Introduction to Continuous Integration with Jenkins

ALM CI/CD/CD

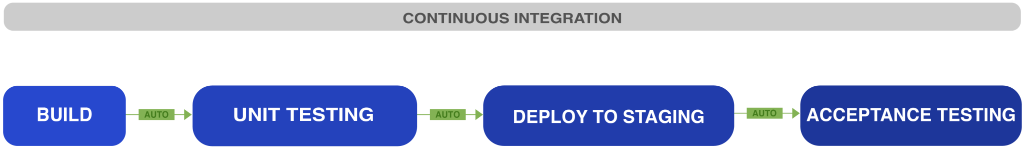
What is ALM?

Aligned to our corporate **COM** methodology, Application Lifecycle Management (ALM) covers the entire lifecycle from project strategy through design, construction, and testing to product deployment or delivery.

It supports **Agile** and **DevOps** development approaches by integrating these disciplines and enabling teams to collaborate and organize themselves more effectively.

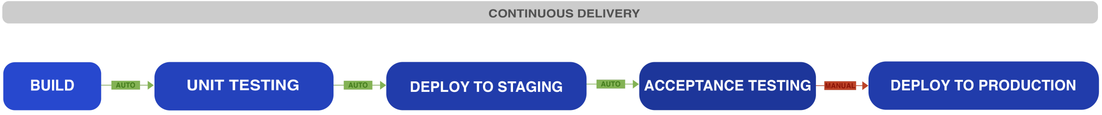
Continuos integration

A working model that consists of performing **compilations** and running **tests** on a project to check correct integration, and **detect bugs as soon as possible**.



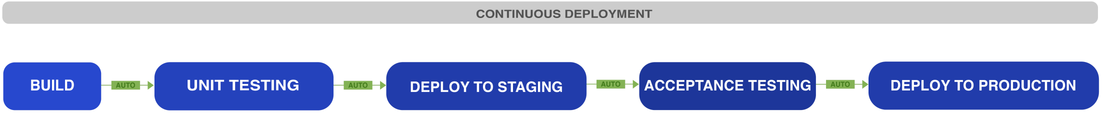
Continuous delivery

Every time a change is made to the application source code, a **deliverable is generated ready** to promote to Production. **The move to Production is manual**.

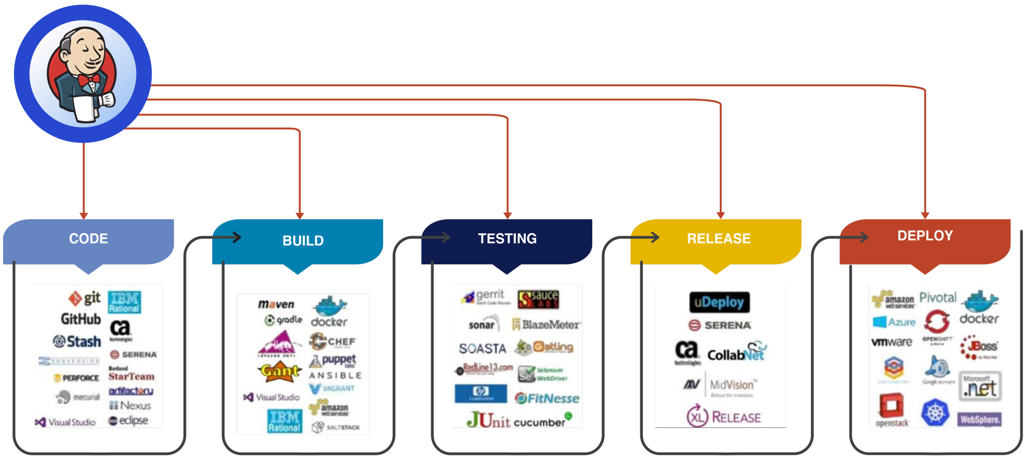


Continuous deployment

Every time a change is made to the application source code, it is automatically promoted through the environments until it is **promoted to Production. The move to Production is automatic from the development stage**.



What is Jenkins?



- **Jenkins** is a **Continuous Integration** engine that performs **automated** operational and flow verification of a software project tailored to its environment.

- **Jenkins** can be extended by plugins, which provide additional functionality to those originally supported by the Jenkins core.
- **Jenkins** can be installed via native system packages, Docker images, or even run independently on any machine with Java installed (JRE).

What nomenclature do you need to get started?

Jobs

Put simply, any automated process implemented in **Jenkins** is a **Jenkins** job. For example:

- Downloading programs from code repositories.
- Build and test the software.
- Monitor the execution and outcome of the *jobs* executed.

Credential Manager

This allows you to centrally store and use credentials to access all the repositories and applications we want to use.

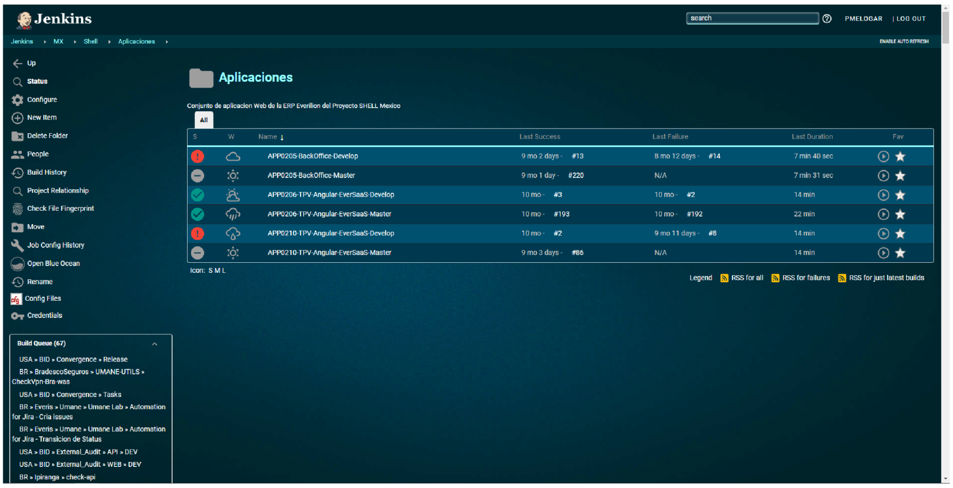
Nodes

These are the **Jenkins** executors. Nodes run the *jobs* and each node is a custom environment that contains configuration files and specific versions of the build and deployment tools.

Dashboard

The **Jenkins** dashboard displays a summary of the latest job executions.

What does the Dashboard look like?



Conjunto de aplicacion Web de la ERP Everilion del Proyecto SHELL Mexico

S	W	Name ↓	Last Success	Last Failure	Last Duration	Fav
❗	☁	APP0205-BackOffice-Develop	9 mo 2 days - #13	8 mo 12 days - #14	7 min 40 sec	★
⊖	☀	APP0205-BackOffice-Master	9 mo 1 day - #220	N/A	7 min 31 sec	★
✓	⛅	APP0206-TPV-Angular-EverSaaS-Develop	10 mo - #3	10 mo - #2	14 min	★
✓	🌧	APP0206-TPV-Angular-EverSaaS-Master	10 mo - #193	10 mo - #192	22 min	★
❗	🌧	APP0210-TPV-Angular-EverSaaS-Develop	10 mo - #2	9 mo 11 days - #8	14 min	★
⊖	☀	APP0210-TPV-Angular-EverSaaS-Master	9 mo 3 days - #86	N/A	14 min	★

Build Queue (67)

- USA » BID » Convergence » Release
- BR » BradescoSeguros » UMANE-UTILS » CheckVpn-Bra-was
- USA » BID » Convergence » Tasks
- BR » Everis » Umane » Umane Lab » Automation for Jira - Cria issues
- BR » Everis » Umane » Umane Lab » Automation for Jira - Transicion de Status
- USA » BID » External_Audit » API » DEV
- USA » BID » External_Audit » WEB » DEV
- BR » Ipiranga » check-api

What is a pipeline?

Pipeline concept not only applies in the **DevOps** context, it is used in many areas ranging from Marketing to computing.

In this module we will talk specifically about ***pipelines*** in **Jenkins.** Before we begin, we will look at some of the meanings of the word ***pipeline***.

- **In computing:** A ***pipeline*** is a set of elements of **serially connected data processing** where the output of one element is the input of the next.
- In CI/CD/CD: A ***pipeline*** is **an automated expression of the process** to bring the software from the SVC (*Source Control Version*) to the client or end user.
- In Jenkins: A ***pipeline*** is a set of *plugins* that provide support for modelling *pipelines* of varying complexity "***as Code***" through the pipeline *Domain Specific Language* (DSL) syntax , **defining them as code**.

What are the components of a in Jenkins?

Step

It corresponds to a single task that is part of the sequence and there are different types:

- *checkout*,
- *tools*,
- *sh*,

- *batch*,
- etc...

Node

They correspond to a workspace. A node may group many *"steps"*, which will run in the selected environment.

Stage

It is a way of grouping tasks together; it is the name of the stages in *the pipeline*.

Pipeline as Code (PAC)

Groovy

The language used to define ***pipelines*** as code is **Groovy** and we can use it to orchestrate **all the steps of the process**.

Scripting

Pros:

- Allows us to create more **complex** pipelines because any **Groovy code can be applied**.
- It has **imperative logic**.
- It offers great **flexibility and extensibility** of functions, enabling use of all the functions provided by Groovy.
- Ideal for **more advanced users** and **more complex requirements**.

Cons:

- It has **a steeper learning curve**.
- It is more **difficult to maintain** because it is **not easily readable**.
- **Does not admit** integration with Blue Ocean.
- The restart from *a Stage* **option is not available**.

Declarative

Pros:

- **Modern way** of developing Pipelines approved by **Cloudbees**.
- **It has a strict structure** with **simpler syntax**.
- **Easy to read** due to its structure.
- It has a **gentle learning curve**.
- **Integrates** with the Blue Ocean interface.
- **Allows us to restart** from a specific stage in the process.

Cons:

- Less suitable for ***pipelines*** **with more complex logic**.
- **Not compatible** with *older plugins* .
- More restrictive syntax.

Using the *Declarative* variant is a best practice.

Jenkins example

```
pipeline {
    agent any

    environment {
        MY_ENV_VAR = 'SomeValue'
    }

    stages {
        stage('Build') {
            steps {
                echo 'Building...'
                // Comandos para construir seu projeto, por exemplo:
                // sh 'make'
            }
        }

        stage('Test') {
            steps {
                echo 'Testing...'
                // Comandos para testar seu projeto, por exemplo:
                // sh './run-tests.sh'
            }
        }

        stage('Deploy') {
            when {
                branch 'main'
            }
            steps {
                echo 'Deploying...'
                // Comandos para implantar seu projeto, por exemplo:
                // sh './deploy.sh'
            }
        }
    }

    post {
        success {
            echo 'The pipeline completed successfully.'
        }
        failure {
            echo 'The pipeline failed.'
        }
    }
}
```

Explain

- `pipeline`: Define o início do script do pipeline.
- `agent`: Determina onde o pipeline irá rodar. `agent any` significa que o Jenkins pode executar o pipeline em qualquer agente disponível.
- `environment`: Permite definir variáveis de ambiente que serão usadas em todo o pipeline. `MY_ENV_VAR` é uma variável de exemplo.
- `stages`: Define as etapas principais do pipeline.

    - `stage('Build')`: Uma etapa chamada "Build", onde você colocaria comandos para construir seu projeto. O `echo` é apenas um comando de exemplo para imprimir uma mensagem no log.
    - `stage('Test')`: Uma etapa chamada "Test", destinada a executar testes em seu projeto. Aqui também, o `echo` serve para mostrar uma mensagem no log.
    - `stage('Deploy')`: Uma etapa chamada "Deploy", que contém comandos para implantar seu projeto. Esta etapa só será executada quando o pipeline for rodado na branch `main`, graças à condição `when`.
- `when`: Usado dentro de uma `stage` para especificar condições sob as quais a etapa deve ser executada. No exemplo, `Deploy` só ocorre quando a branch atual é `main`.
- `steps`: Dentro de cada `stage`, `steps` define os comandos específicos a serem executados.
- `post`: Define ações que são executadas ao final do pipeline, dependendo do resultado. No exemplo, há mensagens para sucesso e falha.
    - `success`: Bloco executado se o pipeline for concluído com sucesso.
    - `failure`: Bloco executado se o pipeline falhar em algum ponto.

Jenkinsfile

- Creation: Create a file named **Jenkinsfile** without extension, which will contain the necessary instructions for the creation of the ***Pipeline as Code***.
- File versioning: The **Jenkinsfile** must be in the root directory of the project in the **SCM** tool, for later management.
- Plugins: Remember that you must have the necessary *plugins* installed in **Jenkins** to connect to the SCM, to create ***Multibranch Pipeline*** *jobs* and to interpret *pipelines*.
- Job Creation: Finally, you must create *a Multibranch Pipeline* job **that points to the SCM source repository** so that **Jenkins** can run the process automatically later.

Ideal pipeline

What is the model of ?

In a CI/CD *pipeline*, you define all *stages* through which the code must pass until it is promoted, with corresponding steps to be executed.

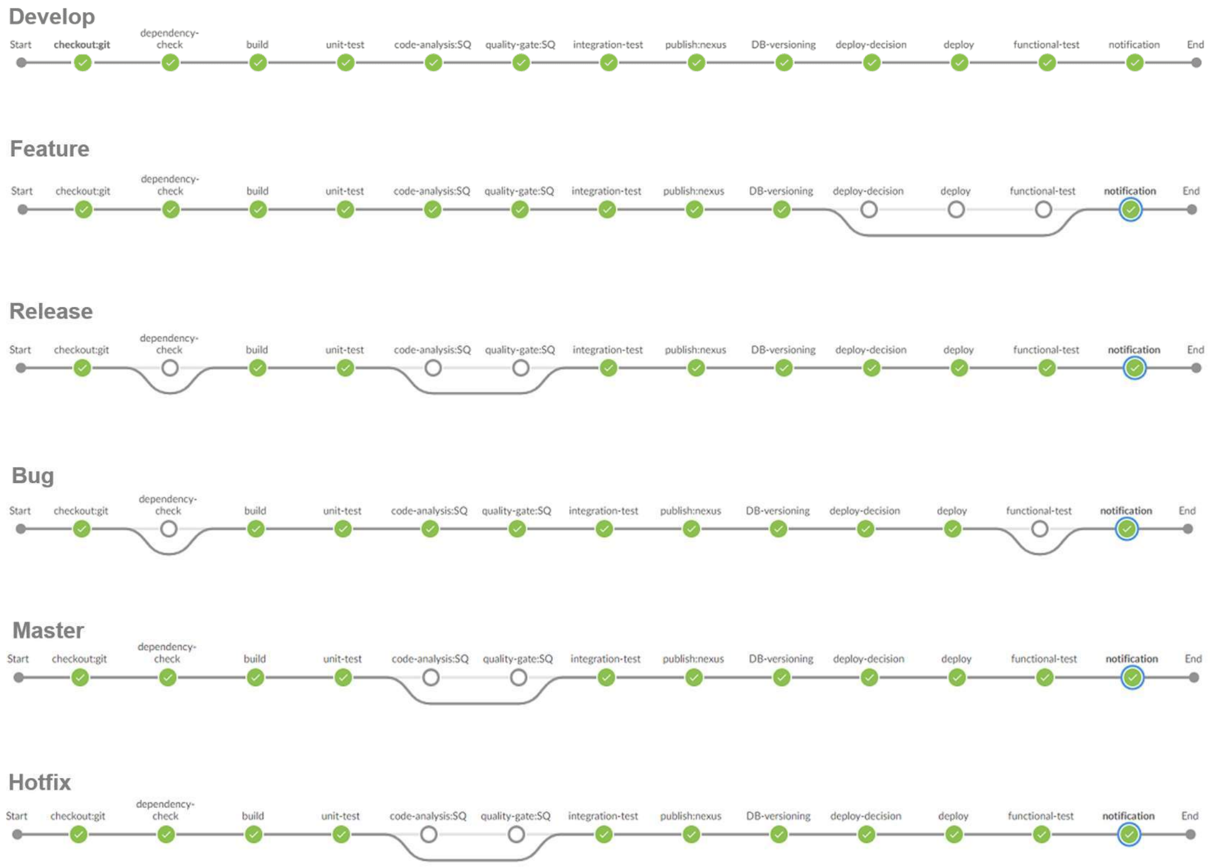
Depending on the *GitFlow* branch in question, we can define **different stages** of the ideal *pipeline*.

This **ideal *pipeline*** is linked to **changes in the code repository**, so that specific ***stages*** belonging to **the branch that has undergone the change** in code will be executed.

The *Multibranch pipeline* will scan the repository every so often, so when it finds a change in a branch, **Jenkins will start running the pipeline specific to that branch**.

Below are ideal *pipelines* according to the branch they serve.

Examples



Best practices

- ***Pipeline*** to automate **CI/CD processes.**
- ***Pipeline* as Code.**

- Use **declarative** syntax in the ***pipeline***.
- Use the *pipeline syntax* **wizard** provided by Jenkins.
- Execute steps on **a specific agent**.
- **Do not occupy** an agent at an interactive stage (*input*).
- Start with a **clean *workspace*.**
- Keep the *pipeline* (**Jenkinsfile**) in the source repository**, with code**.
- Use ***Multibranch Pipeline***.
- Link *pipeline* execution to **events** in the code repository.

Quizz

If there are no free "executors" in Jenkins:

- Jobs are automatically sorted into an execution queue, awaiting a performer to be released.

Possible job completion statuses are:

- **Success**
- **Unstable**
- **Failed**
- **Aborted**

Some of the benefits of a Continuous Integration cycle are:

- Early error detection
- Fewer repetitive manual processes
- Less time and risk

The stages defined in a *pipeline:*

- They are executed or not depending on certain conditions defined with the WHEN directive.

Which of the three definitions corresponds to "*Continuous Delivery*"?

- A model in which each time a change is made to the application source code, a deliverable is generated ready to promote to production

A *Pipeline* as Code can be defined in a plain text file, which will be named by default:

- Jenkinsfile

The goal of an ALM cycle is:

- Record and control the product-development and maintenance process

It is not possible to prevent and/or reduce production errors by applying ALM to our development

- False

In a continuous-integration process configured as Multibranch, it would make sense to send the static code-quality analysis of the branches...

- Develop
- Bug
- Release
- Feature
- Master
- Hotfix
