

Nombre del estudiante: Daniel Alconchel Vázquez
Semestre: 2
Año Académico: 2023/2024



UNIVERSIDAD
DE GRANADA

E.T.S. de Ingenierías Informática y de Telecomunicación
Práctica 3: Seguridad (certificados SSL)
Servidores Web de Altas Prestaciones

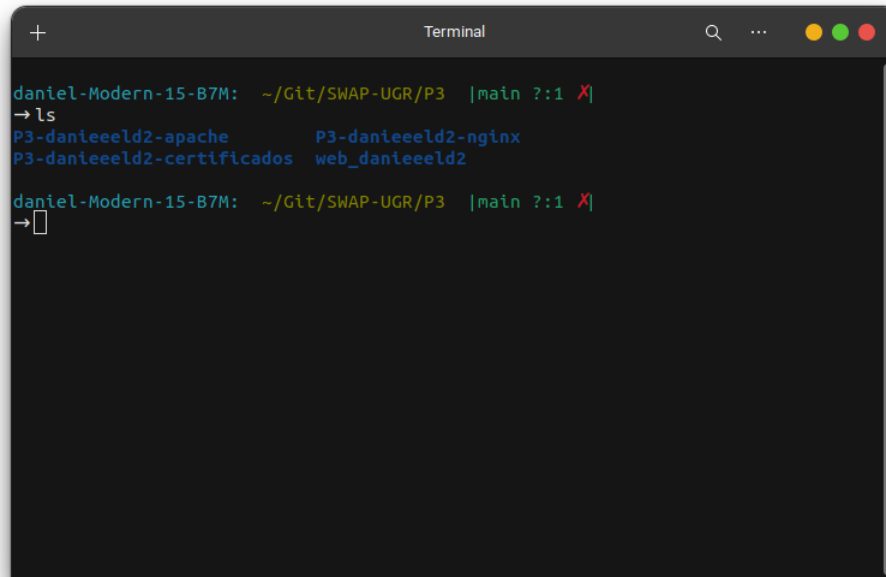
Índice

1. Tareas Básicas	2
1.1. Creación del espacio de trabajo SSL	2
1.2. Creación de certificados SSL	2
1.3. Configuración de Servidores Web Apache con SSL	4
1.4. Configuración del balanceador de carga Nginx con SSL	5
1.5. Configuración de Docker Compose para la Granja Web con SSL	6
1.6. Verificación y pruebas del escenario con SSL	7
2. Tareas Avanzadas	9
2.1. Exploraciones Avanzadas de creación de certificados SSL	9
2.2. Optimización de la configuración SSL en los servidores web	11
2.3. Configuración de Caché y Tickets de Sesión SSL en el balanceador	13
2.4. Optimización de conexiones HTTPS y cifrado en el balanceador	14

1. Tareas Básicas

1.1. Creación del espacio de trabajo SSL

Vamos a comenzar creando la estructura de directorios que nos indica el guión de la práctica:

A terminal window titled 'Terminal' with a dark background. The prompt is 'daniel-Modern-15-B7M: ~/Git/SWAP-UGR/P3 |main ? :1 X|'. The user enters 'ls' and the output shows a directory listing: 'P3-danieeeld2-apache', 'P3-danieeeld2-nginx', 'P3-danieeeld2-certificados', and 'web_danieeeld2'. The prompt then changes to 'daniel-Modern-15-B7M: ~/Git/SWAP-UGR/P3 |main ? :1 X|' and the user enters a command that results in a blank line.

```
daniel-Modern-15-B7M: ~/Git/SWAP-UGR/P3 |main ? :1 X|
→ ls
P3-danieeeld2-apache      P3-danieeeld2-nginx
P3-danieeeld2-certificados web_danieeeld2

daniel-Modern-15-B7M: ~/Git/SWAP-UGR/P3 |main ? :1 X|
→
```

1.2. Creación de certificados SSL

Nos piden crear un certificado SSL con las siguientes características:

- Generar un certificado **autofirmado**.
- Validez del certificado: 1 año (365 días).
- Usar encriptación RSA de 2048 bits.
- No usar **passphrase**.

Para ello, vamos a preguntarle a ChatGPT cómo generarlo:

- **Yo:** Cómo creo una clave privada en openssl usando el algoritmo RSA de 2048 bits y que no use passphrase. La clave generada se llamará certificado_danieeeld2.key.
- **ChatGPT:** Para generar una clave privada en OpenSSL utilizando el algoritmo RSA de 2048 bits sin una frase de contraseña, puedes usar:

```
openssl genpkey -algorithm RSA -out certificado_danieeeld2.key -pkeyopt
rsa_keygen_bits:2048 -aes256 -pass pass:
```

Análisis de la salida: La salida no es incorrecta, pero hay algunas consideraciones a tener en cuenta. Como vemos, indica correctamente el uso de RSA usando 2048 bits, pero, a continuación, indica que usará AES con una longitud de 256 bits para cifrar. Acto seguido, le indica con **-pass pass:** que no se va a

utilizar frase de contraseña, luego, tenemos un problema y es que si ciframos con AES sin **passphrase**, pese a que aumentamos la seguridad, no podríamos descifrar la clave privada. Debido a esto, vamos a omitir esta última parte del comando (aunque sea menos seguro al no estar cifrado) y vamos a seguir estrictamente los pasos del guión. Luego, para generar la clave privada ejecutaríamos:

```
openssl genpkey -algorithm RSA -out certificado_danieeeld2.key -pkeyopt  
rsa_keygen_bits:2048
```

Otra posible forma de generar la clave privada sería:

```
openssl genrsa -out certificado_danieeeld2.key 2048
```

Vamos a continuar ahora preguntándole a ChatGPT:

- **Yo:** He generado una clave privada usando RSA de 2048 bits, llamada certificado_danieeeld2.key. Tengo que generar un certificado autofirmado con validez de 365 días. Este certificado se llamará certificado_danieeeld2.crt
- **ChatGPT:** Nos proporciona el siguiente comando:

```
openssl req -x509 -nodes -days 365 -key certificado_danieeeld2.key -out  
certificado_danieeeld2.crt
```

Análisis de la salida: El comando parece correcto. Usa **-x509** para generar un certificado X.509 autofirmado. Esto abrirá un diálogo de terminal, donde iremos rellenando los datos.

Nota: Como curiosidad, se puede usar **-config** para automatizar algunos parámetros y el relleno del diálogo y la opción **-new** para generar clave privada y certificado en el mismo paso.


```

certificado_tuusuariougr.key

# Configurar los permisos adecuados
RUN chmod 600 /etc/apache2/ssl/certificado_tuusuariougr.crt

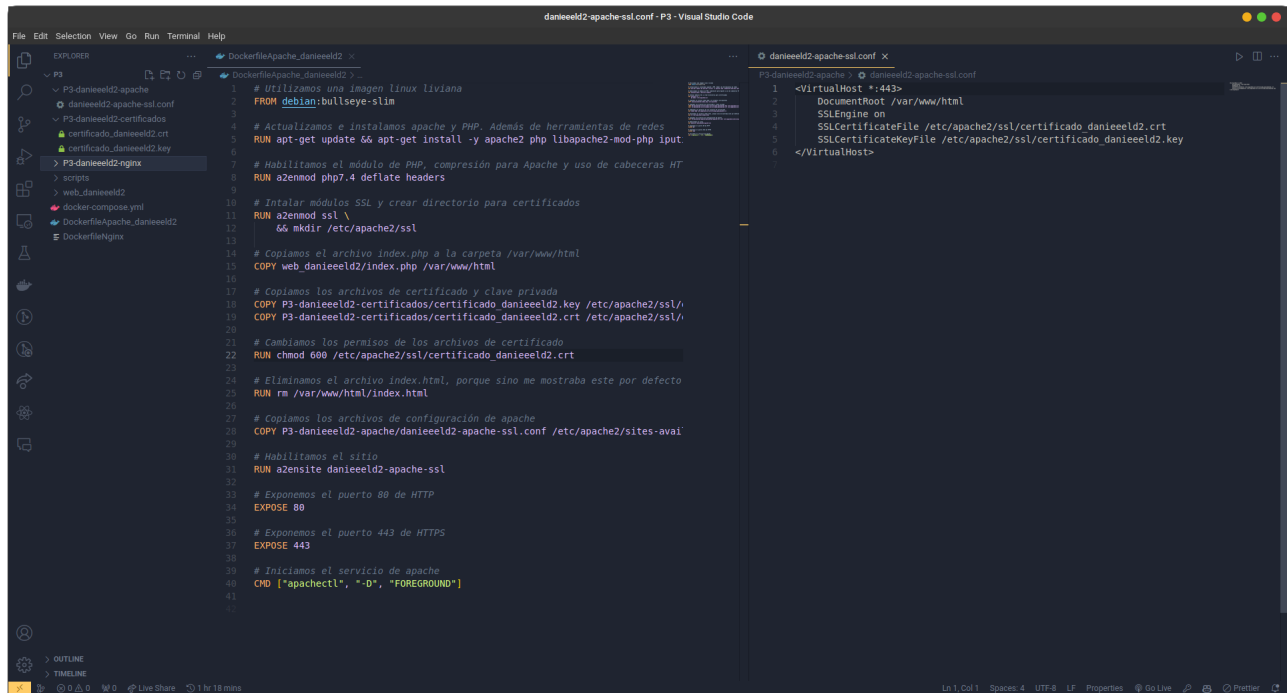
# Incluir la configuracion SSL
COPY tuusuariougr-ssl.conf /etc/apache2/sites-available/tuusuariougr-ssl.conf

# Exponer el puerto HTTPS
EXPOSE 443

```

Vamos a seguir estos pasos para la creación de la imagen, aunque, de todas formas, montaremos volúmenes posteriormente en el `docker-compose`.

Finalmente, la configuración de Apache quedaría de la siguiente forma:



```

danieeeld2-apache-ssl.conf - P3 - Visual Studio Code
DockerfileApache_danieeeld2
# Utilizamos una imagen linux liviana
FROM debian:bullseye-slim

# Actualizamos e instalamos apache y PHP. Además de herramientas de redes
RUN apt-get update && apt-get install -y apache2 php libapache2-mod-php iputils

# Habilitamos el módulo de PHP, compresión para Apache y uso de cabeceras HT
RUN a2enmod php7.4 deflate headers

# Instalamos módulos SSL y creamos directorio para certificados
RUN a2enmod ssl \
    && mkdir /etc/apache2/ssl

# Copiamos el archivo index.php a la carpeta /var/www/html
COPY web_danieeeld2/index.php /var/www/html

# Copiamos los archivos de certificado y clave privada
COPY P3-danieeeld2-certificados/certificado_danieeeld2.key /etc/apache2/ssl/
COPY P3-danieeeld2-certificados/certificado_danieeeld2.crt /etc/apache2/ssl/

# Cambiamos los permisos de los archivos de certificado
RUN chmod 600 /etc/apache2/ssl/certificado_danieeeld2.crt

# Eliminamos el archivo index.html, porque sino me mostraba este por defecto
RUN rm /var/www/html/index.html

# Copiamos los archivos de configuración de apache
COPY P3-danieeeld2-apache/danieeeld2-apache-ssl.conf /etc/apache2/sites-available

# Habilitamos el sitio
RUN a2ensite danieeeld2-apache-ssl

# Exponemos el puerto 80 de HTTP
EXPOSE 80

# Exponemos el puerto 443 de HTTPS
EXPOSE 443

# Iniciamos el servicio de apache
CMD ["apachectl", "-D", "FOREGROUND"]

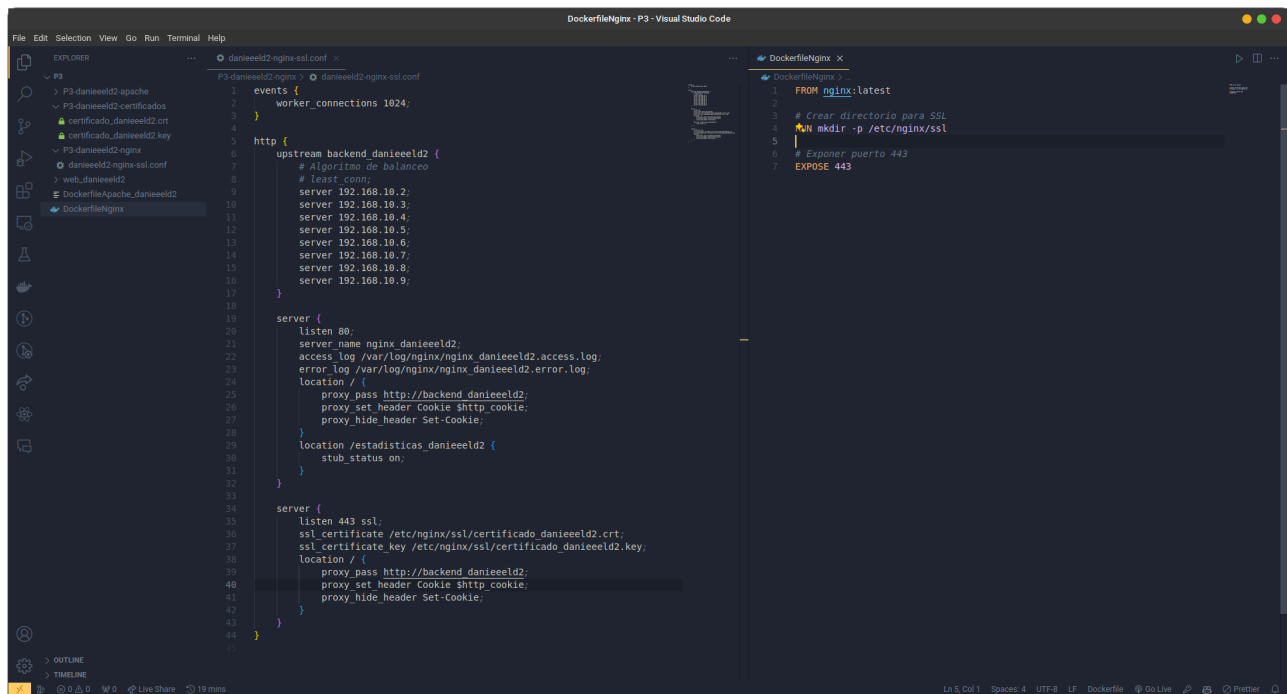
```

Vamos a dejar ya construida la imagen, para lo que ejecutamos:

```
docker build -t danieeeld2-apache-image:p3 -f DockerfileApache_danieeeld2 .
```

1.4. Configuración del balanceador de carga Nginx con SSL

Para esta parte, vamos a rescatar el fichero de configuración de Nginx de la práctica anterior y le añadiremos la nueva sección de configuración básica que indica el guión. Además, modificaremos el Dockerfile para habilitar el puerto 443 (Los archivos los gestionaré posteriormente con un volumen en `docker-compose`).



Vamos a dejar construida la imagen:

```
docker build -t danieeeld2-nginx-image:p3 -f DockerfileNginx .
```

1.5. Configuración de Docker Compose para la Granja Web con SSL

Para este apartado, vamos a rescatar la script de generación de docker-compose de la práctica anterior y le vamos a realizar algunos ajustes para obtener lo que buscamos:

```
#!/bin/bash

if [ "$#" -ne 1 ]; then
    echo "Usage: $0 <number_of_instances>"
    exit 1
fi

num_instances=$1

cat <<EOF >docker-compose.yml
version: '3.0'

services:
EOF

for ((i=1; i<=$num_instances; i++)); do
    cat <<EOF >>docker-compose.yml
web$i:
    image: danieeeld2-apache-image:p3
    container_name: web$i
    privileged: true
    command: apachectl -D FOREGROUND
    volumes:
        - ./web_danieeeld2:/var/www/html
        - ./P3-danieeeld2-certificados:/etc/apache2/ssl
        - ./P3-danieeeld2-apache/danieeeld2-apache-ssl.conf:/etc/apache2/sites-available/danieeeld2-apache-ssl.conf
```

```

    networks:
      red_web:
        ipv4_address: 192.168.10.$((i+1))
      red_servicios:
        ipv4_address: 192.168.20.$((i+1))

EOF
done

cat <<EOF >>docker-compose.yml
balanceador-nginx:
  image: danieeeld2-nginx-image:p3
  container_name: balanceador-nginx
  ports:
    - "80:80"
    - "443:443"
  command: ['nginx', '-g', 'daemon off;']
  volumes:
    - ./P3-danieeeld2-nginx/danieeeld2-nginx-ssl.conf:/etc/nginx/nginx.conf
    - ./P3-danieeeld2-certificados:/etc/nginx/ssl
  networks:
    red_web:
      ipv4_address: 192.168.10.50
  depends_on:
    EOF

for ((i=1; i<=$num_instances; i++)); do
  echo "      - web$i" >>docker-compose.yml
done

cat <<EOF >>docker-compose.yml

networks:
  red_web:
    driver: bridge
    ipam:
      config:
        - subnet: 192.168.10.0/24
  red_servicios:
    driver: bridge
    ipam:
      config:
        - subnet: 192.168.20.0/24

EOF

```

Nota: Los cambios más relevantes son la creación de volúmenes para los certificados, tanto en Nginx como en Web, la creación del volumen para la configuración de Apache, el uso de `command: apachectl -D FOREGROUND` para recarga la configuración (*la configuración ya está habilitada, luego si cambiamos la misma solo tenemos que recargar apache*) y bindear el puerto 443 de nuestro pc al de Nginx.

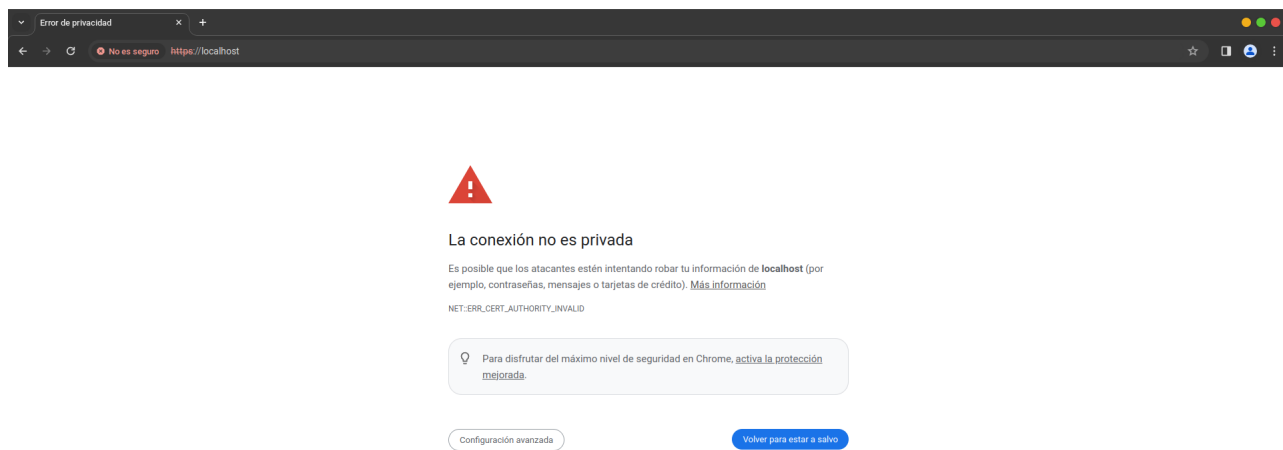
Como siempre, ejecutamos:

```
./scripts/generar-docker-compose.sh 8
```

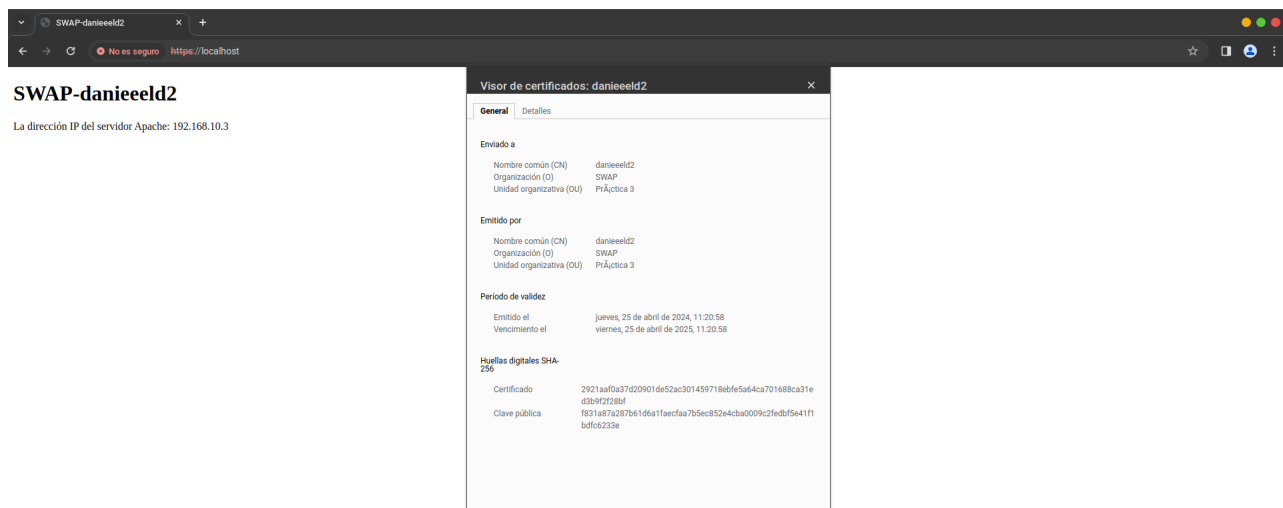
y ya tendríamos listo el `docker-compose`.

1.6. Verificación y pruebas del escenario con SSL

Para comprobar el funcionamiento, accedemos a `https://localhost/` y veamos que ocurre:



En primera instancia nos dice que el sitio no es seguro. Esto es evidente, porque si los certificados auto-firmados fueran seguros, todo el mundo podría crear páginas seguras sin necesidad de nadie que las avale y vele por la seguridad de las mismas. Para que el certificado fuese válido, tendría que avalarnos una entidad certificadora. Dicho esto, vamos a continuar diciéndole al navegador que aceptamos el riesgo:



Como vemos aparece el certificado y Nginx está balanceando correctamente.

2. Tareas Avanzadas

2.1. Exploraciones Avanzadas de creación de certificados SSL

Para esta sección, tenemos que seguir los siguientes pasos:

1. Generar la clave privada del certificado raíz **CA**.
2. Generar el certificado autofirmado de **CA**.
3. Crear la clave privada de la Autoridad Certificadora Intermedia (**subCA**).
4. Generar la solicitud de certificado de **subCA**.
5. Firmar la solicitud con **CA**, creando así la cadena de confianza.

Para los tres primeros puntos es bastante simple, puesto que es el procedimiento que hemos seguido en 1.2. Vamos a adaptar los comandos para que tengan los nombres adecuados:

[illegible]

Para los siguientes puntos, vamos a comenzar preguntándole a Gemini:

- **Yo:** He generado con openssl una clave privada y un certificado autofirmado con esa clave para una entidad llamada CA. Por otro lado, he generado otra clave privada para una entidad subCA. ¿Cómo genero una solicitud de certificado (CSR) de subCA para que esta sea firmada por CA?
- **Gemini:** Para generar una solicitud de certificado (CSR) de subCA para que sea firmada por CA, sigue estos pasos:

1. Generar la clave privada de la subCA:

```
openssl genrsa -out subca.key 2048
```

2. Crear la solicitud de certificado (CSR):

```
openssl req -new -key subca.key -out subca.csr
```

3. Firmar la CSR con la CA:

```
openssl x509 -req -in subca.csr -CA ca.crt -CAkey ca.key  
-CAcreateserial -out subca.crt -days 365
```

Análisis de la salida: Si observamos, el procedimiento es prácticamente el mismo. Antes lo que hacíamos era, en la misma línea, crear el CSR y firmarlo nosotros mismos, obteniendo así el certificado. Como ahora hay dos partes involucradas, hay que separarlo en dos pasos.

Por otro lado, notemos que, sin indicar nada, nos ha hecho un certificado de 365 días de vigor y sin passphrase.

```
daniel-Modern-15-B7M: ~/.../P3/P3-danieeeld2-certificados |main ? :3 X|
→openssl req -new -key certificado_subCA.key -out subCA.csr
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:ES
State or Province Name (full name) [Some-State]:Granada
Locality Name (eg, city) []:Granada
Organization Name (eg, company) [Internet Widgits Pty Ltd]:SWAP-subCA
Organizational Unit Name (eg, section) []:P3
Common Name (e.g. server FQDN or YOUR name) []:danieeeld2-subCA
Email Address []:subCA-danieeeld2@correo.ugr.es

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:

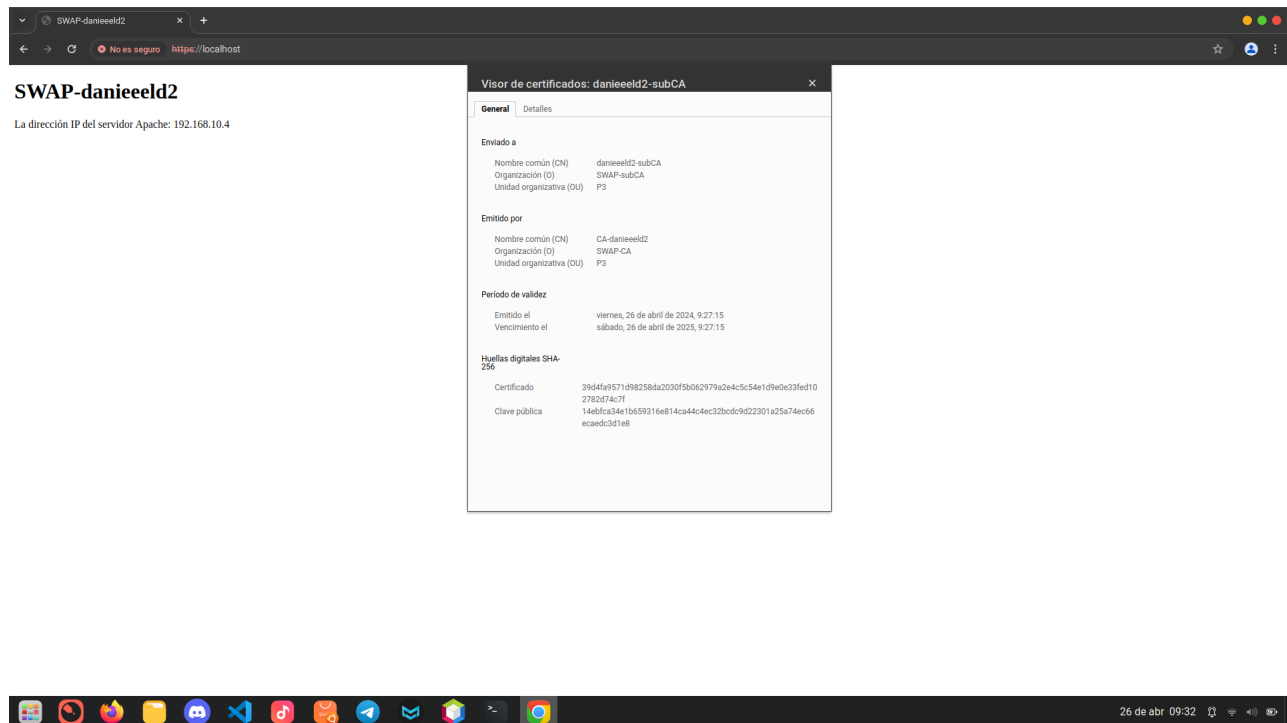
daniel-Modern-15-B7M: ~/.../P3/P3-danieeeld2-certificados |main ? :4 X|
→openssl x509 -req -in subCA.csr -CA certificado_CA.crt -CAkey certificado_CA.key -CAcreateserial -out
certificado_subCA.crt -days 365
Certificate request self-signature ok
subject=C = ES, ST = Granada, L = Granada, O = SWAP-subCA, OU = P3, CN = danieeeld2-subCA, emailAddress
= subCA-danieeeld2@correo.ugr.es

daniel-Modern-15-B7M: ~/.../P3/P3-danieeeld2-certificados |main ? :5 X|
→
```

Para probarlo sin tener que cambiar la configuración, voy a modificar ligeramente el `docker-compose` para que monte el certificado y clave de subCA sobre los que teníamos antes. Esto es:

```
# En los servicios web
- ./P3-danieeeld2-certificados/certificado_subCA.crt:/etc/apache2/ssl/
  certificado_danieeeld2.crt
- ./P3-danieeeld2-certificados/certificado_subCA.key:/etc/apache2/ssl/
  certificado_danieeeld2.key
# En el servicio de nginx
- ./P3-danieeeld2-certificados/certificado_subCA.crt:/etc/nginx/ssl/
  certificado_danieeeld2.crt
- ./P3-danieeeld2-certificados/certificado_subCA.key:/etc/nginx/ssl/
  certificado_danieeeld2.key
```

Podemos observar el correcto funcionamiento:



Nota: Voy a dejar las líneas de estos volúmenes comentadas en la script de generación, por si se quiere probar basta con descomentarlas y comentar la de los volúmenes originales y, finalmente, ejecutar la script.

2.2. Optimización de la configuración SSL en los servidores web

Como siempre, vamos a comenzar preguntándole a ChatGPT:

- **Yo:** Tengo el siguiente archivo de configuración de Apache: (*Le paso la configuración de Apache 1.3*). Tengo que mejorar la seguridad y rendimiento del mismo, mediante el deshabilitado de protocolos inseguros y cifrados débiles. Por ejemplo, permitir o denegar protocolos TLS v1, TLS v1.1 y TLS v1.2 así como cifrados MD5, RC4 y 3DES. Justifícame las elecciones que haces.
- **ChatGPT:** Para mejorar la seguridad y el rendimiento de tu servidor Apache, es importante deshabilitar protocolos inseguros y cifrados débiles. Aquí tienes una versión mejorada de tu archivo de configuración, donde se deshabilitan los protocolos y cifrados mencionados:

```

<VirtualHost *:443>
    DocumentRoot /var/www/html
    SSLEngine on
    SSLCertificateFile /etc/apache2/ssl/certificado_danieeeld2.crt
    SSLCertificateKeyFile /etc/apache2/ssl/certificado_danieeeld2.key

    # Configuración para deshabilitar protocolos y cifrados débiles
    SSLProtocol all -TLSv1 -TLSv1.1 -SSLv2 -SSLv3
    SSLCipherSuite ECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH
</VirtualHost>

```

Vamos a justificar cada elección:

1. Deshabilitar los protocolos TLS v1, TLS v1.1:
 - TLSv1: Fue introducido en 1999 y ya ha sido reemplazado por versiones más seguras.
 - TLSv1.1: Aunque es una mejora sobre TLSv1, también tiene vulnerabilidades conocidas. Es mejor usar versiones más recientes y seguras del protocolo TLS.
2. Deshabilitar cifrados débiles:
 - MD5: Este algoritmo de hash es vulnerable a colisiones y ha sido ampliamente abandonado debido a su debilidad.
 - RC4: Aunque fue popular en el pasado, RC4 es vulnerable a varios ataques, incluyendo ataques de flujo nulo y biases en el flujo de salida.
 - 3DES: Aunque fue un estándar durante mucho tiempo, 3DES utiliza claves de 56 bits, lo que lo hace susceptible a ataques de fuerza bruta.
3. Elegir una configuración de cifrado más segura:
 - Se ha seleccionado una lista de suites de cifrado que proporcionan una seguridad sólida y un buen rendimiento.
 - Se da prioridad a los algoritmos de cifrado simétrico modernos, como AES256, que ofrecen una buena seguridad y rendimiento.

Análisis de la salida: La solución proporcionada es válida y está bien justificada, pero tiene algunos inconvenientes. Por un lado, habilita explícitamente todos los protocolos menos TLS 1.0 y 1.1, por lo que se pueden seguir utilizando protocolos antiguos/inseguros como SSLv2 o SSLv3. Por otro lado, limita la selección de cifrados a una lista específica, que, aunque solo incorpora cifrados seguros, puede no ser exhaustiva y dejar fuera otros cifrados modernos y eficientes.

Para ello, voy a proponer la siguiente solución:

```

<VirtualHost *:443>
    DocumentRoot /var/www/html
    SSLEngine on
    SSLCertificateFile /etc/apache2/ssl/certificado_danieeeld2.crt
    SSLCertificateKeyFile /etc/apache2/ssl/certificado_danieeeld2.key
    # Configuración para deshabilitar protocolos y cifrados débiles
    SSLProtocol -all +TLSv1.3 +TLSv1.2
    SSLCipherSuite ALL:+HIGH:!ADH:!EXP:!SSLv2:!SSLv3:!MEDIUM:!LOW:!NULL:!aNULL
</VirtualHost>

```

Análisis:¹ Por un lado, indicamos que usaremos el protocolo TLSv1.3, que es el más moderno y seguro disponible y el TLSv1.2, que es el más estandar y ofrece buen balance entre seguridad y compatibilidad.

¹La información ha sido extraída consultando en [Google](#). Algunos enlaces de interés son enl1 o enl2

Por otro lado, con **+HIGH** especificamos que se usen los cifrados fuertes. Por otro lado, deshabilitamos los cifrados ADH (Anonymous Diffie-Hellman), los EXP (experimentales), los relativos a los protocolos SSLv2 y SSLv3, que son antiguos e inseguros, los cifrados de "fuerza media" con MEDIUM, los cifrados débiles con LOW, los cifrados que no usan encriptación con NULL y, finalmente, los cifrados con autenticación anónima. La solución anterior aporta gran seguridad, pero tiene menor compatibilidad con clientes antiguos. Si queremos una solución algo más permisiva, podríamos permitir el uso de cifrados de "fuerza media." optando por una lista como la siguiente:

```
SSLCipherSuite "EECDH+ECDSA+AESGCM EECDH+aRSA+AESGCM EECDH+ECDSA+SHA384
EECDH+ECDSA+SHA256 EECDH+aRSA+SHA384 EECDH+aRSA+SHA256 EECDH+aRSA+RC4 EECDH
EDH+aRSA !RC4 !aNULL !eNULL !LOW !3DES !MD5 !EXP !PSK !SRP !DSS !RC4"
```

Con el comando `openssl s_client -connect 192.168.10.<numero de host>:443` podemos ver algunos datos de interés sobre la configuración que hemos creado:

```

+ Terminal
subject=C = ES, ST = Granada, L = Granada, O = SWAP, OU = Pr(C3)83(C2)Aictica 3, CN = danieeld2, emailAddress = danieeld2@correo.ugr.es
issuer=C = ES, ST = Granada, L = Granada, O = SWAP, OU = Pr(C3)83(C2)Aictica 3, CN = danieeld2, emailAddress = danieeld2@correo.ugr.es
...
No client certificate CA names sent
Peer signing digest: SHA256
Peer signature type: RSA-PSS
Server Temp Key: X25519, 253 bits
...
SSL handshake has read 1599 bytes and written 373 bytes
Verification error: self-signed certificate
...
New, TLSv1.3, Cipher is TLS_AES_256_GCM_SHA384
Server public key is 2048 bit
Secure Renegotiation IS NOT supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
Early data was not sent
Verify return code: 18 (self-signed certificate)
...
Post-Handshake New Session Ticket arrived:
SSL-Session:
    Protocol : TLSv1.3
    Cipher : TLS_AES_256_GCM_SHA384
    Session-ID: 599592ABA3869D349A9F2E8386E83C5CF9194A714886518F4921D535C45BEC2
    Session-ID-ctx:
    Resumption PSK: 5D17D08F36FF496CB44F1E856362B575627529E47529CB5D4F2B280FA31680B1EA628689E81E646607E53B88B489D161
    PSK Identity: None
    PSK Identity hint: None
    SRP username: None
    TLS session ticket lifetime hint: 300 (seconds)
    TLS session ticket:
0000 - 95 60 3f 0c 0b 74 dc ae 06 78 d3 56 c0 b4 5c      .f7.k.t...x.V.\
0010 - 51 16 39 8b 8b 37 9c de c3 4f 8c 3a bc f0 4f 7f      Q.9..7...0...0.
0020 - 08 58 38 3f 7b 17 3a 0b 56 77 41 97 eb 94 52 0b      .P87{..VwA...R.
0030 - 8b 74 0b 32 d3 91 51 9f 16 07 e1 13 a8 8a c1 25      .tk2..Q..g....N
0040 - ee 25 d7 9b 4a 06 45 0d 02 3c f0 03 da 05 07 e2      .x..JEm<<c....
0050 - df f6 46 74 4b 71 56 07 cc 93 8e cf f7 8a c2 93      ..FtkQv.....
0060 - eb dd 20 c6 48 1c c5 7c e0 a0 3b cd 84 c5 ce 21      ...H..|.....!
0070 - 05 49 43 b3 1e 11 3c 15 37 07 51 23 cf c8 c6 c3      .IC...<.7g0#...
0080 - 51 08 47 65 af 2a e8 05 34 d5 5f a7 43 98 42 f7      Q.G.e..x...c.B.
0090 - 46 5d f6 52 13 da c8 78 5f 31 a0 70 c6 d0 0b aa      F}.R...x..1.p...
00a0 - a4 0b 13 4f 91 2d 26 36 a2 c2 0f 18 46 1c 47 06      ...O..86...F.G.
00b0 - 4a 9a 7d f6 0b d1 3b dc 08 a4 d7 2d 03 ff b8 f1      J.j.k.;...c...
00c0 - b0 b4 af d1 00 c0 93 ea d2 7c bc 00 5f ad 85 9a      ....|...|....
00d0 - a0 c0 fd f7 b7 1e ef dd e0 59 80 cb 7d 49 81 5f      .....V..)I..
...
Start Time: 1714122889
Timeout : 7200 (sec)
Verify return code: 18 (self-signed certificate)
Extended master secret: no
Max Early Data: 0
...
read R BLOCK

```

2.3. Configuración de Caché y Tickets de Sesión SSL en el balanceador

Como siempre, comenzamos preguntándole a ChatGPT:

- **Yo:** Estoy utilizando nginx como balanceador de carga con la siguiente configuración: (*Le paso el fichero de configuración, pero sin la parte del servidor http en el puerto 80*) quiero mejorar la velocidad de las conexiones mediante el uso de caché de sesiones SSL y tickets de sesión.
- **ChatGPT:** Para mejorar la velocidad de las conexiones mediante el uso de caché de sesiones SSL y tickets de sesión en NGINX, puedes agregar algunas directivas de configuración adicionales en tu archivo de configuración. Aquí hay una versión actualizada de tu configuración que incluye la configuración para caché de sesiones SSL y tickets de sesión:

```
...
# Configuración de caché de sesiones SSL
```

```

ssl_session_cache shared:SSL:10m;
ssl_session_timeout 10m;
server {
    listen 443 ssl;
    ...
    # Habilitar caché de tickets de sesión
    ssl_session_tickets on;
    ssl_session_ticket_key /etc/nginx/ssl/session_ticket.key;
    ...
}

```

Análisis de la salida: La salida es correcta. Por un lado, habilita una caché compartida de sesiones SSL mediante la directiva `ssl_session_cache shared:SSL:10m`, donde 10m indica el tamaño en megabytes. A continuación, configura un tiempo de espera de sesión de 10 minutos, es decir, indica que las sesiones SSL son válidas durante 10 minutos. Por último, habilita los tickets de sesión con `ssl_session_tickets on` y especifica la ubicación de la clave de cifrado para los tickets de sesión.

El único ajuste que tenemos que hacer es cambiar el path de la clave de cifrado por la de la clave privada que vayamos a usar. Al intentar usar la que ya teníamos me da el siguiente error:

```

balanceador-nginx | 2024/04/26 21:11:46 [emerg] 1#1: "/etc/nginx/ssl/
certificado_danieeeld2.key" must be 48 or 80 bytes in /etc/nginx/nginx.conf:53
balanceador-nginx | nginx: [emerg] "/etc/nginx/ssl/certificado_danieeeld2.key"
must be 48 or 80 bytes in /etc/nginx/nginx.conf:53
balanceador-nginx exited with code 1

```

Por lo que vamos a generar un archivo de clave específico para los tickets de sesión de 80 bytes²:

```
openssl rand 80 > tickets_danieeeld2.key
```

Como tenemos un volumen, no es necesario modificar el `docker-compose`. Solo será necesario indicar el nuevo archivo en la configuración.

2.4. Optimización de conexiones HTTPS y cifrado en el balanceador

Como siempre, vamos a comenzar preguntándole a la IA:

- **Yo:** Estoy usando nginx como balanceador de carga. Tengo que personalizar los protocolos SSL/TLS y suites de cifrado para equilibrar seguridad y rendimiento y activar HTTP/2 para mejorar la eficiencia de las conexiones HTTPS. Por ejemplo, protocolos TLSv1.2 y TLSv1.3 así como cifrado ECDH, AESGCM, AES256, etc.
- **Gemini:** Here's how to personalize your Nginx configuration for SSL/TLS and HTTP/2:

1. Secure Protocols and Ciphers:

```

ssl_protocols TLSv1.2 TLSv1.3;
ssl_ciphers ECDHE-ECDSA-AES128-GCM-SHA256:
ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:
ECDHE-RSA-AES256-GCM-SHA384;

```

²He buscado cómo se crea en la documentación

2. Enabling HTTP/2:

```
http2_enabled on;
```

Análisis de la salida: Me parece una salida apropiada. Por un lado, habilita HTTP/2, permitiendo así múltiples peticiones desde una sola conexión (*El comando que indica si es erróneo, abajo pongo la configuración completa*). Por otro lado, habilita TLSv1.2, TLSv1.3, cuyas ventajas vimos previamente en 2.2. Por último, habilita los cifrados ECDH (Elliptic Curve Diffie-Hellman), que proporciona una mayor seguridad de las claves que RSA, AES-GCM (Advanced Encryption Standard - Galois/Counter Mode), que es un algoritmo moderno, seguro y eficiente, y AES128 y AES256, que ambos son algoritmos de encriptación simétrica fuertes, que ofrecen un equilibrio entre seguridad y rendimiento.

Finalmente, la configuración queda como:

```
events {
    worker_connections 1024;
}

http {
    upstream backend-danieeeld2 {
        # Algoritmo de balanceo
        # least_conn;
        server 192.168.10.2;
        server 192.168.10.3;
        server 192.168.10.4;
        server 192.168.10.5;
        server 192.168.10.6;
        server 192.168.10.7;
        server 192.168.10.8;
        server 192.168.10.9;
    }

    server {
        listen 80;
        server_name nginx-danieeeld2;
        access_log /var/log/nginx/nginx-danieeeld2.access.log;
        error_log /var/log/nginx/nginx-danieeeld2.error.log;
        location / {
            proxy_pass http://backend-danieeeld2;
            proxy_set_header Cookie $http_cookie;
            proxy_hide_header Set-Cookie;
        }
        location /estadisticas-danieeeld2 {
            stub_status on;
        }
    }

    # Configuración de cache de sesiones SSL
    ssl_session_cache shared:SSL:10m;
    ssl_session_timeout 10m;

    server {
        listen 443 ssl http2;
        ssl_certificate /etc/nginx/ssl/certificado-danieeeld2.crt;
        ssl_certificate_key /etc/nginx/ssl/certificado-danieeeld2.key;

        # Habilitar cache de tickets de sesion
        ssl_session_tickets on;
        ssl_session_ticket_key /etc/nginx/ssl/tickets-danieeeld2.key;
```

```

# Habilitar protocolos TLSv1.2 y TLSv1.3
ssl_protocols TLSv1.2 TLSv1.3;

# Habilitar suites de cifrado ECDHE, AESGCM y AES256
ssl_ciphers ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-
ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384;

location / {
    proxy_pass http://backend.danieeeld2;
    proxy_set_header Cookie $http_cookie;
    proxy_hide_header Set-Cookie;
}
}

```

Y podemos comprobar que todo sigue funcionando correctamente:

SWAP-danieeeld2

La dirección IP del servidor Apache: 192.168.10.3

The screenshot shows a web browser window with the address bar displaying `https://localhost`. Below the browser window, the developer tools are open, showing the 'Network' tab. A table of network requests is visible, with the first request being a GET to `localhost` with a status of 200. The 'Headers' panel on the right shows the response headers for the GET request to `http://localhost/`, including `content-encoding: gzip`, `content-length: 232`, `content-type: text/html; charset=UTF-8`, `date: Fri, 26 Apr 2024 22:39:14 GMT`, `server: nginx/1.25.5`, and `vary: Accept-Encoding`. The 'Request Headers' section shows `Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8`, `Accept-Encoding: gzip, deflate, br`, `Accept-Language: es-ES;q=0.8,en-US;q=0.5,en;q=0.3`, `Connection: keep-alive`, `Host: localhost`, `Sec-Fetch-Dest: document`, `Sec-Fetch-Mode: navigate`, `Sec-Fetch-Site: none`, `Sec-Fetch-User: ?1`, `TE: trailers`, `Upgrade-Insecure-Requests: 1`, and `User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:125.0) Gecko/20100101 Firefox/125.0`.