



**KALEIDO** | COWORKING  
CENTER

opsou

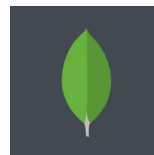
pedrofigueras



Altia **Senior Developer**  
Disfrutando del desarrollo web  
desde 1998.

 @rolando\_caldas

<https://rolandocaldas.com>



# Preparando el entorno: Docker

Necesitamos tener Docker funcionando en nuestro equipo:

- Ubuntu: <https://docs.docker.com/install/linux/docker-ce/ubuntu/>
- Windows: <https://store.docker.com/editions/community/docker-ce-desktop-windows>
- Mac: <https://store.docker.com/editions/community/docker-ce-desktop-mac>
- Legacy Support: <https://docs.docker.com/toolbox/>
- Post-Install:
  - Ubuntu: Lanzar docker sin ser root + Activar acceso remoto (IDE)
  - Windows: Activar acceso remoto + Permitir conexiones no seguras
  - Mac: Activar acceso remoto

# Preparando el entorno: Docker sin ser root

- Es necesario crear el grupo docker y asociarlo al usuario con el que se va a utilizar docker:

```
$ sudo groupadd docker  
$ sudo usermod -aG docker $USER
```

- Logout/Login
- GO GO GO

*Preparando el entorno: Estamos listos!*



# ¿Qué es la integración continua?

*Integración continua*

# Compilación

*Integración continua*

Compilación  
+



*Integración continua*

Compilación  
+  
Tests

# Integración continua

*“La integración continua es un modelo informático propuesto inicialmente por Martin Fowler que consiste en hacer integraciones automáticas de un proyecto lo más a menudo posible para así poder detectar fallos cuanto antes. Entendemos por integración la compilación y ejecución de pruebas de todo un proyecto.”*

– Wikipedia

# *Integración continua: La compilación*

## **La compilación en proyectos en PHP**

# *Integración continua: La compilación*

## **La compilación en proyectos en PHP**

- Ejecución de composer para la carga de librerías.

# *Integración continua: La compilación*

## **La compilación en proyectos en PHP**

- Ejecución de composer para la carga de librerías.
- Generación de caché del framework.

# *Integración continua: La compilación*

## **La compilación en proyectos en PHP**

- Ejecución de composer para la carga de librerías.
- Generación de caché del framework.
- Creación de assets

# *Integración continua: La compilación*

## **La compilación en proyectos en PHP**

- Ejecución de composer para la carga de librerías.
- Generación de caché del framework.
- Creación de assets
- etc

# *Integración continua: La compilación*

## **La compilación en proyectos en Javascript**



# *Integración continua: La compilación*

## **La compilación en proyectos en Javascript**

- Transformación de TypeScript a Javascript.

# *Integración continua: La compilación*

## **La compilación en proyectos en Javascript**

- Transformación de TypeScript a Javascript.
- Ejecución de NPM o similar.

# *Integración continua: La compilación*

## **La compilación en proyectos en Javascript**

- Transformación de TypeScript a Javascript.
- Ejecución de NPM o similar.
- Ejecución de WebPack o similar.

# *Integración continua: La compilación*

## **La compilación en proyectos Dockerizados**

# *Integración continua: La compilación*

## **La compilación en proyectos Dockerizados**

- Creación de la imagen docker.

# *Integración continua: La compilación*

## **La compilación en proyectos Dockerizados**

- Creación de la imagen docker.
- Asignación de tag o etiqueta a la imagen.

# *Integración continua: La compilación*

## **La compilación en proyectos Dockerizados**

- Creación de la imagen docker.
- Asignación de tag o etiqueta a la imagen.
- Almacenamiento en un registry.

# *Integración continua: Los tests*

**Ejecución de todo tipo de test disponibles**



# *Integración continua: Los tests*

## **Ejecución de todo tipo de test disponibles**

- Test unitarios.

# Integración continua: Los tests

## Ejecución de todo tipo de test disponibles

- Test unitarios.
- Test funcionales o de aceptación.

# Integración continua: Los tests

## Ejecución de todo tipo de test disponibles

- Test unitarios.
- Test funcionales o de aceptación.
- Test de integración.

¿Qué es la entrega continua?

*Entrega continua*

# Integración continua

*Entrega continua*

Integración continua

+

*Entrega continua*

# Integración continua + Despliegues automatizados

# Entrega continua

La entrega continua (continuous delivery) nos permite tener en todo momento entornos con la aplicación en ejecución y actualizada.



# Entrega continua

La entrega continua (continuous delivery) nos permite tener en todo momento entornos con la aplicación en ejecución y actualizada.

Para llevarlo a cabo, se suelen utilizar las ramas del control de versiones para mantener una versión accesible preProducción (rama dev) y una de producción (rama master).

# *Integración y entrega continua*

De forma tradicional, los proceso de CI/CD conllevaba el uso de múltiples herramientas independientes, conectadas entre sí.

# *Integración y entrega continua*

De forma tradicional, los proceso de CI/CD conllevaba el uso de múltiples herramientas independientes, conectadas entre sí.

GIT

JENKINS

REDMINE

SVN

TRAVIS

ANSIBLE

TRAC

# Entrega continua



# *Entrega continua*



# Gitlab y Gitlab CI/CD

Gitlab es conocido como el “GitHub libre” y nos permite gestionar proyectos de software con su servicio en la nube o instalándolo en nuestro propio servidor.

**Es una solución integral.**

# *Gitlab y Gitlab CI/CD*

**Características:**

# Gitlab y Gitlab CI/CD

## Características:

- Control de versiones: GIT



# Gitlab y Gitlab CI/CD

## Características:

- Control de versiones: GIT
- Gestión de proyectos: Gestión de tickets e incidencias

# Gitlab y Gitlab CI/CD

## Características:

- Control de versiones: GIT
- Gestión de proyectos: Gestión de tickets e incidencias
- Agile: Panel Kanban y agrupación de tareas por milestones

# Gitlab y Gitlab CI/CD

## Características:

- Control de versiones: GIT
- Gestión de proyectos: Gestión de tickets e incidencias
- Agile: Panel Kanban y agrupación de tareas por milestones
- Snippets: Almacén de código/plantillas a compartir

# Gitlab y Gitlab CI/CD

## Características:

- Control de versiones: GIT
- Gestión de proyectos: Gestión de tickets e incidencias
- Agile: Panel Kanban y agrupación de tareas por milestones
- Snippets: Almacén de código/plantillas a compartir
- Wiki: Una wiki propia por proyecto

# Gitlab y Gitlab CI/CD

## Características:

- Control de versiones: GIT
- Gestión de proyectos: Gestión de tickets e incidencias
- Agile: Panel Kanban y agrupación de tareas por milestones
- Snippets: Almacén de código/plantillas a compartir
- Wiki: Una wiki propia por proyecto
- Registry: Un registro de imágenes docker

# Gitlab y Gitlab CI/CD

## Características:

- Control de versiones: GIT
- Gestión de proyectos: Gestión de tickets e incidencias
- Agile: Panel Kanban y agrupación de tareas por milestones
- Snippets: Almacén de código/plantillas a compartir
- Wiki: Una wiki propia por proyecto
- Registry: Un registro de imágenes docker
- CI/CD: Sistema propio de integración y despliegue continuo utilizando gitlab-runner

# Gitlab y Gitlab CI/CD

## Características:

- Control de versiones: GIT
- Gestión de proyectos: Gestión de tickets e incidencias
- Agile: Panel Kanban y agrupación de tareas por milestones
- Snippets: Almacén de código/plantillas a compartir
- Wiki: Una wiki propia por proyecto
- Registry: Un registro de imágenes docker
- CI/CD: Sistema propio de integración y despliegue continuo utilizando gitlab-runner
- Métricas

¿En qué nos centraremos?



# *Gitlab y Gitlab CI/CD*

Registry: Un registro de imágenes docker

# *Gitlab y Gitlab CI/CD*

Registry: Un registro de imágenes docker

CI/CD: Sistema propio de integración y despliegue continuo

# Gitlab y Gitlab CI/CD



## *Preparando el entorno: Descargando el proyecto*

**<https://github.com/rolando-caldas/workshop-gitlab>**

```
$ mkdir $HOME/devFestWorkshop  
$ cd $HOME/devFestWorkshop  
$ git clone https://github.com/rolando-caldas/workshop-gitlab
```

# Preparando el entorno: Configurando el proyecto

<https://github.com/rolando-caldas/workshop-gitlab>

```
$ cd workshop-gitlab && cp .env.example .env  
$ vim .env
```

```
# gitlab  
GITLAB_PATH=./gitlab  
GITLAB_HOST=gitlab.example.com
```

```
# gitlab runner  
GITLAB_RUNNER_PATH=./runner
```

# Preparando el entorno: Configurando el proyecto

<https://github.com/rolando-caldas/workshop-gitlab>

```
$ sudo vim /etc/hosts
```

```
127.0.0.1          localhost
127.0.1.1          rolando-Lenovo-Z50-70
172.24.0.2         gitlab.example.com
```

Windows: C:\Windows\System32\drivers\etc\hosts

# Preparando el entorno: Configurando el proyecto

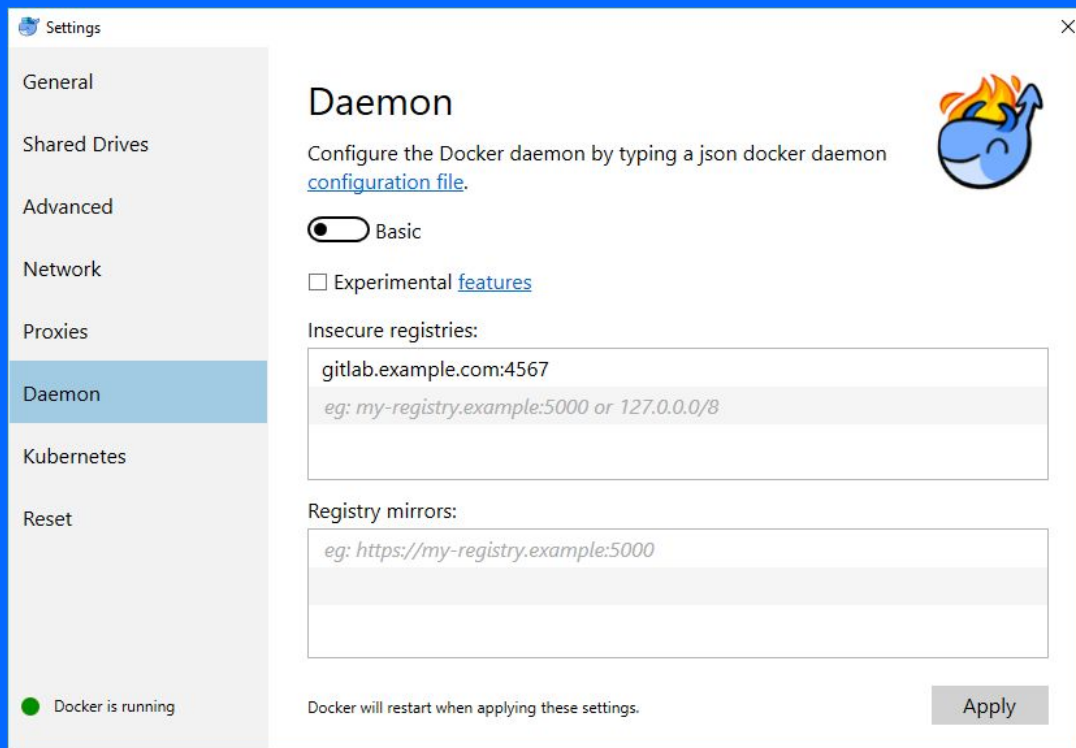
<https://github.com/rolando-caldas/workshop-gitlab>

```
$ sudo vim /etc/docker/daemon.json  
  
{  
    "insecure-registries" : [ "gitlab.example.com:4567" ]  
}
```

Permitimos el acceso de Docker al Registry en modo inseguro (HTTP)

En Windows y Mac click en el icono de Docker => Preferencias => +Daemon

# Preparando el entorno: Configurando el proyecto





*Preparando el entorno: Configurando el proyecto*

¿Y todo esto para qué?

# Levantando nuestro entorno de Gitlab

```
~/devFestWorkshop/workshop-gitlab$ docker-compose up -d
```

```
Creating network "workshop-gitlab_default" with the default driver
```

```
Recreating workshop-gitlab_gitlab_1 ... done
```

```
Recreating workshop-gitlab_gitlab-runner_1 ... done
```

# *Levantando nuestro entorno de Gitlab*

## Windows special bug: Part 1 - docker-compose.yml

```
volumes:  
  - '${GITLAB_PATH}/config:/etc/gitlab'  
  - '${GITLAB_PATH}/gitlab/logs:/var/log/gitlab'  
  - '${GITLAB_PATH}/gitlab/data:/var/opt'
```

# Levantando nuestro entorno de Gitlab

## Windows special bug: Part 2 - Convert Windows Paths

```
> SET COMPOSE_CONVERT_WINDOWS_PATHS=1  
> docker-compose up -d
```

```
Creating network "workshop-gitlab_default" with the default driver  
Recreating workshop-gitlab_gitlab_1          ... done  
Recreating workshop-gitlab_gitlab-runner_1    ... done
```

# *Levantando nuestro entorno de Gitlab*

## Windows special bug: Part 3 - Resolve IP range

```
#> route add 172.24.0.0 mask 255.255.0.0 10.0.75.2 -p  
> ping 172.24.0.2
```

# *Levantando nuestro entorno de Gitlab*

## Windows special bug: Part 4 - Resolve IP range on Docker Legacy

<https://forums.docker.com/t/how-to-access-docker-container-from-another-machine-on-local-network/4737/16>

```
#> cd 'C:\Program Files\Oracle\VirtualBox\'  
#> ./VBoxManage controlvm "default" natpf1  
"rule-name,tcp,,<port>,,<port>"  
#> ping 172.24.0.2
```

¿Funciona? Perfecto... ¿no funciona? Painfull path...

# *Levantando nuestro entorno de Gitlab*

## **Windows special bug: Part 5 – Resolve IP range on Docker Legacy**

<https://forums.docker.com/t/how-to-access-docker-container-from-another-machine-on-local-network/4737/16>

```
> docker-machine.exe ssh default
$ ifconfig
> ipconfig
docker address=192.168.99.100
windows address=192.168.99.1
> route add 172.24.0.0 mask 255.255.0.0 192.168.99.1 -p
```

# Levantando nuestro entorno de Gitlab

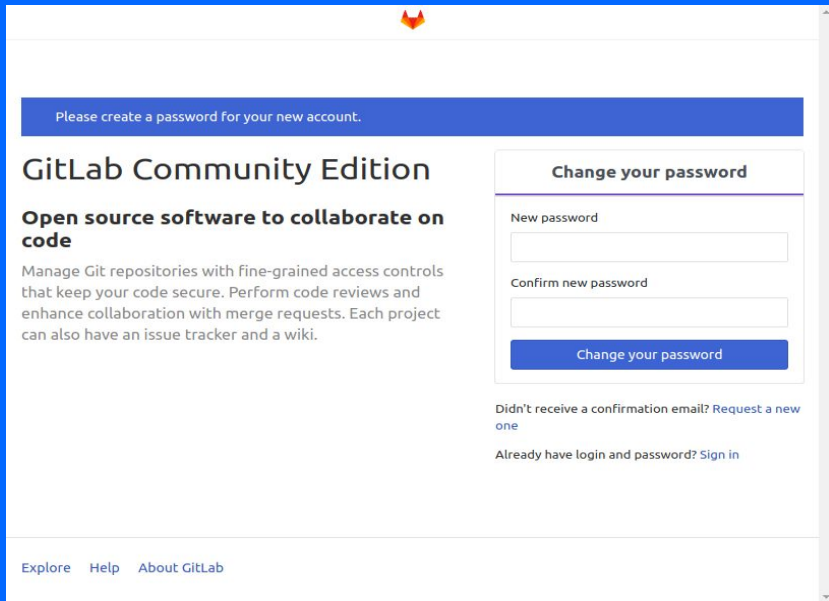
```
$ vim docker-compose.yml
```

- <http://gitlab.example.com:9090/> Es la ruta de nuestro Gitlab
- <http://gitlab.example.com:4567/> Es la ruta del registry de Docker
- La configuración de Gitlab y los datos de proyectos y logs se almacenan en `$HOME/devFestWokshop/workshop-gitlab/gitlab`
- La configuración de los runners se almacenan en `$HOME/devFestWokshop/workshop-gitlab/runner`
- La “máquina” Gitlab tiene la IP 172.24.0.2
- La “máquina” de Gitlab-runner tiene la IP 172.24.0.3



# Preparando el entorno: Configurando cuenta admin

<http://gitlab.example.com:9090/>

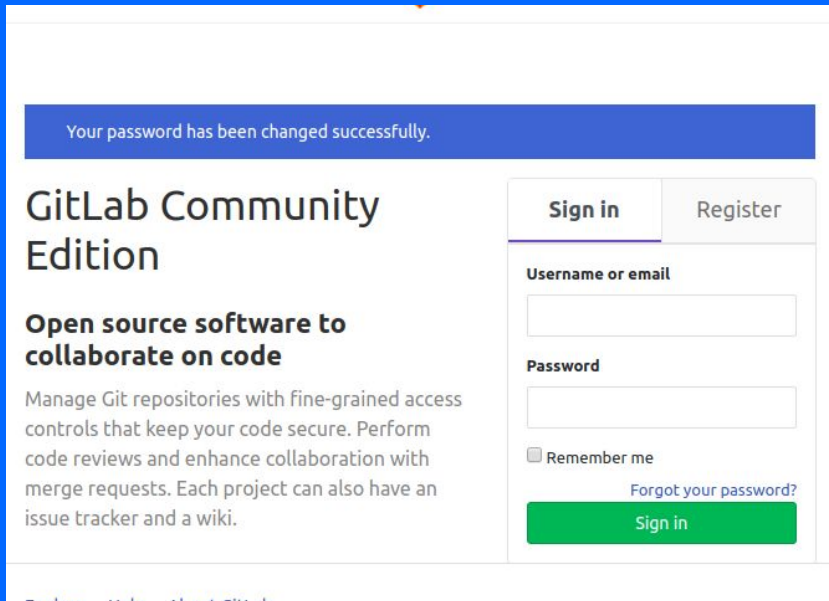


The screenshot shows the GitLab Community Edition password creation interface. At the top, a blue banner reads "Please create a password for your new account." Below this, the page is titled "GitLab Community Edition" with the subtitle "Open source software to collaborate on code". A brief description of GitLab's features is provided. On the right, a "Change your password" form contains two input fields for "New password" and "Confirm new password", followed by a "Change your password" button. Below the form, there are links for "Didn't receive a confirmation email? Request a new one" and "Already have login and password? Sign in". The footer includes links for "Explore", "Help", and "About GitLab".

- Ya tenemos una cuenta **root** con plenos poderes.
- Al acceder por primera vez hay que establecer una nueva contraseña de 8 caracteres

# Preparando el entorno: Configurando cuenta admin

<http://gitlab.example.com:9090/>



Your password has been changed successfully.

## GitLab Community Edition

**Open source software to collaborate on code**

Manage Git repositories with fine-grained access controls that keep your code secure. Perform code reviews and enhance collaboration with merge requests. Each project can also have an issue tracker and a wiki.

**Sign in** Register

Username or email

Password

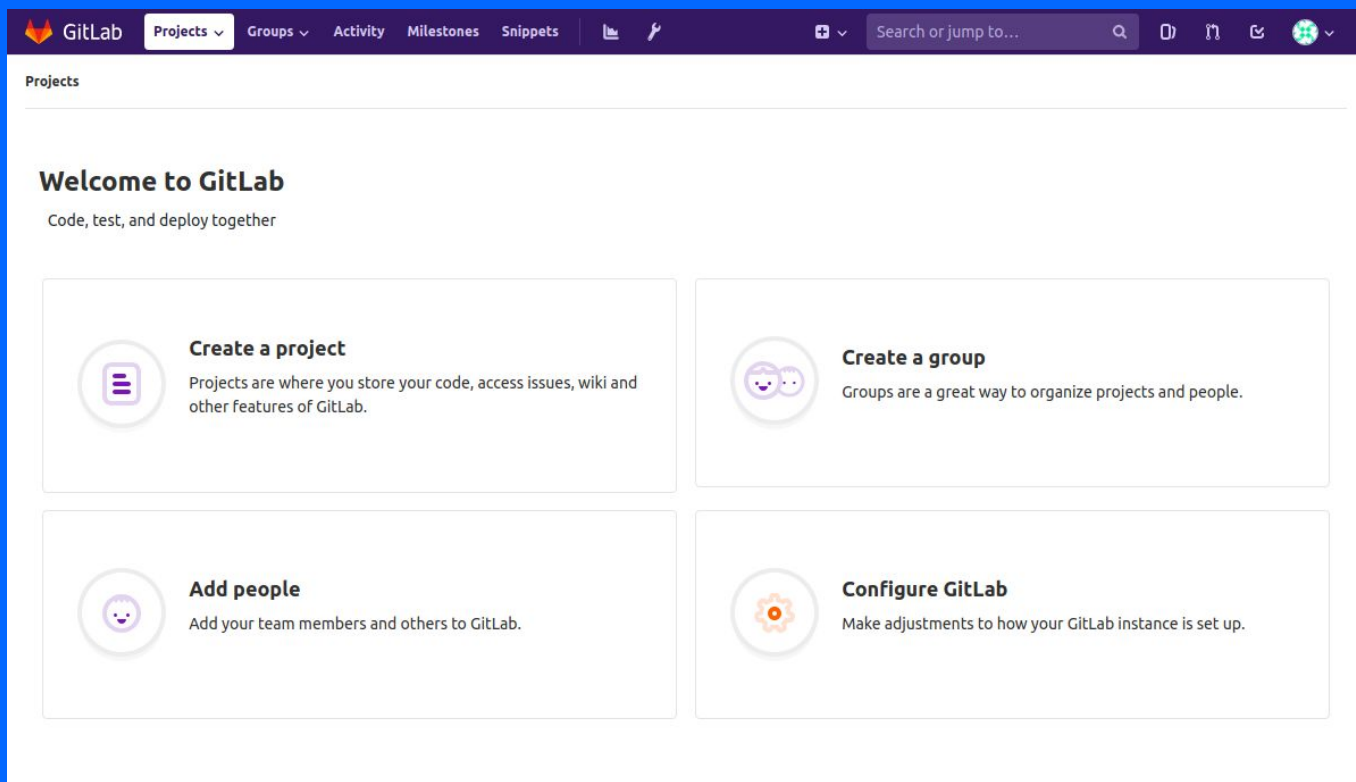
☐ Remember me

[Forgot your password?](#)

**Sign in**

- Tras cambiar la contraseña accedemos a la pantalla de login.
- El username es **root** y la contraseña la introducida en la pantalla anterior.

# Estamos dentro!



The screenshot shows the GitLab web interface. At the top is a dark purple navigation bar with the GitLab logo, a dropdown menu for 'Projects', and links for 'Groups', 'Activity', 'Milestones', and 'Snippets'. There is also a search bar and several utility icons. Below the navigation bar, the page title 'Projects' is displayed. The main content area has a heading 'Welcome to GitLab' followed by the tagline 'Code, test, and deploy together'. Below this, there are four white cards arranged in a 2x2 grid, each with a circular icon and a title. The first card has a purple icon of a document with a list and is titled 'Create a project'. The second card has a purple icon of two overlapping circles with faces and is titled 'Create a group'. The third card has a purple icon of a person's head and shoulders and is titled 'Add people'. The fourth card has an orange icon of a gear with a flower inside and is titled 'Configure GitLab'.


**GitLab** Projects Groups Activity Milestones Snippets

Search or jump to...

Projects


## Welcome to GitLab

Code, test, and deploy together




### Create a project

Projects are where you store your code, access issues, wiki and other features of GitLab.




### Create a group

Groups are a great way to organize projects and people.



### Add people

Add your team members and others to GitLab.

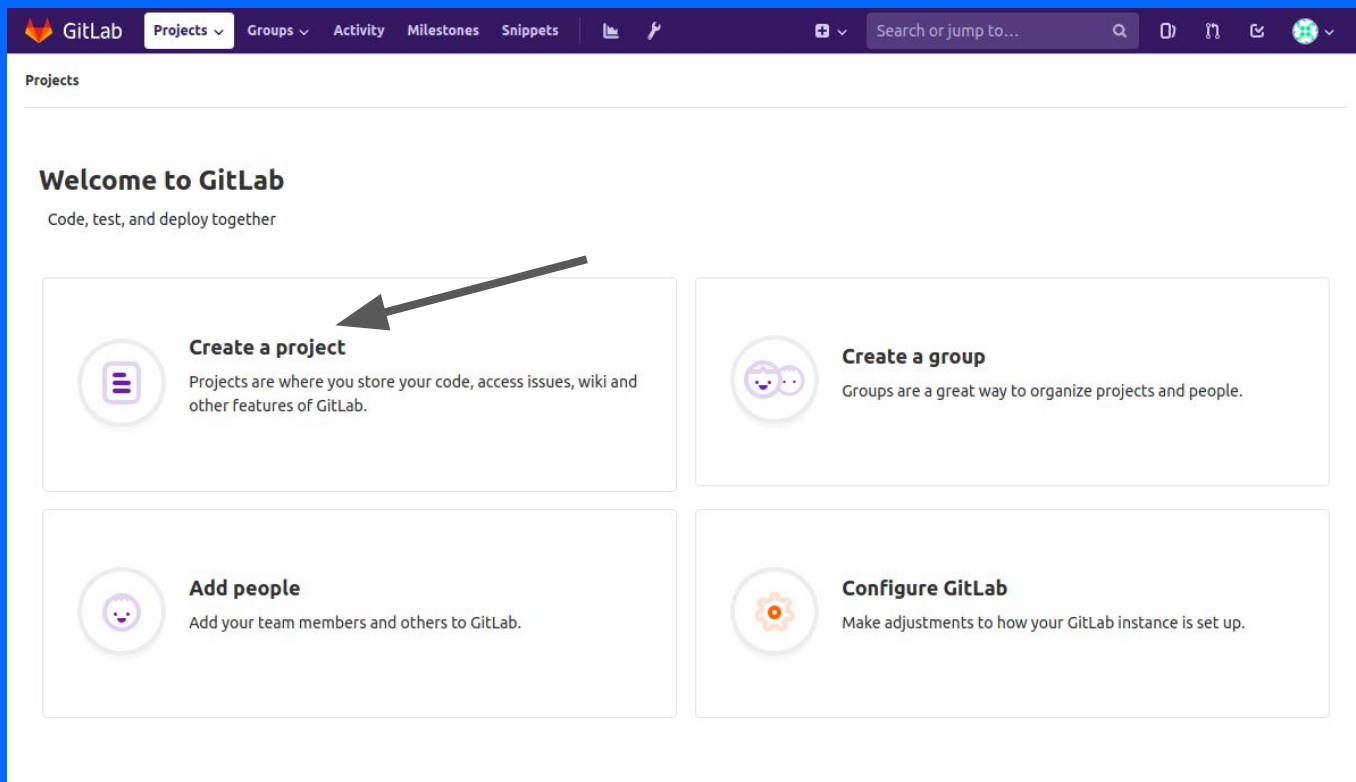


### Configure GitLab










Make adjustments to how your GitLab instance is set up.

Meetups

# Creamos nuestro primer proyecto en GitLab



# Creamos nuestro primer proyecto en GitLab

 **GitLab** Projects Groups Activity Milestones Snippets    Search or jump to...     

Projects

## New project

A project is where you house your files (repository), plan your work (issues), and publish your documentation (wiki), among other things.

All features are enabled for blank projects, from templates, or when importing, but you can disable them afterward in the project settings.

**Tip:** You can also create a project from the command line. [Show command](#)

Blank project>Create from template>Import project

**Project name**

Test

**Project URL**

http://gitlab.example.com:9090/ root


**Project slug**

test

Want to house several dependent projects under the same namespace? [Create a group](#).

**Project description (optional)**

Description format

**Visibility Level** 

☒ Private

Project access must be granted explicitly to each user.

☐ Internal

The project can be accessed by any logged in user.

☐ Public

The project can be accessed without any authentication.

☐ Initialize repository with a README

Allows you to immediately clone this project's repository. Skip this if you plan to push up an existing repository.

Create projectCancel

Meetups

# Creamos nuestro primer proyecto en GitLab

The screenshot displays the GitLab web interface for a newly created project named 'Test'. The interface includes a top navigation bar with the GitLab logo and various menu items like Projects, Groups, Activity, Milestones, and Snippets. A left sidebar contains a 'Project' section with options like Details, Activity, and Cycle Analytics, as well as other project management tools like Issues, Merge Requests, CI/CD, Operations, Registry, Wiki, Snippets, and Settings. The main content area shows the project details for 'Test', which is private and has no license. It indicates that the repository is empty and provides instructions on how to push files using command-line instructions. It also mentions that the master branch is automatically protected and that Auto DevOps is enabled. At the bottom, there are buttons for 'New file', 'Add Readme', and 'Add Kubernetes cluster', along with a section for 'Command line instructions' showing the necessary git configuration commands.

GitLab Projects Groups Activity Milestones Snippets

Search or jump to...

**T Test**

You won't be able to pull or push project code via SSH until you [add an SSH key](#) to your profile. [Don't show again](#) | [Remind later](#)

The Auto DevOps pipeline has been enabled and will be used if no alternative CI configuration file is found. [More information](#) [Settings](#) | [Dismiss](#)

Administrator > Test > Details

Project "Test" was successfully created.

**T Test** Private Add license

Project ID: 1

0 ☆ Star HTTP http://gitlab.example. + Global

### The repository for this project is empty

If you already have files you can push them using the [command line instructions](#) below.

*Note that the master branch is automatically protected. [Learn more about protected branches](#)*

You can automatically build and test your application if you [enable Auto DevOps](#) for this project. You can automatically deploy it as well, if you add a [Kubernetes cluster](#).

Otherwise it is recommended you start with one of the options below.

Files (0 Bytes) Commits (0) Branches (0) Tags (0) Auto DevOps enabled

New file Add Readme Add Kubernetes cluster

### Command line instructions

#### Git global setup

```
git config --global user.name "Administrator"
git config --global user.email "admin@example.com"
```

# Creamos nuestro primer proyecto en GitLab

You won't be able to pull or push project code via SSH until you **add an SSH key** to your profile

[Don't show again](#) | [Remind later](#)

- Al igual que en Github, en lugar de interactuar con git con el usuario y contraseña, se debe crear una llave SSH y asociarla a tu perfil en GitLab.
- De no tener clave asociada, puedes interactuar con tu usuario y contraseña, aunque no es un método recomendado.

The Auto DevOps pipeline has been enabled and will be used if no alternative CI configuration file is found. [More information](#)

[Settings](#) | [Dismiss](#)

- GitLab tiene un Auto DevOps: De configurar un cluster de Kubernetes, GitLab puede encargarse de las acciones de CI/CD automáticamente.
- Nosotros configuraremos manualmente el comportamiento de Gitlab CI/CD

# Gitlab: Agregando una SSH Key a nuestra cuenta

User Settings > SSH Keys

## SSH Keys

SSH keys allow you to establish a secure connection between your computer and GitLab.

### Add an SSH key

To add an SSH key you need to [generate one](#) or use an existing key.

### Key

Paste your public SSH key, which is usually contained in the file '~/.ssh/id\_rsa.pub' and begins with 'ssh-rsa'. Don't use your private SSH key.

Typically starts with "ssh-rsa ..."

### Title

e.g. My MacBook key

Name your individual key via a title

Add key

### Your SSH keys (0)

There are no SSH keys with access to your account.

- Podemos utilizar una SSH key ya existente en nuestro equipo o generar una nueva.
- Podemos hacer clic en cualquiera de los enlaces de ayuda para ver cómo generar la clave.
- El correo de la cuenta es **admin@example.com**



# Gitlab: Agregando una SSH Key a nuestra cuenta

```
~/devfest/test$ ssh-keygen -o -t rsa -C "admin@example.com" -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (/home/rolando/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/rolando/.ssh/id_rsa.
Your public key has been saved in /home/rolando/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:QMkF/HDQOhYrz5LtLqmMPjmfjavMm80cGJ8LDoHbDU4 admin@example.com
The key's randomart image is:
+---[RSA 4096]-----+
|      o+*.          |
|      .* o          |
|      .B            |
|.    . =..          |
|o.E  B .S          |
|  *+o+ +           |
|oo+++.+            |
|==** O .           |
|ooo@.+.            |
+---[SHA256]-----+
```

# Gitlab: Agregando una SSH Key a nuestra cuenta

```
~/devfest/test$ vim $HOME/.ssh/id_rsa.pub
```

```
ssh-rsa
```

```
AAAAB3NzaC1yc2EAAAADAQABAAQCAQC6phecjEP4Id/SMYqJ7kJkZsJ0Q4L2KS+VeC8D7SQ3ktTgHlAk8rvQOf1w3pLKF  
YQ8YLV+ZWoS cu9jaGPAd1vOOMggx6EZfzS4igBZwvAGsbMK1X8wLldBJVZcK8VHqd/ IHoh92N9X42VJzEVB0abyC/H8UT  
F9MNAJyq/qSaPaGv8eD2ln4C57nhZzjQvtJGpiS1usze46RIMyhr9igVNRMDYnG+iy+PcRLG1SNpq2SETh2ZuqlhPKTgh  
ZgzoUMlHiodkeZ1Mh6TNoTKWwQEajKRnyF/4s7dTf2nm3K2cyAM1tCjeF3RXmtOgoXTCfLzcRddZNAiMm6lR2I+nzwuK  
3+9bIpJOw8d5MbEW14S+2SYT2+IVNzQtvNfFSFScs3Cj+gr3cb/WIiTUC12nhj2VHrENuJLaGV2naZWWTty2e0jgetkV  
OgISbRz5CYZsoFDgLeL1qjtRdZlwAcREAEaleum4JzYs85wPw2kuq6QitH8LsDglZeXvar4DWFCJtpvHM58+j6m+eLK3A  
QR4QJu08id7olPwfGZr9xR5jE4u27c1JlBJOr7dMc/ajpxiD/UGm5CkHIVpt8fKJV1BvRbkUqrU7VsphlbXHZYJeRQXO/  
mZhMEBbKETI5stmH9vzoayAubc3IX9CVTgeINMd7+SNhyn4JHb8jl cQcJ+Ll9yQ== admin@example.com
```

# Gitlab: Agregando una SSH Key a nuestra cuenta

## Add an SSH key

To add an SSH key you need to generate one or use an existing key.

### Key

Paste your public SSH key, which is usually contained in the file '~/.ssh/id\_rsa.pub' and begins with 'ssh-rsa'. Don't use your private SSH key.

```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAQCAQC6phcejEP4Id/SMYqJ7kJKzsJ0Q4L2KS+VeC8D7SQ3k
tTgHlAk8rvQOf1w3pLKFYQ8YLV+ZWoS5cu9jaGPAd1vOOMMgq6EZfzS4igBZwvAGsbMK1X8wLld
BJVZcK8VHqd/IHoh92N9X42VJzEVB0abyC/H8UTF9MNAJyq/qSaPaGv8eD2ln4C57nhZzJQtJGpi
S1usze46RIMyhr9igVNRMDYnG+iy+PcRLGlsNpq2SETh2ZuqlhPKTghZqzoUMLHiodkeZ1Mh6TNoT
KWwQEajKRnyF/4s7dTf2nm3K2cyaM1tCjeF3RXmtOgoXTCfLzcRddZNaiMm6lR2l+nzwuK3+9bl
pJOW8d5MbEW14S+2SYT2+IVNzQvNfF5F5cs3Cj+gr3cb/WliTUC12nhj2VHrENuJLaGV2naZWW
CTty2e0jgetkVOglSbRz5CYZsoFDgLeL1qjtRdZlwAcREAEaleum4JzYs85wPw2kuq6QitH8LsDglZ
eXvar4DWFCJtpvHM58+j6m+eLK3AQR4QJu08id7olPwfGZr9xR5JE4u27c1JlBJOr7dMc/ajpxiD/U
Gm5CkHlVpt8fKJV1BvRbkUqrU7VsphlbXHZYJeRQXO/mZhMEBbKETl5stmH9vzoayAubc3lX9Cv
TgeINMd7+SNhyn4JHb8j1cQCj+Ll9yQ== admin@example.com
```

### Title

Name your individual key via a title

Add key

### Your SSH keys (0)

There are no SSH keys with access to your account.

- Copiamos la clave pública y la pegamos en el área de texto.
- Automáticamente se rellena el título y se activa el botón para agregar la clave.
- Si hemos creado la clave con contraseña, cada vez que interactuemos con el repositorio deberemos introducirla.

# Gitlab: Agregando una SSH Key a nuestra cuenta

User Settings > SSH Keys > admin@example.com

SSH Key

Title: admin@example.com

Created on: Nov 11, 2018 8:13pm

Last used on: N/A

Fingerprint: 45:00:70:fa:8b:37:c9:85:a4:53:cf:bd:86:84:aa:2b

```
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQCAQC6phecjEP4Id/SMYqJ7kJkZsJ0Q4L2KS+VeC8D7SC
```

Remove

# *Gitlab: Agregando una SSH Key a nuestra cuenta*



Tu Proyecto => Tus Dockers

# Gitlab Registry

## Container Registry

With the Docker Container Registry integrated into GitLab, every project can have its own space to store its Docker images.

Learn more about [Container Registry](#).

No container images stored for this project. Add one by following the instructions above.

### How to use the Container Registry

First log in to GitLab's Container Registry using your GitLab username and password. If you have 2FA enabled you need to use a [personal access token](#):

```
docker login gitlab.example.com:4567
```

You can also use a [deploy token](#) for read-only access to the registry images.

Once you log in, you're free to create and upload a container image using the common `build` and `push` commands

```
docker build -t gitlab.example.com:4567/root/test .  
docker push gitlab.example.com:4567/root/test
```

### Use different image names

GitLab supports up to 3 levels of image names. The following examples of images are valid for your project:

```
gitlab.example.com:4567/root/test:tag  
gitlab.example.com:4567/root/test/optional-image-name:tag  
gitlab.example.com:4567/root/test/optional-name/optional-image-name:tag
```

Meetups

# Gitlab Registry: Imágenes para nuestro proyecto

```
~/devFestWorkshop/workshop-gitlab$ cd ..
```

```
~/devFestWorkshop$ vim Dockerfile
```

```
FROM alpine:latest
```

```
RUN apk add -U git
```

```
~/devFestWorkshop$ docker login gitlab.example.com:4567
```

```
Username: root
```

```
Password:
```

```
Login Succeeded
```

```
~/devFestWorkshop$ docker build -t gitlab.example.com:4567/root/test/git-image:test .
```

```
Sending build context to Docker daemon 63.01MB
```

```
Step 1/2 : FROM alpine:latest
```

```
---> 196d12cf6ab1
```

```
Step 2/2 : RUN apk add -U git
```

```
---> Using cache
```

```
---> a1278ba9aa87
```

```
Successfully built a1278ba9aa87
```

```
Successfully tagged gitlab.example.com:4567/root/test/git-image:test
```



# Gitlab Registry: Imágenes para nuestro proyecto

```
~/devFestWorkshop$ docker push gitlab.example.com:4567/root/test/git-image:test
The push refers to repository [gitlab.example.com:4567/root/test/git-image]
094f06112dcc: Pushed
df64d3292fd6: Pushed
test: digest: sha256:4496ddd3cd8201461b506631e2cf92bc943392253b5fcd52a6e6a2c7f4ddcb6c size:
739
```

# Gitlab Registry

Administrator > Test > Container Registry



## Container Registry

With the Docker Container Registry integrated into GitLab, every project can have its own space to store its Docker images.

Learn more about [Container Registry](#).

[^root/test/git-image](#) 



Tag	Tag ID	Size	Created	
test 	a1278ba9a	10.80 MiB	1 day ago	

### How to use the Container Registry

First log in to GitLab's Container Registry using your GitLab username and password. If you have [2FA enabled](#) you need to use a [personal access token](#):

```
docker login gitlab.example.com:4567
```

You can also use a [deploy token](#) for read-only access to the registry images.

Once you log in, you're free to create and upload a container image using the common `build` and `push` commands

```
docker build -t gitlab.example.com:4567/root/test .
```

Meetups

# Gitlab Registry: Imágenes para nuestro proyecto

```
$ docker run -it gitlab.example.com:4567/root/test/git-image:test sh
/ # git --version
git version 2.18.1
/ # exit
$ docker run -it gitlab.example.com:4567/root/test/git-image:test git --version
git version 2.18.1
```

# *Gitlab Registry: Imágenes para nuestro proyecto*



*Meetups*

# La integración continua

# Gitlab CI/CD

## Specific Runners

### Set up a specific Runner automatically

You can easily install a Runner on a Kubernetes cluster.  
[Learn more about Kubernetes](#)

1. Click the button below to begin the install process by navigating to the Kubernetes page
2. Select an existing Kubernetes cluster or create a new one
3. From the Kubernetes cluster details view, install Runner from the applications list

Install Runner on Kubernetes

### Set up a specific Runner manually

1. Install GitLab Runner
2. Specify the following URL during the Runner setup:  
`http://gitlab.example.com:9090/`
3. Use the following registration token during setup:  
`qTxk0f5Gc7xqd31y39`

Reset runners registration token

4. Start the Runner!

## Shared Runners

GitLab Shared Runners execute code of different projects on the same Runner unless you configure GitLab Runner Autoscale with MaxBuilds 1 (which it is on GitLab.com).

Disable shared Runners for this project

This GitLab instance does not provide any shared Runners yet. Instance administrators can register shared Runners in the admin area.

## Group Runners

GitLab Group Runners can execute code for all the projects in this group. They can be managed using the Runners API.

This project does not belong to a group and can therefore not make use of group Runners.

- Settings => CI/CD
- Expandimos "Runners"
- Set up a specific Runner manually
- Los runners los creamos en las máquinas que tengan instalado gitlab-runner
- Necesitamos copiar la URL y el token para usarla al configurar el runner.

# Gitlab CI/CD

## Specific Runners

### Set up a specific Runner automatically

You can easily install a Runner on a Kubernetes cluster.  
[Learn more about Kubernetes](#)

1. Click the button below to begin the install process by navigating to the Kubernetes page
2. Select an existing Kubernetes cluster or create a new one
3. From the Kubernetes cluster details view, install Runner from the applications list

Install Runner on Kubernetes

### Set up a specific Runner manually

1. Install GitLab Runner
2. Specify the following URL during the Runner setup:  
`http://gitlab.example.com:9090/`
3. Use the following registration token during setup:  
`qTxkQf56C7xqd31y39`

Reset runners registration token

4. Start the Runner!

## Shared Runners

GitLab Shared Runners execute code of different projects on the same Runner unless you configure GitLab Runner Autoscale with MaxBuilds 1 (which it is on GitLab.com).

Disable shared Runners for this project

This GitLab instance does not provide any shared Runners yet. Instance administrators can register shared Runners in the admin area.

## Group Runners

GitLab Group Runners can execute code for all the projects in this group. They can be managed using the Runners API.

This project does not belong to a group and can therefore not make use of group Runners.

- Settings => CI/CD
- Expandimos "Runners"
- Set up a specific Runner manually
- Los runners los creamos en las máquinas que tengan instalado gitlab-runner
- Necesitamos copiar la URL y el token para usarla al configurar el runner.

# Gitlab CI/CD: Los runners

- Los runners son los encargados de ejecutar las instrucciones que determinados para la integración y despliegue de nuestras aplicaciones.



# Gitlab CI/CD: Los runners

- Los runners son los encargados de ejecutar las instrucciones que determinados para la integración y despliegue de nuestras aplicaciones.
- Las instrucciones las almacenamos en un fichero `.gitlab-ci.yml` en la raíz del proyecto.

# Gitlab CI/CD: Los runners

- Los runners son los encargados de ejecutar las instrucciones que determinados para la integración y despliegue de nuestras aplicaciones.
- Las instrucciones las almacenamos en un fichero `.gitlab-ci.yml` en la raíz del proyecto.
- Crearemos un runner con el modo interactivo y otro con el “modo comando”.

# Gitlab CI/CD: Los runners

- Los runners son los encargados de ejecutar las instrucciones que determinados para la integración y despliegue de nuestras aplicaciones.
- Las instrucciones las almacenamos en un fichero `.gitlab-ci.yml` en la raíz del proyecto.
- Crearemos un runner con el modo interactivo y otro con el “modo comando”.
- Empezaremos con runners con Docker como executor.
- Utilizaremos el “socket binding” para montar el socket de Docker de nuestra máquina en el contenedor del gitlab-runner, lo que habilitará Docker en el contexto del contenedor.

# Gitlab CI/CD: Los runners

- Los runners son los encargados de ejecutar las instrucciones que determinados para la integración y despliegue de nuestras aplicaciones.
- Las instrucciones las almacenamos en un fichero `.gitlab-ci.yml` en la raíz del proyecto.
- Crearemos un runner con el modo interactivo y otro con el “modo comando”.
- Empezaremos con runners con Docker como executor.
- Utilizaremos el “socket binding” para montar el socket de Docker de nuestra máquina en el contenedor del gitlab-runner, lo que habilitará Docker en el contexto del contenedor.
- Docker y Kubernetes son los executors recomendados, pero el executor shell es el utilizado en los entornos más tradicionales... los servidores de toda la vida.

# Gitlab CI/CD: Mi primer runner

```
$ cd workshop-gitlab/  
$ docker-compose exec gitlab-runner bash  
bash-4.4# gitlab-runner register  
Runtime platform arch=amd64 os=linux pid=28 revision=cf91d5e1 version=11.4.2  
Running in system-mode.  
Please enter the gitlab-ci coordinator URL (e.g. https://gitlab.com/):  
http://gitlab.example.com/  
Please enter the gitlab-ci token for this runner:  
qTxkQf5GC7xqd31y39  
Please enter the gitlab-ci description for this runner:  
[5b41c71326d3]: myFirstRunner  
Please enter the gitlab-ci tags for this runner (comma separated):  
Registering runner... succeeded runner=qTxkQf5G  
Please enter the executor: parallels, ssh, docker+machine, docker-ssh+machine, kubernetes,  
docker-ssh, shell, virtualbox, docker:  
docker  
Please enter the default Docker image (e.g. ruby:2.1):  
docker:stable  
Runner registered successfully. Feel free to start it, but if it's running already the  
config should be automatically reloaded!
```

# Gitlab CI/CD: Mi segundo runner

```
bash-4.4# gitlab-runner register -n --url http://gitlab.example.com:9090/  
--registration-token qTxkQf5GC7xqd31y39 --executor docker  
--description "My Second Runner" --docker-image "docker:stable"  
--docker-volumes /var/run/docker.sock:/var/run/docker.sock
```

```
Runtime platform          arch=amd64 os=linux pid=45 revision=cf91d5e1 version=11.4.2  
Running in system-mode.
```

```
Registering runner... succeeded                runner=qTxkQf5G  
Runner registered successfully. Feel free to start it, but if it's running  
lijjji99ju88uhby7hnjiim.jjk, already the config should be automatically reloaded!  
bash-4.4# exit  
exit  
~/devFestWorkshop/workshop-gitlab$ cd runner/config/  
~/devFestWorkshop/workshop-gitlab/runner/config$ ls  
config.toml  
~/devFestWorkshop/workshop-gitlab/runner/config$ sudo vim config.toml
```

# Gitlab CI/CD: Mi segundo runner

```
concurrent = 1
check_interval = 0

[session server]
  session_timeout = 1800

[[runners]]
  name = "myFirstRunner"
  url = "http://gitlab.example.com:9090/"
  token = "915887b623a59fe7775285f55347a2"
  executor = "docker"
[runners.docker]
  tls_verify = false
  image = "docker:stable"
  privileged = false
  disable_entrypoint_overwrite = false
  oom_kill_disable = false
  disable_cache = false
  volumes = ["/cache"]
  shm_size = 0
[runners.cache]
  [runners.cache.s3]
  [runners.cache.gcs]
```

# Gitlab CI/CD: Mi segundo runner

```
[[runners]]
  name = "My Second Runner"
  url = "http://gitlab.example.com:9090/"
  token = "bfc2b21f7674f52260d6904a3a59fa"
  executor = "docker"
[runners.docker]
  tls verify = false
  image = "docker:stable"
  privileged = false
  disable entrypoint overwrite = false
  oom kill disable = false
  disable cache = false
  volumes = ["/var/run/docker.sock:/var/run/docker.sock", "/cache"]
  shm size = 0
[runners.cache]
  [runners.cache.s3]
  [runners.cache.gcs]
```



## *Gitlab CI/CD: Los runners*

OJO! El runner creado de forma interactiva no tiene montado el socket de Docker...

... tenemos que agregarlo a la entrada de los volúmenes

# Gitlab CI/CD: Los runners

```
concurrent = 1
check_interval = 0

[session server]
  session_timeout = 1800

[[runners]]
  name = "myFirstRunner"
  url = "http://gitlab.example.com:9090/"
  token = "915887b623a59fe7775285f55347a2"
  executor = "docker"
  [runners.docker]
    tls_verify = false
    image = "docker:stable"
    privileged = false
    disable_entrypoint_overwrite = false
    oom_kill_disable = false
    disable_cache = false
    volumes = ["/var/run/docker.sock:/var/run/docker.sock", "/cache"]
    shm_size = 0
  [runners.cache]
    [runners.cache.s3]
    [runners.cache.gcs]
```

# Gitlab CI/CD: Los runners

## Set up a specific Runner manually

1. Install GitLab Runner
2. Specify the following URL during the Runner setup:  
`http://gitlab.example.com:9090/`
3. Use the following registration token during setup:  
`qTxkQf5GC7xqd31y39`

Reset runners registration token

4. Start the Runner!

in this group. They can be managed using the Runners API.

This project does not belong to a group and can therefore not make use of group Runners.

## Runners activated for this project

 bfc2b21f  

Pause Remove Runner

My Second Runner

#2

 915887b6  

Pause Remove Runner

myFirstRunner

#1

# *Gitlab CI/CD: Los runners*



*Meetups*

# *Gitlab CI/CD: Probandos los runners*

Para probar si nuestros runners funcionan correctamente vamos a hacer lo siguiente:

1. Clonar el proyecto test en nuestro equipo
2. Crear un fichero .gitlab-ci.yml donde definimos las acciones del runner
3. Crear un Dockerfile que será utilizado para crear la imagen Docker del proyecto.
4. Subir los cambios.
5. Ver resultados.

# Gitlab CI/CD: Probandos los runners

```
$ cd $HOME/devFestWorkshop
$ git clone ssh://git@gitlab.example.com/root/test.git
  Clonando en 'test'...
  warning: Pareces haber clonado un repositorio sin contenido.
$ cd test
$ vim Dockerfile
FROM alpine:latest
RUN apk add -U git
$ vim .gitlab-ci.yml
```

Si no tienes una clave SSH agregada a tu perfil en gitlab, en lugar de hacer el clone vía SSH tiene que ser vía HTTP:

```
git clone http://gitlab.example.com/root/test.git
```

# Gitlab CI/CD: Probandos los runners

```
image: docker:stable

before script:
  - docker login -u gitlab-ci-token -p $CI_JOB_TOKEN gitlab.example.com:4567

build:
  stage: build
  script:
    - docker build -t test-image .
    - docker tag test-image gitlab.example.com:4567/root/test:runner
    - docker push gitlab.example.com:4567/root/test:runner
```

# Gitlab CI/CD: Probando los runners


```
$ git add .
$ git commit -m "Probando los runners"
[master (commit-raíz) 2c70d89] Probando los runners
 2 files changed, 13 insertions(+)
 create mode 100644 .gitlab-ci.yml
 create mode 100644 Dockerfile



$ git push
Enumerando objetos: 4, listo.
Contando objetos: 100% (4/4), listo.
Compresión delta usando hasta 4 hilos
Comprimiendo objetos: 100% (3/3), listo.
Escribiendo objetos: 100% (4/4), 470 bytes | 470.00 KiB/s, listo.
Total 4 (delta 0), reusado 0 (delta 0)
To ssh://gitlab.example.com/root/test.git
 * [new branch]      master -> master
```



# Gitlab CI/CD: Probando los runners


master test / + History Find file Web IDE



 **Probando los runners**  
Administrator authored 4 minutes ago  2c70d897

Name	Last commit	Last update
 .gitlab-ci.yml	Probando los runners	4 minutes ago
 Dockerfile	Probando los runners	4 minutes ago

# Gitlab CI/CD: Probando los runners

master test / + History Find file Web IDE

 **Probando los runners**  
Administrator authored 5 minutes ago 2c70d897

Name	Last commit	Last update
 .gitlab-ci.yml	Probando los runners	5 minutes ago
 Dockerfile	Probando los runners	5 minutes ago

# *Gitlab CI/CD: Probandos los runners*



# Gitlab CI/CD: Probandos los runners

Administrator > Test > Jobs > #1

Failed

Job #1 triggered 1 minute ago by Administrator

New issue

Running with gitlab-runner 11.4.2 (cf91d5e1)  
on My Second Runner bfc2b21f  
Using Docker executor with image docker:stable ...  
Pulling docker image docker:stable ...  
Using docker image sha256:062267097b77e3ecf374b437e93fefe2bbb2897da989f930e4750752ddfc822a for docker:stable ...  
Running on runner-bfc2b21f-project-1-concurrent-0 via 5b41c71326d3...  
Cloning repository...  
Cloning into '/builds/root/test'...  
fatal: unable to access 'http://gitlab-ci-token:xxxxxxxxxxxxxxxx@gitlab.example.com:9090/root/test.git/': Could not resolve host: gitlab.example.com  
/bin/bash: line 63: cd: /builds/root/test: No such file or directory  
ERROR: Job failed: exit code 1

build

Retry

Duration: 11 seconds  
Timeout: 1h (from project)  
Runner: My Second Runner (#2)

Commit 2c70d897

Probandos los runners

Pipeline #1 from master

build

→ build

*Gitlab CI/CD: Los runners*

KEEP CALM!!

## *Gitlab CI/CD: Los runners*

Estamos ante un problema de “Docker in Docker”...  
Debemos especificarle a nuestro runner la red a utilizar.

docker network ls

# Gitlab CI/CD: Probandos los runners

```
$sudo vim $HOME/devFestWorkshop/workshop-gitlab/runner/config/config.toml
concurrent = 1
check_interval = 0

[session server]
  session_timeout = 1800

[[runners]]
  name = "myFirstRunner"
  url = "http://gitlab.example.com:9090/"
  token = "915887b623a59fe7775285f55347a2"
  executor = "docker"
[runners.docker]
  tls_verify = false
  image = "docker:stable"
  privileged = false
  disable_entrypoint_overwrite = false
  oom_kill_disable = false
  disable_cache = false
  volumes = ["/var/run/docker.sock:/var/run/docker.sock", "/cache"]
  shm_size = 0
  network_mode = "workshop-gitlab_gitlabWorkshop"
[runners.cache]
  [runners.cache.s3]
  [runners.cache.gcs]
```

# Gitlab CI/CD: Probandos los runners

```
[[runners]]
  name = "My Second Runner"
  url = "http://gitlab.example.com:9090/"
  token = "bfc2b21f7674f52260d6904a3a59fa"
  executor = "docker"
[runners.docker]
  tls verify = false
  image = "docker:stable"
  privileged = false
  disable entrypoint overwrite = false
  oom kill disable = false
  disable cache = false
  volumes = ["/var/run/docker.sock:/var/run/docker.sock", "/cache"]
  shm size = 0
  network mode = "workshop-gitlab_gitlabWorkshop"
[runners.cache]
  [runners.cache.s3]
  [runners.cache.gcs]
```



# Gitlab CI/CD: Probandos los runners

Administrator > Test > Jobs > #1

Failed Job #1 triggered 1 minute ago by Administrator New issue

```
Running with gitlab-runner 11.4.2 (cf91d5e1)
  on My Second Runner bfc2b21f
Using Docker executor with image docker:stable ...
Pulling docker image docker:stable ...
Using docker image sha256:062267097b77e3ecf374b437e93fefe2bbb2897da989f930e4750752ddfc822a for docker:stable ...
Running on runner-bfc2b21f-project-1-concurrent-0 via 5b41c71326d3...
Cloning repository...
Cloning into '/builds/root/test'...
fatal: unable to access 'http://gitlab-ci-token:xxxxxxxxxxxxxxxx@gitlab.example.com:9090/root/test.git/': Could not resolve host: gitlab.example.com
/bin/bash: line 63: cd: /builds/root/test: No such file or directory
ERROR: Job failed: exit code 1
```

**build** Retry

Duration: 11 seconds  
Timeout: 1h (from project)  
Runner: My Second Runner (#2)

Commit 2c70d897

Probandos los runners

✖ Pipeline #1 from master

build

→ ✖ build

# Gitlab CI/CD: Probandos los runners

Administrator > Test > Jobs > #2

passed

Job #2 triggered 37 minutes ago by Administrator

```
Running with gitlab-runner 11.4.2 (cf91d5e1)
  on myFirstRunner 915887b6
Using Docker executor with image docker:stable ...
Pulling docker image docker:stable ...
Using docker image sha256:062267897b77e3ecf374b437e93fefe2bbb2897da909f930e4750752ddfc822a for docker:stable ...
Running on runner-915887b6-project-1-concurrent-0 via Sb41c71326d3...
Cloning repository...
Cloning into '/builds/root/test'...
Checking out 2c70d897 as master...
Skipping Git submodules setup
$ docker login -u gitlab-ci-token -p $CI_JOB_TOKEN gitlab.example.com:4567
WARNING! Using --password via the CLI is insecure. Use --password-stdin.
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
$ docker build -t test-image .
Sending build context to Docker daemon 25.6kB

Step 1/2 : FROM alpine:latest
--> 196d12cf6ab1
Step 2/2 : RUN apk add -U git
--> Using cache
--> a1278ba9aa87
Successfully built a1278ba9aa87
Successfully tagged test-image:latest
$ docker tag test-image gitlab.example.com:4567/root/test:runner
$ docker push gitlab.example.com:4567/root/test:runner
The push refers to repository [gitlab.example.com:4567/root/test]
094f96112dcc: Preparing
df64d3292fd6: Preparing
df64d3292fd6: Mounted from root/test/git-image
094f96112dcc: Mounted from root/test/git-image
runner: digest: sha256:4496ddd3cd8201461b506631e2cf92bcb943392253b5fcd52a6e6a2c7f4ddcb6c size: 739
Job succeeded
```

build

Retry

Duration: 22 seconds  
Timeout: 1h (from project)  
Runner: myFirstRunner (#1)

Commit 2c70d897

Probandos los runners

Pipeline #1 from master

build

→ build

build

# Gitlab CI/CD: Probandos los runners

Administrator > Test > Container Registry

## Container Registry

With the Docker Container Registry integrated into GitLab, every project can have its own space to store its Docker images.



Learn more about [Container Registry](#).

> [root/test/git-image](#) 



^ [root/test](#) 



Tag	Tag ID	Size	Created	
runner 	a1278ba9a	10.80 MiB	1 day ago	

# *Gitlab CI/CD: Probandos los runners*



# *Gitlab CI/CD*

**THE END**

# *Gitlab CI/CD*



SÓLO HEMOS CALENTADO

# Stages, Jobs & Environments



# Gitlab CI/CD

Hasta ahora sólo hemos usado un stage (build) y un job (build).

Podemos tener múltiples stages y varios jobs por build.

# Gitlab CI/CD: Los stages

Un stage es un escenario dentro del proceso de CI/CD.

Un stage se compone de diferentes jobs.

Los stages se ejecutan de forma secuencial.

Cando un stage falla, se detiene el proceso de CI/CD.

## *Gitlab CI/CD: Los jobs*

Un jobs es una tarea concreta dentro de un stage.

Se pueden definir tantos Jobs por stage como se desee.

Los jobs se ejecutan de forma paralela dentro del mismo stage.

Cuando un build falla, el stage al que pertenece falla.

# Gitlab CI/CD: Adaptando .gitlab-ci.yml

~/devFestWorkshop/test\$ vim .gitlab-ci.yml

```
stages:
  - build
  - test
  - deploy
before_script:
  - docker login -u gitlab-ci-token -p $CI_JOB_TOKEN gitlab.example.com:4567
test-build:
  stage: build
  image: docker:stable
  script:
    - docker build -t test-image .
    - docker tag test-image gitlab.example.com:4567/root/test:runner
    - docker push gitlab.example.com:4567/root/test:runner
test-test:
  stage: test
  script:
    - echo "fin"
test-deploy:
  stage: deploy
  script:
    - echo "fin"
```

# Gitlab CI/CD: Adaptando .gitlab-ci.yml

 **passed** Pipeline #5 triggered 20 seconds ago by  Administrator

## test stages

 3 jobs from master


 0ff6c08d  

Pipeline Jobs 3


Build

Test


Deploy

 test-build



 test-test



 test-deploy



# Gitlab CI/CD: Adaptando .gitlab-ci.yml

```
stages:
  - test
  - build
  - deploy
before script:
  - docker login -u gitlab-ci-token -p $CI_JOB_TOKEN gitlab.example.com:4567
tests-unit:
  stage: test
  script:
    - echo "fin"
test-build:
  stage: build
  image: docker:stable
  script:
    - docker build -t test-image .
    - docker tag test-image gitlab.example.com:4567/root/test:runner
    - docker push gitlab.example.com:4567/root/test:runner
test-behat:
  stage: test
  script:
    - echo "fin"
test-deploy:
  stage: deploy
  script:
    - echo "fin"
```

# Gitlab CI/CD: Stages y Jobs

 **passed** Pipeline #7 triggered 1 minute ago by  Administrator


## test stages

🕒 4 jobs from `master` in 59 seconds (queued for 1 second)

🔑 27dd1206  

Pipeline Jobs 4

### Test


 test-behat



 tests-unit




### Build

 test-build



### Deploy

 test-deploy



# *Gitlab CI/CD: Stages y Jobs*



*Meetups*



VAMOS A HACERLO MÁS...

VAMOS A HACERLO MÁS...  
...REAL

# Gitlab CI/CD: Proyecto PHP - Symfony4

Administrator > Test > Runners

## Runner #2

Active	<input checked="" type="checkbox"/> Paused Runners don't accept new jobs
Protected	<input type="checkbox"/> This runner will only run on pipelines triggered on protected branches
Run untagged jobs	<input checked="" type="checkbox"/> Indicates whether this runner can pick jobs without tags
Lock to current projects	<input type="checkbox"/> When a runner is locked, it cannot be assigned to other projects
Token	<input type="text" value="bfc2b21f7674f52260d6904a3a59fa"/>
IP Address	<input type="text" value="172.24.0.3"/>
Description	<input type="text" value="My Second Runner"/>
Maximum job timeout	<input type="text"/>
	This timeout will take precedence when lower than Project-defined timeout
Tags	<input type="text"/>
	You can set up jobs to only use Runners with specific tags. Separate tags with commas.

[Save changes](#)

- Editamos el Runner #2 de nuestro proyecto test.
- Desmarcamos “Lock to current projects”.
- Así, podemos asignar un runner a otros proyectos.

# Gitlab CI/CD: Proyecto PHP - Symfony4

Blank project

Create from template

Import project

Project name

Workshop Gitlab Backend

Project URL

http://gitlab.example.com:9090/ root

Project slug

workshop-gitlab-backend

Want to house several dependent projects under the same namespace? [Create a group.](#)

Project description (optional)

Description format

Visibility Level

☒ Private

Project access must be granted explicitly to each user.

☐ Internal

The project can be accessed by any logged in user.

☐ Public

The project can be accessed without any authentication.

☐ Initialize repository with a README

Allows you to immediately clone this project's repository. Skip this if you plan to push up an existing repository.

Create project

Cancel

- Creamos un nuevo proyecto.
- Lo llamamos “Workshop Gitlab Backend”.
- El namespace es “root” porque es nuestro usuario, podríamos crear un grupo “project”, asociarnos a él y utilizarlo como namespace.

# Gitlab CI/CD: Proyecto PHP - Symfony4

[Install Runner on Kubernetes](#)

### Set up a specific Runner manually

1. Install GitLab Runner
2. Specify the following URL during the Runner setup:  
`http://gitlab.example.com:9090/`
3. Use the following registration token during setup:  
`L-WsLx2pzJkH5o_4Mz8`
4. Start the Runner!

Reset runners registration token

### Group Runners

GitLab Group Runners can execute code for all the projects in this group. They can be managed using the [Runners API](#).

This project does not belong to a group and can therefore not make use of group Runners.

### Available specific runners

● bfc2b21f

My Second Runner

#2

Enable for this project

### Variables ?

Expand

- En el nuevo proyecto vamos a “Settings => CI/CD”.
- Clicamos en “Expand” en “Runners”.
- Vemos cómo “My Second Runner” está disponible para el proyecto.
- Lo activamos para el proyecto.

# Gitlab CI/CD: Proyecto PHP - Symfony4

**Test coverage parsing**

/  /

A regular expression that will be used to find the test coverage output in the job trace. Leave blank to disable ?

Below are examples of regex for existing tools:

- Simplecov (Ruby) - `\\(\\d+\\.\\d+\\%\\) covered`
- pytest-cov (Python) - `\\d+\\%\\s*$`
- phpunit --coverage-text --colors=never (PHP) - `^\\s*Lines:\\s*\\d+\\.\\d+\\%`
- gcovr (C/C++) - `^TOTAL\\. *\\s+(\\d+\\%)$`
- tap --coverage-report=text-summary (NodeJS) - `^Statements\\s*:\\s*([\\^%]+)`
- excoveralls (Elixir) - `\\[TOTAL\\]\\s+(\\d+\\.\\d+\\%)`
- JaCoCo (Java/Kotlin) `Total\\. *?(\\[0-9\\]{1,3}\\%)`
- go test -cover (Go) `coverage: \\d+\\.\\d+\\% of statements`

[Save changes](#)

- En el nuevo proyecto vamos a “Settings => CI/CD”.
- Clicamos en “Expand” en “General pipelines”.
- Cubrimos “Test coverage parsing” con la expresión regular indicada para PHP.

# Gitlab CI/CD: Proyecto PHP - Symfony4

```
$ cd $HOME/devFestWorkshop
$ git clone https://github.com/rolando-caldas/workshop-gitlab-backend
Clonando en 'workshop-gitlab-backend'...
remote: Enumerating objects: 246, done.
remote: Counting objects: 100% (246/246), done.
remote: Compressing objects: 100% (138/138), done.
remote: Total 246 (delta 88), reused 239 (delta 81), pack-reused 0
Recibiendo objetos: 100% (246/246), 47.36 KiB | 557.00 KiB/s, listo.
Resolviendo deltas: 100% (88/88), listo.
$ cd workshop-gitlab-backend
$ rm -fR .git
```

# Gitlab CI/CD: Proyecto PHP - Symfony4

```
$ git init
Inicializado repositorio Git vacío en
/home/rolando/devFestWorkshop/workshop-gitlab-backend/.git/

$ git remote add origin ssh://git@gitlab.example.com:22/root/workshop-gitlab-backend.git
$ vim .gitlab-ci.yml
```



# Gitlab CI/CD: Proyecto PHP - Symfony4

```
stages:
  - test
  - build
  - deploy

php-unit:
  stage: test
  image: rolandocaldas/php:7.2-dev-mysql
  script:
    - cd application
    - php -d memory_limit=4084M /usr/local/bin/composer update
    - ./bin/phpunit --coverage-text --colors=never

other-test:
  stage: test
  script:
    - echo "Nothing to do"
```

# Gitlab CI/CD: Proyecto PHP - Symfony4

```
test-build:
  stage: build
  image: docker:stable
  before script:
    - docker login -u gitlab-ci-token -p $CI_JOB_TOKEN gitlab.example.com:4567
  script:
    - docker build -t workshop-gitlab-backend .
    - docker tag workshop-gitlab-backend gitlab.example.com:4567/root/workshop-gitlab-backend
    - docker push gitlab.example.com:4567/root/workshop-gitlab-backend
test-deploy:
  stage: deploy
  script:
    - echo "fin"
```

# Gitlab CI/CD: Proyecto PHP - Symfony4

```
$ vim Dockerfile
FROM rolandocaldas/php:7.2-dev-mysql

ENV APP_ENV=prod



COPY application /application
RUN php -d memory_limit=-1 /usr/local/bin/composer install --no-dev --optimize-autoloader
RUN php -d memory_limit=-1 /application/bin/console cache:clear --env=prod --no-debug

RUN usermod -u 1000 www-data && groupmod -g 1000 www-data
```


# *Gitlab CI/CD: Proyecto PHP - Symfony4*




```
$ git add .  
$ git commit -m "Initial commit"  
$ git push -u origin master
```

# Gitlab CI/CD: Adaptando .gitlab-ci.yml

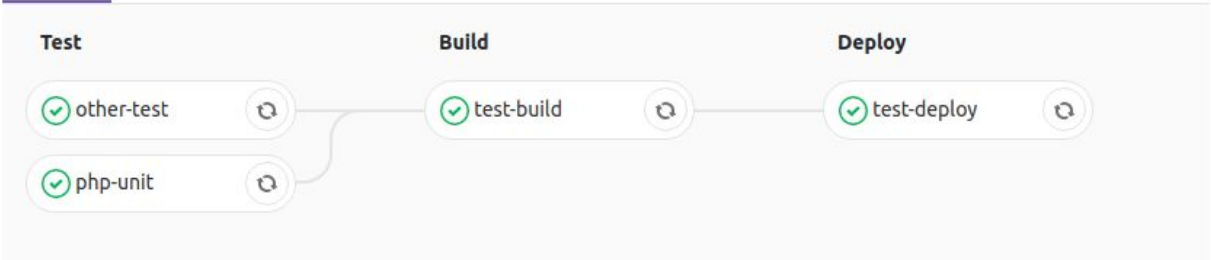
 **passed** Pipeline #14 triggered 19 hours ago by  Administrator

## Initial commit

 4 jobs from `master` in 35 minutes and 24 seconds (queued for 1 second)

 2dcf926a  

**Pipeline** Jobs 4



```
graph LR; subgraph Test; other-test[other-test] --- php-unit[php-unit]; end; subgraph Build; test-build[test-build]; end; subgraph Deploy; test-deploy[test-deploy]; end; other-test --> test-build; php-unit --> test-build; test-build --> test-deploy;
```

The pipeline consists of three stages: **Test**, **Build**, and **Deploy**. The **Test** stage contains two jobs: `other-test` and `php-unit`. Both jobs are successful (indicated by green checkmarks) and are connected to the `test-build` job in the **Build** stage. The `test-build` job is also successful and is connected to the `test-deploy` job in the **Deploy** stage. Each job has a circular arrow icon next to it, indicating it can be retried.

# Gitlab CI/CD: Adaptando .gitlab-ci.yml

Status	Job	Pipeline	Stage	Name	Coverage
passed	#72 Y master -> 5569f159	#19 by	deploy	test-deploy	00:14 10 minutes ago
passed	#71 Y master -> 5569f159	#19 by	build	test-build	00:23 10 minutes ago
passed	#70 Y master -> 5569f159	#19 by	test	other-test	00:14 10 minutes ago
passed	#69 Y master -> 5569f159	#19 by	test	php-unit	01:05 10 minutes ago 7.82%

# *Gitlab CI/CD: Stages y Jobs*



## *Gitlab CI/CD: Environments*

Ya hemos “trabajado” con el Job “build” en nuestro CI/CD

Sin embargo, normalmente necesitamos diferentes builds...

... por ejemplo, una demo en preProd y la copia de Producción...

... pudiendo ver su estado y hacer rollback de forma sencilla...



Y PARA ESO TENEMOS  
LOS ENVIRONMENTS

# Gitlab CI/CD: Activando Environment

```
$ cd $HOME/devFestWorkshop
$ vim .gitlab-ci.yml
[...]
test-deploy:
  stage: deploy
  script:
    - echo "fin"
  environment:
    name: production
    url: https://172.24.0.5

$ git add .gitlab-ci.yml
$ git commit -m "Environment production on"
$ git push
```







# Gitlab CI/CD: Activando Environment

```
$ echo "hola" > README.md  
$ git add README.md  
$ git commit -m "added readme"  
$ git push
```

# Gitlab CI/CD: Environment activo!

Administrator > Workshop Gitlab Backend > Pipelines > **Environments**

Available **1** Stopped **0** New environment









Environment	Deployment	Job	Commit	Updated	
production	#2 by 	test-deploy #80	 1a9af0ea  readme added	5 minutes ago	  

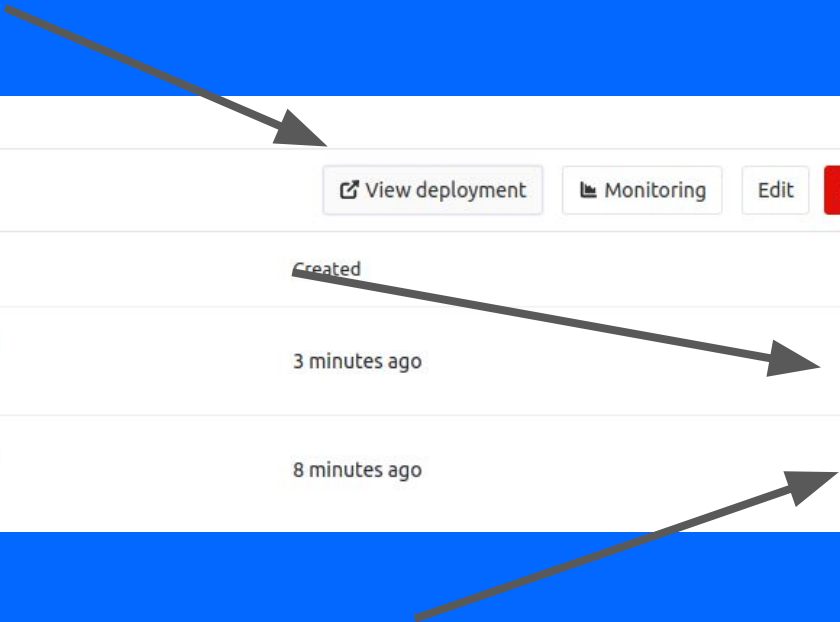
# Gitlab CI/CD: Environment activo!

Administrator > Workshop Gitlab Backend > Environments > **production**

**production**

[View deployment](#) [Monitoring](#) [Edit](#) [Stop](#)

ID	Commit	Job	Created	
#2	 master -o- 1a9af0ea  readme added	test-deploy (#80) by 	3 minutes ago	
#1	 master -o- d7cc573e  environent activate	test-deploy (#76) by 	8 minutes ago	



# *Gitlab CI/CD: Environment activo!*



# ¿CÓMO VA ESTO EN EL MUNDO REAL?

# *Gitlab CI/CD: Environments en el mundo real*

Nuestro proyecto debe tener dos ramas principales: Dev y Master

Master hace el deploy sobre el environment production

Dev hace su deploy sobre el environment staging

Necesitamos dos jobs deploy... pero que uno procese dev y otro master



# Environments + Restricciones “only”

# *Gitlab CI/CD: Environments en el mundo real*



# Gitlab CI/CD: Activando Environment

```
$ vim .gitlab-ci.yml
```

```
[...]  
test-build:  
  stage: build  
  image: docker:stable  
  before script:  
    - docker login -u gitlab-ci-token -p $CI_JOB_TOKEN gitlab.example.com:4567  
  script:  
    - docker build -t workshop-gitlab-backend .  
    - docker tag workshop-gitlab-backend gitlab.example.com:4567/root/workshop-gitlab-backend  
    - docker push gitlab.example.com:4567/root/workshop-gitlab-backend  
  only:  
    - master  
[...]
```

# Gitlab CI/CD: Activando Environment

```
[...]
build-staging:
  stage: build
  image: docker:stable
  before script:
    - docker login -u gitlab-ci-token -p $CI_JOB_TOKEN gitlab.example.com:4567
  script:
    - docker build -t backend dev .
    - docker tag backend dev gitlab.example.com:4567/root/workshop-gitlab-backend:dev
    - docker push gitlab.example.com:4567/root/workshop-gitlab-backend:dev
  only:
    - dev
[...]
```

# Gitlab CI/CD: Activando Environment

```
[...]  
deploy-production:  
  stage: deploy  
  script:  
    - echo "fin"  
  environment:  
    name: production  
    url: https://172.24.0.5  
  only:  
    - master  
[...]
```

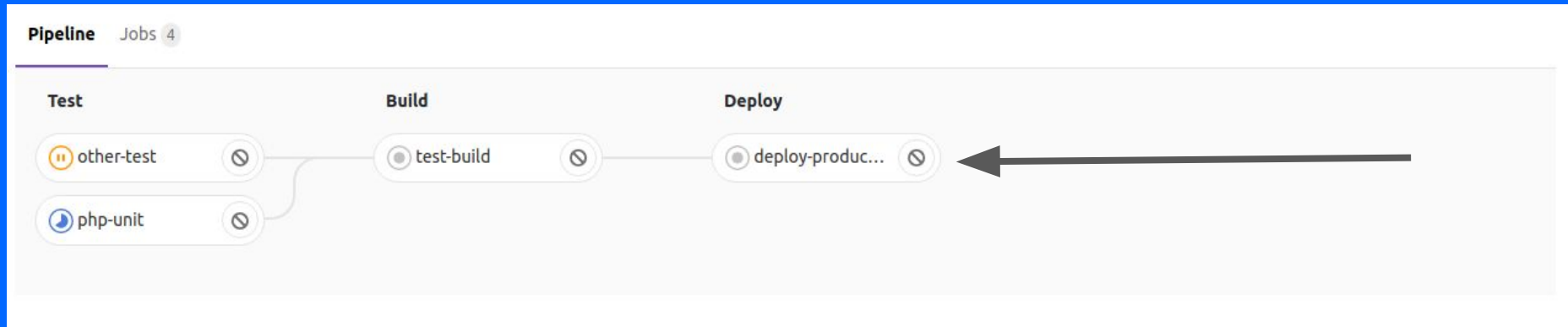
# Gitlab CI/CD: Activando Environment

```
[...]  
deploy-staging:  
  stage: deploy  
  script:  
    - echo "fin"  
  environment:  
    name: staging  
    url: https://172.24.0.4  
  only:  
    - dev  
[...]
```

# Gitlab CI/CD: Activando Environment

```
$ git add .gitlab-ci.yml  
$ git commit -m "Custom deploy by environment"  
$ git push
```

# Gitlab CI/CD: Environments en el mundo real





# Gitlab CI/CD: Activando Environment

```
$ git checkout -b dev
Cambiado a nueva rama 'dev'
$ git branch
* dev
  master
$ echo "Cambios en readme" > README.md
$ git add README.md
$ git commit -m "New branch and README changes"
$ git push -u origin dev
```

# Gitlab CI/CD: Environments en el mundo real

Pipeline Jobs 4

Test

other-test

php-unit







Build

test-build

Deploy

deploy-staging

# Gitlab CI/CD: Environments en el mundo real

<div>Available 2 Stopped 0</div> <div>New environment</div>				
Environment	Deployment	Job	Commit	Updated
production	#3 by 	deploy-production #87	 d19157ae  Custom deploy by environm...	13 minutes ago
staging	#4 by 	deploy-staging #91	 4b2623f3  New brach and README cha...	7 minutes ago

# *Gitlab CI/CD: Environments en el mundo real*



AHORA, VEAMOS TODO...  
...PARA ANGULAR

# Gitlab CI/CD: Proyecto Angular

Administrator > Test > Runners

## Runner #1

Active	<input checked="" type="checkbox"/> Paused Runners don't accept new jobs
Protected	<input type="checkbox"/> This runner will only run on pipelines triggered on protected branches
Run untagged jobs	<input checked="" type="checkbox"/> Indicates whether this runner can pick jobs without tags
Lock to current projects	<input type="checkbox"/> When a runner is locked, it cannot be assigned to other projects

Token: 915887b623a59fe7775285f55347a2

IP Address: 172.24.0.3

Description: myFirstRunner

Maximum job timeout:

This timeout will take precedence when lower than Project-defined timeout

Tags:

You can set up jobs to only use Runners with specific tags. Separate tags with commas.

[Save changes](#)

- Editamos el Runner #1 de nuestro proyecto test.
- Desmarcamos “Lock to current projects”.
- Así, podemos asignar un runner a otros proyectos.

# Gitlab CI/CD: Proyecto Angular

Blank project

Create from template

Import project

Project name

Workshop Gitlab Frontend

Project URL

http://gitlab.example.com:9090/ root

Project slug

workshop-gitlab-frontend

Want to house several dependent projects under the same namespace? [Create a group.](#)

Project description (optional)

Description format

Visibility Level

☒ Private

Project access must be granted explicitly to each user.

☐ Internal

The project can be accessed by any logged in user.

☐ Public

The project can be accessed without any authentication.

☐ Initialize repository with a README

Allows you to immediately clone this project's repository. Skip this if you plan to push up an existing repository.

Create project

Cancel

- Creamos un nuevo proyecto.
- Lo llamamos “Workshop Gitlab Frontend”.

# Gitlab CI/CD: Proyecto Angular

## Set up a specific Runner manually

1. Install GitLab Runner
2. Specify the following URL during the Runner setup:  
`http://gitlab.example.com:9090/`
3. Use the following registration token during setup:  
`s-TNimzSsyQhfsa6yhu5`
4. Start the Runner!

Reset runners registration token

GitLab Group Runners can execute code for all the projects in this group. They can be managed using the [Runners API](#).

This project does not belong to a group and can therefore not make use of group Runners.

## Available specific runners

● bfc2b21f

My Second Runner

#2

Enable for this project

● 915887b6

myFirstRunner

#1

Enable for this project



- En el nuevo proyecto vamos a “Settings => CI/CD”.
- Clicamos en “Expand” en “Runners”.
- Vemos cómo “myFirstRunner” está disponible para el proyecto.
- Lo activamos para el proyecto.



# Gitlab CI/CD: agu.gitlab-ci.xml

```
$ cd $HOME/devFestWorkshop
$ git clone https://github.com/rolando-caldas/workshop-gitlab-frontend
Clonando en 'workshop-gitlab-frontend'...
remote: Enumerating objects: 160, done.
      remote: Counting objects: 100% (160/160), done.
remote: Compressing objects: 100% (97/97), done.
remote: Total 160 (delta 51), reused 155 (delta 46), pack-reused 0
Recibiendo objetos: 100% (160/160), 138.24 KiB | 725.00 KiB/s, listo.
Resolviendo deltas: 100% (51/51), listo.
$ cd workshop-gitlab-frontend
$ rm -fR .git
```

# Gitlab CI/CD: Proyecto PHP - Symfony4

```
$ git init
  Inicializado repositorio Git vacío en
  /home/rolando/devFestWorkshop/workshop-gitlab-frontend/.git/

$ git remote add origin ssh://git@gitlab.example.com:22/root/workshop-gitlab-frontend.git
$ vim .gitlab-ci.yml
stages:
  - test
  - build
  - deploy

test:
  stage: test
  script:
    - echo "Nothing to do"
[...]
```

# Gitlab CI/CD: Proyecto PHP - Symfony4

```
[...]
build-staging:
  stage: build
  image: docker:stable
  before script:
    - docker login -u gitlab-ci-token -p $CI_JOB_TOKEN gitlab.example.com:4567
  script:
    - ls -lha
    - docker build -t frontend dev .
    - docker tag frontend dev gitlab.example.com:4567/root/workshop-gitlab-frontend:dev
    - docker push gitlab.example.com:4567/root/workshop-gitlab-frontend:dev
  only:
    - dev
[...]
```

# Gitlab CI/CD: Proyecto PHP - Symfony4

```
[...]
build-production:
  stage: build
  image: docker:stable
  before script:
    - docker login -u gitlab-ci-token -p $CI_JOB_TOKEN gitlab.example.com:4567
  script:
    - ls -lha
    - docker build -t frontend latest .
    - docker tag frontend latest gitlab.example.com:4567/root/workshop-gitlab-frontend
    - docker push gitlab.example.com:4567/root/workshop-gitlab-frontend
  only:
    - master
[...]
```

# Gitlab CI/CD: Proyecto PHP - Symfony4

```
[...]
deploy-production:
  stage: deploy
  script:
    - echo "fin"
  environment:
    name: production
    url: https://172.24.0.7
  only:
    - master

deploy-staging:
  stage: deploy
  script:
    - echo "fin"
  environment:
    name: staging
    url: https://172.24.0.6
  only:
    - dev
```

# Gitlab CI/CD: Proyecto PHP - Symfony4

```
$ vim Dockerfile
FROM rolandocaldas/angular

COPY application /application
COPY server /server

RUN cd /application/marvel-like && npm install
RUN cd /application/marvel-like && ng build --prod
RUN cd /server &&
RUN cp -r /application/marvel-like/dist/marvel-like /server/dist

WORKDIR "/server"

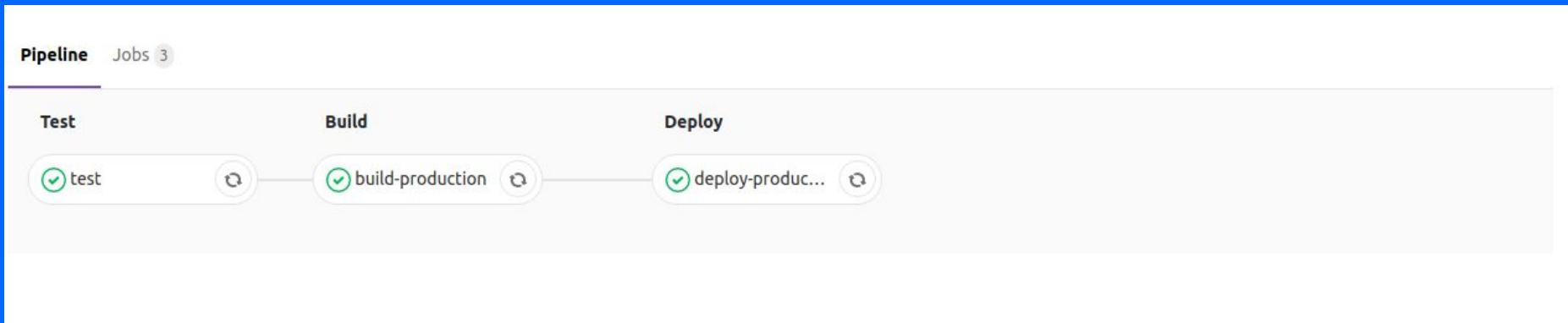
EXPOSE 80

CMD [ "node", "server.js" ]
```

# Gitlab CI/CD: Activando Environment







```
$ git add * .gitlab-ci.yml .gitignore  
$ git commit -m "Initial commit"  
$ git push --set-upstream origin master
```

# Gitlab CI/CD: Environments en el mundo real





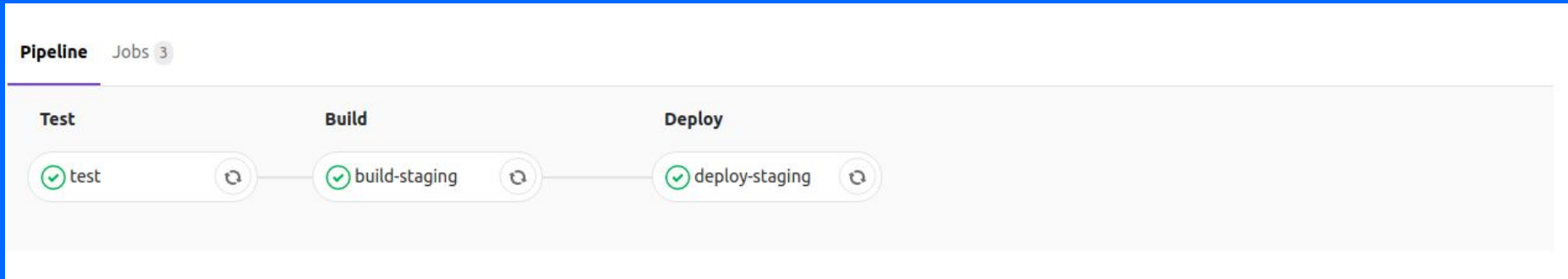
# Gitlab CI/CD: Environments en el mundo real

Available 1 Stopped 0 <span>New environment</span>				
Environment	Deployment	Job	Commit	Updated
production	#1 by 	deploy-production #100	 f0315e29  bad Dockerfil	2 minutes ago   











# Gitlab CI/CD: Activando Environment

```
$ git checkout -b dev
Cambiado a nueva rama 'dev'
$ git branch
* dev
  master
$ echo "Fichero readme" > README.md
$ git add README.md
$ git commit -m "New branch and README changes"
$ git push -u origin dev
```

# Gitlab CI/CD: Environments en el mundo real



# Gitlab CI/CD: Environments en el mundo real

Available 2 Stopped 0					New environment
Environment	Deployment	Job	Commit	Updated	
production	#1 by 	deploy-production #100	-o f0315e29  bad Dockerfil	15 minutes ago	  
staging	#2 by 	deploy-staging #103	-o df17ed4a  New branch and README ch...	10 minutes ago	  

# *Gitlab CI/CD: Environments en el mundo real*



# LAST CHALLENGE...

# LAST CHALLENGE...

# DEPLOY

# Gitlab CI/CD: Activando Environment

```
$ sudo apt-get install openssh-server -y
$ cp $HOME/.ssh/id_rsa.pub $HOME/.ssh/authorized_keys
$ vim $HOME/devFestWorkshop/workshop-gitlab/runner/config/config.toml
[...]
volumes = ["/home/rolando/.ssh:/root/.ssh", "/var/run/docker.sock:/var/run/docker.sock",
"/cache"]
[...]

$ cd $HOME/devFestWorkshop/workshop-gitlab-frontend
```



# Gitlab CI/CD: Activando Environment

```
$ vim .gitlab-ci.yml
```

```
[...]
```

```
deploy-staging:
```

```
stage: deploy
```

```
image: gotechnies/alpine-ssh
```

```
script:
```

```
- ssh -o StrictHostKeyChecking=no rolando@172.17.0.1 "docker login -u gitlab-ci-token -p
```

```
$CI_JOB_TOKEN gitlab.example.com:4567 && docker-compose -f
```

```
/home/rolando/devFestWorkshop/workshop-gitlab/prod/docker-compose.yml stop && docker-compose -f
```

```
/home/rolando/devFestWorkshop/workshop-gitlab/prod/docker-compose.yml up -d --build"
```

```
environment:
```

```
name: staging
```

```
url: https://172.24.0.6
```

```
only:
```

```
- dev
```

# *Gitlab CI/CD: Environments en el mundo real*



# *Gitlab CI/CD: Environments en el mundo real*



**KALEIDO** | COWORKING  
CENTER

**opsou**

[www.opsou.com](http://www.opsou.com)

**pedrofigueras**

[www.pedrofigueras.com](http://www.pedrofigueras.com)