# DevOps

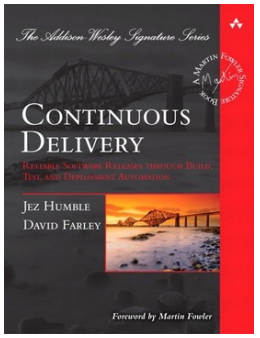
UA.DETI.IES - 2021/22

José Luís Oliveira

# Resources & Credits

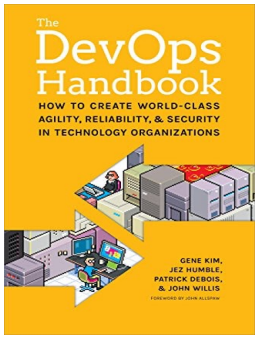
## Main

- C Vassallo, G Schermann,
  Advanced Software Engineering, Department of Informatics, UZH, 2021
  - https://www.ifi.uzh.ch/en/seal/teaching/courses/ase.html
- Jez Humble, David Farley,
  Continuous Delivery, Addison-Wesley, 2010.



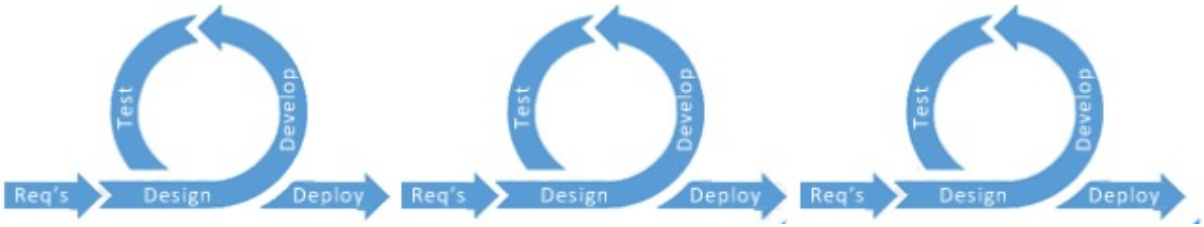

## Other resources

- DevOps Tutorials
  - https://www.tutorialspoint.com/devops_tutorials.htm
- G Kim, J Humble, P Debois, J Willis,
  The DevOps Handbook,
  IT Revolution Press, 2016.

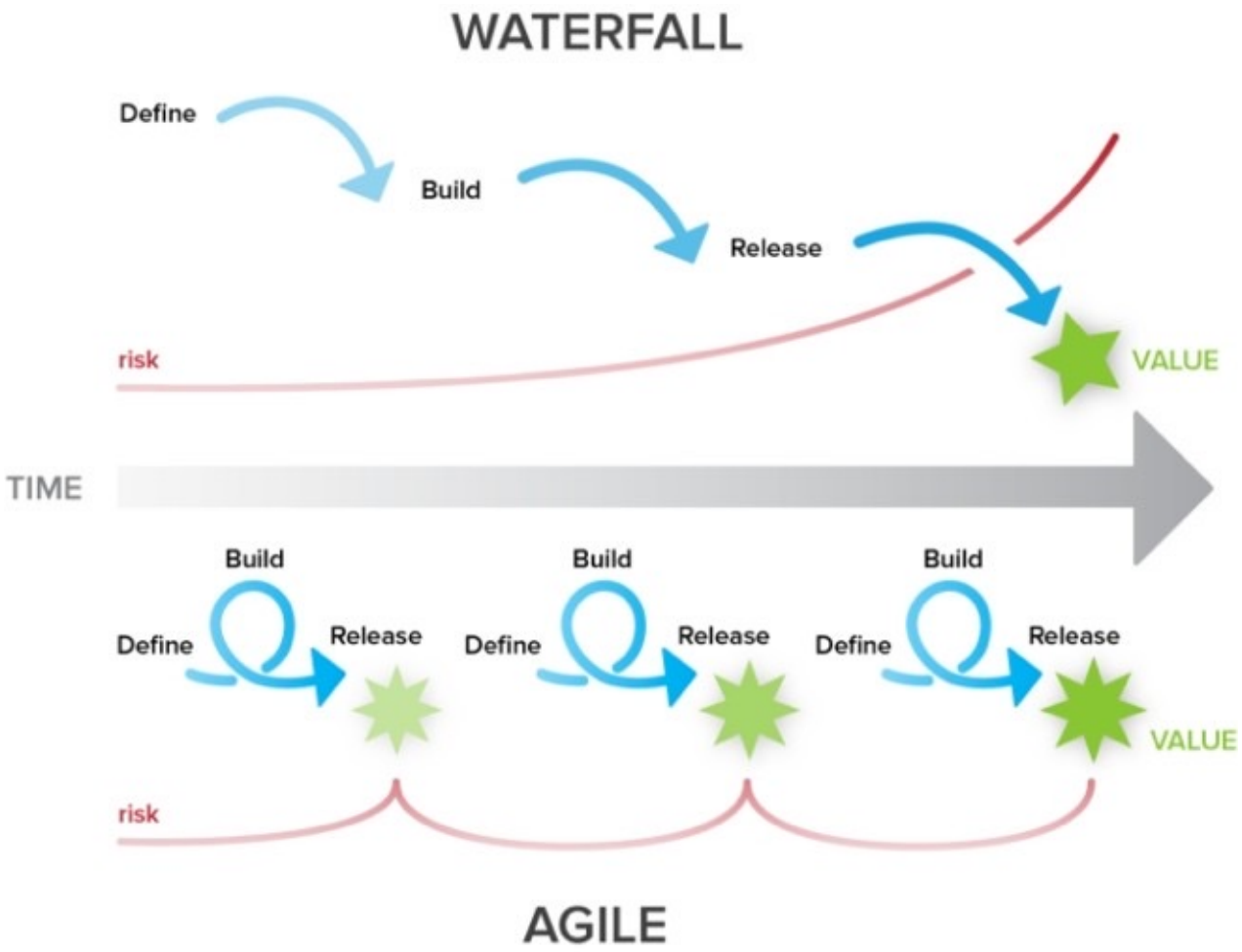


# Agile Methodology

- Each project is broken up into several iterations
- All iterations should be of the same time duration (between 2 to 8 weeks)
- At end of each iteration a working product should be delivered



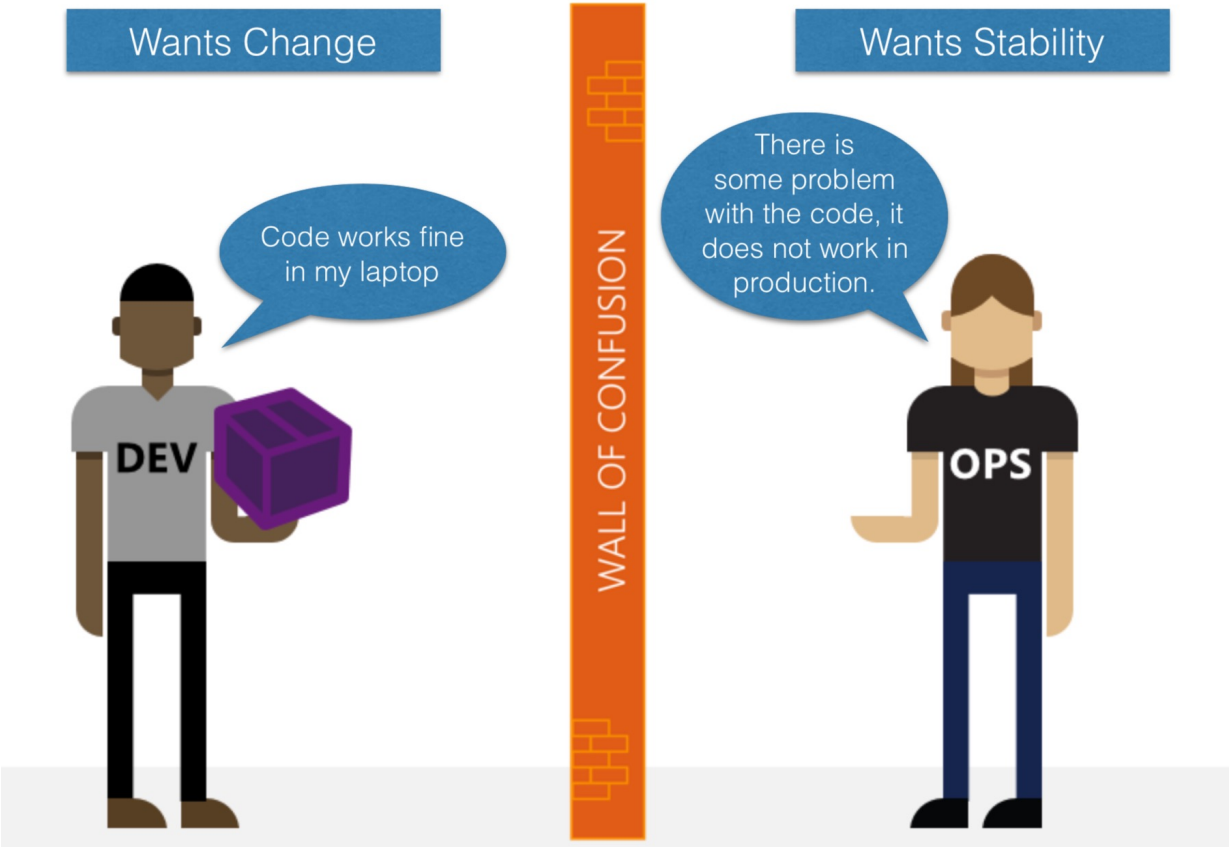
# Waterfall vs Agile




https://blog.ganttpro.com/en/waterfall-vs-agile-with-advantages-and-disadvantages/

# What are the limitations of Agile?

- The Development part is continuous
- The Operation is NOT continuous




https://blogs.msdn.microsoft.com/uk_faculty_connection/2016/06/23/devops-the-wall-of-confusion-understanding-the-basics-of-devops/

# Dev vs Ops

- The **development** team kicks things off by "throwing" a software release "over the wall" to Operations.

- **Operations** picks up the release artifacts and begins preparing for their deployment.
- They also hand edit configuration files to reflect the production environment, which is significantly different than the Development environments.

# Dev vs Ops

- In case of failure…

  - Developers are called in to help troubleshoot.
  - Operations claims that Development gave them faulty code.
  - Developers respond by pointing out that it worked just fine in their environments
    - So, it must be the case that Operations did something wrong.
  - Developers are having a difficult time even diagnosing the problem
    - because the configuration, file locations, and procedure used to get into this state is different then what they expect.

**Solution?**

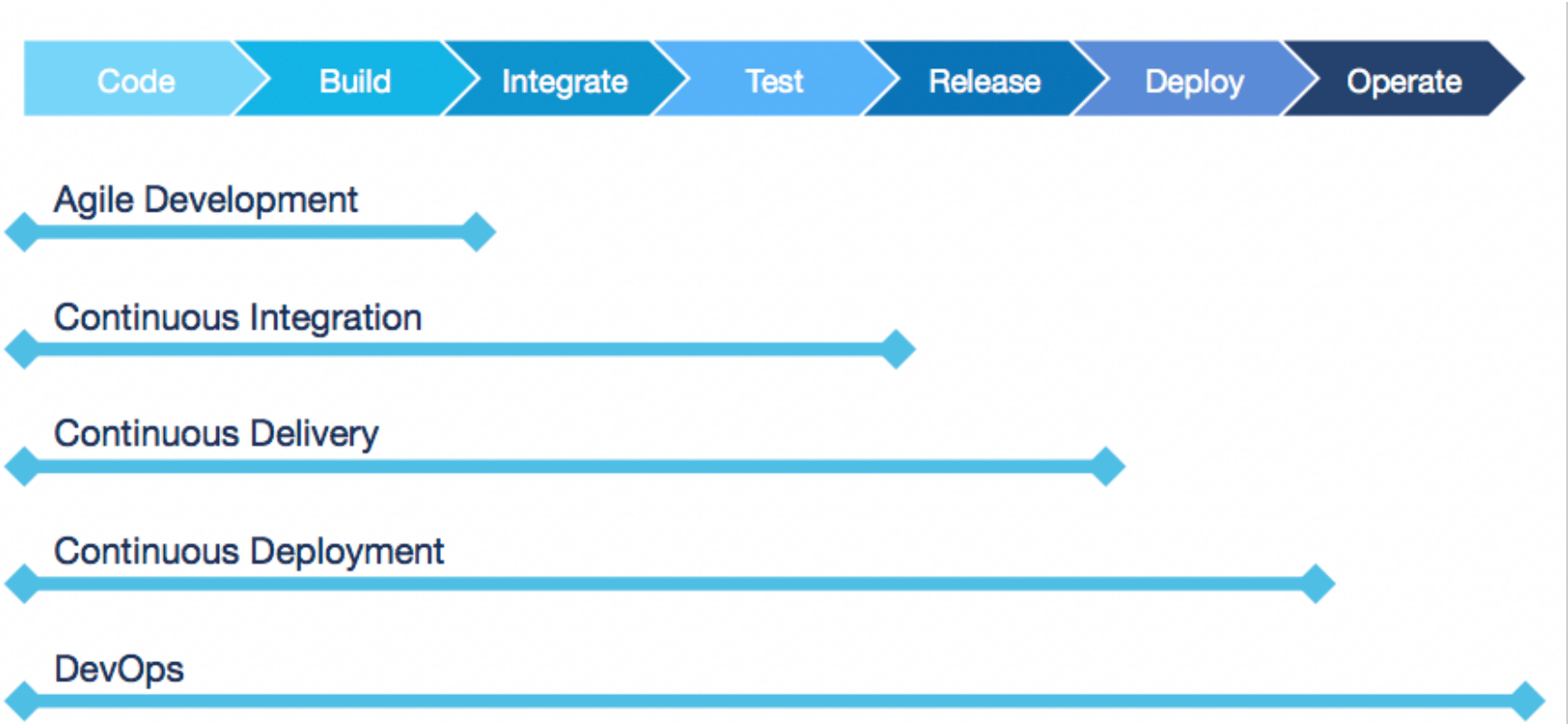
# DevOps

- DevOps is about removing the barriers between two traditionally siloed teams
  - development and operations.
- In some organizations, there may not even be separate teams;
  - engineers may do both.

# DevOps

- The **two teams work together**
  - to optimize both the productivity of developers and the reliability of operations.
- They **communicate frequently**
  - increase efficiencies and improve the quality of services they provide to customers.
- They take **full ownership for their services**
  - thinking about the end customer's needs and how they can contribute to solving those needs.
- Teams view the entire development and infrastructure lifecycle as part of their responsibilities.

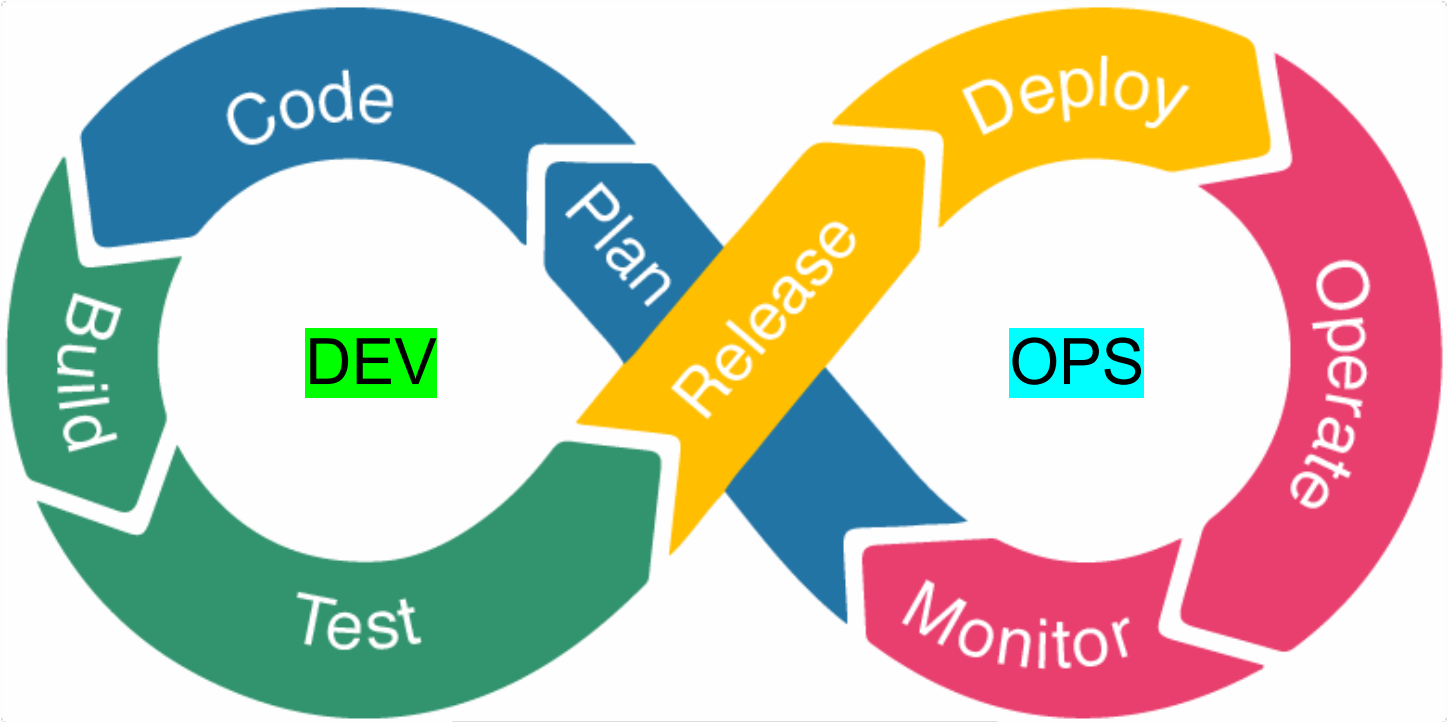
# DevOps Stages



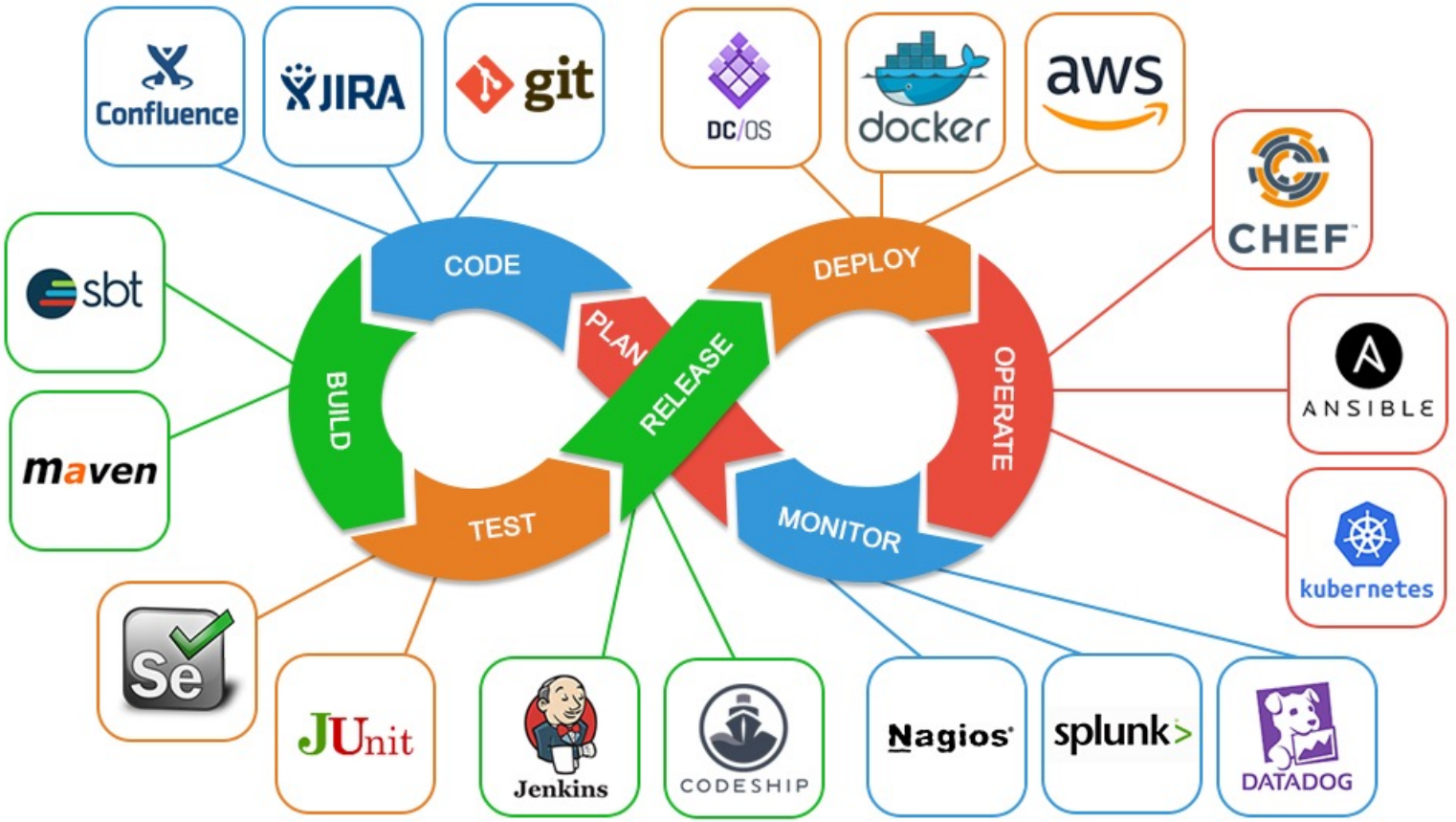
# DevOps Stages

- Agile Development
  - Source Code Management (Plan, Code, Build)
  - Maintain different versions of the code.
- Continuous Integration (Integrate, Test)
  - Compile, Code Review, Unit Testing, Integration Testing.
- Continuous Delivery (Release)
  - Deploying the build application
  - Performing user acceptance testing (UAT).
- Continuous Deployment (Deploy, Operate)
  - Deploying the tested/accepted application.
- Continuous Monitoring

# DevOps Lifecycle



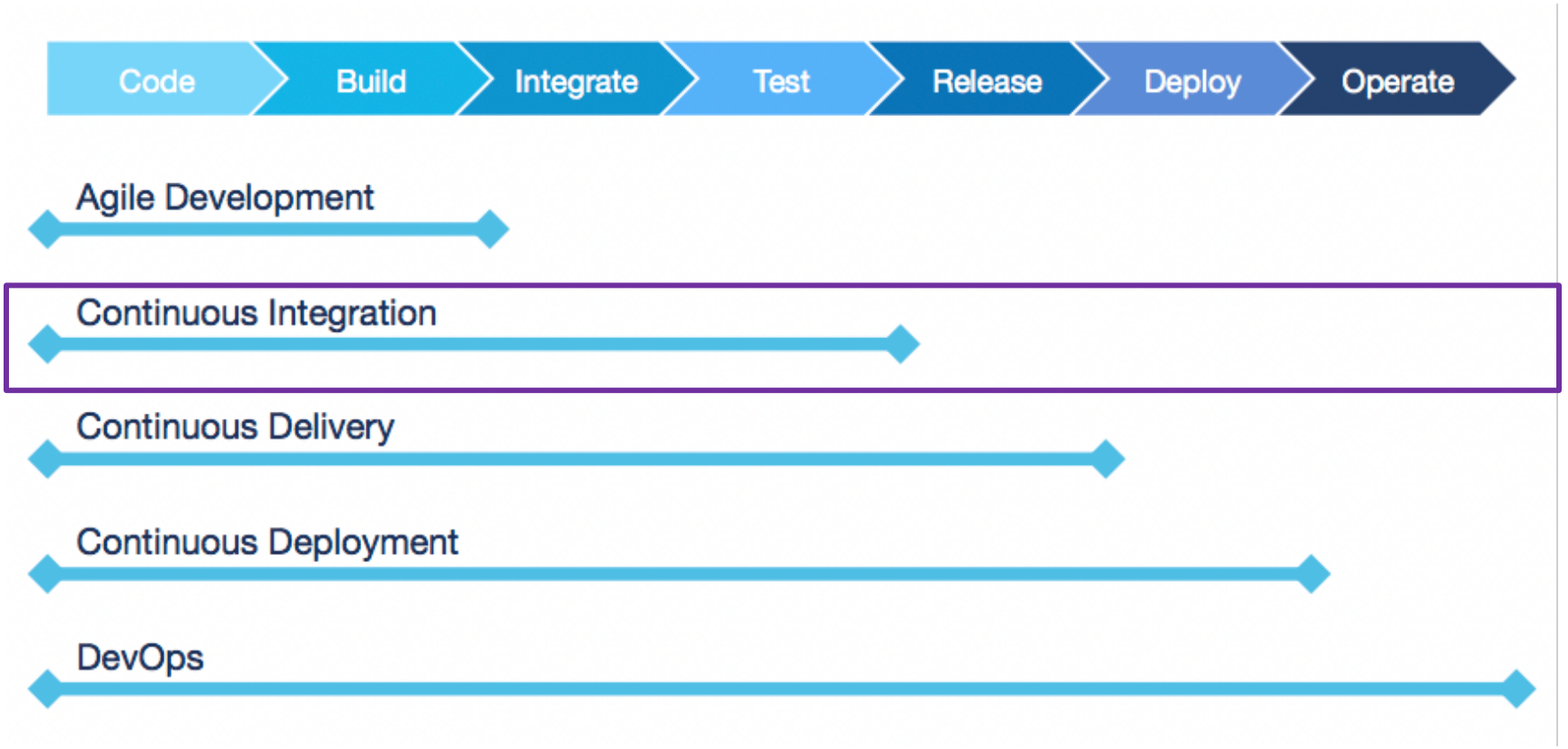
# How to implement DevOps?



# A Key asset: Infrastructure as Code (IaC)

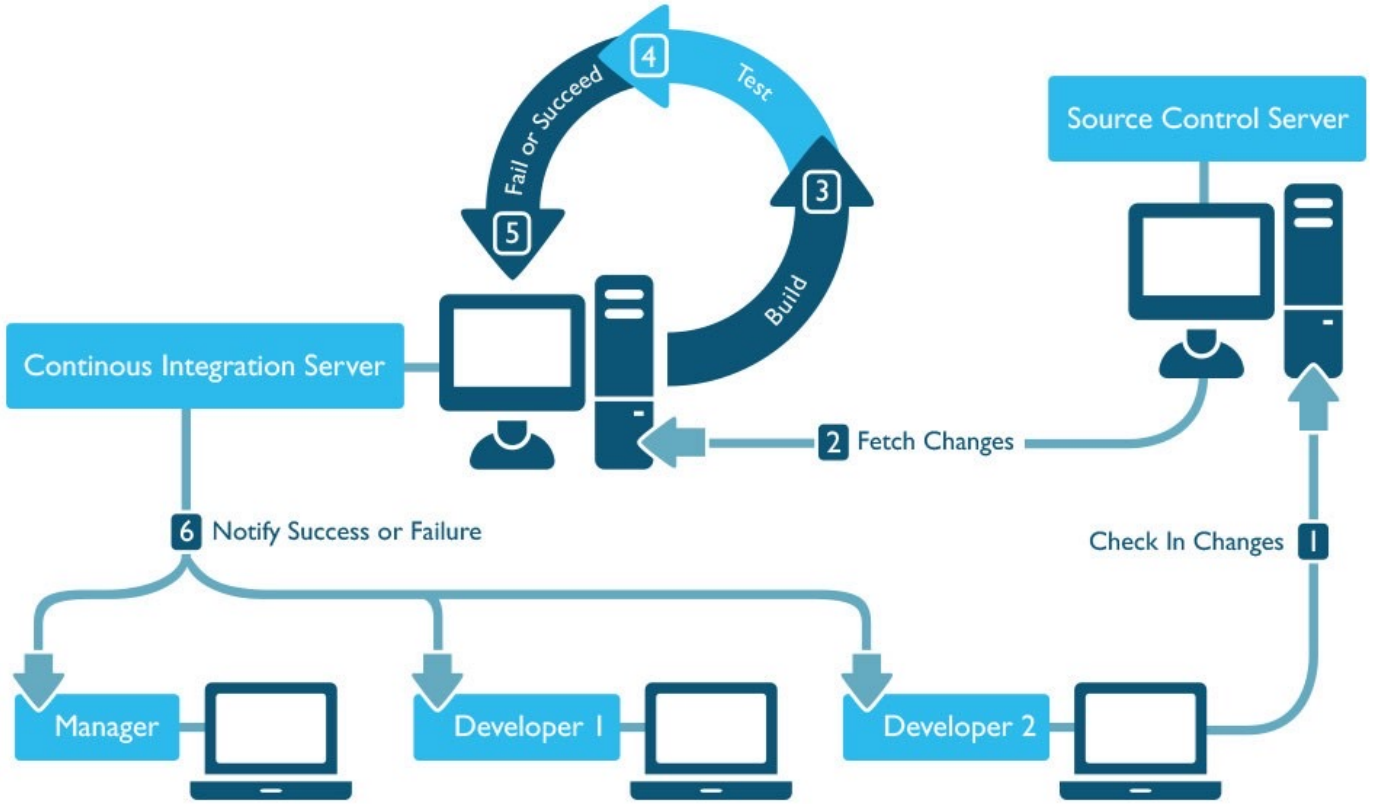
- ***a.k.a.*, Programmable infrastructure** or **software-defined infrastructure.**
  - The infrastructure is described by code and can be tracked, validated, and reconfigured in an automated way.
  - DevOps engineers can interface with infrastructure using code-based tools and treat infrastructure in a manner similar to how they treat application code.
- Configuration Management
  - Developers and system administrators use code to automate operating system and host configuration, operational tasks, and more.
  - The use of code makes configuration changes repeatable and standardized.

# Continuous Integration (CI)



# Core Idea of CI

- Continuous integration is a process where the code is checked into a repository very frequently.




https://insights.sei.cmu.edu/devops/2015/01/continuous-integration-in-devops-1.html

# Why is CI crucial for DevOps?

- Errors Detected Early
  - If it is an error in the local copy or code, a build failure will occur at the "appropriate stage".
  - It forces the developer to fix the bug before proceeding further. QA teams will also be benefited from this as they will mostly be working on stable builds.
- Setting the stage for CI/CD
  - CI reduces manual intervention because build, sanity, and other tests are all supposed to be automated.
  - This paves way for a successful continuous delivery.
- → Project Confidence

# Getting started with CI

- Build Script
  - e.g., make, ant, maven, gradle, …
  - It is a script, or a set of scripts, we use to compile, test, inspect, and deploy software.
- Version Control System
  - e.g., git, github, gitlab, …
  - Allow recording changes to a file or set of files over time so that we can recall specific versions later.
- CI Server
  - e.g., Hudson, TravisCI, Jenkins, …
  - A server that runs an integration build whenever a change is committed to the version control repository.
  - Although it is recommended to run a build at every change builds can also be scheduled (e.g., nightly builds).
- Automation testing framework
  - e.g., Selenium, Appium, TestComplete, UFT, …

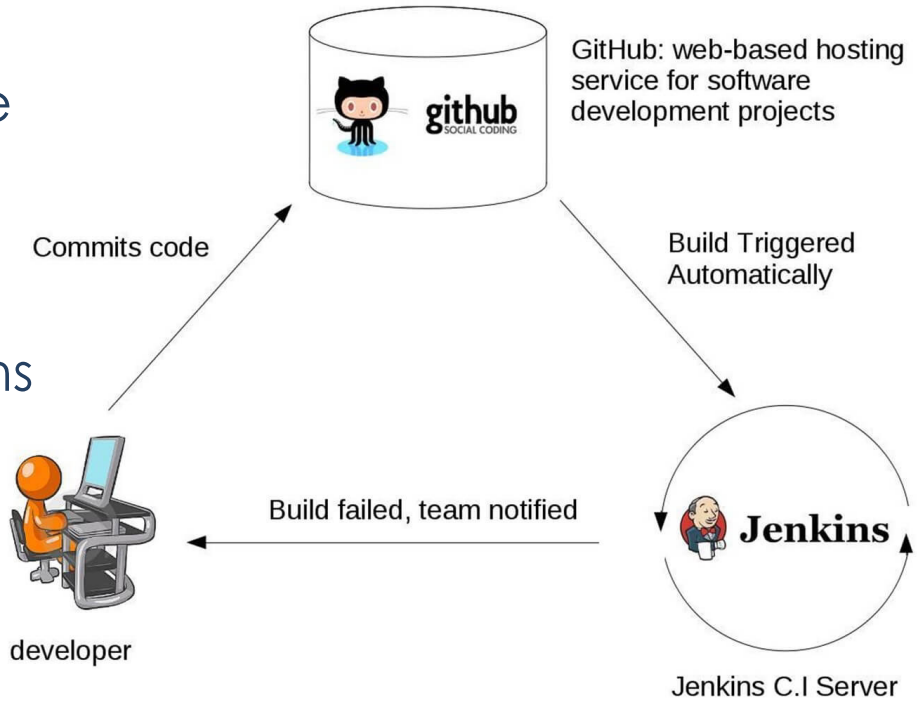
# Build script with Maven

- Lifecycle phases (or goals)
  - **validate** - check if all information necessary for the build is available
  - **compile** - compile the source code of the project
  - **test-compile**: compile the test source code
  - **test**: run unit tests
  - **package**: package compiled source code into the distributable format (jar, war, …)
  - i**ntegration-test**: process and deploy the package if needed to run integration tests
  - **install** - install the package into the local repository, for use as a dependency in other projects locally
  - **deploy** - copies the final package to the remote repository for sharing with other developers and projects.

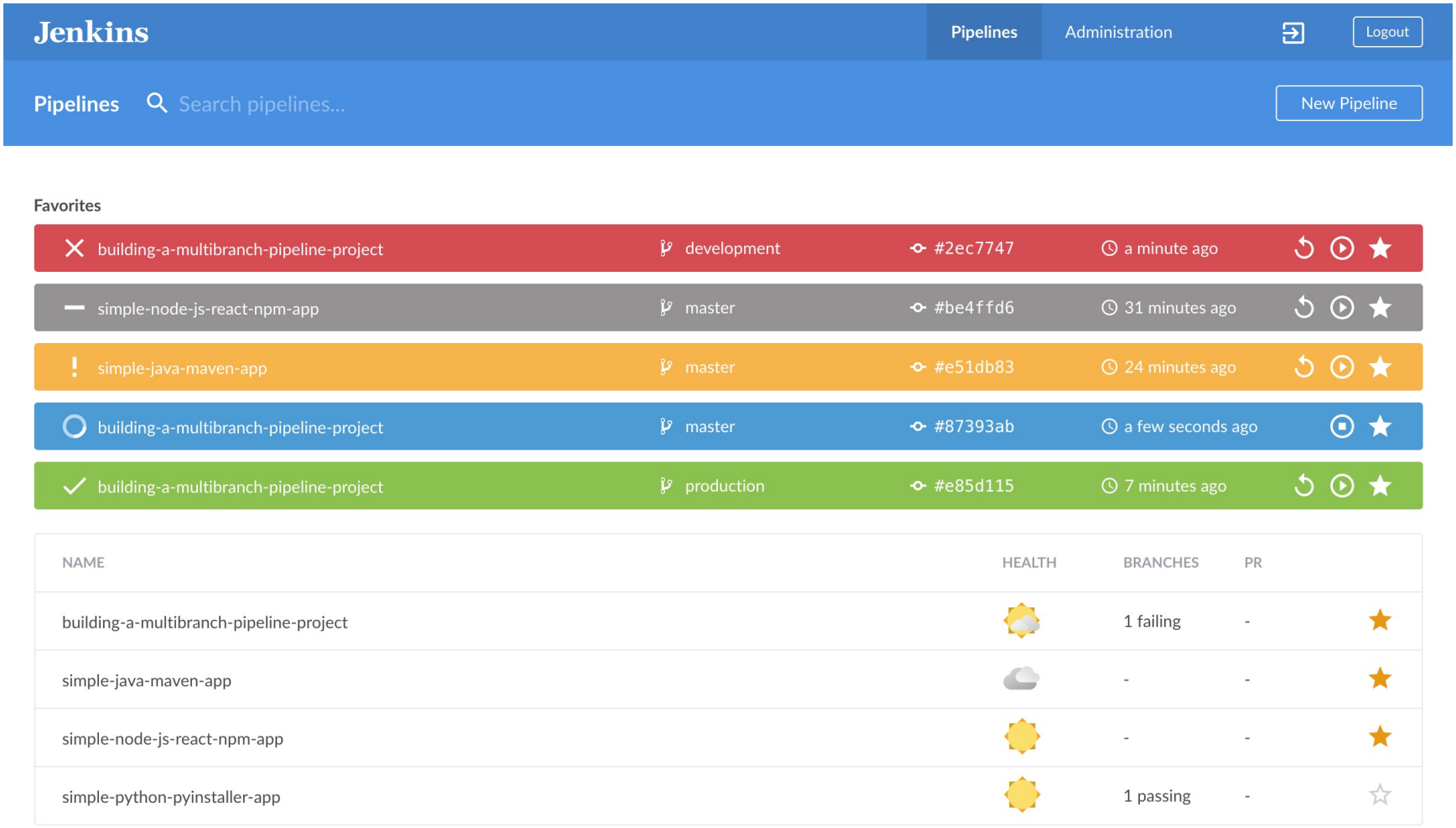
- Convention over Configuration

# CI Server, with Jenkins

- The code is built and test as soon as Developer commits code.
- If the build is successful, then Jenkins will deploy the source into the test server and notifies the deployment team.
- If the build fails, then Jenkins will notify the errors to the developer team.



# CI Server, with Jenkins



# CI Best practices

- Versioning
  - Shared repository to maintain code
  - Merge Daily
  - Short-Lived Branches
- Build Configuration
  - Independent build
  - Build at every change
  - Build Threshold
- Team Policy
  - Stop the line
  - Commit Often
  - Continuous Feedback
  - Fail Fast
  - Fast Builds

# CI Best practices: Versioning

- **Shared repository** to maintain code
  - All source files (executable code, configuration, host environment, and data) are committed to a version-control repository.
- **Merge Daily**
  - Changes committed to the mainline are applied to each branch on at least a daily basis.
- **Short-Lived Branches**
  - Branches must be short lived – ideally less than a few days and never more than an iteration.

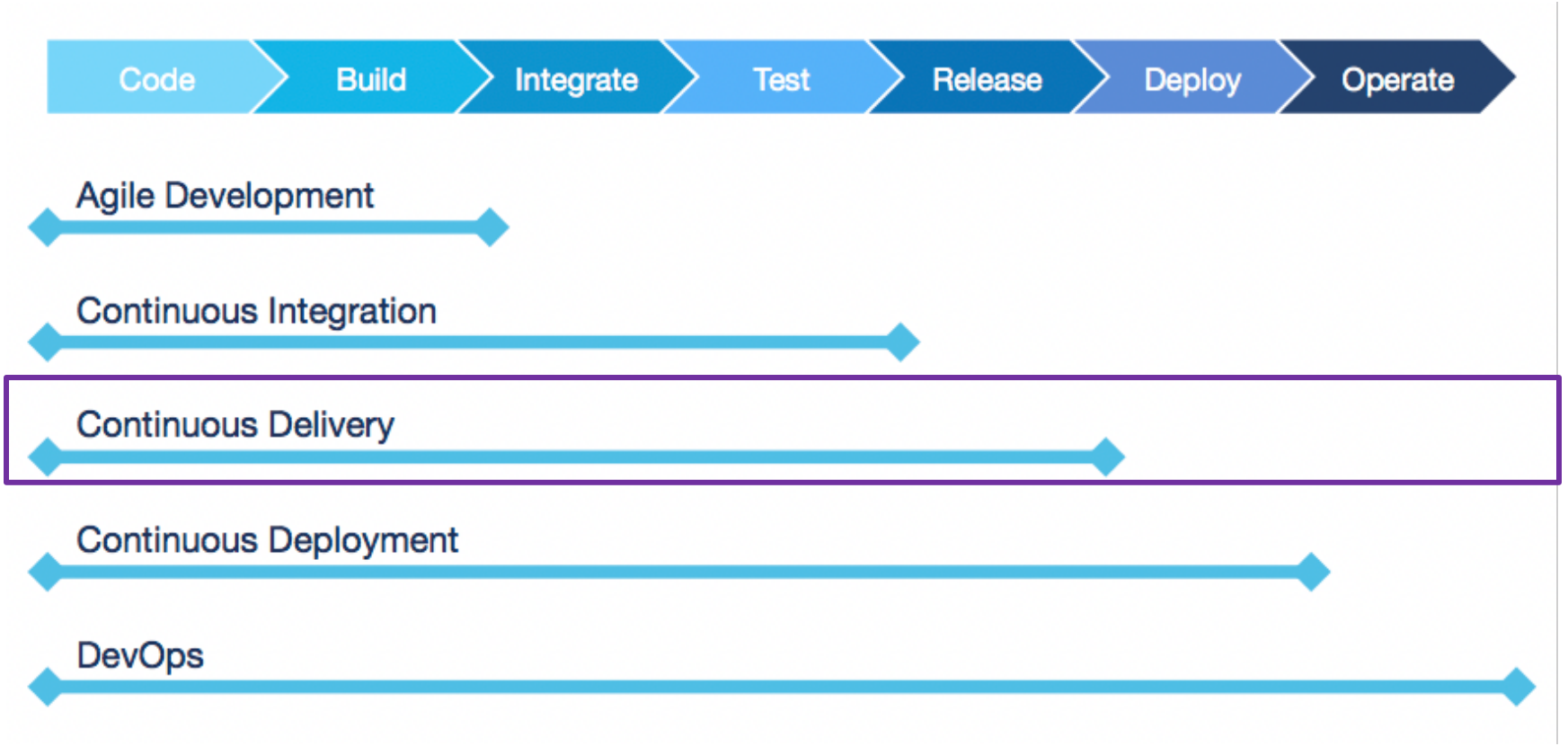
# CI Best practices: Build Configuration

- **Independent** build
  - Write build scripts that are decoupled from IDEs. These build scripts are executed by a CI system so that software is built at every change.
- Build at **every change**
  - Building and testing software with every change committed to a project's version control repository.
- Build **Threshold**
  - Fail a build when a project rule is violated – such as architectural breaches, slow tests, and coding standard violations.

# CI Best practices: Team Policy

- Stop the line
  - Fix software delivery errors as soon as they occur. No one checks in on a broken build as the fix becomes the highest priority.
- Commit Often
  - Each team member checks in regularly to trunk - at least once a day but preferably after each task to trigger the CI system.
- Continuous Feedback
  - Send automated feedback from CI system to all Cross-Functional Team members.
- Fail Fast
  - Fail the build as soon as possible. Design scripts so that processes that commonly fail run first.
- Fast Builds
  - A build provides feedback on common build problems as quickly as possible - usually in under 10 minutes.

# Continuous Delivery (CD)



# Continuous Delivery

- Software Eng. based on **CI**
  - CI focuses on development teams; the output of CI is the input to manual testing process and to the rest of the release process
  - Plenty of time may be "wasted" on this way through testing and operations
    - e.g., testers wait for "good" builds of the software, operations teams wait for documentation or fixes
- **CD** prevents this by:
  - close, collaborative working relationship between the stakeholders involved in delivery
    - DevOps culture
  - extensive automation of all parts of the delivery process
    - CD pipeline

# CI/CD pipeline (example as code in Jenkins)

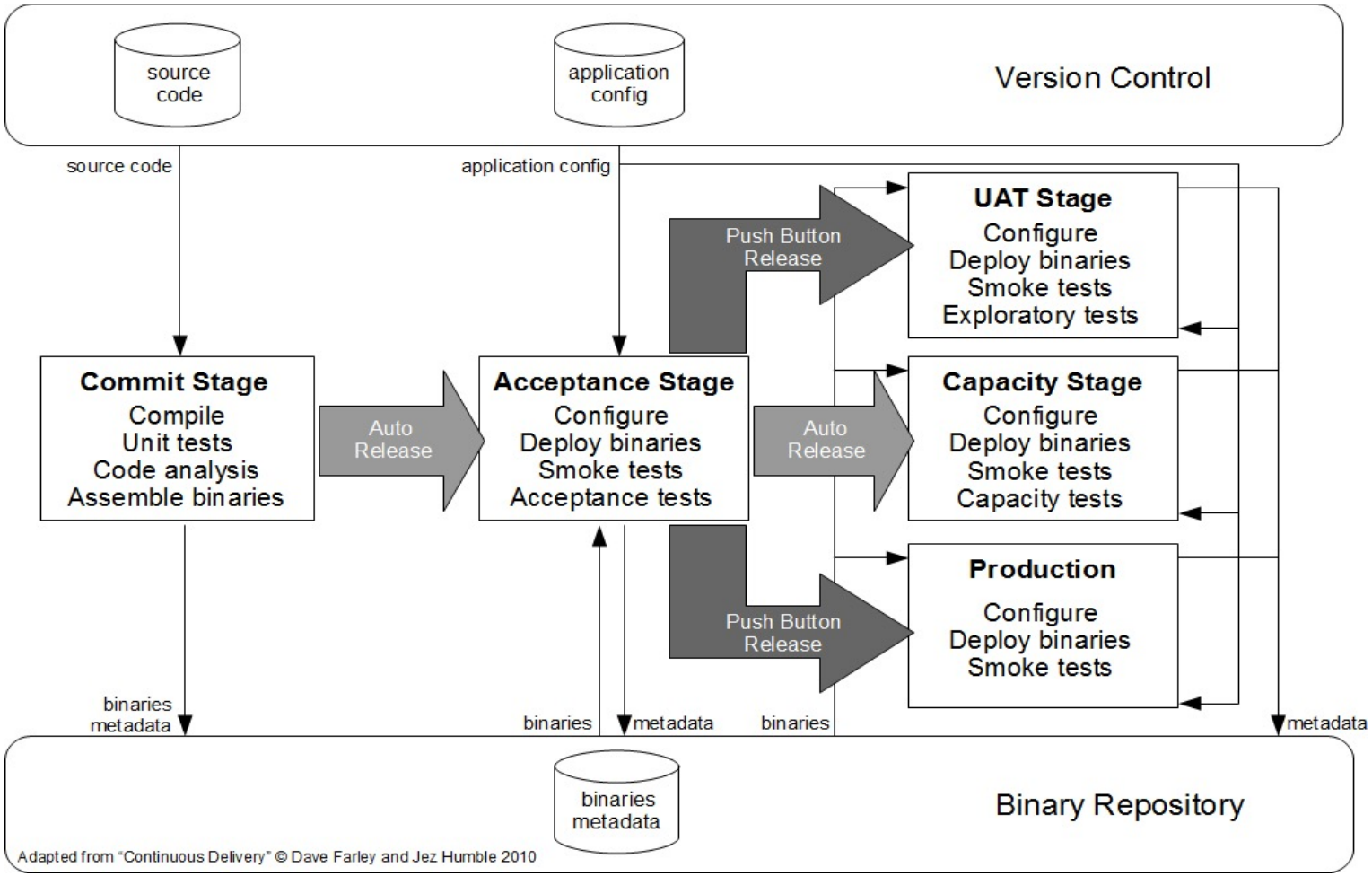
*Jenkinsfile (Declarative Pipeline)*

```
pipeline {
    agent any ❶
    stages {
        stage('Build') { ❷
            steps {
                // ❸
            }
        }
        stage('Test') { ❹
            steps {
                // ❺
            }
        }
        stage('Deploy') { ❻
            steps {
                // ❼
            }
        }
    }
}
```

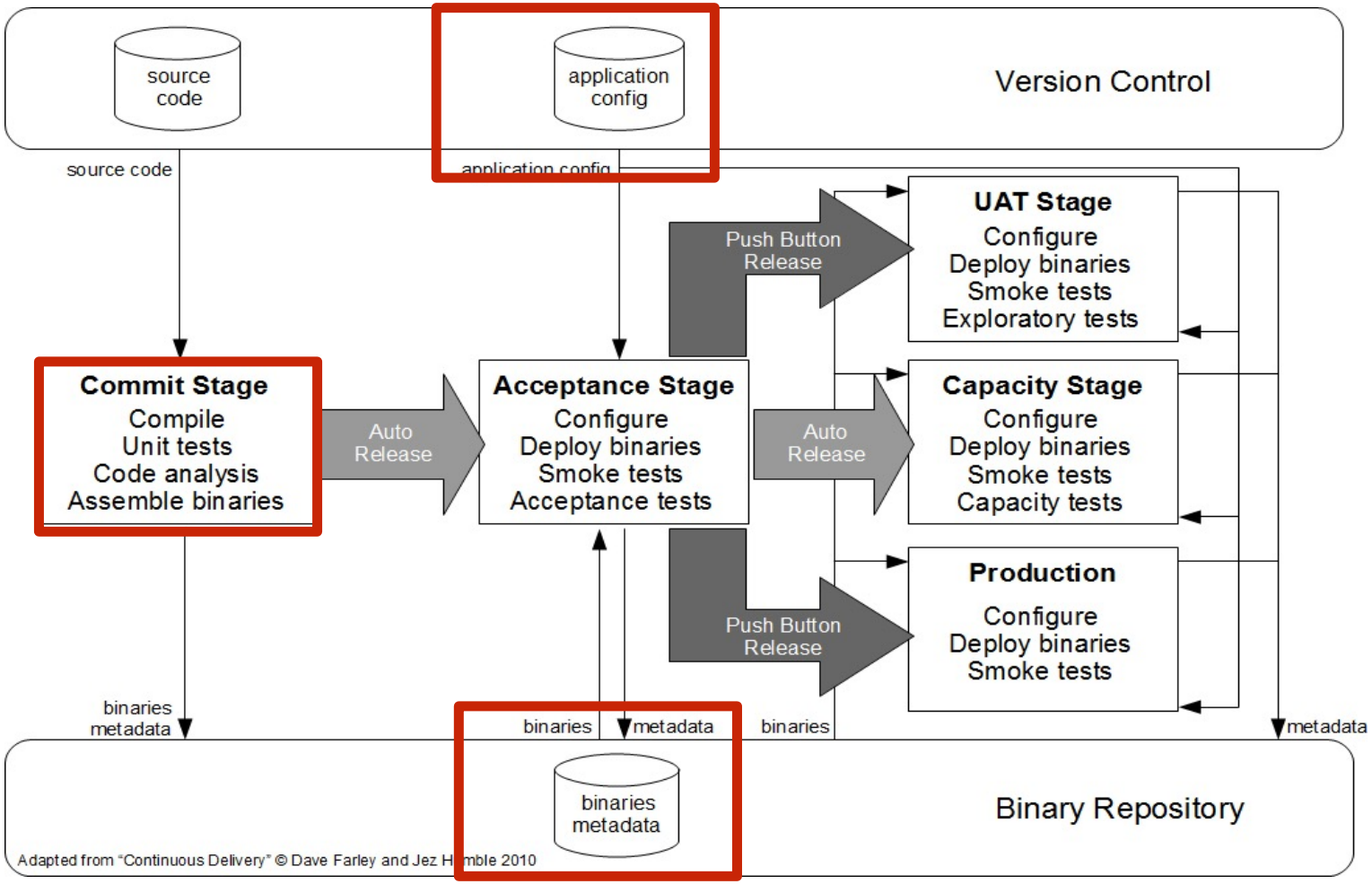
1. Execute this Pipeline or any of its stages, on any available agent.
2. Defines the "Build" stage.
3. Perform some steps related to the "Build" stage.
4. Defines the "Test" stage.
5. Perform some steps related to the "Test" stage.
6. Defines the "Deploy" stage.
7. Perform some steps related to the "Deploy" stage.

*https://jenkins.io/doc/book/pipeline/#pipeline-example*

# Continuous Delivery Pipeline

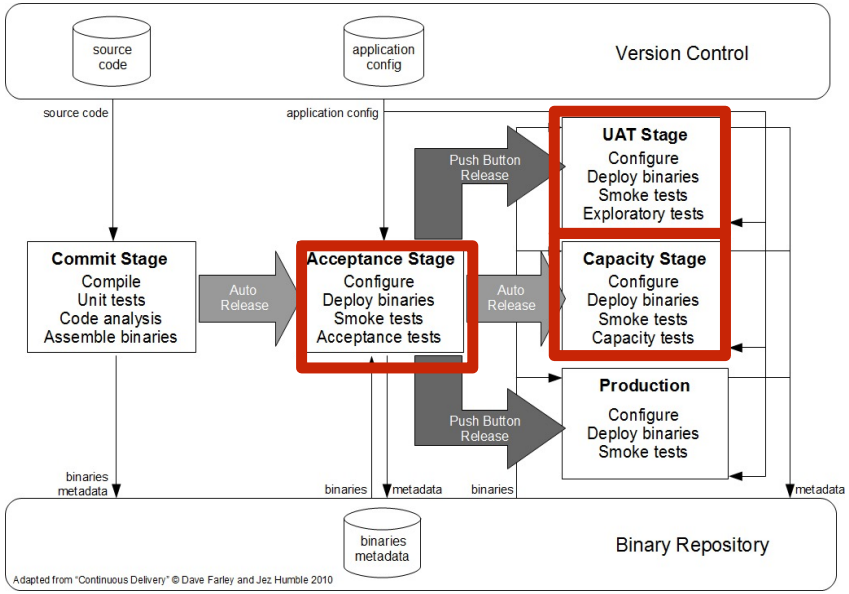


# CD Pipeline: CI Phase



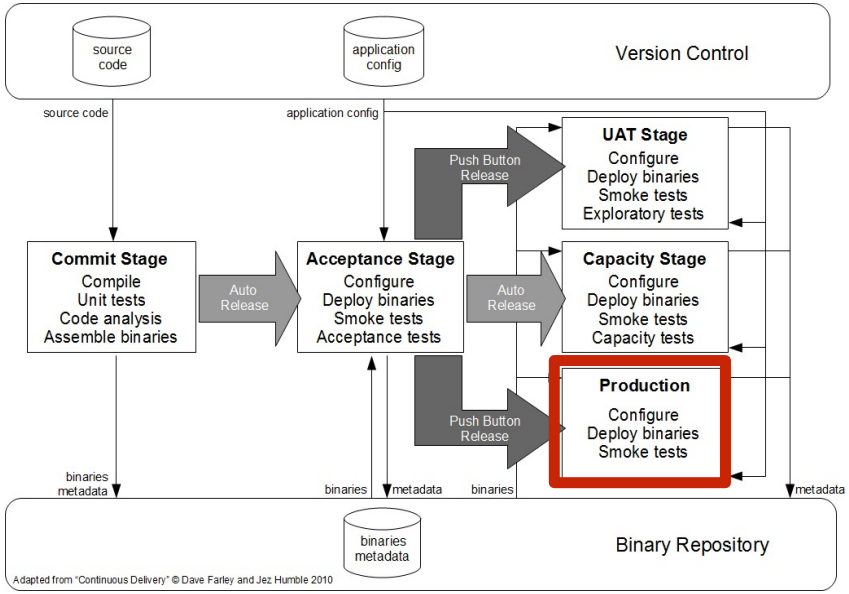
# CD Pipeline: Post CI Phase

- Consist of multiple environments/stages for testing purposes
- Include manual and automated testing
- Some stages might run in parallel (e.g., user acceptance testing (UAT) on UI and performance tests on backend systems)
- Stages are either automatically executed when previous stage is finished, or after manual approval (e.g., push of a button)



# CD Pipeline: CD Phase

- New version is available to be released to customers
- Various strategies and practices exist for rolling out:
  - Eat your own dog food
  - canary release
  - dark launches
  - gradual rollouts
  - A/B testing
  - blue/green deployments



# Eat your own dog food

- **Idea**: Employees use the newest version internally
- If everything is as expected it is roll it out to further users

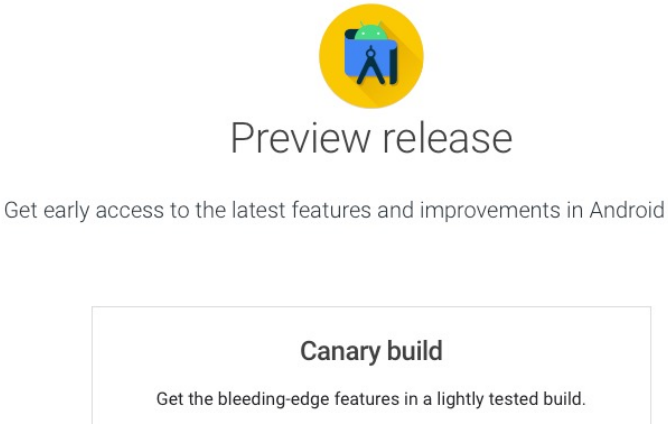
> *"Eating your own dog food, also called dogfooding, is a slang term used to reference a scenario in which a company uses its own product to test and promote the product."*
>
> *— Wikipedia*

# Canary Releases

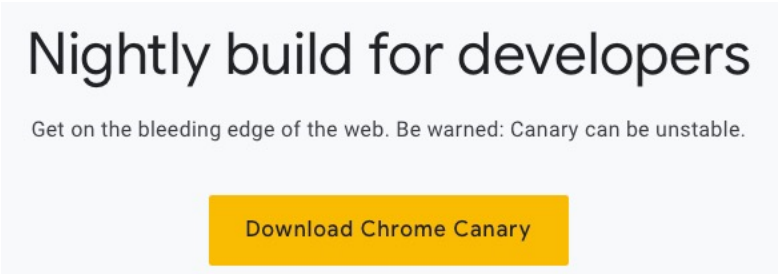
- **Idea**: release a new version to a subset of users first, while all other users interact with the old, stable version
- In case of issues with new version, only a small number of users is affected, thus the impact of the issue is kept small
- Canary is compared to the existing version in terms of a set of criteria such as stability, performance, or correctness
- User selection based on:
  - (geographic) location
  - role (admin, early, stable, etc.)
  - random basis (e.g., 1% of traffic)

# Canary Releases – examples

- https://developer.android.com/studio/preview/




- https://www.google.com/chrome/canary/



# Dark Launches

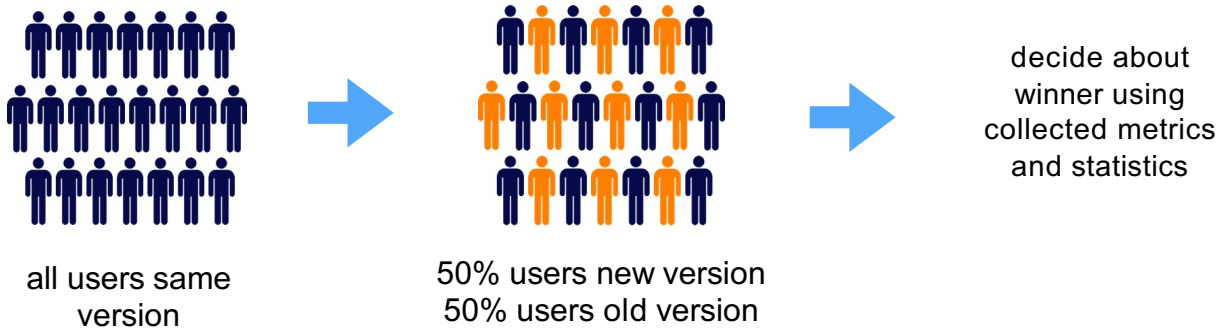
- **Idea**: mitigate performance and reliability issues of new features when facing production-like load levels by deploying those features on production without being visible for users
- Dark launches are commonly released to a group of users that doesn't know they're being tested on and don't have the new feature pointed out to them in any way
- Through monitoring, dev teams can identify and fix remaining bugs and scalability concerns before enabling the feature for users

# Gradual Rollouts

- **Idea**: increase the number of users assigned to the newest version in a stepwise manner until it completely replaces the former version
- Shows whether the new version can cope with increasing load and scales correctly
- Often used in combination with canary releases and dark launches

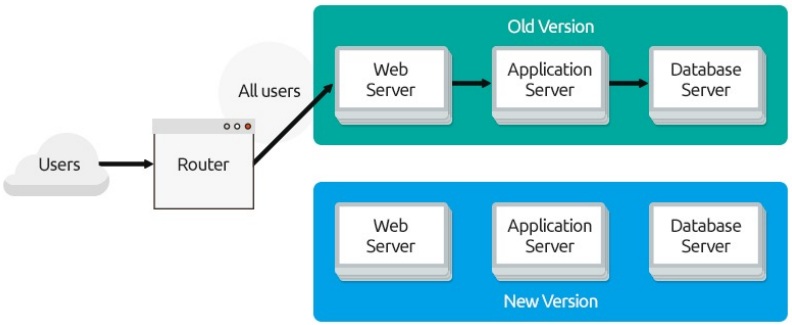
# A/B Testing

- **Idea**: compare two versions of software with each other, often only differentiated in one tested aspect, to determine the effect of a certain change
  - form of statistical hypothesis testing, thus requires big enough sample size to have "statistical power"
  - mainly used on UI for testing various layouts or design aspects



# Blue/Green Deployments

- **Idea**: have two identical production environments, one hosting (green) the current, stable version, the other (blue) represents the new version.
  - Release: simply switch the router so that all incoming traffic goes to the blue (or green) environment and use the green (or blue) environment for testing the next version.
  - Advantage: in case of problems directly after the release, a quick rollback (switch) to the previous version is possible
  - Obstacle: DBs are part of both environments, thus a switch requires DB migration/ synchronization



# Benefits of DevOps

- Speed
  - **Move at high velocity** so you can innovate for customers faster, adapt to changing markets better, and grow more efficient at driving business results.
  - For example, continuous delivery let teams take ownership of services and then release updates to them quicker.
- Reliability
  - Ensure the **quality of application updates** and infrastructure changes so you can reliably deliver at a more rapid pace while maintaining a positive experience for end users.
  - Use practices like continuous integration and continuous delivery to test that each change is functional and safe.

# Benefits of DevOps

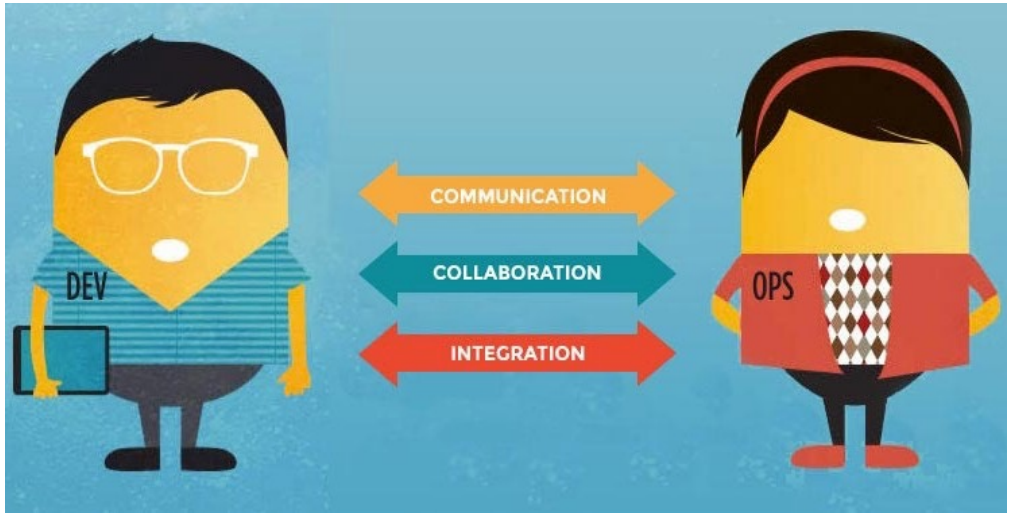
- Scale
  - Automation and consistency **help you manage complex or changing systems** efficiently and with reduced risk.
  - For example, infrastructure as code helps you manage your development, testing, and production environments in a repeatable and more efficient manner.
- Improved collaboration
  - **Developers and operations teams collaborate** closely, share many responsibilities, and combine their workflows.
  - This reduces inefficiencies and saves time.
    - writing code that considers the environment in which it is run.
    - reduced handover periods between developers and operations.

# Benefits of DevOps

- Security
  - Move quickly while retaining control and preserving compliance.
  - One can adopt a DevOps model without sacrificing security by using **automated compliance policies**, fine-grained controls, and configuration management techniques.
  - For example, using infrastructure as code (and policy as code), you can define and then track compliance at scale.

# DevOps – CALMS

- DevOps is not purely technical, includes several aspects summarized under CALMS:
  - Culture
  - Automation
  - Lean
  - Measurement
  - Sharing



# DevOps – CALMS

| | |
|---|---|
| **Culture** | Teams over individuals<br>Cross-functional teams instead of "silos"<br>Embrace change & experimentation |
| **Automation** | Continuous Delivery / Deployment<br>Infrastructure as Code |
| **Lean** | Be minimalistic (meeting numbers and times, team sizes, …)<br>Focus on producing value for the end user |
| **Measurement** | Collect data on everything<br>Ensure to provide visibility into all systems and events (e.g., dashboards) |
| **Sharing** | Collaboration & communication instead of "throwing things over the fence"<br>Not just reporting facts, regular exchange of ideas |

# Summary

- DevOps Technical benefits:
  - Continuous software delivery
  - Less complex problems to fix
  - Faster resolution of problems

- DevOps Business benefits:
  - Faster delivery of features (time to market)
  - More stable operating environments
  - More time available to add value (rather than fix/maintain)