

ANÁLISE DE SISTEMAS

# Integração contínua & Entrega contínua (CI/CD)

Ilídio Oliveira

v2022/06/14

# Objetivos de aprendizagem

Identificar os passos de um ciclo de CI

Distinguir entre *C. Integration*, *C. Deployment* e *C. Delivery*

Relacionar o CI/CD com a natureza iterativa e incremental dos métodos ágeis de desenvolvimento

Explicar as tarefas englobadas numa “build”

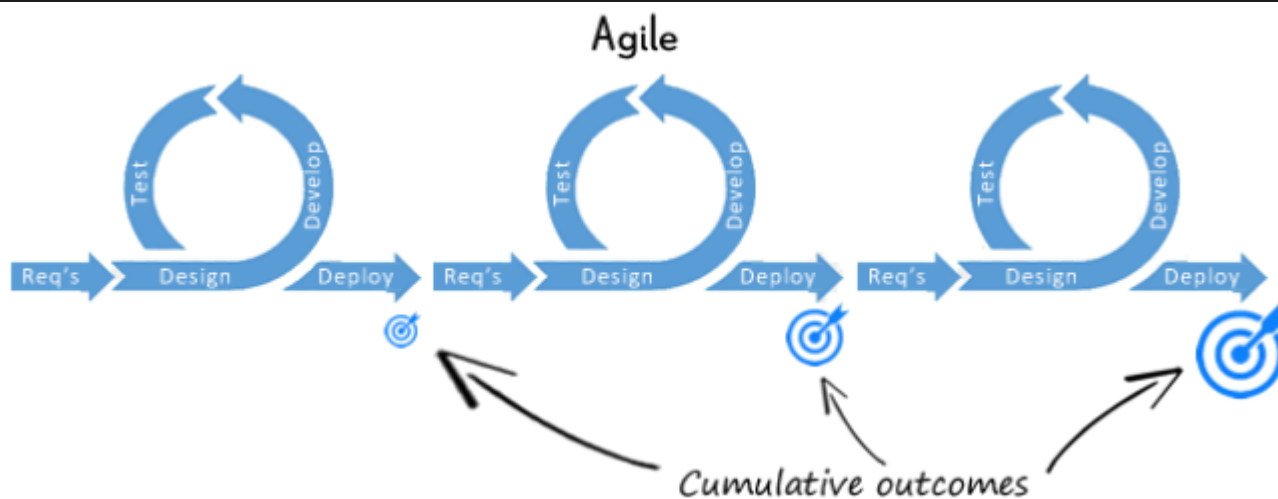
Explicar as principais boas práticas na implementação de CI numa equipa

Relacionar o CI/CD e DevOps.

# Desenvolvimento iterativo

Cada iteração envolve escolher um pequeno subconjunto dos requisitos, para projetar/desenhar, implementar e testar.

- Desenvolvimento em ciclos curtos
- Cada ciclo dá um incremento executável (parcial)
- Cada incremento é testado e integrado
- O feedback de cada iteração leva ao refinamento e adaptação da próxima.



# Continuously integrating the “units”

The essence of it lies in the simple practice of everyone on the team integrating frequently.

Feel comfortable and set up the tools to integrate at any time.

CI makes the development process smoother and less risky

↓ “it runs on my computer”

Early detection of failures (react quickly)

spot errors earlier  
shared code ownership  
everybody is co-responsible

big, unpredictable effort to integrate  
app state is not executable most of the time



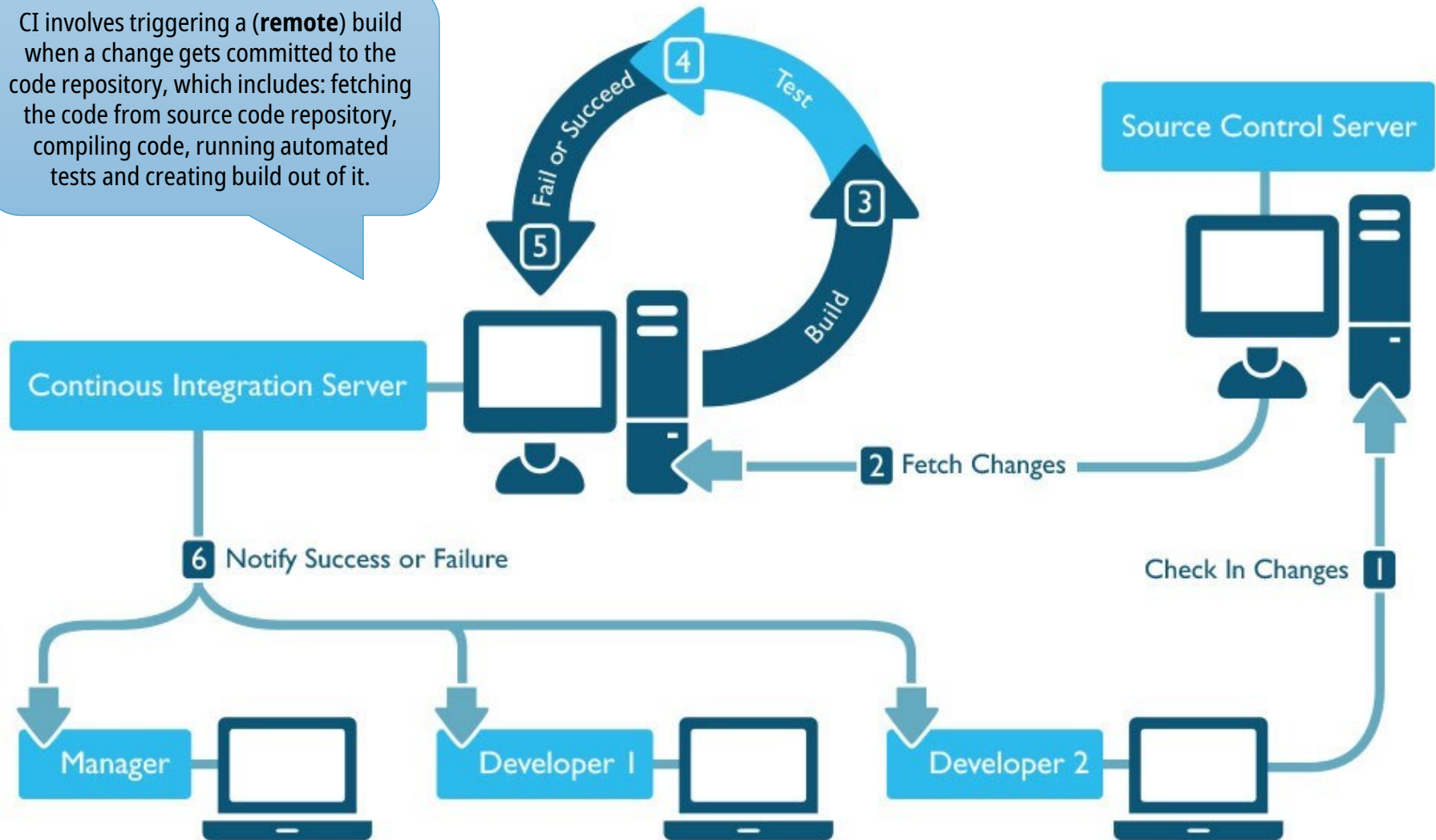
integrate early and often

Integration hell

「\ (ツ) /」

**IT WORKS**  
*on my machine*

CI involves triggering a (**remote**) build when a change gets committed to the code repository, which includes: fetching the code from source code repository, compiling code, running automated tests and creating build out of it.



<https://insights.sei.cmu.edu/blog/continuous-integration-in-devops/>

# Suggested development workflow

1- Update from shared SCM

2- Code a new feature (tests + code) in a dedicate feature branch

3- Run automated build on local machine

Repeat #2 and #3 till tests pass

4- Commit (integrate with "central")

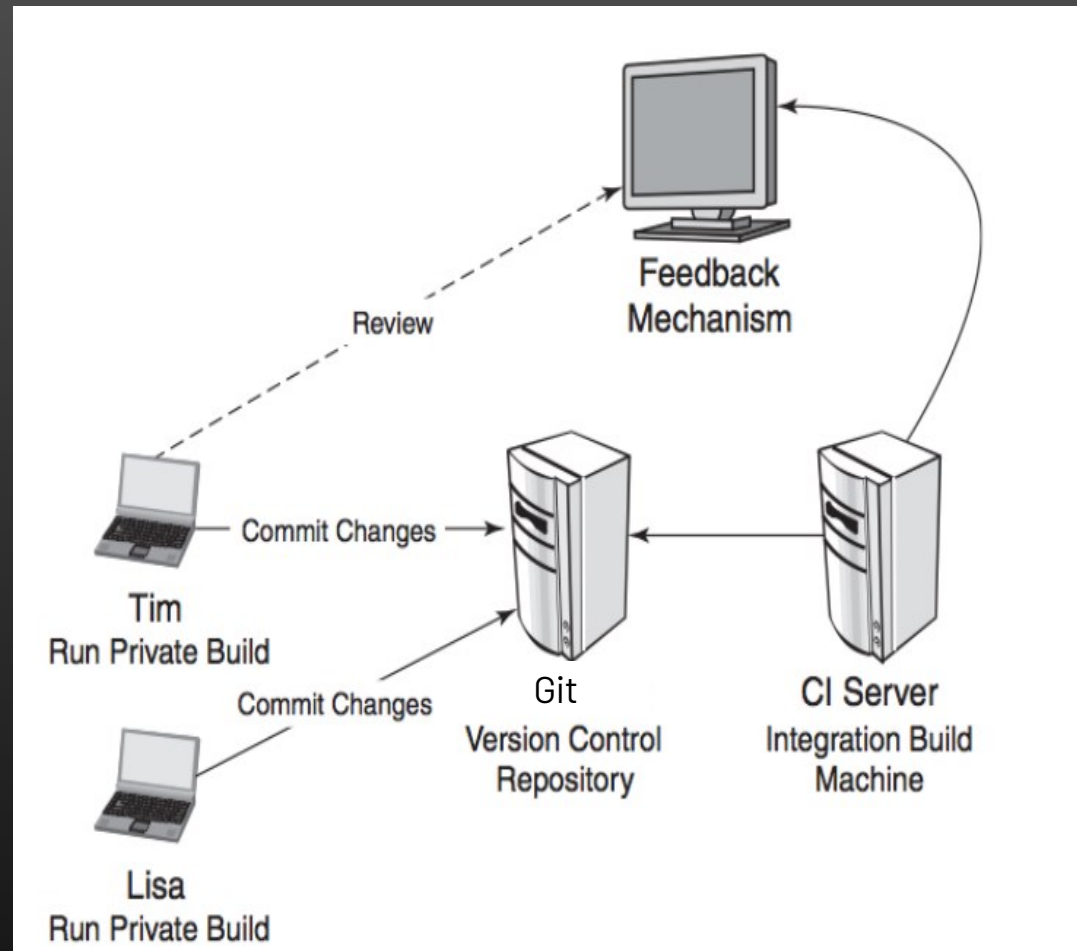
Pull requests advised (with peer code review)

5- Run a build on a clean machine

Update artifacts, update build status (feedback)

Immediately fix bugs and integration issues

**Not only tools: CI culture required!**



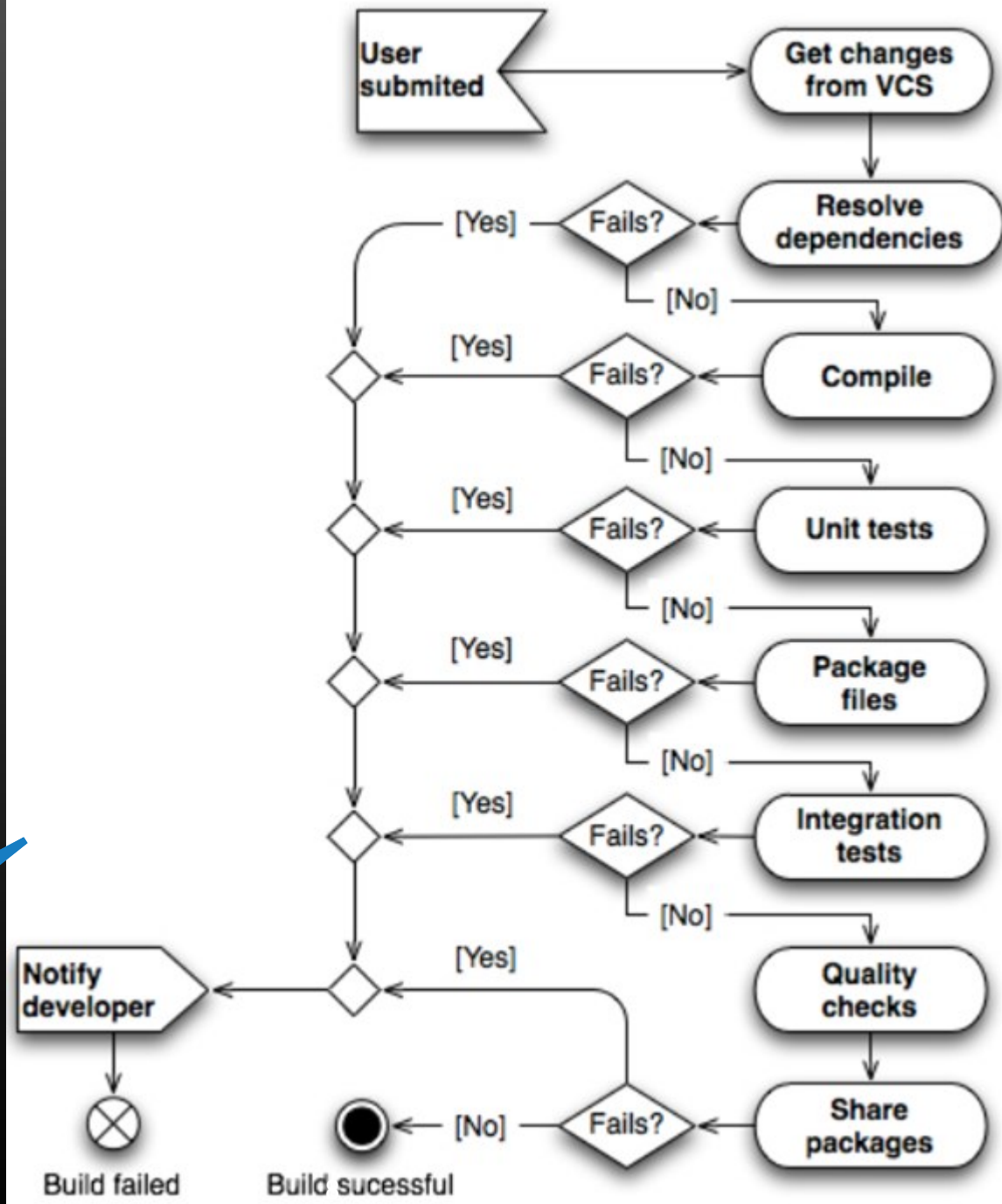
# The build process

A build has several stages.

A successful build implies success in code correctness and quality checks.

Automatic build tools run quality checks (e.g.: unit testing, code inspections)

Not just compiling...





# Quality checks → análise estática do código

## Analisar a qualidade do Código sem o executar

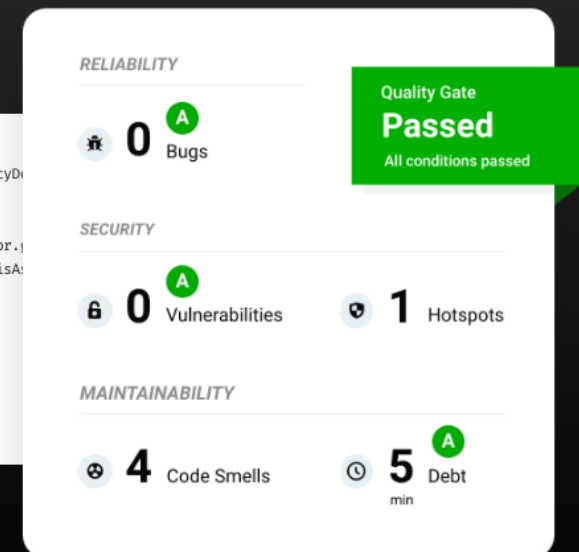
- vulnerabilidades conhecidas
- potenciais erros (e.g.: *null pointers*,...)
- construções desaconselhadas

<https://www.sonarqube.org/>

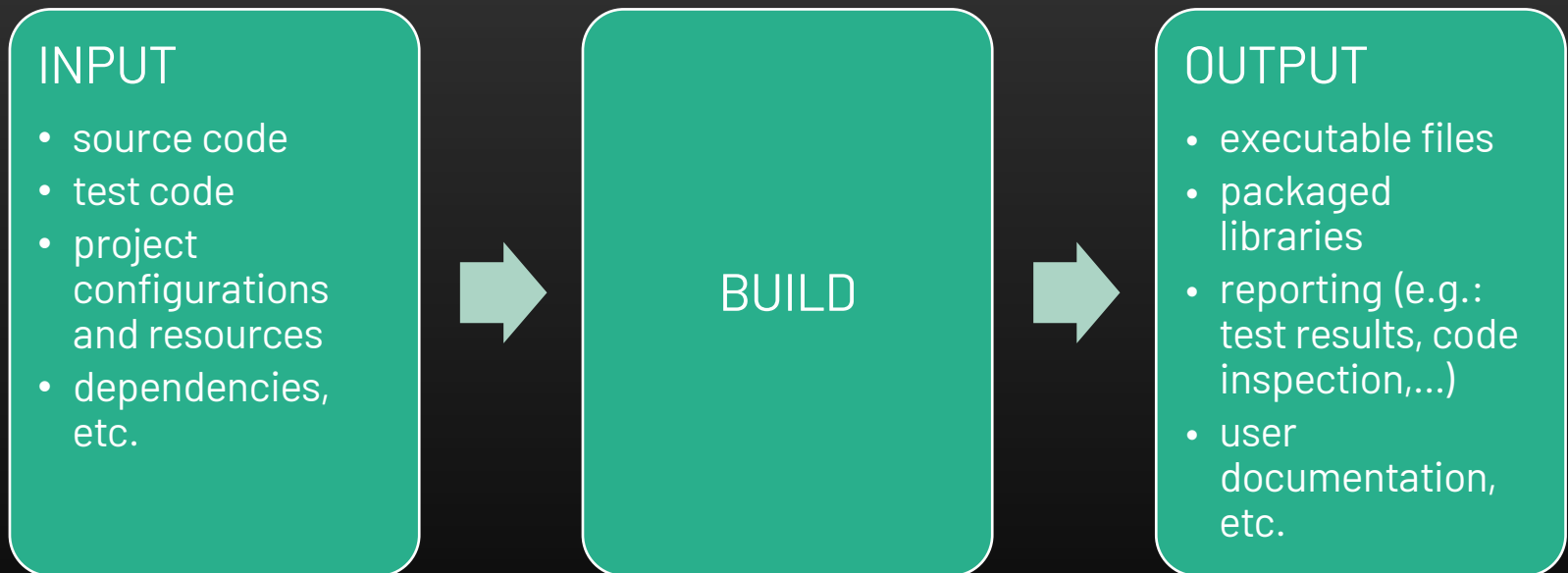
```
246 if (Provider.class == roleTypeClass) {
247     Type providedType = ReflectionUtils.getLastTypeGenericArgument(dependencyD
248     2 Class providedClass = 1 ReflectionUtils.getTypeClass(providedType);
249
250     if (this.componentManager.hasComponent(providedType, dependencyDescriptor.)
251         || 3 providedClass.isAssignableFrom(List.class) || providedClass.isA

A "NullPointerException" could be thrown; "providedClass" is nullable here.
Bug Major cert, cwe

252     continue;
253 }
```



# A *build*é mais que compilar...



# Práticas da integração contínua

- Os *developers* gravam para o repositório partilhado com regularidade (*commit*)
- As alterações no SCM são monitorizadas e acionam automaticamente o ciclo de CI
- Feedback imediato sobre falhas nas *builds* (as *builds* falhadas têm alta prioridade)
- Opcional: instalação de artefactos num repositório de referência
- Opcional: acionamento de testes de integração/aceitação

Quanto mais frequente for o processo de integração, menos penoso será

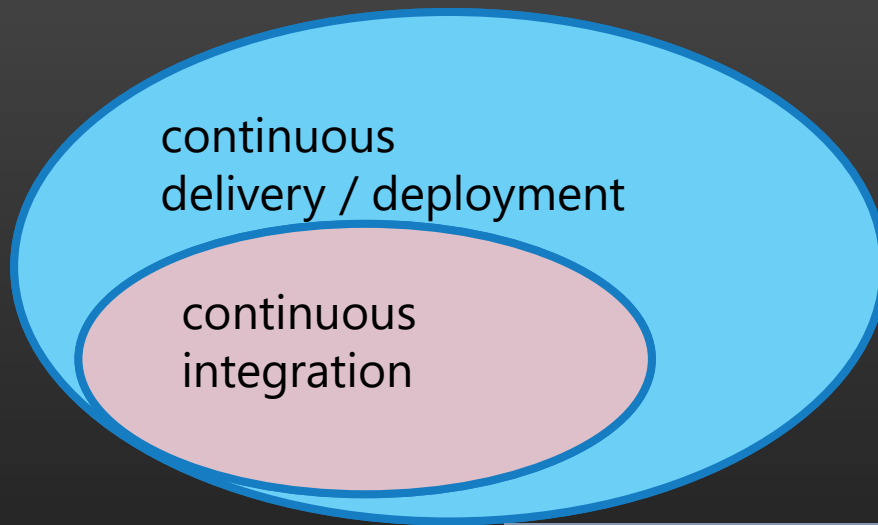
“Continuous” → *mindset* e ferramentas preparados para a integração a qualquer altura

# Fowler's 10 CI practices

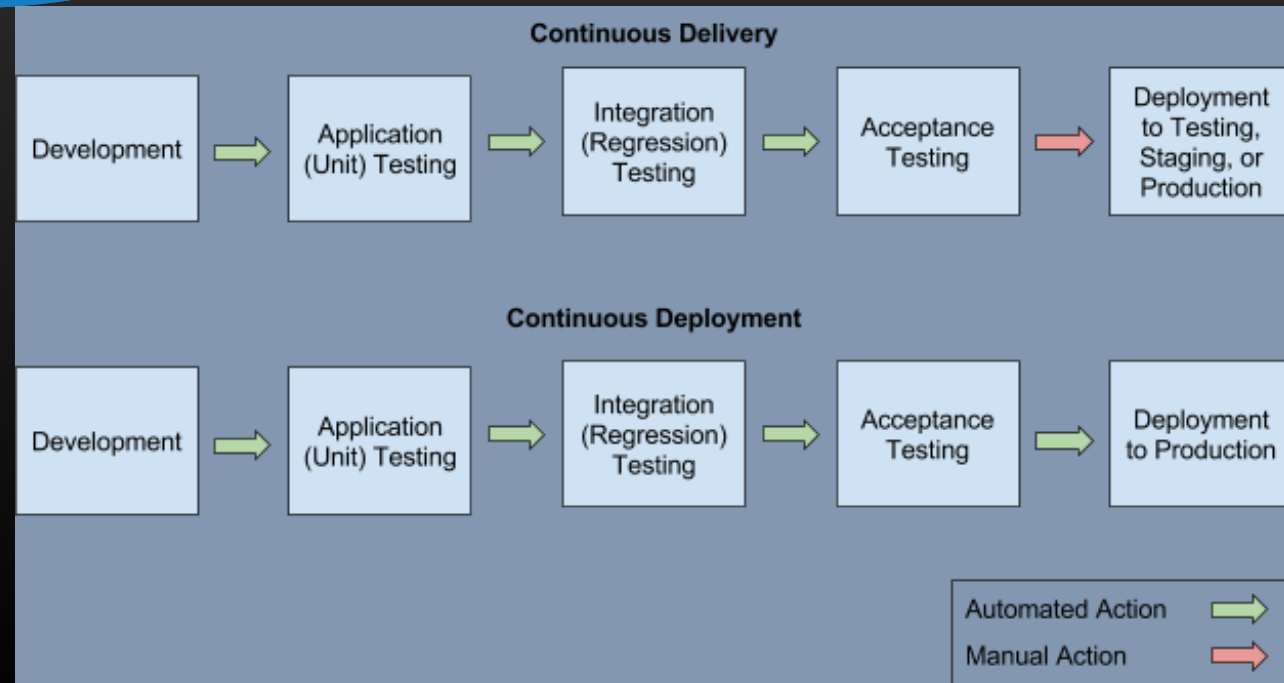
- Maintain a Single Source Repository.
- Automate the Build
- Make Your Build Self-Testing
- Everyone Commits To the Mainline Every Day
- Every Commit Should Build the Mainline on an Integration Machine
- Keep the Build Fast
- Test in a Clone of the Production Environment
- Make it Easy for Anyone to Get the Latest Executable
- Everyone can see what's happening
- Automate Deployment

<http://martinfowler.com/articles/continuousIntegration.html>

# Termos relacionados



É possível fazer instalações frequentes, mas pode-se optar por não o fazer (geralmente relacionadas com a estratégia empresarial)



# Continuous...

## Continuous Delivery

sw development practice in which you build software in such a way that it **can be released** to production at any time.

You're doing continuous delivery when:

Focus on quality of working software

Your software is deployable throughout its lifecycle

Your **team prioritizes keeping the software deployable over working on new features**

Anybody can get fast, automated feedback on the production readiness

## Continuous Deployment/release

every change goes through the pipeline and **automatically gets put into production.**

Focus on speed and agility to deploy to production

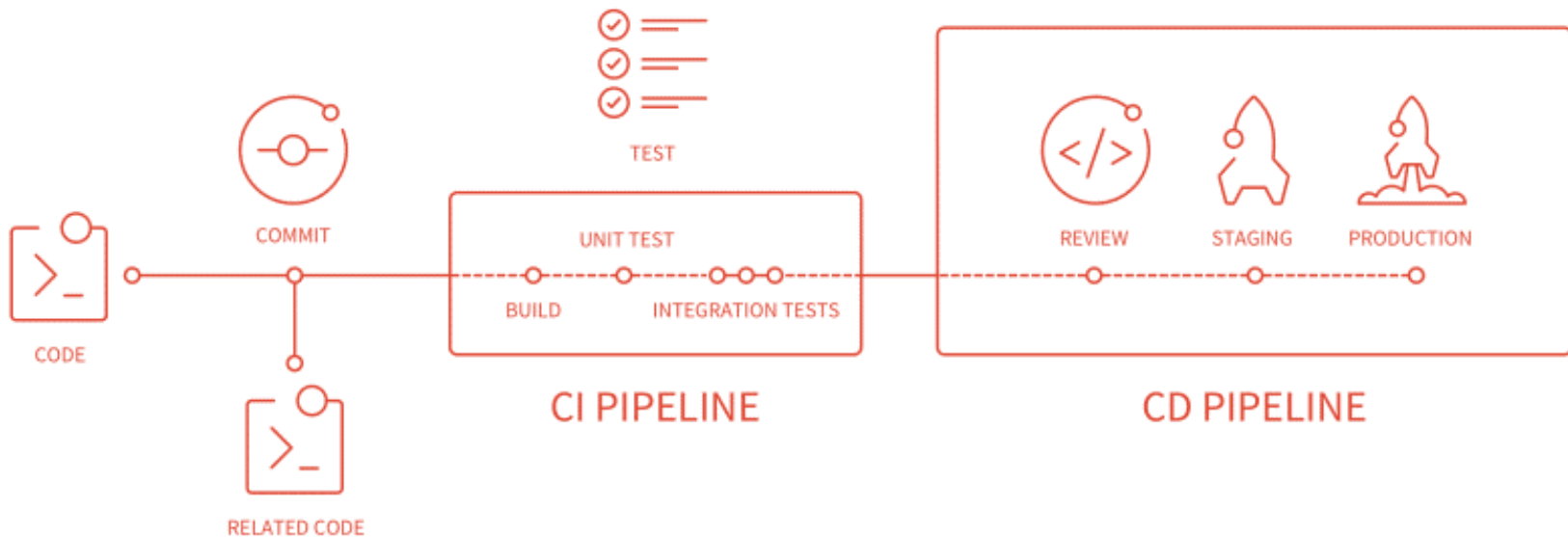
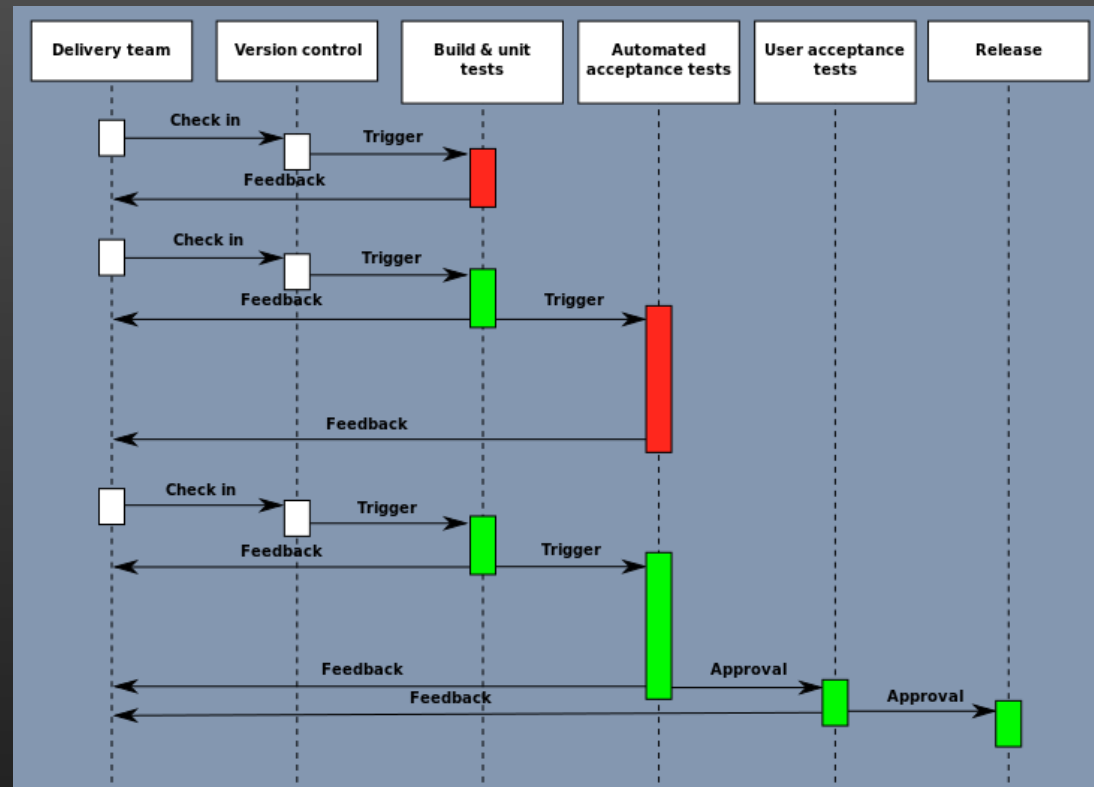
## Continuous Integration

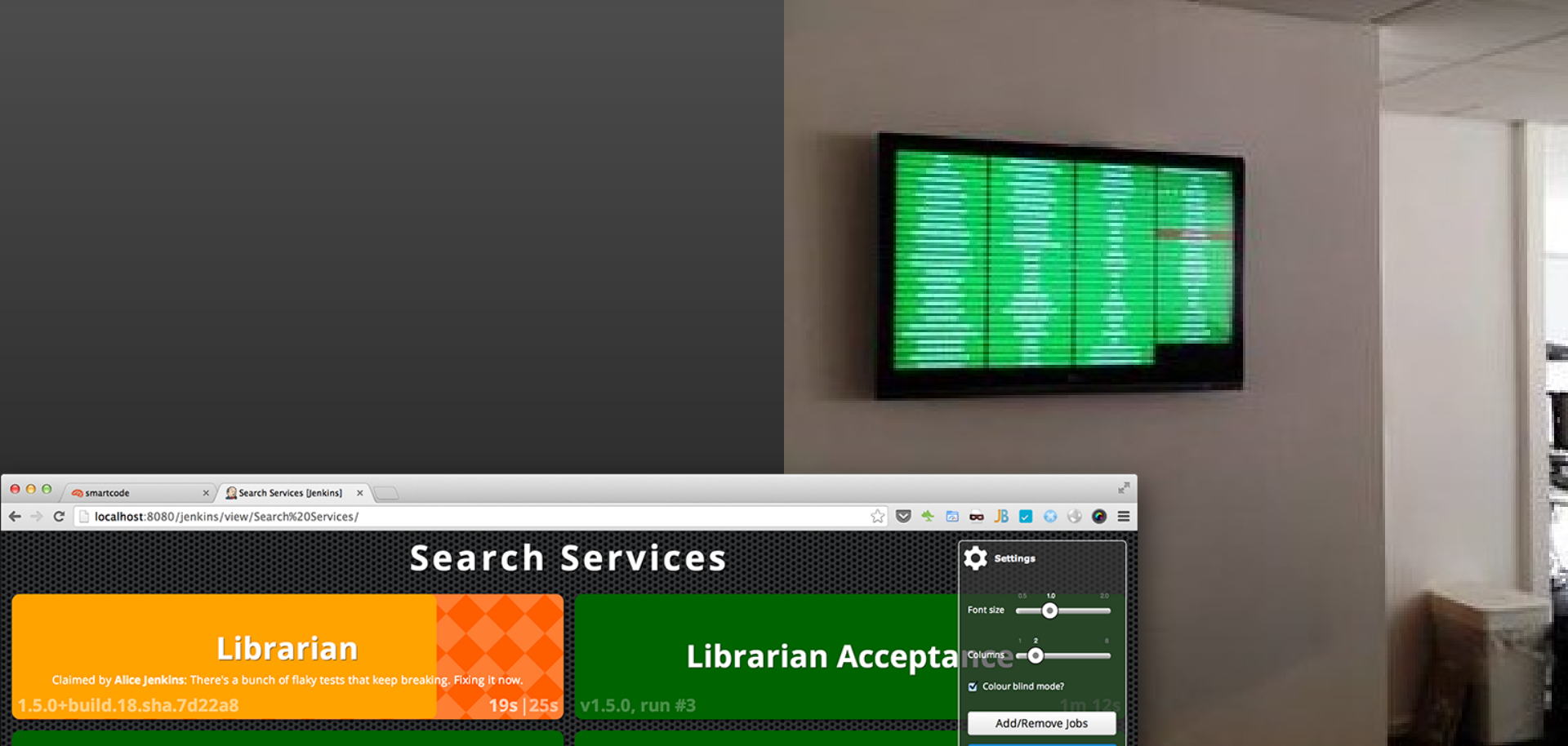
Automatically integrating, building, and testing code within the development **environment.**

Pre-delivery steps.

# Entrega continua

<https://about.gitlab.com>





smartcode

Search Services [Jenkins]

localhost:8080/jenkins/view/Search%20Services/

# Search Services

Librarian

Claimed by Alice Jenkins: There's a bunch of flaky tests that keep breaking. Fixing it now.

1.5.0+build.18.sha.7d22a8

19s | 25s

Performance Benchmark

#1

1m 13s

Promote to UAT

Identified Problems: Invalid credentials

#12

22s | 2m 7s

Librarian Acceptance

v1.5.0, run #3

1m 13s

Promote to PROD

#2

1m 2s

Search API Contract Tests (PROD)

#1

17s

Search API Contract Tests (UAT)

#2

37s

Settings

Font Size 0.5 1.0 2.0

Columns 1 2 3

☒ Colour blind mode?

Add/Remove Jobs

Done

Brought to you by Jan Molak



# Continuous feedback

**Errors are easier to detect in an earlier stage, near the point where they have been introduced:**

The detection mechanism of such bugs becomes simpler because the natural step in diagnosing the problem is to check what was the latest submitted change.

problems followed by atomic commits are easiest to correct than to fix several problems at once, after bulk commits

**There must be an effective mechanism that automatically informs programmers, testers, database administrators and managers about the status of the build**

**Feedback → generate reaction in a more accurate and prompter way**



# Continuous testing

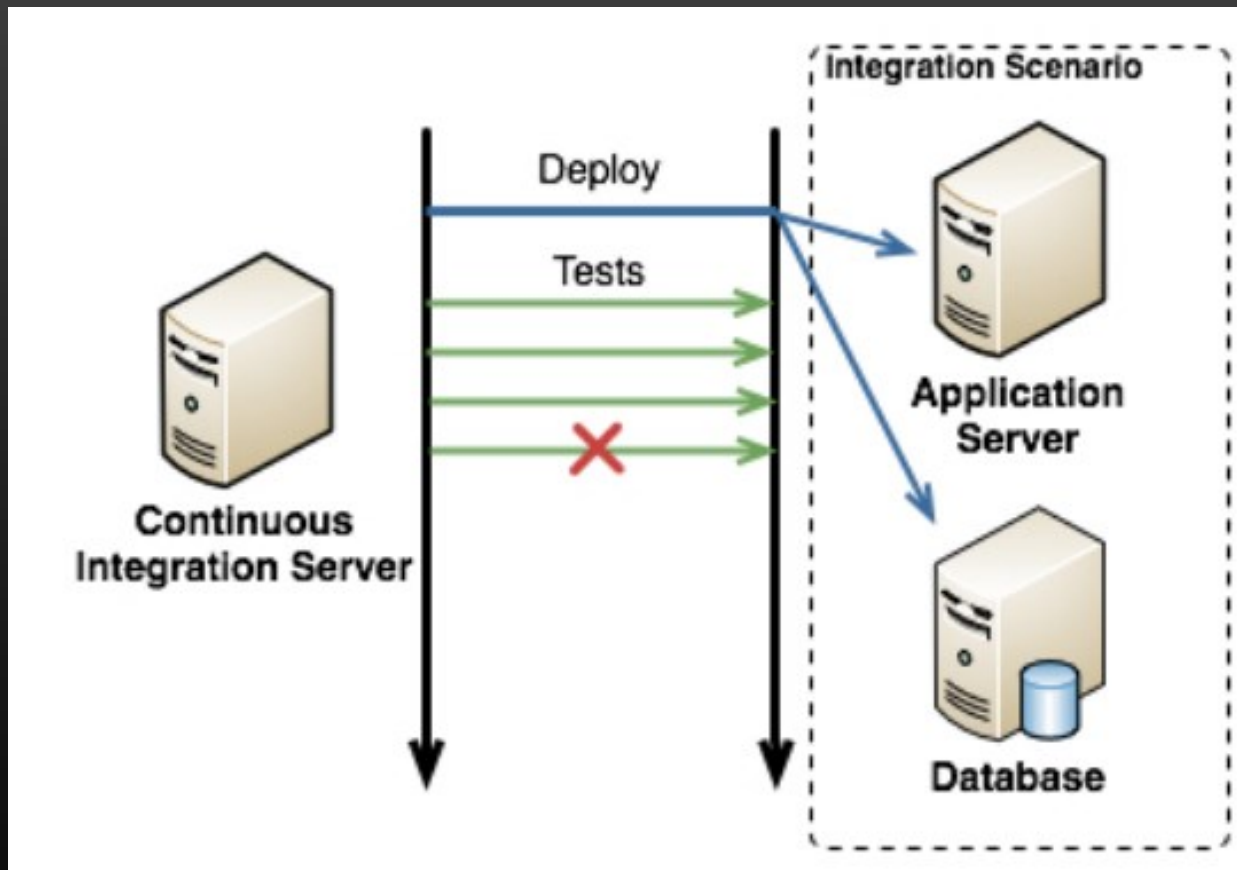
Quality checks at all system levels  
and involve all individuals, not just the  
elements of the QA team

Most of the tests can be automated  
and should be run in the CI pipeline to  
be carried out repeatedly:

unit testing, integration testing,  
regression testing, system testing, load  
and performance testing, etc.

Build tools can take a crucial role on  
automating tests

# Integration tests



# Jenkins



**Easy of use and extremely extensible**

Plugins-oriented

**Hosted**

vs cloud-centric

**Distributed builds**

Master/slaves architecture

**Jenkins vocabulary**

Job: a runnable task

Node: a master or slave machine

Build Executor: a stream of builds to run

Plugin: module that extends the core functionality.

Pipeline: definition of the steps to be executed

Jenkins supports building Java projects since its inception, and for a reason! It's both the language Jenkins is written in, plus the language in use by many if not all the projects Kohsuke Kawaguchi wanted to watch out when he created the tool many years ago.

If you want to build a Java project, there are a bunch of different options. The most typical ones nowadays are generally Apache Maven, or Gradle.

## Apache Maven

In any FreeStyle job, as **currently** Maven is supported in standard, you can use the dedicated step. One advantage is, as for all Jenkins tools, that you can select a specific Maven version and have Jenkins automatically install it on the build node it's going to run on.

image::/images/solution-images/jenkins-maven-step.png

## Gradle

As the associated plugin is not installed by default, first install the [Gradle plugin](#). Once done, you should be able to add a Gradle step.

image::/images/solution-images/jenkins-gradle-step.png

## Java plugins for Jenkins



### JUnit plugin

publishes JUnit XML formatted test reports for trending and analysis



### Gradle plugin

support invoking Gradle as a build step and listing executing tasks per build



### Findbugs plugin

generate trending and analysis for FindBugs reports



### PMD plugin

generate trending and analysis for PMD reports



### Cobertura plugin

publish and trend code coverage reports from Cobertura



### SonarQube plugin

integrate reporting from the SonarQube code quality/inspection platform



### Repository Connector plugin

adds features for resolving artifacts from a Maven repository such as Nexus or Artifactory.

→ <https://jenkins.io/solutions/java/>

# Pipeline as Code with Jenkins



The default interaction model with Jenkins, historically, has been very web UI driven, requiring users to manually create jobs, then manually fill in the details through a web browser. This requires additional effort to create and manage jobs to test and build multiple projects, it also keeps the configuration of a job to build/test/deploy separate from the actual code being built/tested/deployed. This prevents users from applying their existing CI/CD best practices to the job configurations themselves.

## Jenkins ♥ Continuous Delivery Articles

[Multibranch Workflows in Jenkins](#)  
[jenkins-ci.org](#)

## Continuous Delivery

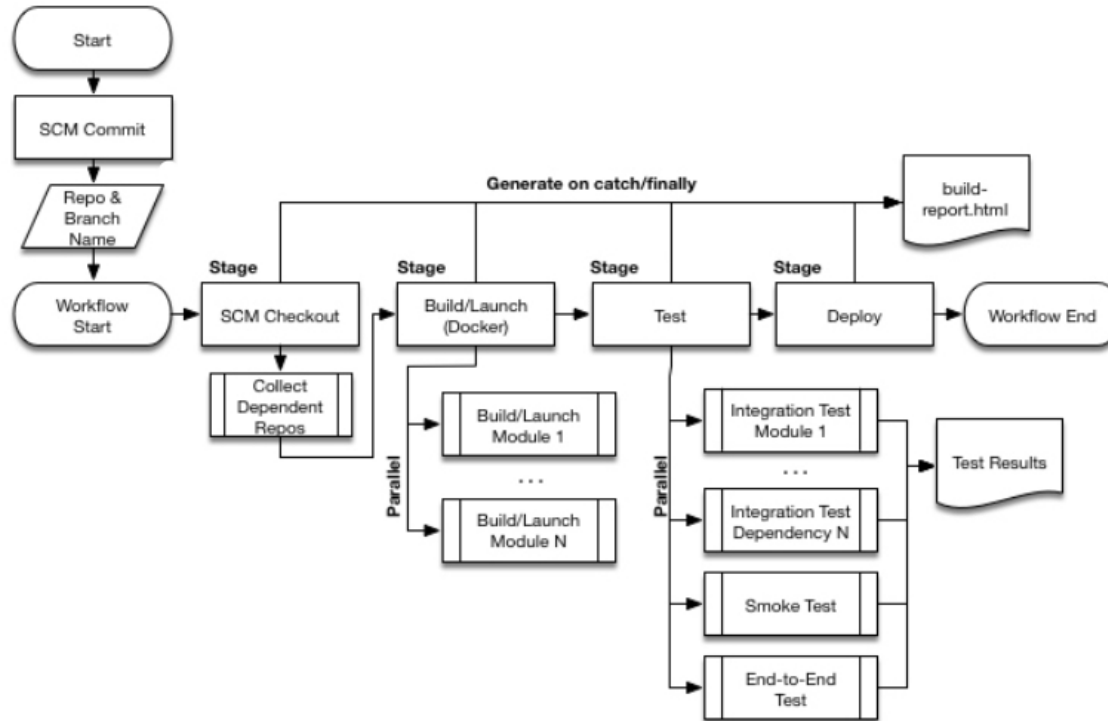
*A continuous delivery (CD) pipeline is an automated expression of your process for getting software from version control right through to your users and customers. Every change to your software (committed in source control) goes through a complex process on its way to being released.*

Pipeline provides an extensible set of tools for modeling simple-to-complex delivery pipelines "as code" via the Pipeline domain-specific language (DSL) syntax.

## Pipeline

With the introduction of the Pipeline plugin, you can define a build/test/deploy pipeline in a `Jenkinsfile` and store that as another piece of code checked into source control.

# Jenkins pipelines



```

pipeline {
    agent {
        docker {
            image 'maven:3-alpine'
            args '-v /root/.m2:/root/.m2'
        }
    }
    options {
        skipStagesAfterUnstable()
    }
    stages {
        stage('Build') {
            steps {
                sh 'mvn -B -DskipTests clean package'
            }
        }
        stage('Test') {
            steps {
                sh 'mvn test'
            }
            post {
                always {
                    junit 'target/surefire-reports/*.xml'
                }
            }
        }
        stage('Deliver') { ❶
            steps {
                sh './jenkins/scripts/deliver.sh' ❷
            }
        }
    }
}

```

```

pipeline {
    agent any
    stages{
        stage('Build'){
            steps {
                sh 'mvn clean package'
            }
            post {
                success {
                    echo 'Now Archiving...'
                    archiveArtifacts artifacts: '**/target/*.war'
                }
            }
        }
        stage ('Deploy to Staging'){
            steps {
                build job: 'Deploy-to-staging'
            }
        }
        stage ('Deploy to Production'){
            steps{
                timeout(time:5, unit:'DAYS'){
                    input message:'Approve PRODUCTION Deployment?'
                }

                build job: 'Deploy-to-Prod'
            }
            post {
                success {
                    echo 'Code deployed to Production.'
                }
                failure {
                    echo ' Deployment failed.'
                }
            }
        }
    }
}

```



# GitHub Actions

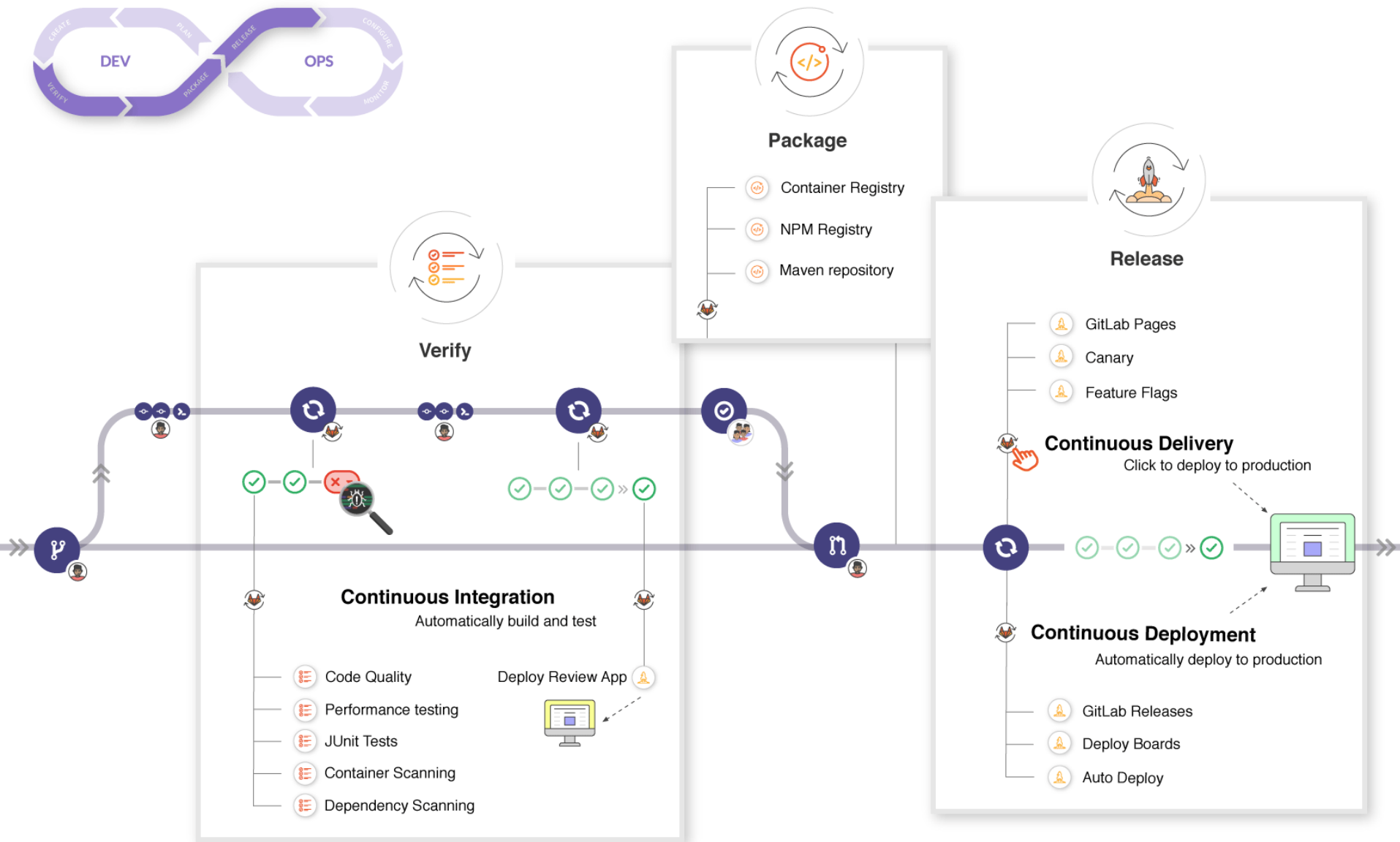
```
name: Node CI

on: [push]

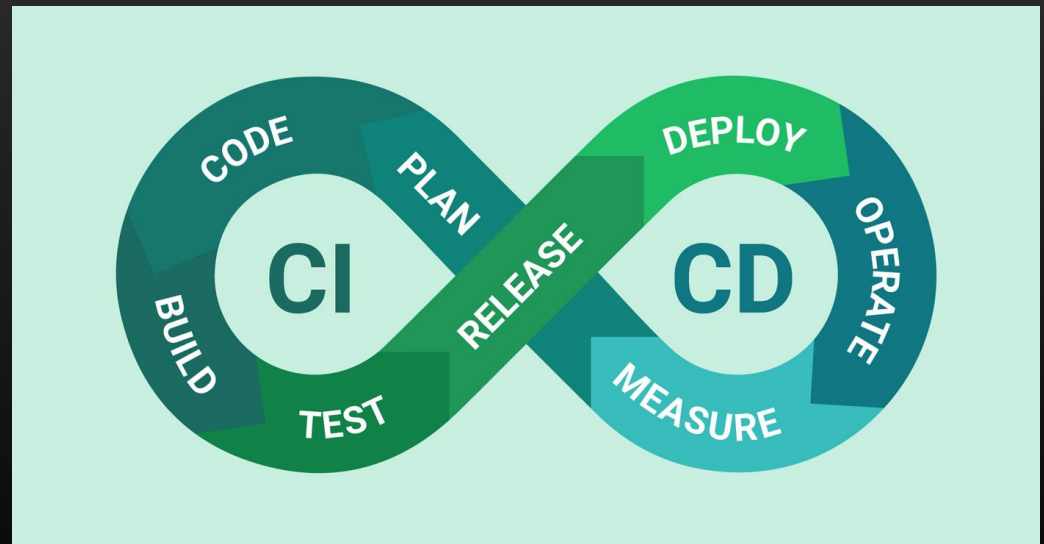
jobs:
  build_and_test:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout repository
        uses: actions/checkout@v3
      - name: npm install, build, and test
        run: |
          npm install
          npm run build --if-present
          npm test
      - name: Archive production artifacts
        uses: actions/upload-artifact@v3
        with:
          name: dist-without-markdown
          path: |
            dist
            !dist/**/*.md
      - name: Archive code coverage results
        uses: actions/upload-artifact@v3
        with:
          name: code-coverage-report
          path: output/test/code-coverage.html
```

# GitLab CI/CD

<https://docs.gitlab.com/ee/ci/introduction/>  
→ Includes video demos



**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, communication and collaboration, and technology automation.



<https://www.atlassian.com/devops/what-is-devops>



# More to explore

## Books on Continuous integration:

Duvall's Continuous Integration: <http://www.amazon.com/Continuous-Integration-Improving-Software-Reducing/dp/0321336380>

Humble's "Continuous Delivery": <http://www.amazon.com/Continuous-Delivery-Deployment-Automation-Addison-Wesley/dp/0321601912>

## Hudson/Jenkins

Extensive information:

<http://www.youtube.com/watch?v=6k0S402PnTc#!>

## Maven:

Free ebook: <http://www.sonatype.com/books/mvnref-book/reference/public-book.html>