Preparar el puesto de trabajo para la nube

Entorno local seguro, reproducible y verificable para las prácticas del curso

Josep Garcia

Introducción a la nube pública · 2026/2027

Curso 2026/2027 - Actualizado: 08/10/2026

Índice general

Cómo trabajar en P00 . . . 4
Objetivos . . . 4
Antes de empezar: qué es Docker y por qué lo usamos . . . 5
    Tutorial Docker (David Martínez) . . . 5
Docker vs Virtualbox . . . 7
Actividad 1 · Información docker . . . 8
Actividad 2 · Un contenedor sin nombre y otro con nombre . . . 8
    Primero: deja que Docker elija el nombre . . . 8
    Después: asigna un nombre útil . . . 9
Actividad 3 · Una aplicación que sigue activa . . . 9
Actividad 4 · Una terminal Debian temporal . . . 10
Actividad 5 · Crear el entorno persistente del curso . . . 11
    Linux . . . 11
    Windows PowerShell . . . 11
    Comprobar que los archivos sobreviven en ambos sentidos . . . 13
    Comprobar el archivo desde el anfitrión . . . 13
    Volver a entrar y comprobar el archivo del anfitrión . . . 14
    Cerrar Docker Desktop y recuperar el entorno . . . 14
    Crear el usuario de trabajo . . . 15
Cierre individual . . . 16
Fuentes para ampliar . . . 16
Qué es la shell y por qué la usamos aquí . . . 17
    Entrar y orientarse . . . 17
    Leer una salida antes de continuar . . . 18
Crear un sistema de ficheros y directorios . . . 18
Rutas absolutas y rutas relativas . . . 20
Leer, escribir y buscar texto . . . 20
    Referencia de los comandos practicados . . . 21
Veinte ejercicios de comandos Linux . . . 22
    Cinco ejercicios de mayor dificultad . . . 23
    Cinco ejercicios con un comando nuevo . . . 23
Cuatro scripts básicos . . . 26
    1. Mostrar un mensaje . . . 26
    2. Mostrar el entorno actual . . . 27
    3. Listar el proyecto . . . 27
    4. Buscar texto en un fichero conocido . . . 27

De órdenes repetidas a un script pequeño . . . 28
Crear el script de inventario . . . 28
Probar antes de ejecutar . . . 30
Variar el script con un argumento controlado . . . 30
Refuerzo: repetición con un bucle pequeño . . . 31
Cierre de sesión y conexión con lo que sigue . . . 31
Fuente para ampliar . . . 31
Antes de empezar: retoma tu puesto de trabajo . . . 32
Git: decidir qué historia conserva el proyecto . . . 33
Las tres zonas de Git . . . 33
Crear el repositorio local . . . 33
Proteger antes de guardar: `.gitignore` . . . 34
Primera fotografía: estructura y protección . . . 35
Segunda fotografía: una página y una bitácora . . . 36
Corregir antes de confirmar . . . 37
Retos de comprobación de Git . . . 38
Compartir solo cuando exista un destino aprobado . . . 38
SSH: un canal cifrado que exige verificar la identidad . . . 39
Instalar el cliente y preparar la carpeta privada . . . 40
Crear una pareja de claves para una práctica autorizada . . . 40
Primera conexión: verificar primero el host . . . 41
Retos de comprobación de SSH . . . 42
A entregar . . . 42
Ideas que reaparecerán en las prácticas de nube . . . 42
Fuentes para ampliar y comprobar . . . 42
Diagnosticar no es probar órdenes al azar . . . 43
Preparar el contexto y las herramientas . . . 44
El caso de prueba: una web local publicada en el puerto 8080 . . . 45
Ejemplo resuelto: el puerto no coincide . . . 46
Comprobar por capas . . . 46
1. Contexto y dirección . . . 46
2. Puerto TCP . . . 46
3. Respuesta HTTP y cabeceras . . . 47
4. Procesos, puertos y registros . . . 48
Resolver tres incidencias guiadas . . . 48
Incidencia A: el navegador muestra «No se puede conectar» . . . 48
Incidencia B: `curl` devuelve `404 Not Found` . . . 49
Incidencia C: desde Debian no se alcanza el anfitrión . . . 49
Registrar un producto que sirva para aprender . . . 49
Fuentes para ampliar . . . 50
Del archivo HTML a una aplicación reproducible . . . 50
Un cambio, tres lugares distintos . . . 50
1. Preparar una web pequeña pero identificable . . . 51
2. Convertir el contenido en una imagen . . . 52

3. Ejecutar, comprobar y cerrar un contenedor . . . 53
4. Por qué Compose mejora la repetición . . . 55
Red interna: comunicación no es lo mismo que publicar un puerto . . . 55
Crear `compose.yaml` . . . 56
5. Ciclo de vida guiado con Compose . . . 57
6. De una incidencia a una corrección razonada . . . 58
7. A entregar y cierre responsable . . . 59
Fuentes para ampliar . . . 60
Protocolo de AWS Academy Learner Lab . . . 60
Plantilla de bitácora para cualquier práctica . . . 61
Ejercicios cortos de refuerzo . . . 61
RF00. Ciclo de vida de `p00-lab` . . . 61
RF01. Bitácora y script de Shell . . . 62
RF02. Detective de secretos . . . 62
RF03. Diagnóstico guiado . . . 62
RF04. SSH seguro . . . 63
RF05. Plan de trabajo responsable en Learner Lab . . . 63
Ejemplo guiado / prácticas . . . 64
EG00. Entorno Debian reproducible . . . 64
EG01. Repositorio seguro con web estática y productos de práctica . . . 64
EG02. Web estática en Docker . . . 65
EG03. Web estática en Docker Compose . . . 65
EG04. Evolución reproducible de `p00-lab` . . . 65
Ejercicios . . . 66
EA01. Incidencia local . . . 66
EA02. Auditoría de seguridad . . . 66
Proyecto o reto final . . . 67
RFN01. Puesto listo . . . 67
RFN02. Entrega reproducible . . . 67
Checklist del portafolio final . . . 67
Fuentes de consulta . . . 68

Antes de desplegar una carga en la nube hay que poder construirla, comprobarla y documentarla en un entorno controlado. En este proyecto prepararás un puesto de trabajo reproducible a partir de Docker: un contenedor Debian común, un repositorio seguro, una página web mínima y las herramientas necesarias para ejecutarla y verificarla localmente.

El trabajo se podrá realizar en Windows o Linux. El sistema anfitrión se limita a Docker, el navegador y la carpeta compartida `optativa-nube/`; el trabajo técnico se realiza dentro de un contenedor Debian persistente por estudiante. Si Docker no fuera viable en un equipo, una VM Debian podrá actuar como alternativa de contingencia. En todos los casos, la evidencia debe explicar qué entorno se utilizó y cómo se validó.

El hilo conductor es un repositorio de práctica que irá creciendo durante las doce sesiones. Primero crearás y administrarás el contenedor de trabajo; después usarás su shell, protegerás

los datos con Git y, más adelante, ejecutarás la web en un contenedor específico. Ese mismo hábito de comprobar, registrar y limpiar recursos continuará en los proyectos de AWS.

Cómo trabajar en P00

Cada página sigue la misma secuencia. No se trata de copiar órdenes hasta que «parezcan funcionar», sino de entender qué cambia en cada paso.

1. **Sitúate:** identifica si estás en el anfitrión o dentro de `p00-lab`, y comprueba tu ruta actual.
2. **Observa:** ejecuta primero la comprobación que responde a una pregunta concreta.
3. **Cambia una cosa:** crea, edita o ejecuta solo el recurso indicado.
4. **Verifica:** interpreta una salida, una URL o un archivo; no avances solo porque la orden no haya mostrado error.
5. **Registra y cierra:** conserva la evidencia en `optativa-nube/` y detén los recursos temporales.

Por ejemplo, si una web no aparece en el navegador, no reconstruyas la imagen inmediatamente: primero pregunta si el contenedor sigue activo, si el puerto está publicado y qué responde HTTP. Esta forma de trabajar será la misma cuando los recursos estén en AWS y tengan coste.



Los bloques marcados como «desde el anfitrión» se ejecutan en Windows o Linux. Los demás se ejecutan dentro de `p00-lab`. Si dudas, detente y revisa el encabezado del bloque antes de pegar una orden.

Objetivos

Al finalizar el proyecto podrás:

- crear, iniciar, detener, inspeccionar y limpiar contenedores Docker, diferenciándolos de las imágenes y de las máquinas virtuales;
- operar sobre un espacio de trabajo compartido mediante comandos básicos de Bash dentro de Debian, incluidos permisos, procesos, redirecciones, tuberías y una automatización sencilla;
- versionar un repositorio y documentar evidencias sin incorporar secretos, claves privadas ni credenciales;
- generar y proteger un par de claves SSH, verificando la identidad del servidor antes del primer acceso;
- instalar con apt herramientas necesarias para una práctica y mantener en un Dockerfile las que sean requisitos recurrentes;
- publicar y comprobar localmente una página web estática mínima;
- diagnosticar incidencias elementales de HTTP, DNS, puertos o ejecución mediante comprobaciones autorizadas;

- aplicar un protocolo de inventario, coste, cierre y limpieza para trabajar de forma responsable en AWS Academy Learner Lab.

En este bloque comprobarás tres formas de ejecutar un contenedor: un programa que termina, una aplicación web que permanece activa y una terminal interactiva. Después crearás `p00-lab`, tu contenedor de trabajo persistente. Ese será el punto de partida del resto de P00.

Las órdenes de Docker se ejecutan siempre en la **terminal del anfitrión** (Windows o Linux). Solo entrarás en una terminal Debian cuando se indique expresamente. Dentro de Debian no ejecutes órdenes que empiecen por `docker`.



No copies una orden sin saber desde qué terminal la ejecutas. Docker pertenece al anfitrión; la shell, Git, SSH y el diagnóstico del curso se harán después dentro de `p00-lab`.

Antes de empezar: qué es Docker y por qué lo usamos

Docker es una plataforma para **crear, distribuir y ejecutar aplicaciones en contenedores**. Un contenedor reúne una aplicación y el entorno mínimo que necesita para funcionar: por ejemplo, Debian y las herramientas que utilizaremos durante el curso.

Esto evita el problema de «en mi ordenador funciona»: si cada equipo tiene versiones, configuraciones o sistemas distintos, una misma práctica puede dar resultados diferentes. En P00 usaremos Docker para que todas las personas dispongáis del mismo entorno Debian, aunque el anfitrión sea Windows o Linux. Así podremos centrarnos en la shell, Git, SSH y las comprobaciones de red, sin reinstalar esas herramientas en cada equipo.

Docker no sustituye a tu sistema operativo ni es una máquina virtual completa. El contenedor utiliza el motor Docker del anfitrión y, en Linux, comparte su kernel; por eso es más ligero y arranca más rápido que una máquina virtual. En este curso mantendrás un único contenedor, `p00-lab`, y tu trabajo quedará en la carpeta compartida `optativa-nube/`.

Tutorial Docker (David Martínez)

https://martinezpenya.es/ModelosIA/UD00/UD00_ES.html ### Componentes

Componente	Qué es	Papel en P00
Cliente Docker	El comando `docker` que escribes en la terminal del anfitrión.	Envía órdenes como crear, iniciar, detener e inventariar contenedores.
Motor Docker	El servicio que recibe esas órdenes y gestiona los recursos. En Windows suele ejecutarlo Docker Desktop.	Crea y ejecuta `p00-lab`; por eso `docker version` debe mostrar *Client* y *Server*.
Imagen	Una plantilla de solo lectura con un sistema base y las herramientas preparadas.	Usarás la imagen oficial `debian:trixie-slim` como punto de partida del contenedor de trabajo.
Contenedor	Una instancia concreta creada desde una imagen, con su propio estado de ejecución.	`p00-hola`, `p00-excalidraw` y `p00-lab` son contenedores distintos.
Registro	Un servicio que almacena y distribuye imágenes, como Docker Hub.	Docker descargará una imagen si aún no existe en tu equipo; solo usarás las imágenes indicadas en la práctica.
Montaje	Una conexión entre una carpeta o volumen y el contenedor.	`optativa-nube/` se monta en `/optativa-nube` para que repositorios y evidencias persistan al detener el contenedor.

Docker vs Virtualbox

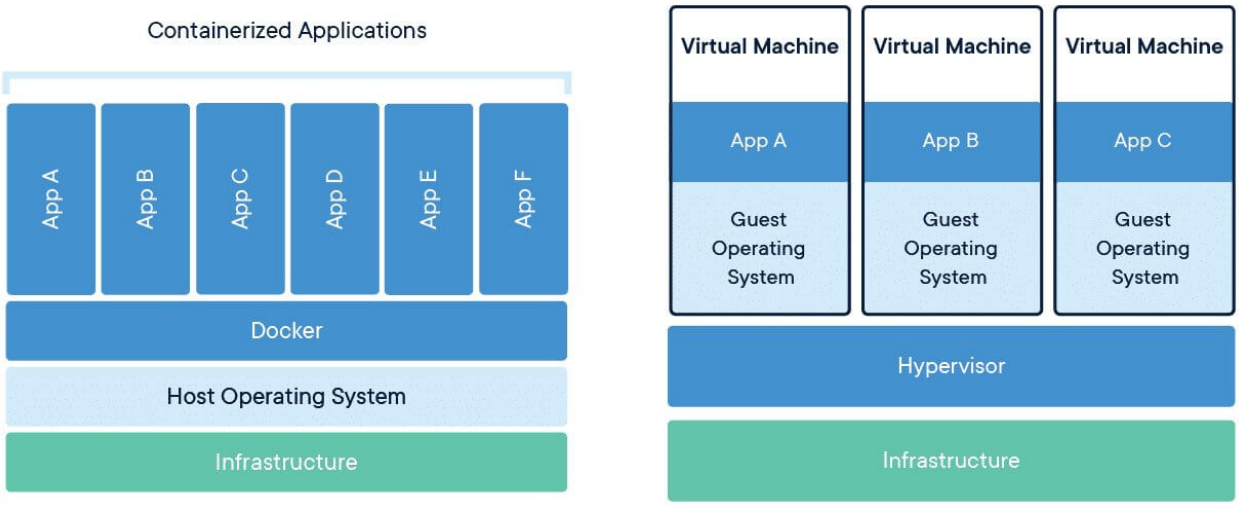


Imagen vs Contenedor

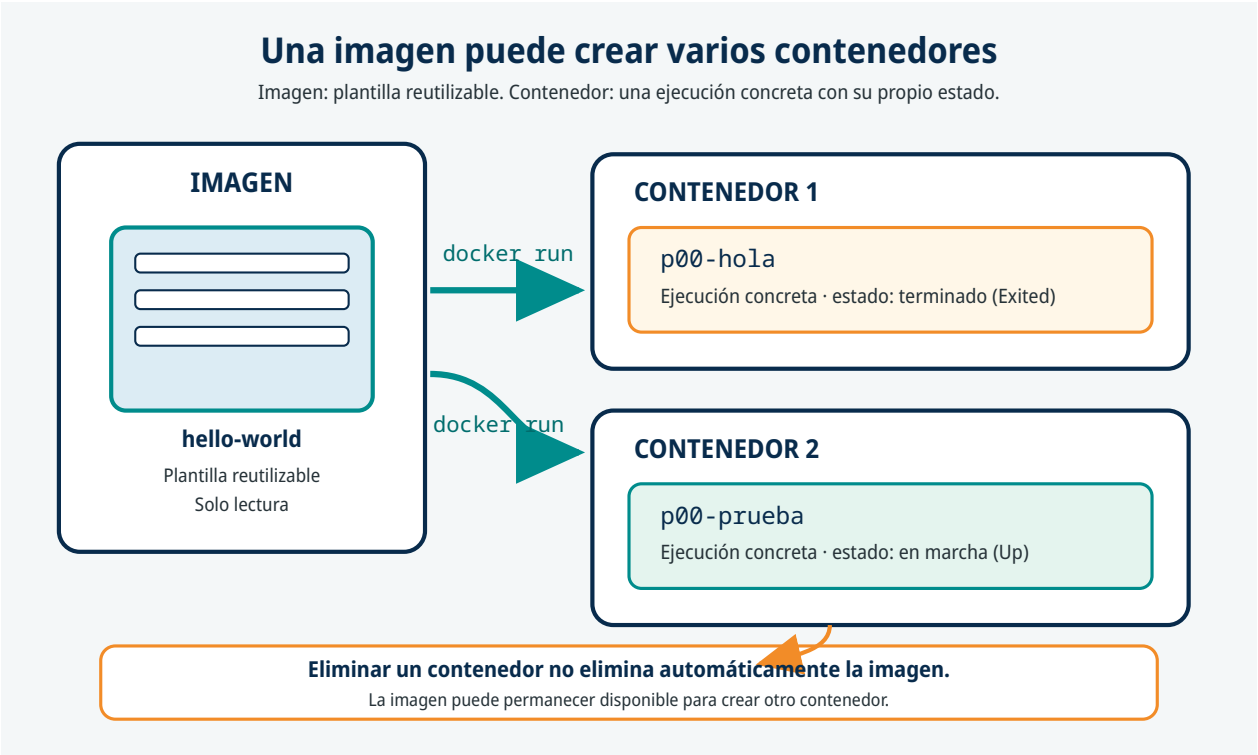


Figura 1: Una imagen reutilizable de solo lectura crea mediante docker run dos contenedores concretos con estados propios; eliminar un contenedor no elimina automáticamente la imagen.

Actividad 1 · Información docker

Entorno: terminal del anfitrión, con Docker Desktop abierto si usas Windows.

Primero comprueba que Docker puede comunicarse con su motor:

```
docker version
```

Debe aparecer información de **Client** y **Server**. Si falta el servidor o aparece un error de conexión, registra el mensaje en una frase y sigue la demostración indicada por el docente. No instales componentes ni uses sudo durante esta práctica.

Ahora consulta dos inventarios:

```
docker ps
docker ps -a
```

`docker ps` muestra solo los contenedores en marcha. `docker ps -a` muestra también los detenidos. Durante el curso usarás normalmente estas dos órdenes completas para comprender qué recursos hay en tu equipo.

Actividad 2 · Un contenedor sin nombre y otro con nombre

La imagen `hello-world` ejecuta un programa pequeño y termina. Es una forma segura de observar un contenedor detenido.

Primero: deja que Docker elija el nombre

Predicción: después de ejecutar la orden, ¿qué aparecerá en `docker ps`? ¿Y en `docker ps -a`?

```
docker run hello-world
docker ps
docker ps -a
```

En la columna `NAMES` de `docker ps -a`, Docker habrá asignado un nombre automático. Anótalo en tu tabla. El estado será equivalente a `Exited`: el programa terminó, pero el contenedor sigue inventariado.

Elimina únicamente ese contenedor, sustituyendo el texto por el nombre que has observado tú:

```
docker rm NOMBRE_ASIGNADO_POR_DOCKER
docker ps -a
```

Después: asigna un nombre útil

Hasta ahora Docker eligió el nombre del contenedor automáticamente. Cuando crees un contenedor que vas a revisar o reutilizar, asígnale un nombre claro con `--name`.

```
docker run --name p00-hola hello-world
```

Docker crea un contenedor llamado `p00-hola`, muestra el mensaje de prueba y lo detiene porque el programa `hello-world` ya ha terminado.

Ahora consulta los dos inventarios:

```
docker ps
docker ps -a
```

Comprobación: `p00-hola` debe aparecer en `docker ps -a` con un estado parecido a `Exited`, pero no en `docker ps`.

¿Por qué aparece en un inventario y no en el otro?

Actividad 3 · Una aplicación que sigue activa

Ahora probarás Excalidraw, una aplicación de dibujo que se abrirá solo en tu propio equipo. A diferencia de `hello-world`, su proceso sigue activo mientras la aplicación está disponible.

Punto de partida: `p00-hola` está detenido y el puerto local `8080` está libre. Comprueba los contenedores antes de crear otro:

```
docker ps
docker ps -a
```

Cambio: crea `p00-excalidraw` con la imagen pública de Excalidraw.

```
docker run -d --rm --name p00-excalidraw -p 127.0.0.1:8080:80
↪  excalidraw/excalidraw:latest
```

- `-d` deja la aplicación ejecutándose sin ocupar tu terminal.

- `-p 127.0.0.1:8080:80` conecta el puerto 8080 de tu propio equipo con el puerto 80 del contenedor. La dirección `127.0.0.1` impide que la aplicación se publique en la red del aula.
- `--rm` elimina este contenedor temporal cuando se detenga; la imagen descargada puede permanecer.

Comprobación: abre http://127.0.0.1:8080 en el navegador, crea una figura simple y consulta el estado:

```
docker ps
docker ps -a
```

Debes ver `p00-excalidraw` en marcha. Ciérralo de manera deliberada:

```
docker stop p00-excalidraw
docker ps
docker ps -a
```

Como se creó con `--rm`, ya no debe aparecer en `docker ps -a`. Anota en la tabla qué permanece y qué no permanece tras detenerlo.

Actividad 4 · Una terminal Debian temporal

Un contenedor también puede ejecutar una terminal. Este Debian no será aún tu espacio de trabajo del curso: servirá para comprobar qué ocurre al salir de un proceso interactivo.

```
docker run -it --name p00-debian-temporal debian:trixie-slim bash
```

- `-i` mantiene abierta la entrada estándar para que puedas escribir.
- `-t` crea una terminal.
- `-it` reúne las dos opciones para que puedas trabajar en Bash dentro del contenedor.
- `debian:trixie-slim` es la imagen oficial ligera de Debian 13 que servirá de base para el contenedor persistente del curso.

Entorno: ya estás dentro de Debian. Antes de ejecutar las órdenes, predice dónde quedará `mensaje.txt` después de salir.

```
pwd
whoami
printf '%s\n' 'Hola desde Debian temporal.' > mensaje.txt
cat mensaje.txt
exit
```

De vuelta en el anfitrión, consulta el inventario completo:

```
docker ps
docker ps -a
```

`p00-debian-temporal` debe aparecer detenido. El fichero sigue dentro de ese contenedor mientras exista, pero no está en una carpeta compartida ni será parte del trabajo del curso. Elimínalo ahora junto con `p00-hola`, después de verificar ambos nombres:

```
docker rm p00-hola p00-debian-temporal
docker ps -a
```

Actividad 5 · Crear el entorno persistente del curso

Hasta ahora los contenedores eran temporales. Ahora crearás el único contenedor que reutilizarás durante P00: `p00-lab`.

`p00-lab` será un contenedor Debian 13 creado directamente desde la imagen oficial `debian:trixie-slim`. Empezarás ahora a instalar dentro de él las herramientas necesarias y registrarás los cambios durante el curso. Por ahora no crearás una imagen propia ni usarás un Dockerfile.

```
debian:trixie-slim → contenedor persistente p00-lab → herramientas instaladas durante
↪ el curso
```

Trabajarás en una carpeta llamada `optativa-nube` dentro de tu carpeta personal. Si prefieres tenerla visible en el Escritorio, crea allí la misma carpeta y usa la orden alternativa. Esa carpeta guarda repositorios, evidencias y la web; Docker la mostrará dentro del contenedor como `/optativa-nube`.

Linux

```
mkdir -p "$HOME/optativa-nube"
docker run -it --name p00-lab -v "$HOME/optativa-nube:/optativa-nube"
↪ debian:trixie-slim bash
```

Windows PowerShell

```
New-Item -ItemType Directory -Force "$HOME\optativa-nube"
docker run -it --name p00-lab -v "${HOME}\optativa-nube:/optativa-nube"
↪  debian:trixie-slim bash
```

Para usar el Escritorio, cambia solo `optativa-nube` por `Desktop/optativa-nube` en Linux o por `Desktop\optativa-nube` en PowerShell.

- `-i` mantiene abierta la entrada estándar para que puedas escribir.
- `-t` crea una terminal.
- `-it` reúne las dos opciones para que puedas trabajar en Bash dentro de `p00-lab`.
- `--name p00-lab` asigna el nombre que reutilizarás durante el curso.
- `-v` conecta tu carpeta `optativa-nube/` con `/optativa-nube`.
- `debian:trixie-slim` identifica la imagen oficial ligera de Debian 13 desde la que se crea este contenedor.

La imagen oficial abre la primera terminal como `root`, el usuario administrador del contenedor. Es normal en una imagen Debian mínima: no es el sistema anfitrión y sus privilegios se limitan a `p00-lab`. Compruébalo con `whoami`. En este bloque instalarás las primeras herramientas de trabajo; más adelante crearás un usuario de trabajo sin privilegios para el uso cotidiano. No uses esta cuenta para crear archivos fuera de `/optativa-nube` ni copies contraseñas, claves o credenciales al contenedor.

Cambio dentro de `p00-lab`: actualiza el índice de paquetes e instala las herramientas que utilizarás al inicio del curso:

```
apt-get update
apt-get install --yes --no-install-recommends git ca-certificates procps sudo
git --version
ps
```

- `apt-get update` actualiza la lista de paquetes disponibles.
- El indicador `--yes` confirma la instalación.
- `--no-install-recommends` evita instalar complementos no necesarios.
- `git` permite guardar una historia local de los cambios del proyecto y revisar qué archivos vas a entregar.
- `ca-certificates` instala las autoridades de certificación que permiten a programas como Git comprobar de forma segura conexiones HTTPS.
- `procps` incorpora, entre otras utilidades, `ps`, que usarás para consultar los procesos en ejecución.

- sudo permite que un usuario sin privilegios ejecute puntualmente una orden administrativa autorizada. En esta primera sesión todavía eres `root`, pero será necesario cuando se cree el usuario de trabajo.

La carpeta local `optativa-nube` se monta directamente como `/optativa-nube`, una carpeta situada bajo la raíz del contenedor. En Linux, sus permisos son los de la carpeta creada en el anfitrión. En Windows, Docker Desktop gestiona ese acceso a las carpetas compartidas. La actividad siguiente comprueba el montaje en los dos sentidos. Si aparece `Permission denied`, no uses sudo: guarda el mensaje y revisa el montaje antes de continuar.

Comprobar que los archivos sobreviven en ambos sentidos

Primero estás dentro de `p00-lab`. Crea un archivo en la ruta montada, léelo y sal del contenedor:

```
mkdir -p /optativa-nube/p00-puesto-nube/evidencias
printf '%s\n' 'Creado dentro de p00-lab.' >
 ↪  /optativa-nube/p00-puesto-nube/evidencias/desde-contenedor.txt
cat /optativa-nube/p00-puesto-nube/evidencias/desde-contenedor.txt
exit
```

`exit` detiene `p00-lab`, pero no lo elimina. Ahora estás de nuevo en el **anfitrión**. El archivo debe estar en la carpeta local que montaste al crear el contenedor.

Comprobar el archivo desde el anfitrión

Usa el bloque correspondiente a tu sistema operativo:

```
cat "$HOME/optativa-nube/p00-puesto-nube/evidencias/desde-contenedor.txt"
printf '%s\n' 'Creado desde el anfitrión.' >
 ↪  "$HOME/optativa-nube/p00-puesto-nube/evidencias/desde-anfitrion.txt"
cat "$HOME/optativa-nube/p00-puesto-nube/evidencias/desde-anfitrion.txt"
```

Linux

```
Get-Content "$HOME\optativa-nube\p00-puesto-nube\evidencias\desde-contenedor.txt"
Set-Content -Path "$HOME\optativa-nube\p00-puesto-nube\evidencias\desde-anfitrion.txt"
 ↪  -Value "Creado desde el anfitrión."
Get-Content "$HOME\optativa-nube\p00-puesto-nube\evidencias\desde-anfitrion.txt"
```

Windows PowerShell El primer comando de cada bloque demuestra que el archivo creado dentro de Debian está realmente en tu equipo. El segundo crea otro archivo desde el anfitrión y el tercero lo comprueba antes de continuar.

Volver a entrar y comprobar el archivo del anfitrión

`p00-lab` debe seguir detenido, no eliminado. Inicia exactamente el mismo contenedor y comprueba los dos archivos desde dentro:

```
docker ps -a
docker start -ai p00-lab
cat /optativa-nube/p00-puesto-nube/evidencias/desde-contenedor.txt
cat /optativa-nube/p00-puesto-nube/evidencias/desde-anfitrion.txt
exit
```

- `-a` conecta la salida estándar y los mensajes de error del contenedor con tu terminal. Así puedes ver el *prompt*, los textos de `cat` y los mensajes que produzca Bash.
- `-i` mantiene abierta la entrada estándar para que puedas escribir.
- `-ai` reúne ambas opciones: vuelves a entrar en el mismo `p00-lab` sin crear otro contenedor.

Por ejemplo, si ejecutas desde el anfitrión `docker start p00-lab`, Docker inicia el contenedor y devuelve enseguida el nombre `p00-lab`, pero no ves su *prompt* ni puedes trabajar dentro. En cambio, `docker start -ai p00-lab` conecta tu terminal con Bash: ves algo parecido a `root@…:/#`, puedes ejecutar `cat` y lees su resultado en la misma ventana. `-a` permite ver; `-i` permite escribir.

Si aparecen ambos textos, has comprobado el montaje en ambos sentidos: un cambio realizado dentro del contenedor se ve en tu equipo y un cambio realizado en tu equipo se ve dentro del contenedor. A partir del siguiente bloque entrarás con `docker start -ai p00-lab`; no vuelvas a ejecutar `docker run` para crear otro contenedor con ese nombre.

Cerrar Docker Desktop y recuperar el entorno

Tras el último `exit`, comprueba desde el anfitrión que no queda ningún contenedor en marcha:

```
docker ps
docker ps -a
```

`docker ps` no debe mostrar contenedores activos y `docker ps -a` debe mostrar `p00-lab` con estado `Exited`. Cierra Docker Desktop mediante **Quit Docker Desktop** o **Salir de Docker Desktop**. Al cerrarlo, el motor se apaga y cualquier contenedor que siguiera activo se detendría.

En la siguiente sesión, abre Docker Desktop y espera a que `docker version` muestre información tanto de **Client** como de **Server**. Después recupera el mismo entorno:

```
docker start -ai p00-lab
cat /optativa-nube/p00-puesto-nube/evidencias/desde-contenedor.txt
cat /optativa-nube/p00-puesto-nube/evidencias/desde-anfitrion.txt
```

Los dos archivos permanecen porque están en tu carpeta local `optativa-nube`, no dentro del sistema de archivos interno de Docker. El contenedor `p00-lab` también seguirá disponible mientras no desinstales, restablezcas o elimines los datos de Docker Desktop.

Crear el usuario de trabajo

Sigues dentro de `p00-lab` como `root`. Crea ahora el usuario ordinario `alumno`, que utilizarás desde el bloque siguiente. Los identificadores `1000` permiten que, en un anfitrión Linux habitual, `alumno` pueda trabajar con los archivos de la carpeta compartida creada por tu usuario:

```
groupadd --gid 1000 alumno
useradd --uid 1000 --gid 1000 --create-home --shell /bin/bash alumno
printf '%s\n' 'alumno ALL=(ALL) NOPASSWD: ALL' > /etc/sudoers.d/alumno
chmod 440 /etc/sudoers.d/alumno
# Solo si tu anfitrión es Linux:
chown -R alumno:alumno /optativa-nube/p00-puesto-nube
su - alumno
whoami
pwd
exit
exit
```

`groupadd` crea el grupo y `useradd` crea la cuenta con su carpeta personal `/home/alumno`. El archivo de `sudoers` permite que `alumno` use `sudo` solo dentro de su propio contenedor y sin una contraseña que no existe. Si tu anfitrión es Linux, `chown` deja la carpeta compartida a nombre de `alumno`; no ejecutes esa línea si trabajas con Windows. `su - alumno` abre una sesión como ese usuario; `whoami` debe mostrar `alumno` y `pwd`, `/home/alumno`. El primer `exit` vuelve a `root`; el segundo detiene `p00-lab`.

Como el contenedor se creó inicialmente desde la imagen oficial, `docker start -ai p00-lab` abrirá primero Bash como `root`. A partir de P00/02 ejecutarás después `su - alumno` para pasar a tu usuario de trabajo.



No uses `--rm` con `p00-lab`, no ejecutes `docker commit` y no montes el socket Docker dentro del contenedor. Cuando acabes, sal con `exit`: el contenedor se detiene y tus archivos de `optativa-nube/` permanecen.

Cierre individual

Completa la tabla de observación y responde en dos o tres frases. En **Dónde lo has visto**, escribe la orden con la que localizaste el recurso (por ejemplo, `docker ps` o `docker ps -a`). En **Estado**, anota si estaba en marcha, detenido o ya eliminado. En **Dato que lo demuestra**, copia un dato concreto de la salida: el nombre, `Up`, `Exited` o que no aparecía en el inventario.

Recurso observado	Dónde lo has visto (orden)	Estado observado	Dato concreto que lo demuestra
Contenedor sin nombre			
`p00-hola`			
`p00-excalidraw`			
`p00-debian-temporal`			
`p00-lab`			

1. ¿Qué diferencia observaste entre `p00-hola` y `p00-excalidraw`?
2. ¿Por qué el fichero de Debian temporal no era todavía una evidencia del curso?
3. ¿Qué dos archivos demuestran que el montaje funciona en ambos sentidos?
4. ¿Qué estado debe mostrar `p00-lab` antes de cerrar Docker Desktop y qué debes comprobar al volver a abrirlo?

Fuentes para ampliar

- Tutorial Docker de David Martínez: conceptos y uso básico
- Docker Docs: conceptos básicos
- Docker Docs: publicar puertos
- Docker Docs: `docker run`
- Docker Hub: imagen oficial de Debian
- Docker Hub: imagen de Excalidraw

Qué es la shell y por qué la usamos aquí

La **terminal** es la ventana en la que escribes; la **shell** es el programa que interpreta las órdenes que escribes en esa ventana. En este curso la shell será Bash dentro del contenedor Debian. Por eso las prácticas serán iguales para quien use Windows y para quien use Linux en el equipo anfitrión.

Una orden suele tener esta forma:

```
comando [opciones] [argumentos]
```

Por ejemplo, en `ls -la /optativa-nube`, `ls` es el comando, `-la` son opciones y `/optativa-nube` es el argumento que indica qué carpeta se quiere listar. No memorices órdenes aisladas: identifica primero qué quieres averiguar o cambiar y comprueba el resultado después.

El *prompt* termina normalmente en `$` cuando trabaja un usuario ordinario y en `#` cuando se usa una cuenta administrativa. En `p00-lab` trabajarás como `alumno`, así que verás `$`. Cuando una práctica necesite instalar una herramienta, la orden comenzará por `sudo`; esto eleva permisos solo para esa orden dentro del contenedor. No uses `sudo` en el anfitrión ni ejecutes instrucciones que no entiendas.

Entrar y orientarse

Todos los comandos de esta página se ejecutan dentro de `p00-lab`. Desde la terminal del anfitrión, inicia una sesión:

```
docker start -ai p00-lab
su - alumno
```

`docker start` reutiliza la configuración con la que creaste `p00-lab`, por lo que abre primero Bash como `root`. `su - alumno` inicia entonces una sesión del usuario de trabajo creado en P00/01. No intentes cambiarlo añadiendo una opción a `docker start`: esa orden no permite modificar el usuario del contenedor. `-a` conecta la salida a tu terminal e `-i` mantiene abierta la entrada para que puedas escribir.

Punto de partida: ya estás en Bash dentro del contenedor. Averigua dónde estás, qué usuario ejecuta la sesión y qué shell se ha iniciado:

```
pwd
whoami
ps
```

- `pwd` muestra la carpeta de trabajo actual.
- `whoami` muestra la cuenta efectiva.

- `ps` muestra los procesos asociados a tu terminal. Debes reconocer al menos Bash.

Si un resultado no coincide con lo esperado, no sigas con comandos que modifiquen archivos. Comprueba primero si has entrado en el contenedor correcto y vuelve a ejecutar `pwd`.

Leer una salida antes de continuar

No necesitas copiar toda la salida de una orden. Para este primer diagnóstico, anota solo una conclusión por comando:

Orden	Ejemplo de conclusión útil
`pwd`	«Estoy dentro de `/home/alumno`; antes de crear evidencias entraré en `/optativa-nube`».
`whoami`	«La shell usa la cuenta `alumno`; no estoy administrando el anfitrión».
`ps`	«Bash es un proceso activo de mi terminal».

Si `whoami` no muestra `alumno` o `pwd` no muestra `/home/alumno`, no sigas con comandos que modifiquen archivos: vuelve a comprobar que ejecutaste `su - alumno`. En el siguiente paso entrarás deliberadamente en `/optativa-nube`.



El lugar de trabajo del curso es `/optativa-nube`. Repositorios, páginas web, scripts y evidencias se crean ahí porque es la carpeta compartida con el anfitrión. Lo que guardes en otras rutas del contenedor podría perderse al recrearlo.

Crear un sistema de ficheros y directorios

Construirás el espacio que usarás durante P00. Primero entra en la carpeta compartida y lista su contenido más simple:

```
cd /optativa-nube
pwd
ls
```

`ls` muestra los nombres visibles. Cuando necesites más información, usa estas opciones:

Orden	Qué añade
`ls -l`	Permisos, propietario, tamaño y fecha.
`ls -a`	También muestra nombres que empiezan por `..`
`ls -la`	Combina ambas opciones.
`ls -lh`	Muestra los tamaños en un formato más fácil de leer.

Ahora crea la carpeta principal y entra en ella. Ejecuta una orden, observa el resultado y continúa con la siguiente:

```
mkdir p00-puesto-nube
ls
cd p00-puesto-nube
pwd
```

Crea el primer nivel del sistema de ficheros. Aquí todavía no necesitas `-p`, porque cada carpeta se crea directamente dentro de la carpeta actual:

```
mkdir evidencias
mkdir notas
mkdir scripts
mkdir web
ls
```

Después crea algunos ficheros vacíos y comprueba el árbol que has construido:

```
touch README.md
touch notas/ideas.txt
touch web/index.html
ls
ls -la notas
```

`touch` crea un fichero vacío si no existe. Si ya existe, no borra su contenido: solo actualiza su fecha de modificación. Aquí lo usas para preparar los archivos que rellenarás después.

Debes tener esta estructura:

```
/optativa-nube/p00-puesto-nube/
├── evidencias/
├── notas/
│   └── ideas.txt
├── scripts/
```

```
├── web/
│   └── index.html
└── README.md
```

Rutas absolutas y rutas relativas

Una ruta **absoluta** empieza por / y no depende de la carpeta actual. Una ruta **relativa** empieza desde donde estás. Usa el mismo fichero creado antes de las dos maneras:

```
cat notas/ideas.txt
cat /optativa-nube/p00-puesto-nube/notas/ideas.txt
```

Las dos órdenes señalan el mismo fichero. Ahora cambia de carpeta y usa una ruta relativa que incluye `..`, que significa «la carpeta anterior»:

```
cd notas
pwd
cat ideas.txt
cat ../README.md
cd ..
pwd
```

Para crear varias carpetas anidadas de una vez, sí usarás `mkdir -p`:

```
mkdir -p practicas/shell/entradas
mkdir -p practicas/shell/salidas
ls practicas/shell
```

No uses rutas que no puedas leer y explicar. En este proyecto evita espacios en los nombres: usa minúsculas, guiones y extensiones claras.

Leer, escribir y buscar texto

echo muestra texto en pantalla. Con > crea o reemplaza un fichero; con >> añade una línea sin borrar las anteriores. Crea un fichero de práctica con un texto breve:

```
echo "Lorem ipsum dolor sit amet." > notas/lorem.txt
echo "Lorem ipsum es un texto de prueba." >> notas/lorem.txt
echo "Dolor sit amet, consectetur adipiscing elit." >> notas/lorem.txt
echo "La nube permite usar recursos cuando los necesitas." >> notas/lorem.txt
cat notas/lorem.txt
```

`cat` muestra todo el fichero. Para leer solo las primeras líneas, usa head:

```
head -n 2 notas/lorem.txt
```

`grep` busca texto. `-n` añade el número de línea e `-i` ignora mayúsculas y minúsculas:

```
grep -n 'Lorem' notas/lorem.txt
grep -in 'dolor' notas/lorem.txt
```

Si buscas una palabra que no existe, `grep` no muestra líneas y termina con un código distinto de cero. Compruébalo:

```
grep -n 'NoExiste' notas/lorem.txt
echo "$?"
```

`0` significa que la búsqueda encontró texto; otro valor significa que debes interpretar el resultado antes de continuar.



Antes de usar > en un fichero, lee su contenido con `cat`: > reemplaza todo lo que hubiera. No escribas claves, contraseñas ni credenciales en la carpeta del proyecto.

Referencia de los comandos practicados

Antes de los ejercicios, revisa esta tabla. Cada orden se usará sobre el árbol que ya has creado; no necesitas probarla sobre rutas desconocidas.

Orden	Para qué la usas aquí
`pwd`	Muestra la ruta de la carpeta actual.
`whoami`	Muestra el usuario con el que estás trabajando.
`ls`	Lista nombres visibles; `-l` añade detalles, `-a` incluye ocultos y `-h` simplifica tamaños.
`cd carpeta`	Entra en una carpeta; `cd ..` vuelve al nivel anterior.
`mkdir carpeta`	Crea una carpeta; `mkdir -p ruta/anidada` crea también los niveles que falten.
`touch fichero`	Crea un fichero vacío sin escribir contenido.

Orden	Para qué la usas aquí
`cat fichero`	Muestra el contenido completo de un fichero de texto.
`echo "texto"`	Muestra texto; con > lo escribe reemplazando un fichero y con >> lo añade al final.
`cp origen destino`	Copia un fichero conservando el original.
`mv origen destino`	Mueve o renombra un fichero.
`find . -maxdepth 2 -type f`	Busca ficheros desde la carpeta actual, como máximo dos niveles por debajo.
`sort`	Ordena las líneas que recibe; | le entrega la salida de la orden anterior.
`head -n 2 fichero`	Muestra solo las dos primeras líneas.
`grep -n texto fichero`	Busca texto y muestra sus números de línea; `-i` ignora mayúsculas y minúsculas.
`wc -l fichero`	Cuenta las líneas de un fichero.
`ps`	Muestra los procesos de tu terminal.
`top`	Muestra procesos que se actualizan en tiempo real; pulsa q para salir.
`chmod u+x fichero`	Da permiso de ejecución al propietario del fichero.
`bash fichero.sh`	Ejecuta un fichero usando Bash.
`./fichero.sh`	Ejecuta un script desde la carpeta actual cuando ya tiene permiso x.

Veinte ejercicios de comandos Linux

Punto de partida: estás dentro de `/optativa-nube/p00-puesto-nube` y has creado el árbol anterior. Realiza los ejercicios en orden. En cada uno, anota el comando y una frase sobre lo que has comprobado.

1. Comprueba que estás dentro de la carpeta del proyecto.
2. Identifica el usuario con el que estás trabajando.
3. Muestra los nombres visibles de la carpeta actual.
4. Muestra también los nombres ocultos e identifica al menos uno que empiece por `..`
5. Muestra el contenido con detalles y tamaños fáciles de leer; indica qué información has añadido respecto al listado simple.
6. Crea una carpeta vacía llamada `borrador`.

7. Entra en `borrador`, comprueba la ruta y vuelve a la carpeta anterior.
8. Crea el fichero vacío `borrador/nota.txt` y comprueba que existe.
9. Escribe `Práctica de shell` en ese fichero y después léelo.
10. Añade una segunda línea al mismo fichero sin borrar la primera; vuelve a leerlo.
11. Copia `borrador/nota.txt` como `evidencias/nota-copiada.txt` y comprueba que existen ambos ficheros.
12. Renombra la copia como `evidencias/nota-shell.txt`.
13. Muestra `nota-shell.txt` primero con una ruta relativa y después con la ruta absoluta `/optativa-nube/p00-puesto-nube/evidencias/nota-shell.txt`.
14. Crea la ruta `practicas/shell/repaso` usando una sola orden.
15. Obtén una lista ordenada de los ficheros creados hasta dos niveles por debajo de la carpeta actual.
16. Busca las líneas que contienen `Lorem` en `notas/lorem.txt` e incluye su número de línea.
17. Muestra únicamente las dos primeras líneas de `notas/lorem.txt`.
18. Cuenta las líneas de `notas/lorem.txt`.
19. Muestra los permisos de la carpeta `scripts` e interpreta los del propietario.
20. Observa los procesos de tu terminal; después abre el visor de procesos, espera una actualización y ciérralo sin detener ningún proceso.

Cinco ejercicios de mayor dificultad

21. Crea la ruta `practicas/shell/repaso/resultado.txt` en un único paso y escribe dentro una línea que identifique la práctica.
22. Guarda en `evidencias/lineas-lorem.txt` solo las líneas de `notas/lorem.txt` que contienen `Lorem`, con sus números de línea.
23. Muestra los ficheros de `notas/` ordenados y guarda ese listado en `evidencias/listado-notas.txt`.
24. Desde la carpeta `notas/`, copia `lorem.txt` a `../evidencias/lorem-copia.txt` y verifica que ambas rutas apuntan a ficheros existentes.
25. Obtén el número de ficheros creados en el proyecto hasta dos niveles de profundidad. Puedes combinar las órdenes que ya conoces.

Cinco ejercicios con un comando nuevo

Crea el archivo `/optativa-nube/p00-puesto-nube/notas/lorem.txt` y realiza los siguientes ejercicios:

El contenido del fichero lorem.txt debe ser el siguiente:

```
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc ullamcorper laoreet
↪  neque, sit amet imperdiet turpis congue non. Nullam suscipit nibh vitae auctor
↪  feugiat. Duis ac felis lacus. Duis blandit, dolor eu cursus pellentesque, tellus
↪  nunc vestibulum velit, vel cursus turpis turpis ac massa. Suspendisse nisl risus,
↪  laoreet ut dolor id, pulvinar interdum sem. Ut tempor quam sed nisi condimentum, in
↪  malesuada velit dapibus. Duis vel pharetra quam. Duis tincidunt luctus purus eget
↪  dignissim. Donec ut ligula metus. In et diam mattis, volutpat nisi in, egestas
↪  tortor. Praesent ornare dui, rutrum auctor quis, auctor molestie ex. Pellentesque
↪  cursus purus quis sodales scelerisque. Aliquam consequat egestas posuere.
```

```
In condimentum sem sed tellus aliquam, et tincidunt mi tincidunt. Nam at lacinia lacus,
 ↪  sit amet malesuada enim. Pellentesque ut velit eu mauris cursus imperdiet. Aenean
 ↪  quis leo eu massa semper ultrices. Cras ac ligula vitae augue aliquet blandit ac
 ↪  vitae eros. Mauris gravida pulvinar dui, placerat vulputate ipsum consectetur
 ↪  bibendum. Aliquam sollicitudin eros et mauris interdum viverra. Vivamus eget metus
 ↪  quis quam iaculis porta semper ac eros. Proin euismod metus et dui posuere
 ↪  accumsan. Integer varius, elit lacinia auctor vulputate, sapien velit efficitur
 ↪  lectus, id imperdiet nunc enim at velit.

Nullam vitae lectus quis leo accumsan vehicula sit amet in dolor. Mauris tristique
 ↪  libero in nulla venenatis rutrum. Suspendisse ex ex, pellentesque eu tristique
 ↪  sollicitudin, maximus a diam. Nam et sagittis arcu. Cras nec sapien non velit
 ↪  tempus porttitor. Integer sed dolor dui. Aenean vehicula ipsum vitae sapien
 ↪  convallis, et cursus ex mollis. Quisque porttitor nibh eu mi tristique, in bibendum
 ↪  augue rutrum. Donec molestie ornare faucibus. Suspendisse eu tortor tempus, auctor
 ↪  quam at, malesuada metus. Proin rutrum porta nunc a sagittis. Aenean tincidunt
 ↪  commodo neque, sit amet euismod tortor placerat consectetur. Praesent facilisis ut
 ↪  eros nec dignissim. Sed non finibus libero.

Aenean nec semper augue. Nam ligula neque, auctor in leo at, porttitor consequat risus.
 ↪  Donec massa sapien, commodo vitae urna eget, tristique porttitor nibh. Etiam non
 ↪  ipsum aliquam, accumsan tortor ac, semper sem. Morbi cursus mauris augue, non
 ↪  suscipit ipsum bibendum id. Fusce congue vehicula volutpat. Aliquam venenatis eros
 ↪  eget risus vulputate tempus. Sed pulvinar dictum vulputate. Interdum et malesuada
 ↪  fames ac ante ipsum primis in faucibus. Aenean sollicitudin, felis in imperdiet
 ↪  lobortis, diam ipsum molestie ex, a pretium ex nisi vitae quam. Curabitur posuere
 ↪  placerat massa, vel sollicitudin odio egestas sit amet. Nunc vitae scelerisque
 ↪  elit. Mauris nec ante nulla.

Quisque vehicula cursus tortor non pulvinar. Cras tincidunt placerat arcu id tempus. In
 ↪  tempus mi pretium leo convallis, quis imperdiet sapien varius. Pellentesque eget
 ↪  erat eu nibh vestibulum interdum. Proin eu erat nec mauris cursus luctus nec eget
 ↪  quam. Ut eu aliquet ante. Donec leo diam, eleifend id vehicula ullamcorper,
 ↪  placerat vitae dolor. Nam sed elementum eros, in tempus nulla. Ut rhoncus ultricies
 ↪  arcu, non sagittis risus tempus laoreet. Nullam id purus turpis. Integer id rhoncus
 ↪  arcu, at euismod mauris.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc ullamcorper laoreet
 ↪  neque, sit amet imperdiet turpis congue non. Nullam suscipit nibh vitae auctor
 ↪  feugiat. Duis ac felis lacus. Duis blandit, dolor eu cursus pellentesque, tellus
 ↪  nunc vestibulum velit, vel cursus turpis turpis ac massa. Suspendisse nisl risus,
 ↪  laoreet ut dolor id, pulvinar interdum sem. Ut tempor quam sed nisi condimentum, in
 ↪  malesuada velit dapibus. Duis vel pharetra quam. Duis tincidunt luctus purus eget
 ↪  dignissim. Donec ut ligula metus. In et diam mattis, volutpat nisi in, egestas
 ↪  tortor. Praesent arcu dui, rutrum at tortor quis, auctor molestie ex. Pellentesque
 ↪  cursus purus quis sodales scelerisque. Aliquam consequat egestas posuere.
```

```
In condimentum sem sed tellus aliquam, et tincidunt mi tincidunt. Nam at lacinia lacus,
 ↪  sit amet malesuada enim. Pellentesque ut velit eu mauris cursus imperdiet. Aenean
 ↪  quis leo eu massa semper ultrices. Cras ac ligula vitae augue aliquet blandit ac
 ↪  vitae eros. Mauris gravida pulvinar dui, placerat vulputate ipsum consectetur
 ↪  bibendum. Aliquam sollicitudin eros et mauris interdum viverra. Vivamus eget metus
 ↪  quis quam iaculis porta semper ac eros. Proin euismod metus et dui posuere
 ↪  accumsan. Integer varius, elit lacinia auctor vulputate, sapien velit efficitur
 ↪  lectus, id imperdiet nunc enim at velit.

Nullam vitae lectus quis leo accumsan vehicula sit amet in dolor. Mauris tristique
 ↪  libero in nulla venenatis rutrum. Suspendisse ex ex, pellentesque eu tristique
 ↪  sollicitudin, maximus a diam. Nam et sagittis arcu. Cras nec sapien non velit
 ↪  tempus porttitor. Integer sed dolor dui. Aenean vehicula ipsum vitae sapien
 ↪  convallis, et cursus ex mollis. Quisque porttitor nibh eu mi tristique, in bibendum
 ↪  augue rutrum. Donec molestie ornare faucibus. Suspendisse eu tortor tempus, auctor
 ↪  quam at, malesuada metus. Proin rutrum porta nunc a sagittis. Aenean tincidunt
 ↪  commodo neque, sit amet euismod tortor placerat consectetur. Praesent facilisis ut
 ↪  eros nec dignissim. Sed non finibus libero.

Aenean nec semper augue. Nam ligula neque, auctor in leo at, porttitor consequat risus.
 ↪  Donec massa sapien, commodo vitae urna eget, tristique porttitor nibh. Etiam non
 ↪  ipsum aliquam, accumsan tortor ac, semper sem. Morbi cursus mauris augue, non
 ↪  suscipit ipsum bibendum id. Fusce congue vehicula volutpat. Aliquam venenatis eros
 ↪  eget risus vulputate tempus. Sed pulvinar dictum vulputate. Interdum et malesuada
 ↪  fames ac ante ipsum primis in faucibus. Aenean sollicitudin, felis in imperdiet
 ↪  lobortis, diam ipsum molestie ex, a pretium ex nisi vitae quam. Curabitur posuere
 ↪  placerat massa, vel sollicitudin odio egestas sit amet. Nunc vitae scelerisque
 ↪  elit. Mauris nec ante nulla.

Quisque vehicula cursus tortor non pulvinar. Cras tincidunt placerat arcu id tempus. In
 ↪  tempus mi pretium leo convallis, quis imperdiet sapien varius. Pellentesque eget
 ↪  erat eu nibh vestibulum interdum. Proin eu erat nec mauris cursus luctus nec eget
 ↪  quam. Ut eu aliquet ante. Donec leo diam, eleifend id vehicula ullamcorper,
 ↪  placerat vitae dolor. Nam sed elementum eros, in tempus nulla. Ut rhoncus ultricies
 ↪  arcu, non sagittis risus tempus laoreet. Nullam id purus turpis. Integer id rhoncus
 ↪  arcu, at euismod mauris.
```

26. Muestra únicamente las dos últimas líneas de `notas/lorem.txt`.

Pista: `tail`

27. Averigua cuánto espacio ocupa la carpeta actual y muestra el resultado de forma fácil de leer.

Pista: `du`

28. A partir de la ruta absoluta `/optativa-nube/p00-puesto-nube/notas/lorem.txt`, muestra solo el nombre final del fichero.

Pista: `basename`

29. Muestra el UID, el GID y los grupos del usuario con el que trabajas.

Pista: `id`

30. Muestra la información completa del sistema Linux que ejecuta el contenedor.

Pista: `uname`

Cuando los hayas intentado, conserva esta idea de cada orden: `tail` lee el final de un fichero; `du` calcula el espacio usado; `basename` extrae el último nombre de una ruta; `id` identifica el usuario y sus grupos; `uname` informa del sistema operativo. Para descubrir las opciones de cada una, consulta su ayuda integrada con `NOMBRE_DEL_COMANDO --help`.

Cuatro scripts básicos

Un script es un fichero de texto que contiene órdenes de Bash. Primero ejecutarás scripts muy cortos con comandos que ya conoces. Todos se crean dentro de `scripts/` y se ejecutan desde `/optativa-nube/p00-puesto-nube`.

1. Mostrar un mensaje

Con tu editor, crea `scripts/01-saludo.sh`:

```
#!/usr/bin/env bash
echo "Hola desde mi primer script de Bash"
```

La primera línea indica que se usará Bash. La segunda es la misma orden `echo` que ya has ejecutado en la terminal. Pruébalo con Bash:

```
bash scripts/01-saludo.sh
```

Ahora permite ejecutarlo directamente y vuelve a probarlo:

```
chmod u+x scripts/01-saludo.sh
./scripts/01-saludo.sh
```

2. Mostrar el entorno actual

Crea `scripts/02-entorno.sh`:

```
#!/usr/bin/env bash
echo "Usuario:"
whoami
echo "Carpeta actual:"
pwd
```

Ejecútalo y comprueba que aparecen `alumno` y una ruta dentro de `/optativa-nube`:

```
bash scripts/02-entorno.sh
```

3. Listar el proyecto

Crea `scripts/03-listar-proyecto.sh`:

```
#!/usr/bin/env bash
echo "Contenido del proyecto:"
ls
echo "Ficheros de notas:"
ls notas
```

Antes de ejecutarlo, lee las órdenes y predice qué dos grupos de nombres mostrará. Después compruébalo:

```
bash scripts/03-listar-proyecto.sh
```

4. Buscar texto en un fichero conocido

Crea `scripts/04-buscar-lorem.sh`:

```
#!/usr/bin/env bash
echo "Líneas que contienen Lorem:"
grep -n 'Lorem' notas/lorem.txt
```

Ejecuta el script y compara su salida con el resultado del ejercicio 16:

```
bash scripts/04-buscar-lorem.sh
```

Estos scripts no reciben argumentos, no modifican ficheros y no usan variables: solo reúnen órdenes conocidas. Antes de pasar al siguiente apartado, comprueba que los cuatro existen:

```
ls -l scripts
```

De órdenes repetidas a un script pequeño

Un script de Bash es un archivo de texto con una secuencia de órdenes. Sirve para repetir una tarea y dejar claro el procedimiento. No convierte automáticamente una tarea en segura: debes acotar sobre qué archivos actúa y comprobar los resultados.

Crear el script de inventario

Con tu editor, crea `scripts/registrar-entorno.sh` con el contenido siguiente. El archivo pertenece a `/optativa-nube`, por lo que estará disponible al reiniciar el contenedor.

```
#!/usr/bin/env bash
# Registra información mínima y no sensible del puesto de P00.
set -eu

RUTA_BASE="${1:-/optativa-nube/p00-puesto-nube}"
DIRECTORIO_EVIDENCIAS="$RUTA_BASE/evidencias"
ARCHIVO="$DIRECTORIO_EVIDENCIAS/entorno-$(date +%F).txt"

if [[ ! -d "$RUTA_BASE" ]]; then
  echo "No existe la ruta de trabajo: $RUTA_BASE" >&2
  exit 1
fi

mkdir -p "$DIRECTORIO_EVIDENCIAS"

{
  echo "Inventario básico de P00"
  echo "Fecha: $(date --iso-8601=seconds)"
  echo "Ruta: $RUTA_BASE"
  echo "Usuario del contenedor: $(whoami)"
  echo "Contenido inicial:"
  ls -la "$RUTA_BASE"
} > "$ARCHIVO"

echo "Evidencia creada: $ARCHIVO"
```

Este ejemplo reúne varias ideas importantes:

Elemento	Función en el script
`#!/usr/bin/env bash`	Indica que el archivo debe interpretarse con Bash.
`# ...`	Es un comentario: explica una decisión y Bash lo ignora.
`set -eu`	Detiene el script ante un error y ante una variable sin definir. No sustituye la revisión humana.
`RUTA_BASE=...`	Guarda una ruta en una variable; no se dejan espacios alrededor de =.
`${1:-valor}`	Usa el primer argumento si existe; si no, usa una ruta segura por defecto.
`$(comando)`	Ejecuta una orden y coloca su resultado dentro de un texto o una variable.
`date +%F`	Produce la fecha en formato `AAAA-MM-DD` para el nombre del archivo.
`[[ -d ruta ]]`	Comprueba que una ruta existe y es una carpeta.
`if ...; then ... fi`	Toma una decisión según el resultado de una comprobación.
`exit 1`	Detiene el script e indica que no pudo completar su tarea.
`"$RUTA_BASE"`	Conserva la variable como una sola ruta, incluso si algún día contiene espacios.
`>&2`	Envía un mensaje de error a la salida de errores.
`{ ... } > archivo`	Agrupa varias salidas y las guarda en una única evidencia.
`bash -n archivo`	Revisa la sintaxis sin ejecutar el script.
`bash -x archivo`	Muestra cada orden justo antes de ejecutarla para facilitar la revisión.

No incluyas en el script credenciales, contenido de `.env`, claves SSH ni la salida completa de configuraciones sensibles. Un script puede automatizar un error tan rápido como una tarea útil.

Probar antes de ejecutar

Primero revisa la sintaxis sin ejecutar el script:

```
bash -n scripts/registrar-entorno.sh
echo "$?"
```

El comando bash -n se utiliza para verificar la sintaxis de un script de Bash sin llegar a ejecutarlo. La opción -n viene de "noexec" (no ejecución). Cuando corres este comando, Bash lee todo el archivo de arriba a abajo y busca errores de estructura o de escritura (como un if sin cerrar con su respectivo fi, comillas mal cerradas o paréntesis faltantes).

Si la sintaxis es correcta, el código de salida será `0` y no aparecerá salida adicional. Ahora ejecútalo explícitamente con Bash y comprueba el archivo creado:

```
bash scripts/registrar-entorno.sh
ls -l evidencias/entorno-*.txt
head -n 20 evidencias/entorno-"$(date +%F)".txt
```

Para ejecutarlo directamente, concede permiso de ejecución solo al propietario y vuelve a comprobar los permisos:

```
chmod u+x scripts/registrar-entorno.sh
ls -l scripts/registrar-entorno.sh
./scripts/registrar-entorno.sh
```

La línea de permisos debe incluir una `x` en el bloque del propietario. Si el script falla, no lo arregles a ciegas. Ejecuta el modo de trazado para ver cada orden justo antes de que Bash la ejecute:

```
bash -x scripts/registrar-entorno.sh
```

El modo `-x` es útil para localizar una ruta o variable inesperada. No copies su salida a un repositorio público sin revisarla: puede revelar nombres de archivos y rutas del entorno.

Variar el script con un argumento controlado

El script acepta opcionalmente una ruta como primer argumento. Esta ejecución usa la carpeta `web` y, como no tiene todavía `evidencias/`, el script la crea dentro de ella:

```
bash scripts/registrar-entorno.sh /optativa-nube/p00-puesto-nube/web
ls -la web/evidencias
```

Esta variante demuestra que `$1` puede adaptar un script. Antes de pasar una ruta real, léela y compárala con `pwd`; no ejecutes scripts de inventario sobre rutas que no comprendas.

Refuerzo: repetición con un bucle pequeño

Cuando la misma comprobación se aplica a varios archivos conocidos, un bucle evita copiar y pegar. El siguiente ejemplo solo informa de si existen tres rutas del proyecto; no crea, borra ni modifica ninguna:

```
for ruta in README.md web evidencias; do
  if [[ -e "$ruta" ]]; then
    echo "Existe: $ruta"
  else
    echo "Falta: $ruta"
  fi
done
```

El bucle toma un valor cada vez y lo guarda en `ruta`. La condición `[[ -e ... ]]` pregunta si existe. Esta estructura prepara las comprobaciones repetibles que harás más adelante con archivos de proyecto, servicios y evidencias; no se usa para recorrer rutas desconocidas ni para ejecutar borrados masivos.

Cierre de sesión y conexión con lo que sigue

Antes de salir, deja la evidencia donde corresponde y revisa la estructura:

```
cd /optativa-nube/p00-puesto-nube
find . -maxdepth 2 -type f | sort
exit
exit
```

El primer `exit` cierra la sesión de `alumno` y vuelve a la shell inicial de `root`; el segundo cierra Bash y detiene `p00-lab`. En la siguiente página convertirás esta misma carpeta en un repositorio Git: los nombres claros, las rutas conocidas y la bitácora que has creado facilitarán revisar qué cambia antes de cada *commit*.

Fuente para ampliar

- Tutorial de programación de Bash para principiantes de freeCodeCamp: conceptos de shell, scripts, variables, entradas/salidas, condiciones, bucles y depuración.
- Manual de Bash de GNU: referencia de tuberías, redirecciones y características básicas de la shell.

Antes de empezar: retoma tu puesto de trabajo

En P00/02 creaste el árbol de `/optativa-nube/p00-puesto-nube`. Ahora lo convertirás en un proyecto que conserva una historia de sus cambios. Durante el curso contendrá la web de prueba, scripts y evidencias, pero nunca secretos.

```
/optativa-nube/p00-puesto-nube/
├── README.md             # qué es el proyecto y cómo comprobarlo
├── notas/                # ficheros de las prácticas de shell
├── scripts/              # automatizaciones pequeñas
├── web/                  # web estática de prácticas posteriores
└── evidencias/           # notas y comprobaciones realizadas
```

Los comandos de esta página se ejecutan **dentro del contenedor `p00-lab`**. Desde la terminal del anfitrión, entra en él:

```
docker start -ai p00-lab
su - alumno
```

`docker start` abre la shell inicial del contenedor como `root`; `su - alumno` inicia la sesión de trabajo sin privilegios creada en P00/01. `-a` conecta la salida a tu terminal e `-i` mantiene abierta la entrada para que puedas escribir.

Punto de partida: ya ves el *prompt* que termina en `$`. Comprueba dónde trabajas, qué usuario eres y que Git forma parte de la imagen del curso:

```
pwd
whoami
git --version
```

- `pwd` debe mostrar `/home/alumno`; entrarás en `/optativa-nube` antes de trabajar con el repositorio.
- `whoami` debe mostrar `alumno`.
- `git --version` muestra la versión instalada durante P00/01; no hace falta instalarlo de nuevo en cada sesión mientras conserves `p00-lab`.

Si alguno de esos resultados no coincide, no inicialices ningún repositorio. Comprueba que has iniciado `p00-lab` y que has ejecutado `su - alumno`. Git se instaló en el contenedor durante P00/01; no se ha creado ninguna imagen propia ni se ha usado `docker commit`.



Todo lo que versionarás se guarda en `/optativa-nube`, la carpeta compartida con el anfitrión. Las claves SSH se guardarán fuera de ella, en `/home/alumno/.ssh`. Así no entran por error en el repositorio.

Git: decidir qué historia conserva el proyecto

Git es un sistema de control de versiones distribuido. Te permite comparar cambios, recuperar una versión anterior y explicar qué decisión representa cada cambio. No sustituye una copia de seguridad completa, pero evita depender de archivos llamados `final-definitivo-ahora-si.md`.

Las tres zonas de Git

Antes de escribir comandos, distingue estas tres zonas:

Zona	Qué contiene	Pregunta que debes hacerte
Directorio de trabajo	Los archivos que editas ahora.	«¿Qué he cambiado?»
Área de preparación (*staging*)	Solo los cambios que eliges guardar juntos.	«¿Quiero incluir exactamente esto?»
Historial local	Los *commits* ya confirmados.	«¿Puedo explicar por qué se guardó?»



Figura 2: Flujo seguro de Git: directorio de trabajo, área de preparación e historial

Un *commit* no es un botón de guardar automático: es una fotografía pequeña y razonada. El recorrido será siempre: **observar → elegir → volver a revisar → confirmar**.

Crear el repositorio local

Punto de partida: terminaste P00/02 y existe `/optativa-nube/p00-puesto-nube`. Entra en la carpeta y observa su contenido antes de cambiar nada:

```
cd /optativa-nube/p00-puesto-nube
pwd
ls
```

Ahora inicializa Git y consulta su estado:

```
git init --initial-branch=main
git status
```

- `git init` crea la carpeta oculta `.git`, donde Git guarda el historial. No la borres ni la copies a mano.
- `--initial-branch=main` asigna el nombre `main` a la primera rama.
- `git status` indica la rama, los archivos que Git todavía no sigue y los cambios pendientes. Será tu comando de orientación antes y después de cada operación importante.

Git necesita saber qué autor figura en los *commits*. Esta configuración es **local**: solo afecta a este repositorio, no a otros proyectos ni al anfitrión. Sustituye los ejemplos por los valores indicados para la práctica:

```
git config --local user.name "Nombre Apellido"
git config --local user.email "identificador@centro.example"
git config --local --list
```

`git config` guarda una configuración; `--local` la limita a este proyecto; `--list` muestra lo guardado. La última orden no publica nada.

Proteger antes de guardar: `.gitignore`

`.gitignore` indica qué patrones debe ignorar Git **antes** de preparar un archivo. Con tu editor, crea `.gitignore` en la raíz del proyecto con este contenido:

```
# Secretos: nunca se versionan
.env
.env.*
*.pem
*.key

# Archivos generados o temporales
node_modules/
dist/
*.log
```

Los patrones `.env` y `.env.*` cubren configuraciones habituales; `*.pem` y `*.key` cubren nombres frecuentes de claves. Esta regla evita un descuido futuro, pero no borra un secreto que ya se haya incluido en un *commit*.

Comprueba que Git ve `.gitignore` y el resto de archivos de la práctica:

```
git status --short
```

`--short` muestra una versión abreviada del estado. Los archivos sin seguimiento empiezan con `??`. Todavía no estás guardando nada: solo observas.

Primera fotografía: estructura y protección

Con tu editor, revisa que `README.md` contiene al menos esta información. Si ya contiene otro texto correcto de P00/02, añade lo que falte sin borrar tu trabajo:

```
# Puesto de nube

Repositorio de prácticas de P00.

## Comprobación

La web se servirá desde la carpeta `web/` en prácticas posteriores.
```

Prepara los archivos conocidos que forman esta primera decisión y mira exactamente su contenido preparado. No añadas una carpeta cuyo contenido no hayas revisado:

```
git add README.md .gitignore notas scripts web
git diff --staged
git status --short
```

- `git add` copia al área de preparación la versión actual de cada ruta indicada. Aquí las rutas proceden del árbol conocido de P00/02; no uses `git add .`, porque podría añadir un archivo que no has revisado.
- `git diff --staged` muestra la fotografía preparada; léela antes de continuar.
- En `git status --short`, la letra A indica «preparado para añadir».

Si aparece una contraseña, una clave privada o un archivo que no esperabas, no hagas *commit*. Retíralo del área de preparación sin borrar el archivo de tu carpeta:

```
git restore --staged nombre-del-archivo
git status --short
```

`git restore --staged` deshace únicamente la selección previa de `git add`. Sustituye `nombre-del-archivo` por el nombre real que acabas de revisar; no copies esas palabras literalmente.

Cuando la fotografía sea correcta, confírmala y comprueba el resultado:

```
git commit -m "Crea la estructura inicial y protege secretos"
git status
git log --oneline --decorate -3
```

- `git commit -m "mensaje"` crea una entrada de historial con una explicación breve.
- `git log --oneline` muestra un *commit* por línea; `--decorate` añade nombres de ramas y `-3` limita la salida a los tres últimos.

Verificación: `git log` debe mostrar tu *commit* en `main`. Si `git status` todavía muestra algún ejercicio de P00/02, no lo añadas por rutina: revisa su contenido y decide si pertenece a esta historia o debe permanecer fuera.

Segunda fotografía: una página y una bitácora

Una práctica posterior necesita una página. Con tu editor, crea o completa `web/index.html` con este ejemplo mínimo:

```
<!doctype html>
<html lang="es">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <title>Puesto de nube</title>
  </head>
  <body>
    <h1>Mi puesto de nube</h1>
    <p>Esta página se versiona con Git.</p>
  </body>
</html>
```

Crea también `evidencias/bitacora-git.md` y escribe la fecha, una conclusión sobre Git y una sobre SSH. Son evidencias de tu proceso, no un lugar para copiar salidas completas ni datos personales.

Antes de seleccionar archivos, compara los cambios con el último *commit*:

```
git status
git diff
```

`git diff` muestra cambios del directorio de trabajo que aún no has preparado. Decide que la página y la bitácora son una misma mejora, prepáralas, revísalas y confírmalas:

```
git add web/index.html evidencias/bitacora-git.md
git diff --staged
git commit -m "Añade la página inicial y la bitácora de Git"
git log --oneline --decorate -3
```

El patrón que repetirás es: **editar → `git status` → `git diff` → `git add` selectivo → `git diff --staged` → `git commit`**. Los mensajes empiezan con un verbo y explican el cambio: «Añade…», «Corrige…», «Documenta…». No uses `cambios`, `prueba` o `final`.

Corregir antes de confirmar

Equivocarse forma parte del proceso; publicar el error sin revisarlo no es inevitable. Practica con un fichero inocuo que no forma parte de la entrega:

```
touch notas-privadas.txt
git add notas-privadas.txt
git status --short
git restore --staged notas-privadas.txt
git status --short
```

`touch` crea el fichero vacío y ya lo conoces de P00/02. Tras `git restore --staged`, el fichero continúa en tu directorio de trabajo, pero deja de estar preparado. Si no necesitas conservarlo, elimínalo con el gestor de archivos del anfitrión. Si fuera una clase de archivo que nunca debe versionarse, añade un patrón concreto a `.gitignore`, revisa el cambio y crea un *commit* que documente esa protección.

Para que ese ejemplo no quede como cambio pendiente, muévelo a `.env`, que ya está excluido por la regla creada antes:

```
mv notas-privadas.txt .env
git status --ignored --short
```

`mv` cambia la ubicación o el nombre de un fichero. `--ignored` añade a la salida los archivos excluidos: debes ver `.env` con `!!`, no preparado para un *commit*.

Para renombrar un archivo que ya está versionado, usa `git mv`; equivale a mover o renombrar el archivo y preparar el cambio a la vez:

```
git mv evidencias/bitacora-git.md evidencias/bitacora-p00.md
git status
git diff --staged
git commit -m "Renombra la bitácora general de P00"
```

No ejecutes comandos de recuperación que no entiendas ni copies instrucciones que usen `--force` para «arreglar» un historial. Ante la duda, detente, ejecuta `git status` y conserva una evidencia útil antes de pedir ayuda.

Retos de comprobación de Git

Punto de partida: has realizado las dos fotografías y estás en `/optativa-nube/p00-puesto-nube`. Realiza cada reto en orden. En `evidencias/bitacora-p00.md`, anota la orden que has usado y una conclusión; no copies secretos ni salidas completas.

1. Comprueba en qué rama estás y describe qué cambios pendientes hay, si los hubiera. Antes de añadirlos, indica por qué pertenecen o no al repositorio.
2. Muestra los dos últimos *commits* en una línea cada uno e identifica qué decisión representa cada uno.
3. Modifica una frase de `README.md`; compara el cambio antes de prepararlo y después de prepararlo.
4. Retira `README.md` del área de preparación sin perder su cambio y explica qué zona de Git ha cambiado.
5. Prepara de nuevo `README.md`, revísalo y crea un *commit* con un mensaje que explique la mejora.
6. Crea un fichero vacío llamado `.env` y comprueba que Git lo ignora. **Pista:** `git status --ignored --short`.
7. Añade al menos un patrón concreto a `.gitignore` para un fichero temporal que hayas creado durante P00. Revisa y confirma solo esa modificación.

Después de intentar el reto 6, conserva esta idea: `--ignored` muestra también los archivos excluidos y la salida `!! .env` confirma que `.gitignore` funciona. No escribas una credencial real para hacer la prueba.

Compartir solo cuando exista un destino aprobado

Un repositorio local ya sirve para aprender y para entregar una carpeta comprimida si se solicita. Git no exige GitHub ni publicar el proyecto en Internet.

La conexión con un remoto queda fuera del bloque hasta que estén confirmados plataforma, URL y flujo de entrega. No inventes ni reutilices URLs, usuarios, *tokens* ni claves de otra persona. Tampoco clones repositorios dentro de `p00-puesto-nube` ni sustituyas el repositorio local de esta práctica.

SSH: un canal cifrado que exige verificar la identidad

SSH (*Secure Shell*) permite ejecutar órdenes o transferir archivos mediante una conexión cifrada. Aquí el objetivo no es memorizar una conexión: es saber a quién te conectas y qué credencial utilizas.

SSH interviene en dos identidades diferentes:

Identidad	Para qué sirve	Dónde se registra
Clave de host del servidor	Demuestra que el servidor al que llegas es el esperado.	`known_hosts` del cliente, tras verificar su huella.
Clave del usuario	Demuestra que puedes acceder a una cuenta autorizada.	Clave privada en tu entorno y clave pública en el servidor autorizado.

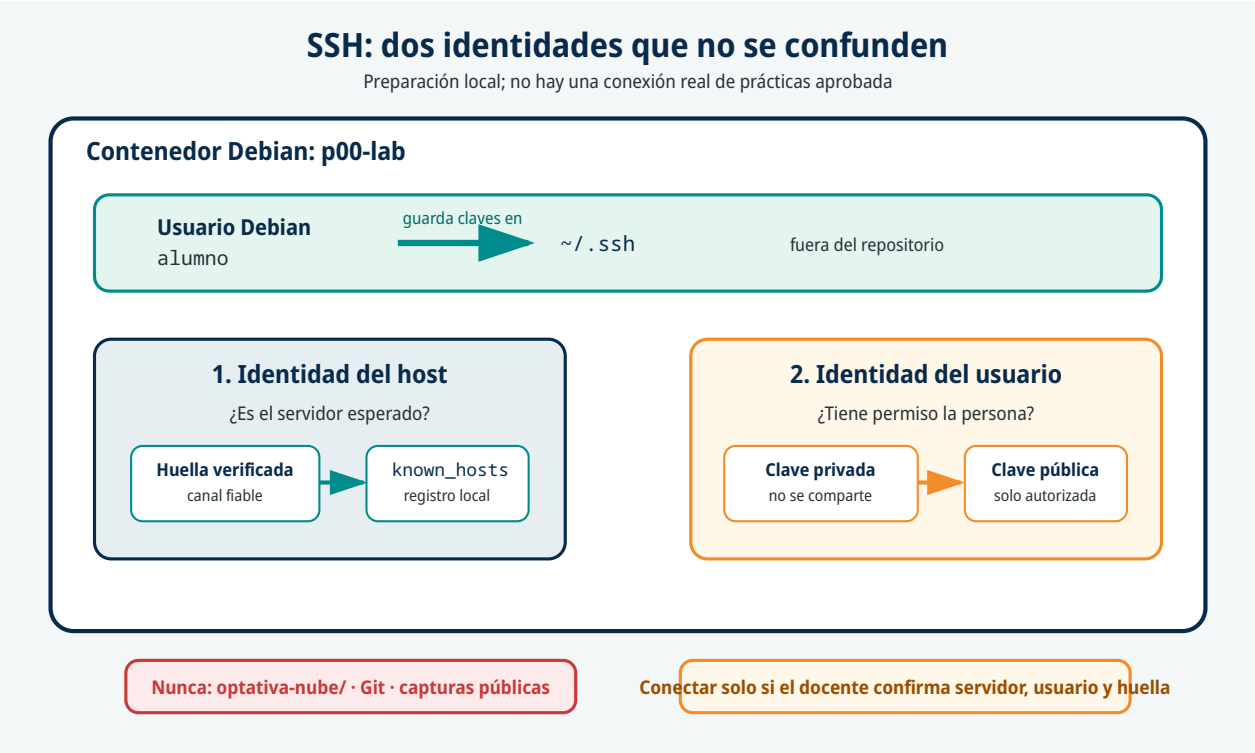


Figura 3: Identidades SSH separadas: huella del host y known_hosts por un lado; clave privada y pública del usuario alumno en su carpeta personal, fuera de optativa-nube y Git.

Aceptar una clave de host desconocida sin comprobar su huella equivale a no verificar quién está al otro lado. Antes de una primera conexión, el docente o la documentación oficial debe proporcionar la huella del servidor por un canal fiable.

Instalar el cliente y preparar la carpeta privada

Git se instaló al preparar `p00-lab` en P00/01; SSH se instala ahora porque se introduce en esta práctica. **Punto de partida:** sigues dentro de `p00-lab` como `alumno`.

```
sudo apt-get update
sudo apt-get install --yes openssh-client
ssh -V
```

`sudo` eleva permisos solo para esa orden; `apt-get update` actualiza el índice de paquetes; `apt-get install --yes openssh-client` instala el cliente sin pedir confirmación; `ssh -V` muestra su versión. Esta instalación permanece mientras exista el contenedor. Si SSH se convierte en una herramienta recurrente, se añadirá al Dockerfile versionado con una etiqueta nueva de `p00-lab`; no se usará `docker commit`.

Las claves pertenecen a tu identidad, no al proyecto. `~` representa tu carpeta personal y, como eres `alumno`, `~/.ssh` equivale a `/home/alumno/.ssh`:

```
mkdir -p ~/.ssh
chmod 700 ~/.ssh
ls -ld ~/.ssh
```

`mkdir -p` crea la carpeta solo si falta; `chmod 700` permite acceder únicamente a su propietario; `ls -ld` muestra los permisos de la carpeta, sin listar su interior. No apliques `chmod -R` de forma indiscriminada ni intentes corregir con permisos un problema de un montaje del anfitrión.

Crear una pareja de claves para una práctica autorizada

Primero mira qué hay en la carpeta. Si ves `p00_ed25519` o `p00_ed25519.pub`, no ejecutes la orden de creación con el mismo nombre: conserva la clave existente y consulta qué hacer.

```
ls -la ~/.ssh
```

Solo si esos nombres no aparecen, crea la pareja de práctica:

```
ssh-keygen -t ed25519 -a 64 -f ~/.ssh/p00_ed25519 -C "p00-clave-practica"
chmod 600 ~/.ssh/p00_ed25519
chmod 644 ~/.ssh/p00_ed25519.pub
ls -l ~/.ssh/p00_ed25519*
```

- `ssh-keygen` crea y gestiona claves SSH.
- `-t ed25519` elige el tipo de clave Ed25519.

- `-a 64` aumenta el trabajo necesario para comprobar la frase de paso de una clave privada cifrada.
- `-f` fija el nombre y la ruta de los dos archivos; `-C` añade una etiqueta reconocible a la clave pública.
- `chmod 600` deja la clave privada solo para su propietario; `chmod 644` permite leer la pública, que no es secreta.

La orden te pedirá una frase de paso: elige una que recuerdes y no la compartas. Si la olvidas, no se puede recuperar de forma legítima; será necesario crear otra clave y revocar la anterior si se hubiera autorizado.

Archivo	¿Se puede compartir?	Uso
`~/.ssh/p00_ed25519`	No. Es privado.	Demuestra que controlas la identidad de usuario.
`~/.ssh/p00_ed25519.pub`	Sí, solo con el destino autorizado.	Se instala en el servidor para autorizar el acceso.

Para revisar qué clave pública vas a facilitar, muestra únicamente el archivo terminado en `.pub`:

```
cat ~/.ssh/p00_ed25519.pub
```

`cat` muestra el contenido de un fichero de texto. No ejecutes la misma orden sobre `p00_ed25519`: una clave privada no se copia a una actividad, captura, correo, repositorio ni conversación. El docente o administrador añade la clave pública a la cuenta remota autorizada; no modifiques a ciegas `authorized_keys` de un servidor compartido.

Primera conexión: verificar primero el host

Una conexión real requiere tres datos obtenidos por un canal fiable: nombre del servidor, usuario autorizado y huella SHA256 de su clave de host. Esos datos todavía no están aprobados para P00, por lo que aquí no hay una orden de conexión, transferencia ni remoto Git por SSH.

La preparación sí es comprobable: antes de aceptar una clave de host nueva, compara su huella carácter por carácter con la proporcionada. Solo entonces SSH registra esa identidad de host en `~/.ssh/known_hosts`. Una advertencia de cambio de clave no se ignora: puede corresponder a un cambio legítimo, pero debes verificarlo antes de actualizar nada.

Retos de comprobación de SSH

1. Comprueba que `~/.ssh` pertenece a `alumno` y que sus permisos no dan acceso a otros usuarios.
2. Identifica cuál de los dos archivos de la pareja puede facilitarse a un servidor autorizado y explica por qué.
3. Busca el lugar donde SSH registrará la identidad de un host y explica qué verificación falta antes de crearlo.
4. Explica qué harías ante una advertencia de cambio de huella. No borres `known_hosts` para resolverla.

A entregar

Entrega o conserva en tu carpeta de trabajo:

- El repositorio `/optativa-nube/p00-puesto-nube` con `.gitignore`, `README.md`, la página mínima y una bitácora sin datos sensibles.
- Al menos tres *commits* pequeños, revisados y con mensajes explicativos.
- En `evidencias/bitacora-p00.md`, las conclusiones de los retos de Git y SSH; no incluyas claves privadas, contraseñas, *tokens*, credenciales ni salidas completas.
- La pareja SSH almacenada exclusivamente en `/home/alumno/.ssh`, con los permisos comprobados. Si no hay destino autorizado, no se realiza una conexión real.

Ideas que reaparecerán en las prácticas de nube

- Trabajarás desde una carpeta persistente y comprobarás el estado antes de cambiar nada.
- Harás *commits* pequeños que expliquen una mejora concreta; así podrás justificar decisiones y recuperar una versión estable.
- No versionarás secretos: `.env`, claves privadas y archivos de credenciales se excluyen antes del primer *commit*.
- Verificarás la identidad de un host SSH antes de enviar una clave o contraseña. La misma precaución se aplicará a dominios, certificados y servicios de nube.

Fuentes para ampliar y comprobar

- Documentación oficial de Git: referencia de comandos y conceptos.
- Libro oficial Pro Git: explicación progresiva de repositorios, área de preparación e historial.
- Manual de `ssh-keygen` de OpenSSH: referencia para claves, huellas y archivos conocidos.
- Manual de `ssh` de OpenSSH: referencia del cliente y de la comprobación de hosts.

Diagnosticar no es probar órdenes al azar

Una aplicación web puede fallar por motivos distintos: el nombre no apunta al destino esperado, no hay ningún proceso escuchando, el puerto no se publicó, la aplicación responde con un error HTTP o el contenedor ni siquiera sigue ejecutándose. Decir «la web no va» mezcla todas esas posibilidades y no ayuda a repararla.

En esta práctica cada prueba responde a una pregunta concreta. Primero anotamos una hipótesis, ejecutamos la comprobación mínima, interpretamos el resultado y solo entonces elegimos la siguiente prueba. Trabajaremos únicamente con `localhost`, con el contenedor de práctica y con servicios que haya indicado el docente.

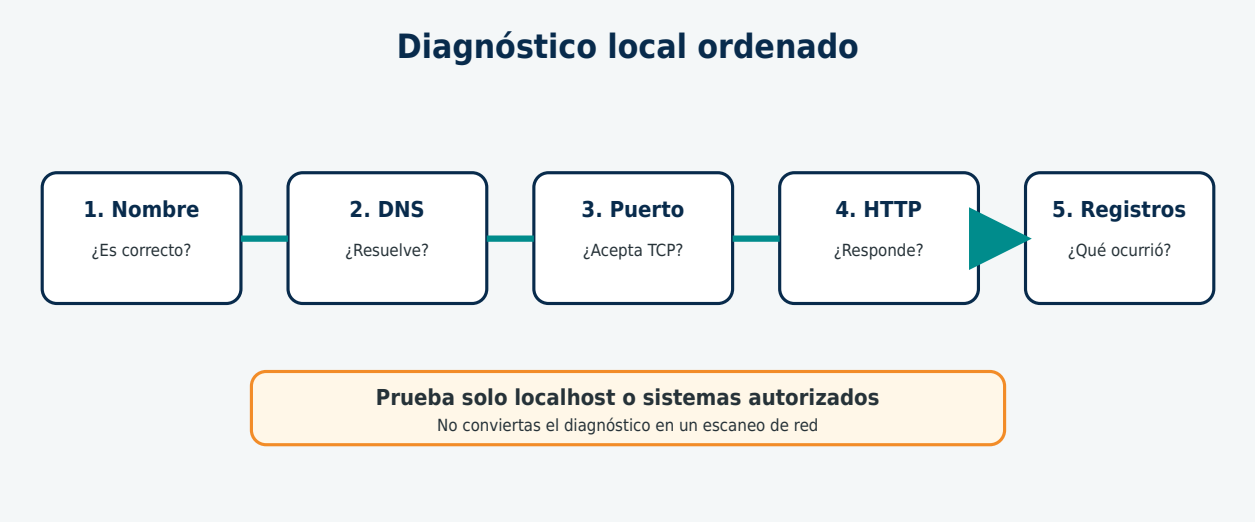


Figura 4: Flujo de diagnóstico local: nombre, DNS, puerto, HTTP y registros; las pruebas se limitan a localhost o a sistemas autorizados.

```
Navegador no muestra la página
          │
          ▼
¿Estoy en el equipo y URL correctos? ── no ──► corrige contexto o URL
          │ sí
          ▼
¿El contenedor está en ejecución? ── no ──► revisa estado y registros
          │ sí
          ▼
¿El puerto está publicado y acepta conexión? ── no ──► revisa el mapeo
          │ sí
          ▼
¿HTTP devuelve el estado esperado? ── no ──► revisa ruta, contenido y registros
          │ sí
          ▼
La comprobación funciona: registra un resultado mínimo y útil.
```

Preparar el contexto y las herramientas

Las comprobaciones de Docker se hacen en el **anfitrión**; las de nombre y TCP se pueden hacer también desde `p00-lab`. Si vienes de P00/03, ejecuta primero `exit` para salir de `alumno` y un segundo `exit` para volver al anfitrión. Cuando necesites comprobar el host desde Debian, vuelve a entrar con `docker start -ai p00-lab` y después `su - alumno`.

Punto de partida: estás dentro de `p00-lab` como `alumno`. Instala solo las herramientas que se usarán en esta página y confirma que están disponibles:

```
sudo apt-get update
sudo apt-get install --yes curl iproute2 netcat-openbsd
curl --version
ss --version
```

Herramienta	Pregunta que ayuda a responder	No demuestra por sí sola
`getent`	«¿El nombre se puede resolver?»	Que una web funcione.
`nc -vz`	«¿Se acepta una conexión TCP a este puerto?»	Que el servidor hable HTTP correctamente.
`curl`	«¿Qué responde HTTP y qué cabeceras entrega?»	Que el navegador vaya a mostrar exactamente lo mismo.
`ss`	«¿Qué puertos escucha este sistema?»	Qué servicio hay detrás de otro equipo.
`docker logs`	«¿Qué ha registrado el proceso del contenedor?»	Que el puerto sea accesible desde fuera.

`curl --version` y `ss --version` confirman dos de los paquetes instalados; `nc` se comprobará al realizar una única conexión TCP controlada. `getent` ya forma parte de Debian y consulta la resolución configurada en el sistema. Esta instalación permanece mientras exista `p00-lab`, pero no modifica la imagen. Si las herramientas se convierten en requisito recurrente, se incorporarán al Dockerfile versionado con una etiqueta nueva; no se usará `docker commit`.



No realices escaneos de puertos, consultas masivas de DNS ni pruebas contra direcciones ajenas. Esta página se limita a `localhost`, al nombre del anfitrión cuando esté autorizado y a los servicios indicados en clase.

El caso de prueba: una web local publicada en el puerto 8080

En la siguiente página crearás la web del proyecto y la publicarás en `127.0.0.1:8080`. Para practicar el diagnóstico antes, crea un contenedor temporal desde el **anfitrión**. Antes, comprueba si el nombre de práctica ya está ocupado:

```
docker container ls --all --filter name=p00-diagnostico-web
```

Si aparece un contenedor con ese nombre, no crees otro: revisa su estado y sus registros, o pide la instrucción de cierre correspondiente. Si no aparece ninguno, inicia el contenedor de prueba con la referencia fijada para esta edición:

```
docker run --detach --name p00-diagnostico-web \
  --publish 127.0.0.1:8080:80 \
  nginx@sha256:adad2ae9204d0fd7a34f40299bc838c3782be1293b10005eac4315ff5a1abf4e
```

`docker container ls --all` muestra contenedores activos y detenidos; `--filter name=...` limita el resultado a este nombre. En `docker run`, `--detach` lo deja ejecutándose en segundo plano, `--name` le asigna un nombre reutilizable y `--publish` crea el único acceso desde el anfitrión al puerto interno de Nginx.

El significado de `127.0.0.1:8080:80` se lee de izquierda a derecha:

Parte	Significado
`127.0.0.1`	La aplicación se expone solo en el propio anfitrión.
`8080`	Puerto que usarán el navegador y `curl` en el anfitrión.
`80`	Puerto interno donde Nginx escucha dentro del contenedor.

Comprueba desde el anfitrión que el contenedor existe, está activo y publicó el puerto esperado:

```
docker ps --filter name=p00-diagnostico-web
docker port p00-diagnostico-web
docker logs p00-diagnostico-web
```

Resultado esperado: `docker ps` debe mostrar estado `Up`; `docker port` debe relacionar el puerto `80/tcp` del contenedor con `127.0.0.1:8080`; los registros no deben mostrar un cierre inesperado. Si alguno de estos pasos falla, todavía no pases a `curl`. El *digest* fija el contenido exacto de la imagen; se revisará de forma controlada antes del curso, no durante la práctica.

Ejemplo resuelto: el puerto no coincide

Imagina este resultado:

```
$ docker port p00-diagnostico-web
80/tcp -> 127.0.0.1:8081
```

La URL `http://localhost:8080` fallará aunque Nginx esté funcionando. La causa no es DNS ni el HTML: el puerto publicado es `8081`. Primero prueba `http://localhost:8081`; después decide si basta con documentar la URL correcta o si debes recrear el contenedor con el puerto requerido. Esta distinción evita reconstruir imágenes que no tienen ningún problema.

Comprobar por capas

1. Contexto y dirección

Desde el **anfitrión**, la dirección correcta es `http://localhost:8080`. `localhost` es un nombre especial que se refiere al equipo desde el que se ejecuta la orden; por tanto, dentro de `p00-lab` no significa el anfitrión, sino el propio contenedor Debian.

Para comprobar la web desde el contenedor Debian, se necesita un nombre o dirección del anfitrión proporcionado por Docker o por el docente. En Docker Desktop suele estar disponible `host.docker.internal`; no lo des por supuesto, compruébalo:

```
getent hosts host.docker.internal
```

Si no devuelve una dirección, no pruebes direcciones de la red al azar. Usa la URL local desde el anfitrión o pregunta qué nombre autorizado se debe emplear en ese equipo.

2. Puerto TCP

Hipótesis: el nombre se resuelve, pero quizá ningún servicio acepta conexiones en el puerto publicado. Desde `p00-lab`, si el nombre anterior se resolvió, prueba una sola conexión al puerto esperado:

```
nc -vz host.docker.internal 8080
```

Una conexión correcta indica que hay un servicio TCP accesible en ese destino y puerto. Un mensaje de «connection refused» suele significar que el host es alcanzable pero nada escucha en ese puerto; revisa `docker ps`, el mapeo `--publish` y los registros. Un tiempo de espera puede indicar que el destino elegido no es correcto o que una regla de red interviene.

`ping` no sustituye esta prueba: usa ICMP, mientras que una aplicación web necesita TCP y HTTP. Que un equipo responda a `ping` no demuestra que su puerto 8080 ni su web funcionen.

3. Respuesta HTTP y cabeceras

HTTP añade significado a la conexión: método, ruta, cabeceras, código de estado y, en muchas peticiones, un cuerpo. Una respuesta típica empieza por una línea como `HTTP/1.1 200 OK`; el código y la descripción indican cómo ha resultado la petición.

Desde el anfitrión, solicita solo las cabeceras de la página inicial:

```
curl --head --show-error http://localhost:8080/
```

`--head` pide las cabeceras y `--show-error` muestra el motivo si `curl` no puede completar la petición. Para ver cabeceras y cuerpo en una sola salida durante una práctica controlada, usa:

```
curl --include --show-error http://localhost:8080/ | head -n 30
```

La opción `--include` muestra la línea de estado y las cabeceras antes del HTML. Limitar la salida con head evita añadir páginas completas al producto que guardarás.

Resultado	Interpretación inicial	Siguiente comprobación
`200 OK`	El servidor respondió correctamente a esa ruta.	Comprueba que el contenido identifica tu práctica.
`301` o `302`	El servidor pide ir a otra URL.	Mira la cabecera `Location`; no supongas que es un fallo.
`404 Not Found`	Hay servidor HTTP, pero la ruta no existe.	Revisa la ruta solicitada y los archivos copiados o montados.
`403 Forbidden`	El servidor entendió la petición, pero no permite acceder.	Revisa la configuración y permisos sin exponerlos.
`500`–`599`	El servidor o una dependencia no completó correctamente la petición.	Revisa registros y estado del servicio.
Error de conexión de `curl`	No se llegó a intercambiar HTTP.	Vuelve a contenedor, puerto y dirección.

Para un resultado más breve y fácil de comparar, guarda solo el código HTTP y el tiempo de respuesta:

```
curl --silent --show-error --output /dev/null \
  --write-out 'HTTP=%{http_code} tiempo=%{time_total}s\n' \
  http://localhost:8080/
```

No confundas una respuesta `200` con una prueba completa: confirma también que el cuerpo contiene la página que has creado y no una página por defecto inesperada.

4. Procesos, puertos y registros

Cuando una prueba HTTP falle, vuelve al anfitrión y recoge información en este orden:

```
docker ps --all --filter name=p00-diagnostico-web
docker logs --tail 30 p00-diagnostico-web
docker inspect --format '{{.State.Status}}' p00-diagnostico-web
```

- `docker ps --all` muestra también contenedores que ya se han detenido.
- `docker logs --tail 30` limita el resultado a las últimas líneas relevantes.
- `docker inspect` permite comprobar el estado exacto sin mostrar toda la configuración, que podría contener información no necesaria.

En el anfitrión Linux, `ss -ltn` puede ayudarte a ver puertos TCP en escucha. No se espera que lo uses en Windows ni que memorices su salida: el procedimiento común sigue siendo `docker ps`, `docker port`, `curl` y `docker logs`.

Resolver tres incidencias guiadas

Incidencia A: el navegador muestra «No se puede conectar»

1. Confirma que la URL es exactamente `http://localhost:8080/` y que la orden se ejecuta en el anfitrión.
2. Ejecuta `docker ps --filter name=p00-diagnostico-web`.
3. Si no aparece como `Up`, consulta `docker ps --all` y `docker logs --tail 30 p00-diagnostico-web`.
4. Si está activo, ejecuta `docker port p00-diagnostico-web` y comprueba el mapeo `127.0.0.1:8080`.
5. Ejecuta `curl --head --show-error http://localhost:8080/` y registra el resultado.

La causa puede ser una URL mal escrita, un contenedor detenido o un puerto no publicado; el informe debe mostrar qué prueba descartó cada hipótesis.

Incidencia B: `curl` devuelve `404 Not Found`

Un `404` demuestra que la red, el puerto y el servidor HTTP responden. No pierdas tiempo reinstalando Docker. En la web de P00, revisa primero que `web/index.html` existe y que el `Dockerfile` copia o el volumen monta la carpeta correcta. Después reconstruye o reinicia según el modo de ejecución usado y repite la petición.

Incidencia C: desde Debian no se alcanza el anfitrión

Si `curl http://localhost:8080/` se ejecuta dentro de `p00-lab`, está intentando conectar con el propio Debian, no con el anfitrión. Comprueba `host.docker.internal` con `getent hosts`; si no está disponible, documenta esa limitación y realiza la prueba equivalente desde el anfitrión. No elimines el aislamiento de Docker ni montes su socket para «arreglarlo».

Registrar un producto que sirva para aprender

Guarda en `evidencias/rf03-diagnostico.md` un resumen, no un volcado de terminal. Puedes seguir esta plantilla:

```
# Diagnóstico de la web local

## Hipótesis inicial

La página no responde en `http://localhost:8080/`.

## Pruebas realizadas

| Pregunta | Orden | Resultado resumido | Conclusión |
| --- | --- | --- | --- |
| ¿Está activo el contenedor? | `docker ps ...` | `Up` | La aplicación no se ha
 ↪  detenido. |
| ¿Está publicado el puerto? | `docker port ...` | `127.0.0.1:8080` | El mapeo es el
 ↪  previsto. |
| ¿Qué devuelve HTTP? | `curl --head ...` | `HTTP/1.1 200 OK` | La página inicial
 ↪  responde. |

## Causa y corrección

La URL inicial no incluía el puerto `8080`; se corrigió y se volvió a comprobar con
 ↪  `curl`.
```

Al terminar la prueba temporal, el anfitrión debe liberar el recurso:

```
docker stop p00-diagnostico-web
docker rm p00-diagnostico-web
```

La página siguiente convertirá la web del proyecto en una imagen propia y sustituirá estas órdenes aisladas por una configuración `compose.yaml` reproducible.

Fuentes para ampliar

- Mensajes HTTP de MDN: estructura de peticiones, respuestas, línea de estado y cabeceras.
- Ejemplo de cabeceras HTTP con `curl`: uso de `--dump-header` para inspección puntual.
- Documentación de `docker compose up`: referencia de creación, arranque y reconstrucción de servicios.

Del archivo HTML a una aplicación reproducible

En Git creaste la carpeta `web/` y una página mínima. Ahora la usaremos como carga de prueba: no se evalúa el diseño, sino que el mismo contenido pueda empaquetarse, ejecutarse, comprobarse y cerrarse de forma repetible.

Hay tres piezas distintas. Conviene distinguirlas antes de escribir código:

Pieza	Responde a	Archivo o recurso
Contenido	«¿Qué verá el navegador?»	`web/index.html`
Imagen	«¿Cómo se prepara el servidor web?»	`Dockerfile` y una imagen como `p00-web:local`
Aplicación local	«¿Cómo se ejecuta y configura este servicio?»	`compose.yaml` y el contenedor creado por Compose

El contenido y los archivos de configuración se guardan en `/optativa-nube/p00-puesto-nube`; por ello podrán versionarse con Git. Las órdenes Docker se ejecutan siempre desde el **anfitrión**, nunca dentro de `p00-lab`.

Un cambio, tres lugares distintos

Supón que cambias el título de `web/index.html`.

1. El cambio ya existe en la carpeta compartida y Git puede mostrarlo con `git diff` desde `p00-lab`.

2. La imagen `p00-web:local` **no** cambia por sí sola: es una fotografía creada en un momento anterior.
3. El contenedor en ejecución solo mostrará el cambio si monta la carpeta como volumen o si construyes una imagen nueva y lo recreas.

Esta cadena será importante para diagnosticar la frase «he cambiado el HTML pero el navegador muestra lo antiguo»: antes de editar otra vez, identifica si estás viendo un volumen, una imagen anterior o un contenedor que no se ha recreado.

1. Preparar una web pequeña pero identificable

Punto de partida: estás en el anfitrión. Entra en `p00-lab` y abre con tu editor `/optativa-nube/p00-puesto-nube/web/index.html`:

```
docker start -ai p00-lab
su - alumno
```

Sustituye o completa su contenido con esta página de prueba:

```
<!doctype html>
<html lang="es">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <title>P00 · Comprobación local</title>
    <style>
      body { max-width: 48rem; margin: 3rem auto; padding: 0 1rem; font-family:
↪   system-ui, sans-serif; }
      main { border: 2px solid #2563eb; border-radius: 0.75rem; padding: 1.5rem; }
      h1 { color: #1e3a8a; }
      code { background: #eff6ff; padding: 0.1rem 0.25rem; }
    </style>
  </head>
  <body>
    <main>
      <h1>Comprobación local correcta</h1>
      <p>La página se sirve desde el entorno de práctica P00.</p>
      <p>Puerto publicado: <code>127.0.0.1:8080</code>.</p>
    </main>
  </body>
</html>
```

El estilo está incluido en el propio archivo para que la comprobación no dependa de una red, de Node ni de una biblioteca externa. No se evalúa diseño web: basta con que la página sea

identificable y su HTML sea correcto. Tailwind podrá emplearse posteriormente solo como mejora visual opcional si el docente confirma el método y las herramientas de aula.

Antes de continuar, comprueba desde el contenedor que el archivo existe y tiene un encabezado HTML reconocible:

```
cd /optativa-nube/p00-puesto-nube
ls -l web/index.html
head -n 12 web/index.html
```

Después revisa el cambio con `git status` y `git diff`, pero todavía no hagas *commit*: incorporarás la página, el Dockerfile y Compose tras comprobar el ciclo completo. El HTML, el Dockerfile y `compose.yaml` se versionan; una imagen Docker, los contenedores y los registros no.

2. Convertir el contenido en una imagen

Sal de `p00-lab` con dos órdenes: la primera sale de `alumno` y la segunda vuelve al anfitrión. Desde la terminal del **anfitrión**, sitúate en la carpeta compartida del proyecto y crea un archivo llamado `Dockerfile`:

```
exit
exit
```

```
FROM nginx@sha256:adad2ae9204d0fd7a34f40299bc838c3782be1293b10005eac4315ff5a1abf4e
COPY web/ /usr/share/nginx/html/
```

La primera instrucción parte de una versión fijada de la imagen oficial de Nginx. La segunda copia los archivos de `web/` al directorio que Nginx sirve por defecto. La imagen es una fotografía: si después modificas `web/index.html`, la imagen existente no cambiará hasta que vuelvas a construirla. El *digest* se revisa de forma controlada antes del curso; no lo sustituyas por una etiqueta mutable durante la práctica.

Antes de construir, crea `.dockerignore` en la raíz de `p00-puesto-nube`. Este archivo limita qué ficheros se envían como contexto de construcción; no reemplaza `.gitignore`, porque cada uno protege un momento distinto del flujo.

```
.git/
.env
.env.*
*.pem
*.key
*.log
node_modules/
```

Punto de partida: editas la carpeta compartida `p00-puesto-nube`; el archivo se versionará junto al Dockerfile. **Qué cambia:** esos patrones no viajarán al contexto de `docker build`. **Cómo comprobarlo:** revísalo con `git diff` y confirma que solo excluye archivos no necesarios para la web. **No hagas:** no incluyas el directorio `web/` ni supongas que `.dockerignore` arregla un secreto que ya se hubiera incluido en un *commit*.

Punto de partida: estás en el terminal del anfitrión y existen `Dockerfile` y `web/index.html` dentro de `p00-puesto-nube`. Entra en esa carpeta según tu sistema:

```
cd "$HOME/optativa-nube/p00-puesto-nube"
```

```
Set-Location "$HOME\optativa-nube\p00-puesto-nube"
```

Después construye una imagen con una etiqueta local y comprueba que existe:

```
docker build --tag p00-web:local .
docker image ls p00-web
```

Si creaste `optativa-nube` en el Escritorio, usa `Desktop/optativa-nube` en Bash o `Desktop\optativa-nube` en PowerShell. Estas órdenes se ejecutan en el anfitrión; no uses la ruta interna `/optativa-nube` ahí.

`docker build --tag p00-web:local .` usa el `Dockerfile` de la carpeta actual; `--tag` da a la imagen un nombre y una etiqueta reconocibles; el punto final indica que esa carpeta es el contexto de construcción. `docker image ls p00-web` lista las imágenes cuyo nombre coincide. **Qué cambia:** se crea o actualiza la imagen local; todavía no hay ningún contenedor web en ejecución. **Verificación:** la lista debe mostrar `p00-web` con la etiqueta `local`. Si la construcción falla en `COPY web/`, revisa que estás en la raíz del proyecto y que existe `web/index.html`. **No hagas:** no ejecutes estas órdenes dentro de `p00-lab` ni copies al Dockerfile archivos distintos de la web estática.



> Una etiqueta como `p00-web:local` identifica el propósito de la imagen. Más adelante usarás etiquetas de versión cuando convenga distinguir una construcción estable de otra en pruebas.

3. Ejecutar, comprobar y cerrar un contenedor

Punto de partida: `p00-web:local` aparece en `docker image ls`. Antes de crear el servidor, comprueba que el nombre de práctica está libre:

```
docker container ls --all --filter name=p00-web
```

Si aparece un contenedor `p00-web`, no ejecutes otra creación con el mismo nombre: revisa su estado o ciérralo con la secuencia de cierre de este apartado. Cuando el nombre esté libre, inicia el servidor desde el anfitrión y publica su puerto únicamente en la interfaz local:

```
docker run --detach --name p00-web \
  --publish 127.0.0.1:8080:80 \
  p00-web:local
```

Qué cambia: se crea y arranca el contenedor `p00-web`; el puerto 80 interno queda accesible solo como `127.0.0.1:8080` en el anfitrión. Ahora comprueba el estado, la publicación y la respuesta. Cada comando responde a una pregunta diferente:

```
docker ps --filter name=p00-web
docker port p00-web
curl --include --show-error http://localhost:8080/ | head -n 30
docker logs --tail 30 p00-web
```

Comprobación	Resultado esperado
`docker ps`	El contenedor aparece como `Up`.
`docker port`	Muestra `80/tcp -> 127.0.0.1:8080`.
`curl`	Devuelve una línea `200` y el HTML que contiene «Comprobación local correcta».
`docker logs`	No muestra un cierre ni errores inesperados.

Abre también `http://localhost:8080/` en el navegador del anfitrión. Navegador y `curl` son dos comprobaciones complementarias: el primero verifica la experiencia visible y el segundo permite leer la respuesta HTTP de forma precisa.

Cómo comprobarlo: los cuatro comandos anteriores deben confirmar estado `Up`, publicación exclusiva en `127.0.0.1`, estado HTTP `200` y registros sin un cierre inesperado. **No hagas:** no reemplaces la dirección de publicación por `8080:80`, no compartas el puerto con la red del aula y no crees un contenedor desde dentro de `p00-lab`.

Punto de partida para el cierre: la respuesta HTTP ya se ha registrado. Cuando termines esta prueba con `docker run`, libera el nombre y el puerto:

```
docker stop p00-web
docker rm p00-web
```



No cambies `127.0.0.1:8080:80` por `8080:80` sin comprenderlo. El segundo formato puede publicar el servicio en todas las interfaces del anfitrión; la práctica debe ser accesible solo desde el propio equipo.

4. Por qué Compose mejora la repetición

`docker run` es útil para comprender las opciones de un único contenedor, pero empieza a ser frágil cuando hay varias opciones o servicios. Docker Compose guarda la configuración declarativa de una aplicación en `compose.yaml`. Así el puerto, el montaje y el proceso de arranque se revisan como un archivo, se versionan con Git y se repiten con órdenes cortas.

Compose no es «Docker dentro de Docker»: es una herramienta que se ejecuta en el anfitrión y crea contenedores Docker a partir del archivo de configuración.

Red interna: comunicación no es lo mismo que publicar un puerto

Cada contenedor tiene una interfaz de red. En una red *bridge* definida por el usuario, los contenedores conectados pueden encontrarse por el nombre del contenedor o del servicio; esa comunicación ocurre dentro de Docker. `ports`, en cambio, crea una puerta desde el anfitrión hacia un puerto del contenedor.

En esta primera web hay un único servicio, así que no necesitamos añadir un backend ni abrir más puertos. Compose creará una red propia para el proyecto y conectará `web` a ella. Más adelante, si hubiera otro servicio aprobado, podría usar el nombre `web` dentro de esa red; desde el navegador seguiría siendo necesario un puerto publicado si se quisiera acceder desde el anfitrión.

```
Navegador del anfitrión
        │ http://127.0.0.1:8080
        ▼
Puerto publicado (solo anfitrión)
        │ 80/tcp
        ▼
Contenedor web ──── red interna de Compose ──── futuros servicios autorizados
```

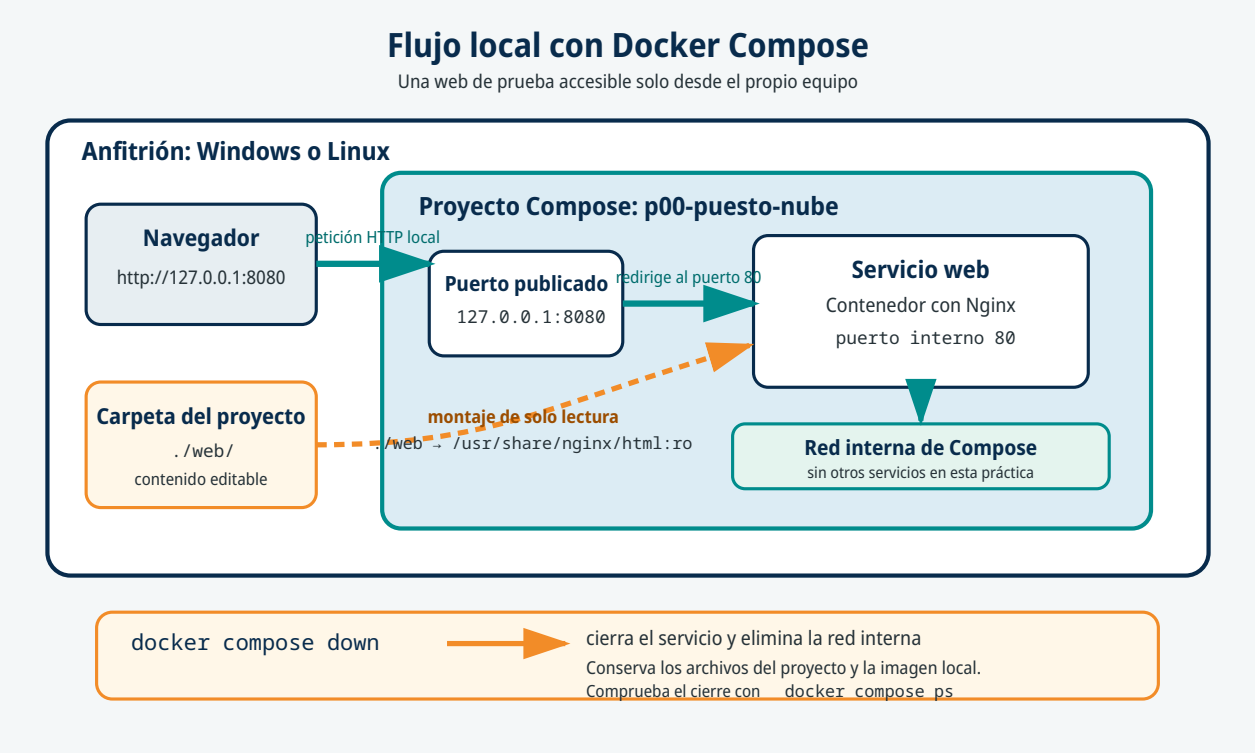


Figura 5: Flujo local de Compose: el navegador accede a 127.0.0.1:8080; Compose administra el contenedor web, la red interna y un montaje de solo lectura, con cierre mediante docker compose down.

No hagas: no uses `network_mode: host`, no cambies a una red compartida ni publiques puertos adicionales para una web estática. La red interna prepara el modelo mental para servicios posteriores; no es una invitación a crear una arquitectura compleja en P00.

Crear `compose.yaml`

En la raíz de `p00-puesto-nube`, crea este archivo desde el anfitrión o desde el editor sobre la carpeta compartida:

```yaml
name: p00-puesto-nube

services:
  web:
    build:
      context: .
      dockerfile: Dockerfile
    image: p00-web:local
    ports:
      - "127.0.0.1:8080:80"
```

```
    volumes:
      - ./web:/usr/share/nginx/html:ro
    environment:
      P00_ENTORNO: local
```

Clave	Significado en esta práctica
`services.web`	Declara un servicio llamado `web`.
`build.context: .`	Construye usando la carpeta actual como contexto.
`image`	Asigna un nombre claro a la imagen resultante.
`ports`	Publica el puerto del contenedor solo en el anfitrión local.
`volumes`	Hace visible `web/` en Nginx durante la práctica.
`:ro`	El contenedor puede leer el contenido, pero no modificarlo.
`environment`	Demuestra una variable no sensible; no es un lugar para secretos.

El volumen tiene un efecto didáctico importante: al editar `web/index.html`, Nginx ve el cambio sin reconstruir la imagen. Eso es útil durante el desarrollo, pero no cambia la imagen `p00-web:local`; para una entrega reproducible deberás reconstruirla y comprobar que funciona también sin ese montaje.

Punto de partida: `compose.yaml` está en la raíz del proyecto y aún no has creado servicios con Compose. **Qué cambiará:** al iniciar, Compose creará el servicio `web` y una red interna del proyecto. **Cómo comprobarlo:** usa primero `docker compose config`; no hace falta declarar una red manual para este único servicio. **No hagas:** no añadas servicios, imágenes o dependencias de aplicaciones externas; los materiales Node/Mongo de referencia están fuera del alcance de P00.

5. Ciclo de vida guiado con Compose

Punto de partida: estás en la raíz del proyecto en el anfitrión y no queda activo el contenedor creado con `docker run`. Si quedó alguno, detenlo y elimínalo antes para liberar el puerto 8080. Sigue siempre esta secuencia:

```
docker compose config
docker compose up --detach --build
docker compose ps
docker compose logs --tail 30
curl --include --show-error http://localhost:8080/ | head -n 30
```

1. `docker compose config` valida y muestra la configuración que Compose interpretará. Corrige un error de YAML aquí, antes de crear recursos.
2. `docker compose up --detach --build` construye la imagen si hace falta, crea el servicio y la red interna y los deja ejecutándose en segundo plano.
3. `docker compose ps` indica el nombre, estado y puertos del servicio.
4. `docker compose logs --tail 30` ofrece las últimas líneas para diagnosticar sin volcar registros completos.
5. `curl` cierra el ciclo: comprueba que el servicio publicado responde HTTP.

Qué cambia: aparecen el contenedor administrado por Compose y una red interna; la imagen puede reconstruirse si el Dockerfile o su contexto han cambiado. `docker compose ps` debe mostrar el servicio web activo y el puerto publicado. No necesitas conocer ni usar la dirección IP interna de la red: para la web local la dirección estable es `http://localhost:8080/`.

Modifica una frase de `web/index.html` y guarda el archivo. Como existe el volumen de solo lectura, actualiza el navegador y repite `curl`: el cambio debe aparecer sin reconstrucción. Documenta en la bitácora que este comportamiento procede del volumen, no de que Docker haya modificado la imagen.

Punto de partida para el cierre: has terminado la comprobación y has guardado un resultado resumido. Cuando termines, detén y elimina los recursos que Compose creó:

```
docker compose down
docker compose ps
```

El segundo comando debe indicar que no quedan servicios del proyecto en ejecución. Tras `down`, Compose elimina el contenedor y su red propia; la imagen local puede mantenerse para una prueba posterior. **No hagas:** no uses `docker system prune`, `docker network prune` ni limpieza global: podrían afectar a recursos que no pertenecen a P00.

6. De una incidencia a una corrección razonada

Síntoma	Primera comprobación	Corrección probable
`docker compose config` falla	Lee línea y sangría indicadas.	Corrige YAML; no añadas tabuladores ni claves inventadas.

Síntoma	Primera comprobación	Corrección probable
El servicio no queda Up	`docker compose ps` y `docker compose logs --tail 30`.	Revisa el Dockerfile o el error mostrado por Nginx.
El navegador no conecta	`docker compose ps`, después `curl`.	Revisa que el puerto sea `127.0.0.1:8080:80` y que no esté ocupado.
Aparece 404	Comprueba `web/index.html` y el destino del volumen.	Corrige la ruta `./web:/usr/share/nginx/html:ro`.
El HTML no cambia	Confirma que editas la carpeta compartida y que el volumen está presente.	Guarda el archivo correcto; no reconstruyas sin entender por qué.
Aparece un secreto en un archivo	Detén la entrega y retíralo antes de hacer *commit*.	Usa valores no sensibles y revisa `.gitignore`.

Una corrección se considera completa cuando se verifica con la misma prueba que reveló el problema. Por ejemplo, tras corregir un puerto, repite `docker compose ps` y `curl`; no basta con que Compose deje de mostrar un error.

7. A entregar y cierre responsable

Con tu editor, crea `evidencias/web-local.md` con la URL usada, una respuesta HTTP resumida, el resultado de `docker compose ps` y la confirmación de cierre. No copies registros completos. Después vuelve a `p00-lab` para revisar exactamente los archivos que guardarás:

```
docker start -ai p00-lab
su - alumno
cd /optativa-nube/p00-puesto-nube
git status
git diff
git add Dockerfile .dockerignore compose.yaml web/index.html evidencias/web-local.md
git diff --staged
git commit -m "Publica la web de comprobación con Compose"
exit
exit
```

No añadas imágenes, contenedores, registros extensos, `.env`, claves privadas ni credenciales. El producto contiene la configuración, una comprobación HTTP resumida y el cierre realizado. El primer `exit` sale de `alumno` y el segundo vuelve al anfitrión antes de ejecutar el cierre.

Termina la sesión deteniendo el servicio Compose y el contenedor de trabajo:

```
docker compose down
docker stop p00-lab
```

La página de diagnóstico será el procedimiento de referencia cuando esta web o servicios posteriores no respondan.

Fuentes para ampliar

- Cómo funciona Docker Compose: servicios, redes, volúmenes y ciclo de vida de Compose.
- Referencia de `docker compose up`: creación, arranque, reconstrucción y modo desacoplado.
- Guía rápida de Docker Compose: ejemplo progresivo de archivo `compose.yaml`, registros y cierre.

Protocolo de AWS Academy Learner Lab

El trabajo responsable empieza antes de crear un recurso. En Learner Lab, cada activación comienza con **Start Lab** y dura hasta cuatro horas; al finalizar el tiempo o usar **End Lab**, los recursos se detienen pero permanecen creados. Algunos recursos detenidos pueden seguir generando coste.



El crédito disponible es de 50 USD por estudiante. Antes, durante y después de una práctica se debe revisar qué recursos existen y si pueden tener coste.

Para cada práctica en AWS se seguirá este protocolo:

1. **Antes de iniciar:** anota los recursos ya existentes, comprueba la región prevista y define qué crearás, cómo lo verificarás y cómo lo eliminarás.
2. **Durante la práctica:** usa únicamente `us-east-1` o `us-west-2`; cuando un servicio requiera rol, utiliza `LabRole`. No crees roles IAM propios, no uses AMI de Marketplace y no inicies veinte o más instancias simultáneas.
3. **Al comprobar:** registra el recurso creado, la prueba funcional realizada y cualquier coste o limitación observada. No expongas direcciones, credenciales ni ficheros privados en los productos que guardes.
4. **Al cerrar:** detén y elimina los recursos creados cuando corresponda; vuelve a inventariar para confirmar que no queda nada pendiente. No confundas «detenido» con «sin coste».

Plantilla de bitácora para cualquier práctica

Cada producto AWS puede empezar con estas cinco líneas. Completarlas antes de una captura evita que quede sin contexto:

```
## Objetivo

## Región y límites comprobados

## Recursos y cambios realizados

## Prueba funcional y resultado

## Limpieza y comprobación final
```

No incluyas identificadores personales, credenciales, tokens ni datos de otras personas. Si una acción queda denegada, anota el mensaje resumido, su causa probable y la comprobación alternativa utilizada.

AWS CLI no es un requisito de P00. Cualquier automatización local será una extensión opcional de DAM tras validar credenciales temporales y permisos.

Ejercicios cortos de refuerzo

Realiza estas actividades individualmente y guarda los productos solicitados en el repositorio como Markdown. No incluyas claves, *tokens*, contraseñas ni datos personales.

RF00. Ciclo de vida de `p00-lab`

Desde el anfitrión, crea el contenedor Debian, entra en él y crea `p00-puesto-nube/evidencias/rf00-contenedor.md` dentro de `/optativa-nube`. Sal y vuelve a iniciarlo. Demuestra que el fichero persiste y que el contenedor queda detenido, no eliminado, al usar `exit`.

- **RA0/CE:** a, d, g.
- **A entregar:** `p00-puesto-nube/evidencias/rf00-contenedor.md`, con imagen usada, nombre del contenedor, carpeta personal de `alumno` y comprobación de `optativa-nube/`.
- **Comprueba antes de entregar:** diferencia imagen, contenedor, volumen y carpeta compartida; no uses `docker commit` ni publiques información sensible.

RF01. Bitácora y script de Shell

Construye `/optativa-nube/p00-puesto-nube` y documenta una operación de navegación, creación de estructura y consulta de información del sistema desde Bash dentro de `p00-lab`. Después crea `scripts/registrar-entorno.sh`, valida su sintaxis y ejecútalo para generar un registro mínimo.

1. Demuestra una ruta absoluta y una ruta relativa hacia `evidencias/`.
2. Crea o actualiza `evidencias/rf01-shell.md` usando `>` una sola vez y `>>` para añadir información sin sobrescribirla.
3. Ejecuta `bash -n scripts/registrar-entorno.sh`, comprueba el código de salida y ejecuta el script con Bash.
4. Revisa el archivo de inventario creado, explicando qué información se ha incluido y qué datos sensibles se han excluido.

- **RA0/CE:** a.
- **A entregar:** `evidencias/rf01-shell.md`, `scripts/registrar-entorno.sh` y el inventario generado por el script.
- **Comprueba antes de entregar:** rutas y redirecciones correctas, script validado, variables protegidas con comillas y salida interpretada; todo permanece en `optativa-nube/`.

RF02. Detective de secretos

Clasifica una lista de archivos de un proyecto y decide cuáles se versionan, cuáles se excluyen y por qué. Añade las reglas razonadas al `.gitignore` del repositorio de práctica.

Clasifica esta lista sin crear ni copiar ningún secreto real: `README.md`, `web/index.html`, `.env`, `clave-practica.pem`, `evidencias/arranque.log`, `compose.yaml` y `Dockerfile`. Después, revisa el resultado con `git status` antes de preparar el cambio.

- **RA0/CE:** b, g.
- **A entregar:** `.gitignore` y `evidencias/rf02-secretos.md`.
- **Comprueba antes de entregar:** detecta claves privadas, ficheros de entorno y artefactos locales; justifica las exclusiones sin publicar valores sensibles.

RF03. Diagnóstico guiado

Interpreta resultados de comprobaciones autorizadas de DNS, HTTP y un puerto local para determinar si una incidencia está en la resolución de nombre, el servicio o la conectividad.

Realiza las pruebas únicamente sobre `localhost:8080` o sobre un servicio autorizado. El producto debe separar como mínimo: estado del contenedor, publicación del puerto, respuesta HTTP y registros. Si pruebas desde `p00-lab`, explica qué significa allí `localhost` y verifica antes si `host.docker.internal` está disponible.

Para cada prueba, registra: pregunta, comando usado, resultado resumido y conclusión. No ejecutes escaneos de rangos de red ni utilices herramientas contra equipos ajenos.

- **RA0/CE:** f.
- **A entregar:** `evidencias/rf03-diagnostico.md`.
- **Comprueba antes de entregar:** hipótesis razonada, pruebas mínimas y conclusión sustentada; no escanees equipos o servicios no autorizados.

RF04. SSH seguro

Genera un par de claves Ed25519 de práctica en `~/.ssh` del usuario `alumno`, no en el repositorio. Comprueba los permisos de la carpeta y de ambos archivos, identifica cuál puede compartirse y revisa su huella pública. Después, analiza un caso de primera conexión con una huella de host proporcionada por el docente.

El caso de host lo facilitará el docente y no contendrá una clave funcional. Debes indicar qué dato se puede compartir (clave pública), cuál no (clave privada), qué papel tiene `known_hosts` y qué comprobación falta antes de aceptar una nueva huella de host. La conexión real solo se realiza si el docente facilita servidor, usuario y huella por un canal fiable.

- **RA0/CE:** c.
- **A entregar:** `evidencias/rf04-ssh-seguro.md`, con permisos, huella pública y decisión de verificación; nunca la clave privada ni su contenido.
- **Comprueba antes de entregar:** protege la clave privada, reconoce la diferencia entre host y usuario, usa `known_hosts` correctamente y no elimina avisos de identidad sin verificación.

RF05. Plan de trabajo responsable en Learner Lab

Antes de cualquier práctica de AWS, prepara un checklist personal para una sesión de Learner Lab. No se crean recursos AWS en esta actividad.

1. Anota región prevista (`us-east-1` o `us-west-2`), objetivo, recursos que se prevé crear y comprobación funcional esperada.
2. Define cómo consultarás los recursos existentes y el coste antes, durante y después de la práctica.
3. Incluye la parada, eliminación y segundo inventario de cierre; indica que se usará `LabRole` cuando un servicio requiera un rol.
4. Explica por qué no se crean roles propios, no se usan AMI de Marketplace ni se inician veinte o más instancias simultáneas.

- **RA0/CE:** g.
- **A entregar:** `evidencias/rf05-learner-lab.md`.
- **Comprueba antes de entregar:** plan concreto, límites del laboratorio correctamente aplicados y cierre verificable; no incluyas credenciales ni capturas con datos privados.

Ejemplo guiado / prácticas

Las prácticas son hitos del mismo repositorio. Se desarrollan de forma individual; una pareja puede realizar revisión cruzada, pero cada estudiante entrega sus propios productos y explicación.

EG00. Entorno Debian reproducible

Crea `p00-lab` desde la imagen oficial de Debian, usa `apt` para instalar una herramienta indicada y documenta cómo el Dockerfile incorpora las herramientas que pasan a ser recurrentes.

La práctica debe separar los cambios temporales del contenedor de la receta versionada. Conserva el trabajo en `optativa-nube/`, detén el contenedor al acabar y no utilices `docker commit`.

- **RA0/CE:** a, d, g.
- **A entregar:** una nota breve en `evidencias/eg00-entorno.md` que distinga instalación puntual e imagen versionada.
- **Comprueba antes de entregar:** el contenedor se puede detener y reiniciar; `optativa-nube/` persiste; la función de `apt` y del Dockerfile se explica correctamente.

EG01. Repositorio seguro con web estática y productos de práctica

Inicializa la estructura de un repositorio de práctica, añade una página estática mínima y documenta los resultados. Aplica el criterio de exclusión de secretos antes de crear el historial.

1. Crea la estructura `web/` y `evidencias/` dentro de `/optativa-nube/p00-puesto-nube`.
2. Añade `README.md` y `.gitignore`, inicializa Git y crea un primer *commit* tras revisar con `git status` y `git diff --staged`.
3. Añade `web/index.html` con una presentación mínima autocontenida y una bitácora breve; revísalas y crea un segundo *commit* con un mensaje que explique la mejora.
4. Incluye en el README el entorno utilizado y cómo se comprobará la página; no incluyas datos personales ni rutas locales completas.
5. No configures ni publiques un remoto hasta que el docente haya confirmado plataforma, URL y flujo de entrega.

- **RA0/CE:** b, g.
- **A entregar:** repositorio con historial, `.gitignore`, página mínima y bitácora sin datos sensibles.
- **Comprueba antes de entregar:** el historial contiene al menos dos cambios explicables, el `.gitignore` es coherente y la web es identificable.

EG02. Web estática en Docker

Empaqueta la página de P00 en una imagen y ejecútala en un contenedor local, publicando únicamente el puerto necesario y registrando la comprobación HTTP y los registros relevantes.

Usa el `Dockerfile` de la página de web y Compose o una variante funcional justificada. Antes de abrir el navegador, comprueba estado con `docker ps`, publicación con `docker port` y respuesta con `curl`. Conserva un producto resumido de construcción, URL local, respuesta HTTP, registro relevante y parada/eliminación del contenedor.

- **RA0/CE:** d, e, f.
- **A entregar:** `evidencias/web-local.md` con construcción, URL local, respuesta HTTP resumida y cierre.
- **Comprueba antes de entregar:** la página responde localmente, el contenedor es identificable y documentas arranque, verificación y parada.

EG03. Web estática en Docker Compose

Describe y ejecuta la prueba web con Compose. Incluye configuración mínima de puerto, volumen o variable de entorno no sensible cuando la práctica lo requiera. Comprueba en el navegador que la página identificable se carga sin depender de servicios externos.

Parte de la configuración presentada en el desarrollo, revísala con `docker compose config` antes de iniciarla y documenta el resultado de `docker compose ps`. Comprueba la página con `curl` y explica si un cambio de `web/index.html` proviene del volumen de solo lectura o de una reconstrucción de imagen. Al finalizar, ejecuta el cierre controlado de la aplicación y confirma que no quedan servicios del proyecto activos.

- **RA0/CE:** d, e, f.
- **A entregar:** `compose.yaml` y actualización de `evidencias/web-local.md` con la comprobación de Compose y su cierre.
- **Comprueba antes de entregar:** la configuración se puede revisar, el servicio se inicia y se detiene de forma controlada, y el producto recoge solo los registros necesarios.

EG04. Evolución reproducible de `p00-lab`

Actualiza la receta versionada del entorno cuando Git, el cliente SSH y las herramientas de diagnóstico se conviertan en requisitos recurrentes. Construye la etiqueta indicada y explica cómo se reemplazaría el contenedor sin perder el contenido de `optativa-nube/`. Si se reemplaza el contenedor, genera de nuevo las claves de práctica de `~/.ssh`.

No se aplica `docker commit` ni se elimina el contenedor de trabajo durante esta práctica sin confirmar antes que los productos están en `optativa-nube/`. Una VM Debian solo se utilizará como alternativa de contingencia cuando Docker no sea viable en el equipo.

- **RA0/CE:** a, c, d, f, g.
- **A entregar:** `evidencias/eg04-evolucion-entorno.md`, Dockerfile revisado y salida resumida de la construcción autorizada.
- **Comprueba antes de entregar:** distingue paquete instalado en un contenedor de requisito incorporado al Dockerfile, identifica imagen, contenedor y carpeta compartida, y documenta un reemplazo seguro.

Ejercicios

EA01. Incidencia local

Resuelve una incidencia preparada de HTTP, DNS o puerto en un entorno local controlado. Registra la hipótesis inicial, las pruebas realizadas y la causa final.

Caso: la web debería responder en `http://localhost:8080`, pero no carga. Recibirás una de estas causas preparadas: el contenedor está detenido, el puerto publicado no es 8080 o el fichero `web/index.html` no se ha copiado en la imagen. Tu informe debe descartar las otras dos causas con pruebas autorizadas.

- **Modalidad:** individual.
- **RA0/CE:** f.
- **A entregar:** informe breve con pruebas mínimas, interpretación correcta y corrección verificada con la misma prueba que reveló el problema.

EA02. Auditoría de seguridad

Revisa una entrega de ejemplo con secretos o configuraciones inseguras y propón correcciones trazables antes de su publicación.

Caso: una entrega de ejemplo contiene un fichero `.env`, un nombre de clave privada en el historial y un servicio Docker publicado sin justificación. Enumera los riesgos, indica qué debe retirarse o corregirse y describe cómo verificarías la entrega antes de compartirla. No reproduzcas valores de secretos, aunque sean ficticios.

- **Modalidad:** individual.
- **RA0/CE:** b, c, d, g.
- **A entregar:** lista de hallazgos, riesgo asociado y corrección propuesta sin copiar ni divulgar secretos.

Proyecto o reto final

RFN01. Puesto listo

Consolida los productos previos en un repositorio seguro con una web estática que pueda ejecutarse localmente en Docker o Compose. Incluye la comprobación HTTP, la parada ordenada y el checklist de trabajo responsable para AWS Academy Learner Lab.

- **Modalidad:** individual.
- **RA0/CE:** a–g.
- **A entregar:** repositorio, README, script de Shell, `.gitignore`, producto SSH sin secretos, página estática, Dockerfile, `compose.yaml`, diagnóstico HTTP y checklist de cierre de Learner Lab.
- **Comprueba antes de entregar:** funcionalidad local, trazabilidad, ausencia de secretos, comprobaciones justificadas, separación correcta entre anfitrión y contenedor y limpieza documentada.

RFN02. Entrega reproducible

Una pareja intercambia temporalmente sus repositorios de práctica y verifica, con el README, si puede preparar y comprobar la web en un entorno autorizado. Cada estudiante conserva un producto individual de la revisión recibida y realizada.

- **Modalidad:** parejas para la revisión; productos y revisión individuales.
- **RA0/CE:** b, d, e, f, g.
- **A entregar:** incidencia o confirmación de reproducción, mejora aplicada y actualización del README si procede.
- **Comprueba antes de entregar:** instrucciones suficientes, reproducción verificable, comunicación de incidencias y correcciones justificadas.

Checklist del portafolio final

Aspecto	Comprueba que está logrado	Si todavía está en progreso	Si está pendiente
Entorno y Shell	Imagen, contenedor, volumen y carpeta compartida identificados; script validado, ejecutado y explicado.	Comprobaciones del script incompletas.	No se identifica el entorno o no hay un resultado útil.

Aspecto	Comprueba que está logrado	Si todavía está en progreso	Si está pendiente
Git y SSH seguros	Historial revisado, `.gitignore` razonado, clave privada protegida y verificación de host explicada.	Hay reglas, revisión o verificación incompletas, sin filtración de datos.	Incluye secretos, clave privada o cambios sin revisar.
Web, Docker y Compose	La web responde localmente; Dockerfile y Compose son coherentes; se documentan puertos, volumen, registros y cierre.	La web funciona, pero falta una comprobación, explicación o confirmación de cierre.	No se verifica la ejecución local o quedan recursos sin gestionar.
Diagnóstico y documentación	Las pruebas siguen la secuencia contenedor, puerto, HTTP y registros; el README permite repetir la práctica.	Hay pruebas, pero no sostienen claramente la conclusión o el README es parcial.	Las pruebas son indiscriminadas o no permiten reproducir el trabajo.
Trabajo responsable en AWS	Checklist completo de inventario, coste, límites, cierre y limpieza.	Checklist parcial, pero reconoce las restricciones principales.	Ignora coste, regiones, `LabRole` o limpieza.

Fuentes de consulta

- Manual de referencia de GNU Bash.
- Documentación oficial de Git y referencia de `.gitignore`.
- Manual de OpenSSH `ssh` y OpenSSH for Windows.
- Docker Get Started y Docker Compose.
- Imagen oficial de Debian, usando una etiqueta o *digest* validado antes del curso.
- Referencia de `docker run` y Docker Compose.
- Manual de `apt-get` y documentación de los paquetes instalados durante P00.
- Fuentes y decisiones locales: `RA.md`, `CONTENIDO.md`, `TEMARIO.md`, `AGENTS.md` y `DOCUMENTACION/P00/FUENTES_PENDIENTES.md`.