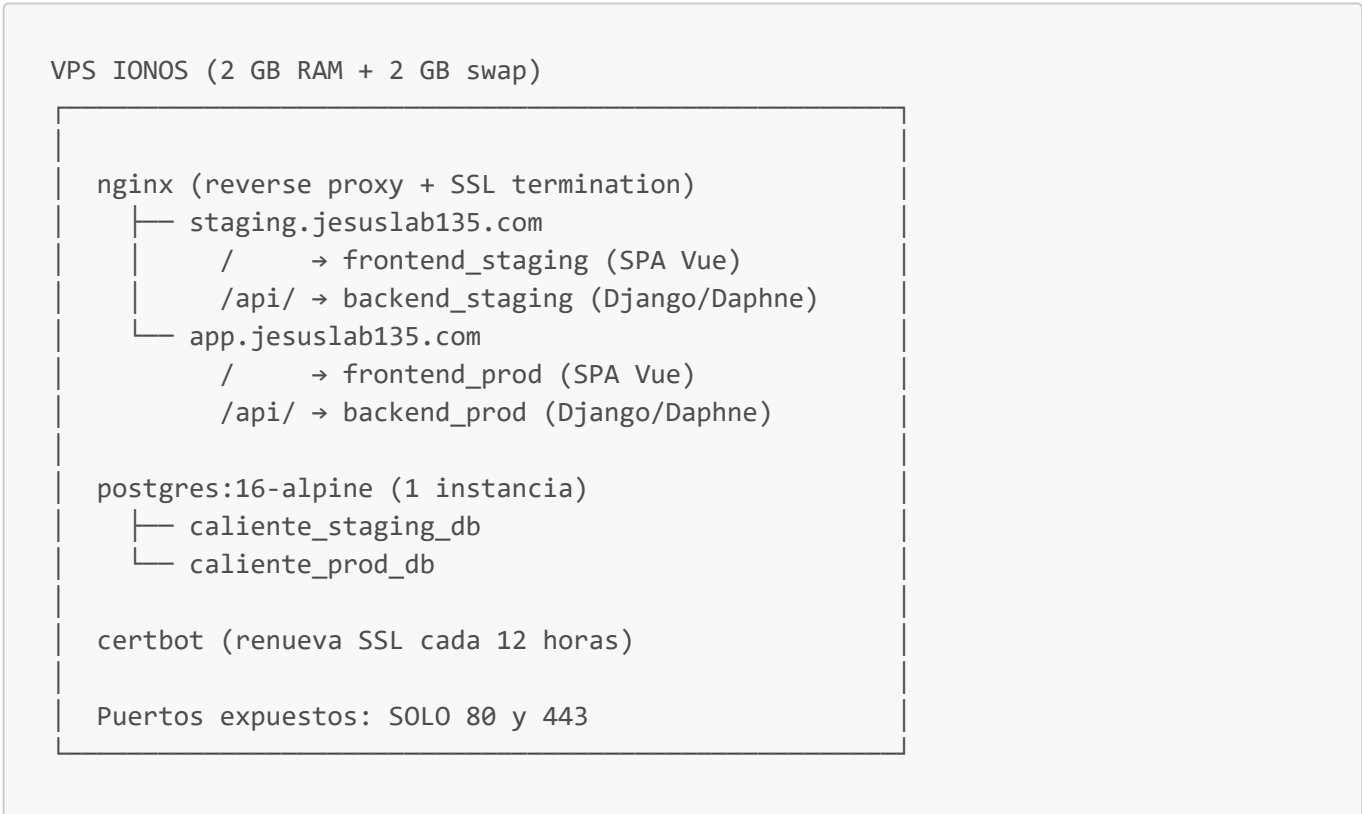
Guia de Despliegue Manual — Caliente en IONOS VPS

> **Prerequisito**: Todos los archivos de infraestructura ya fueron creados en el proyecto. Esta guia cubre UNICAMENTE los pasos manuales que debes ejecutar tu.

Tabla de Contenidos

1. Vista General de la Arquitectura



Flujo de deploy automatico:

```
Push a develop  →  GitHub Actions  →  Build imagen Docker  →  Push a ghcr.io  →
SSH al VPS  →  Pull + restart  →  staging.jesuslab135.com
Push a main     →  GitHub Actions  →  Build imagen Docker  →  Push a ghcr.io  →
SSH al VPS  →  Pull + restart  →  app.jesuslab135.com
```

---

2. Archivos Ya Creados (Referencia)

Estos archivos ya existen en el proyecto. **No necesitas crearlos**, solo subirlos al VPS.

Archivo	Ubicacion	Que hace
`docker-compose.yml`	`infra/vps/`	Orquesta los 7 contenedores en el VPS
`init-db.sh`	`infra/vps/`	Crea la base de datos de staging (se ejecuta 1 sola vez)
`init-letsencrypt.sh`	`infra/vps/`	Genera certificados SSL por primera vez
`staging.conf`	`infra/vps/nginx/`	Nginx: enruta staging.jesuslab135.com
`prod.conf`	`infra/vps/nginx/`	Nginx: enruta app.jesuslab135.com
`.env.example`	`infra/vps/`	Template para variables de docker-compose
`.env.staging.example`	`infra/vps/`	Template para variables del backend staging
`.env.prod.example`	`infra/vps/`	Template para variables del backend produccion
`deploy.yml`	`frontend/.github/workflows/`	Pipeline CI/CD del frontend
`deploy.yml`	`backend/.github/workflows/`	Pipeline CI/CD del backend

3. PASO 1: Configurar DNS en IONOS

> **Tiempo estimado**: 5 minutos de configuracion + 5-30 minutos de propagacion

3.1 — Entrar al panel de IONOS

1. Abre https://my.ionos.com e inicia sesion
2. Ve a **Domains & SSL** en el menu lateral
3. Haz clic en tu dominio `jesuslab135.com`
4. Haz clic en la pestana **DNS**

3.2 — Obtener la IP de tu VPS

1. En IONOS, ve a **Servers & Cloud** (o **Servidores**)
2. Haz clic en tu VPS

3. Copia la **IPv4 Address** (ejemplo: `82.165.123.456`)

3.3 — Crear los registros A

Haz clic en **Add Record** (o **Agregar registro**) y crea estos DOS registros:

Campo	Registro 1	Registro 2
Type	A	A
Host	`staging`	`app`
Points to	`TU_IP_DEL_VPS`	`TU_IP_DEL_VPS`
TTL	1 Hour	1 Hour

> **Nota**: Si ya existe un registro A para `staging` o `app`, editalo en vez de crear uno nuevo.

3.4 — Verificar la propagacion

Espera 5-30 minutos y luego abre una terminal en tu **computadora local** (no el VPS):

```
# Windows (PowerShell)
nslookup staging.jesuslab135.com
nslookup app.jesuslab135.com

# Mac/Linux
dig staging.jesuslab135.com +short
dig app.jesuslab135.com +short
```

Resultado esperado: Ambos deben mostrar la IP de tu VPS.

> **Si no resuelve despues de 30 minutos**: Prueba con https://dnschecker.org — busca tu subdominio y verifica que propago globalmente. Algunos ISPs tardan mas.

---

4. PASO 2: Preparar el VPS

> **Tiempo estimado**: 10-15 minutos

4.1 — Conectarte al VPS por SSH

```
# Desde tu computadora local
ssh root@TU_IP_DEL_VPS
```

> **Primera vez**: IONOS te da la password de root por email o en el panel. Usala para conectarte.

4.2 — Actualizar el sistema

```
apt-get update && apt-get upgrade -y
```

4.3 — Instalar Docker Engine + Docker Compose

```
# Instalar dependencias necesarias
apt-get install -y ca-certificates curl gnupg

# Agregar la llave GPG oficial de Docker
install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | gpg --dearmor -o
/etc/apt/keyrings/docker.gpg
chmod a+r /etc/apt/keyrings/docker.gpg

# Agregar el repositorio de Docker
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  tee /etc/apt/sources.list.d/docker.list > /dev/null

# Instalar Docker + Compose
apt-get update
apt-get install -y docker-ce docker-ce-cli containerd.io docker-buildx-plugin
docker-compose-plugin
```

4.4 — Verificar la instalacion

```
docker --version
# Esperado: Docker version 27.x.x

docker compose version
# Esperado: Docker Compose version v2.x.x
```

4.5 — Configurar el Firewall (UFW)

```
# PRIMERO: Permitir SSH (si no haces esto primero, te puedes quedar fuera del
servidor)
ufw allow OpenSSH

# Permitir HTTP y HTTPS para nginx
ufw allow 80/tcp
ufw allow 443/tcp

# Activar el firewall
ufw --force enable
```

```
# Verificar las reglas
ufw status
```

Resultado esperado:

```
Status: active

To                         Action      From
--                         ------      ----
OpenSSH                    ALLOW       Anywhere
80/tcp                     ALLOW       Anywhere
443/tcp                    ALLOW       Anywhere
```

4.6 — Crear 2 GB de Swap

Tu VPS tiene solo 2 GB de RAM. El swap previene que se congele durante builds o picos de uso.

```
# Crear archivo de swap de 2 GB
fallocate -l 2G /swapfile
chmod 600 /swapfile
mkswap /swapfile
swapon /swapfile

# Hacer permanente (sobrevive reinicios del servidor)
echo '/swapfile none swap sw 0 0' >> /etc/fstab

# Verificar
free -h
```

Resultado esperado (la linea de Swap):

```
               total        used        free
Mem:           1.9Gi       ...         ...
Swap:          2.0Gi       0B          2.0Gi
```

4.7 — Crear el directorio del proyecto

```
mkdir -p /opt/caliente/nginx
```

---

5. PASO 3: Generar Llave SSH para GitHub Actions

> GitHub Actions necesita poder conectarse al VPS para hacer deploy. Para eso usamos una llave SSH.

5.1 — Generar la llave (en tu computadora LOCAL, no el VPS)

Abre una terminal en tu computadora:

```
# Generar un par de llaves Ed25519
ssh-keygen -t ed25519 -C "github-actions-deploy" -f ~/.ssh/caliente_deploy
```

Cuando te pregunte por passphrase, **deja en blanco** (presiona Enter dos veces). GitHub Actions no puede introducir passphrase.

Esto crea dos archivos:

- `~/.ssh/caliente_deploy` — **LLAVE PRIVADA** (para GitHub Secrets)
- `~/.ssh/caliente_deploy.pub` — **LLAVE PUBLICA** (para el VPS)

5.2 — Agregar la llave publica al VPS

```
# Copiar la llave publica al servidor
ssh-copy-id -i ~/.ssh/caliente_deploy.pub root@TU_IP_DEL_VPS
```

> **Si `ssh-copy-id` no funciona en Windows**: Hazlo manualmente:
>
> ```
> # 1. Mostrar la llave publica
> cat ~/.ssh/caliente_deploy.pub
> # Copia el texto que aparece
>
> # 2. Conectarte al VPS
> ssh root@TU_IP_DEL_VPS
>
> # 3. Agregar la llave
> echo "PEGA_AQUI_LA_LLAVE_PUBLICA" >> ~/.ssh/authorized_keys
> ```

5.3 — Verificar que funciona

```
# Deberia conectarte SIN pedir password
ssh -i ~/.ssh/caliente_deploy root@TU_IP_DEL_VPS
```

Si entras sin password, esta listo. Escribe `exit` para salir.

5.4 — Guardar la llave privada (la necesitaras en el Paso 7)

```
# Mostrar la llave privada completa
cat ~/.ssh/caliente_deploy
```

Copia TODO el contenido, incluyendo las lineas `-----BEGIN OPENSSH PRIVATE KEY-----` y `-----END OPENSSH PRIVATE KEY-----`. La necesitaras mas adelante para GitHub Secrets.

---

6. PASO 4: Subir Archivos de Infraestructura al VPS

> Estos archivos ya existen en tu proyecto local en `infra/vps/`. Solo necesitas copiarlos al VPS.

6.1 — Copiar archivos con SCP

Desde tu **computadora local**, en la carpeta raiz del proyecto (`CalienteFinal/`):

```
# Archivo principal de orquestacion
scp infra/vps/docker-compose.yml root@TU_IP_DEL_VPS:/opt/caliente/

# Script de inicializacion de base de datos
scp infra/vps/init-db.sh root@TU_IP_DEL_VPS:/opt/caliente/

# Script de generacion de SSL
scp infra/vps/init-letsencrypt.sh root@TU_IP_DEL_VPS:/opt/caliente/

# Configuraciones de Nginx
scp infra/vps/nginx/staging.conf root@TU_IP_DEL_VPS:/opt/caliente/nginx/
scp infra/vps/nginx/prod.conf root@TU_IP_DEL_VPS:/opt/caliente/nginx/

# Plantillas de variables de entorno (los editaremos en el siguiente paso)
scp infra/vps/.env.example root@TU_IP_DEL_VPS:/opt/caliente/.env
scp infra/vps/.env.staging.example root@TU_IP_DEL_VPS:/opt/caliente/.env.staging
scp infra/vps/.env.prod.example root@TU_IP_DEL_VPS:/opt/caliente/.env.prod
```

> **Tip para Windows**: Si `scp` no funciona, puedes usar FileZilla (SFTP) o WinSCP con la IP del VPS, usuario `root`, y tu password.

6.2 — Hacer scripts ejecutables

```
# Conectarte al VPS
ssh root@TU_IP_DEL_VPS

# Dar permisos de ejecucion
chmod +x /opt/caliente/init-letsencrypt.sh
chmod +x /opt/caliente/init-db.sh
```

6.3 — Verificar que todo se copio bien

```
ls -la /opt/caliente/
```

Resultado esperado:

```
docker-compose.yml
init-db.sh
init-letsencrypt.sh
.env
.env.staging
.env.prod
nginx/
  staging.conf
  prod.conf
```

---

7. PASO 5: Configurar Variables de Entorno (.env)

> **MUY IMPORTANTE**: Los archivos `.env` contienen passwords. NUNCA los subas a GitHub.

7.1 — Generar passwords seguras

Antes de editar, genera 3 passwords aleatorias. Puedes usar este comando:

```
# En el VPS, ejecuta esto 3 veces para generar 3 passwords diferentes
openssl rand -base64 32
```

Anota las 3 passwords. Las usaras para:

1. **POSTGRES_PASSWORD** — password de la base de datos
2. **SECRET_KEY staging** — secret key de Django staging
3. **SECRET_KEY prod** — secret key de Django produccion

7.2 — Editar `.env` (variables de Docker Compose)

```
nano /opt/caliente/.env
```

Cambia estos valores:

```
# Pon la password de PostgreSQL que generaste
POSTGRES_PASSWORD=tu-password-postgres-generada-aqui

# Tu usuario de GitHub (en minusculas)
# Ejemplo: si tu repo es github.com/JesusLab135/caliente-backend → pon jesuslab135
GITHUB_OWNER=tu-usuario-github-en-minusculas
```

Guarda: `Ctrl+O` → `Enter` → `Ctrl+X`

7.3 — Editar `.env.staging` (variables del backend staging)

```
nano /opt/caliente/.env.staging
```

Cambia estos valores:

```
DEBUG=True
SECRET_KEY=tu-secret-key-staging-generada-aqui
ALLOWED_HOSTS=staging.jesuslab135.com
CORS_ALLOWED_ORIGINS=https://staging.jesuslab135.com
FRONTEND_DOMAIN=staging.jesuslab135.com

DB_NAME=caliente_staging_db
DB_USER=caliente
DB_PASSWORD=tu-password-postgres-generada-aqui   # <-- LA MISMA que
POSTGRES_PASSWORD
HOST=postgres
PORT=5432
```

> **IMPORTANTE**: `DB_PASSWORD` debe ser EXACTAMENTE igual a `POSTGRES_PASSWORD` del archivo `.env`. Son la misma base de datos.

7.4 — Editar `.env.prod` (variables del backend produccion)

```
nano /opt/caliente/.env.prod
```

```
DEBUG=False
SECRET_KEY=tu-secret-key-produccion-generada-aqui   # <-- DIFERENTE al de staging
ALLOWED_HOSTS=app.jesuslab135.com
CORS_ALLOWED_ORIGINS=https://app.jesuslab135.com
FRONTEND_DOMAIN=app.jesuslab135.com

DB_NAME=caliente_prod_db
DB_USER=caliente
DB_PASSWORD=tu-password-postgres-generada-aqui   # <-- LA MISMA que
POSTGRES_PASSWORD
HOST=postgres
PORT=5432
```

7.5 — Resumen de que password va donde

```
POSTGRES_PASSWORD (en .env)     = "abc123..."
DB_PASSWORD (en .env.staging)   = "abc123..."  ← MISMA
```

```
DB_PASSWORD (en .env.prod)      = "abc123..."  ← MISMA
SECRET_KEY (en .env.staging)    = "xyz456..."  ← UNICA
SECRET_KEY (en .env.prod)       = "mno789..."  ← UNICA (diferente a staging)
```

---

8. PASO 6: Levantar PostgreSQL y Generar SSL

> **Prerequisito**: El DNS ya debe estar propagado (Paso 1 completo).

8.1 — Levantar PostgreSQL primero

```
cd /opt/caliente

# Levantar solo postgres para que cree las bases de datos
docker compose up -d postgres

# Esperar 10 segundos a que arranque
sleep 10

# Verificar que creo las dos bases de datos
docker compose logs postgres | grep "caliente_staging_db"
```

Resultado esperado: Debes ver un mensaje como `"Base de datos caliente_staging_db creada exitosamente."` o el log de `CREATE DATABASE`.

8.2 — Generar certificados SSL

```
cd /opt/caliente

# Ejecutar el script de SSL
./init-letsencrypt.sh
```

Que hace este script (paso a paso):

1. **Guarda** las configs reales de nginx (staging.conf, prod.conf) como `.bak`
2. **Crea** configs temporales que solo sirven el reto ACME de Let's Encrypt
3. **Levanta** nginx con las configs temporales
4. **Ejecuta** certbot para generar certificados para ambos subdominios
5. **Restaura** las configs reales con SSL habilitado
6. **Reinicia** nginx

Resultado esperado al final:

```
==========================================
 SSL configurado exitosamente!
 - https://staging.jesuslab135.com
```

```
 - https://app.jesuslab135.com
==========================================
```

> **Si falla con "DNS problem"**: El DNS no ha propagado todavia. Espera mas tiempo y vuelve a intentar. Puedes verificar con:
>
> ```
> curl -v http://staging.jesuslab135.com/.well-known/acme-challenge/test
> # Si ves "connection refused" o "Could not resolve", el DNS no ha propagado
> ```

8.3 — Verificar que los certificados se generaron

```
docker compose exec nginx ls /etc/letsencrypt/live/
```

Resultado esperado: Debes ver dos carpetas:

```
staging.jesuslab135.com
app.jesuslab135.com
```

---

9. PASO 7: Configurar GitHub Secrets

> GitHub Actions usa "Secrets" para almacenar datos sensibles (IP del server, llave SSH, etc.) Debes configurarlos en **AMBOS** repositorios (backend Y frontend).

9.1 — Que secrets necesitas

Nombre del Secret	Valor	De donde lo sacas
`VPS_HOST`	La IP de tu VPS (ej. `82.165.123.456`)	Panel de IONOS → tu VPS → IPv4
`VPS_USER`	`root`	Es el usuario por defecto
`VPS_SSH_KEY`	La llave privada completa	El archivo `~/.ssh/caliente_deploy` (ver Paso 3.4)

> `GITHUB_TOKEN` NO necesitas crearlo. GitHub lo genera automaticamente en cada workflow.

9.2 — Agregar secrets al repo BACKEND

1. Abre tu navegador y ve a tu repo de backend en GitHub
2. Haz clic en **Settings** (pestana superior)
3. En el menu lateral, busca **Secrets and variables** → **Actions**
4. Haz clic en **New repository secret**

Repite para cada secret:

Secret 1: VPS_HOST

- Name: `VPS_HOST`
- Secret: `82.165.123.456` (tu IP real)
- Click **Add secret**

Secret 2: VPS_USER

- Name: `VPS_USER`
- Secret: `root`
- Click **Add secret**

Secret 3: VPS_SSH_KEY

- Name: `VPS_SSH_KEY`
- Secret: (pega TODO el contenido de `~/.ssh/caliente_deploy`)
- **IMPORTANTE**: Incluye las lineas `-----BEGIN` y `-----END`
- Click **Add secret**

9.3 — Agregar los MISMOS secrets al repo FRONTEND

Repite exactamente el paso 9.2 pero en el repositorio del frontend. Son los mismos 3 secrets con los mismos valores.

9.4 — Verificar

En cada repo, ve a Settings → Secrets → Actions. Debes ver:

```
VPS_HOST        Updated just now
VPS_USER        Updated just now
VPS_SSH_KEY     Updated just now
```

---

10. PASO 8: Primer Deploy — Staging

> **Nota**: El primer deploy va a fallar parcialmente porque los contenedores de backend/frontend aun no tienen imagen en GHCR. Eso es normal — los crearemos ahora.

10.1 — Hacer las imagenes publicas en GHCR (una sola vez)

Despues de que GitHub Actions suba la primera imagen, los paquetes se crean como **privados** por defecto. Para que el VPS pueda descargarlos:

1. Ve a https://github.com/TU_USUARIO?tab=packages
2. Haz clic en cada paquete (`caliente-backend`, `caliente-frontend`)
3. Ve a **Package settings** (parte inferior derecha)
4. Cambia **Visibility** a **Public** (o configura un PAT con `read:packages` en el VPS)

> **Alternativa si quieres mantenerlos privados**: En el VPS, loguea Docker a GHCR:
>
> ```
> echo "TU_GITHUB_PAT" | docker login ghcr.io -u TU_USUARIO --password-stdin
> ```
>
> Para crear el PAT: GitHub → Settings → Developer Settings → Personal Access Tokens → Generate new token → marcar `read:packages`.

10.2 — Push backend a develop

En tu computadora local, en el repo del **backend**:

```
cd backend

# Crear la rama develop si no existe
git checkout -b develop

# Si ya existe:
# git checkout develop
# git merge main

# Subir a GitHub
git push -u origin develop
```

10.3 — Verificar el pipeline del backend

1. Ve a tu repo de backend en GitHub
2. Haz clic en la pestana **Actions**
3. Deberias ver un workflow "Deploy Backend" corriendo
4. Haz clic para ver los logs

Que debe pasar:

- ☑ Checkout code
- ☑ Set environment variables → `ENV_TAG=staging`
- ☑ Login to GHCR
- ☑ Build and push Docker image → sube a `ghcr.io/tu-usuario/caliente-backend:staging`
- ☑ Deploy to VPS → SSH al server, pull imagen, restart contenedor

10.4 — Push frontend a develop

En tu computadora local, en el repo del **frontend**:

```
cd frontend

git checkout -b develop
git push -u origin develop
```

10.5 — Verificar staging en el navegador

Una vez que ambos pipelines terminen (2-5 minutos cada uno):

```
https://staging.jesuslab135.com        → Deberia mostrar la app Vue
https://staging.jesuslab135.com/api/   → Deberia mostrar JSON o DRF
https://staging.jesuslab135.com/admin/ → Deberia mostrar Django admin login
```

10.6 — Crear el superusuario de Django (primera vez)

```
# En el VPS
ssh root@TU_IP_DEL_VPS

# Crear superusuario para staging
docker compose exec -it backend_staging python3 manage.py createsuperuser
# Te pedira: email, password
```

---

11. PASO 9: Primer Deploy — Production

> Solo haz esto cuando staging funcione correctamente.

11.1 — Merge develop → main (backend)

```
cd backend

git checkout main
git merge develop
git push origin main
```

Esto dispara el pipeline con `ENV_TAG=prod`.

11.2 — Merge develop → main (frontend)

```
cd frontend

git checkout main
git merge develop
git push origin main
```

11.3 — Verificar produccion

```
https://app.jesuslab135.com        → App Vue
https://app.jesuslab135.com/api/   → API
https://app.jesuslab135.com/admin/ → Django admin
```

11.4 — Crear superusuario de produccion

```
# En el VPS
docker compose exec -it backend_prod python3 manage.py createsuperuser
```

---

12. Troubleshooting

"Connection refused" al visitar el sitio

```
# En el VPS, verificar que todos los contenedores estan corriendo
docker compose ps

# Si alguno esta en "Exit" o "Restarting":
docker compose logs <nombre_del_servicio>
# Ejemplo:
docker compose logs backend_staging
docker compose logs nginx
```

"502 Bad Gateway"

Nginx esta corriendo pero no puede conectar al backend/frontend.

```
# Verificar que los backends estan levantados
docker compose ps backend_staging backend_prod

# Ver logs del backend
docker compose logs --tail 50 backend_staging

# Causas comunes:
# 1. La imagen de GHCR aun no existe → push a develop primero
# 2. El backend crasheo al arrancar → revisar logs
# 3. Migraciones fallaron → docker compose exec backend_staging python3 manage.py
migrate
```

"Certificate not valid" o "ERR_CERT_AUTHORITY_INVALID"

```
# Verificar que los certificados existen
docker compose exec nginx ls /etc/letsencrypt/live/
```

```
# Re-generar si es necesario
./init-letsencrypt.sh
```

Pipeline de GitHub Actions falla en "Deploy to VPS"

```
# Verificar que la llave SSH funciona
ssh -i ~/.ssh/caliente_deploy root@TU_IP_DEL_VPS

# Si pide password, la llave no esta configurada:
ssh-copy-id -i ~/.ssh/caliente_deploy.pub root@TU_IP_DEL_VPS

# Verificar que el secret VPS_SSH_KEY tiene la llave PRIVADA completa
# (incluyendo BEGIN y END)
```

"Image not found" en el VPS

```
# El VPS necesita poder descargar imagenes de GHCR
# Opcion 1: Hacer los paquetes publicos en GitHub
# Opcion 2: Login manual en el VPS:
echo "TU_GITHUB_PAT" | docker login ghcr.io -u TU_USUARIO --password-stdin
```

El VPS se queda sin RAM

```
# Verificar uso de memoria
free -h
docker stats --no-stream

# Si se lleno, limpiar imagenes viejas
docker system prune -af

# Verificar que el swap esta activo
swapon --show
```

La base de datos staging no se creo

```
# Conectar al contenedor de postgres
docker compose exec postgres psql -U caliente -d caliente_prod_db

# Listar bases de datos
\l

# Si no existe caliente_staging_db, crearla manualmente:
CREATE DATABASE caliente_staging_db;
```

```
GRANT ALL PRIVILEGES ON DATABASE caliente_staging_db TO caliente;
\q
```

---

13. Mantenimiento Continuo

Flujo de trabajo diario

```
1. Desarrollas en tu computadora local
2. Push a 'develop' → staging se actualiza automaticamente
3. Pruebas en staging.jesuslab135.com
4. Cuando este listo: merge develop → main
5. Production se actualiza automaticamente en app.jesuslab135.com
```

Renovacion de SSL

Los certificados se renuevan **automaticamente** cada 12 horas via el contenedor `certbot`. No necesitas hacer nada.

Para verificar manualmente:

```
# Ver cuando expiran los certificados
docker compose exec certbot certbot certificates
```

Ver logs de los servicios

```
# Todos los servicios
docker compose logs --tail 100

# Un servicio especifico
docker compose logs --tail 50 -f backend_prod

# Solo errores de nginx
docker compose logs nginx | grep error
```

Reiniciar un servicio

```
docker compose restart backend_staging
docker compose restart frontend_prod
docker compose restart nginx
```

Limpiar espacio en disco

```
# Eliminar imagenes Docker que ya no se usan
docker image prune -af

# Ver cuanto espacio usa Docker
docker system df
```

Backup de la base de datos

```
# Backup de produccion
docker compose exec postgres pg_dump -U caliente caliente_prod_db >
backup_prod_$(date +%Y%m%d).sql

# Backup de staging
docker compose exec postgres pg_dump -U caliente caliente_staging_db >
backup_staging_$(date +%Y%m%d).sql
```

Ejecutar migraciones manualmente

```
# Staging
docker compose exec backend_staging python3 manage.py migrate

# Production
docker compose exec backend_prod python3 manage.py migrate
```

Abrir Django shell

```
docker compose exec backend_prod python3 manage.py shell
```

Checklist Final

Usa esta lista para verificar que todo esta en orden:

- [ ] DNS: `staging.jesuslab135.com` resuelve a la IP del VPS
- [ ] DNS: `app.jesuslab135.com` resuelve a la IP del VPS
- [ ] VPS: Docker y Docker Compose instalados
- [ ] VPS: UFW permite puertos 22, 80, 443
- [ ] VPS: Swap de 2 GB activo
- [ ] VPS: Directorio `/opt/caliente/` con todos los archivos
- [ ] VPS: `.env`, `.env.staging`, `.env.prod` con passwords reales
- [ ] VPS: PostgreSQL corriendo con ambas bases de datos
- [ ] VPS: Certificados SSL generados para ambos subdominios
- [ ] GitHub: Secrets configurados en el repo del backend

- [ ] GitHub: Secrets configurados en el repo del frontend
- [ ] GitHub: Paquetes GHCR visibles (publicos o con PAT)
- [ ] Deploy: Push a `develop` despliega en staging correctamente
- [ ] Deploy: Push a `main` despliega en produccion correctamente
- [ ] App: `https://staging.jesuslab135.com` carga la Vue app
- [ ] App: `https://app.jesuslab135.com` carga la Vue app
- [ ] App: `/api/` responde en ambos entornos
- [ ] App: Superusuario creado en ambos entornos