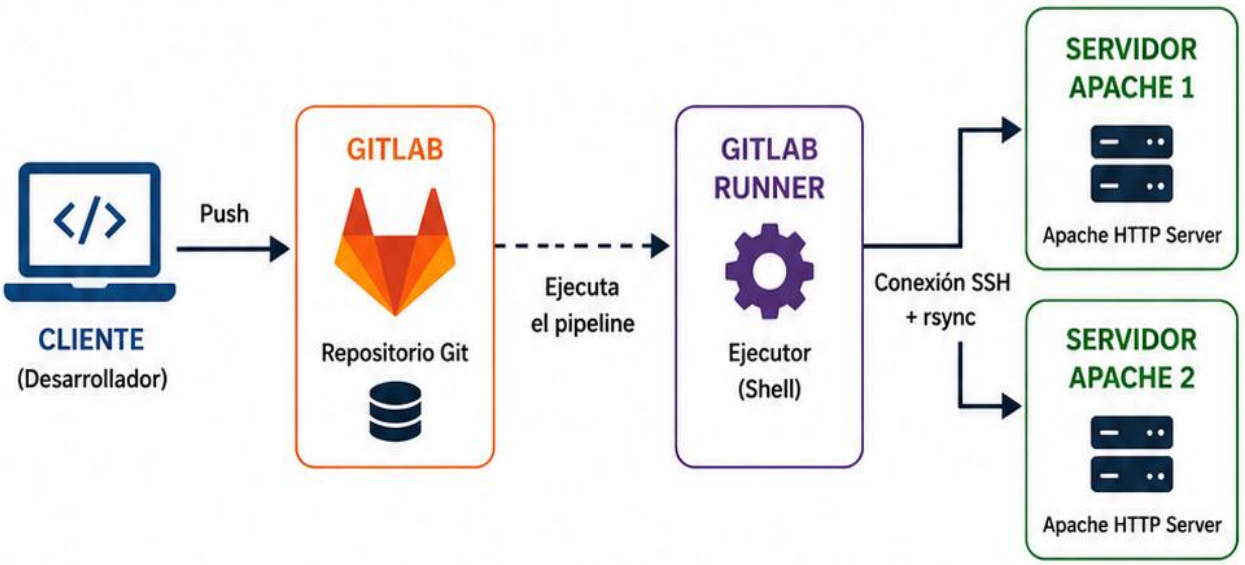
DEVOPS EN ACCIÓN:

AUTOMATIZACIÓN DE DESPLIEGUES CON GITLAB CI/CD

Proyecto Fin de Grado

Administración de Sistemas Informáticos en Red (ASIR)



Alumno:
Javier Ávila Irisarri

Ciclo Formativo:
Administración de Sistemas Informáticos en Red (ASIR)

Curso Académico:
2025 – 2026

Índice de contenido

DESARROLLO DEL PROYECTO

1. Identificación de componentes y necesidades

1.1 Introducción a él DevOps:

En la actualidad, las metodologías DevOps y los sistemas de integración y despliegue continuo (CI/CD) se han convertido en una parte fundamental del desarrollo y mantenimiento de aplicaciones y servicios web. Estas tecnologías permiten automatizar tareas repetitivas, reducir errores manuales y agilizar la publicación de cambios en entornos de producción.

El objetivo principal de este proyecto es diseñar e implementar una infraestructura capaz de automatizar el despliegue de una página web utilizando GitLab CI/CD. Para ello, se ha desarrollado un entorno virtualizado compuesto por distintos servidores encargados de gestionar el repositorio, ejecutar los pipelines y alojar la aplicación web.

La infraestructura implementada utiliza herramientas como GitLab, GitLab Runner, Docker, Apache HTTP Server y SSH, permitiendo que cualquier modificación realizada en el código fuente pueda desplegarse automáticamente en múltiples servidores web mediante un pipeline automatizado.

Con este proyecto se busca demostrar el funcionamiento de una solución CI/CD básica, funcional y fácilmente escalable, aplicando conocimientos relacionados con virtualización, administración de sistemas Linux, automatización y despliegue de servicios.

1.2 Descripción general del proyecto

El presente proyecto consiste en la implantación de un sistema de integración y despliegue continuo (CI/CD) que permita automatizar la publicación de una página web en servidores de producción.

El sistema estará basado en el uso de GitLab como plataforma de control de versiones y automatización, junto con GitLab Runner para la ejecución de tareas automatizadas.

El objetivo principal es que, tras realizar modificaciones en el código fuente desde un equipo cliente, estas se envíen al repositorio y se desplieguen automáticamente en servidores web sin intervención manual.

1.3 Tipo de empresa

Este sistema está orientado a una empresa del sector tecnológico o de desarrollo web, donde se requiere desplegar aplicaciones o páginas web de forma rápida, eficiente y sin errores manuales.

Este tipo de solución es especialmente útil en entornos donde se realizan cambios frecuentes en el código y se necesita garantizar su disponibilidad inmediata en producción.

1.4 Componentes hardware y software

Hardware

- Servidor de virtualización
- Contenedores Docker:
    - Servidores Apache (web) (SRVAPACHE-TFG01 Y SRVAPACHE-TFG02)
    - Servidor Gitlab Runner (SRVGITLABRUNNER-TFG01)
- Máquinas virtuales:
    - Servidor GitLab (SRVGITLAB-TFG01)
    - Equipo cliente (W10) (W10-TFG01)

Software

- Sistema de virtualización: Proxmox VE
- Sistema operativo: Ubuntu Server / Ubuntu Desktop
- Plataforma CI/CD: GitLab
- Ejecutor de tareas: GitLab Runner
- Servidor web: Apache HTTP Server
- Cliente de desarrollo: Cursor/VScode

Direccionamiento IP de las Maquinas y contenedores

MAQUINA	**Dirección IP**
SRVGITLAB-TFG01	**10.10.65.10/24**
SRVGITLABRUNNER-TFG01	**10.10.65.11/24**
SRVAPACHE-TFG01	**10.10.65.12/24**
W10-TFG	**10.10.65.13/24**
SRVAPACHE-TFG02	**10.10.65.14/24**

2. Sistemas operativos y herramientas

2.1 GitLab

GitLab es una plataforma de desarrollo colaborativo que permite gestionar repositorios de código, controlar versiones y automatizar procesos mediante pipelines CI/CD.

En este proyecto se ha seleccionado GitLab como plataforma principal debido a que integra en un único entorno todas las funcionalidades necesarias para la automatización del despliegue, evitando depender de herramientas externas adicionales.

Las principales funciones de GitLab dentro de la infraestructura son:

- Almacenamiento centralizado del código fuente.
- Gestión de versiones y control de cambios.
- Ejecución automática de pipelines CI/CD.
- Administración de usuarios y permisos.
- Integración con GitLab Runner para automatizar despliegues.

La elección de GitLab frente a otras alternativas, como GitHub o Jenkins, se debe principalmente a su integración nativa con sistemas CI/CD, su facilidad de configuración y la posibilidad de alojarlo en una infraestructura propia, proporcionando un mayor control sobre los datos y servicios desplegados.

2.2 GitLab Runner

GitLab Runner es el componente encargado de ejecutar las tareas definidas en los pipelines de GitLab.

En la infraestructura desarrollada, el Runner actúa como intermediario entre GitLab y los servidores Apache, ejecutando automáticamente los scripts de despliegue cuando se detectan cambios en el repositorio.

Entre sus funciones principales destacan:

- Ejecución automática de tareas CI/CD.
- Conexión con la instancia GitLab.
- Transferencia automatizada de archivos a los servidores web.
- Automatización del proceso de despliegue.

Se ha optado por utilizar el ejecutor de tipo *Shell* debido a su simplicidad y facilidad de integración con comandos SSH y herramientas del sistema Linux. Esta elección permite realizar despliegues remotos de forma sencilla sin necesidad de utilizar contenedores adicionales para cada tarea.

Además, separar el Runner en un servidor independiente mejora la organización de la infraestructura y facilita futuras ampliaciones del sistema.

2.3 Apache HTTP Server

Apache HTTP Server es el servidor web encargado de publicar y servir el contenido de la página web desplegada.

Para este proyecto se han utilizado dos servidores Apache con el objetivo de simular un entorno de producción real y demostrar la capacidad del sistema CI/CD para sincronizar automáticamente múltiples servidores.

La elección de Apache se debe a varios factores:

- Facilidad de configuración y administración.
- Amplia documentación y soporte.
- Gran estabilidad en entornos de producción.
- Compatibilidad con sistemas Linux.
- Sencillez para configurar hosts virtuales.

Aunque existen alternativas como Nginx, Apache resulta especialmente adecuado para entornos educativos y de pruebas debido a su configuración más accesible y a la gran cantidad de documentación disponible.

2.4 Proxmox VE

Proxmox VE es una plataforma de virtualización utilizada para la creación y administración de máquinas virtuales y contenedores.

En este proyecto, Proxmox constituye la base de toda la infraestructura virtual utilizada para desplegar los distintos servicios.

Se ha seleccionado esta solución debido a las siguientes ventajas:

- Administración centralizada de máquinas virtuales y contenedores.
- Facilidad para crear entornos aislados.
- Bajo consumo de recursos.

- Compatibilidad con contenedores LXC y máquinas virtuales KVM.
- Flexibilidad para ampliar la infraestructura.

Gracias a Proxmox, ha sido posible separar cada componente del proyecto en distintos servidores virtuales, mejorando la organización, el aislamiento y la seguridad del entorno.

2.5 Docker

Docker es una plataforma de contenedores que permite desplegar aplicaciones de forma aislada y portable.

En este proyecto se utiliza Docker para desplegar la instancia de GitLab dentro de un contenedor, simplificando tanto la instalación como el mantenimiento del servicio.

La utilización de Docker aporta varias ventajas:

- Despliegue rápido y reproducible.
- Aislamiento de servicios.
- Facilidad para realizar actualizaciones.
- Portabilidad entre sistemas.
- Simplificación de dependencias.

Además, se ha utilizado docker compose para definir la configuración del servicio mediante un archivo YAML, permitiendo gestionar la infraestructura de forma más organizada y automatizada.

2.6 SSH

OpenSSH es el protocolo utilizado para realizar conexiones remotas seguras entre servidores.

En el proyecto se emplea SSH para permitir que el GitLab Runner pueda conectarse automáticamente a los servidores Apache y desplegar el contenido web.

Se ha optado por utilizar autenticación mediante claves pública/privada en lugar de contraseñas debido a las siguientes ventajas:

- Mayor seguridad.
- Automatización de conexiones.

- Reducción de intervención manual.
- Protección frente a ataques de fuerza bruta.

Esta configuración permite que los despliegues se realicen de manera automática y segura desde el Runner hacia los servidores web.

2.7 Visual Studio Code / Cursor

Visual Studio Code y Cursor han sido utilizados como entornos de desarrollo para la edición y gestión del código fuente.

Estas herramientas permiten trabajar de forma eficiente gracias a funcionalidades como:

- Integración con Git.
- Resaltado de sintaxis.
- Terminal integrada.
- Gestión de extensiones.
- Edición rápida de archivos de configuración.

La utilización de estos editores facilita el desarrollo y mantenimiento del proyecto, especialmente durante la creación de archivos YAML, scripts y configuraciones relacionadas con el pipeline CI/CD.

3. Fases del proyecto y planificación

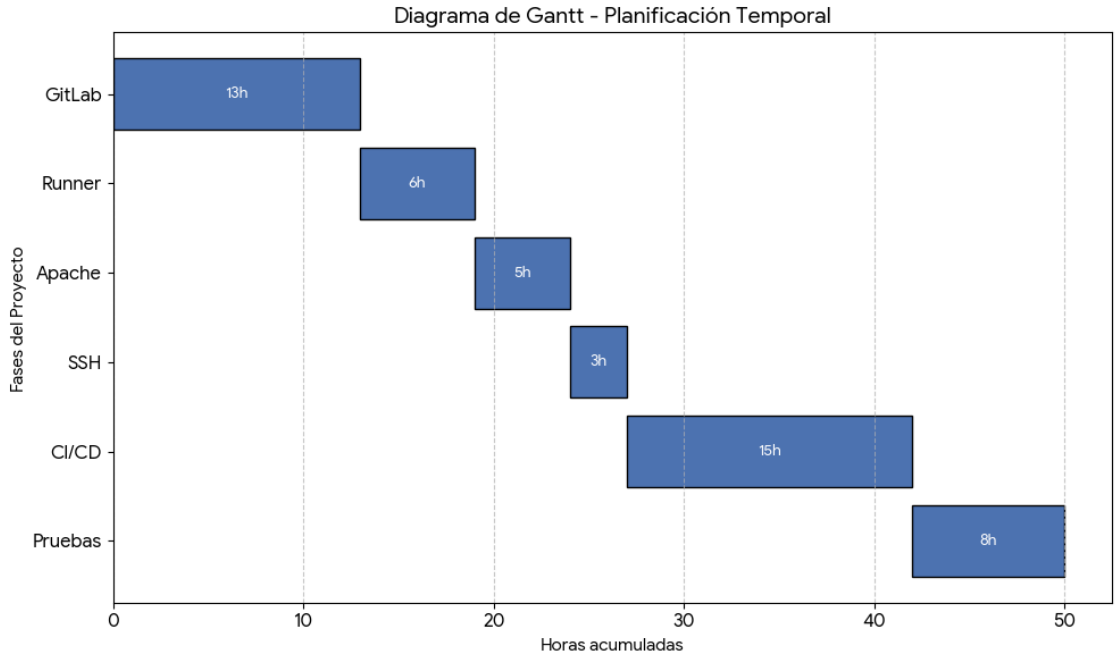
3.1 Fases del proyecto

1. Configuración del servidor GitLab
2. Configuración del GitLab Runner
3. Configuración de servidores Apache
4. Configuración de acceso SSH
5. Desarrollo del pipeline CI/CD
6. Pruebas de funcionamiento

3.2 Planificación temporal y diagrama Gantt

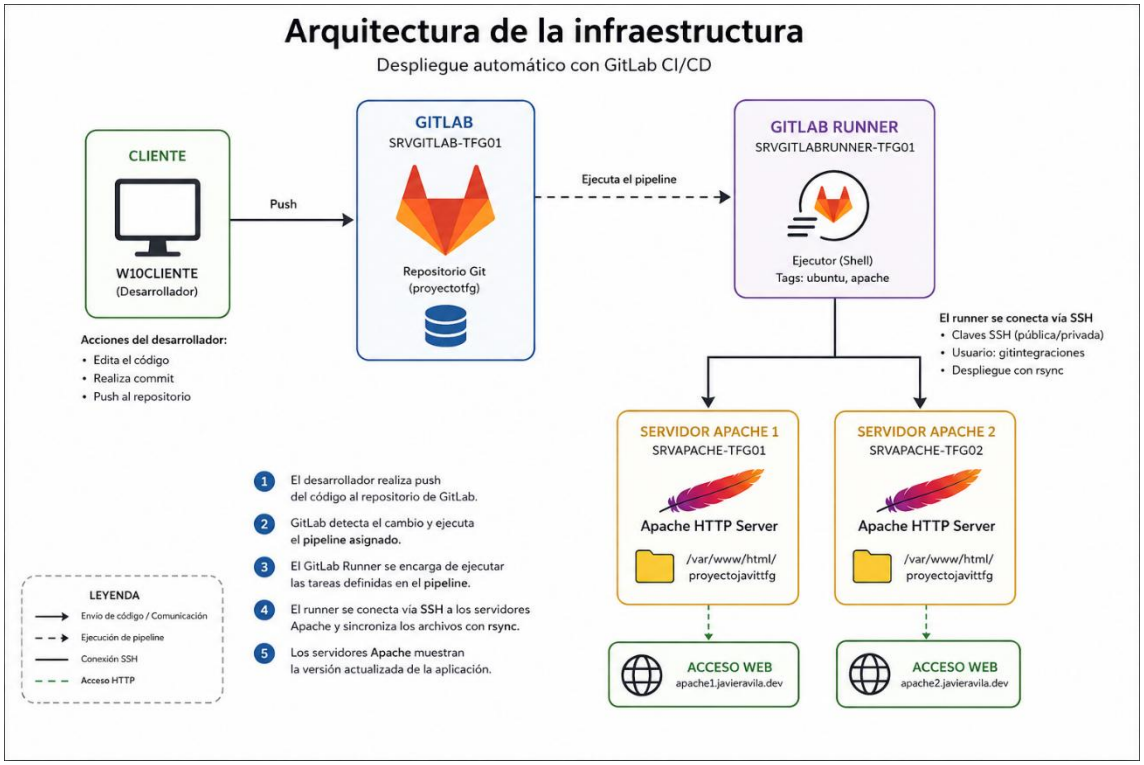
Fase	Duración
GitLab	13 horas
Runner	6 horas
Apache	5 horas
SSH	3 horas
CI/CD	15 horas
Pruebas	8 horas

Con esta planificación realizaremos el diagrama de Gantt que quedaría así:



3.2 Esquema del proyecto

Este sería el esquema final del proyecto y que haría:



4. Implantación del sistema

En esta fase se realiza la implementación de los distintos componentes necesarios para el funcionamiento del sistema CI/CD.

4.1 Configuración del servidor GitLab

4.1.1 Instalación de dependencias (Docker) **en SRVGITLAB01-TFG01**

Para el despliegue de GitLab es necesaria la instalación previa de Docker y Docker Compose, ya que la plataforma se ejecutará mediante contenedores.

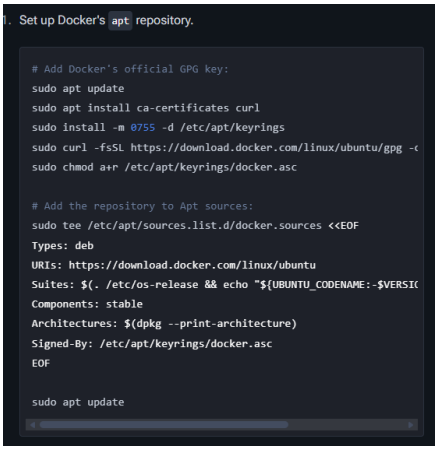
Subtareas:

- Actualización del sistema, en primer lugar, se accede al servidor mediante SSH utilizando las credenciales previamente configuradas y ejecutamos sudo apt update && sudo apt upgrade -y:

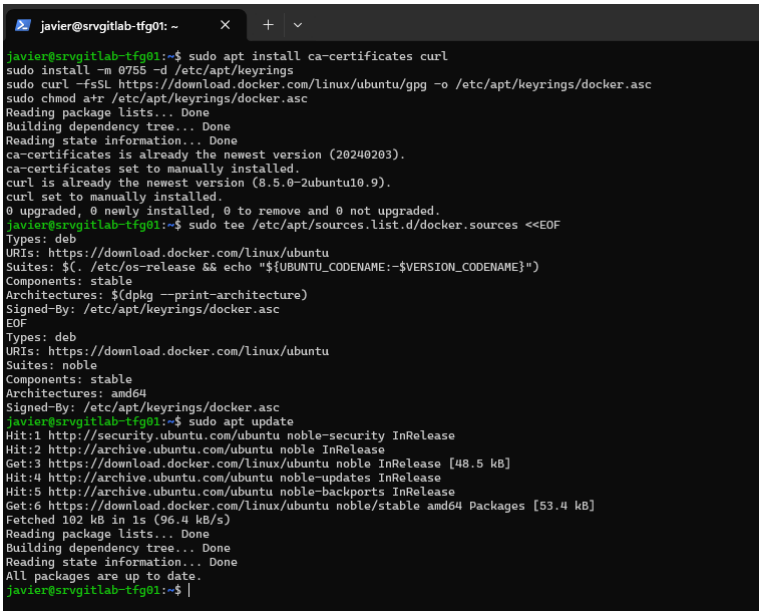
```
javier@srvgitlab-tfg01:~$ sudo apt update && sudo apt upgrade -y
[sudo] password for javier:
Sorry, try again.
[sudo] password for javier:
Hit:1 http://security.ubuntu.com/ubuntu noble-security InRelease
Hit:2 http://archive.ubuntu.com/ubuntu noble InRelease
Hit:3 http://archive.ubuntu.com/ubuntu noble-updates InRelease
Hit:4 http://archive.ubuntu.com/ubuntu noble-backports InRelease
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
37 packages can be upgraded. Run 'apt list --upgradable' to see them.
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Calculating upgrade... Done
The following packages will be upgraded:
  apparmor cloud-init coreutils distro-info-data fwupd iproute2 libapparmor1 libfwupd2 libnetplan1 libnf
```

- Instalación de Docker:

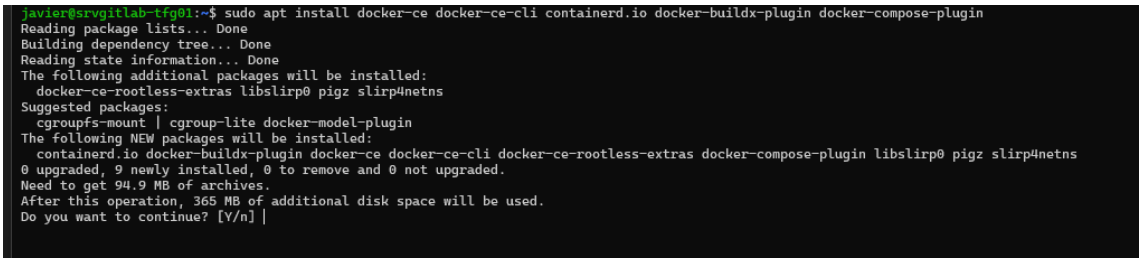
  Para ello tenemos que primero añadir el repositorio oficial de Docker, vamos a la página* (dicha en el anexo) y se siguen las instrucciones proporcionadas en la documentación oficial.:



  Tras la ejecución de los comandos, se verifica que el repositorio ha sido añadido correctamente al sistema:



  Ahora instalaremos Docker Engine, sus dependencias y Docker Compose:



- Verificación del servicio:

  Finalmente, se verifica la correcta instalación mediante el comando docker --version:

4.1.2 Despliegue de GitLab en SRVGITLAB01-TFG01

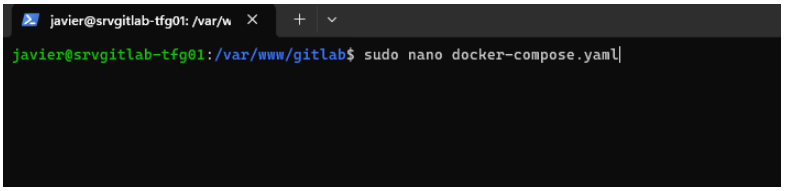
Una vez instalado Docker, se procede al despliegue de GitLab mediante un archivo docker-compose.yml. El archivo YAML permite definir la imagen del contenedor y los parámetros necesarios para su configuración

Subtareas:

- Creación del fichero docker-compose y configuración de la vm:

  En primer lugar, se define la ubicación donde se almacenará la configuración del contenedor y los datos persistentes de GitLab, se ha decidido almacenar la configuración y los datos persistentes en la ruta /var/www/gitlab

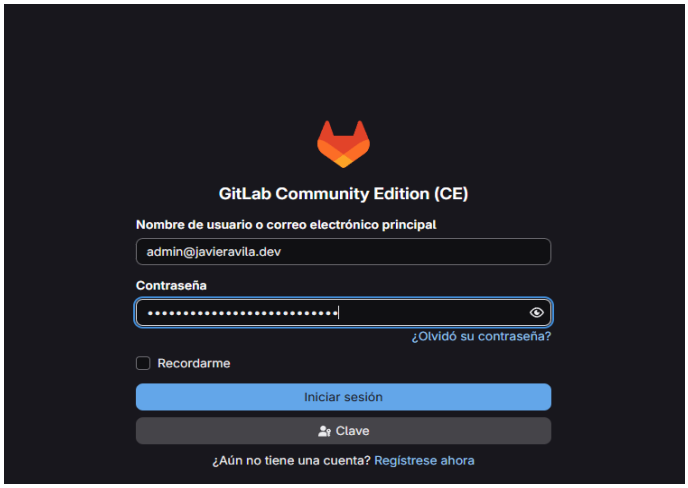
  A continuación, se crea el archivo docker-compose.yml, encargado de definir la configuración del contenedor:



  En este archivo se especifica el uso de la imagen oficial gitlab-ce (community edition), así como los parámetros necesarios para su funcionamiento:

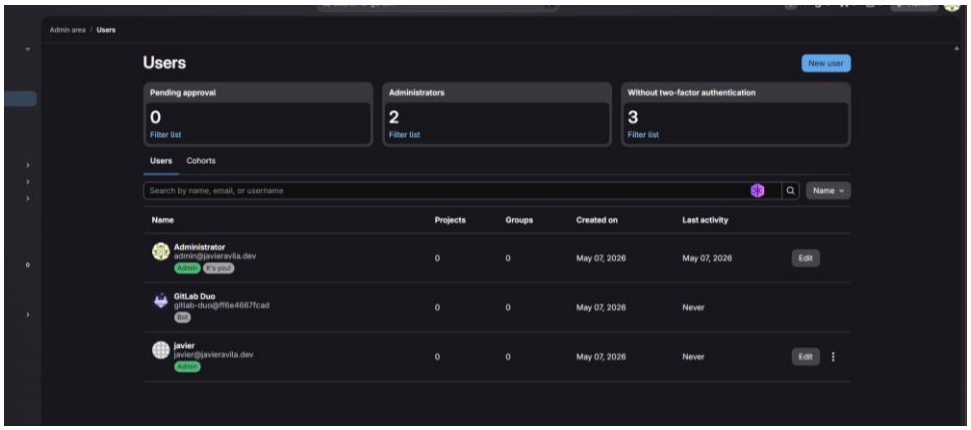
- Levantamiento del contenedor
    - A continuación, se ejecuta el comando docker compose up -d con el objetivo de desplegar el contenedor de GitLab en segundo plano:

- Acceso vía web
    - Desde el navegador del equipo cliente se accede a la dirección IP del servidor GitLab.:

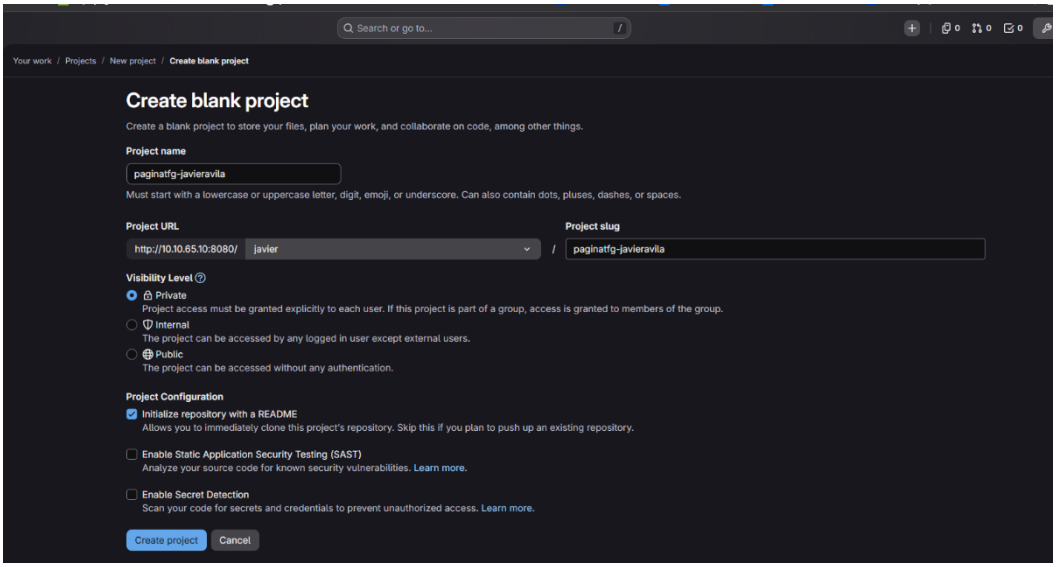


4.1.3 Configuración inicial

- Creación de Usuario
    - A continuación, se crea un usuario dentro de la plataforma GitLab y entrar con él, en este caso, el usuario será javier:



- Creación de Repositorio:
    - Ahora crearemos el repositorio de código, donde estará almacenado el código de la página web del proyecto:



4.2 Configuración del GitLab Runner

4.2.1 Instalación del Runner

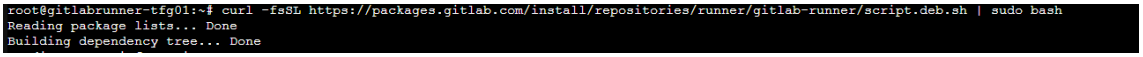
Instalación de GitLab Runner en la maquina SRVGITLABRUNNER-TFG01 esta máquina servirá de ejecutor. Cuando subamos el código a Gitlab se encargará de sincronizar automáticamente el contenido del repositorio en nuestros servidores apaches garantizando que ambos servidores mantengan la misma versión de la web

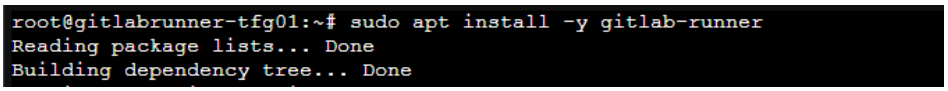
---

4.2.2 Registro del Runner en GitLab

- Obtención del token
    - Para poder vincular nuestro runner con Gitlab, lo primero que tenemos que hacer es obtener un token, ese token nos servirá para asociar el Runner con la instancia GitLab correspondiente, para ello obtendremos el token desde la instancia de GitLab (estará en la parte CI/CD de nuestro apartado admin)

- Registro del runner:
    - Para el registro del runner usaremos un script (adjunto en el anexo y sacado desde la propia documentación de Gitlab) en el servidor SRVGITLABRUNNER-TFG01Este proceso permite instalar el binario gitlab-runner en el sistema, aunque sin registrar todavía la instancia:

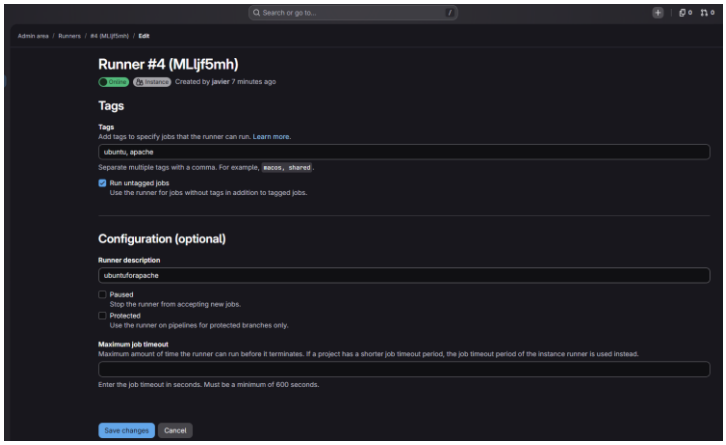




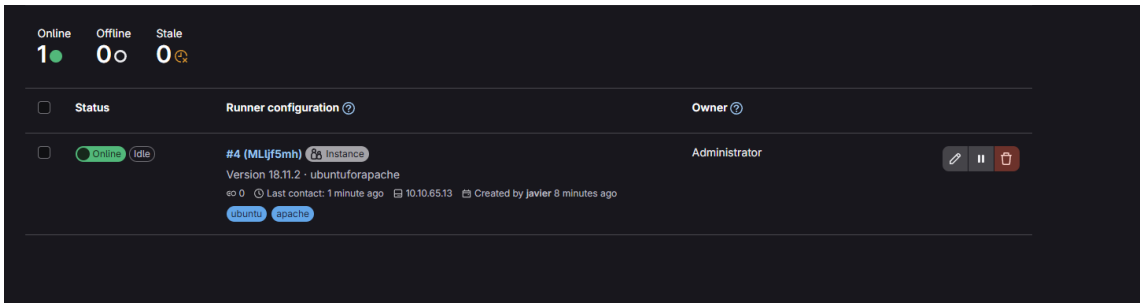
    - Una vez instalado el servicio, se procede al registro del Runner en la instancia de GitLab para permitir la ejecución de los pipelines definidos, para ello ejecutaremos la sentencia que hemos visto en la parte admin, como ejecutor (que es el método que va a utilizar) pondremos Shell ya que, aunque nos vayamos a conectar remotamente a otra maquina por vía ssh la conexión la realizaremos desde la propia maquina:

- Asignación de etiquetas:
    - Las etiquetas en los runners sirven para identificar que va a realizar y para qué sirve, por ejemplo, puedes tener varios runners y que cada uno realice una acción (Docker, apache, NPM…) en este caso, le

pondremos el tag “Ubuntu” y “apache”, en nuestro caso no sería necesario poner tags, pero es una buena práctica:



- Como vemos nuestro runner aparece como up así que ya lo habremos configurado correctamente:



4.3 Configuración de los servidores Apache

4.3.1 Instalación de Apache

Instalaremos el servidor web apache en nuestros dos servidores (SRVAPACHE-TFG01 **y** SRVAPACHE-TFG02) para ello se accede a ambos servidores, los actualizaremos y se instala el paquete apache2y lo verificaremos

```
root@srvapache-tfg01:~# apache2 -v
Server version: Apache/2.4.58 (Ubuntu)
Server built:   2026-05-05T13:22:45
root@srvapache-tfg01:~#
```

Se ha seleccionado Apache HTTP Server debido a su facilidad de configuración, estabilidad y amplia documentación disponible

---

4.3.2 Configuración del entorno web y creación del sitio

- Creación del directorio /var/www/HTML

- Esta es una acción que haremos en los dos servidores y será acceder a él servidor y crear la ruta donde ira nuestro sitio web:

```
root@srvapache-tfg01:~# mkdir -p /var/www/html/
```

Dentro de esa carpeta crearemos proyectojavitfg ya que hay irán nuestros archivos de código (página web):

```
root@srvapache-tfg01:~# cd /var/www/html/
root@srvapache-tfg01:/var/www/html# mkdir proyectojavitfg
root@srvapache-tfg01:/var/www/html# ls
index.html  proyectojavitfg
root@srvapache-tfg01:/var/www/html#
```

- Creación del sitio virtual:
  - Ahora crearemos el sitio virtual de apache*(anexo), se despliega un archivo HTML de prueba con el objetivo de verificar el correcto funcionamiento del servidor web y lo habilitamos:

```
root@srvapache-tfg01:/var/www/html/proyectojavitfg# ls
index.html
root@srvapache-tfg01:/var/www/html/proyectojavitfg# a2ensite proyectojavitfg.conf
Enabling site proyectojavitfg.
To activate the new configuration, you need to run:
  systemctl reload apache2
root@srvapache-tfg01:/var/www/html/proyectojavitfg# systemctl reload apache2
```

- Prueba de acceso vía navegador

  La prueba confirma el correcto funcionamiento del servidor web.

4.3.3 Creación de usuario de despliegue

Para que el runner se conecte a los servidores de apache necesita un usuario dedicado exclusivamente a las tareas de despliegue, este usuario es el que va a estar en las variables de gitlab y utilizado por el Runner para conectarse a las maquinas, por tanto, hay que crearlo manualmente en las maquinas (srvapache-tfg01y srvapache-tfg02)

- Creación de usuario gitintegraciones en las maquinas:
  - Para ello nos conectamos a las máquinas y creamos el usuario:

```
root@srvapache-tfg01:/var/www/html/proyectojavitfg# adduser gitintegraciones
```

- Asignación de permisos, en este caso, le asignaremos permisos sobre la carpeta de proyectojavitfg para que el usuario (a través del pipeline) pueda actuar en esa carpeta en concreto:

```
root@srvapache-tfg01:/var/www/html/proyectojavitfg# chown -R gitintegraciones:www-data /var/www/html/proyectojavitfg/
root@srvapache-tfg01:/var/www/html/proyectojavitfg# chmod -R 775 /var/www/html/proyectojavitfg/
root@srvapache-tfg01:/var/www/html/proyectojavitfg#
```

4.4 Configuración de acceso SSH

Con el objetivo de automatizar las conexiones entre el Runner y los servidores Apache, se implementa autenticación mediante claves SSH pública/privada. Esta solución evita el uso de contraseñas en texto plano y mejora la seguridad del sistema

4.4.1 Generación de claves SSH

En el servidor Runner:

- Generamos la clave publico privada, y la guardamos en el servidor del runner:

```
root@gitlabrunner-tfg01:~# ssh-keygen -t ed25519 -C "gitlab-cicd"
Generating public/private ed25519 key pair.
Enter file in which to save the key (/root/.ssh/id_ed25519):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_ed25519
Your public key has been saved in /root/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:ZJGAVISgp+c/JCDqpFTR5TTfhET5YbuvetrClQ4ROUg gitlab-cicd
The key's randomart image is:
+--[ED25519 256]--+
|  .oo=*E=++.     |
| . .o.o.+Boo     |
|. . .  .o.=.o    |
|oo .   o . o     |
|+.o     S . o    |
|.=. .     . +    |
|= .o    . + .    |
|.. ..    o.o .   |
|    ..   o=o.    |
+----[SHA256]-----+
root@gitlabrunner-tfg01:~#
```

```
root@gitlabrunner-tfg01:~# ls .ssh/
id_ed25519  id_ed25519.pub
root@gitlabrunner-tfg01:~#
```

4.4.2 Distribución de claves

- Copia de la clave pública a los servidores Apache

- Ahora lo que haremos será copiar la clave publica y añadir la clave pública al archivo authorized_keys del usuario de despliegue (gitlabintegraciones) en los servers apache, permitiendo la autenticación automática desde el Runner.:

```
gitintegraciones@srvapache-tfg01:~/.ssh$ cat authorized_keys
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIDTARUDsF+BhSGBgQKbnRCnSUeMCAz+XErgGPuFbslgD gitlab-cicd
gitintegraciones@srvapache-tfg01:~/.ssh$
```

- Verificar la conexión desde el runner a los servidores apache

Como vemos, conseguimos llegar a él servidor apache:

```
root@gitlabrunner-tfg01:~#
root@gitlabrunner-tfg01:~# ssh gitintegraciones@10.10.65.12
The authenticity of host '10.10.65.12 (10.10.65.12)' can't be established.
ED25519 key fingerprint is SHA256:fxa38SKss8WeyH3DGmMUUoDLckpKGVq3qemQo+N+yjs.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.10.65.12' (ED25519) to the list of known hosts.
Welcome to Ubuntu 24.04.4 LTS (GNU/Linux 6.17.13-2-pve x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

gitintegraciones@srvapache-tfg01:~$
```

4.5 Desarrollo del pipeline CI/CD

4.5.1 Subida de código al repositorio

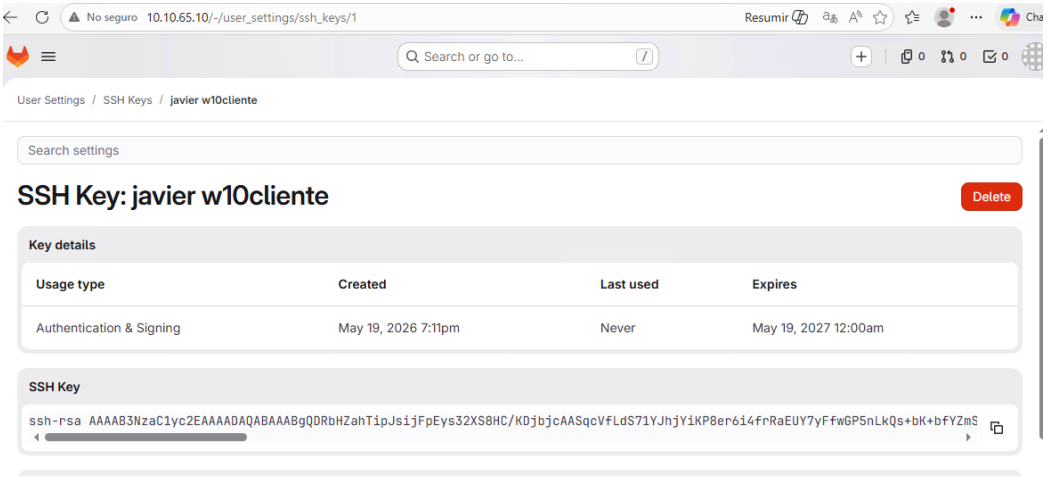
- Subida del código inicial desde el cliente para ello tenemos que preparar el cliente para que pueda clonar el proyecto (hacer pull, traérselo a local) y enviarlo a gitlab (hacer push)
    - instalación de git: para hacer lo dicho anteriormente necesitamos disponer de git, para ello nos lo descargaremos de la página web y lo instalaremos en el equipo w10cliente
    - Configurar git-global username y git-global email, estos valores se tienen que corresponder con los que creamos en el gitlab en este caso javier como username y email javier@javieravila.dev

```
C:\Users\javier>git config --global user.mail "javier@javieravila.dev"

C:\Users\javier>git config --global user.name "javier"

C:\Users\javier>
```

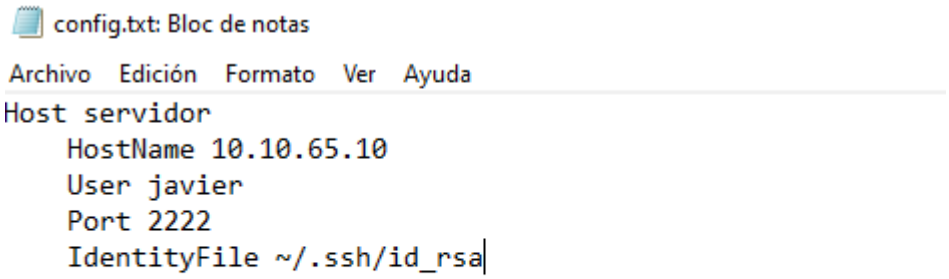
- Instalar VSCode (editor con el que editaremos el código) y clonar el repo:
  - Antes haremos la configuración del archivo config en el fichero de ssh del usuario, en este fichero definimos un hostname en este caso la ip y con que usuario y que clave ssh va a usar para conectarse a git y clonar y subir para ello también crearemos la clave publico/privada en el servidor:

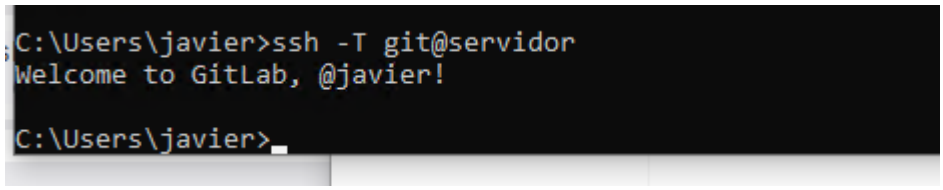
```
C:\Users\javier>ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (C:\Users\javier/.ssh/id_rsa):
Created directory 'C:\Users\javier/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in C:\Users\javier/.ssh/id_rsa.
Your public key has been saved in C:\Users\javier/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:8Zq3Ho1wgdE8eQcY8syGW7DJtGtDDjDV/4cjdRlCEXk javier@DESKTOP-FEJTO5V
The key's randomart image is:
+---[RSA 3072]----+
|    o...=o.===   |
|     o o.&* + E  |
|      . X.Bo + o |
|       + B... o  |
|        S oo o   |
|       . *.o+ .  |
|        o +..o   |
|         . o     |
|          .o     |
+----[SHA256]-----+

C:\Users\javier>
```

  - Ahora con la clave creada la subimos a gitlab en nuestro perfil:

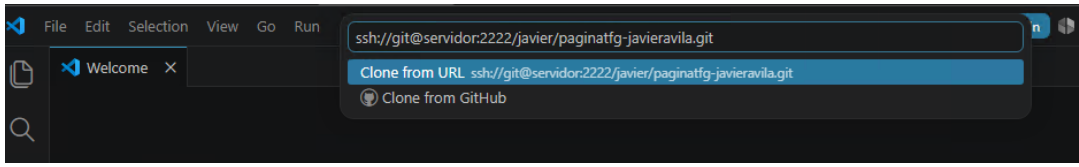


  - Ahora lo que haremos será crear el archivo de configuración y lo guardaremos en el directorio de .ssh del usuario

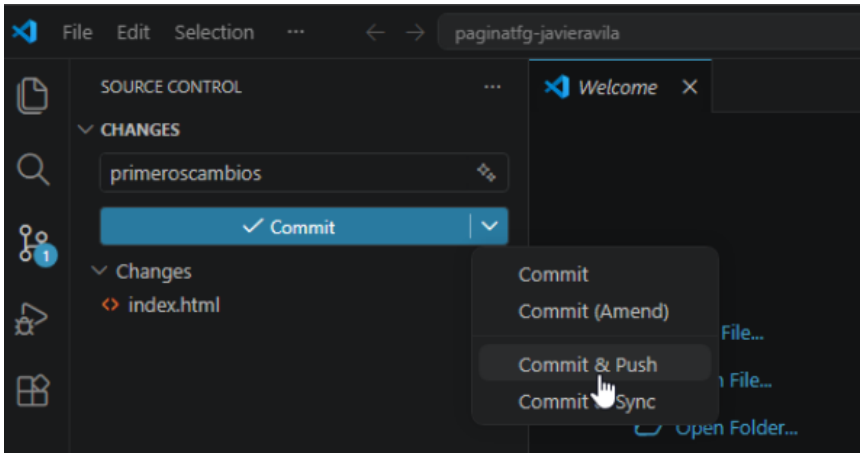




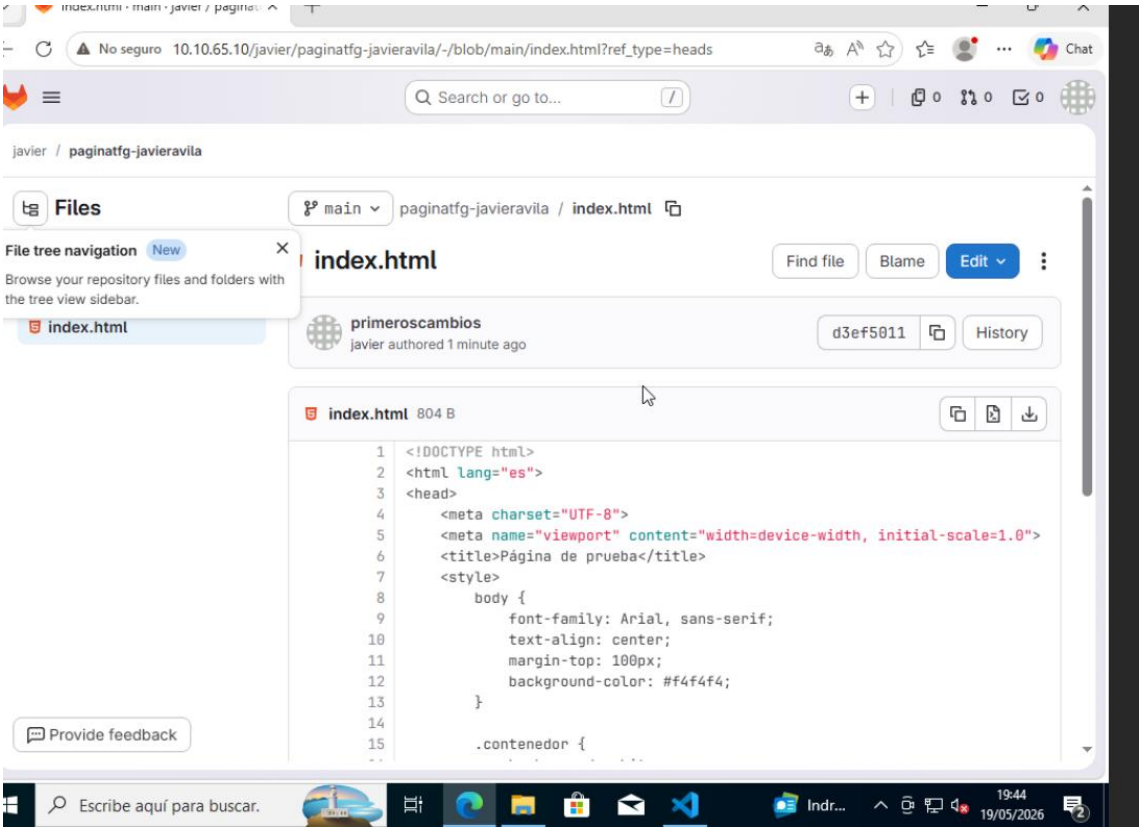
- Clonaremos con un git clone el repositorio en este caso meteremos “servidor” ya que lo hemos definido como un alias antes en el archivo de configuración:



- Añadir una página y hacer push:



- Ver el código que hemos creado en gitlab:



4.5.2 Creación del archivo .gitlab-ci.yml y configuración del repositorio para despliegue

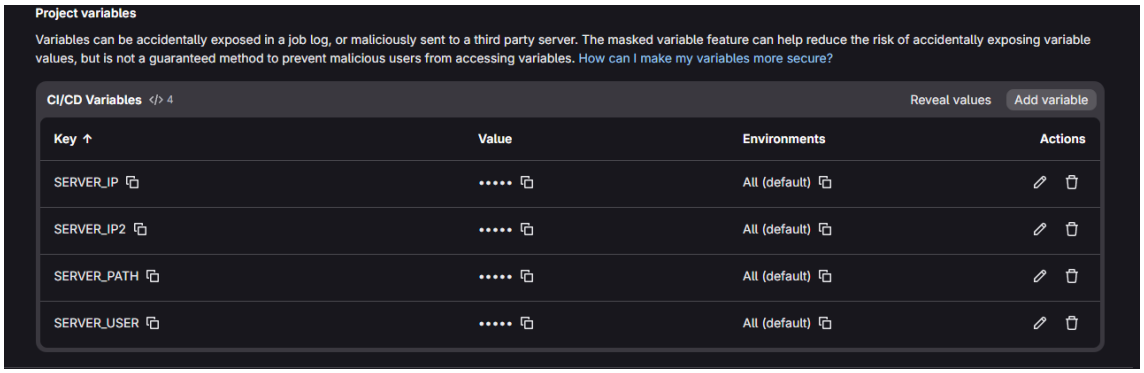
Una vez preparada la infraestructura, se procede a la creación del pipeline de despliegue de forma automática para cada vez que hacemos un push. Para ello lo que haremos será crear el archivo .gitlab-ci.yml que será el que tenga las tareas del pipeline

Definición de las fases del pipeline:

- En este caso, nuestro pipeline llevara solo una fase que será la fase de deploy, tenemos que pensar que esto se ejecutara en el runner entonces lo que vamos a hacer va a ser que copie nuestro repo en la ubicación, como lo haremos a través de rsync (la cual es una herramienta de Linux que permite sincronizar con muchas opciones) Se utilizan los parámetros -avz de rsync para optimizar la transferencia de archivos y para que transfiera únicamente los archivos modificados, optimizando el proceso de despliegue (el CI/CD va anexado)

Una vez tenemos definido el pipeline antes de subirlo, asignamos las variables, en nuestro caso:

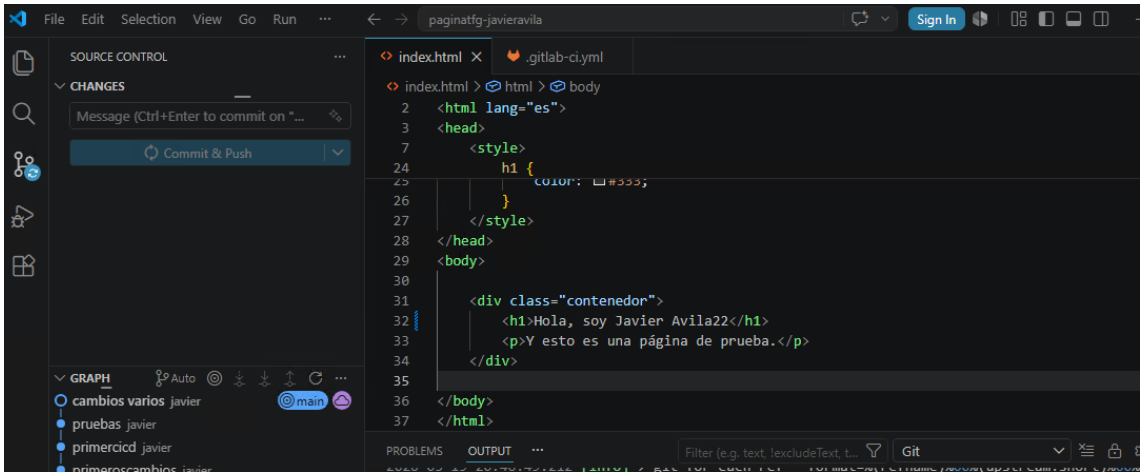
- SERVER_IP = ip del servidor apache 1
- SERVER_IP2 = ip del servidor apache2
- SERVER_PATH = Path donde está en los 2 apaches la página web
- SERVER_USER = el usuario con el que se conecta desde el runner vía ssh

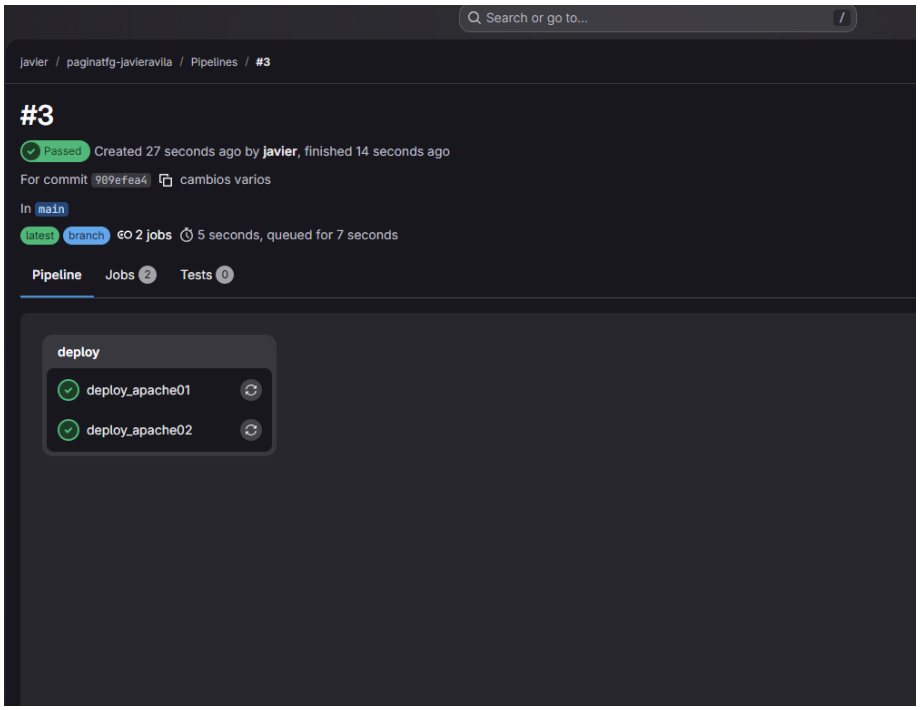


Ahora subimos el pipeline a git y la infraestructura y el entorno quedan preparado para realizar despliegues automáticos y para nuestro primer push (en CI/CD)

4.6 Pruebas de CI/CD

Haremos un push desde el equipo cliente w10 para verificar el correcto funcionamiento del pipeline





Como vemos ahora en las dos webs se han actualizado, llegando a el objetivo que queríamos que era que todo el código estuviera centralizado en git y da igual si tenemos 1 o infinitos servidores, garantizando la sincronización del contenido entre todos los servidores.

4.7 POC

A nivel de prueba de concepto he creado para poder modificar lo mismo a nivel web (apache1.javieravila.dev, apache2.javieravila.dev, cliente.javieravila.dev)

Para demostrar el funcionamiento del sistema, únicamente es necesario acceder al entorno cliente, realizar modificaciones en el código y ejecutar un push al repositorio

5.Conclusiones

Tras la realización de este proyecto, se ha conseguido implementar correctamente una infraestructura CI/CD funcional capaz de automatizar el despliegue de una aplicación web en varios servidores Apache de forma simultánea.

La utilización de GitLab y GitLab Runner ha permitido automatizar tareas que normalmente se realizarían manualmente, reduciendo tiempos de despliegue y minimizando posibles errores humanos. Además, el uso de claves SSH y

herramientas como rsync ha permitido realizar transferencias seguras y eficientes entre servidores.

Durante el desarrollo del proyecto también se han aplicado conocimientos relacionados con virtualización, redes, administración de sistemas Linux, contenedores Docker y automatización de procesos, integrando distintas tecnologías dentro de una misma infraestructura.

Como posibles mejoras futuras, se podrían implementar nuevas funcionalidades como:

- Despliegues automáticos en entornos de pruebas y producción.
- Integración de validaciones automáticas antes del despliegue.
- Balanceadores de carga entre servidores Apache.
- Monitorización de servicios y pipelines.
- Uso de HTTPS mediante certificados SSL.

En conclusión, el proyecto ha permitido desarrollar una solución funcional de integración y despliegue continuo, demostrando las ventajas que aporta la automatización dentro de entornos de administración de sistemas y desarrollo web.

6.Anexos

En este apartado se incluyen los archivos de configuración, scripts, comandos y recursos utilizados durante el desarrollo del proyecto.

ANEXO I – Archivo docker-compose.yml

El siguiente archivo docker-compose.yml ha sido utilizado para desplegar la instancia de GitLab dentro de un contenedor Docker.

```
version: '3.8'

services:
 gitlab:
  image: gitlab/gitlab-ce: latest
  container_name: gitlab
  restart: always
  hostname: 'gitlab.javieravila.dev'

  ports:
   - '80:80'
   - '443:443'
   - '22:22'

  volumes:
   - '/var/www/gitlab/config:/etc/gitlab'
   - '/var/www/gitlab/logs:/var/log/gitlab'
   - '/var/www/gitlab/data:/var/opt/gitlab'

  shm_size: '256m'
```

ANEXO II – Archivo.gitlab-ci.yml

Archivo encargado de automatizar el despliegue de la página web en los servidores Apache.

```
deploy_apache01:
 stage: deploy
 script:
  - ssh $SERVER_USER@$SERVER_IP "mkdir -p $SERVER_PATH"
```

```
  - rsync -avz --delete --exclude='. env'./
$SERVER_USER@$SERVER_IP:$SERVER_PATH
 only:
  - main
deploy_apache02:
 stage: deploy
 script:
  - ssh $SERVER_USER@$SERVER_IP2 "mkdir -p $SERVER_PATH"
  - rsync -avz --delete --exclude='.env' ./
$SERVER_USER@$SERVER_IP2:$SERVER_PATH
 only:
  - main
```

ANEXO III – Configuración VirtualHost Apache

Archivo de configuración utilizado para publicar la página web en Apache.

```
<VirtualHost *:80>

  DocumentRoot /var/www/html/proyectojavitfg

  <Directory /var/www/html/proyectojavitfg>
    Options Indexes FollowSymLinks
    AllowOverride All
    Require all granted
  </Directory>

  ErrorLog ${APACHE_LOG_DIR}/error.log
  CustomLog ${APACHE_LOG_DIR}/access.log combined

</VirtualHost>
```

ANEXO IV – Comandos utilizados

Instalación Docker

sudo apt update && sudo apt upgrade -y
sudo apt install docker.io docker-compose-plugin -y

Levantamiento de GitLab

sudo docker compose up -d

Instalación Apache

sudo apt install apache2 -y

Instalación GitLab Runner

curl -L https://packages.gitlab.com/install/repositories/runner/gitlab-runner/script.deb.sh | sudo bash

sudo apt install gitlab-runner -y

Registro del Runner

sudo gitlab-runner register

Generación de claves SSH

ssh-keygen -t rsa -b 4096

Copia de clave pública

ssh-copy-id gitintegraciones@10.10.65.12

Clonado del repositorio

git clone git@gitlab:javier/proyectotfg.git

ANEXO V – URLs y documentación utilizada

GitLab

GitLab Documentation

GitLab Runner

GitLab Runner Documentation

Docker

Docker Documentation

Apache HTTP Server

Apache Documentation

Proxmox VE

Proxmox Documentation

OpenSSH

OpenSSH Documentation

Rsync

Rsync Documentation