AFS
Agile Full Stack
Developer Bootcamp
Thoughtworks @

Cloud Native

thoughtworks

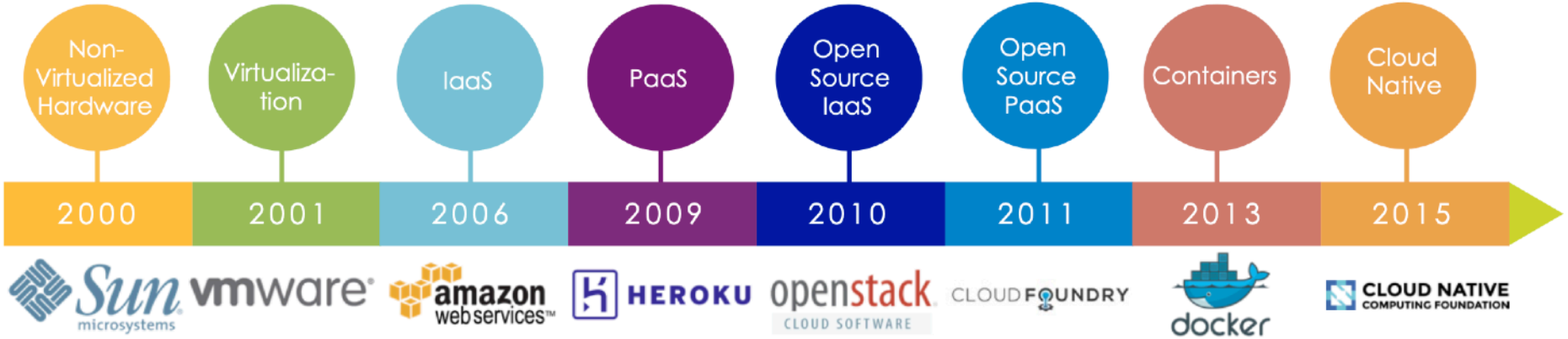
A Brief History of the Cloud

thoughtworks

A Brief History of the Cloud

CLOUD NATIVE COMPUTING FOUNDATION

Cloud Native Computing Foundation (CNCF) serves as the vendor-neutral home for many of the fastest-growing open source projects, including Kubernetes, Prometheus, and Envoy.



What is Cloud computing?

Cloud computing is a model for:

enabling ubiquitous, convenient, on-demand **network access**

to a shared pool of **configurable computing resources** (e.g., networks, servers, storage,

applications and services)

that can be rapidly **provisioned and released**

with **minimal management effort** or service provider interaction.

Cloud computing essential characteristics

on-demand self-service

broad network access

resource pooling

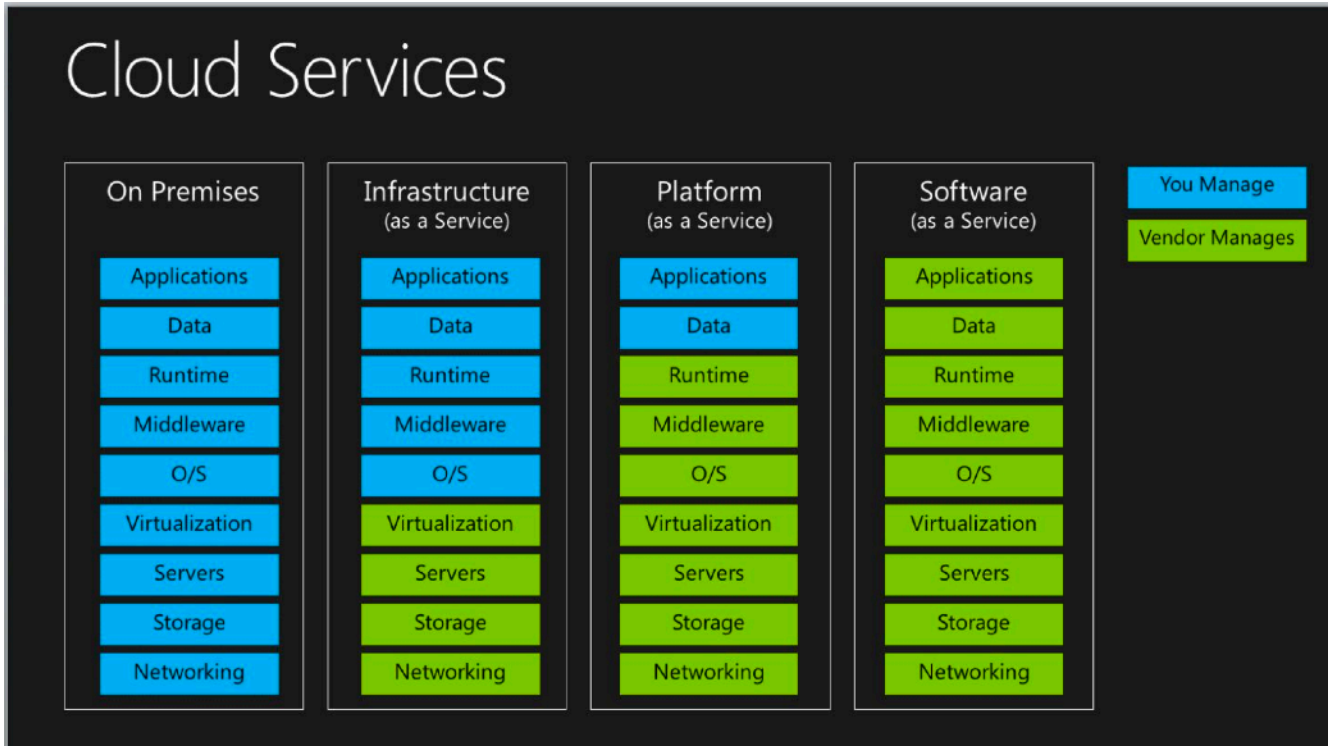
rapid elasticity or expansion

measured service

Cloud computing service models

- Software as a Service (SaaS)
- Platform as a Service (PaaS)
- Infrastructure as a Service (IaaS)

XaaS - the delivery of something as a service



What is Cloud Native?

What is Cloud native?

Cloud native technologies empower organizations to **build and run scalable applications** in modern, dynamic environments such as public, private, and hybrid clouds. Containers, service meshes, microservices, immutable infrastructure, and declarative APIs exemplify this approach.

— CNCF

What is Cloud native?

“Cloud native is an approach to building and running applications that **exploits the advantages of the cloud computing** delivery model.”

Cloud Native Apps vs. Traditional Enterprise Apps

Cloud Native Apps	Traditional Enterprise Apps
Predictable	Unpredictable
OS abstraction	OS dependent
Right-sized capacity	Oversized capacity
Collaborative	Siloed
Continuous delivery	Waterfall development
Independent	Dependent
Automated scalability	Manual scaling
Rapid recovery	Slow recovery

Why Cloud native?

“One of the things we’ve learned is that if you can’t **get it to market more quickly**, there is no doubt that the market will have changed and no matter how well you’ve engineered it or built it or deployed it or trained your folks, it’s not going to be quite right because it’s just a little too late”

James McGlennon

Executive VP and CIO, Liberty Mutual Insurance Group

Why Cloud native?

When companies build and operate applications **using a cloud native architecture**, they bring new ideas to market **faster and respond sooner** to customer demands.

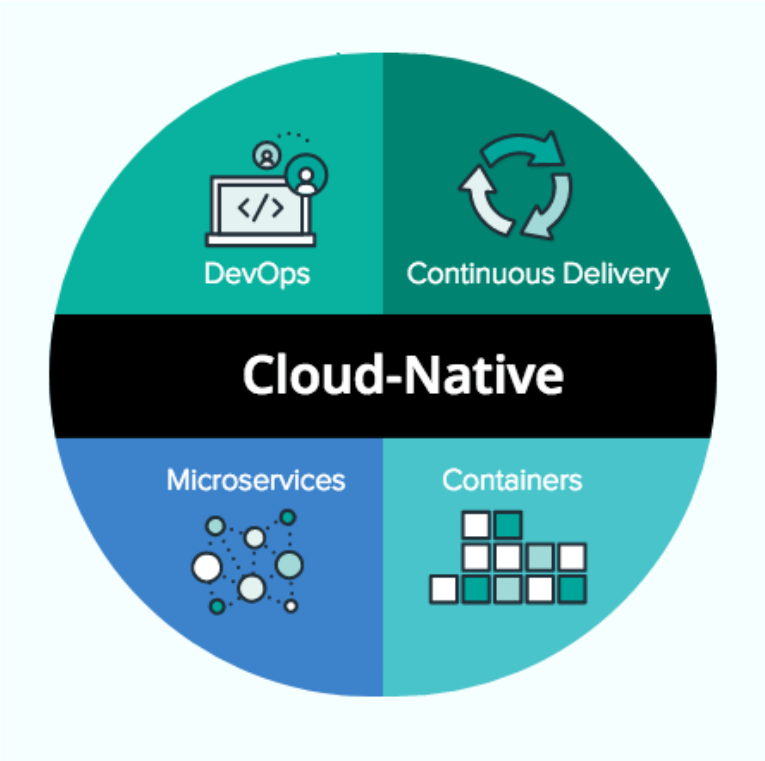
Why do cloud native applications matter?

- Gain a competitive advantage.
- Enable teams to focus on resilience.
- Achieve greater flexibility.
- Align operations with business needs.

What Makes Cloud Native?

thoughtworks

What Makes Cloud Native?



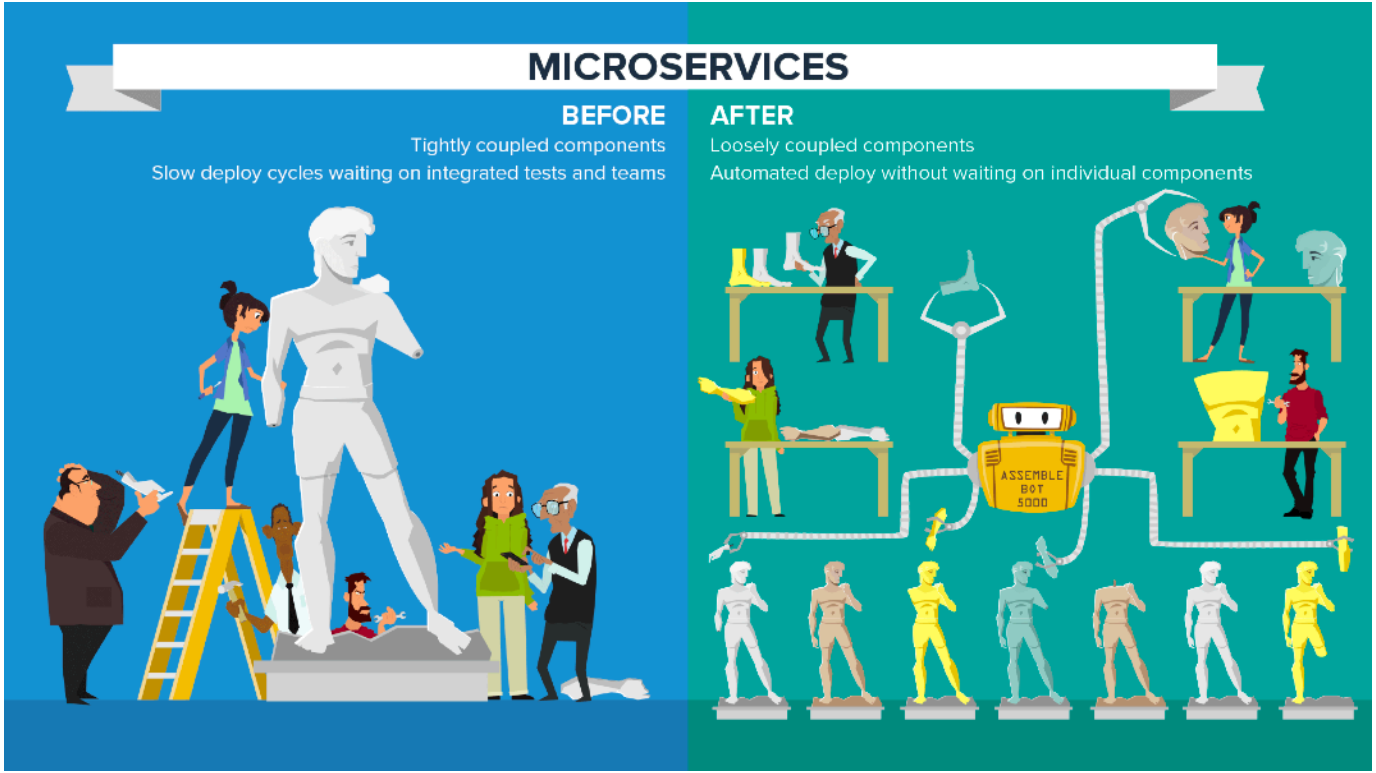
Cloud native development incorporates the concepts of :

- **DevOps**,
- **Continuous delivery**,
- **Microservices**,
- **Containers**.

Microservice

thoughtworks

Microservice



What is Microservice?

Microservices is an architectural approach to developing an application as **a collection of small services**; each service implements business capabilities, runs in its own process, and communicates via HTTP APIs or messaging.

Each microservice **can be deployed, upgraded, scaled, and restarted independent** of other services in the same application, typically as part of an automated system, enabling frequent updates to live applications without impacting customers.

Microservices vs. Traditional Architecture

Microservices Architecture	Traditional Architecture
Has a single focus	Has a wide focus
Is loosely coupled	Is tightly coupled
Is delivered continuously	Relies on scheduled delivery
Has independent teams that own the service lifecycle	Has many teams that own the service lifecycle
Has design patterns and technology emphasizing distributed systems at scale	Has design patterns and technology, putting process first

Why Microservices?

- Teams can be more **responsive** to customer needs.
- There's greater software **team throughput.**
- Successful Microservices architecture improve system **scalability and reliability**.

Guiding principles for qualifying a Microservice

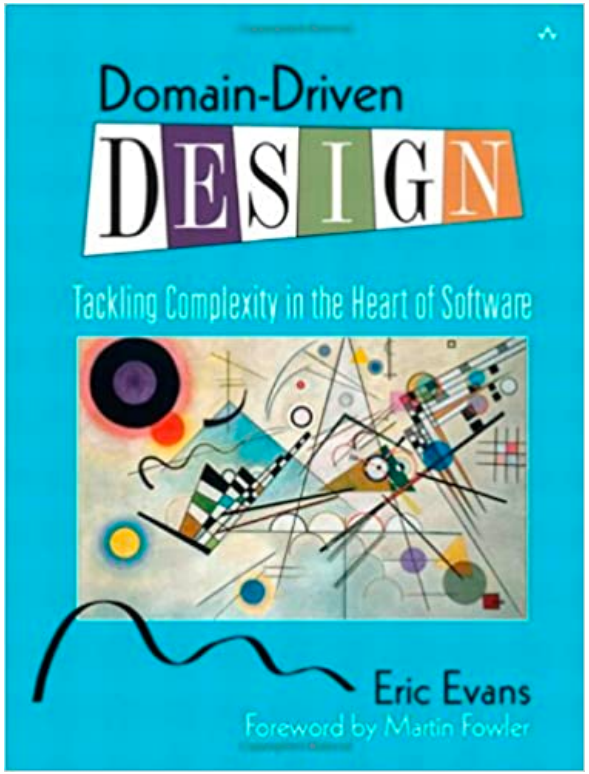
- **Multiple rates of change.**
- **Independent lifecycles.**
- **Independent scalability.**
- **Need for isolated failure.**
- **Need to simplify interactions with eternal dependencies.**
- **Freedom of choice.**

Microservice Trade-Offs

Martin Fowler

Benefits	Costs
Strong module Boundaries	Distribution
Independent Deployment	Eventual Consistency
Technology Diversity	Operational Complexity

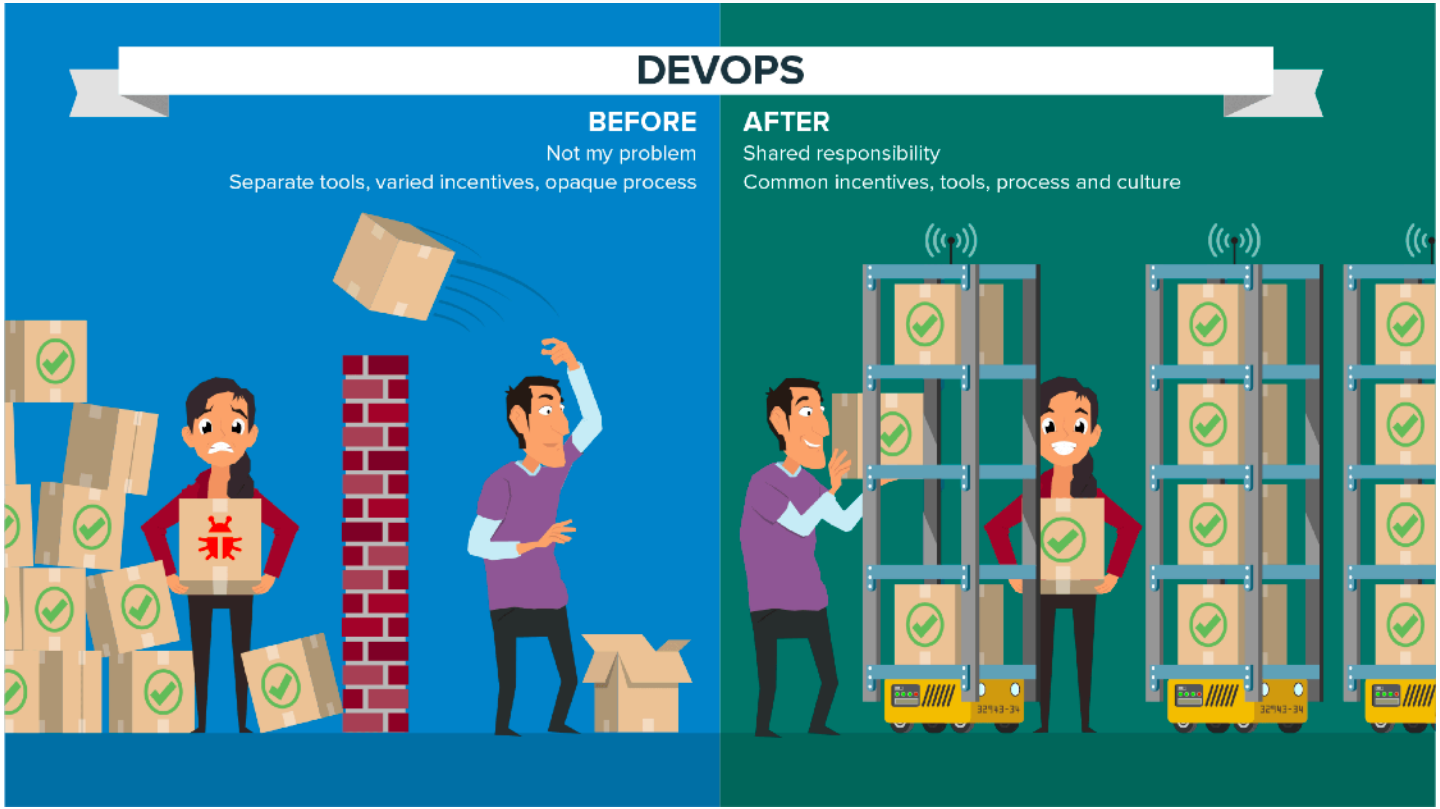
Microservices Books



DevOps

thoughtworks

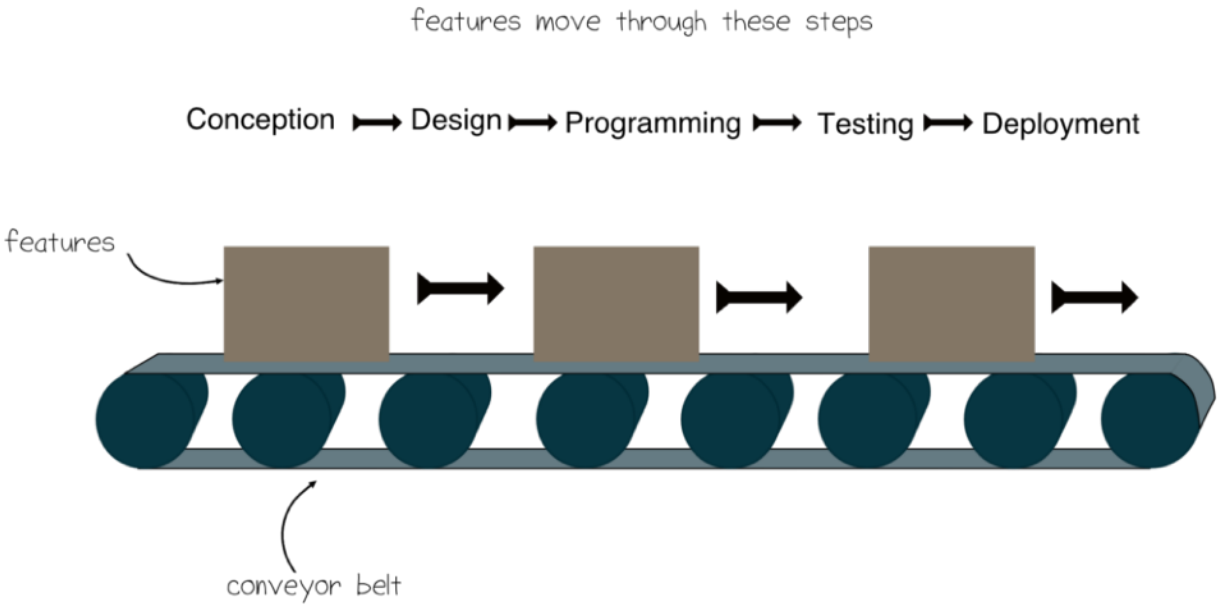
DevOps



Prerequisite Knowledge

Software Development Life Cycle

Software as a factory output ?



Software Development Life Cycle


Release


Operations


Planning


Risk Analysis


Delivery


Testing


Coding

How could we arrange those stages?


Deployment


Build


Requirements

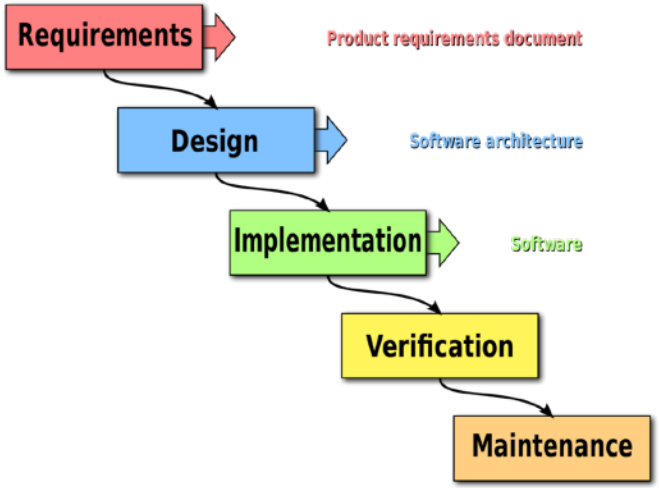

Maintenance


Feedback

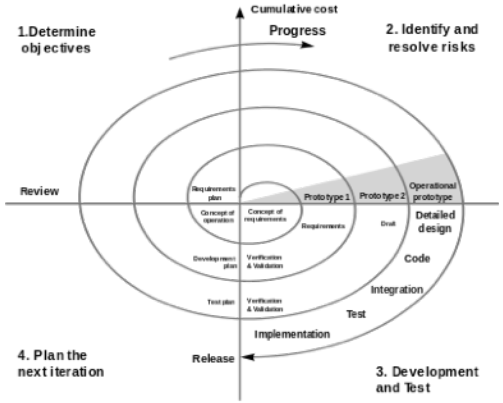

Integration

SLDC Models

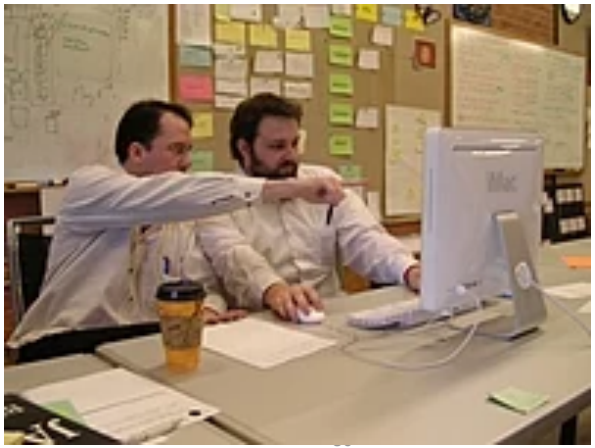
Waterfall



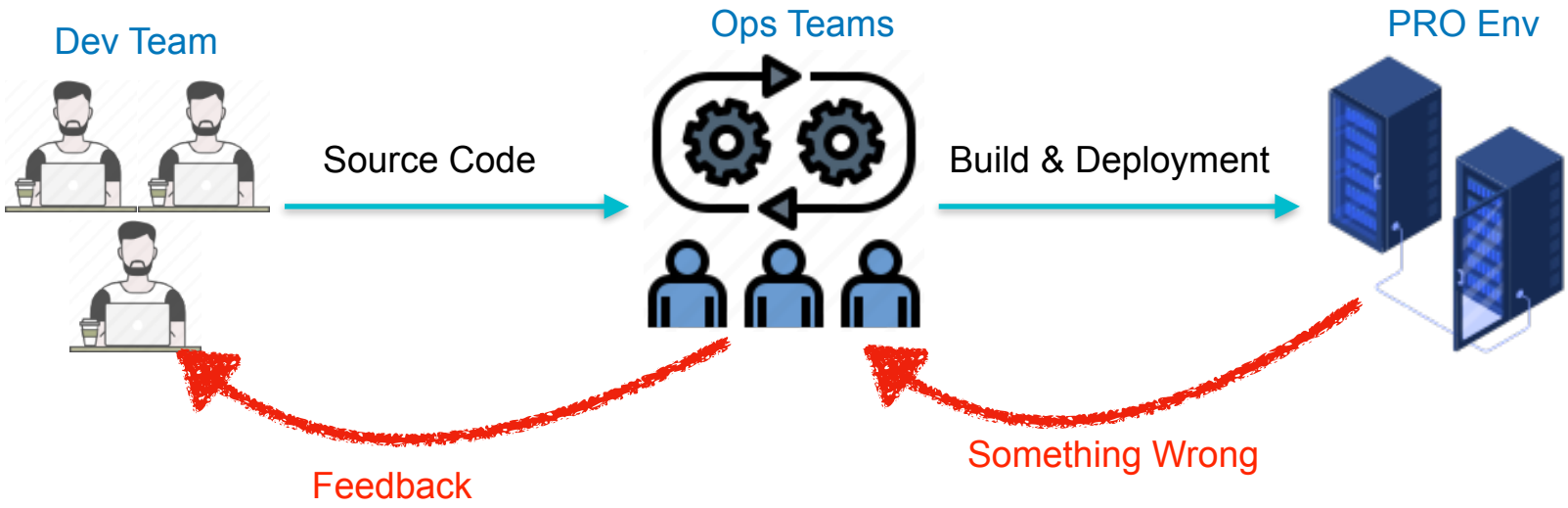
Spiral



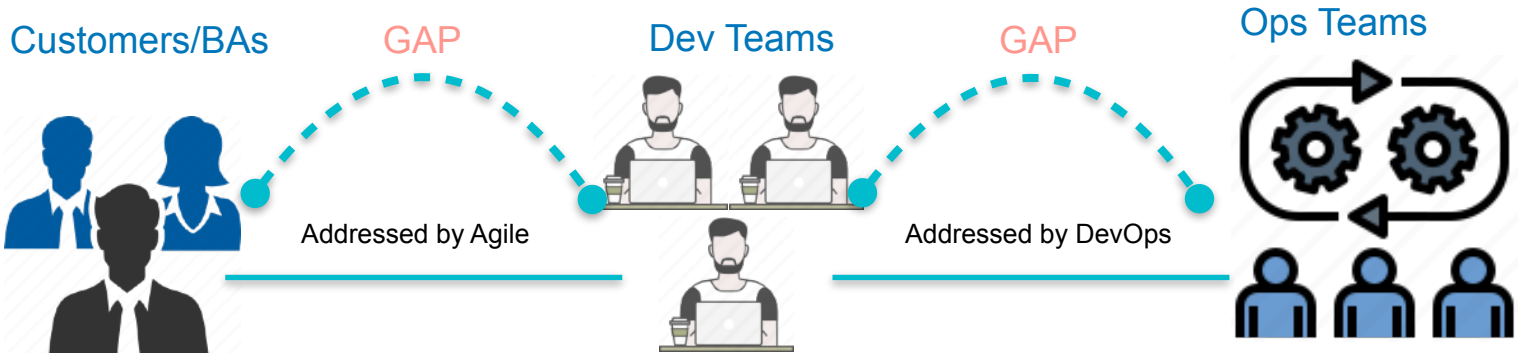
Agile



The disadvantage of SDLC Agile Model



The motivation of DevOps in Agile SLDC



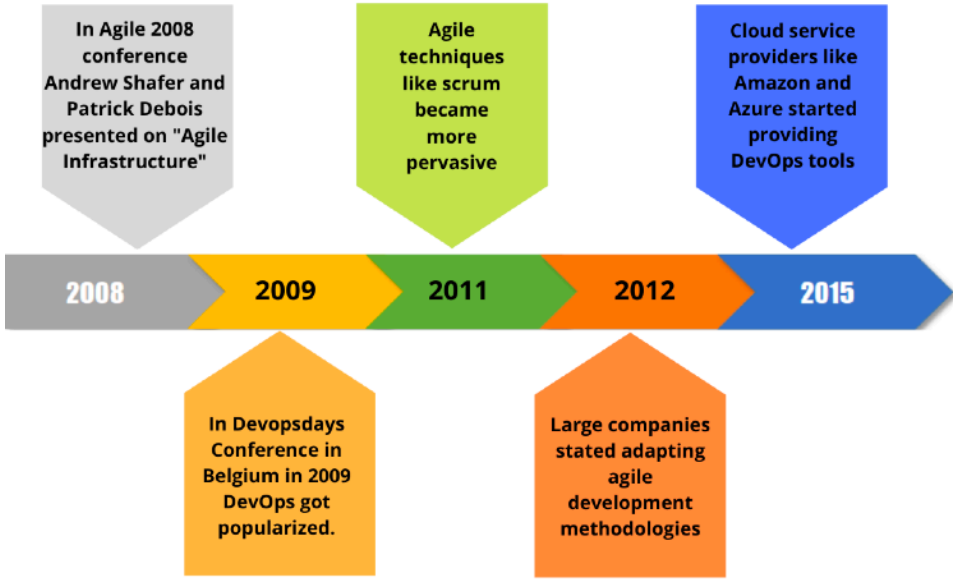
DevOps vs. traditional approaches

DevOps Practices	Traditional Approaches
Collaboration oriented	Silo driven
Structured and automated	Snowflake-driven and mostly manual
Self-service oriented	“IT ticket” oriented
Business focused	Function focused
Designed for change	Change averse

Quick Overview of DevOps

The history of DevOps

The DevOps movement started to coalesce some time between 2007 and 2008, when IT operations and software development communities raised concerns what they felt was a fatal level of dysfunction in the industry.



https://www.cantiktech.com/blog/devops-a-great-career-option/

https://www.jedi.be/presentations/agile-infrastructure-agile-2008.pdf

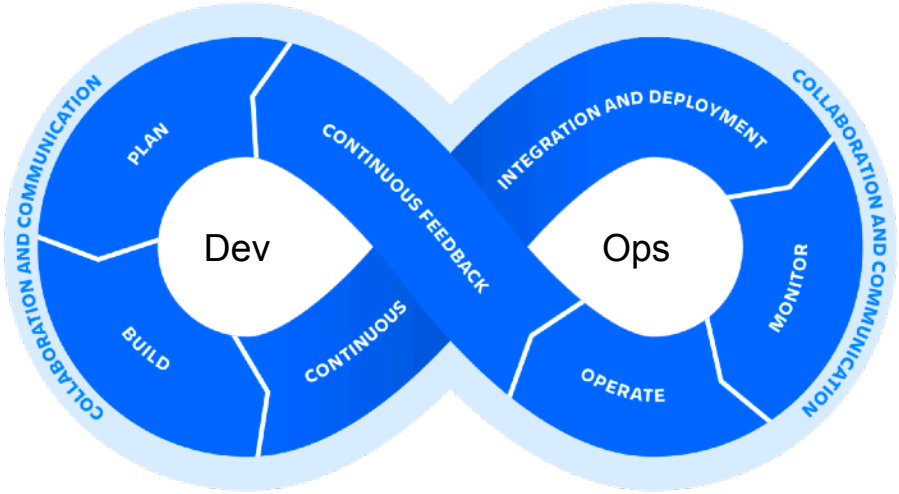
What is DevOps

If you ask 10 people for a definition of DevOps , probably get 10 different answers.

What is DevOps? - Atlassian

DevOps is a set of practices, tools, and a cultural philosophy that automate and integrate the processes between software development and IT teams. It emphasizes team empowerment, cross-team communication and collaboration, and technology automation.

The DevOps lifecycle consists of six phases representing the **processes**, **capabilities**, and **tools** needed for development (on the left side of the loop) and operations (on the right side of the loop). Throughout each phase, teams collaborate and communicate to maintain alignment, velocity, and quality.

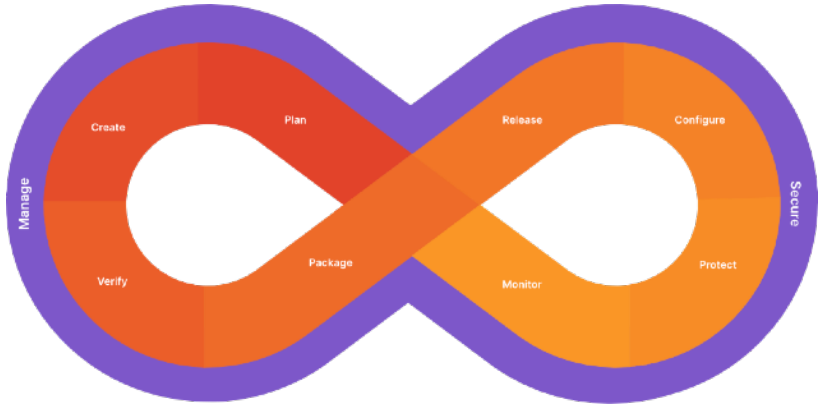


https://www.atlassian.com/devops

What is DevOps? - Gitlab

DevOps is a combination of software developers (dev) and operations (ops). It is defined as a software engineering methodology which aims to integrate the work of software development and software operations teams by facilitating a culture of collaboration and shared responsibility.

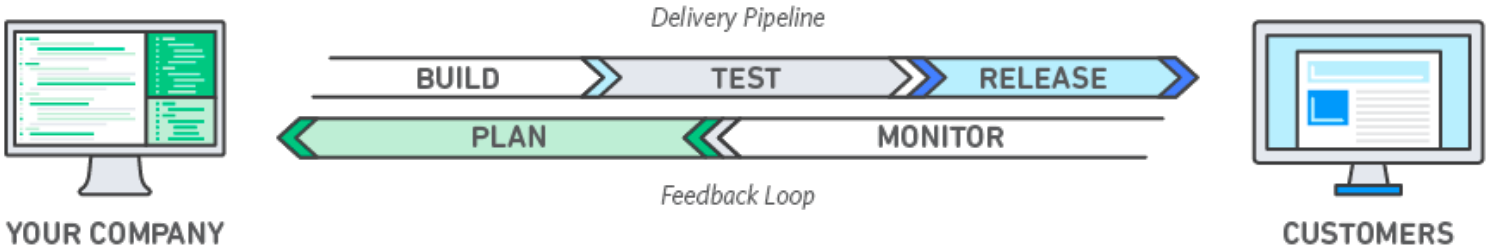
DevOps represents a change in mindset for IT culture. In building on top of Agile, lean practices, and systems theory, DevOps focuses on incremental development and rapid delivery of software. Success relies on the ability to create a culture of accountability, improved collaboration, empathy, and joint responsibility for business outcomes.



https://about.gitlab.com/topics/devops/

What is DevOps? - AWS

DevOps is the combination of cultural philosophies, practices, and tools that increases an organization's ability to deliver applications and services at high velocity: evolving and improving products at a faster pace than organizations using traditional software development and infrastructure management processes. This speed enables organizations to better serve their customers and compete more effectively in the market.

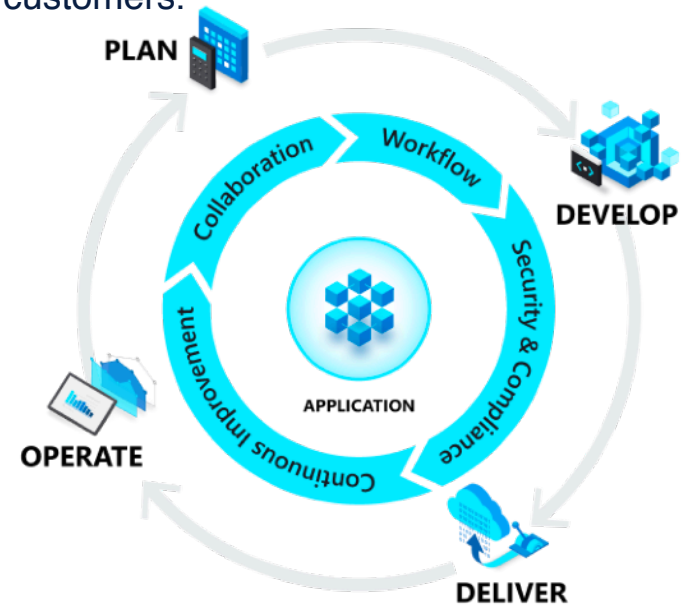


https://aws.amazon.com/devops/what-is-devops/

What is DevOps? - Microsoft Azure

DevOps is a compound of development (Dev) and operations (Ops), DevOps is the union of people, process, and technology to continually provide value to customers.

DevOps influences the application lifecycle throughout its plan, develop, deliver, and operate phases. Each phase relies on the others, and the phases are not role-specific. In a true DevOps culture, each role is involved in each phase to some extent.



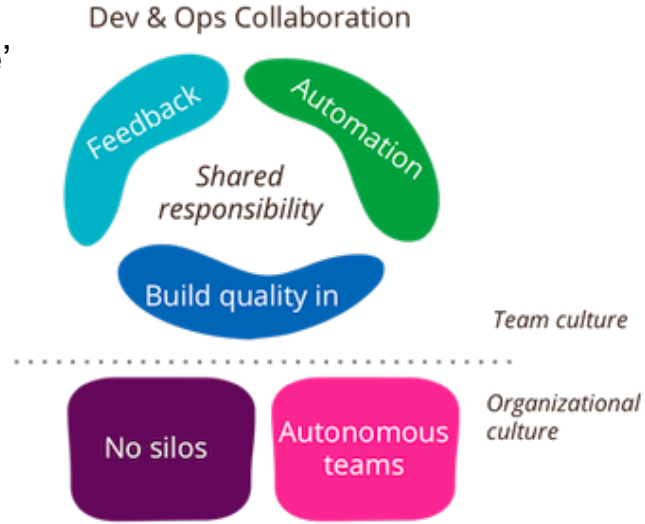
https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-devops/#devops-overview

What is DevOps? - Thoughtworks

The term DevOps is derived from the verbs '**develop**' and '**operate**' — and not, as sometimes mistakenly assumed, from '**developers**' and '**operations**'. The distinction is important: DevOps requires team members to come from a wide variety of backgrounds and have a multitude of skills. This includes developers, QAs, database specialists, network operators.

DevOps teams can handle security, compliance, customer support requests or introducing new features. The key feature of their work is that they have ownership: they own what they build.

https://www.thoughtworks.com/insights/decoder/d/devops



The DevOps movement is aimed at removing these silos and encouraging collaboration between development and operations.

https://martinfowler.com/bliki/DevOpsCulture.html

DevOps tools, concepts and fundamentals

Source Code Management	Teams looking for better ways to manage changes to documents, software, images, large web sites, and other collections of code, configuration, and metadata among disparate teams.
Agile Project & Portfolio Management	Teams looking for a better way of initiating, planning, executing, controlling, and closing the work of a team to achieve specific goals and meet specific success criteria at the specified time.
Continuous Integration (CI)	Teams looking for ways to automate the build and testing processes to consistently integrate code and continuously test to minimise the manual efforts spent in frequent runs of unit and integration tests.
Continuous Delivery (CD)	Teams looking for ways to automate the build, test and packaging, configuration and deployment of applications to a target environment.
Shift Left Security	Teams looking for ways to identify vulnerabilities during development with actionable information to empower dev to remediate vulnerabilities earlier in the lifecycle have specific goals and meet specific success criteria at the specified time.
Monitoring and Feedback	Teams looking for ways to embed monitoring into every deployed version and the impact of application changes to the business value and user experience.
Rapid Innovation	Teams looking for ways to provide feedback back into the development, test, packaging & deployment stages to complete the loop to integrate dev and ops teams and provide real time feedback from production environments and customers.

https://about.gitlab.com/topics/devops/

DevOps tools

Even with the best tools, DevOps is just another buzzword if you don't have the right culture!

What are the benefits of DevOps

When development and operations teams come together, they reduce lead time, deploy more frequently, and produce higher-quality software.

- **Collaboration and trust**
- **Faster time to market (BizDevOps)**
- **Lower risk and smoother deployments, at scale**
- **Faster recoveries**
- **Higher customer satisfaction and better product-market fit (BizDevOps)**
- **Built-in security (DevSecOps)**

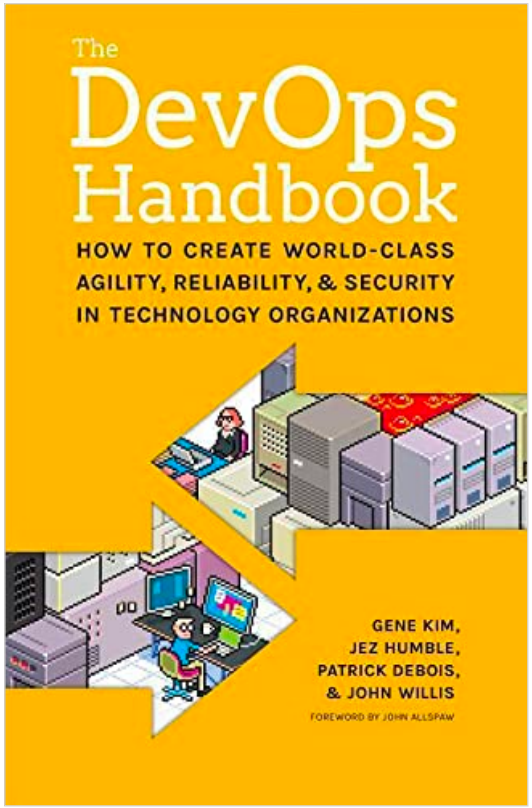
State Of DevOps Report

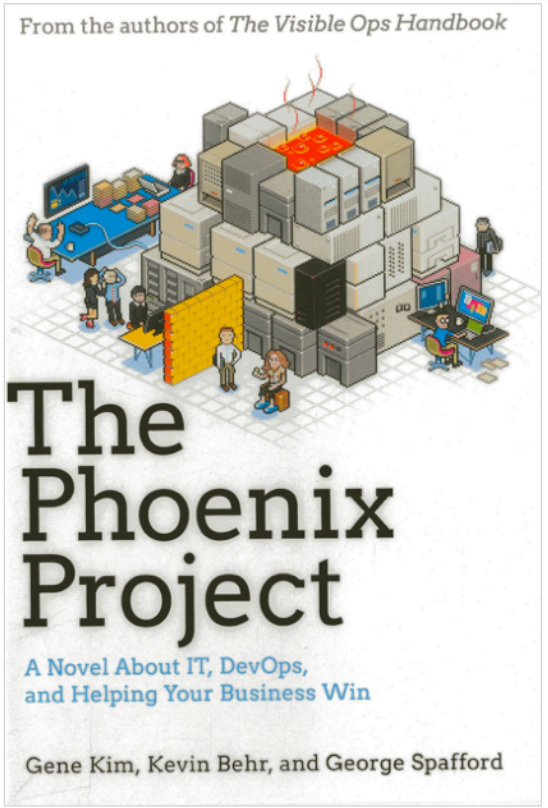
https://cloud.google.com/devops/state-of-devops

DORA stands for DevOps Research and Assessment

https://puppet.com/resources/report/2021-state-of-devops-report

DevOps Books



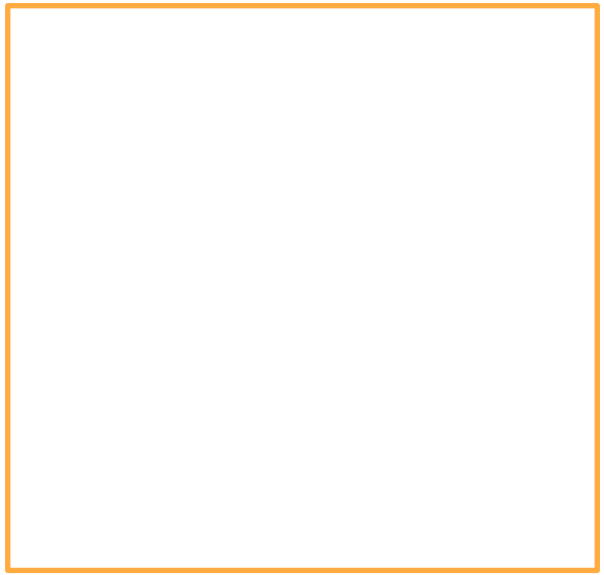


Containers

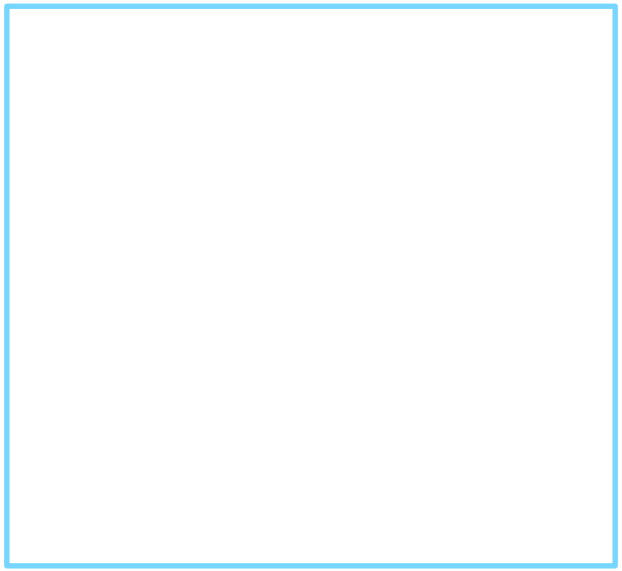
thoughtworks

If your application crashed in container, what should we do next?

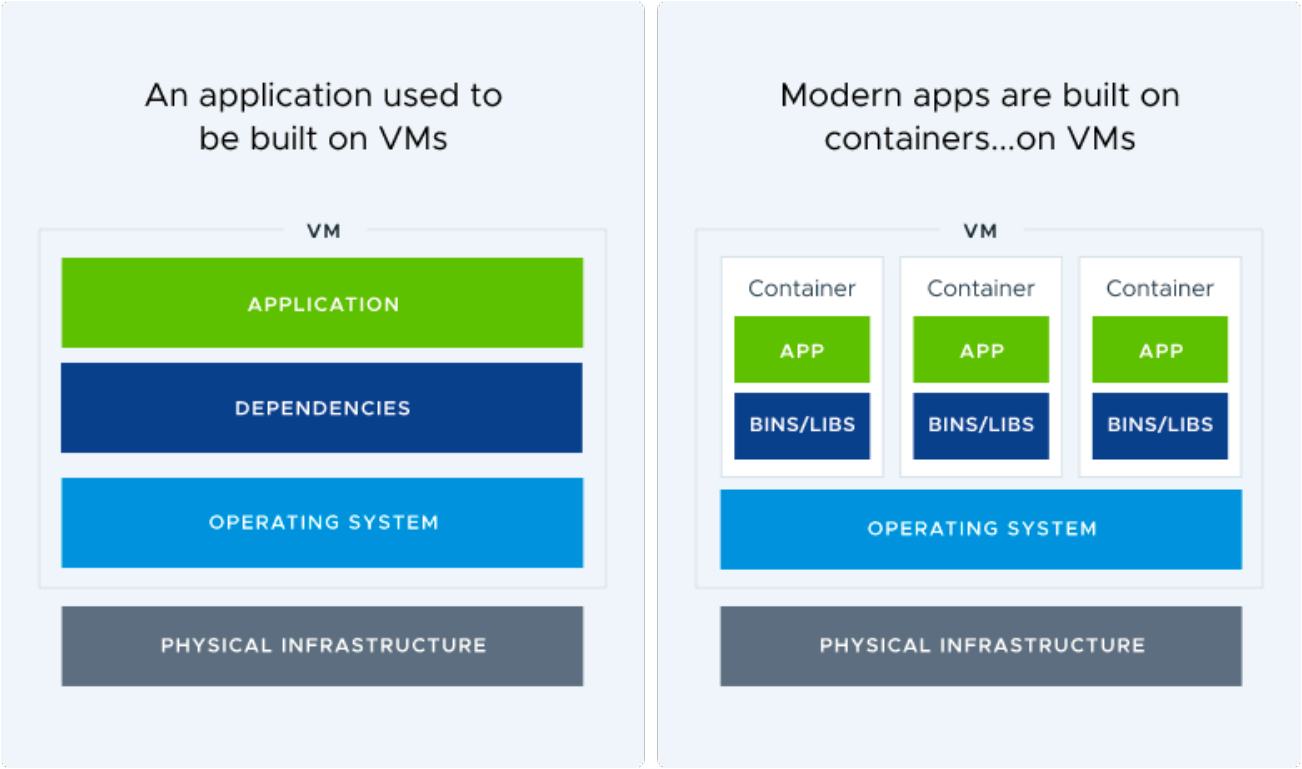
Re-create



Restart



Containers vs. VMs



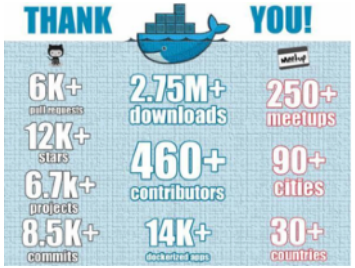
What is a Container?

A container encapsulates an application in a form that's **portable and easy to deploy**.

Containers can **run without changes on any compatible system**—in any private cloud or public cloud—and they consume resources efficiently, enabling high density and resource utilization.

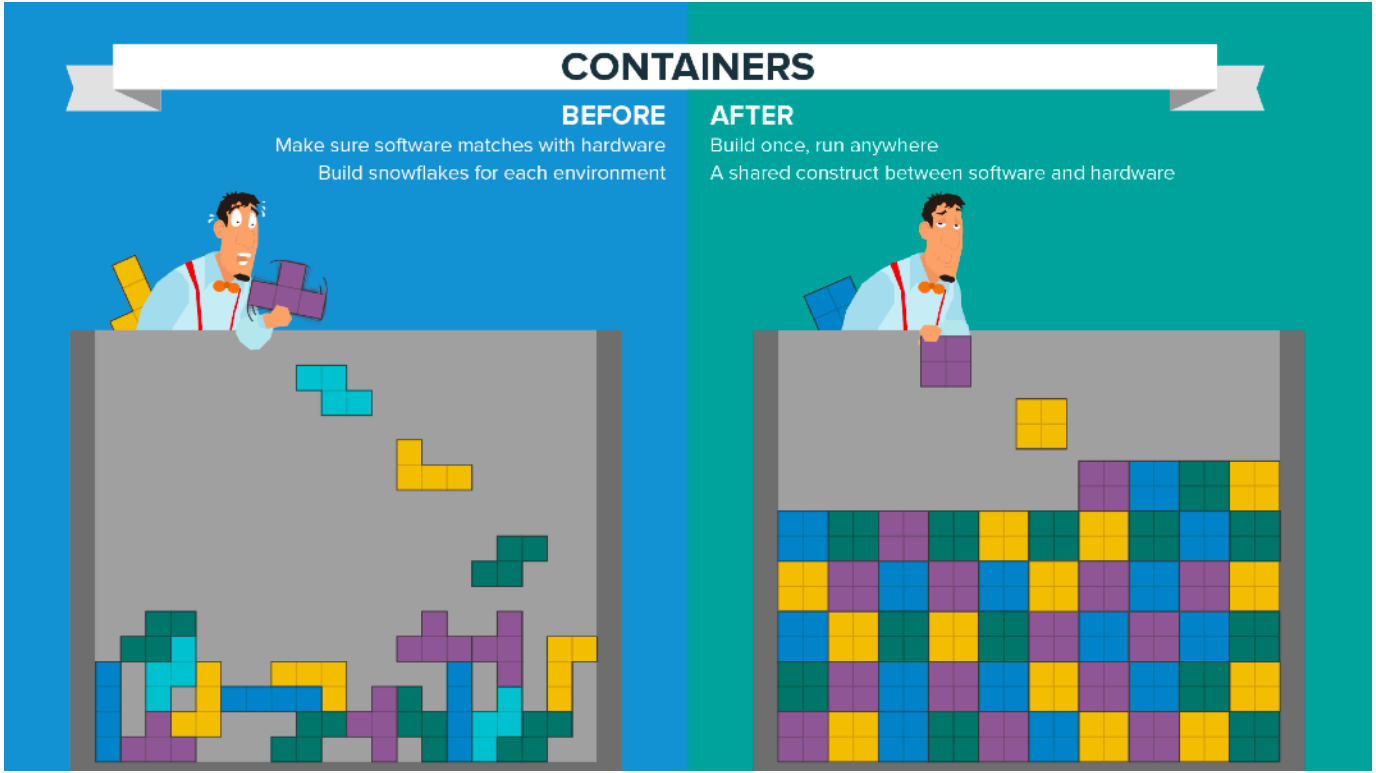
What is a Container?

File Access -> Resource -> System -> Application -> Cluster



Build once，Run anywhere

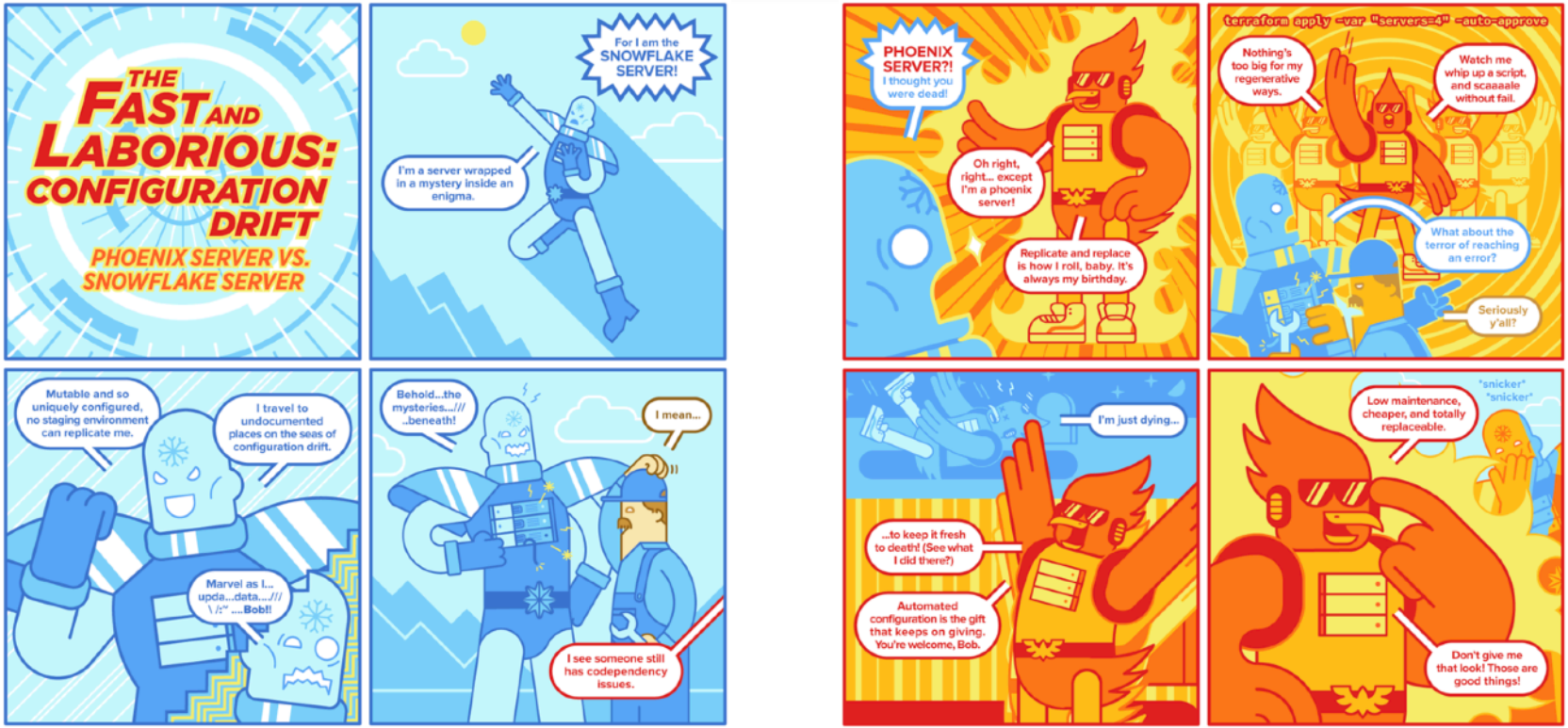
What is a Container?



The Benefits of Containers

- Deployment consistency **reduces time** to market.
- Execution consistency **improves quality**.
- Easier developer-to-operator handoffs **speed delivery**.
- **Better isolation** protects against failures.
- **Updating apps** to respond to CVEs is easier.

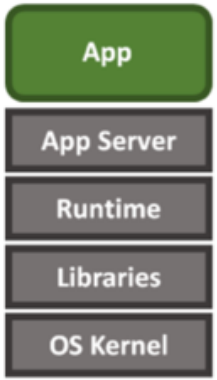
Immutable Infrastructure



https://www.digitalocean.com/community/tutorials/configuration-drift-phoenix-server-vs-snowflake-server-comic

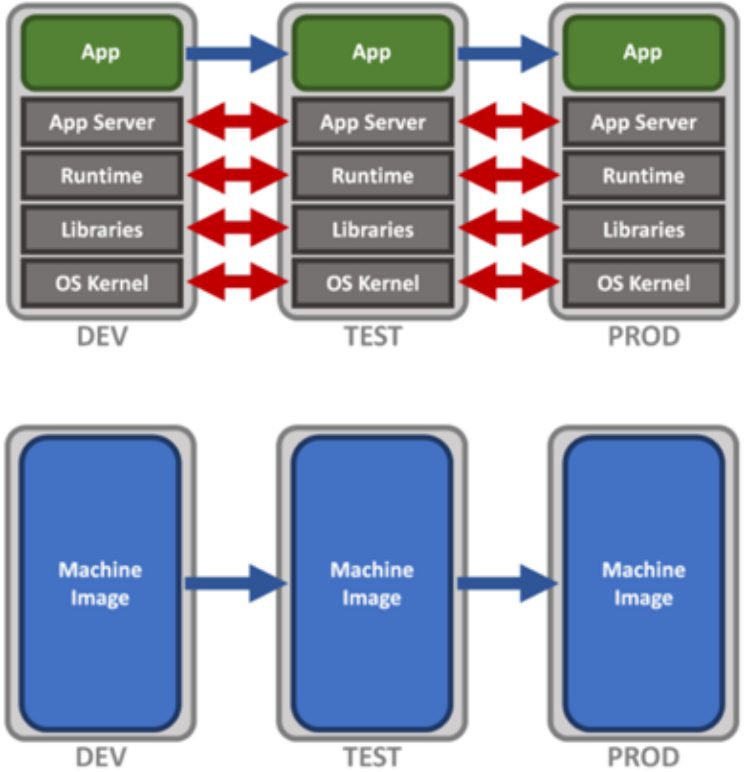
Immutable Infrastructure

An *immutable infrastructure* is an infrastructure paradigm in which servers are never modified after they're deployed.



Snowflakes

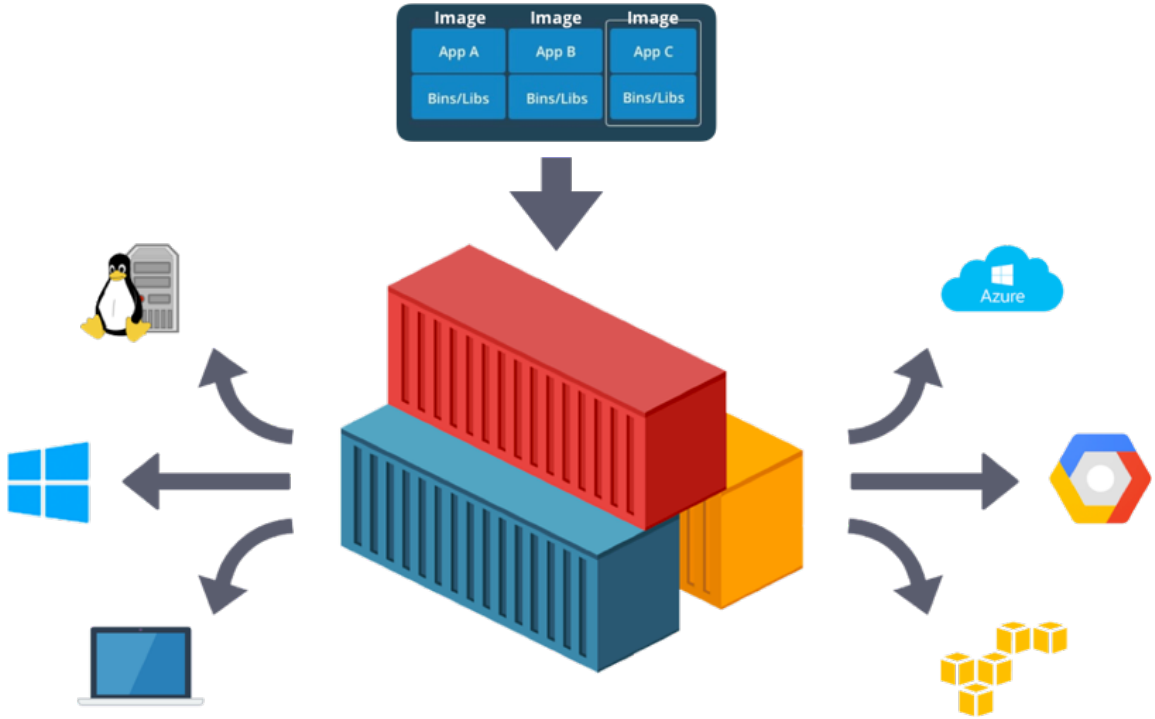
Phoenixes



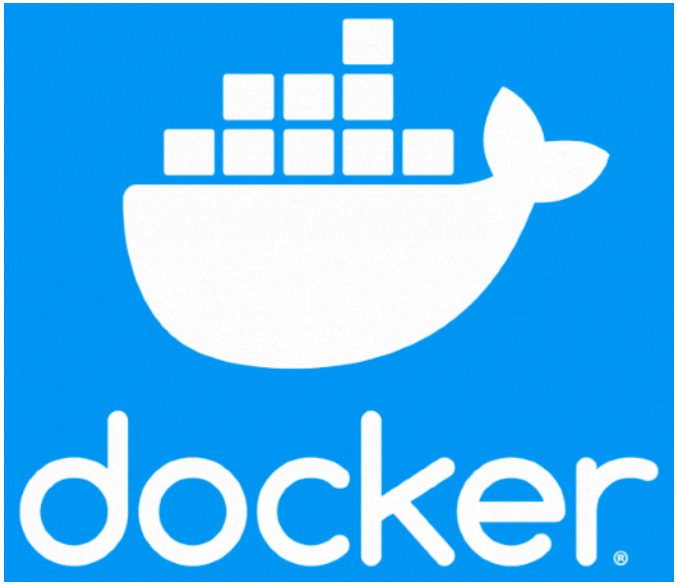
Container - In Fact Immutable

Build Once

Run Anywhere



Containers



CI/CD

thoughtworks

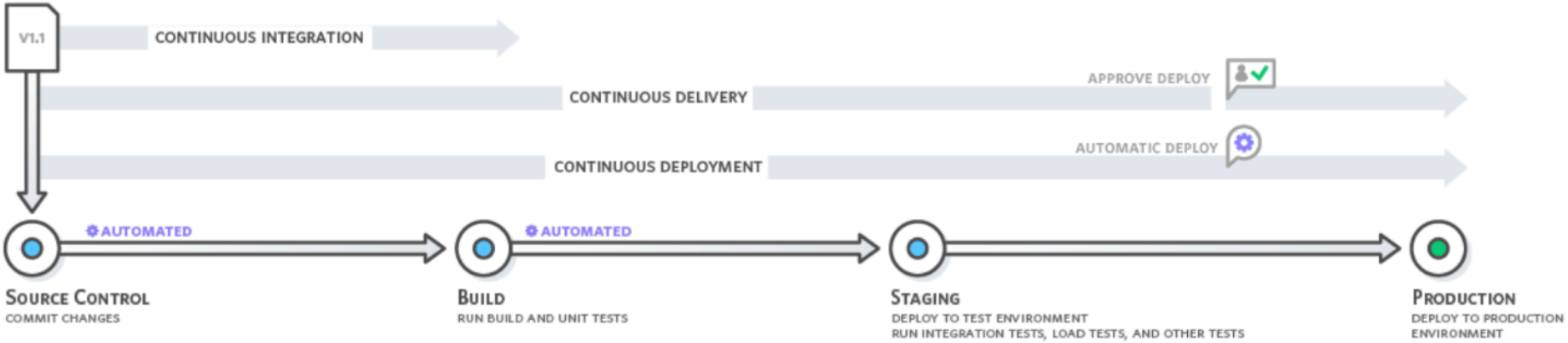
What is CI/CD(Continuous Integration and Continuous Delivery)?

Continuous Integration, **automates the build and unit testing stages** of the software release process.

Continuous Delivery, **automates the delivery** of software to a production-like environment.

Continuous delivery makes the act of releasing software reliable, so organizations can deliver software more frequently with less risk and get feedback faster.

CI/CD

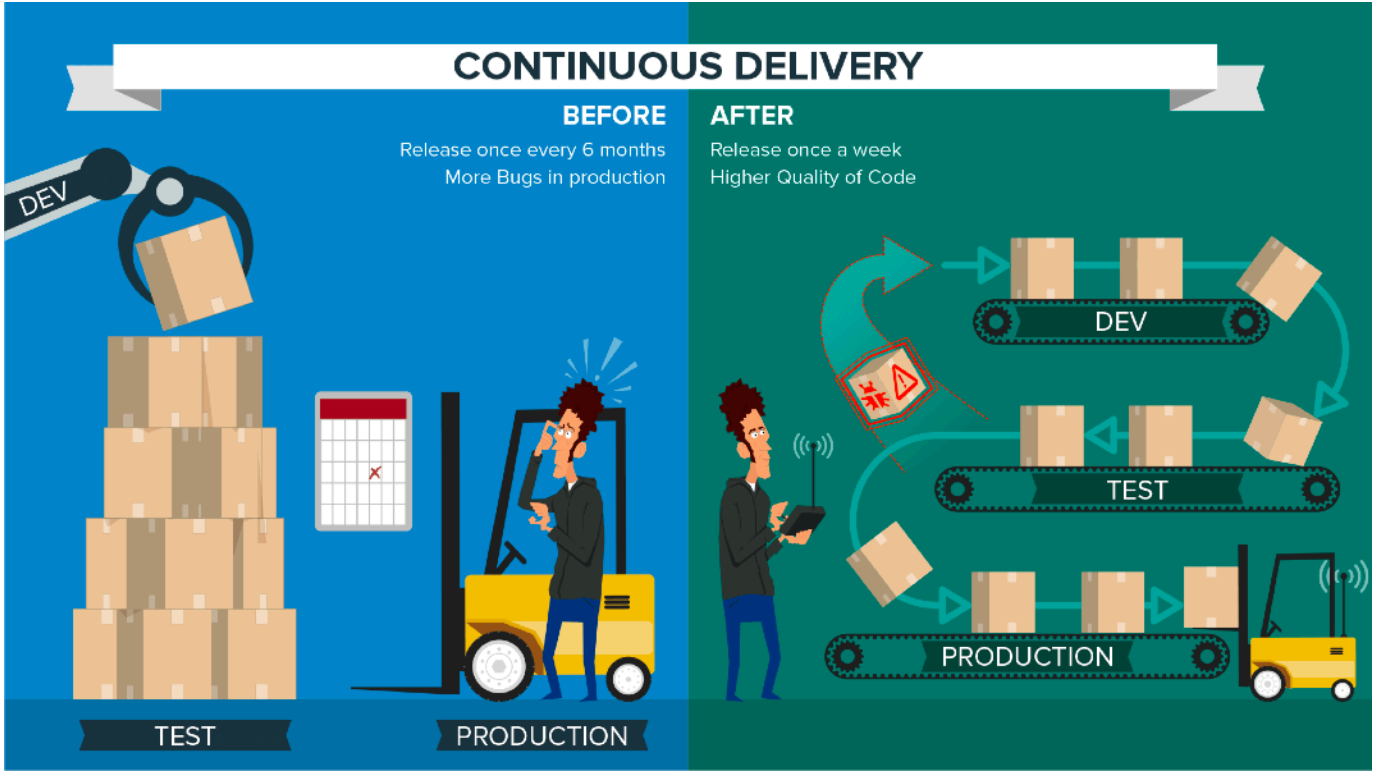


amazon.com/devops/continuous-integration/

CI/CD vs. Traditional Development

CI/CD	Traditional Approaches
Software is developed iteratively in small chunks based on **frequent user feedback**.	Software is developed in large, complex units with **less timely user feedback**.
Tests are written during development and applied throughout the development process to ensure code quality.	Software passes to a **separate QA team** for testing after the development process is completed.
Security patches and bug fixes **are quickly deployed via automation**.	Security patches and bug fixes are delivered immediately **through manual** exception processes or in bulk at irregular intervals.
New application code is **integrated frequently** with the existing code base and tested in real-world scenarios to ensure software is always ready for production.	New code is **integrated infrequently** with existing software, usually just prior to deployment and occurs only in predetermined release windows that may be quarterly or even less frequent.

Continuous Delivery



CI/CD Best Practice

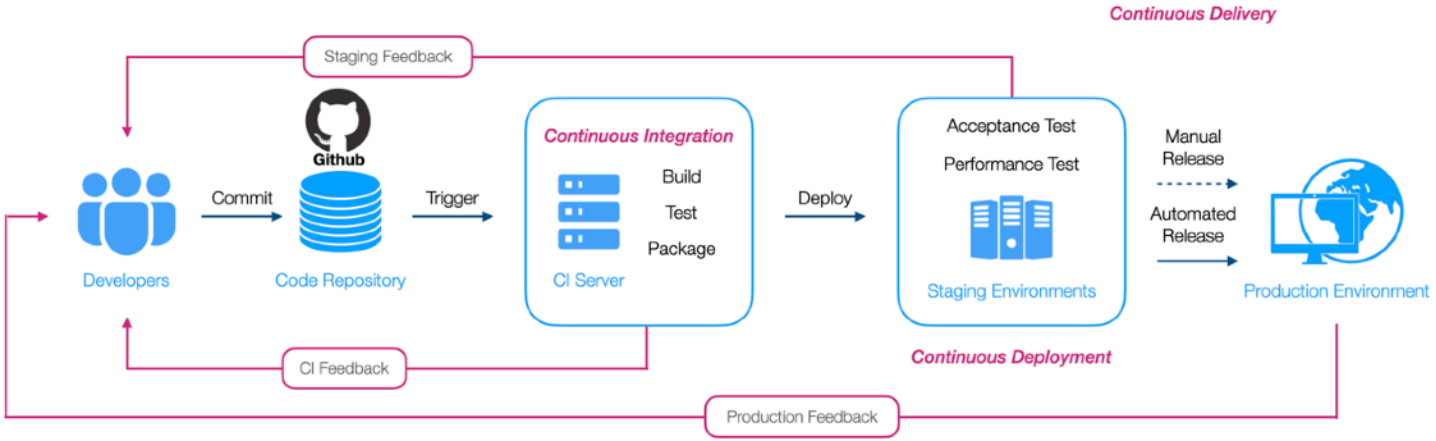
- Break down siloed **teams**.
- Commit to writing a lot more **tests**.
- Introduce new CI/CD tools and **automation**.

The Benefits of CI/CD

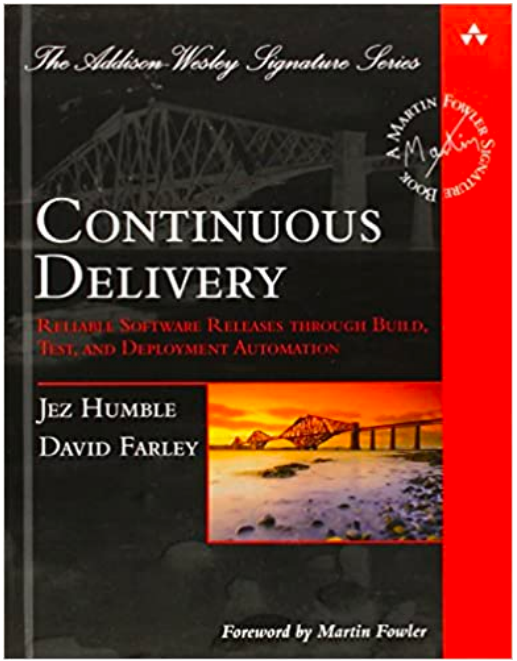
- **Reduce the risk** of software not functioning properly in production.
- Make **rapid iteration** a reality by combining CI/CD and DevOps or CI/CD and agile.
- **Automate** software configuration and all the processes needed to deploy to production.

CI/CD Tools

- **Version Control System**
- **CI Server**
- **Build Tool**



CI/CD Books



Continuous Integration

Jez Humble

integrate early & often

Deployment

deploy as the final stage of CI

Delivery

software is always deployable

Cloud Native Application Methodology

thoughtworks

The Twelve-Factor

The twelve-factor app is a methodology for building software-as-a-service apps that:

- Use **declarative formats** for setup automation, to minimize time and cost for new developers joining the project;
- Have a **clean contract** with the underlying operating system, offering maximum portability between execution environments;
- Are suitable for deployment on modern cloud platforms, obviating the need for servers and systems administration;
- **Minimize divergence** between development and production, enabling continuous deployment for maximum agility;
- And can **scale up** without significant changes to tooling, architecture, or development practices.

The Twelve-Factor

- I. **Codebase** -- One codebase tracked in revision control, many deploys
- II. **Dependencies** -- Explicitly declare and isolate dependencies
- III. **Config** -- Store config in the environment
- IV. **Backing services** -- Treat backing services as attached resources
- V. **Build, release, run** -- Strictly separate build and run stages
- VI. **Processes** -- Execute the app as one or more stateless processes
- VII. **Port binding** -- Export services via port binding
- VIII. **Concurrency** -- Scale out via the process model
- IX. **Disposability** -- Maximize robustness with fast startup and graceful shutdown
- X. **Dev/prod parity** -- Keep development, staging, and production as similar as possible
- XI. **Logs** -- Treat logs as event streams
- XII. **Admin processes** -- Run admin/management tasks as one-off processes

https://12factor.net/

The Twelve-Factor

- All these factors are *there to* ***help us develop an application which is modular, independent, portable, scalable, and observable***. Depending upon the application, we may be able to achieve them through other means better. It's also not necessary to adopt all the factors together, adopting even some of these may make us better than we were.

- These factors are quite simple and elegant. They hold greater importance in an age where we demand our applications to **have higher throughput and lower latency** with virtually no downtime and failure. *Adopting these factors gives us the right start from the beginning*. Combined with microservice architecture and containerization of applications, they just seem to hit the right spot.

Thank You !

thoughtworks