Resumen de la Clase 01 - Configuración de Servidores y Despliegue de Aplicaciones

**1. Temas que se vieron**

La primera clase se centró en la introducción a la configuración de servidores y despliegue de aplicaciones. Los temas principales fueron:

1. **Introducción a AWS**
   * Creación de una cuenta en AWS.
   * Explicación sobre el modelo de cobro y la capa gratuita.
   * Uso de instancias EC2 para crear servidores virtuales.
2. **Conceptos de DevOps y su importancia**
   * Relación entre desarrollo y operaciones.
   * Metodologías ágiles y su impacto en DevOps.
   * Continuous Integration y Continuous Deployment (CI/CD).
3. **Infraestructura de servidores y escalabilidad**
   * Diferencia entre servidores físicos y virtuales.
   * Cómo las grandes empresas manejan la carga y el escalado.
   * Balanceadores de carga y redes de distribución de contenido (CDN).
4. **Introducción a Linux y SSH**
   * Uso de la terminal en servidores Linux.
   * Configuración de una máquina Ubuntu en AWS.
   * Conexión remota a servidores mediante SSH.
5. **Almacenamiento y gestión de datos**
   * Sistemas de bases de datos en servidores.
   * Uso de almacenamiento en la nube como S3.
   * Implementación de caché para mejorar la velocidad.

**2. Explicación a grandes rasgos de cada uno**

Aquí te hago un resumen explicativo de cada punto tratado:

* **AWS y EC2**: Amazon Web Services permite crear servidores virtuales. La capa gratuita permite usar una instancia micro por un año sin costo. EC2 es el servicio de Amazon para gestionar servidores en la nube.
* **DevOps**: Es una metodología que busca integrar desarrollo y operaciones para agilizar el proceso de entrega de software. Permite mejorar la comunicación entre equipos, automatizar procesos y desplegar software de manera continua.
* **Infraestructura de servidores**: Un servidor es una computadora que está en funcionamiento 24/7 para recibir peticiones de usuarios. Para aplicaciones grandes, se usan balanceadores de carga y CDN para distribuir el tráfico.
* **Linux y SSH**: Linux es el sistema operativo más utilizado en servidores. SSH es el protocolo de conexión remota que se usa para administrar servidores sin necesidad de interfaz gráfica.
* **Bases de datos y caché**: Para mejorar la velocidad y escalabilidad, se usan bases de datos distribuidas y almacenamiento en caché, como Redis o Memcached. Esto permite que las aplicaciones respondan más rápido y no sobrecarguen la base de datos.

**3. Proceso seguido en la clase**

El proceso general que se siguió en la clase fue:

1. **Registro en AWS y configuración de EC2**
   * Creación de una cuenta en AWS.
   * Selección de la región y tipo de instancia EC2.
   * Generación de claves de acceso para conectarse al servidor.
2. **Configuración del servidor Linux**
   * Instalación de Ubuntu en EC2.
   * Conexión al servidor usando SSH.
   * Configuración inicial y revisión del sistema operativo.
3. **Explicación de conceptos de escalabilidad**
   * Cómo funcionan los balanceadores de carga.
   * Uso de CDN para distribuir contenido estático.
   * Implementación de almacenamiento en la nube.
4. **Gestión de bases de datos y caché**
   * Cómo manejar bases de datos en servidores.
   * Uso de caché para mejorar el rendimiento.
   * Implementación de colas de tareas para procesar trabajos en segundo plano.
5. **Preparación para la siguiente clase**
   * Configuración de acceso SSH permanente.
   * Revisión de permisos y seguridad en los servidores.
   * Planificación del despliegue de aplicaciones.

**Próximos Pasos**

La próxima clase se enfocará en:

* Seguridad en servidores.
* Instalación de software necesario (Nginx, Node.js, etc.).
* Configuración de servidores web.
* Implementación de dominios y certificados SSL.

# Resumen de la Clase 02 - Configuración de Servidores y Despliegue de Aplicaciones

**1. Temas que se vieron**

En esta clase se profundizó en la configuración del servidor y el manejo de conexiones SSH. Se trabajó en:

1. **Revisión de la configuración de AWS y el servidor**
   * Repaso de la clase anterior sobre AWS y SSH.
   * Configuración para facilitar la conexión SSH.
2. **Gestión de conexiones SSH**
   * Uso del archivo ~/.ssh/config para simplificar comandos SSH.
   * Creación y gestión de accesos rápidos.
3. **Manejo de servidores en AWS**
   * Diferencias entre *Stop* y *Terminate* en AWS.
   * Cómo apagar y reiniciar servidores correctamente.
   * Cómo evitar que cambie la IP pública del servidor.
4. **Navegación y gestión de archivos en Linux**
   * Comandos básicos de navegación (ls, cd, pwd).
   * Rutas absolutas vs. relativas.
   * Creación, eliminación y permisos en archivos y directorios.
5. **Gestión de usuarios y permisos**
   * Creación de usuarios en Linux (adduser).
   * Configuración de permisos en carpetas y archivos (chmod, chown, chgrp).
   * Trabajo con grupos de usuarios.
   * Control de accesos y restricciones en Linux.
6. **Configuración del acceso SSH con autenticación por contraseña**
   * Modificación del archivo sshd\_config para permitir login con contraseña.
   * Uso de systemctl para recargar la configuración de SSH.
7. **Instalación y configuración del servidor web Nginx**
   * Instalación de Nginx en el servidor (sudo apt install nginx).
   * Verificación de su estado con systemctl status nginx.
   * Gestión del firewall en AWS para permitir conexiones HTTP (puerto 80).

**2. Explicación a grandes rasgos de cada tema**

**1. Revisión de la configuración de AWS y SSH**

Antes de continuar, se repasaron los conceptos de AWS y SSH. Se destacó la importancia de tener configuraciones de acceso eficientes y seguras.

**2. Gestión de conexiones SSH**

Para facilitar la conexión SSH sin escribir comandos largos, se creó un archivo de configuración (~/.ssh/config). Este archivo permite definir accesos rápidos con alias, reduciendo la necesidad de recordar claves e IPs manualmente.

**3. Manejo de servidores en AWS**

Se explicó la diferencia entre apagar (*Stop*) y eliminar (*Terminate*) una instancia en AWS:

* **Stop**: Apaga el servidor sin eliminar datos (pero la IP pública puede cambiar).
* **Terminate**: Elimina el servidor de forma permanente.

También se explicó cómo configurar una **IP elástica** para mantener una IP fija en AWS.

**4. Navegación y gestión de archivos en Linux**

Se revisaron comandos esenciales de Linux para moverse entre directorios y gestionar archivos. Se enfatizó la diferencia entre **rutas absolutas** y **rutas relativas**:

* **Absoluta**: Empieza desde / (ejemplo: /etc/nginx/nginx.conf).
* **Relativa**: Basada en la ubicación actual (../config).

**5. Gestión de usuarios y permisos**

Cada archivo y carpeta en Linux tiene un dueño y un grupo. Se explicó cómo gestionar usuarios y permisos para evitar accesos no autorizados.

* **chmod**: Cambia permisos de lectura, escritura y ejecución.
* **chown**: Cambia el propietario de un archivo o carpeta.
* **chgrp**: Cambia el grupo de un archivo o carpeta.

Se realizaron ejercicios creando usuarios como **Goku y Vegeta**, ajustando permisos y validando accesos.

**6. Configuración del acceso SSH con autenticación por contraseña**

Por defecto, en AWS no se permite conectarse por SSH usando una contraseña. Se modificó el archivo /etc/ssh/sshd\_config, habilitando PasswordAuthentication yes. Luego se recargó el servicio con:

bash

CopiarEditar

sudo systemctl reload ssh

Esto permitió autenticarse en el servidor con usuario y contraseña.

**7. Instalación y configuración de Nginx**

Se instaló el servidor web **Nginx**, se verificó su estado y se configuró el firewall en AWS para permitir acceso HTTP (puerto 80). Se explicó el uso de:

* systemctl para manejar servicios (start, stop, status, reload).
* nginx -t para comprobar la configuración.

**3. Proceso seguido en la clase**

El flujo de la clase siguió este orden:

1. **Repaso de la clase anterior** sobre AWS, SSH y la importancia de entender los conceptos antes de hacer las configuraciones.
2. **Configuración del acceso SSH** para hacerlo más eficiente con un archivo ~/.ssh/config.
3. **Manejo del servidor en AWS**, aprendiendo a apagar y reiniciar correctamente.
4. **Exploración del sistema de archivos en Linux**, aprendiendo navegación y permisos.
5. **Creación y gestión de usuarios y permisos**, probando accesos con diferentes cuentas.
6. **Modificación de la configuración de SSH** para habilitar autenticación por contraseña.
7. **Instalación y configuración de Nginx**, asegurando que el servidor web estuviera accesible desde internet.

**Próximos pasos**

En la siguiente clase se trabajará en:

* **Subir una web estática a Nginx**.
* **Desplegar una aplicación en React**.
* **Implementar una aplicación en Node.js en el servidor**.

**1. Instalación y Configuración del Servidor con Ubuntu en AWS**

Cuando inicias una instancia en **AWS EC2** con **Ubuntu**, AWS crea automáticamente un usuario llamado ubuntu, que tiene permisos administrativos (sudo).

🔹 **Proceso de instalación**:

1. Creas una **instancia EC2** en AWS.
2. Seleccionas **Ubuntu** como sistema operativo.
3. Generas un **par de claves SSH (.pem)** para conectarte de manera segura.
4. AWS configura automáticamente el usuario **ubuntu** como usuario por defecto.
5. El usuario **ubuntu** tiene privilegios de administrador, pero **sin acceso root directo** (el acceso root está deshabilitado por seguridad).

**2. Usuario Ubuntu y Administración del Sistema**

El usuario ubuntu es un usuario con privilegios de **sudo**, lo que significa que puede instalar software y administrar el sistema sin necesidad de ser root directamente.

**Ejemplo de instalación de Nginx con el usuario ubuntu**:

sudo apt update

sudo apt install nginx -y

En este caso, sudo permite al usuario ubuntu ejecutar comandos como si fuera root.

**3. Acceso al Servidor y Configuración de SSH**

El usuario **ubuntu** se conecta a la máquina mediante **SSH** utilizando la clave .pem generada al crear la instancia.

**Ejemplo de conexión SSH desde tu terminal**:

ssh -i clave.pem ubuntu@IP\_DEL\_SERVIDOR

Aquí:

* -i clave.pem → Indica la clave privada para autenticarse.
* ubuntu@IP\_DEL\_SERVIDOR → Es el usuario por defecto y la IP del servidor.

Para evitar escribir este comando largo, se puede configurar en el archivo ~/.ssh/config:

Host servidor\_aws

HostName IP\_DEL\_SERVIDOR

User ubuntu

IdentityFile /ruta/a/clave.pem

Luego, puedes conectarte solo con:

ssh servidor\_aws

**4. Instalación de Software y Configuración Básica**

El usuario ubuntu puede instalar software, pero no tiene acceso root directo. Si necesitas instalar paquetes o servicios, usa sudo.

📌 **Ejemplo de instalación de paquetes esenciales**:

sudo apt update && sudo apt upgrade -y

sudo apt install nginx git curl nano -y

Si necesitas verificar el estado de servicios como **Nginx**:

sudo systemctl status nginx

Para habilitarlo al iniciar el sistema:

sudo systemctl enable nginx

Para iniciar o reiniciar Nginx:

sudo systemctl restart nginx

**5. Creación de Usuarios y Configuración de Permisos**

Si necesitas agregar nuevos usuarios al sistema, puedes hacerlo con:

sudo adduser nuevo\_usuario

sudo passwd nuevo\_usuario # Asignar contraseña

Si quieres que el usuario tenga permisos administrativos:

sudo usermod -aG sudo nuevo\_usuario

Esto lo agregará al grupo de administradores.

:

**6. Configuración del Firewall en AWS y en Ubuntu**

El firewall de **AWS EC2** está gestionado por los **Security Groups**. Debes permitir el tráfico HTTP (80) y HTTPS (443) para acceder a tu servidor web.

**Cómo abrir el puerto 80 en AWS**:

1. Ir a **Security Groups** en AWS.
2. Seleccionar el grupo de seguridad de tu instancia.
3. En **Inbound Rules**, agregar:
   * **Tipo**: HTTP
   * **Protocolo**: TCP
   * **Puerto**: 80
   * **Origen**: Anywhere (0.0.0.0/0)

**Conclusión**

* **El usuario ubuntu es el usuario "master" del sistema** en AWS con permisos sudo.
* **No se accede como root directamente** (se usa sudo para tareas administrativas).
* **ubuntu puede instalar Nginx** y otros servicios usando sudo apt install nginx.
* **La seguridad está gestionada por AWS Security Groups**, que controla qué puertos están abiertos.

**Funcionamiento de sites-available y sites-enabled en Nginx**

En Nginx, la configuración de sitios web se maneja con los directorios **sites-available** y **sites-enabled**. Aquí te explico cómo funcionan y cómo se asignan para renderizar un sitio correctamente.

**1. Explicación de sites-available y sites-enabled**

Nginx separa la configuración de los sitios en dos directorios:

* **/etc/nginx/sites-available/**  
  Contiene archivos de configuración de los sitios web.  
  Aquí se crean las configuraciones de los sitios, pero no están activas aún.
* **/etc/nginx/sites-enabled/**  
  Contiene enlaces simbólicos (ln -s) a los sitios habilitados.  
  Solo los archivos que están aquí se activan y Nginx los procesa al reiniciarse.

**¿Por qué esta separación?**  
Permite mantener configuraciones guardadas sin activarlas.  
Puedes desactivar un sitio sin borrar su configuración.

**2. Creación y Asignación de un Sitio Web**

Para renderizar un sitio en Nginx, sigue estos pasos:

**Paso 1: Crear un nuevo archivo de configuración en sites-available**

Supongamos que tienes una aplicación en /var/www/misitio y quieres que responda en misitio.com:

sudo nano /etc/nginx/sites-available/misitio

**Contenido del archivo de configuración (/etc/nginx/sites-available/misitio)**:

server {

listen 80;

server\_name misitio.com www.misitio.com;

root /var/www/misitio;

index index.html index.htm;

location / {

try\_files $uri $uri/ =404;

}

}

**Explicación de la configuración**:

* listen 80; → Escucha en el puerto 80 (HTTP).
* server\_name misitio.com www.misitio.com; → Dominios que manejará este servidor.
* root /var/www/misitio; → Carpeta donde está el contenido del sitio.
* index index.html index.htm; → Archivos por defecto a cargar.
* try\_files $uri $uri/ =404; → Intenta cargar el archivo o directorio, si no existe, devuelve 404.

**🔹 Paso 2: Activar el sitio (ln -s en sites-enabled)**

Para activar la configuración, crea un **enlace simbólico** en sites-enabled:

sudo ln -s /etc/nginx/sites-available/misitio /etc/nginx/sites-enabled/

**Este comando no copia el archivo, sino que crea un enlace.**

**🔹 Paso 3: Probar la configuración**

Antes de reiniciar Nginx, es recomendable probar la configuración:

sudo nginx -t

Si todo está bien, verás:

nginx: configuration file /etc/nginx/nginx.conf test is successful

**🔹 Paso 4: Reiniciar Nginx**

Para aplicar los cambios, reinicia Nginx:

sudo systemctl restart nginx

**3. ¿Cómo se asigna un sitio para que se renderice?**

Nginx busca la configuración de los sitios en este orden:

1. **Los archivos en /etc/nginx/sites-enabled/** (los habilitados).
2. **El dominio (server\_name) que coincida con la solicitud**.
3. **Si no hay coincidencia, usa el primer server {} encontrado**.

**4. Desactivar un sitio sin borrarlo**

Si quieres desactivar un sitio sin eliminarlo:

sudo rm /etc/nginx/sites-enabled/misitio

sudo systemctl restart nginx

Esto **no borra** el archivo en sites-available, solo desactiva el enlace.

**5. Configurar Nginx para un dominio real**

Si tienes un dominio **registrado** y apuntas su DNS a la IP del servidor, asegúrate de que el server\_name en la configuración coincida con el dominio.

**Ejemplo para misitio.com con HTTPS (SSL):**

server {

listen 80;

server\_name misitio.com www.misitio.com;

return 301 https://$host$request\_uri;

}

server {

listen 443 ssl;

server\_name misitio.com www.misitio.com;

ssl\_certificate /etc/letsencrypt/live/misitio.com/fullchain.pem;

ssl\_certificate\_key /etc/letsencrypt/live/misitio.com/privkey.pem;

root /var/www/misitio;

index index.html index.htm;

location / {

try\_files $uri $uri/ =404;

}

}

return 301 https://$host$request\_uri; → Redirige HTTP a HTTPS.  
listen 443 ssl; → Habilita HTTPS.  
ssl\_certificate /etc/letsencrypt/...; → Configura SSL con Let's Encrypt.

**Conclusión**

**sites-available** guarda configuraciones sin activarlas.  
**sites-enabled** contiene enlaces simbólicos a los sitios activos.  
Para asignar un sitio, debes **crear su configuración, activarlo con ln -s, probar y reiniciar Nginx**.

# Resumen de la Clase 03 - Configuración de Servidores y Despliegue de Aplicaciones

**1. Temas que se vieron**

En esta clase se avanzó en la configuración y uso de **Nginx**, **gestión de procesos en Linux** y **despliegue de aplicaciones web estáticas y React en servidores**. Se trabajó en:

1. **Errores comunes en servidores Linux**
   * Problemas de permisos y rutas mal configuradas.
   * Cómo detectar errores en logs.
2. **Revisión y gestión de procesos en Linux**
   * Uso de top, htop y ps aux para ver procesos en ejecución.
   * Cómo detener procesos con kill.
   * Entender procesos padre-hijo en el sistema.
3. **Configuración de Nginx en detalle**
   * Cómo funciona nginx.conf y sus archivos de configuración.
   * Explicación de sites-available y sites-enabled.
   * Archivos de configuración de sitios web (server {} en Nginx).
   * Configuración de server\_name, root, index y location.
4. **Gestión de logs en Nginx y Linux**
   * Ubicación y análisis de logs en /var/log/nginx/error.log.
   * Cómo interpretar errores HTTP 404 y 403 en Nginx.
5. **Implementación de un sitio web estático en Nginx**
   * Descargar una plantilla HTML desde un servidor externo con wget.
   * Subir archivos al servidor con scp o FileZilla.
   * Configurar permisos y accesibilidad en /var/www/html.
6. **Despliegue de una aplicación React en Nginx**
   * Diferencias entre una web estática y una SPA (Single Page Application).
   * Problema de las rutas en React Router en producción.
   * Configuración de try\_files en Nginx para servir React sin errores.

**2. Explicación a grandes rasgos de cada tema**

**1. Errores comunes en servidores Linux**

Los errores más frecuentes al configurar servidores son:

* **Permisos incorrectos**: El usuario que ejecuta Nginx no tiene acceso a los archivos.
* **Rutas mal definidas**: Si la carpeta raíz (root) no apunta correctamente, el servidor no encuentra los archivos.
* **Errores de firewall**: Si no se abren los puertos en AWS, no se puede acceder al sitio.

**Solución:** Revisar logs con:

sudo cat /var/log/nginx/error.log

Para depurar errores de permisos:

ls -l /ruta/al/archivo

**2. Gestión de procesos en Linux**

Cada proceso en Linux tiene un **ID único (PID)** y un usuario que lo ejecuta. Se puede monitorear con:

top # Muestra procesos activos

htop # Versión mejorada de 'top' con interfaz más visual

ps aux # Lista todos los procesos

Para detener procesos:

kill <PID> # Detiene un proceso

kill -9 <PID> # Mata el proceso de inmediato

**Ejemplo:** Si un proceso de Nginx está fallando, se puede reiniciar con:

sudo systemctl restart nginx

**3. Configuración de Nginx**

El archivo principal está en:

/etc/nginx/nginx.conf

Dentro de este archivo, se incluyen otros archivos con configuraciones de sitios web:

include /etc/nginx/sites-enabled/\*;

Los sitios web se configuran en sites-available, pero solo se activan cuando se enlazan en sites-enabled:

sudo ln -s /etc/nginx/sites-available/misitio /etc/nginx/sites-enabled/

Para verificar la configuración antes de reiniciar:

sudo nginx -t

**4. Gestión de logs en Nginx**

Los logs se encuentran en:

/var/log/nginx/access.log # Peticiones HTTP registradas

/var/log/nginx/error.log # Errores de Nginx

Para ver los últimos errores en tiempo real:

tail -f /var/log/nginx/error.log

**Ejemplo de error de permisos en logs:**

Permission denied while trying to open /var/www/html/index.html

**Solución:** Ajustar permisos con:

**5. Implementación de un sitio web estático**

Para desplegar un sitio web en Nginx:

1. Descargar la plantilla con wget:

wget https://ejemplo.com/miweb.zip

1. Extraer los archivos:

unzip miweb.zip -d /var/www/html/

1. Configurar Nginx en /etc/nginx/sites-available/misitio:

server {

listen 80;

server\_name miweb.com;

root /var/www/html;

index index.html;

location / {

try\_files $uri $uri/ =404;

}

}

1. Habilitar el sitio y reiniciar Nginx:

sudo ln -s /etc/nginx/sites-available/misitio /etc/nginx/sites-enabled/

sudo systemctl restart nginx

**6. Despliegue de una aplicación React en Nginx**

Una aplicación React se **compila** antes de desplegarse:

1. Generar la carpeta de distribución:

npm run build

1. Subir la carpeta build al servidor con scp:

scp -r build ubuntu@mi-servidor:/var/www/react-app

1. Configurar Nginx para servir React en /etc/nginx/sites-available/react:

server {

listen 80;

server\_name miweb.com;

root /var/www/react-app;

index index.html;

location / {

try\_files $uri /index.html;

}

}

**Nota:** La clave para que React Router funcione en producción es:

try\_files $uri /index.html;

Esto redirige todas las rutas desconocidas a index.html, permitiendo que React maneje la navegación.

**3. Proceso seguido en la clase**

1. **Revisión de problemas comunes en servidores Linux**.
2. **Monitoreo y gestión de procesos con top, htop y ps aux**.
3. **Configuración avanzada de Nginx y gestión de sitios web**.
4. **Subida de archivos al servidor con scp y FileZilla**.
5. **Implementación de una web estática en Nginx**.
6. **Configuración y despliegue de una aplicación React**.
7. **Corrección de errores con logs y permisos**.
8. **Explicación de cómo Nginx maneja las rutas en React**.

**Próximos pasos**

En la siguiente clase se trabajará en:

* **Configuración avanzada de Nginx** (proxy inverso para Node.js).
* **Configuración de un backend en el servidor**.
* **Seguridad y buenas prácticas en producción**.

**Conclusión**

* **Nginx** requiere configurar sites-available y activarlo con sites-enabled.
* **React Router en producción necesita try\_files $uri /index.html; en Nginx**.
* **Los logs en /var/log/nginx/error.log ayudan a depurar errores rápidamente**.
* **SCP y FileZilla permiten transferir archivos entre el servidor y el equipo local**.

# Resumen de la Clase 04 - Configuración de Servidores y Despliegue de Aplicaciones

**1. Temas que se vieron**

En esta clase se avanzó en la **gestión de servidores en AWS**, **configuración avanzada de Nginx**, **manejo de IPs fijas en AWS** y **despliegue de una aplicación Node.js con un gestor de procesos**. Se trabajó en:

1. **Configuración de IP fija en AWS (Elastic IPs)**
   * Diferencia entre IP dinámica y Elastic IP.
   * Cómo asignar una IP fija a una instancia EC2.
   * Costos y administración de Elastic IP en AWS.
2. **Configuración avanzada de SSH**
   * Problemas comunes con la conexión SSH y cómo solucionarlos.
   * Configuración del archivo ~/.ssh/config para accesos más eficientes.
   * Manejo de claves SSH y autenticación con contraseña.
3. **Despliegue de una aplicación Node.js en un servidor**
   * Instalación de Node.js en un usuario específico.
   * Creación de un usuario dedicado para la aplicación (seguridad).
   * Instalación de nvm para gestionar versiones de Node.js.
4. **Uso de un gestor de procesos para Node.js**
   * Instalación y configuración de **Supervisor**.
   * Alternativa con **PM2** para manejar procesos en Node.js.
   * Cómo reiniciar automáticamente una aplicación si se detiene.
5. **Configuración de Nginx como Proxy Inverso**
   * Uso de proxy\_pass para redirigir tráfico a Node.js.
   * Manejo de websockets con Nginx.
   * Configuración de subdominios para frontend y backend en un solo servidor.
6. **Asignación de un dominio a la aplicación**
   * Uso de **dacdns.org** para obtener un subdominio gratuito.
   * Configuración de DNS en AWS para dominios personalizados.
   * Diferencia entre server\_name en Nginx y registros DNS.
7. **Implementación de HTTPS con Let's Encrypt**
   * Instalación de Certbot para gestionar certificados SSL.
   * Configuración automática de Nginx para manejar HTTPS.
   * Redirección forzada de HTTP a HTTPS.

**2. Explicación a grandes rasgos de cada tema**

**1. Configuración de IP fija en AWS**

* **Elastic IPs** permiten mantener una IP estática para que el servidor no cambie su IP pública al reiniciar.
* Se asigna desde la consola de AWS en **"Network & Security" > "Elastic IPs"**.
* Una Elastic IP sin asociar a una instancia **genera costos**, pero una asociada a una instancia en uso es gratuita.

**2. Configuración avanzada de SSH**

* Se optimizó el acceso SSH mediante el archivo ~/.ssh/config para definir alias y accesos rápidos.
* Se revisó cómo solucionar errores de conexión y el uso de claves SSH en diferentes sistemas operativos.

**3. Despliegue de una aplicación Node.js**

* Se creó un usuario llamado apps para ejecutar la aplicación con menor riesgo de seguridad.
* Se utilizó nvm para instalar una versión específica de Node.js y evitar problemas de compatibilidad.
* Se realizó un **Git Clone** del repositorio de la aplicación para descargarla en el servidor.

**4. Uso de un gestor de procesos**

* Se configuró **Supervisor** para mantener la aplicación en ejecución incluso después de un reinicio.
* Se utilizó:

sudo apt install supervisor

sudo nano /etc/supervisor/conf.d/nodechat.conf

* Se estableció la configuración de Supervisor para reiniciar automáticamente la aplicación si falla.

**5. Configuración de Nginx como Proxy Inverso**

* Se configuró **Nginx** para actuar como **proxy inverso**, redirigiendo tráfico de la IP pública al puerto donde se ejecuta Node.js.
* Se utilizó proxy\_pass para conectar Nginx con la aplicación Node.js:

location / {

proxy\_pass http://127.0.0.1:3000;

}

* Se ajustaron permisos para permitir WebSockets y otras conexiones en Node.js.

**6. Asignación de un dominio**

* Se usó **dacdns.org** para registrar un subdominio gratuito apuntando a la IP del servidor.
* Se modificó la configuración de Nginx para servir diferentes aplicaciones según el dominio/subdominio.

**7. Implementación de HTTPS**

* Se instaló **Certbot** para obtener certificados SSL gratuitos con Let's Encrypt:

sudo apt install certbot python3-certbot-nginx

sudo certbot --nginx

* Se configuró **redirección automática de HTTP a HTTPS** en Nginx.

**3. Proceso seguido en la clase**

1. **Configuración de Elastic IP** en AWS para obtener una IP fija.
2. **Configuración de SSH y claves de acceso** para mejorar la conexión remota.
3. **Creación del usuario apps y configuración de Node.js** con nvm.
4. **Descarga y ejecución de la aplicación Node.js**.
5. **Instalación y configuración de Supervisor** para gestionar la ejecución de Node.js.
6. **Configuración de Nginx como Proxy Inverso** para manejar las peticiones HTTP.
7. **Asignación de un subdominio gratuito** con **dacdns.org**.
8. **Configuración de HTTPS con Let's Encrypt** para mejorar la seguridad.

# Clase 5 - Configuración de DNS y despliegue de una app con MongoDB

**Conceptos explicados al inicio**

1. **Supervisor vs PM2**
   * Ambos sirven para mantener una app corriendo todo el tiempo.
   * PM2 puede ser más complejo de configurar como servicio del sistema.
   * Supervisor es más sencillo de configurar y se recomienda para producción si estás aprendiendo.
2. **Uso de dist o no**
   * Cuando usas NestJS, el build genera un dist/, y eso es lo que se debe desplegar.
   * Pero si tienes acceso al código y puedes correr npm start, puedes hacerlo sin necesidad de subir dist.

**Explicación de DNS**

1. **Qué es DNS**
   * Traduce nombres como google.com a direcciones IP.
   * Funciona como una agenda telefónica.
2. **Tipos de registros DNS**
   * A: IPv4 (lo que usaremos).
   * AAAA: IPv6.
   * CNAME: alias de otros dominios.
   * MX: correo.
   * NS: name servers, quién gestiona el dominio.
3. **Cómo se resuelve un dominio**
   * Desde el navegador → al sistema operativo → al router → al servidor DNS público (como 8.8.8.8).
   * Si no sabe la IP, va preguntando a otros DNS (raíz, TLD, autoridad).
   * Finalmente obtiene la IP del dominio y navegas a ese servidor.

**Ejemplo práctico con AWS y DAGDNS**

1. **Pasos realizados**
   * Crear subdominio en dagdns.org.
   * Asociar el subdominio con la IP pública de AWS EC2.
   * Comprobar con dnschecker.org si se propaga.
2. **Configurar NGINX**
   * Se modifica el server\_name para incluir nuevos dominios.
   * Se hace sudo nginx -t y sudo systemctl reload nginx.
   * Se renueva el certificado con certbot si es necesario.

**Despliegue de Parse Server (App con MongoDB)**

**Parse: qué es**

* Plataforma tipo Firebase.
* Fue de Facebook, ahora es open source.
* Puedes guardar datos sin necesidad de escribir todo el backend.
* Requiere MongoDB.

**Pasos para el despliegue**

1. **Clonar el repositorio de Parse**
   * Se cambia el nombre de carpeta al clonar si se desea.
2. **Requisitos**
   * Verificar versión compatible de Node con package.json.
   * Usar nvm para instalar versión específica si hace falta.
3. **MongoDB**

* No se instala por defecto desde apt, se agrega el repo oficial.
* Requiere agregar la clave GPG y el repositorio.
* Se instala con sudo apt install mongodb-org.

1. **Securizar MongoDB**

* Por defecto **no tiene autenticación**, se considera entorno "confiable".
* Se crea usuario administrador con db.createUser() desde mongosh.
* Se activa autenticación en /etc/mongod.conf con:

security:

authorization: enabled

* Se reinicia Mongo con sudo systemctl restart mongod.

**Crear usuario específico para la app**

1. **Evitar usar admin**

* Crear base de datos parseDB.
* Crear usuario parseUser con acceso solo a esa base de datos.

1. **Probar que funciona**

* Conectarse con mongosh usando --authenticationDatabase.
* Insertar documentos y verificar que lee/escribe correctamente.

**Configuración de Supervisor**

1. **No usar dotenv**

* Como no se puede modificar el código, se pasan las variables de entorno directamente en el archivo de Supervisor.

1. **Ejemplo config parse.conf:**

command=npm start

user=apps

directory=/home/apps/parse

autostart=true

autorestart=true

environment=PATH="/home/apps/.nvm/versions/node/v16.20.2/bin:%(ENV\_PATH)s",DATABASE\_URI="mongodb://parseUser:parsePass@127.0.0.1:27017/parseDB"

1. **Problema típico:**

* Parse da error si Mongo no está activo al momento de arrancar.
* Solución: habilitar Mongo para que arranque automáticamente:

sudo systemctl enable mongod

**Resumen de puntos clave para el despliegue completo**

1. Crear usuario del sistema (apps, web17, etc.)
2. Clonar la app.
3. Instalar dependencias (npm install).
4. Configurar la base de datos MongoDB:
   * Instalar.
   * Activar autenticación.
   * Crear base y usuario.
5. Probar app con npm start.
6. Configurar Supervisor (con path y env).
7. Probar que se arranca solo tras reinicio.
8. Configurar NGINX + subdominio + certbot.