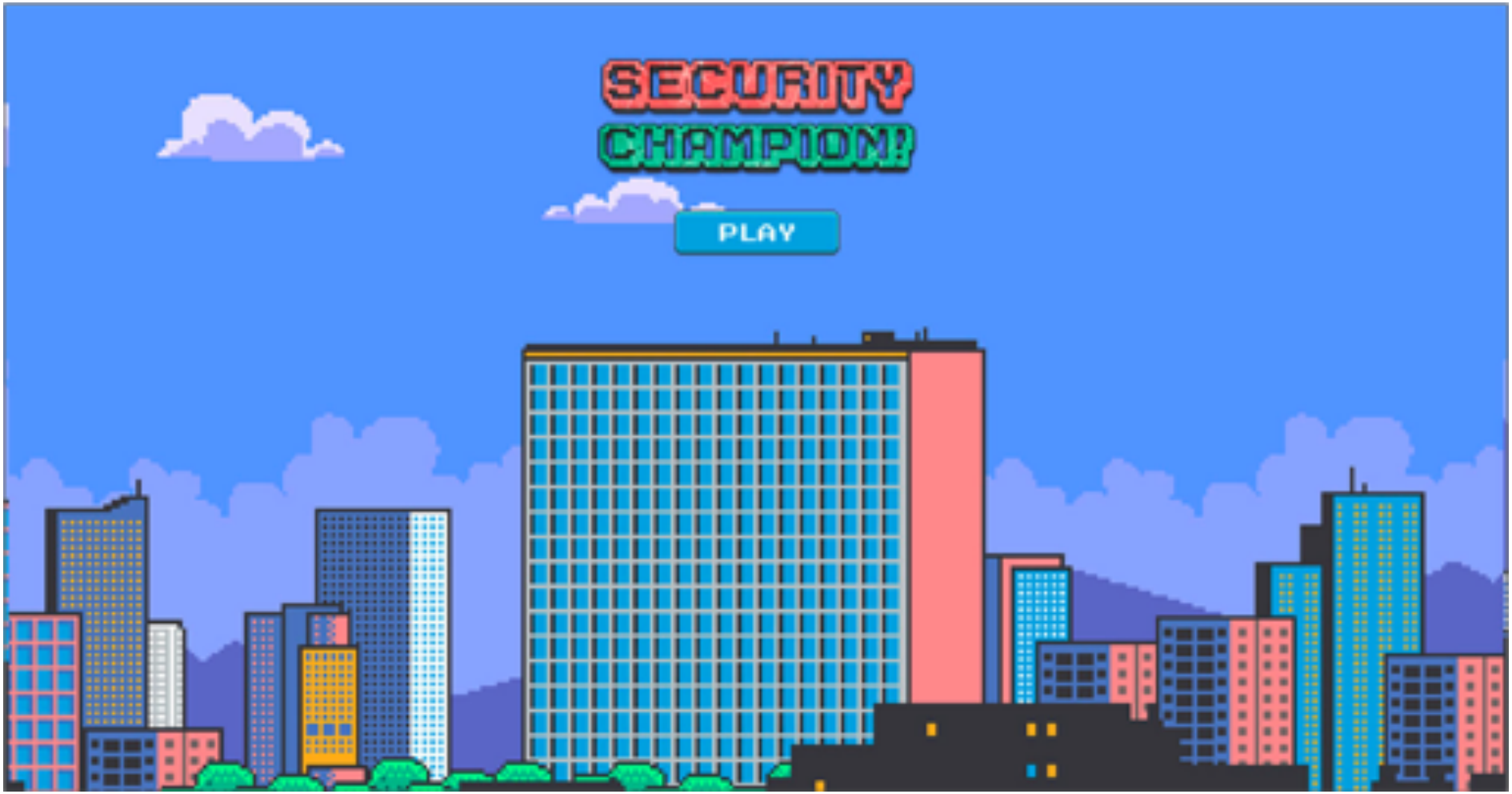
Into this!

“DevSecOps is the process of incorporating and enforcing meaningful security controls without slowing down deployment velocity.”

Enabling DevSecOps Through *People*

Break Down the Silos

Build a Team Beyond Yourself



Be Approachable

Train Others

Radical Transparency

Communication and Collaboration

- Critical piece to the DevOps puzzle
- A culture of trust and empowerment makes for a healthy workplace
- Move towards shipping software faster and more confidently
- Embrace cross-team communication and training
- Feedback available from each step of the pipeline
- Security is a great fit in modern DevOps cultures

Case Study: The ”Two Pizza” Team

Enabling DevSecOps Through *Process*

DevSecOps Pipelines

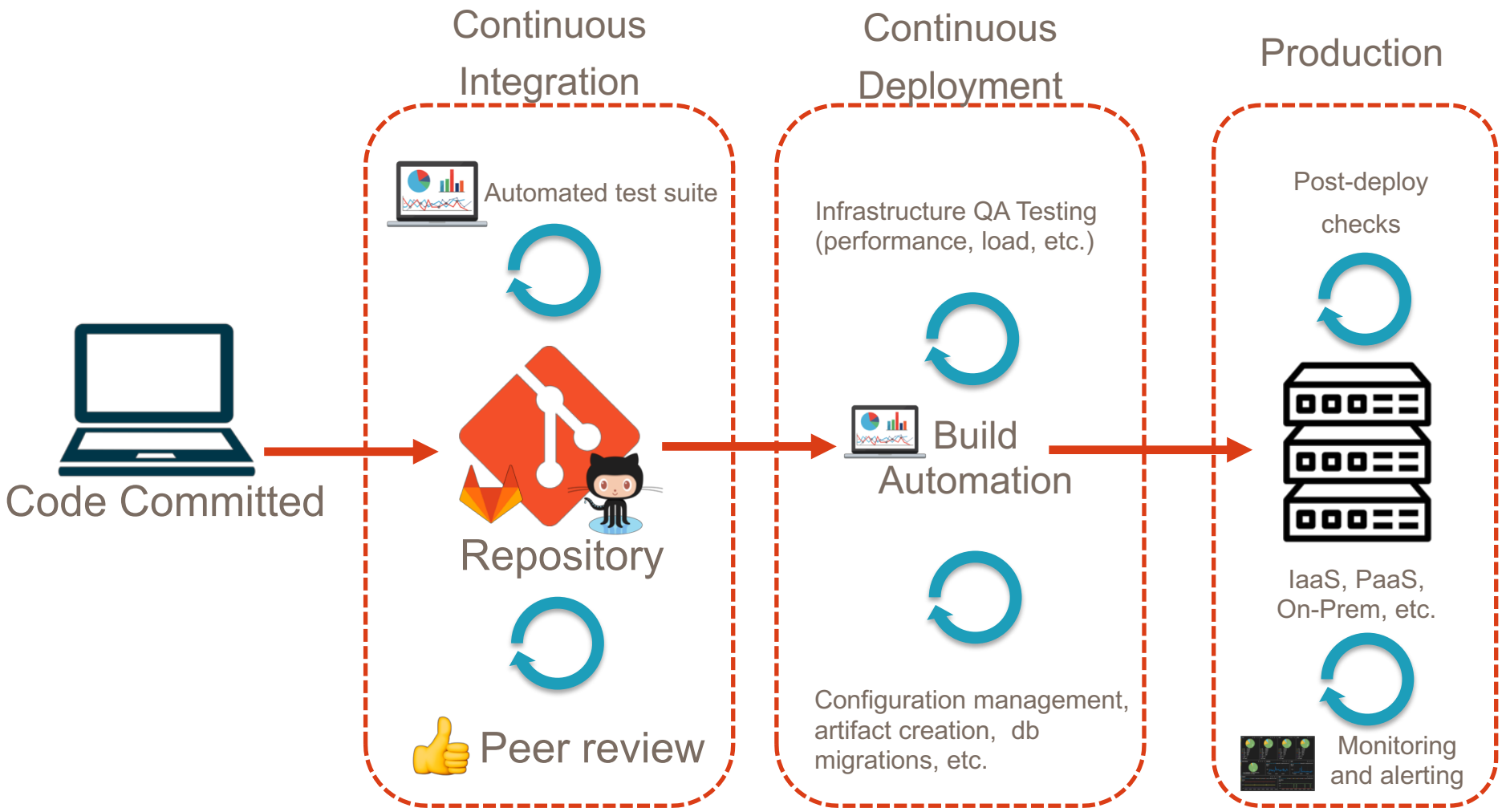
DevOps Processes

- Automate ***building*** the dev and production environment
- Automate software ***testing*** (including security)
- Automate ***deploying*** software and services
- Automate ***monitoring*** and ***alerting***
- ***Tune*** your tools to become more automated and hands-off
- Build the pipeline ***slowly*** and don't fear failure!
- Be careful with ***sensitive areas*** which are difficult to automate (access control, biz logic, complex actions)

Key Goals of DevSecOps Pipelines

- Optimize the critical resource: **Security personnel**
- Automate things that don't require a human brain
- Drive up consistency
- Increase tracking of work status
- Increase flow through the system
- Increase visibility and metrics
- Reduce any dev team friction with application security

Pipeline Security



Development (Pre-Commit)

Code Committed

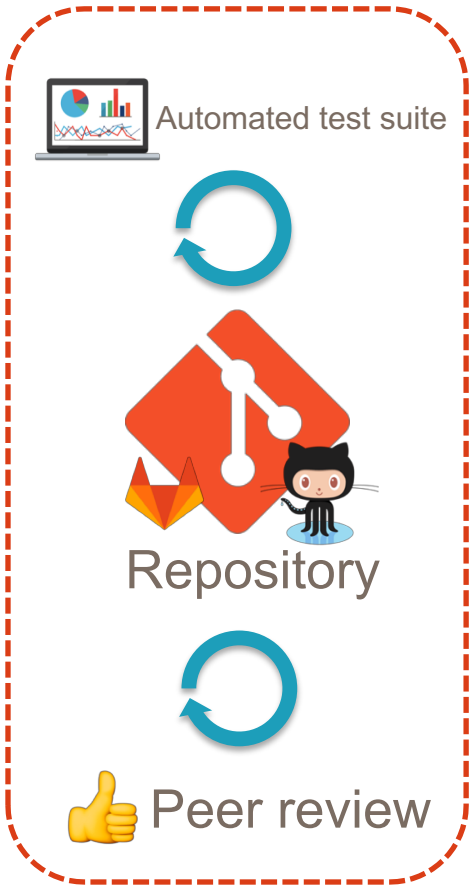
- Developer laptops are the first line of defense in a DevSecOps pipeline
- Moving security to the left prevents costly mistakes and vulnerabilities later
- Required Git pre-commit hooks can offer a simple, effective feedback loop
  - Static analysis scans in the IDE
  - Peer review from security engineers
  - Lightweight, threat modeling in sensitive areas

Git-Secrets

https://github.com/awslabs/git-secrets

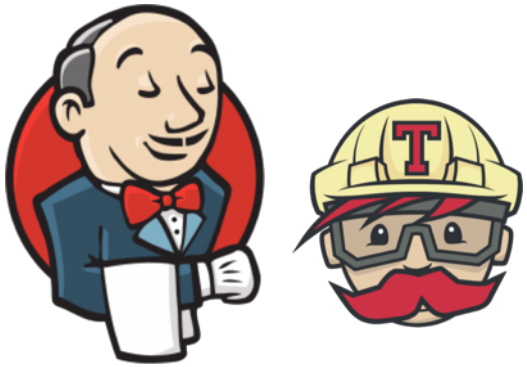
Brakeman Static Scanning (Git Pre-Commit Hook)

Continuous Integration (Commit Stage)



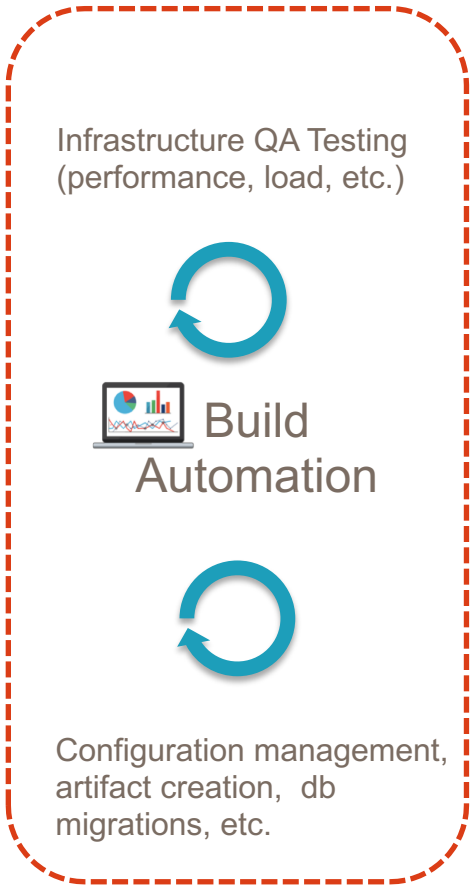
- Basic automated testing is performed after a commit is made
- Must be quick and offer instant feedback
- Key place to include security checks that run in parallel with integration tests, unit tests, etc.
  - Identify risk in third-party components
  - ***Incremental*** static security scanning
  - Alerting on changes to high-risk areas
  - Digital signatures for binaries

Continuous Integration (Commit Stage)



- CI server may include a dedicated security worker
- Third-party dependency checking performed in CI
  - OWASP Dependency Check
  - Node Security Project
  - Bundler-Audit
  - SRC:CLR
- Custom alerts set on repositories and sent to “on-call” security teams
  - Is someone changing pw hashing algorithm?
  - Is a new password policy enabled?

Continuous Deployment (Acceptance)



- Triggered by successful commit and passing build
- Utilize parallel, out-of-band processes for heavyweight security tasks
- IaaS and Config Management should provision latest, known-good environment state (as close to production as possible)
- Security checks during acceptance:
  - Comprehensive fuzzing
  - Dynamic Scanning (DAST)
  - Deep static analysis
  - Manual security testing

Continuous Deployment (Acceptance)

- Zap Baseline scan incorporated into CI stage of the deployment pipeline
- Runs a basic scan scan from a simple Docker run command
- By default will output all results of passive scan rules
- Highly configurable but still struggles in certain areas

https://github.com/zaproxy/community-scripts/tree/master/api/mass-baseline

Production (Post-Deployment)

- After all security checks have passed and deployment is complete
- Security teams job does not stop here:
  - Monitoring and Alerting
  - Runtime Defense (RASP)
  - Red Teaming
  - Bug Bounties
  - External Assessments
  - Web Application Firewalls
  - Vulnerability Management

Enabling DevSecOps Through *Technology*

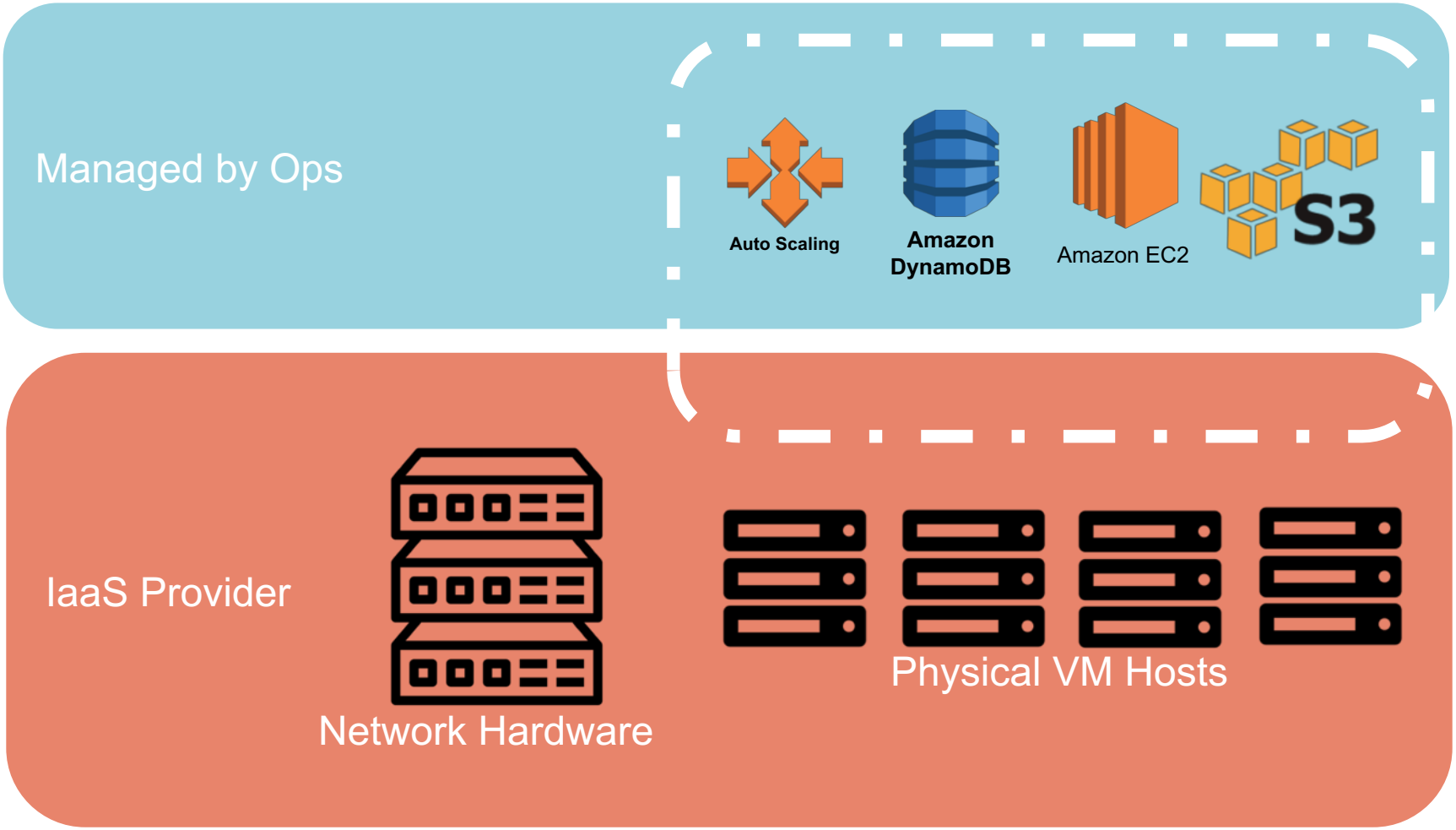
Where are we going?

- Infrastructure as a Service
- Secrets Storage
- Logging and Monitoring
- Containers and Microservices

Infrastructure-as-a-Service (IaaS)

- Delivery of a complete computing foundation
  - Servers (virtualized, physical, or “serverless”)
  - Network
  - Storage
- Infrastructure is exposed to operators using a service
  - Programming network and infrastructure through APIs vs. buying and building physical hardware
- Can be operated by a third-party, hosted in-house (K8s), or a hybrid model

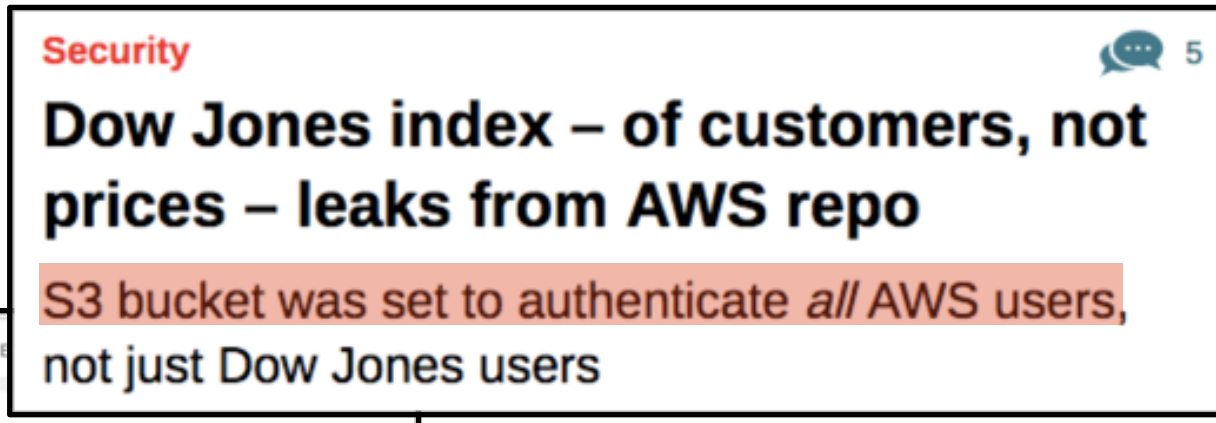
Infrastructure-as-a-Service

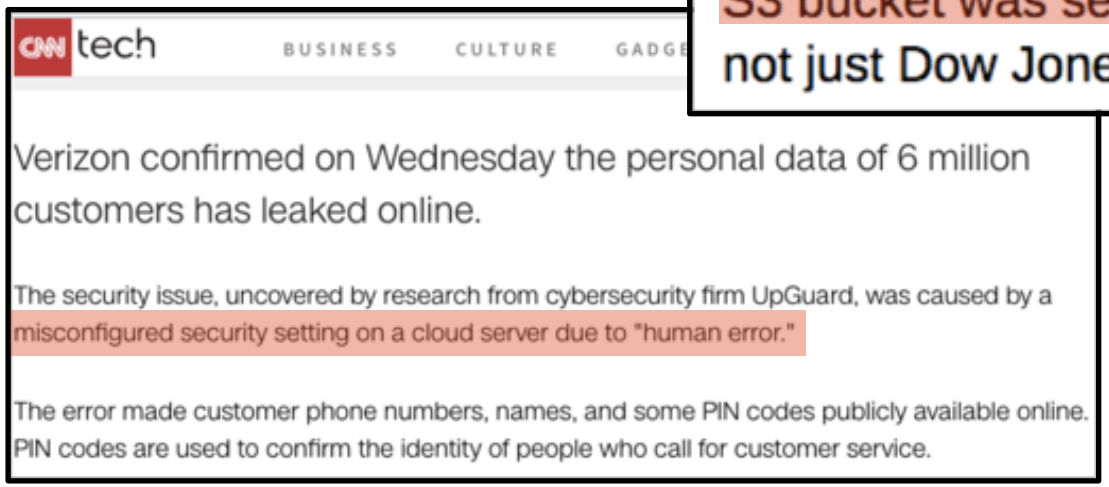


IaaS Security Considerations

- "The Cloud" doesn't *do* security for you – this is your responsibility
- Network and Data Security
- Auditing Capabilities
- Compliance Requirements
  - SOC, PCI, HIPAA, etc.
- Encryption Capabilities
- Third-Party Certificates and Audits
- Secrets Storage, Built-in Security Features, etc.

The Cloud Won't Protect You





Distributing Secrets

- Software systems often need access to a shared credential to operate:
  - Database password
  - Third-Party API key
  - Microservices
- Secret management is full of opinions and could be a course itself
- Many options exist – Choose your own adventure!

Commandments of Sane Secret Management

- Secrets should not be written to disk in cleartext
- Secrets should not be transmitted in cleartext
- Access to secrets should be recorded
- Operator access to secrets should be limited
- Access control to secrets should be granular
- Secrets distribution infrastructure should be mutually authenticated
- Secrets should be version-controlled

HashiCorp's Vault

```
➜ devsecops vault server -dev
==> Vault server configuration:

                 Cgo: disabled
      Cluster Address: https://127.0.0.1:8201
           Listener 1: tcp (addr: "127.0.0.1:8200", cluster address: "127.0.0.1:8201", tls: "disabled")
            Log Level: info
                Mlock: supported: false, enabled: false
     Redirect Address: http://127.0.0.1:8200
              Storage: inmem
              Version: Vault v0.7.3
          Version Sha: 0b20ae0b9b7a748d607082b1add3663a28e31b68
```

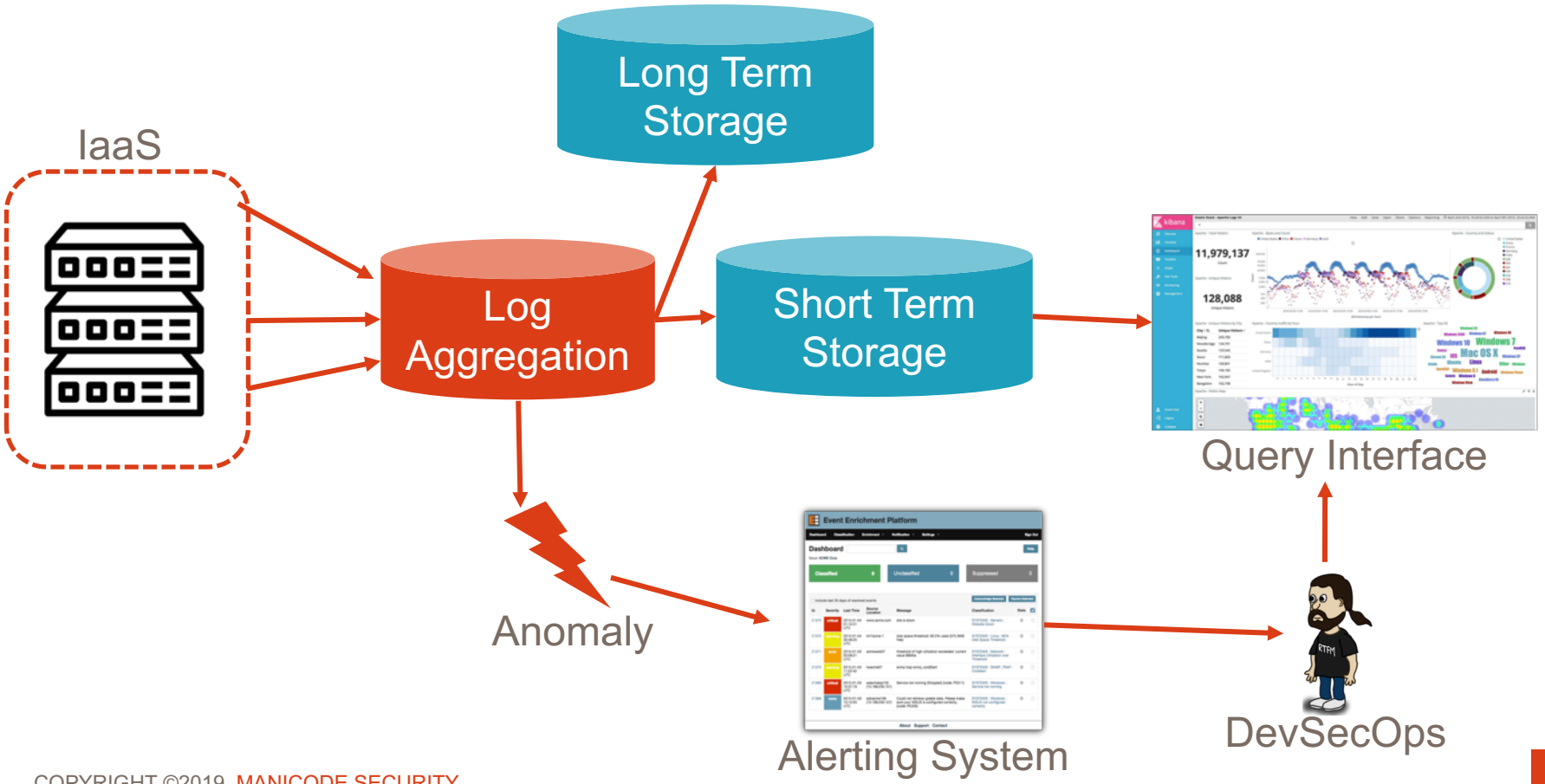
Which is the most secure way to pass secrets to an app running in a container?

1. Pass secrets as an environment variable
2. Mount volume in container that has secrets in a file
3. Build the secrets into the container image
4. Query a ”Secrets API” over your network
5. Other

Logging, Monitoring, and Alerting

- Logs are a part of daily life in the DevOps world
- In security, we focus on particular logs to detect security anomalies and for forensic capabilities
- A basic logging pipeline can be shared between Developers, Operations, and Security teams:
  - **Log Aggregation:** Used to ingest logs from systems, applications, network components, etc.
  - **Long Term Storage:** Filesystem which retains logs for an extended period of time. Good for forensics or breach investigation.
  - **Short Term Storage**: Filesystem or DB which stores logs to be queried quickly and easily.
  - **Alerting:** Anomaly detection system which is responsible for sending alerts to teams when a deviation occurs

Logging and Monitoring Pipeline



Infrastructure as Code

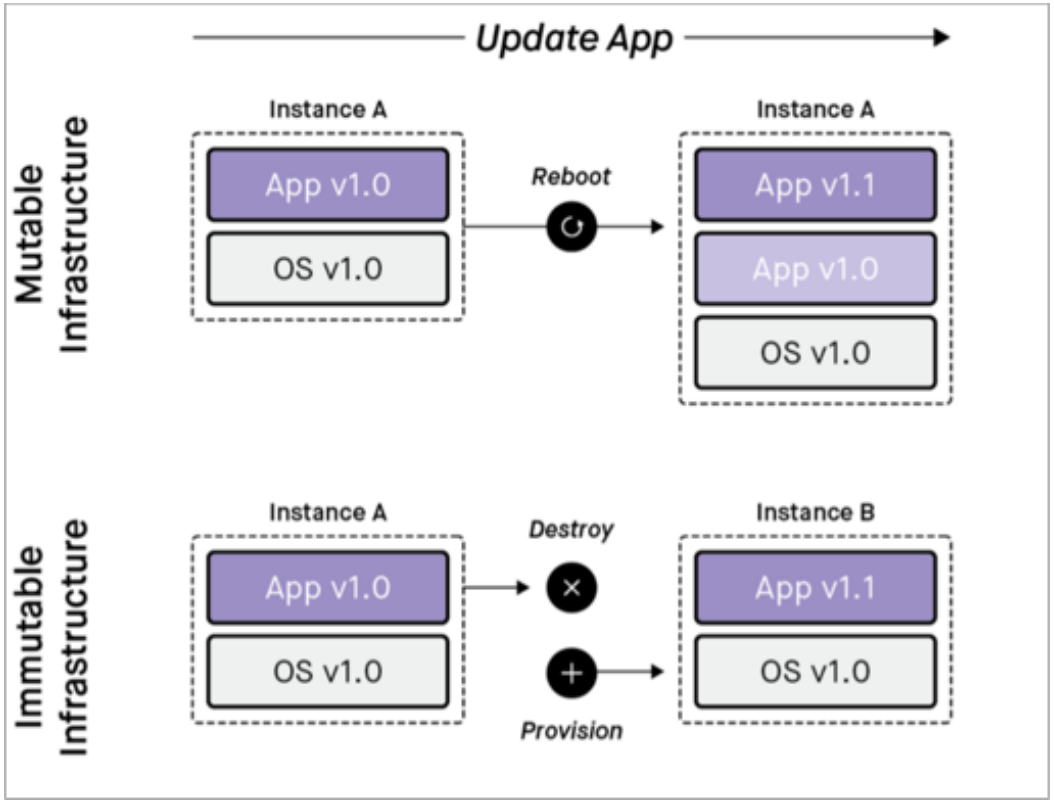
Building Infrastructure

- Is ***your*** infrastructure…
  - Self documenting?
  - Version controlled?
  - Capable of continuous delivery?
  - Integration tested?
  - Immutable?

Remember: ”It’s all software"

Immutable Infrastructure

"Immutable infrastructure is compromised of components which are replaced during deployment rather than being updated in place"

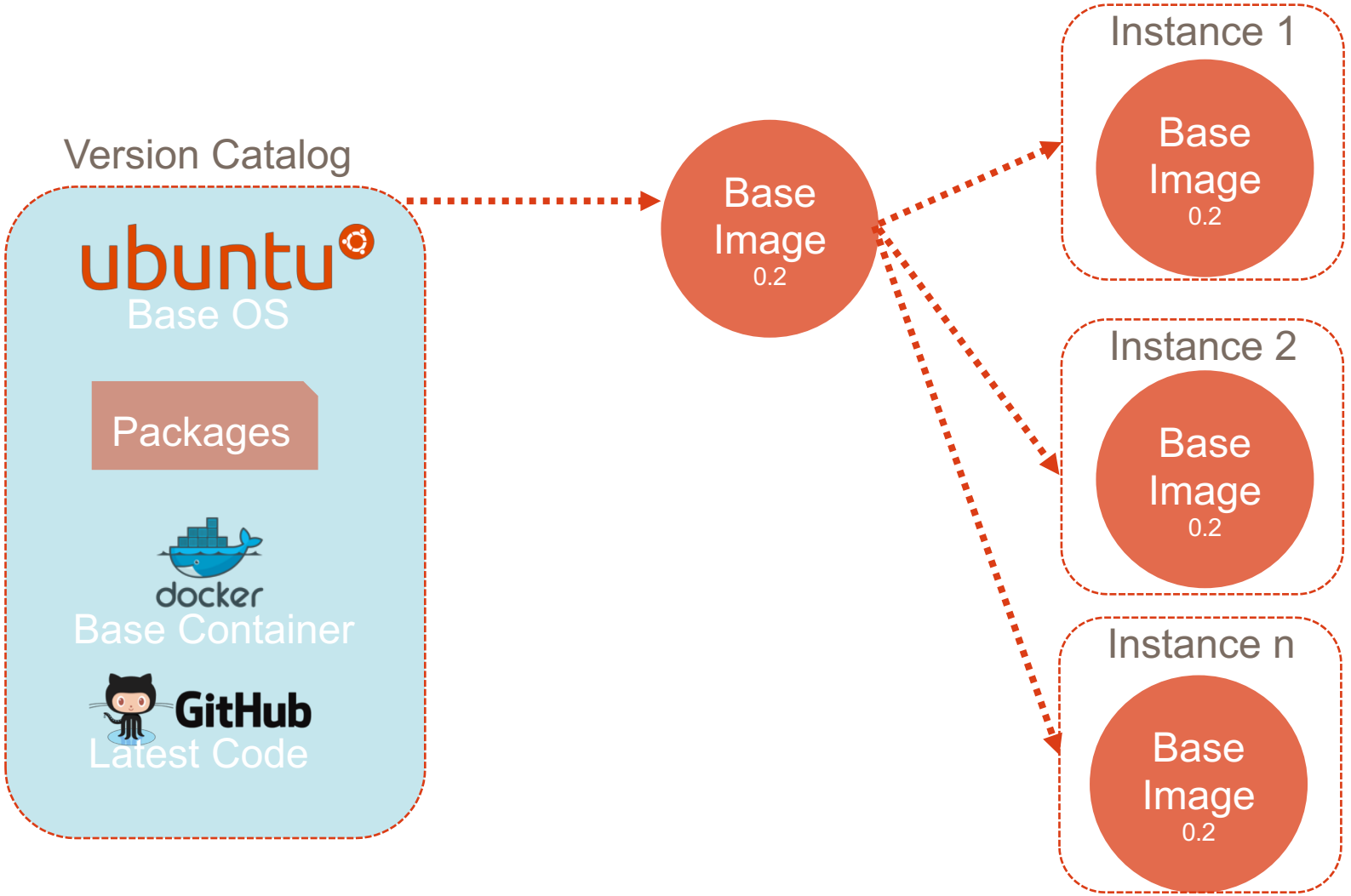


Security and Immutable Infrastructure

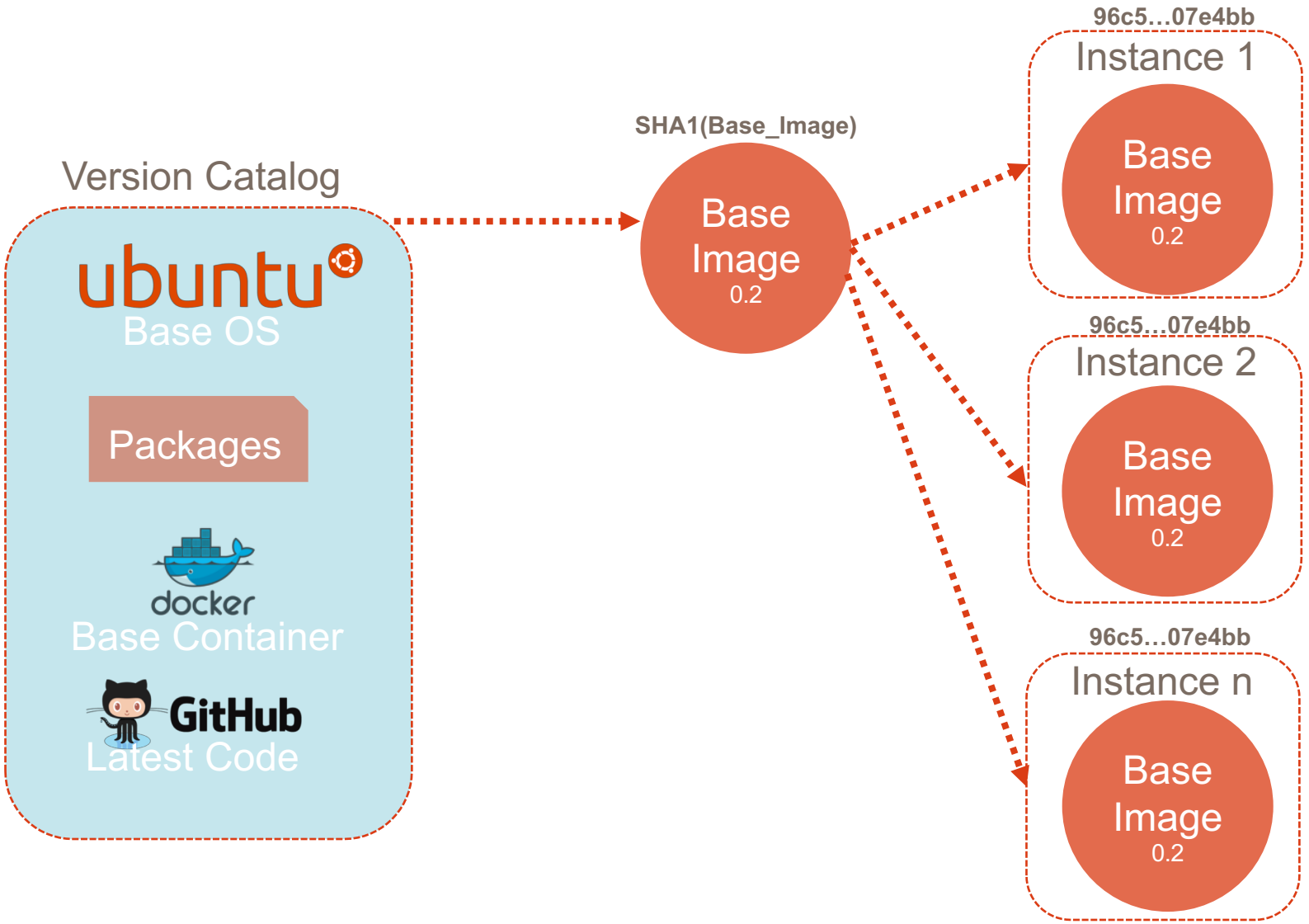
- An immutable infrastructure starts with a "Golden Image" in a version catalog
- Security teams have a central location to validate images as compliant and enforce OS hardening policies
- No more guesswork what is installed Automation can flag security anomalies vs. human intervention
- Tags help teams wrangle infrastructure

"Push Security to the Left"

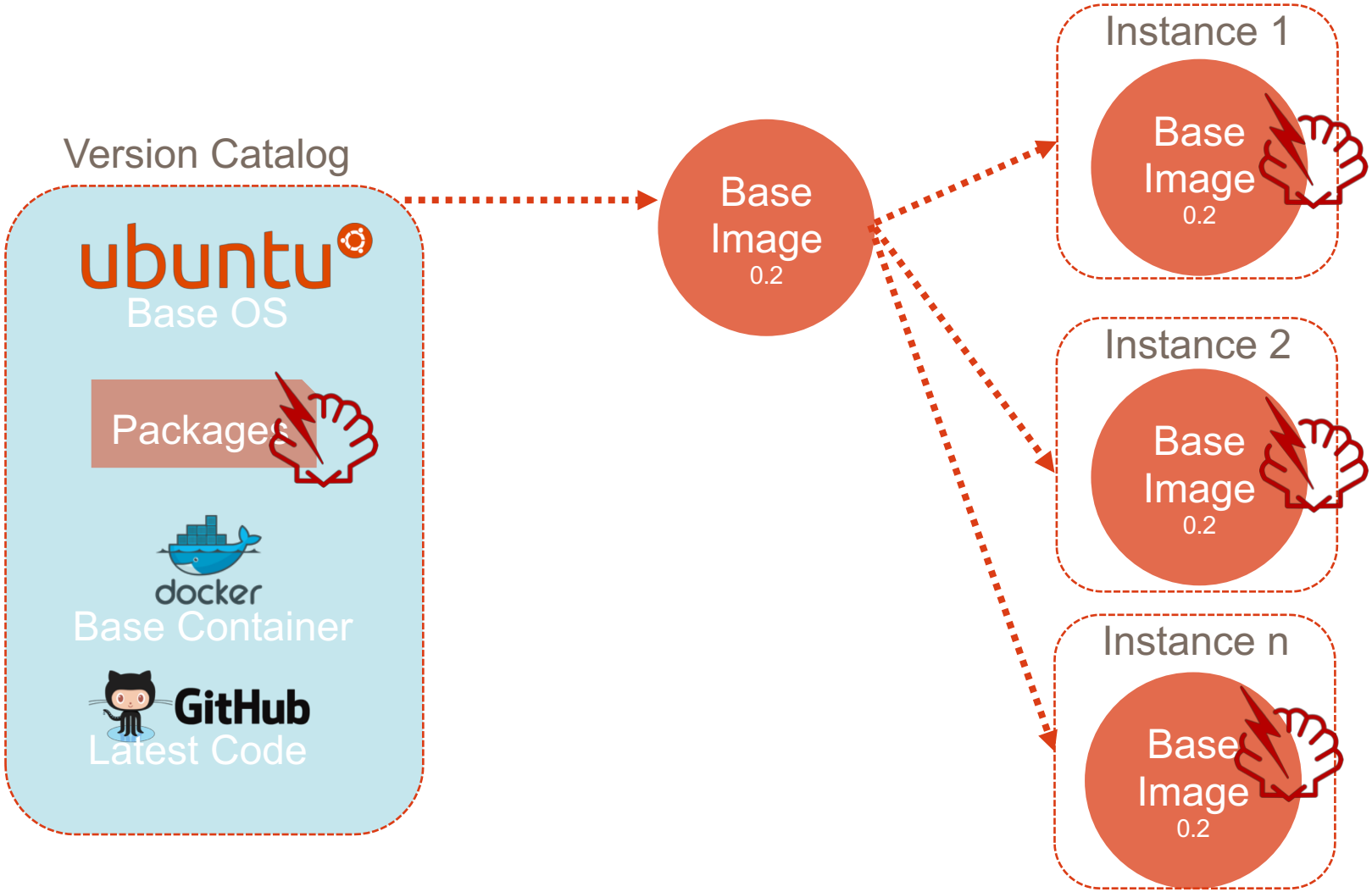
Simple Immutable Infrastructure



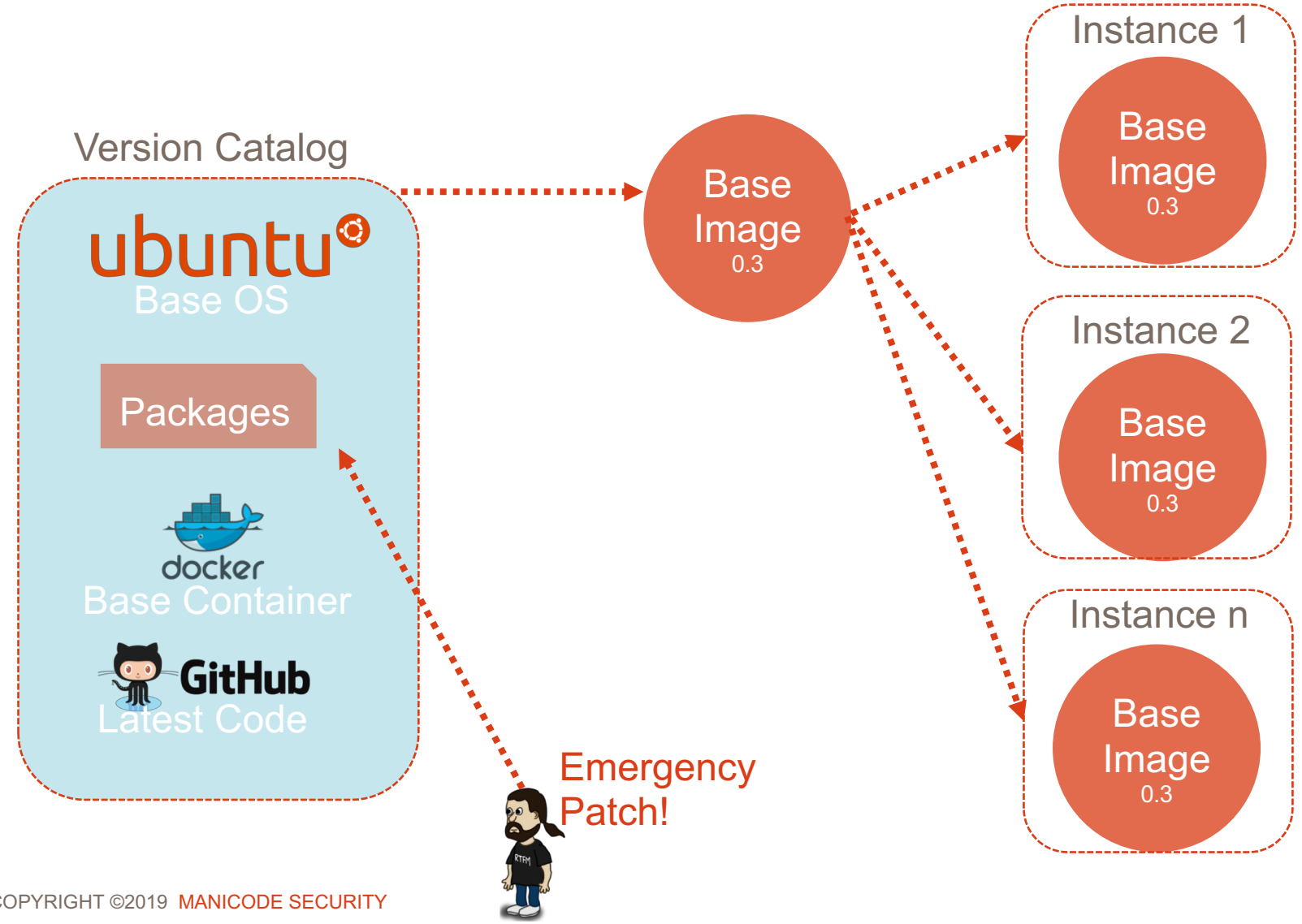
Proving Immutability



Shellshock?



Shellshock?

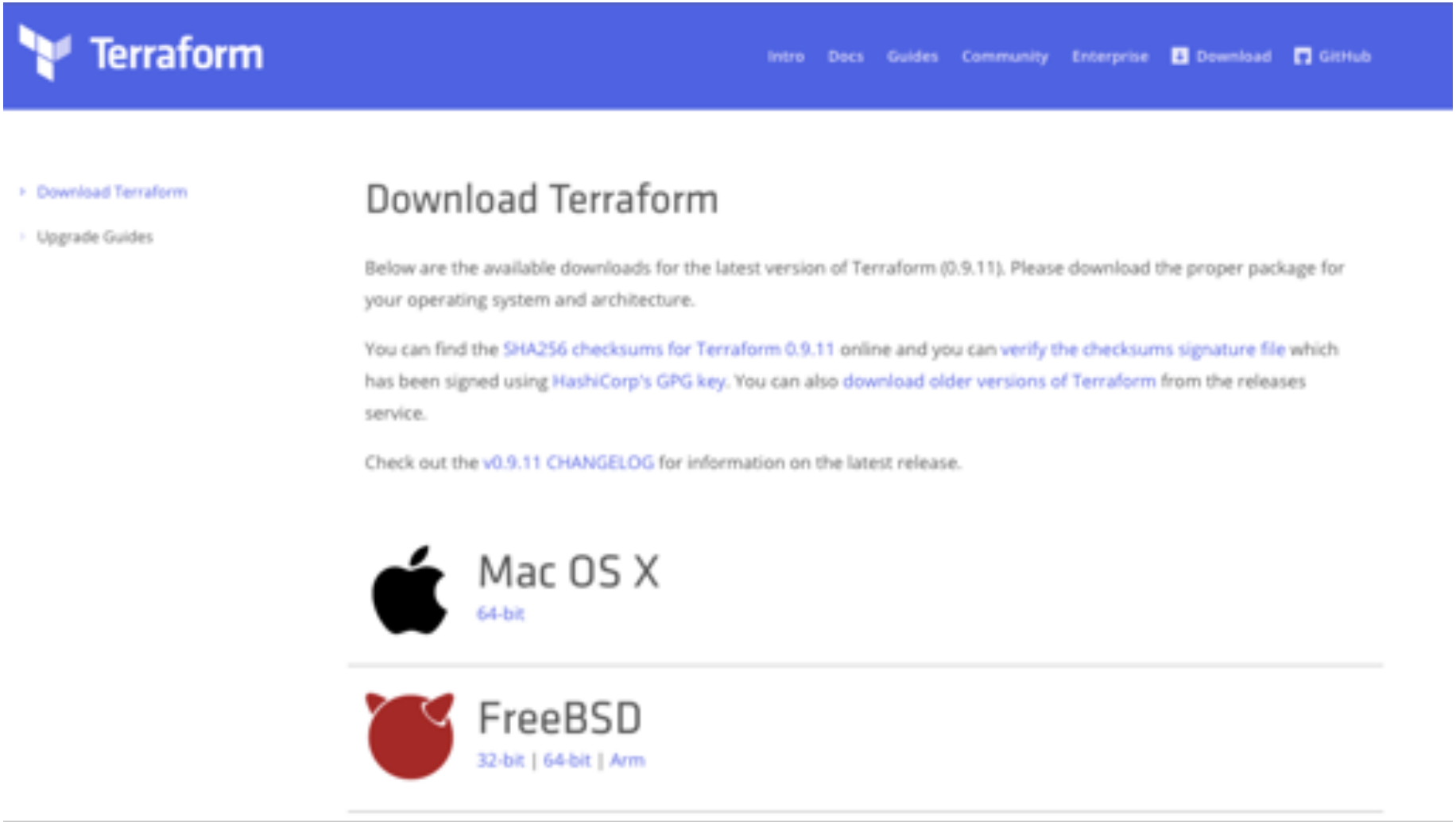


Cattle, not pets.

Security Wins

- Security team now has insight into the entire system
- Infrastructure is auditable and version controlled, just like source code
- Patching can be applied programmatically with a high level of certainty
- Alerting can be built for changes to specific areas of the infrastructure
  - A new firewall rule is created or deleted
  - Administrative user is created
  - New VPC rolled out
- Testing can occur much earlier in the pipeline

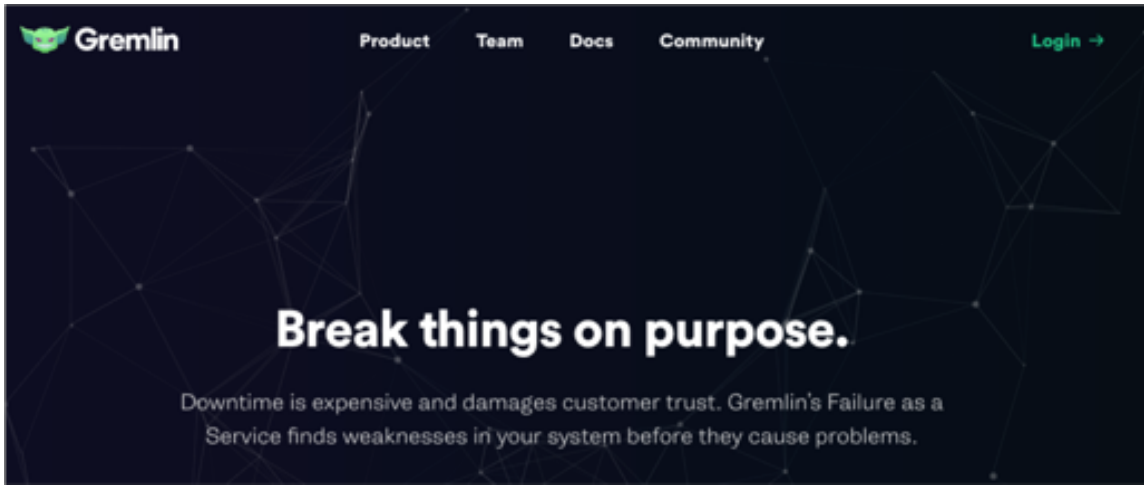
Infrastructure as Code - Terraform



Infrastructure as Code – K8s

```yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: my-site-ingress
  namespace: my-site-prod
  annotations:
    kubernetes.io/tls-acme: "true"
    kubernetes.io/ingress.class: "gce"
    kubernetes.io/ingress.global-static-ip-name: my-site-external-ip
spec:
  tls:
  - hosts:
    - api.my.site
    - my.site
    secretName: my-site-cert
  rules:
  - host: api.my.site
    http:
      paths:
      - path: /*
        backend:
          serviceName: app-api
          servicePort: 80
  - host: my.site
    http:
      paths:
      - path: /*
        backend:
          serviceName: my-site-prod
          servicePort: 80
```

”Chaos” Testing



Brief Introduction to Containers

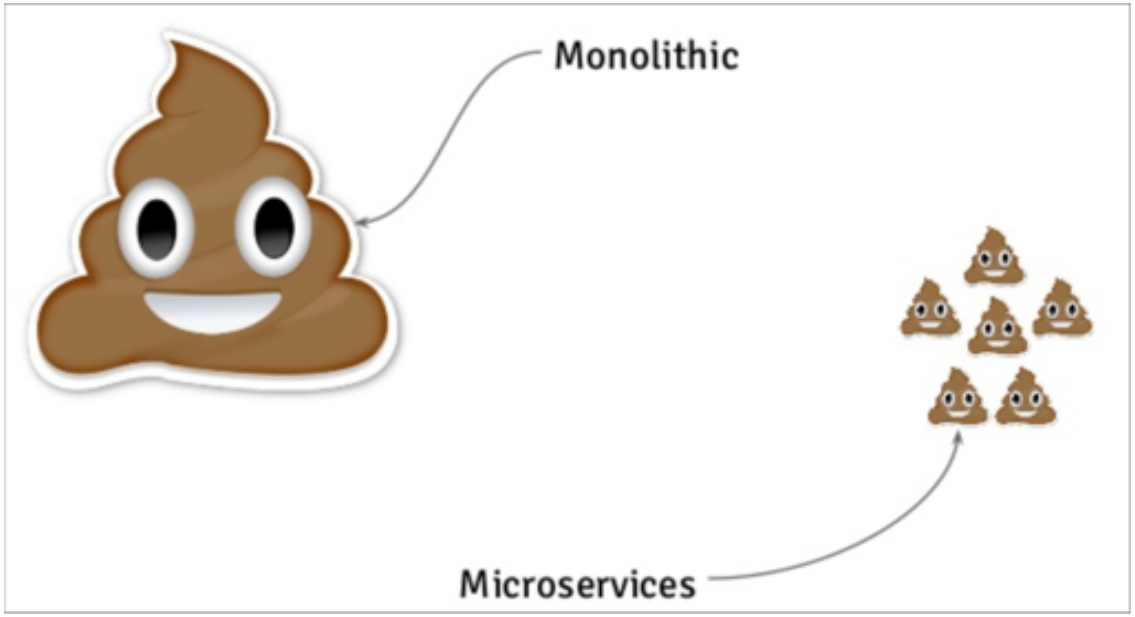
Containers, Containers, Containers, Containers…

Software Deployment is Changing

- Massive shift toward cloud computing
- Increased demand for application and infrastructure portability across environments
- Avoid vendor “lock in” when possible
- Increase in microservices AKA loosely coupled services

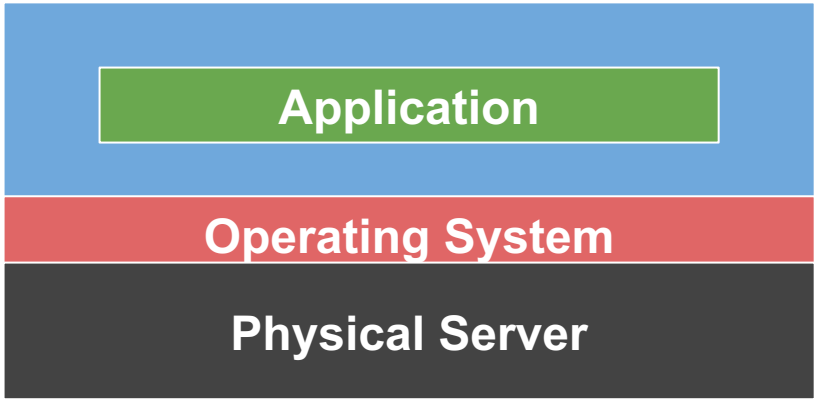
Modern Applications

- Breaking monolithic applications into smaller services offers several advantages:
  - Scale independently
  - Stateless
  - High Availability
  - API-Driven
  - Faster iteration times

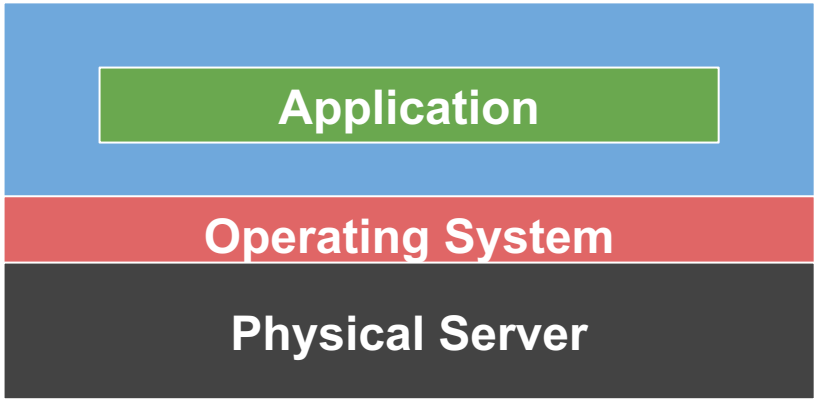


Issues with Modern Applications

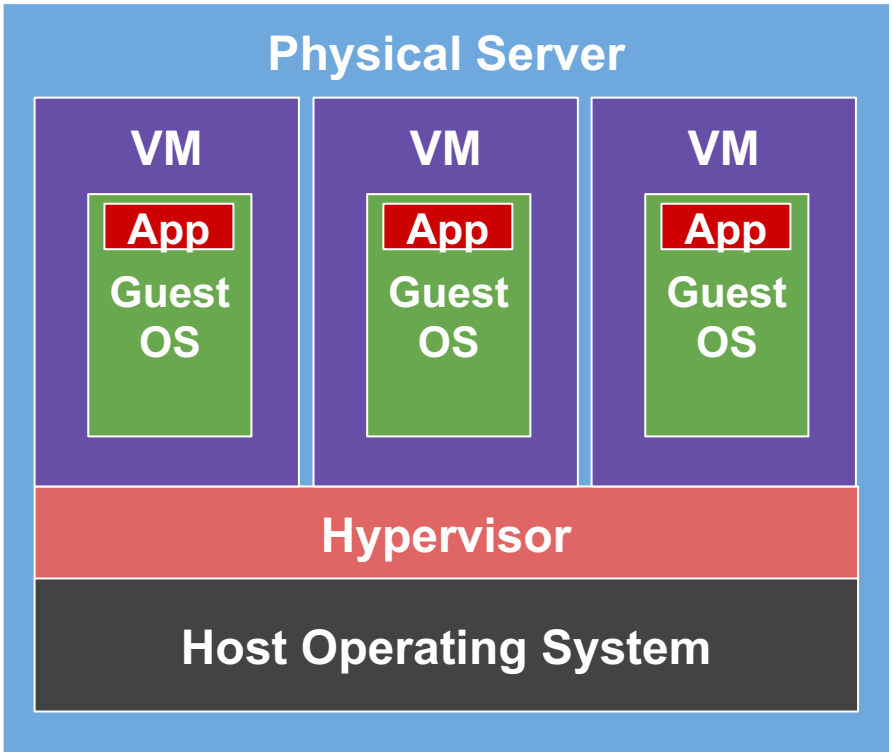
- Organizations often operate in an Ops vs. Dev vs. Sec world
- Applications and microservices are written in a variety of languages and frameworks
- Applications need to run on different technology stacks:
  - Virtual Machines
  - Windows Server
  - Bare Metal Servers
  - Cloud Environments
  - On-Prem Environments
  - Developer Laptops



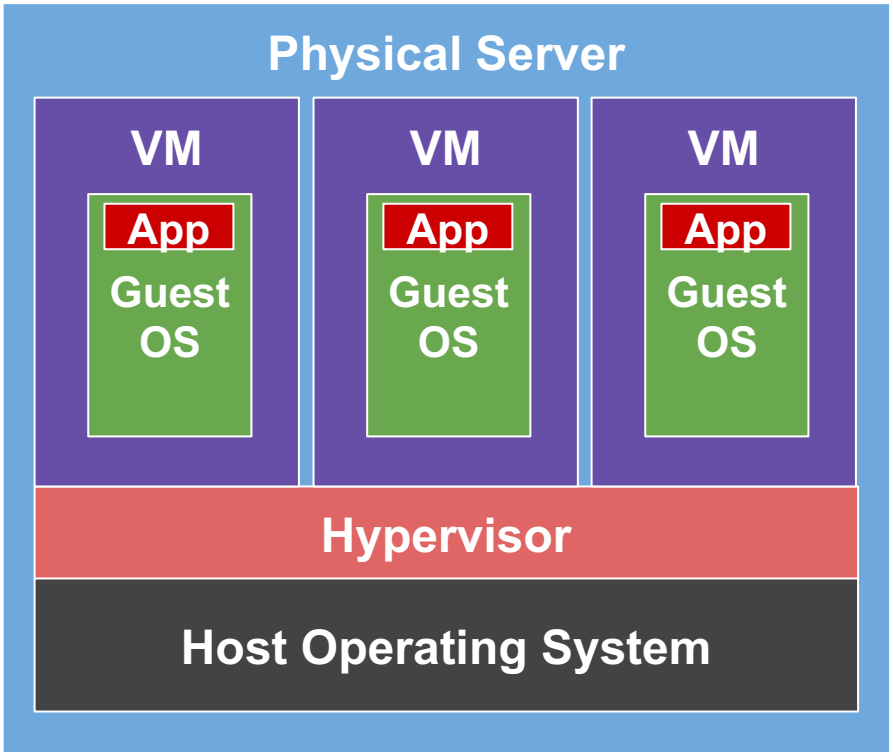
Physical Host



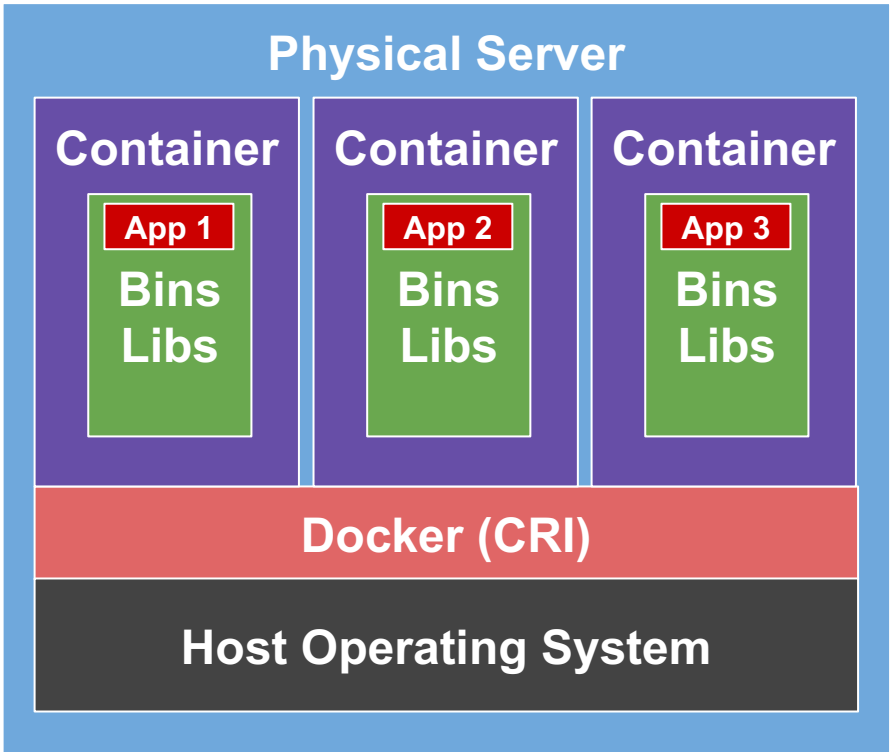
- One application per server
- Slow deployment times
- Low resource utilization
- Scaling challenges
- Migration challenges
- $$$
- Difficult to replicate locally



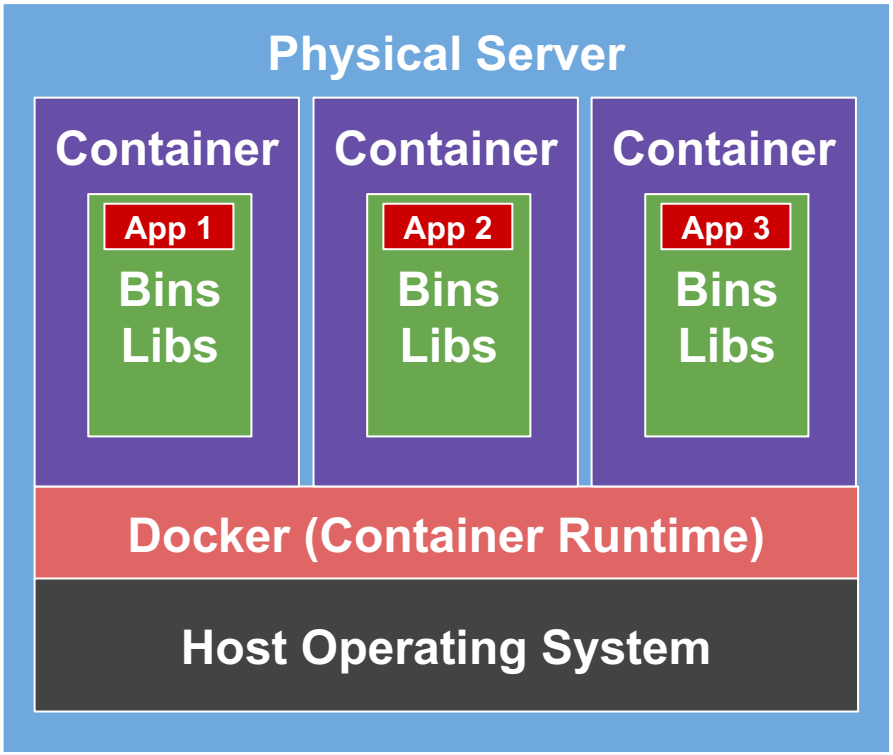
VM



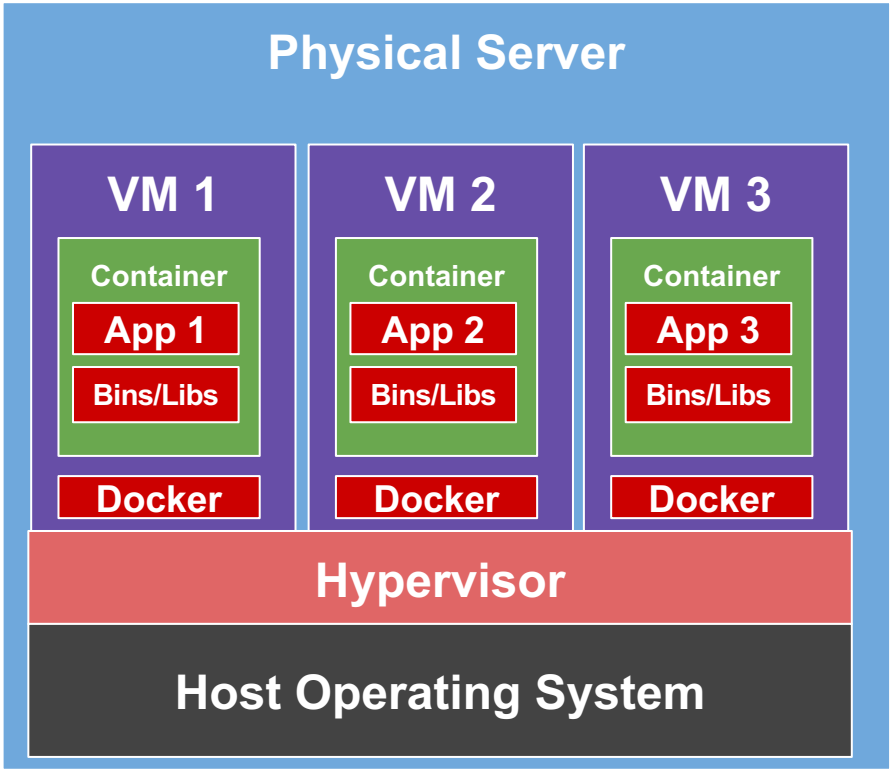
- One physical server and multiple applications
- Each application runs in a Virtual Machine
- Better resource utilization
- Easier to scale
- VMs live in the Cloud
- Still requires complete guest Operating Systems
- Application portability not guaranteed



Container



- Containers are an application layer construct
- VMs allow us to convert one physical machine into many servers
- No Operating System to boot (fast!)
- Most portable out of all options
- Less OS overhead using shared kernel model



Containers and VMs are Happy Together

It's been a pleasure.

jmesta@manicode.com

Jimmy Mesta Secure Coding Instructor *www.manicode.com*