



# Continuous Delivery with Jenkins

DevOps Training

# Continuous Delivery

*Continuous delivery (CD) is a software engineering approach in which teams produce software in short cycles, ensuring that the software can be reliably released at any time. It aims at building, testing, and releasing software faster and more frequently. The approach helps reduce the cost, time, and risk of delivering changes by allowing for more incremental updates to applications in production. **A straightforward and repeatable deployment process is important for continuous delivery.***

# CI vs CD

- Continuous Integration usually refers to integrating, building, and testing code within the development environment. Continuous Delivery builds on this, dealing with the final stages required for production deployment
- Continuous Delivery is a software engineering approach in which teams produce software in short cycles, ensuring that the software can be reliably released at any time. It aims at building, testing, and releasing software faster and more frequently.
- Continuous Deployment means that every change goes through the pipeline and automatically gets put into production, resulting in many production deployments every day. In order to do Continuous Deployment you must be doing Continuous Delivery

# CI Vs CD

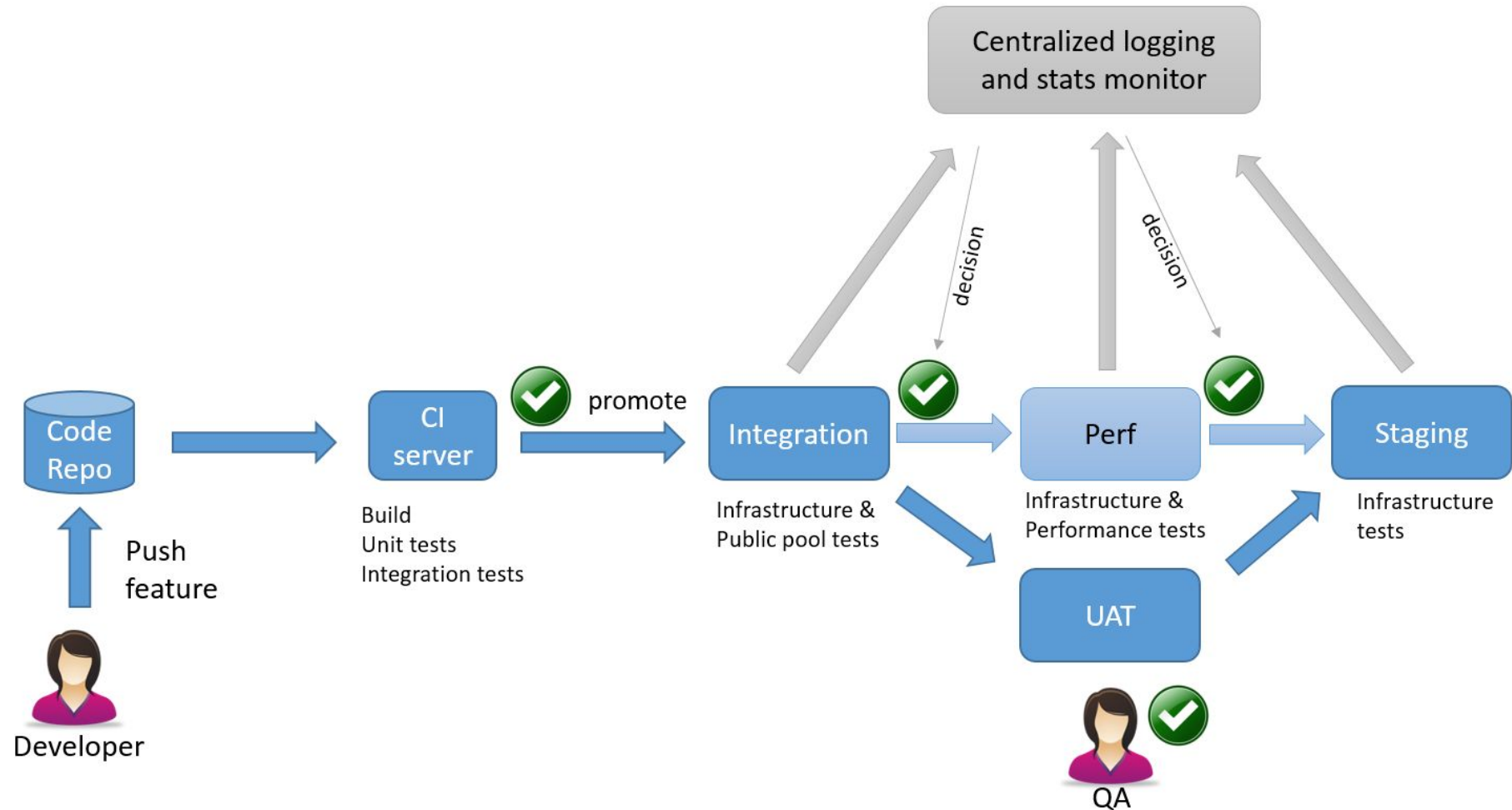
## Continuous delivery



## Continuous deployment



# How it Works



# CI/CD Tool Options



**Jenkins**



**Travis CI**



**circleci**



**GitLab**



**CODESHIP**



powered by

**amazon  
web services™**

Visual Studio Team Services



# Principles

## Continuous Integration

- Automate the build
- Make the build self-testing
- Everyone commits to the baseline every day
- Every commit should be built
- Keep the build fast
- Test in a clone of the PROD environment
- Make it easy to get the latest deliverables
- Everyone can see the results of the latest build
- Automate deployment

## Continuous Delivery

- The process for releasing/deploying software MUST be repeatable and reliable
- Automate everything
- If something is difficult or painful, do it more often
- Keep everything in source control
- Done means "released"
- Build quality in!
- Everybody has responsibility for the release process.
- Improve continuously.



# Best Practices

- **Continuous Integration**

- Don't check-in on a **Broken build**
- Always run all commit tests locally before committing, or get your CI server do it for you
- Wait for commit test to **Pass** before moving on. Never go home on a **Broken build**
- Time-box fixing before reverting (e.g. try to fix in 10 mins, if not then revert)
- User TDD / BDD

- **Continuous Delivery**

- Build the binaries only one
- Deploy the same way to every environment
- **Smock test** your deployment
- Use a container technology (**Docker**) if possible as makes deployment simple




# Jenkins 2.0


# Pipeline as Code


- Introduce “Pipeline” as a new type in Jenkins
- Codify an implicit series of stages into an explicit pipeline definition (`Jenkinsfile`) in source control
- Resumability/durability of pipeline state
- Extend the DSL to meet organizational needs


# Creating a Pipeline


**Enter an item name**  
  
» Required field

**Freestyle project**  
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

**Maven project**  
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

**Pipeline**  
Orchestrates long-running activities that can span multiple build slaves. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

**External Job**  
This type of job allows you to record the execution of a process run outside Jenkins, even on a remote machine. This is designed so that you can use Jenkins as a dashboard of your existing automation system.

**Multi-configuration project**  
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

# Creating a Pipeline

**Pipeline**

Definition Pipeline script ▼

Script

```
1 node {  
2   echo 'Hello World'  
3 }
```

Hello World ▼ ⓘ

☒ Use Groovy Sandbox ⓘ

[Pipeline Syntax](#)

# Jenkins File

## Jenkinsfile

```
node('java8') {  
    stage('Checkout') {  
        checkout scm  
    }  
  
    stage('Build') {  
        sh 'mvn clean install'  
    }  
  
    stage('Test') {  
        sh 'mvn test'  
    }  
  
    archiveArtifacts artifacts: 'target/**/*.jar', fingerprint: true  
}
```

# Snippet Generator

## Steps

Sample Step

stash: Stash some files to be used later in the build

Name

Includes

Excludes

Use Default Ant Excludes ☒

Generate Groovy

# Snippet Generator

## Steps

Sample Step

stage: Stage

git: Git

input: Wait for interactive input

isUnix: Checks if running on a Unix-like node

junit: Publish JUnit test result report

libraryResource: Load a resource file from a shared library

load: Evaluate a Groovy source file into the Pipeline script

mail: Mail

node: Allocate node

parallel: Execute in parallel

properties: Set job properties

publishBrakeman: Publish Brakeman reports

publishHTML: Publish HTML reports

pwd: Determine current directory

readFile: Read file from workspace

readTrusted: Read trusted file from SCM

retry: Retry the body up to N times

script: Run arbitrary Pipeline script

sh: Shell Script

sleep: Sleep

stage: Stage

Generate G



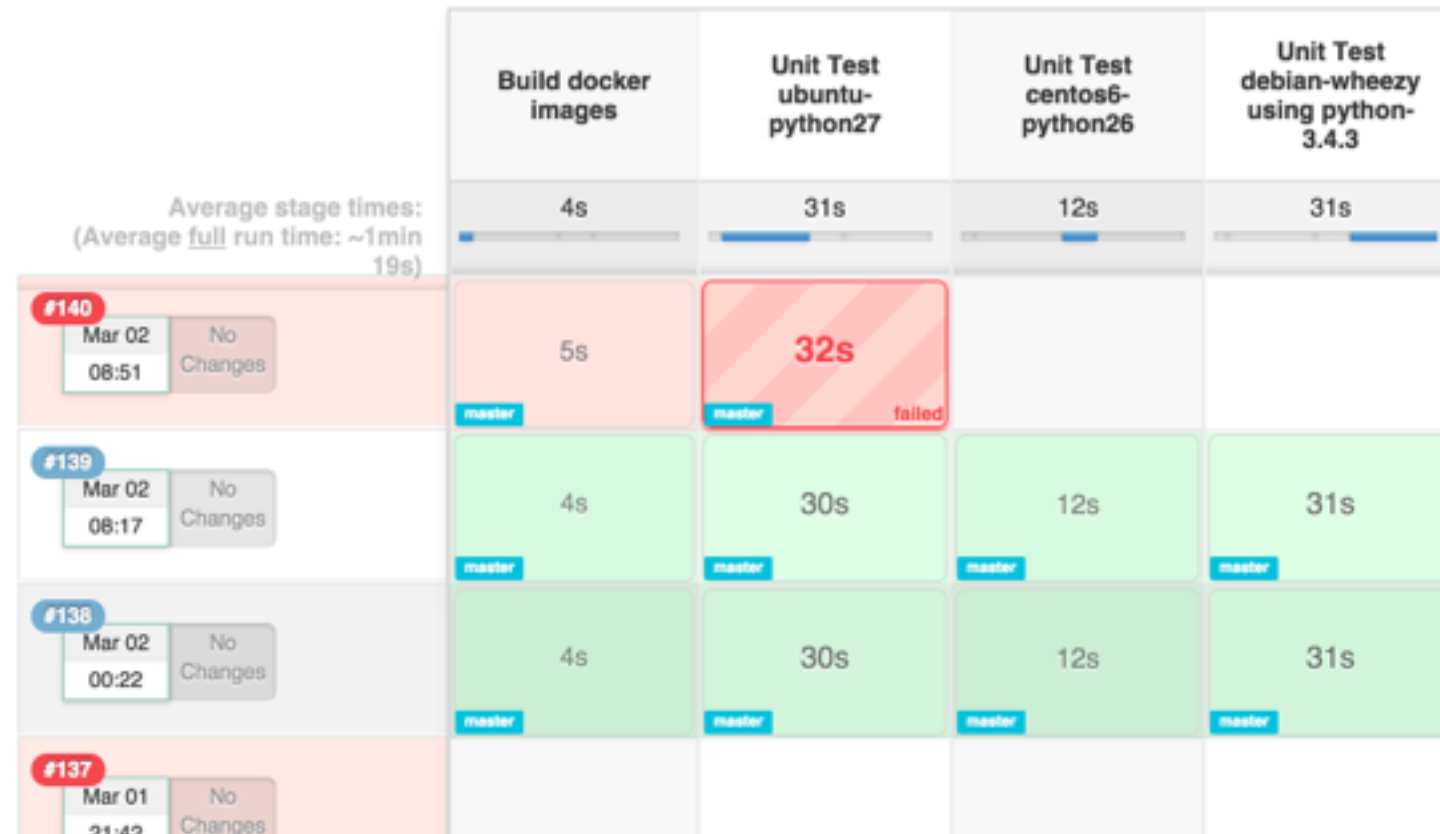
# Pipeline Plugins

## Pipeline Plugins

Extending Pipeline

# Pipeline Stage View plugin

## Stage View



# Pipeline Multibranch plugin

- Represent pipelines for multiple branches in one "Folder"
- Automatically scan a repository for each branch which contains a `Jenkinsfile`

# Blue Ocean

PipelinesAdministrationLogout

DashboardNew Pipeline

✕jenkins / infra-statistics

🔗master

🔗#4c2c3e9

↺▶★

🎯jenkins / jenkins-infra

🔗production

🔗#220ee2a

🎯★

✓jenkins / jenkins

🔗master

🔗#3f5fe7f

▶★

Name	Health	Branches	Pull Reque...
jenkins / Core / jenkins	🌟	13 failing	40 failing ★

# Plugins that support pipeline

- A few plugins which provide custom steps for Pipeline scripts:
  - Tooling: Credentials binding, SSH Agent, Parallel Test
  - Reporters: JUnit, Brakeman, HTML Publisher, Cucumber Test Result
  - Notifiers: Slack, HipChat, email-ext
  - Wrappers: Timestamper
- Plugins work within Pipelines
  - SCM: Git, SVN, Mercurial, P4, CVS
  - Wrappers: Ansi Color, NodeJS
  - Build steps: Ant, etc
- More defined in [COMPATIBILITY.md](#)

# Build Steps

## Steps

Sample Step

step: General Build Step

Build Step

Publish Javadoc

Javadoc directory

docs

Directory relative to the root of the workspace, such as 'myproject/build/javadoc'



Retain Javadoc for each successful build

Generate Groovy

```
step([$class: 'JavadocArchiver', javadocDir: 'docs', keepAll: true])
```

# Build Wrappers

## Steps

Sample Step

wrap: General Build Wrapper



Build Wrapper

Add timestamps to the Console Output



Generate Groovy

```
wrap([$class: TimestampWrapper]) {  
    // some block  
}
```





# Abstracting Pipeline

Sharing code, best practices and templates for success

# Shared Library

A shared library is a collection of independent Groovy scripts which you pull into your Jenkinsfile at runtime.

First you create your Groovy scripts (see below for details on this), and add them into your Git repository.

Then, you add your Shared Library into Jenkins from the Manage Jenkins screen.

Finally, you pull the Shared Library into your pipeline using this annotation (usually at the top of your Jenkinsfile):

```
@Library('your-library-name')
```

it's  
Q & A  
TIME!



THANK YOU!

[training@laksans.com](mailto:training@laksans.com)

@copyright of [www.cloudbearers.com](http://www.cloudbearers.com)