



Trabajo Trevenque

12/05/2025

Marcos Bolívar Muñoz

Grupo Trevenque SLU

IES Zaidín Vergeles

Granada

Diseño del sistema por parte del equipo de administración de sistemas.....	3
Parte 1: Preparación de los servidores necesarios para soportar MariaDB y Laravel. El sistema deberá estar diseñado de manera independiente, es decir en un principio debe de haber un servidor independiente para cada uno de ellos.....	3
Paso 1: Preparar servidor Ubuntu para Laravel.....	3
Paso 2: Preparar servidor Ubuntu para MariaDB.....	5
Paso 3: Conectar Laravel a MariaDB.....	6
Comprobación del servicio.....	6
Parte 2: Añadir un sistema de balanceo de carga al servidor MariaDB usando Galera o MaxScale, por supuesto hacerlo de manera totalmente independiente de Laravel.....	7
Paso a Paso (hacer en cada nodo).....	7
Inicializar el clúster (sólo en el primer nodo).....	8
Conectar los otros nodos.....	8
Prueba rápida de replicación.....	8
Conexión desde Laravel.....	9
Caída de servicio.....	9
Prueba de generación cluster galera.....	9
Parte 3: Preparar un sistema de balanceo de carga usando NGINX para los servidores Laravel.....	10
Objetivo:.....	10
Estructura:.....	10
Paso a Paso.....	10
Tener dos (o más) servidores Laravel funcionando.....	10
Configurar el servidor NGINX (balanceador).....	10
Prueba.....	11
Opcional: Balanceo por peso, salud, etc.....	12
Ejemplo completo con balanceo por peso y estado de salud (simple):.....	12
Laravel listo con balanceo.....	13
Prueba funcionamiento NGINX.....	13
Parte 4: Dockerizar toda la configuración una vez que los sistemas estén funcionando.....	14
Estructura propuesta del proyecto Docker.....	14
Vamos a crear:.....	14
Instrucciones para ejecutarlo.....	15
Parte 5: Instalar un configurar un servidor GIT para el control del proyecto, preparar un día para informar a los desarrolladores sobre el uso de esta herramienta.....	16
Paso 1: Modificar .yaml.....	16
Paso 2: Crear directorios.....	17
Paso 3: Iniciar Gitea.....	17
Parte 4: Accede al servidor Gitea.....	17
Parte 6: Preparar todos los scripts necesarios (python, bash, node) para el control de las configuraciones.....	18
Máquinas virtuales.....	18
Proyecto Dockerizado.....	19
Conclusión.....	19
Conclusión final del proyecto.....	20
Bibliografía del Proyecto.....	20

Diseño del sistema por parte del equipo de administración de sistemas.

Parte 1: Preparación de los servidores necesarios para soportar MariaDB y Laravel. El sistema deberá estar diseñado de manera independiente, es decir en un principio debe de haber un servidor independiente para cada uno de ellos.

Paso 1: Preparar servidor Ubuntu para Laravel

En el servidor Laravel:

1. Actualizar sistema:

```
sudo apt update && sudo apt upgrade -y
```

2. Instalar PHP y extensiones necesarias:

```
sudo apt install php php-cli php-mbstring php-xml php-bcmath php-curl php-mysql  
php-zip unzip curl git -y
```

3. Instalar Composer (gestor de dependencias PHP):

```
curl -sS https://getcomposer.org/installer | php  
sudo mv composer.phar /usr/local/bin/composer
```

4. Instalar servidor web

- Apache:

```
sudo apt install apache2 libapache2-mod-php -y
```

5. Clonar o crear el proyecto Laravel:

```
cd /var/www  
sudo composer create-project --prefer-dist laravel/laravel laravel
```

6. Dar permisos:

```
sudo chown -R www-data:www-data laravel  
sudo chmod -R 775 laravel/storage  
sudo chmod -R 775 laravel/bootstrap/cache
```

7. Crea un nuevo VirtualHost:

```
sudo nano /etc/apache2/sites-available/laravel.conf
```

```
<VirtualHost *:80>
```

```
    ServerName laravel.local
```

```
    DocumentRoot /var/www/laravel/public
```

```
<Directory /var/www/laravel/public>
```

```
    AllowOverride All
```

```
    Require all granted
```

```
</Directory>
```

```
ErrorLog ${APACHE_LOG_DIR}/laravel_error.log
```

```
CustomLog ${APACHE_LOG_DIR}/laravel_access.log combined
```

```
</VirtualHost>
```

8. Activar el sitio Laravel y mod_rewrite

```
sudo a2ensite laravel.conf
```

```
sudo a2enmod rewrite
```

9. Editar /etc/apache2/apache2.conf

```
<Directory /var/www/>
```

```
    AllowOverride None → AllowOverride All
```

```
    Require all granted
```

```
</Directory>
```

10. Reiniciar Apache

```
sudo systemctl restart apache2
```

Paso 2: Preparar servidor Ubuntu para MariaDB

En el servidor MariaDB:

1. Actualizar sistema:

```
sudo apt update && sudo apt upgrade -y
```

2. Instalar MariaDB Server:

```
sudo apt install mariadb-server -y
```

3. Proteger instalación y establecer contraseña root:

```
sudo mysql_secure_installation
```

4. Permitir conexión remota (si Laravel está en otro servidor):

- Edita el archivo de configuración:

```
sudo nano /etc/mysql/mariadb.conf.d/50-server.cnf
```

- Cambia:

```
bind-address = 127.0.0.1
```

- por:

```
bind-address = 0.0.0.0
```

5. Reiniciar MariaDB:

```
sudo systemctl restart mariadb
```

6. Crear un usuario remoto para Laravel:

```
sudo mariadb
```

```
CREATE DATABASE laravel_db;
```

```
CREATE USER 'laravel_user'@'%' IDENTIFIED BY 'tu_clave_segura';
```

```
GRANT ALL PRIVILEGES ON laravel_db.* TO 'laravel_user'@'%';
```

```
FLUSH PRIVILEGES;
```

```
EXIT;
```

Paso 3: Conectar Laravel a MariaDB

En el archivo .env de Laravel, pon la IP del servidor MariaDB.

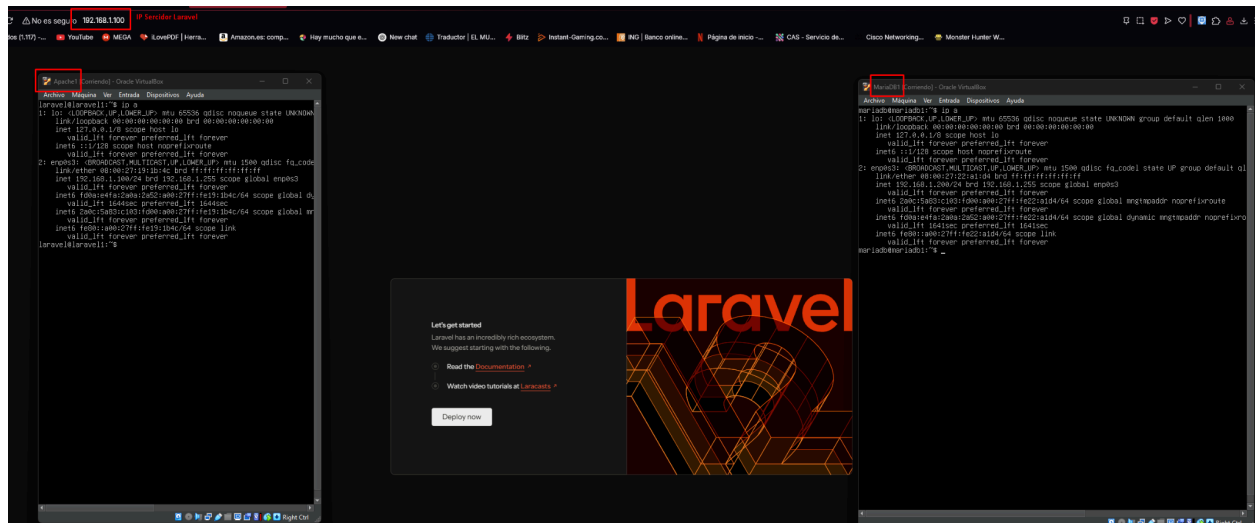
```
DB_CONNECTION=mysql
DB_HOST=IP_DEL_SERVIDOR_MARIADB
DB_PORT=3306
DB_DATABASE=laravel_db
DB_USERNAME=laravel_user
DB_PASSWORD=tu_clave_segura
```

Para comprobarlo pondremos en el servidor laravel/apache

php artisan migrate

Comprobación del servicio.

Hasta ahora, si hemos seguido todos los pasos, desde nuestro navegador cliente podemos acceder a la ip del servidor Laravel, en este caso la 192.168.1.100 y debería salirnos la aplicación preparada.



Parte 2: Añadir un sistema de balanceo de carga al servidor MariaDB usando Galera o MaxScale, por supuesto hacerlo de manera totalmente independiente de Laravel.

Paso a Paso (**hacer en cada nodo**)

1. Actualizar sistema

```
sudo apt update && sudo apt upgrade -y
```

2. Instalar MariaDB + Galera

```
sudo apt install mariadb-server galera-4 -y
```

3. Configurar Galera (en todos los nodos)

Edita:

```
sudo nano /etc/mysql/mariadb.conf.d/60-galera.cnf
```

Agrega o edita lo siguiente:

```
[galera]
```

```
# Cluster name
```

```
wsrep_cluster_name = "galera-cluster"
```

```
# Nodes in el cluster
```

```
wsrep_cluster_address = "gcomm://10.211.22.200,10.211.22.201,10.211.22.202"
```

```
# Node name (CAMBIA en cada servidor)
```

```
wsrep_node_name = galera-node1 # o node2 / node3
```

```
# Node address (IP local de cada nodo)
```

```
wsrep_node_address = 10.211.22.200
```

```
# Database engine
```

```
wsrep_on = ON
```

```
wsrep_provider = /usr/lib/galera/libgalera_smm.so
```

```
# SST method and user
```

```
wsrep_sst_method = rsync
```

4. Permitir conexiones entre nodos

Abre estos puertos en todos los nodos:

```
sudo ufw allow 3306/tcp # MySQL
sudo ufw allow 4567/tcp # Galera replication
sudo ufw allow 4568/tcp # Incremental State Transfer
sudo ufw allow 4444/tcp # State Snapshot Transfer
```

Inicializar el clúster **(sólo en el primer nodo)**

En Nodo 1:

```
sudo systemctl stop mariadb
sudo galera_new_cluster
```

Verifica que esté corriendo:

```
mysql -u root -p -e "SHOW STATUS LIKE 'wsrep_cluster_size';"
```

Debe mostrar 1 como valor.

Conectar los otros nodos

En Nodo 2 y Nodo 3:

```
sudo systemctl stop mariadb
sudo systemctl start mariadb
```

Verifica con:

```
mysql -u root -p -e "SHOW STATUS LIKE 'wsrep_cluster_size';"
```

Debe mostrar 3 como valor cuando todos estén conectados.

Prueba rápida de replicación

Desde cualquier nodo:

```
CREATE DATABASE test_galera;
```

En los otros dos nodos:

```
SHOW DATABASES;
```

Debe aparecer test_galera en todos.

Conexión desde Laravel

Usa la IP de cualquier nodo como host en .env de Laravel:

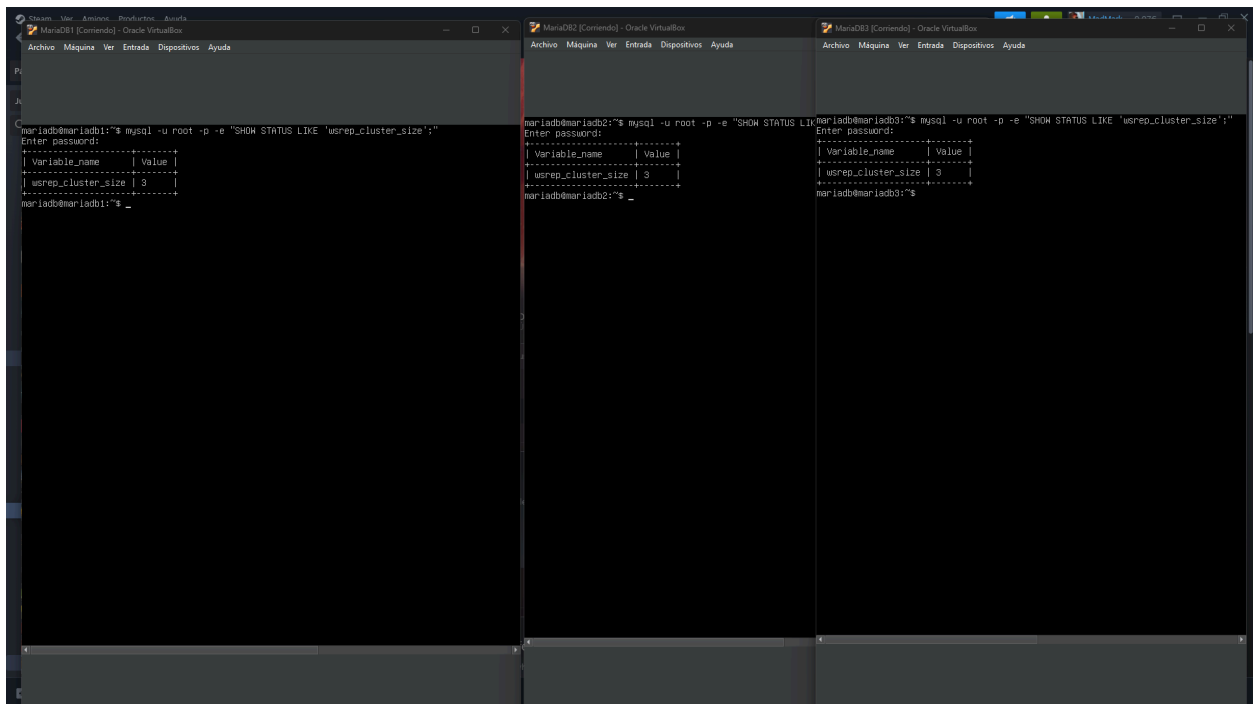
DB_HOST=10.211.22.200 # o cualquiera de los nodos

Caída de servicio.

Los servidores y servicios, en principio, no deberían apagarse pero en caso de que tengamos que apagar las máquinas, para levantarla tendríamos que repetir los pasos de **iniciar el cluster** y en las réplicas **deberíamos reiniciar el servicio**.

Prueba de generación cluster galera

Si todo ha ido bien los tres cluster deben aparecer de la siguiente manera:



```
mariaDB@mariaDB1:~$ mysql -u root -p -e "SHOW STATUS LIKE 'wsrep_cluster_size';"
Enter password:
+-----+
| Variable_name | Value |
+-----+
| wsrep_cluster_size | 3 |
+-----+
mariaDB@mariaDB1:~$

mariaDB@mariaDB2:~$ mysql -u root -p -e "SHOW STATUS LIKE 'wsrep_cluster_size';"
Enter password:
+-----+
| Variable_name | Value |
+-----+
| wsrep_cluster_size | 3 |
+-----+
mariaDB@mariaDB2:~$

mariaDB@mariaDB3:~$ mysql -u root -p -e "SHOW STATUS LIKE 'wsrep_cluster_size';"
Enter password:
+-----+
| Variable_name | Value |
+-----+
| wsrep_cluster_size | 3 |
+-----+
mariaDB@mariaDB3:~$
```

Parte 3: Preparar un sistema de balanceo de carga usando NGINX para los servidores Laravel.

Objetivo:

Tener varios servidores Laravel (por ejemplo, 2 o más) y poner NGINX delante de ellos como balanceador de carga para repartir tráfico y mejorar disponibilidad.

Estructura:

Componente	IP	Función
Laravel App 1	10.211.22.100	Servidor Laravel
Laravel App 2	10.211.22.101	Otro servidor Laravel
NGINX Load Balancer	10.211.22.150	Balancea tráfico hacia ambos

Paso a Paso

Tener dos (o más) servidores Laravel funcionando

- Pueden ser copias exactas del que hiciste en el Punto 1
- Deben tener sus .env correctamente configurados y estar escuchando en puerto 80 o 8000, por ejemplo

Configurar el servidor NGINX (balanceador)

En el servidor balanceador (IP: 10.211.22.150):

1. Instalar NGINX:

```
sudo apt update && sudo apt install nginx -y
```

2. Configura el archivo del balanceador:

```
sudo nano /etc/nginx/sites-available/laravel-lb
```

Contenido de ejemplo:

```
upstream laravel_cluster {  
    server 10.211.22.100;  
    server 10.211.22.101;  
}  
  
server {  
    listen 80;  
  
    location / {  
        proxy_pass http://laravel_cluster;  
        proxy_set_header Host $host;  
        proxy_set_header X-Real-IP $remote_addr;  
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;  
    }  
}
```

3. Activar la configuración:

```
sudo rm /etc/nginx/sites-enabled/default  
sudo ln -s /etc/nginx/sites-available/laravel-lb /etc/nginx/sites-enabled/  
sudo nginx -t  
sudo systemctl reload nginx
```

Prueba

En un navegador, accede a <http://10.211.22.150>

Deberías ver la app Laravel

Para probar el balanceo, cambia algo en la vista `welcome.blade.php` en un nodo y recarga varias veces → debería ir alternando

Opcional: Balanceo por peso, salud, etc.

Puedes afinarlo con:

```
upstream laravel_cluster {  
    server 192.168.1.201 weight=3;  
    server 192.168.1.202 backup;  
}
```

Ejemplo completo con balanceo por peso y estado de salud (simple):

```
upstream laravel_cluster {  
    # balanceo por peso: node1 recibe 3 veces más tráfico  
    server 192.168.1.100 weight=3;  
    server 192.168.1.101 weight=1;  
  
    # si quieres marcar uno como backup (solo entra si los demás fallan):  
    # server 192.168.1.203 backup;  
}  
  
server {  
    listen 80;  
    server_name _;  
    location / {  
        proxy_pass http://laravel_cluster;  
        proxy_set_header Host $host;  
        proxy_set_header X-Real-IP $remote_addr;  
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;  
  
        # timeout opcional  
        proxy_connect_timeout 2;  
        proxy_read_timeout 5;  
    }  
}
```

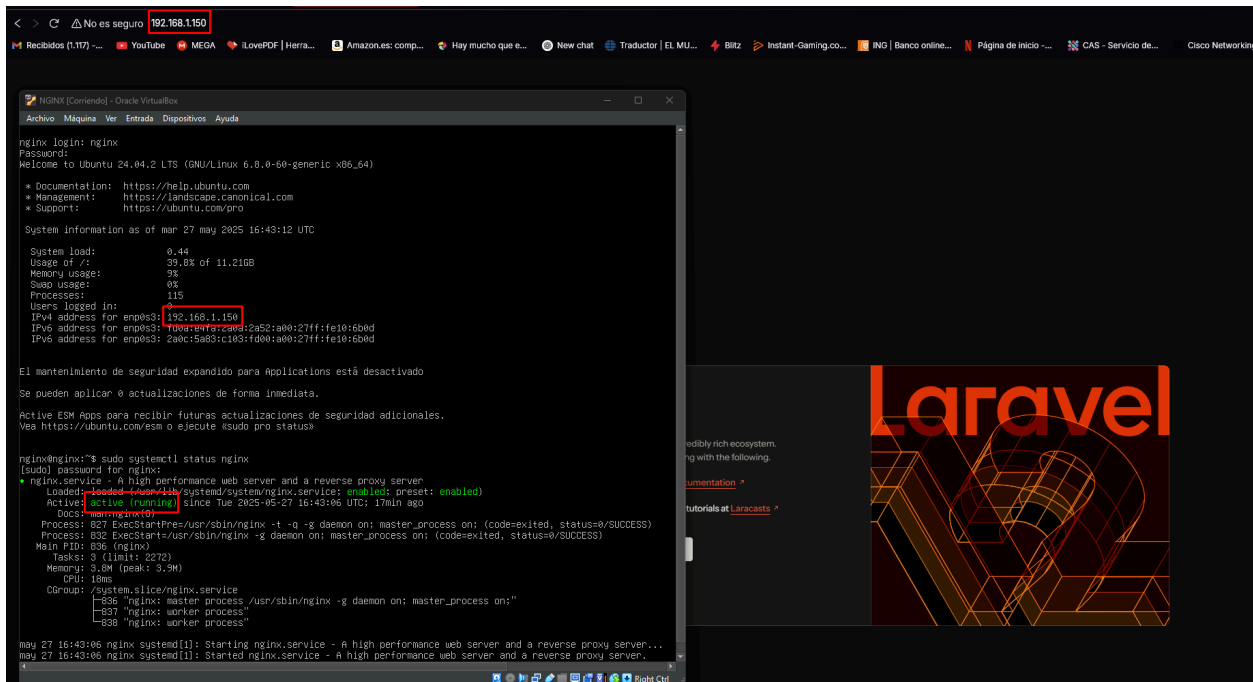
Laravel listo con balanceo

Con esto ya tenemos:

- Balanceo de carga HTTP
- Laravel replicado
- Mejor disponibilidad y escalabilidad

Prueba funcionamiento NGINX

Si todo ha ido como debería, al poner la IP de nuestro servidor con NGINX debería conectarse con Laravel igual que lo haría si pusiéramos la IP de cualquier máquina con laravel anteriormente creada.



```

192.168.1.150
nginx login: nginx
Password:
Welcome to Ubuntu 24.04.2 LTS (GNU/Linux 6.8.0-generic x86_64)

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

System information as of mon 27 may 2025 16:43:12 UTC

System load: 0.44
Usage of /: 39.8% of 11.21GB
Memory usage: 3%
Swap usage: 0%
Processes: 115
Users logged in: 0
IPV4 address for enp0s3: 192.168.1.150
IPV6 address for enp0s3: fd00::2a52:a00:27ff:fe10:6b0d
IPV6 address for enp0s3: 2a0c:5a83:c103:f000:a00:27ff:fe10:6b0d

El mantenimiento de seguridad expandido para Applications está desactivado.
Se pueden aplicar 0 actualizaciones de forma inmediata.
Active ESM Apps para recibir futuras actualizaciones de seguridad adicionales.
Vea https://ubuntu.com/esm o ejecute #sudo pro status

nginx@nginx:~$ sudo systemctl status nginx
[sudo] password for nginx:
● nginx.service - A high performance web server and a reverse proxy server
   Loaded: loaded (/usr/lib/systemd/system/nginx.service; enabled; preset: enabled)
   Active: active (running) since Tue 2025-05-27 16:43:06 UTC; 17min ago
     Docs: man:nginx(8)
    Process: 827 ExecStartPre=/usr/sbin/nginx -t -q -g daemon on; master_process on; (code=exited, status=0/SUCCESS)
    Process: 832 ExecStart=/usr/sbin/nginx -g daemon on; master_process on; (code=exited, status=0/SUCCESS)
   Main PID: 836 (nginx)
    Tasks: 3 (limit: 2272)
   Memory: 3.0M (peak: 3.9M)
      CPU: 10ms
   CGroup: /system.slice/nginx.service
           └─836 "nginx: master process /usr/sbin/nginx -g daemon on; master_process on;"
             └─837 "nginx: worker process"
               └─838 "nginx: worker process"

may 27 16:43:06 nginx systemd[1]: Starting nginx.service - A high performance web server and a reverse proxy server...
may 27 16:43:06 nginx systemd[1]: Started nginx.service - A high performance web server and a reverse proxy server.

```

Parte 4: Dockerizar toda la configuración una vez que los sistemas estén funcionando.

Tener un entorno reproducible con:

1. Laravel App en contenedor PHP-FPM
2. MariaDB en contenedor
3. NGINX como proxy reverso / balanceador (o frontal web)
4. Listo para ampliarlo más adelante con Galera o Gitea

Estructura propuesta del proyecto Docker.

proyecto-dockerizado/

```
|— docker-compose.yml
|— Dockerfile
|— laravel-app/
|   |— (aquí va el código de Laravel)
|   |— .env
|— nginx/
|   |— default.conf
|— mariadb-config/
|   |— galera-db1.cnf
|   |— galera-db2.cnf
|   |— galera-db3.cnf
|   |— galera.cnf
|   |— init.sql
|— apache
|   |— 000-default.conf
|— gitea
|   |— data
```

Vamos a crear:

1. docker-compose.yml: Orquesta Laravel, MariaDB y NGINX.
2. Dockerfile para Laravel: Construye el contenedor con PHP + Laravel + Apache.

3. Configuración de NGINX (reverse proxy): Para servir Laravel y balancear si agregamos más apps.
4. Distintas configuraciones.

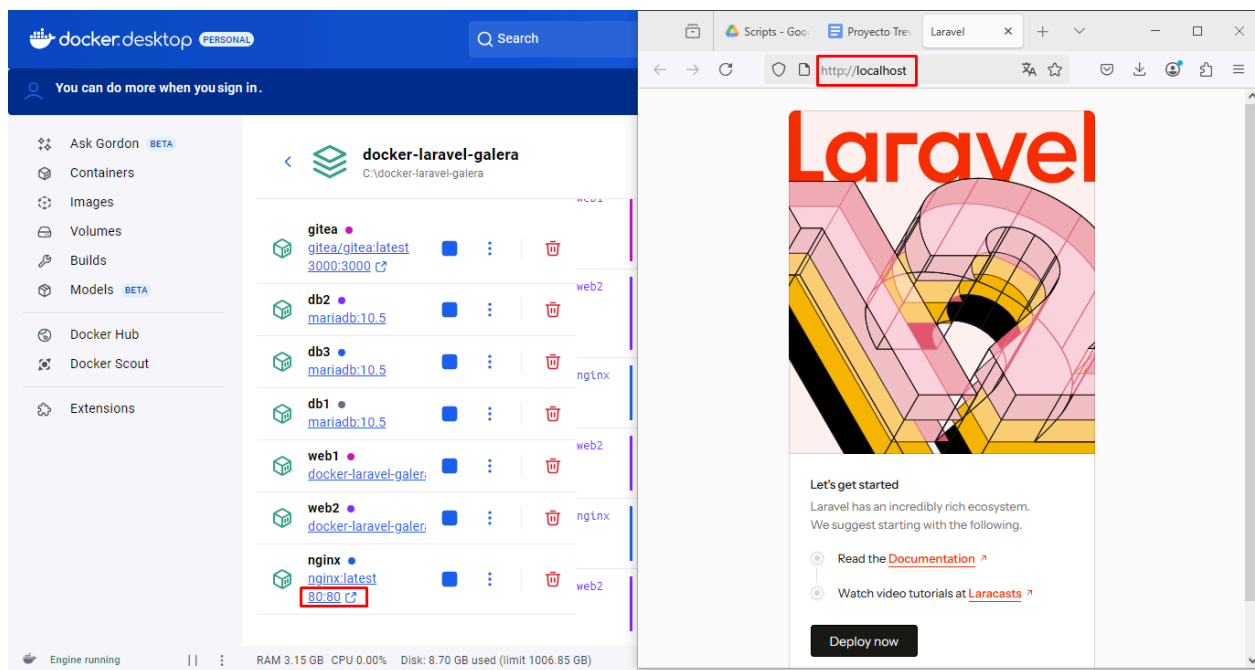
Adjunto un enlace a drive que donde podremos encontrar todo el proyecto dockerizado en un formato .zip listo para ser desplegado.

[Proyecto-Dockerizado](#)

Instrucciones para ejecutarlo.

1. Coloca tu proyecto Laravel en la carpeta laravel/
2. `docker-compose up --build -d`
3. Accede a: `http://localhost`

Si todo ha salido bien deberíamos ver esta estructura y comprobaremos su funcionamiento:



Parte 5: Instalar y configurar un servidor GIT para el control del proyecto, preparar un día para informar a los desarrolladores sobre el uso de esta herramienta.

Paso 1: Modificar .yml.

Para instalar y configurar un servidor GIT en docker añadiremos en el .yml de docker-composer el servicio de git.

El código sería:

```
gitea:
  image: gitea/gitea:latest
  container_name: gitea
  restart: unless-stopped
  environment:
    USER_UID: 1000
    USER_GID: 1000
    GITEA__APP_NAME: "Servidor GIT Interno"
    GITEA__server__DOMAIN: localhost
    GITEA__server__ROOT_URL: http://localhost:3000/
    GITEA__server__HTTP_PORT: 3000
    GITEA__database__DB_TYPE: sqlite3
  volumes:
    - ./gitea/data:/data
  ports:
    - "3000:3000"
  networks:
    - frontend
    - backend
```

Recordemos que las redes tienen que estar acorde a todo el proyecto de docker.

Paso 2: Crear directorios.

Una vez hecho esto dentro del proyecto crearemos una carpeta para git, en este caso gitea con:

```
mkdir -p gitea/data
```

Esta carpeta almacenará los datos persistentes de Gitea, como:

- Repositorios Git
- Configuración
- Avatares, usuarios, etc.

Paso 3: Iniciar Gitea.

Desde el directorio raíz del proyecto, lanza Gitea:

```
docker-compose up -d gitea
```

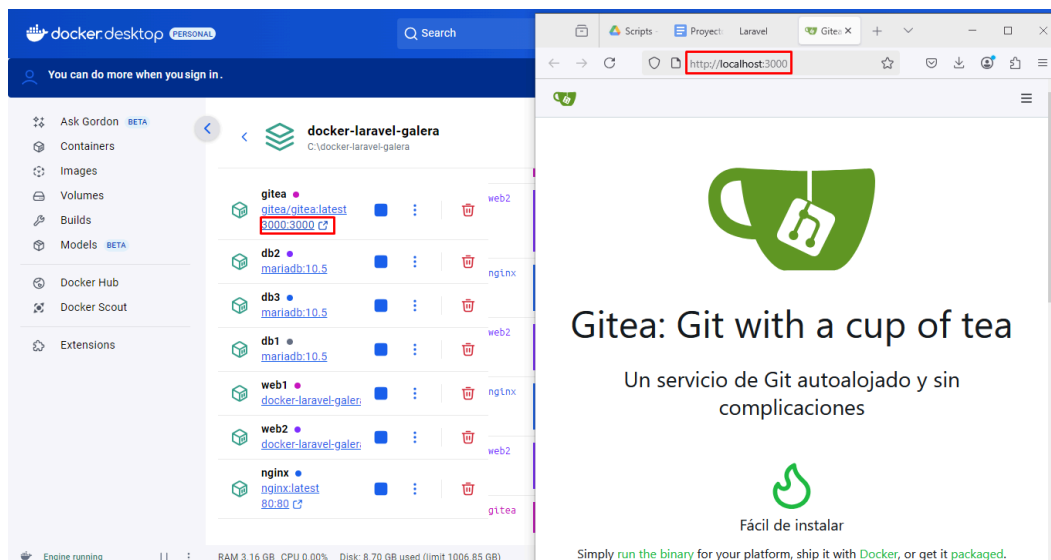
Parte 4: Accede al servidor Gitea.

Abre tu navegador en:

```
http://localhost:3000
```

Y verás el asistente de instalación de Gitea. Una vez hecho esto ya tendríamos instalado y configurado un servidor Git, quedaría iniciar sesión y aprovechar el potencial de esta herramienta.

Gitea también debería funcionar perfectamente:



Parte 6: Preparar todos los scripts necesarios (python, bash, node) para el control de las configuraciones.

En este apartado vamos a crear distintos scripts en bash para automatizar tanto el proceso en máquinas reales como en las máquinas virtuales.

Máquinas virtuales.

Para los scripts tenemos que tener nuestras configuraciones de red muy claras. Dada el aula en el que estamos trabajando nuestra red es la 10.211.0.0/16, nosotros estamos cogiendo el rango 10.211.22.0/16 con lo cual tenemos que jugar con esos rangos de red. Vamos a hacer un resumen de cómo tenemos organizadas las máquinas y sus IPs:

Componente	IP	Función
Laravel App 1	10.211.22.100	Servidor Laravel
Laravel App 2	10.211.22.101	Otro servidor Laravel
NGINX Load Balancer	10.211.22.150	Balancea tráfico hacia ambos
Nodo Galera 1	10.211.22.200	Servidor BD
Nodo Galera 2	10.211.22.201	Otro Servidor BD
Nodo Galera 3	10.211.22.202	Otro Servidor BD

Aclarado esto vamos a proceder con la creación del script que lo organizaremos de tal manera que, como son máquinas virtuales, cada máquina tenga su propio script ya que cada una requiere configuraciones diferentes (a excepción del servidor Laravel qué es exactamente igual). Para no emborronar el documento voy a dejar los enlaces que te llevan a los distintos scripts:

- [Laravel](#)
- [Nginx](#)
- [Nodo1](#)
- [Nodo2](#)
- [Nodo3](#)

Cada script funciona perfectamente teniendo en cuenta la red que le hemos puesto. Se podría realizar cambios de red en el propio script dependiendo de las necesidades.

Proyecto Dockerizado.

Para la parte de automatizar el proyecto de docker lo que haremos será replicar por comandos lo que hicimos para Docker Desktop en Windows 10.

Generamos la estructura de carpetas, el proyecto Laravel, del docker-composer.yml y el Dockerfile además de, por supuesto, todo los archivos de configuración pertinentes.

⚠ TENEMOS QUE TENER EN CUENTA ⚠

Necesitamos un mínimo de interacción manual previa como es la instalación del propio Docker Desktop (que podéis descargar en el siguiente [enlace](#)) y la instalación de Composer para crear el proyecto de Laravel (que podéis descargar en el siguiente [enlace](#))

Además, antes de la ejecución del script debemos tener abierto el programa de docker para que cuando lancemos el comando para levantar los contenedores no haya ningún problema.

Como hicimos anteriormente, ya que el script es bastante largo, dejaré un enlace para no emborronar todo el documento

- [ScriptDocker](#)

Conclusión.

Los scripts que hemos creado se usarán para automatizar toda la instalación de entorno servidor que hemos estado desarrollando hasta ahora tanto en máquinas virtuales/reales como en un entorno como es Docker.

Teniendo en cuenta las consideraciones que hemos especificado anteriormente todo debería estar diseñado para ser lo más rápido y óptimo posible.

Conclusión final del proyecto.

En este proyecto hemos encontrado algunas dificultades puesto que, prácticamente, todos los contenidos que se nos proponían eran nuevos para nosotros.

Al final logramos cumplir con todos los objetivos propuestos en nuestro apartado con lo cual siento gran satisfacción de haberlo completado y de haberme embarcado en esta aventura completamente nueva para mi.

En cuanto al desarrollo de destrezas que adquirido se encuentran:

- Conocimiento de distintos servicios como Galera o NGINX.
- Conocimiento general en Docker y contenedores.
- Conocimiento mayor en el desarrollo de Script y automatización de tareas.

En cuanto aptitudes adquiridas encontramos:

- Mayor soltura a la hora de buscar documentación desconocidas
- Trabajo en equipo
- Uso de nuevas tecnologías para desarrollar el trabajo más eficazmente.

Dejo por aquí el folder de drive con absolutamente todo el proyecto: [ProyectoTrevenque](#).

Bibliografía del Proyecto.

A continuación dejaremos por aquí las distintas fuentes de información que hemos usado para el desarrollo del proyecto. Destacar también que el uso de nuevas tecnologías como la inteligencia artificial ha sido fundamental para el desarrollo rápido y eficiente del mismo el cual cada vez se está implementando más en el entorno laboral.

Vayamos, ahora sí, con la bibliografía:

- [Laravel](#): Información y documentación acerca de esta herramienta.
- [PHP](#): Información acerca de PHP.
- [Composer](#): Información y documentación acerca de esta herramienta.
- [MariaDB](#): ¿Qué es MariaDb y como se instala?
- [Galera Cluster](#): ¿Qué es y cómo se usa?
- [NGINX](#): Función como balanceador de carga.
- [Docker](#): Tutorial y usos de esta herramienta.
- [Docker Hub](#): Página que recoge todas las imágenes para docker.
- [Script Bash](#): Guia básica para empezar a hacer scripts en bash.
- [Script PWS](#): Guia básica para empezar a hacer scripts en PowerShell.