PRÁCTICA 3: CONFIGURACIÓN DE SERVICIOS WEB Y BASE DE DATOS

OBJETIVO

Configurar el servidor para que sirva páginas web con Apache, aplicaciones Node.js, y bases de datos con PostgreSQL.

MATERIAL NECESARIO

- Ubuntu Server con Apache2, Node.js, y PostgreSQL ya instalados (Práctica 2 completada).

PASOS

1. Configurar Apache2 para servir un sitio web

1.1. Crear un directorio para el sitio web:

```
sudo mkdir -p /var/www/misitio
```

1.2. Dar permisos adecuados:

```
sudo chown -R $USER:$USER /var/www/misitio
```

1.3. Crear una página simple:

```
nano /var/www/misitio/index.html
```

Contenido de ejemplo:

```
<html>
  <head><title>Mi primer sitio web en Apache</title></head>
  <body><h1>¡Hola desde Apache en Ubuntu Server!</h1></body>
</html>
```

1.4. Crear un archivo de configuración para el sitio:

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

Contenido:

```
<VirtualHost *:80>
    ServerAdmin webmaster@localhost
    DocumentRoot /var/www/misitio
    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

1.5. Activar el sitio:

```
sudo a2ensite misitio.conf
```

1.6. Desactivar el sitio por defecto (opcional):

```
sudo a2dissite 000-default.conf
```

1.7. Recargar Apache:

```
sudo systemctl reload apache2
```

1.8. Verifica en el navegador que accediendo a la IP del servidor ves tu página "¡Hola desde Apache!".

---

2. Configurar un servidor básico en Node.js

2.1. Crear un directorio para el proyecto Node.js:

```
mkdir ~/miproyectonode
cd ~/miproyectonode
```

2.2. Crear el archivo del servidor:

```
nano server.js
```

Contenido:

```
const http = require('http');


const server = http.createServer((req, res) => {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('¡Hola desde Node.js en Ubuntu Server!\n');
});


server.listen(3000, () => {
  console.log('Servidor Node.js funcionando en puerto 3000');
});
```

2.3. Ejecutar el servidor:

```
node server.js
```

Deberías ver en consola:

```
"Servidor Node.js funcionando en puerto 3000"
```

2.4. Comprobar acceso:

- Desde Windows abre el navegador y accede a:

```
http://IP_DEL_SERVIDOR:3000
```

Deberías ver "¡Hola desde Node.js en Ubuntu Server!".

*(Más adelante podremos usar **PM2** para dejarlo corriendo en segundo plano.)*

3. Para dejar tu servidor Node.js corriendo en segundo plano con PM2:

1. **Instala PM2:** sudo npm install -g pm2
2. **Inicia tu aplicación:** pm2 start server.js --name "mi-servidor"

3. **Configura el inicio automático:** pm2 startup (y ejecuta el comando proporcionado), seguido de pm2 save.

Opciones útiles al iniciar tu aplicación:

- **--name <nombre>**: Asigna un nombre descriptivo a tu proceso de PM2. Esto facilita la gestión y el monitoreo.

```
pm2 start tu_aplicacion.js --name "mi-servidor-web"
```

- **-i <instancias> o -i max**: Ejecuta múltiples instancias de tu aplicación para aprovechar sistemas multi-core y mejorar la disponibilidad. Usar max iniciará tantas instancias como núcleos tenga tu CPU.

```
pm2 start tu_aplicacion.js -i max
```

- **--watch**: Permite que PM2 reinicie automáticamente tu aplicación cuando detecta cambios en los archivos del directorio de la aplicación. Esto es útil durante el desarrollo.

```
pm2 start tu_aplicacion.js --watch
```

- **--log <ruta_del_archivo>**: Especifica la ruta para el archivo de registro de la aplicación.

```
pm2 start tu_aplicacion.js --log "/var/log/mi-servidor/app.log"
```

- **--error <ruta_del_archivo>**: Especifica la ruta para el archivo de registro de errores de la aplicación.

```
pm2 start tu_aplicacion.js --error "/var/log/mi-servidor/error.log"
```

Gestión de procesos con PM2:

Una vez que tu aplicación está corriendo bajo PM2, puedes usar varios comandos para gestionarla:

- **pm2 list o pm2 ls**: Muestra una lista de todos los procesos gestionados por PM2, incluyendo su estado, ID, nombre, uso de CPU y memoria, y tiempo de actividad.

- **pm2 stop <id|nombre|all>**: Detiene uno o varios procesos. Puedes usar el ID del proceso (de la lista), el nombre que le asignaste o all para detener todos los procesos.

```
pm2 stop mi-servidor-web
```

- **pm2 restart <id|nombre|all>**: Reinicia uno o varios procesos.

```
pm2 restart mi-servidor-web
```

- **pm2 reload <id|nombre|all>**: Recarga la configuración de uno o varios procesos sin tiempo de inactividad (útil para recargar cambios en el código sin interrumpir las conexiones).

- **pm2 delete <id|nombre|all>**: Elimina un proceso de la lista de PM2 y lo detiene.

```
pm2 delete mi-servidor-web
```

- **pm2 monit**: Abre una interfaz de monitoreo en tiempo real en la terminal que muestra el estado de tus procesos, uso de recursos y registros.

- **pm2 logs <nombre> o pm2 logs --lines <n>**: Muestra los registros de un proceso específico o los últimos n líneas de todos los procesos.

Asegurar que tu servidor se inicie automáticamente al reiniciar el sistema:

Para que tu servidor Node.js se inicie automáticamente después de un reinicio del sistema, debes configurar PM2 para que se inicie con el sistema operativo. Ejecuta el siguiente comando:

```
pm2 startup
```

PM2 te proporcionará un comando específico que debes ejecutar con sudo para configurar el inicio automático. Cópialo y ejecútalo tal como se indica.

Finalmente, guarda la lista de procesos actual de PM2 para que se restauren en el próximo inicio:

```
pm2 save
```

---

4. Configurar base de datos PostgreSQL

3.1. Entrar en PostgreSQL como usuario postgres:

```
sudo -i -u postgres

psql
```

3.2. Crear una base de datos:

```
CREATE DATABASE midb;
```

3.3. Crear un usuario:

```
CREATE USER miusuario WITH PASSWORD 'miclave';
```

3.4. Dar permisos al usuario sobre la base de datos:

```
GRANT ALL PRIVILEGES ON DATABASE midb TO miusuario;
```

3.5. Salir de PostgreSQL:

```
\q

exit
```

3.6. Comprobar acceso local (opcional):

```
psql -h localhost -U miusuario -d midb
```

(te pedirá la contraseña que has asignado).

---

5. Consejos de buenas prácticas

- **Apache2**: tener configurados los permisos correctos en /var/www.
- **Node.js**: usar **PM2** o **systemd** para lanzar aplicaciones en producción.
- **PostgreSQL**: nunca usar el usuario postgres directamente en producción.

Resultado esperado

- Apache2 sirviendo un sitio web simple.
- Node.js sirviendo una aplicación básica en puerto 3000.

- PostgreSQL instalado, base de datos creada y usuario configurado.