

UNIVERSIDAD POLITÉCNICA DE MADRID  
ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE  
TELECOMUNICACIÓN



Grado en Ingeniería Biomédica  
Arquitectura de Computadores y Sistemas Operativos  
Curso 2024-2025

Práctica del primer parcial  
Desarrollo de un escenario de un sistema de servidor

Fecha del documento: 23 marzo 2025

# Índice

	<b>Página</b>
<b>1. La entrega de la práctica</b>	<b>3</b>
<b>2. Evaluación</b>	<b>3</b>
<b>3. Objetivos</b>	<b>3</b>
<b>4. Requisitos funcionales</b>	<b>4</b>
<b>5. Partes opcionales</b>	<b>5</b>
<b>6. Recomendaciones</b>	<b>5</b>
<b>7. Normas</b>	<b>6</b>

## 1. La entrega de la práctica

Por favor, usad las siguientes directrices. En caso contrario, se puede penalizar la calificación:

- La práctica se puede entregar antes de las 23:59 del 23 de abril.
- El nombre del fichero principal de la práctica tiene que ser `pfinal1.py`.
- Si la práctica tiene varios ficheros, se debe subir un fichero comprimido incluyéndolos.
- Cada grupo tiene que subir un sólo fichero. Sólo un miembro del grupo tiene que subirlo. Los nombres de las personas del grupo se deben incluir al principio del fichero principal, como comentarios.

## 2. Evaluación

La evaluación será oral. Después del plazo de entrega, se propondrán varias sesiones para la evaluación. Los grupos seleccionarán un horario en una sesión. La evaluación se realizará en el laboratorio B123 y todos los miembros del grupo tendrán que asistir a esta actividad.

En la evaluación, se comprobará la corrección del programa desarrollado. Los alumnos deberán contestar una serie de preguntas para comprobar que han adquirido las competencias y contenidos asociados a la práctica y a los laboratorios. La calificación será individual de cada persona, dependiendo de sus respuestas.

El máximo de la calificación del enunciado es 7,5 puntos. Las partes opcionales tienen una calificación máxima de 2,5 puntos.

En la evaluación de la práctica se valorará:

- Respuesta a las preguntas sobre la creación y gestión de los contenedores, que se han incluido en los laboratorios.
- Corrección del programa desarrollado, de acuerdo a los requisitos funcionales establecidos.
- Corrección de las respuestas a las preguntas planteadas.
- Uso de *logs*: esta función es muy útil para detectar errores de ejecución del código. En concreto, es más importante en el desarrollo del código y en la depuración. Es aconsejable usar diferentes niveles para seleccionar las trazas requeridas, por ejemplo, cuando se depura o en ejecución. Los *logs* no se deben borrar cuando se completa el desarrollo.
- Calidad y estilo del código: El código se escribe pocas veces y se lee muchas. Todos los lenguajes de programación disponen de guías de estilo para ayudar al desarrollo de código más correcto y legible. <https://pep8.org/> es un enlace para *Python*. Se aconseja su uso.
- Uso de módulos: El uso de paquetes y funciones es básico para disponer un código bien estructurado y que facilita su desarrollo y legibilidad. Además, es adecuado estructurar el código y encapsular módulos relacionados en diferentes ficheros.
- Enfoque para detectar errores de ejecución: desarrollo incremental, trazas y depurador.

## 3. Objetivos

La práctica final del primer parcial consistirá en el desarrollo de un programa principal (*pfinal1.py*) que automatice la creación del escenario mostrado en la figura 1. Este escenario refleja la plataforma de una aplicación distribuida, basada en el modelo cliente/servidor. En esta práctica se deben crear las máquinas virtuales que se muestra y configurar la comunicación entre los contenedores. En la siguiente práctica se desplegará la aplicación en la plataforma mostrada. Para ello, se cargará, configurará y ejecutará la aplicación en el escenario desarrollada en este enunciado.

Los componentes de escenario son:

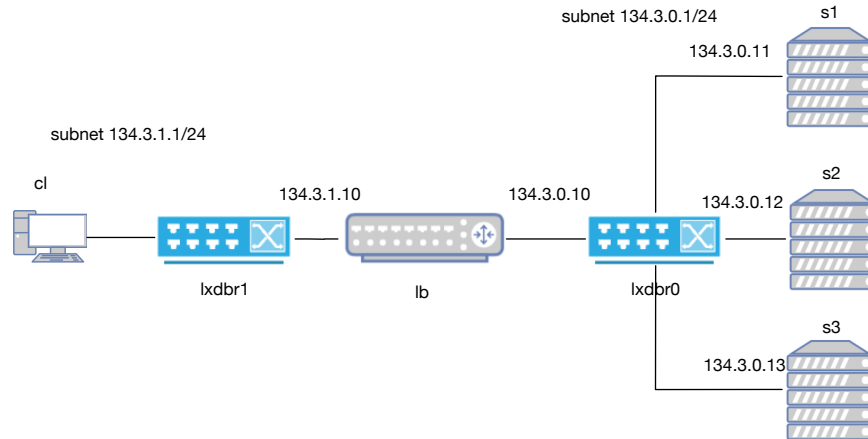


Figura 1: Escenario de la plataforma de un sistema servidor

- **Servidores:** La plataforma de este sistema dispone un conjunto de servidores distribuidos y replicados para proporcionar la funcionalidad de una aplicación. La replicación está motivada para satisfacer las peticiones de servicios de los clientes, cuando la potencia de un servidor no es suficiente. Además, esta plataforma puede tolerar fallos, de forma que si se cae un servidor, las peticiones serán procesadas por otros servidores.

Los servidores tienen que ser transparentes a los clientes. Éstos no tienen que conocer el número de servidores ni sus direcciones. Para ello, se comunicarán con una dirección IP en el balanceador de carga.

- **Balanceador de carga:** Este componente recibe peticiones de los clientes y los reenvía a un servidor. La repartición de mensajes se basa en algoritmos internos para equilibrar la carga de los servidores y contactar con servidores operativos.
- **Cliente:** Ilustra un cliente que solicite peticiones a los servidores

En esta práctica, se propone desarrollar los requisitos en dos fases:

1. Creación de las máquinas virtuales: servidores, balanceador de carga y cliente.
2. Configurar las máquinas virtuales y los componentes necesarios para permitir su comunicación.

## 4. Requisitos funcionales

El objetivo es desarrollar un programa (*pfinal1.py*) que debe recibir varios parámetros para caracterizar la plataforma:

```
python3 pfinal1.py <orden> <parámetros>
```

donde el parámetro <orden> puede tomar los valores siguientes:

- **create**, para crear las máquinas virtuales, así como las interfaces de red y los *bridges* virtuales que soportan las redes del escenario.
- **start**, para arrancar las máquinas virtuales y mostrar su consola (ver la sección 6).
- **list**, para listar la información de los contenedores existentes.
- **delete**, para liberar el escenario, borrando todas las máquinas virtuales creadas y los componentes de comunicación.

El número de servidores que se deben arrancar será configurable (de 1 a 5). Este número se deberá especificar mediante un segundo parámetro de la línea de órdenes, que será opcional: si se proporciona, se tomará el valor especificado; y si no, su valor será 2. El programa deberá comprobar que el valor del número de servidores y la orden son correctos. En caso contrario, deberá emitir un error.

El valor del número de servidores sólo se especificará en el comando **create**. Ese valor se almacenará en un fichero en el directorio de trabajo y el resto de órdenes (**start**, **list**, **delete**) accederán a este fichero.

## 5. Partes opcionales

Como funcionalidades adicionales se propone incluir:

- Funcionalidad para parar y/o arrancar servidores individualmente.
- Funcionalidad para crear y/o destruir servidores individualmente.
- Otras funcionalidades a proponer por el alumno.

## 6. Recomendaciones


1. En el proceso de desarrollar código, es habitual que se produzcan errores de ejecución. Además de los consejos en esta asignatura, es muy útil probar las sentencias que fallan en el intérprete de *Python*. Es más fácil y rápido depurar una sentencia aislada en el intérprete. Asimismo, se recomienda dividir la funcionalidad del programa en módulos y probarlos por separado. Una vez que cada parte está probada, se puede integrar el código de la prueba como una función en el programa principal.

El depurador es una herramienta muy útil para detectar y corregir errores.

2. El programa debe mostrar las consolas de las máquinas virtuales del escenario cuando éste se arranque. Para mostrarlas, se recomienda ejecutar un nuevo terminal como se indica. Esta orden se puede invocar desde *Python* como la llamada correspondiente:

```
 xterm -e "lxc exec vml bash"
```

Puede que esta orden no esté instalado en el sistema operativo. Para instalarlo ejecute:

```
 sudo apt install xterm
```

Esta instrucción hay que ejecutarse en un programa de Python. Hay dos formas de hacerlo en un programa:

- `subprocess.run`: es una llamada bloqueante o síncrona. Cuando se ejecuta esta instrucción en un programa, no se ejecuta la siguiente instrucción hasta que se haya terminado.
- `subprocess.Popen`: es una llamada desbloqueante o asíncrona. Cuando se ejecuta esta instrucción en un programa, no se para el flujo de ejecución. No se espera a que termine. Se ejecutará la siguiente instrucción inmediatamente.

Sus funcionalidades son muy similares. La decisión sobre qué instrucción hay que usar, depende del entorno del programa. Por ejemplo, para usar la creación de un terminal, es aconsejable usar `subprocess.Popen`. No es necesario esperar a que completa la operación. Se ejecutará en segundo plano, al ser asíncrona. Se usaría el siguiente código en *Python*:

```
contenedor = "s1"
orden = "lxc exec " + contenedor + " bash"
subprocess.Popen(["xterm", "-e", orden])
```

Sin embargo, sería más aconsejable usar `subprocess.run` si es necesario que haya terminado su ejecución antes de ejecutar la siguiente. Por ejemplo, un caso de uso para `subprocess.run` es cuando se inician máquinas virtuales, con `lxc init` o `launch`.

A continuación se muestra una forma de ejecutar `xterm` con las siguientes opciones (en orden): tipo de letra, tamaño de letra, color de fondo y color de las letras.

```
 xterm -fa monaco -fs 13 -bg black -fg green -e "lxc exec vm1 bash"
```

## 7. Normas

Todas las entregas y prácticas que se realicen deben ser fruto del trabajo personal del alumno o grupo, aunque se fomentará la discusión y el trabajo en grupo para ayudar a entender mejor los problemas que se intentan resolver. La copia de entregas supondrá el suspenso de la asignatura de forma automática, tanto para quien copia como para quien se deja copiar.

En concreto, se puede:

- debatir los ejercicios con otros compañeros
- ayudar a otros compañeros a depurar sus ejercicios
- usar código publicado en el sitio web de la asignatura
- usar código publicado en otros sitios citando la procedencia, siempre que no sea la propia práctica

No se puede:

- copiar las prácticas de otro grupo, ni permitir la copia de las propias
- usar código publicado sin citar el origen