

Contenido

| | |
|---|----|
| Curso Básico de Jenkins..... | 2 |
| Introducción a Automatización | 2 |
| Jenkins Core | 2 |
| Introducción a Jenkins..... | 2 |
| Instalación y Configuración Básica de Jenkins | 3 |
| Manejo Básico de Usuarios | 4 |
| Notificaciones Email | 5 |
| Jobs (proyectos) | 5 |
| ¿Qué es un Job? | 5 |
| Información de un Job (proyecto)..... | 6 |
| Configuración de un Job (proyecto) | 8 |
| Plugins | 12 |
| Jenkins y su ecosistema de Plugins | 12 |
| Cadenas de Jobs | 12 |
| Conectándonos a GitHub | 14 |
| Pipelines | 15 |
| ¿Qué es un 'Pipeline'? | 15 |
| ¿Cómo puedo acelerar mi development de Pipelines? | 17 |
| Slave | 18 |
| Introducción a Slaves | 18 |
| Conectado un Slave | 19 |
| Notificaciones de escritorio | 19 |

Curso Básico de Jenkins

Introducción a Automatización

Automatizamos porque nos deja reproducir procesos y nos otorga mayor productividad, si los corremos manualmente podemos aburrirnos lo que nos lleva a saltarnos pasos y a generar errores. Por eso los automatizamos y podemos pasar más tiempo haciendo código, implementando nuevos features.

Podemos **automatizar**: correr pruebas, correr deployments, la verificación de la disponibilidad de nuestro sitio, podemos verificar cualquier cosa.

Apuntes:

Un término clave dentro de la *automatización* viene siendo **Integración continua (CI)**. Es el proceso de automatizar la compilación y la prueba de código cada vez que un miembro del equipo confirma cambios en el control de versiones. CI alienta a los desarrolladores a compartir su código y las pruebas unitarias fusionando sus cambios en un repositorio de control de versiones compartido después de cada pequeña tarea completada. La confirmación del código activa un sistema de compilación automatizado para obtener el código más reciente del repositorio compartido y para compilar, probar y validar la rama maestra completa (también conocida como troncal o principal).

Jenkins Core

Introducción a Jenkins

Jenkins es open source y probablemente el software de automatización más usado de todos, escrito en Java y corre en JVM. Es muy conveniente al ser una herramienta **extensible** al tener un ecosistema de **plugins** que te permiten extenderlo, puedes escribir tus propios **plugins** con Java, pero ya la comunidad ha desarrollado un sinfín de ellos.

También nos permite escalar de manera horizontal y verticalmente, puede correr un sin número de trabajos concurrentemente en una sola máquina y si esa máquina no da abasto se le puede dar más recursos a **Jenkins**. O una máquina no es suficiente entonces **Jenkins** nos permite escalar horizontalmente con “*slaves*” y controlar varios nodos para que trabajen por él.

Jenkins siempre está siendo innovado y teniendo actualizaciones de seguridad, esto es importante porque es el *target* más grande de seguridad de una empresa porque lo tiene todo.

Algo que **Jenkins** ha trabajado mucho en los últimos años es que puedes escribir tus “*jobs*” o unidades de trabajo en código. Nosotros queremos que nuestra automatización también sea programática, no solo los comando a ejecutar sino poder migrar nuestro trabajo a un nuevo **Jenkins** de manera reproducible. Han creado *Pipelines as code*

Apuntes:

- Web site: <https://www.jenkins.io>

Instalación y Configuración Básica de Jenkins

Instalación

- <https://www.jenkins.io/doc/book/installing/>
- <https://www.digitalocean.com/community/tutorials/how-to-install-jenkins-on-ubuntu-18-04#step-1---installing-jenkins>

Prerrequisitos

Prerequisites

Minimum hardware requirements:

- 256 MB of RAM
- 1 GB of drive space (although 10 GB is a recommended minimum if running Jenkins as a [Docker](#) container)

Recommended hardware configuration for a small team:

- 1 GB+ of RAM
- 50 GB+ of drive space

Comprehensive hardware recommendations:

- Hardware: see the [Hardware Recommendations](#) page

Software requirements:

- Java: see the [Java Requirements](#) page
- Web browser: see the [Web Browser Compatibility](#) page
- For Windows operating system: [Windows Support Policy](#)

Compatibilidad con Java

Running Jenkins

Modern Jenkins versions have the following Java requirements:

- Java 8 runtime environments, both 32-bit and 64-bit versions are supported
- Since Jenkins [2.164](#) and [2.164.1](#) ^[1], Java 11 runtime environments are supported
 - Running Jenkins with Java 11 is documented [here](#)
 - There are some precautions to take when upgrading from Java 8 to Java 11 in Jenkins, please follow [these guidelines](#).
- Older versions of Java are not supported
- Java 9 and Java 10 are not supported
- Java 12 is not supported

These requirements apply to all components of the Jenkins system including Jenkins master, all types of agents, CLI clients, and other components.

Jenkins project performs a full test flow with the following JDK/JREs:

- OpenJDK JDK / JRE 8 - 64 bits
- OpenJDK JDK / JRE 11 - 64 bits

JRE/JDKs from other vendors are supported and may be used. See [our Issue tracker](#) for known Java compatibility issues.

Variables globales

```
JENKINS_PORT=8080
JENKINS_PERM="--permanent"
JENKINS_SERV="$JENKINS_PERM --service=jenkins"

firewall-cmd $JENKINS_PERM --new-service=jenkins
firewall-cmd $JENKINS_SERV --set-short="Jenkins ports"
firewall-cmd $JENKINS_SERV --set-description="Jenkins port exceptions"
firewall-cmd $JENKINS_SERV --add-port=$JENKINS_PORT/tcp
firewall-cmd $JENKINS_PERM --add-service=jenkins
firewall-cmd --zone=public --add-service=http --permanent
firewall-cmd --reload
```

Apuntes:

- Se recomienda dejar configurado un dominio.

Instance Configuration

Jenkins URL:

<http://173.255.119.54:8080/>

The Jenkins URL is used to provide the root URL for absolute links to various Jenkins resources. That means this value is required for proper operation of many Jenkins features including email notifications, PR status updates, and the `BUILD_URL` environment variable provided to build steps.

The proposed default value shown is **not saved yet** and is generated from the current request, if possible. The best practice is to set this value to the URL that users are expected to use. This will avoid confusion when sharing or viewing links.

Manejo Básico de Usuarios

El acceso se realiza mediante el path `/securityRealm/` o desde las opciones *Manage Jenkins -> Manage Users*.

Jenkins > Jenkins' own user database enable auto refresh

Back to Dashboard

Manage Jenkins

Create User

Users

These users can log into Jenkins. This is a sub set of [this list](#), which also contains auto-created users who really just made some commits on some projects and have no direct Jenkins access.

| User ID | Name |
|----------------------------|--------------------------------|
| willtorber | William Torres |

Apuntes:





- Se pueden crear nuevos usuarios y asignarles diferentes permisos, esto con el fin de poder determinar en todo momento quien hizo que...
- Lo ideal es no compartir mismos usuarios ni misma contraseña.
- La autenticación se puede dar por medio de login con Github o Google, esto con el uso de plugins.
 - Github:

- <https://www.jenkins.io/solutions/github/>
 - <https://plugins.jenkins.io/github-oauth/>
- GitLab: <https://docs.gitlab.com/ee/integration/jenkins.html>
- Para crear, eliminar, editar un usuario: Ir a Manage Jenkins/ Manage Users/ Create user (en caso de crear). Para editar o borrar solo se debe dar clic en user id deseado, después elegir opción Configurar o Borrar

Notificaciones Email

En la sección *Manage Jenkins* -> *Configure System* -> *E-mail Notification* se configura el servidor SMTP y la cuenta de correo a implementar. Se recomienda usar STMP.GMAIL.COM.

E-mail Notification

| | | |
|---|---|---|
| SMTP server | <input type="text" value="smtp.gmail.com"/> | |
| Default user e-mail suffix | <input type="text"/> |  |
| <input checked="" type="checkbox"/> Use SMTP Authentication | |  |
| User Name | <input type="text" value="devtest@unisimonbolivar.edu.co"/> | |
| Password | <input type="password" value="....."/> | |
| Use SSL | <input checked="" type="checkbox"/> |  |
| Use TLS | <input type="checkbox"/> | |
| SMTP Port | <input type="text" value="465"/> |  |
| Reply-To Address | <input type="text"/> | |
| Charset | <input type="text" value="UTF-8"/> | |

Jobs (proyectos)

¿Qué es un Job?

La unidad más importante de Jenkins es un **Job**. Pueden ejecutarse de forma paralela y son controlados por el **Build Executor**.

Podemos tener **Jobs** de diferentes tipos:

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

- **Multi-configuration Project:** Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.
- **Folder:** Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.
- **GitHub Organization:** Scans a GitHub organization (or user account) for all repositories matching some defined markers.
- **Multibranch Pipeline:** Creates a set of Pipeline projects according to detected branches in one SCM repository.

Cada vez que ocurre una ejecución de un **Job** se añade un numero al **Build History** y sirve para tener auditorias de cuál trabajo fue el último success o fail.

Apuntes:

Según la documentación de Jenkins: **Job**, *it's a deprecated term*. The correct term is **Project**, *a user-configured description of work which Jenkins should perform, such as building a piece of software, etc.*

Nota: Se recomienda que los nombre de las proyectos sigan en el estilo snake_case.

Información de un Job (proyecto)

Proyectos realizados en los cursos de Platzi y CódigoFacilito






 [edit description](#)

| All  | | | | | |
|---|---|------------------------------------|------------------------------------|-----------------------------------|---|
| S | W | Name ↑ | Last Success | Last Failure | Last Duration |
|  |  | Watchers | 7 days 15 hr - #11 | N/A | 42 ms  |
|  |  | Parameterized | 6 days 10 hr - #3 | N/A | 36 ms  |
|  |  | Node-Tests | 6 days 13 hr - #3 | N/A | 1.9 sec  |
|  |  | Node-Test-Pipeline | 6 days 10 hr - #3 | 6 days 12 hr - #1 | 6 sec  |
|  |  | hello-platzi | 7 days 15 hr - #19 | N/A | 83 ms  |

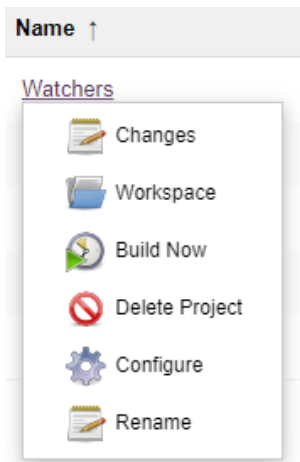
Icon: [S](#) [M](#) [L](#)

[Legend](#)  [Atom feed for all](#)  [Atom feed for failures](#)  [Atom feed for just latest builds](#)

- S: Indica el estado del ultimo “build” (Azul: correcto – Rojo: error – Gris: desactivada/ Sin compilar).
- W: Representa el estado general de un proyecto utilizando iconos del clima, que cambian según el número de compilaciones recientes que han pasado.

| Icon | Health |
|--|---|
|  | Sunny , more than 80% of Runs passing |
|  | Partially Sunny , 61% to 80% of Runs passing |
|  | Cloudy , 41% to 60% of Runs passing |
|  | Raining , 21% to 40% of Runs passing |
|  | Storm , less than 21% of Runs passing |

- Last Success: Fecha de la ultima ejecución exitosa.
- Last Failure: Fecha de la ultima ejecución fallida.
- Last Duration: Duración de la ultima ejecución.
- Name: Nombre del proyecto. Dando clic en la flecha situada a la derecha de cada nombre se despliega un menú de opciones.



Configuración de un Job (proyecto)

General

The screenshot shows the 'General' tab of the Jenkins Job Configuration page. At the top is a 'Description' text area. Below it is a '[Plain text] Preview' link. A list of checkboxes follows: 'Discard old builds', 'GitHub project', 'This build requires lockable resources', 'This project is parameterised', 'Throttle builds', 'Disable this project', and 'Execute concurrent builds if necessary'. To the right of these checkboxes are five question mark icons. At the bottom right is an 'Advanced...' button.

- **Discard old build:** permite configurar el tratamiento que se le darán a los *builds* de un Proyecto en relación al archivado y almacenamiento en disco. Dispone de dos opciones:
 - Días: cantidad de días que se almacenarán los *builds*, por ejemplo, 365 días (puede estar asociado a revisiones de auditoría).
 - Numero de builds: cantidad de builds que se almacenarán (ordenados del más reciente al menos reciente).
- **This project is parameterised:** Permite definir parámetros dentro del build. Por ejemplo:

Boolean Parameter
Choice Parameter
Credentials Parameter
File Parameter
List Subversion tags (and more)
Multi-line String Parameter
Password Parameter
Run Parameter
String Parameter

- **Disable this Project:** Si algo sale mal en un Job y se quiere evitar que el mismo se ha ejecutado, se debe activar esta propiedad.

Source Code Management

Podemos asociar a Jenkins un administrador de código (versionador), en este caso Git y Subversion.

Source Code Management

- ☒ None
- ☐ Git
- ☐ Subversion

Build Triggers

Build Triggers

- ☐ Trigger builds remotely (e.g., from scripts)
- ☐ Build after other projects are built
- ☐ Build periodically
- ☐ GitHub hook trigger for GITScm polling
- ☐ Poll SCM

- **Trigger builds remotely:** permite ejecutar el proyecto desde una API.

Use the following URL to trigger build remotely:
JENKINS_URL/job/NAME_JOB/build?token=TOKEN_NAME.

Ejemplo: `http://35.188.104.196:8080/job/remote_control/build?token=secret_key`

Usos: La solicitud puede ser realizada desde:

- Un navegador
 - Cualquier cliente HTTP
 - CURL. En este caso es requisito autenticarse, por tanto, se debe pasar el usuario y contraseña asociado a Jenkins. Ejemplo, ***curl -u username:password URL***
- **After other projects are built:** Permite configurar un encadenamiento de proyectos (jobs). Por ejemplo, si se ejecutó exitosamente el proyecto A quiero que se ejecute el proyecto B.
- **Build periodically:** Proporciona una función similar a cron para ejecutar periódicamente este proyecto.

Cron es una herramienta para configurar tareas programadas en sistemas Unix. Recomendando el uso de <https://crontab.guru> (editor rápido y simple para expresiones cron).

This field follows the syntax of cron (with minor differences). Specifically, each line consists of 5 fields separated by TAB or whitespace:

MINUTE HOUR DOM MONTH DOW

MINUTE Minutes within the hour (0–59)

HOUR The hour of the day (0–23)

DOM The day of the month (1–31)

MONTH The month (1–12)

DOW The day of the week (0–7) where 0 and 7 are Sunday.

To specify multiple values for one field, the following operators are available. In the order of precedence,

- * specifies all valid values
- M-N specifies a range of values
- M-N/X or */X steps by intervals of X through the specified range or whole valid range
- A,B,...,Z enumerates multiple values

- **Github hook trigger for GITScm polling:** El proyecto se ejecuta una vez se haya realizado un PUSH a Github.

Build Environment

Build Environment

- ☐ Delete workspace before build starts
- ☐ Use secret text(s) or file(s)
- ☐ Abort the build if it's stuck
- ☐ Add timestamps to the Console Output
- ☐ Inspect build log for published Gradle build scans
- ☐ With Ant

- **Delete workspace before build starts:** se recomienda activar esta propiedad en caso de que cada ejecución del proyecto cree un archivo, por ejemplo, la generación de un archivo JAR (por defecto los archivos nunca se borrarán si no se activa).
- **Use secret text(s) or file(s):** Permite añadir secretos (parámetros o configuración que no debería estar expuesto a otros usuarios), que serán accedidos a través de un script y solo estarán a la vista de Jenkins.
- **Abort the build if it's stuck:** permite configurar la acción a realizar cuando la ejecución de un proyecto se ha bloqueado, por ejemplo, que aborte cuando hayan pasado 5 minutos.

Build

Execute Windows batch command

Execute shell

Invoke Ant

Invoke Gradle script

Invoke top-level Maven targets

Run with timeout

Set build status to "pending" on GitHub commit

Post-build Actions

- Aggregate downstream test results
- Archive the artifacts
- Build other projects
- Publish JUnit test result report
- Record fingerprints of files to track usage
- Git Publisher
- E-mail Notification
- Editable Email Notification
- Set GitHub commit status (universal)
- Set build status on GitHub commit [deprecated]
- Delete workspace when build is done

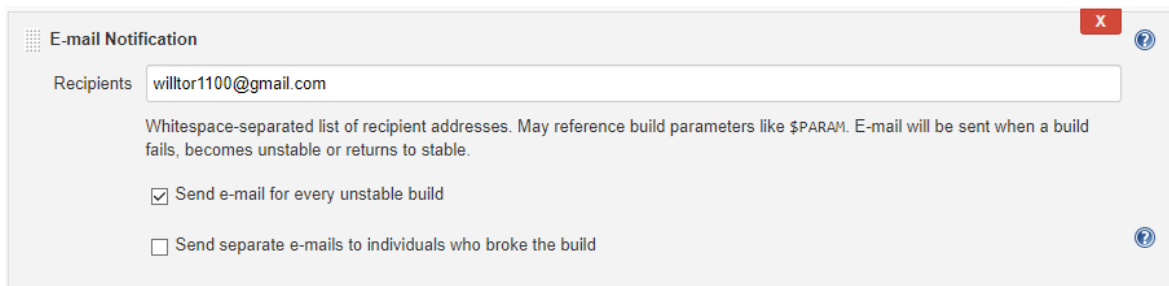
E-mail Notification

Si está configurado, Jenkins enviará un correo electrónico a los destinatarios especificados cuando ocurra un evento importante determinado.

- Cada compilación fallida desencadena un nuevo correo electrónico.
- Una compilación exitosa después de una compilación fallida (o inestable) desencadena un nuevo correo electrónico, lo que indica que una crisis ha terminado.
- Una compilación inestable después de una compilación exitosa desencadena un nuevo correo electrónico, indicando que hay una regresión.
- A menos que esté configurado, cada compilación inestable desencadena un nuevo correo electrónico, lo que indica que la regresión todavía está allí.

Para proyectos perezosos donde las compilaciones inestables son la norma, desmarque "Enviar correo electrónico para cada compilación inestable".

Jenkins permite una lista de direcciones de destinatarios separadas por espacios en blanco. Puede hacer referencia a parámetros de compilación como \$PARAM.



The screenshot shows the 'E-mail Notification' configuration panel in Jenkins. It features a title bar with a close button (X) and a help icon (?). Below the title, there is a 'Recipients' label followed by a text input field containing 'willtor1100@gmail.com'. A descriptive text block explains that the input is a 'Whitespace-separated list of recipient addresses' and that 'E-mail will be sent when a build fails, becomes unstable or returns to stable.' Below this, there are two checkboxes: the first is checked and labeled 'Send e-mail for every unstable build', and the second is unchecked and labeled 'Send separate e-mails to individuals who broke the build'. A help icon (?) is located at the bottom right of the panel.

Plugins

Jenkins y su ecosistema de Plugins

Una de las razones por la cual Jenkins es adorado es porque tiene **Plugins** para una mayoría de cosas.

Los **plugins** son unidades que extiende a Jenkins, después de instalarlo nuestra herramienta puede hacer algo nuevo, es recomendado instalarlos con la opción de `""Download now and Install after restart""` y así **Jenkins** se va a encargar de ejecutar todos los **Jobs** que estaban corriendo y cuando eso termine, lo va a instalar.

- Administración de Plugins: Manage Jenkins -> Manage Plugins
- Configuración Global: Manage Jenkins -> Global Tool Configuration

Cadenas de Jobs

Proceso realizado en clase:

1. Primero instalamos el plugin **Parameterized Trigger**.
2. Luego vamos a crear 2 jobs nuevos:
 - a. **Watchers** (será ejecutado cuando Hello-Platzi se compile)
 - i. Nos dirigimos a la sección *Build Triggers* y marcamos la opción `"Build after other projects are built"`.
 - ii. Establecemos que el proyecto se ejecute únicamente si el proyecto `"Hello-Platzi"` es construido exitosamente.
 - iii. En la sección *Build* seleccionamos la opción *Executed Shell* y escribimos: `echo "Running after hello-platzi success"`.
 - b. **Parameterized** (su ejecución se iniciará cuando Hello-Platzi lo indique)
 - i. Al crearlo se selecciona la propiedad `"This project is parameterized"`.
 - ii. Se agrega el parámetro `ROOT_ID`.
 - iii. En la sección *Build* seleccionamos la opción *execute Shell* y escribimos `echo "calle with $ROOT_ID"`.
3. Modificamos el job Hello-Platzi
 - a. En la sección principal podemos visualizar que Jenkins reconoce la dependencia entre Jobs:

Downstream Projects



- b. Se añade un *build step* desde la sección *Build*. Se selecciona la opción `"Trigger/call build on other projects"`, y referenciamos el proyecto *Parameterized*. Además se añade un parámetro desde la opción *Predefined parameters* y escribimos: `"ROOT_ID=$BUILD_NUMBER"`. `BUILD_NUMBER` es una variable de entorno, que almacena el valor resultante una ejecución.
4. Procedemos a ejecutar Hello-Platzi desde la opción `"build with parameters"`. Podemos notar en las consolas que Hello-Platzi llama declarativamente a Parameterized e indirectamente a Watchers.

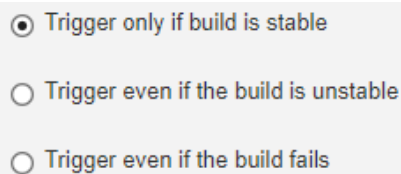
5. Resultado:

Console Output

```
Started by user William Torres
Running as SYSTEM
Building in workspace /var/lib/jenkins/workspace/hello-platzi
[WS-CLEANUP] Deleting project workspace...
[WS-CLEANUP] Deferred wipeout is used...
[WS-CLEANUP] Done
[hello-platzi] $ /bin/sh -xe /tmp/jenkins2438556208404109986.sh
+ echo Author is William Torres
Author is William Torres
Triggering projects: Parameterized
Triggering a new build of Watchers
Finished: SUCCESS
```

Apuntes:

1. Jenkins conoce la cadena de ejecución (incluyendo que job inició el proceso).
2. Se recomienda usar el estilo de *parameterized* (Trigger/call build on other projects) en vez del estilo de *watchers* (Build after other projects are built), porque este ultimo solo ofrece tres opciones.



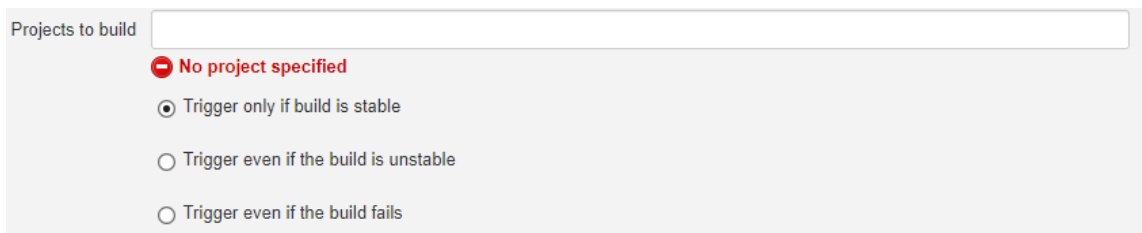
Three radio button options for triggering other projects:

- ☒ Trigger only if build is stable
- ☐ Trigger even if the build is unstable
- ☐ Trigger even if the build fails


3. En la sección Post-build Actions puede encontrar dos opciones para contruir otros proyectos:
 - a. Build other projects

Trigger builds of the other projects once a build is successfully completed. Multiple projects can be specified by using comma, like "abc, def".

Other than the obvious use case where you'd like to build other projects that have a dependency on the current project, this can also be useful to split a long build process in to multiple stages (such as the build portion and the test portion).



Projects to build

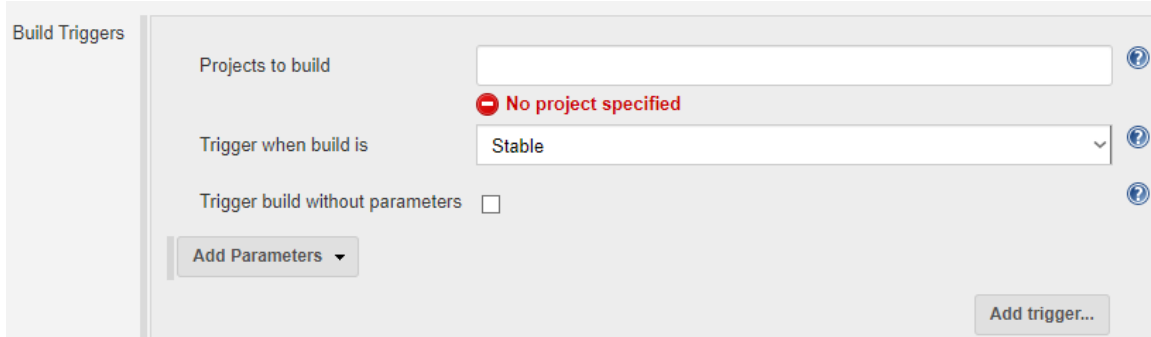
 No project specified

- ☒ Trigger only if build is stable
- ☐ Trigger even if the build is unstable
- ☐ Trigger even if the build fails

- b. Trigger parameterized build on other projects

This plugin triggers builds on other projects, with parameters that are predefined, or supplied by the finished build.

Every parameter will be passed to the target project(s), even if the target is not parameterized, or if no property of that name is defined.



Build Triggers

Projects to build ?
- No project specified

Trigger when build is Stable ?

Trigger build without parameters ☐ ?

Add Parameters ▾

Add trigger...

Conectándonos a GitHub

Es posible conectar un repositorio de GitHub a Jenkins, de modo que cuando ocurra un evento en el repositorio (por ejemplo, un *push*) automáticamente se ejecute un *build* del source code. Para que esto sea posible se deben realizar algunas configuraciones en Jenkins y GitHub.

En Jenkins:

1. Verificar que el **GitHub plugin** esté instalado (en caso de no estarlo proceder a instalarlo).
2. Se crea el Job de acuerdo a la necesidad del proyecto.
3. En la sección *General* se marca la opción "Github Project" y se ingresa la URL de acceso del repositorio.
4. En la sección *SCM (Source Code Management)* se marca la opción "Git", y se ingresa la URL del repositorio que permite la clonación (el host de Jenkins debe tener instalado Git).
 - En el apartado "Branches to build" se especifica el branch que Jenkins identificará y monitoreará.
 - En caso de presentarse este error: **(403) No valid crumb was included in the request Jenkins**, en este [post](#) de Stackoverflow se plantean soluciones.
5. En la sección *Build Triggers* se marca la opción "GitHub hook trigger for GITScm polling".

En GitHub:

1. Ingresar al repositorio de GitHub.
2. Ingresar en la sección Settings -> Webhooks.
 - a. Webhooks permite que servicios externos sean notificados cuando ocurren ciertos eventos. Cuando ocurra un evento específico (por ejemplo, un *push*), GitHub

enviará una solicitud POST a cada una de las URL proporcionadas en la configuración. Obtenga más información en la [Guía de Webhooks](#).

3. Se añade un nuevo Webhook.
4. Se añade la Payload URL (si la URL no acaba en `/github-webhook/`, GitHub lanzara un error.)
5. Se marca la opción "Just the push event".

Apuntes:

- Guia: https://drive.google.com/file/d/1vhBQIPUCZqQoE_NA5BEMHQ4ryJt4U5OS/view

Pipelines

¿Qué es un 'Pipeline'?

Documentación: <https://www.jenkins.io/doc/book/pipeline/>

Jenkins Pipeline (or simply "Pipeline" with a capital "P") is a suite of plugins which supports implementing and integrating *continuous delivery pipelines* into Jenkins.

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. This process involves building the software in a reliable and repeatable manner, as well as progressing the built software (called a "build") through multiple stages of testing and deployment.

Pipeline provides an extensible set of tools for modeling simple-to-complex delivery pipelines "as code" via the [Pipeline domain-specific language \(DSL\) syntax](#).^[1]

The definition of a Jenkins Pipeline is written into a text file (called a [Jenkinsfile](#)) which in turn can be committed to a project's source control repository.^[2] This is the foundation of "Pipeline-as-code"; treating the CD pipeline a part of the application to be versioned and reviewed like any other code.

Creating a Jenkinsfile and committing it to source control provides a number of immediate benefits:

- Automatically creates a Pipeline build process for all branches and pull requests.
- Code review/iteration on the Pipeline (along with the remaining source code).
- Audit trail for the Pipeline.
- Single source of truth^[3] for the Pipeline, which can be viewed and edited by multiple members of the project.

While the syntax for defining a Pipeline, either in the web UI or with a Jenkinsfile is the same, it is generally considered best practice to define the Pipeline in a Jenkinsfile and check that in to source control.

Declarative versus Scripted Pipeline syntax

A Jenkinsfile can be written using two types of syntax - Declarative and Scripted.

Declarative and Scripted Pipelines are constructed fundamentally differently. Declarative Pipeline is a more recent feature of Jenkins Pipeline which:

- provides richer syntactical features over Scripted Pipeline syntax, and
- is designed to make writing and reading Pipeline code easier.

Many of the individual syntactical components (or "steps") written into a Jenkinsfile, however, are common to both Declarative and Scripted Pipeline. Read more about how these two types of syntax differ in [Pipeline concepts](#) and [Pipeline syntax overview](#) below.

Why Pipeline?

Jenkins is, fundamentally, an automation engine which supports a number of automation patterns. Pipeline adds a powerful set of automation tools onto Jenkins, supporting use cases that span from simple continuous integration to comprehensive CD pipelines. By modeling a series of related tasks, users can take advantage of the many features of Pipeline:

- **Code:** Pipelines are implemented in code and typically checked into source control, giving teams the ability to edit, review, and iterate upon their delivery pipeline.
- **Durable:** Pipelines can survive both planned and unplanned restarts of the Jenkins master.
- **Pausable:** Pipelines can optionally stop and wait for human input or approval before continuing the Pipeline run.
- **Versatile:** Pipelines support complex real-world CD requirements, including the ability to fork/join, loop, and perform work in parallel.
- **Extensible:** The Pipeline plugin supports custom extensions to its DSL ^[1] and multiple options for integration with other plugins.

While Jenkins has always allowed rudimentary forms of chaining Freestyle Jobs together to perform sequential tasks, ^[4] Pipeline makes this concept a first-class citizen in Jenkins.

Building on the core Jenkins value of extensibility, Pipeline is also extensible both by users with [Pipeline Shared Libraries](#) and by plugin developers. ^[5]

Pipeline concepts

The following concepts are key aspects of Jenkins Pipeline, which tie in closely to Pipeline syntax (see the [overview](#) below).

Pipeline

A Pipeline is a user-defined model of a CD pipeline. A Pipeline's code defines your entire build process, which typically includes stages for building an application, testing it and then delivering it.

Also, a pipeline block is a [key part of Declarative Pipeline syntax](#).

Node

A node is a machine which is part of the Jenkins environment and is capable of executing a Pipeline.

Also, a node block is a [key part of Scripted Pipeline syntax](#).

Stage

A stage block defines a conceptually distinct subset of tasks performed through the entire Pipeline (e.g. "Build", "Test" and "Deploy" stages), which is used by many plugins to visualize or present Jenkins Pipeline status/progress. ^[6]

Step

A single task. Fundamentally, a step tells Jenkins *what* to do at a particular point in time (or "step" in the process). For example, to execute the shell command make use the sh step: sh 'make'. When a plugin extends the Pipeline DSL, ^[4] that typically means the plugin has implemented a new *step*.

Declarative Pipeline fundamentals

In Declarative Pipeline syntax, the pipeline block defines all the work done throughout your entire Pipeline.

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

- ❶ Execute this Pipeline or any of its stages, on any available agent.
- ❷ Defines the "Build" stage.
- ❸ Perform some steps related to the "Build" stage.
- ❹ Defines the "Test" stage.
- ❺ Perform some steps related to the "Test" stage.
- ❻ Defines the "Deploy" stage.
- ❼ Perform some steps related to the "Deploy" stage.

¿Cómo puedo acelerar mi development de Pipelines?

Pipeline Syntax, this Snippet Generator will help you learn the Pipeline Script code which can be used to define various steps. Pick a step you are interested in from the list, configure it, click

Generate Pipeline Script, and you will see a Pipeline Script statement that would call the step with that configuration. You may copy and paste the whole statement into your script, or pick up just the options you care about. (Most parameters are optional and can be omitted in your script, leaving them at default values.)

Ejemplo (*activar la ejecución de Parameterized*):

Steps

Sample Step build: Build a job

Project to Build Parameterized

☐ Wait for completion

☒ Propagate errors

Quiet period

Parameters ROOT_ID \$BUILD_ID

[?](#) [?](#) [?](#) [?](#)

[Generate Pipeline Script](#)

```
build job: 'Parameterized', parameters: [string(name: 'ROOT_ID', value: '$BUILD_ID')], wait: false
```

Con la opción *Replay* (disponible en el dashboard de cada ejecución) **NO** hay necesidad de hacer un commit para volver a ejecutar algún build en un punto específico, y además es posible modificar (y agregar) se puede cambiar los *steps*.

Slave

Introducción a Slaves

Los Slaves nos permiten correr Jobs distribuidamente. Se conecta al Jenkins master y este le delega trabajos al Slave como si fuese otra máquina puede ser virtual, física como quieras hacerlo, nos permite escalar horizontalmente.

Apuntes:

- [Jenkins User Handbook](#)
- [Distributed builds](#)

Conectado un Slave

Se debe agregar un Nodo. Se recomienda que el método de lanzamiento sea a través de SSH (se debe agregar la llave SSH en el servidor que contiene el Nodo esclavo).

The screenshot displays the Jenkins 'New Node' configuration interface. The main form includes the following fields and values:

- Name:** Slave-1
- Description:** Slave allows at American Server
- # of executors:** 2
- Remote root directory:** /var/jenkins
- Labels:** (empty)
- Usage:** Use this node as much as possible
- Launch method:** Launch agents via SSH
- Host:** 10.0.10.20
- Credentials:** none (with an 'Add' button)
- Host Key Verification Strategy:** Known hosts file Verification Strategy

A red error message is displayed below the Credentials field: "The selected credentials cannot be found". The left sidebar contains navigation links: "Back to Dashboard", "Manage Jenkins", "New Node", "Configure Clouds", and "Node Monitoring". Below these are two expandable sections: "Build Queue" (showing "No builds in the queue.") and "Build Executor Status" (showing "1 Idle" and "2 Idle"). An "Advanced..." button is located at the bottom right of the form.

Comandos utilizados para configurar el Nodo esclavo

1. Agregar un usuario para Jenkins: ***add user jenkins***
2. Actualizar repositorios: ***apt-get update***
3. Instalar Java8: ***apt-get install openjdk-8-jdk wget gnupg git vim***
4. Crear un directorio y asignarle privilegios al usuario Jenkins sobre él

mkdir /jenkins

chown -R jenkins:jenkins /jenkins

5. Manipular la llave SSH

sudo su jenkins

vim /home/jenkins/.ssh/authorized_keys

Notificaciones de escritorio

Mantente al tanto de tu trabajo con [CatLight](#). Obtenga alertas accionables sobre sus compilaciones, tareas y solicitudes de extracción.

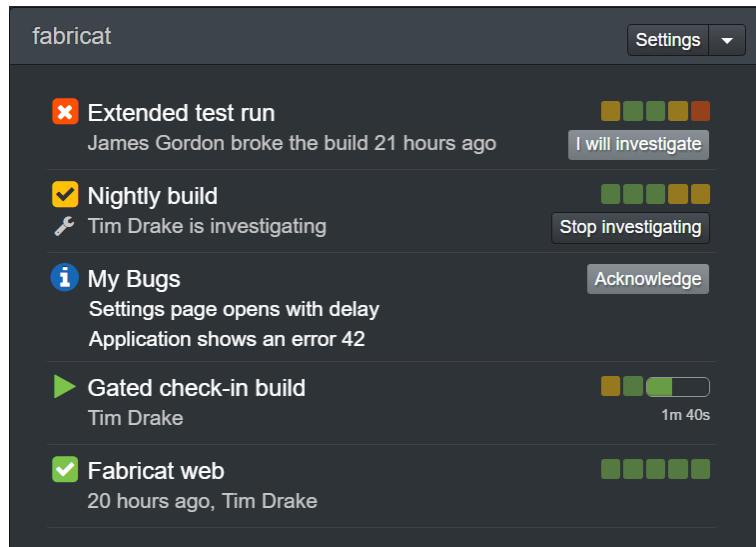
Documentación: <https://support.catlight.io/s1-general/knowledgebase>

Inicio de sesión: <https://service.catlight.io/ui/login>

Characteristics

Personal dashboard

See the status of your builds, releases and issues on one personalized dashboard.

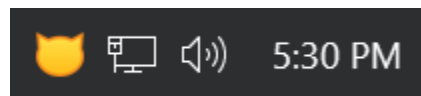


Prioritized action list

For large dashboards, the most important items will be displayed in the action list at the top.

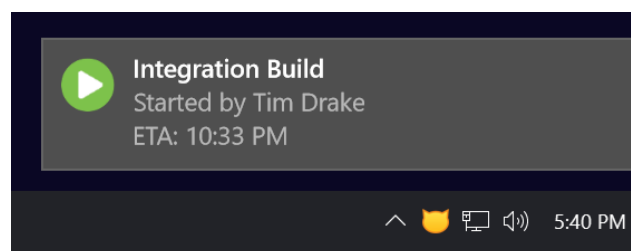
One status icon

See the overall status of your action list in one icon. When it is green, everything is fine. Otherwise, something needs your attention.



Notifications

Get notifications when a build breaks down, a new, important bug appears, or your task list changes.

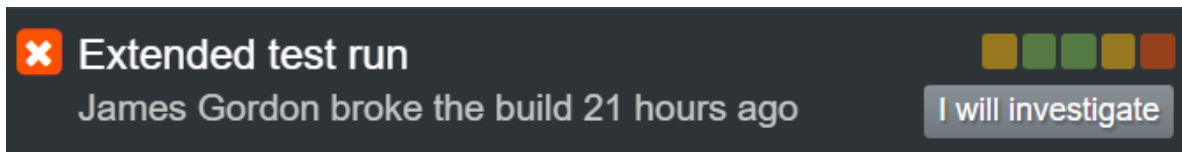


Estimates

Get estimates on build time completion. See them in notifications and on the dashboard.

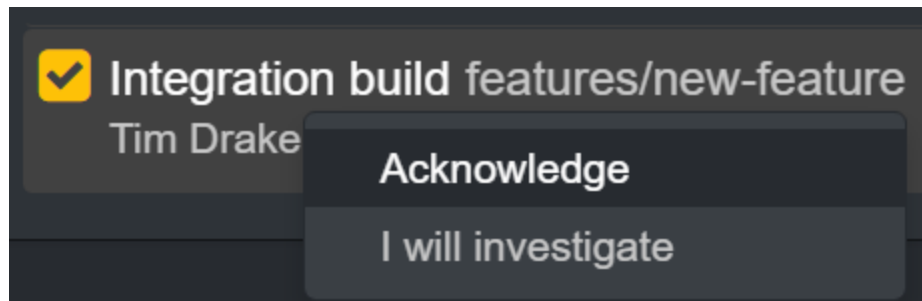
Instant context

Build and release pipelines will show previous history so you can see when the first breakage occurred.



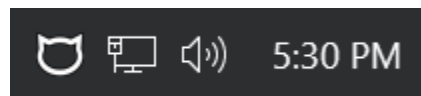
Acknowledgements

Acknowledge alerts that you want to fix later. This can help you identify new action items faster. When all alerts are acknowledged, the tray icon will change to an outline.



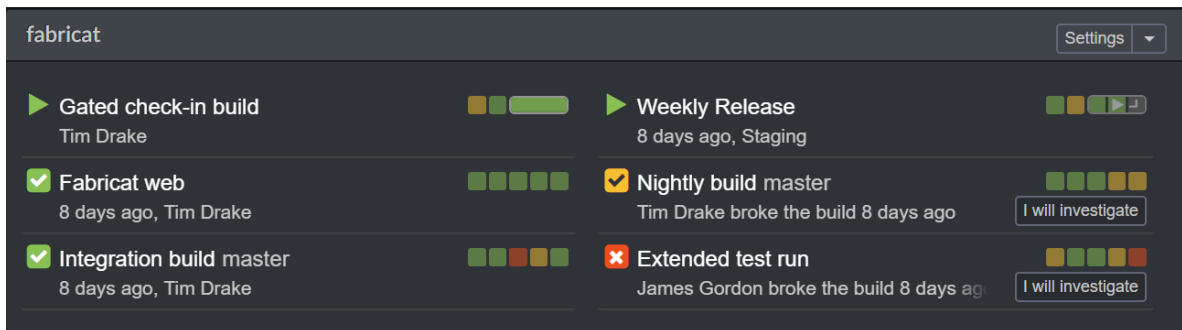
Focus mode

Getting too many notifications? Try focus mode. The app will show notifications only if you break the build, and the CatLight icon will turn white.



Team dashboard

Show the CatLight dashboard on a shared screen to see an overview of your team status.




Investigations


Notify the team that you are checking a broken build or a release. They will be notified and see it on the dashboard. Use investigations to make sure that an engineer is working on a problem, and that it is not handled by multiple engineers simultaneously.




Comments

Use comments to provide a status update, share the reason of the build breakage, or to provide troubleshooting steps for engineers who will investigate next time.


 **Integration build** ■ ■ ■ ■ ■

 Tim Drake is investigating Comment (1) Stop investigating

 Build machine configuration changed. Upgrading scripts, eta 1h

First problem identification

See who broke the build first.

 **Extended test run** ■ ■ ■ ■ ■

James Gordon broke the build 21 hours ago I will investigate

Team status

See the number of active alerts for every team member. Use this to see if some engineers are overloaded and cannot keep up with the work.

| Team status ⓘ | | | | | | Settings ▾ |
|---------------|----|---|---|-------|---|------------|
| | ● | ● | ● | ● | 🔧 | |
| Alice Nettle | 30 | 5 | - | 2 | - | |
| Dave Rogers | 10 | - | 5 | 2 / 3 | 1 | |
| James Gordon | 20 | - | - | - | - | |
| Tim Drake | 20 | - | 3 | - | - | |

Common settings

After configuring the dashboard and setting optimal notifications, you can share them with the team.

Export setting

- ☒ Dashboard (e.g. servers, monitored builds)
- ☒ Application settings (e.g. notifications, branches, polling)

Export

