Docs

Tutoriales

Tutoriales

Anonymous · 27/08/2026

Table of Contents

Tutoriales

Tutoriales

Anonymous · 27/08/2026

Tutoriales

Last updated on 21/08/2026

Configuración de Proxy

Configuración de Proxy para Consultas

Anonymous · 27/08/2026

Configuración de Proxy para Consultas

Un **proxy** es un servidor intermediario que actúa como puente entre tu aplicación y los servicios externos. En el contexto de API Gateway, utilizamos proxies para distribuir las consultas, por ejemplo al SII (Servicio de Impuestos Internos), desde diferentes direcciones IP, evitando así posibles restricciones y limitaciones de uso.

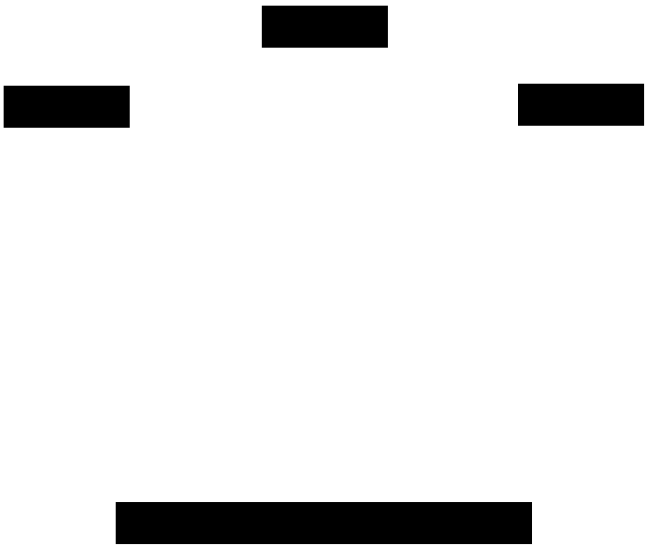
Importante

Cuando cada empresa gestiona sus consultas al SII a través de su propio proxy y dirección IP, se entiende que está haciendo un uso legítimo y responsable de nuestra API. Promovemos y respaldamos la **automatización de consultas tributarias como un derecho fundamental de las empresas** que buscan asegurar el cumplimiento de sus obligaciones fiscales de manera eficiente y transparente.

Riesgo al no utilizar un proxy

El SII puede implementar límites de uso basados en direcciones IP para controlar el acceso a sus servicios. Debido a que API Gateway es la plataforma líder en Chile que ofrece servicios web que el SII no proporciona directamente, debemos procesar un alto volumen de consultas, lo que puede provocar restricciones impuestas por el SII.

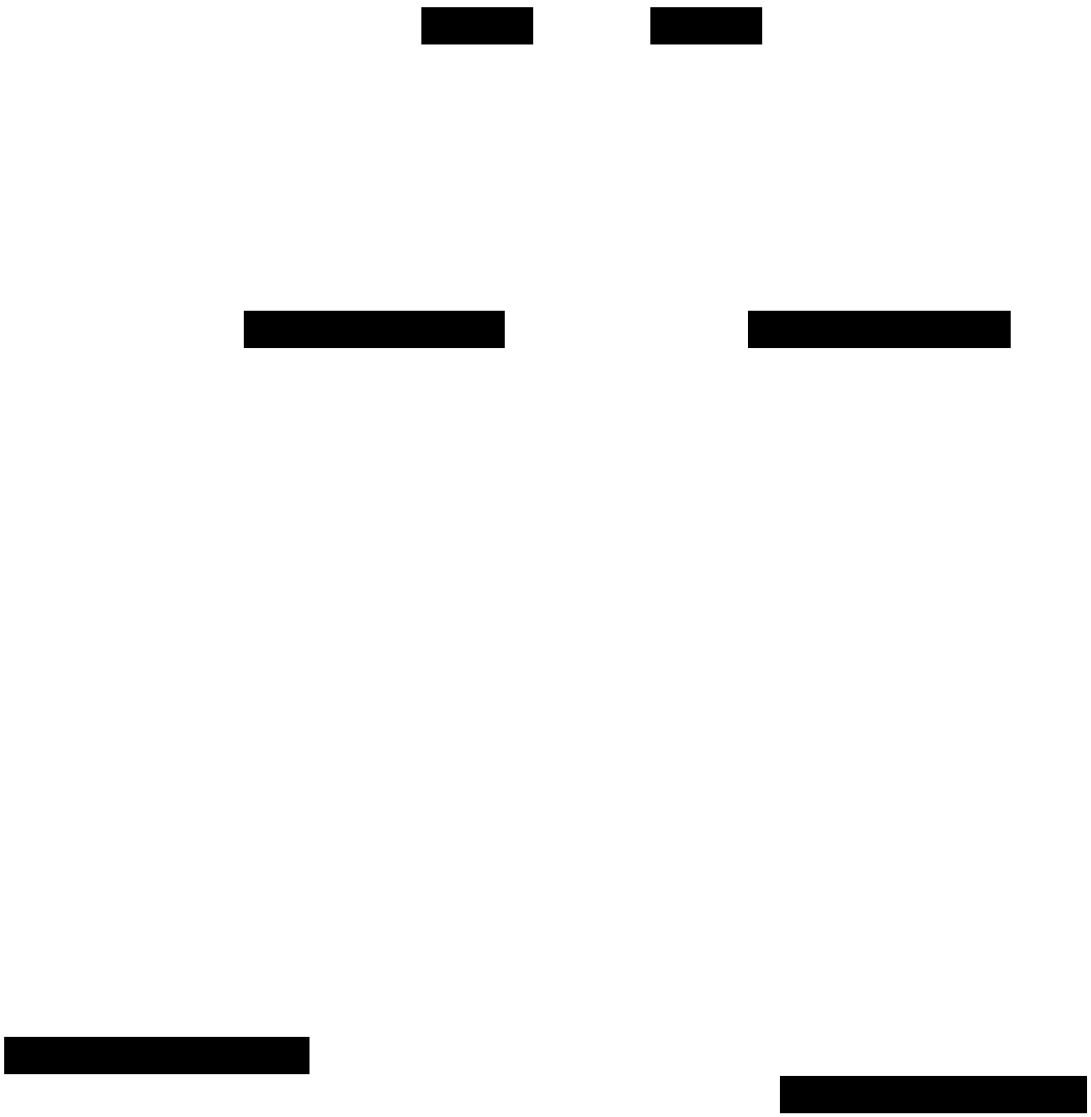
Cuando el SII restringe o limita las consultas desde una IP específica, el riesgo es que afecte a múltiples usuarios. Esto se muestra en el siguiente diagrama:



Funcionamiento del Proxy

Con el proxy, se busca mitigar este problema permitiendo que las consultas se realicen desde direcciones IP diferentes, una por cada empresa/RUT que realiza consultas al SII. Con esto, se distribuye la carga y se minimiza el impacto de las restricciones.

El proxy funciona como un intermediario en la comunicación entre aplicaciones, evitando que una sola dirección IP sea restringida y afecte a múltiples usuarios. Esto se muestra en el siguiente diagrama:



Beneficios del uso de proxy:

- **Aislamiento de restricciones**: Si una IP es limitada, solo afecta a los usuarios de ese proxy específico.
- **Mayor disponibilidad**: Si la IP principal está restringida, los proxies permiten continuar operando.
- **Distribución de carga**: Reduce la presión sobre una sola dirección IP.

Configuración del Proxy

El proxy debe estar ubicado en una red pública accesible desde API Gateway. En este tutorial utilizaremos **Squid**, un proxy HTTP/HTTPS robusto y ampliamente utilizado, pero puedes implementar cualquier solución compatible.

Opciones de despliegue:

- **Servidores propios**: Infraestructura on-premise.
- **Servicios en la nube**: Google Cloud Platform (tiene región en Chile), AWS, Azure, etc.
- **Proveedores locales**: Otros servicios con presencia en Chile.

1. Archivo Docker Compose

Utilizaremos Docker Compose para simplificar la implementación del proxy Squid. Crea un directorio para tu proyecto y dentro de este crea un archivo `docker-compose.yml` con la siguiente configuración:

```
services:
  squid:
    image: ubuntu/squid:latest
    container_name: squid
    restart: unless-stopped
    volumes:
      - "./squid.conf:/etc/squid/squid.conf:ro"
      - "./passwords:/etc/squid/passwords:ro"
    ports:
      - "3128:3128"
```

Nota importante

El puerto 3128 es el puerto estándar de Squid. Si necesitas usar un puerto diferente, asegúrate de modificarlo tanto en el `docker-compose.yml` como en el archivo de configuración `squid.conf`. Este puerto debe ser accesible desde Internet o al menos desde los servidores de API Gateway.

2. Configuración de Squid

Crea el archivo `squid.conf` en el mismo directorio de `docker-compose.yml` con la siguiente

configuración:

```
# Configuración de autenticación.
auth_param basic program /usr/lib/squid/basic_ncsa_auth /etc/squid/passwords
auth_param basic realm Proxy

# Definir ACL para usuarios autenticados.
acl authenticated proxy_auth REQUIRED

# Definir ACL para dominios del SII (recomendado para seguridad).
acl sii_domains dstdomain .sii.cl

# AWS API Gateway (usado por eboleta)
acl eboleta_aws_api dstdomain .amazonaws.com

# Previred
acl previred_domains dstdomain .previred.com

# Permitir acceso solo a usuarios autenticados y solo a dominios del SII.
http_access allow authenticated sii_domains
http_access allow authenticated eboleta_aws_api
http_access allow authenticated previred_domains

# Denegar todas las demás solicitudes.
http_access deny all

# Escuchar en el puerto 3128.
http_port 3128
```

Nota de seguridad

La configuración anterior incluye una restricción que limita el uso del proxy únicamente a dominios del SII (`.sii.cl`), Previred (`.previred.com`) y eBoleta (`.amazonaws.com`). Esto es altamente recomendado para evitar el uso no autorizado del proxy para otros fines. Si necesitas acceder a otros dominios a través del proxy, puedes modificar o eliminar la línea `acl sii_domains dstdomain .sii.cl` y cambiar `http_access allow authenticated sii_domains` por `http_access allow authenticated`.

3. Creación de Credenciales

Genera el archivo de contraseñas usando el comando `htpasswd`:

```
htpasswd -c passwords <nombre_usuario> <contraseña>
```

Ejemplo:

```
htpasswd -c passwords admin mi_contraseña_segura
```

Recomendaciones de seguridad:

- Usa contraseñas fuertes (mínimo 12 caracteres).
- Combina mayúsculas, minúsculas, números y símbolos.
- Evita información personal en las credenciales.

4. Iniciar el Servicio

Ejecuta el siguiente comando para iniciar el proxy:

```
docker compose up -d
```

Verifica que el servicio esté funcionando correctamente:

```
docker compose ps
```

Habilitación en API Gateway

Una vez que tu proxy esté funcionando, necesitas configurarlo en API Gateway. Acá hay 2 formas de hacerlo dependiendo de qué versión de API Gateway estés usando:

- **Versión V2**: Acceder a la configuración de la conexión. Recuerda que en la versión 2 la configuración del proxy se debe realizar por cada conexión que tengas activa.
- **Versión Legacy (V1)**: Acceder a la configuración de tu perfil.

Independientemente de la versión de API Gateway que estés usando, debes configurar el proxy indicando su URL, la cual tendrá el siguiente formato:

```
http://<nombre_usuario>:<contraseña>@<ip_proxy>:3128
```

Donde los parámetros son:

- `<nombre_usuario>`: El usuario que creaste con `htpasswd`.
- `<contraseña>`: La contraseña correspondiente.
- `<ip_proxy>`: La dirección IP pública de tu servidor proxy.
- `3128`: El puerto de Squid (cambia si usaste uno diferente).

Realización de pruebas

Después de guardar la configuración, API Gateway comenzará a utilizar tu proxy para todas las consultas. Esto significa que las consultas se realizarán desde la IP de tu proxy en lugar de la IP

principal de API Gateway.

Para verificar que la configuración es correcta, realiza una consulta a la API y revisa la cabecera `X-Stats-HttpClientProxy` en la respuesta. Si su valor es `1`, significa que el proxy está siendo utilizado. Si el valor es `0`, el proxy no se está usando; en ese caso, revisa la sección de problemas comunes a continuación.

Problemas Comunes

1. Error de autenticación

- Verifica que las credenciales en API Gateway coincidan con las del archivo `passwords`.
- Confirma que el archivo `passwords` esté correctamente montado en el contenedor.

2. Proxy no accesible

- Verifica que el puerto 3128 esté abierto en el firewall.
- Confirma que la IP del proxy sea accesible desde Internet.
- Revisa los logs de Docker: `docker compose logs squid`.

3. Consultas lentas

- Monitorea el uso de recursos del servidor proxy.
- Considera aumentar los recursos asignados al contenedor.
- Verifica la latencia de red entre API Gateway y tu proxy.

Last updated on 21/08/2026

Extraer PEM con OpenSSL

Extraer el certificado y la clave privada con OpenSSL

Anonymous · 27/08/2026

Extraer el certificado y la clave privada con OpenSSL

Para autenticar tus peticiones a la API necesitas dos valores en formato PEM: el certificado (con su cadena de certificación) y la clave privada. Ambos salen del archivo `.p12` o `.pfx` de tu firma electrónica.

Puedes obtenerlos con la herramienta externa tools.libredte.cl o, si prefieres que tu firma no salga de tu computador, extraerlos tú con **OpenSSL**. Este tutorial cubre la segunda opción: todos los comandos se ejecutan localmente y tu archivo no se sube a ningún servicio.

Antes de empezar

Necesitas OpenSSL instalado y el archivo de tu firma con su contraseña.

```
openssl version
```

- **macOS**: viene incluido; si usas Homebrew, `brew install openssl`.
- **Linux (Debian/Ubuntu)**: `sudo apt install openssl`.
- **Windows**: usa Git Bash, WSL o los binarios de slproweb.com/products/Win32OpenSSL.html.

En los ejemplos el archivo de la firma se llama `firma.p12`. Reemplaza ese nombre por el tuyo. OpenSSL te pedirá la contraseña de la firma en cada comando.

Paso 1: extraer el certificado con su cadena

Este es el valor que enviarás como `cert-data`. La opción `-nokeys` excluye la clave privada y, al no usar `-clcerts`, la salida incluye tanto tu certificado como los de la autoridad certificadora (la cadena).

```
openssl pkcs12 -legacy -in firma.p12 -nokeys \
  | LC_ALL=C sed -n '/-----BEGIN/,/-----END/p' \
  | LC_ALL=C awk '{printf "%s%s", $0, "\\n"}' > cert-data.txt
```

El archivo `cert-data.txt` queda con el valor listo para copiar, en una sola línea. Debe contener dos o

más bloques `BEGIN CERTIFICATE`, en el orden correcto: primero tu certificado y después la cadena. Puedes confirmarlo con:

```
grep -o "BEGIN CERTIFICATE" cert-data.txt | wc -l
```

> **Importante:** envía el valor completo con todos sus bloques. Si mandas solo el primer certificado, la autenticación puede ser rechazada por falta de la cadena de certificación.

Paso 2: extraer la clave privada

Este es el valor que enviarás como `pkey-data`. La opción `-nocerts` excluye los certificados y `-nodes` deja la clave sin cifrar, que es como debe viajar en la petición.

```
openssl pkcs12 -legacy -in firma.p12 -nocerts -nodes \
  | LC_ALL=C sed -n '/-----BEGIN/,/-----END/p' \
  | LC_ALL=C awk '{printf "%s%s", $0, "\\n"}' > pkey-data.txt
```

`pkey-data.txt` debe empezar con `-----BEGIN PRIVATE KEY-----` o con `-----BEGIN RSA PRIVATE KEY-----`; la API acepta ambos encabezados.

Si la clave quedó cifrada

Si el archivo empieza con `-----BEGIN ENCRYPTED PRIVATE KEY-----`, la API la rechazará. Revisa primero que no hayas omitido la opción `-nodes`. Si el problema persiste, hay que descifrarla en un archivo intermedio antes de convertirla a una línea:

```
openssl pkcs12 -legacy -in firma.p12 -nocerts -nodes \
  | LC_ALL=C sed -n '/-----BEGIN/,/-----END/p' > pkey_temp.pem
openssl pkcs8 -topk8 -nocrypt -in pkey_temp.pem -out pkey_plano.pem
LC_ALL=C awk '{printf "%s%s", $0, "\\n"}' pkey_plano.pem > pkey-data.txt
rm pkey_temp.pem pkey_plano.pem
```

Paso 3: verificar lo que extrajiste

Como los archivos están en una sola línea, `openssl` no puede leerlos directamente: hay que devolver los saltos con un `awk` antes de pasárselos.

Confirma que el certificado es el que esperas y que está vigente:

```
LC_ALL=C awk '{gsub(/\\n/, "\n"); print}' cert-data.txt \
  | openssl x509 -noout -subject -issuer -dates
```

Verás el titular, la autoridad certificadora y las fechas de validez. Para comprobar que la clave privada corresponde a ese certificado, ambos módulos deben coincidir:

```
LC_ALL=C awk '{gsub(/\\n/, "\n"); print}' cert-data.txt | openssl x509 -noout -modulus
| openssl md5
LC_ALL=C awk '{gsub(/\\n/, "\n"); print}' pkey-data.txt | openssl rsa -noout -modulus
| openssl md5
```

Si los dos comandos devuelven el mismo hash, el par es correcto.

Paso 4: enviar los valores a la API

Los dos valores viajan en el bloque `auth.cert` del cuerpo JSON, en cada petición:

```
{
  "auth": {
    "cert": {
      "cert-data": "-----BEGIN CERTIFICATE-----\n...",
      "pkey-data": "-----BEGIN PRIVATE KEY-----\n..."
    }
  }
}
```

Abre `cert-data.txt` y `pkey-data.txt`, y copia el contenido completo de cada uno en el campo que corresponde. Son valores de una sola línea, así que no necesitas escaparlos ni reformatearlos: sirven igual escribiendo el JSON a mano (Postman, Swagger, curl) que armando la petición por código.

Qué hace cada parte del comando

Los comandos de los pasos 1 y 2 encadenan tres piezas. No necesitas entenderlas para seguir el tutorial, pero saber qué hace cada una te ayudará si algo falla.

La opción `-legacy`

Las firmas electrónicas chilenas vienen cifradas con algoritmos antiguos (RC2 de 40 bits, SHA-1) que OpenSSL 3.x desactivó por defecto, así que sin esa opción los comandos fallan con un error como este:

```
Error outputting keys and certificates
...:error:0308010C:digital envelope routines:inner_evp_generic_fetch:unsupported:
crypto/evp/evp_fetch.c:375:Global default library context,
Algorithm (RC2-40-CBC : 0), Properties ()
```

Si usas OpenSSL 1.1.x, la opción `-legacy` no existe y tampoco hace falta: quítala de los comandos y funcionarán igual.

El filtro de las líneas `Bag Attributes`

OpenSSL no entrega el PEM limpio: antes de cada bloque intercala metadatos como `Bag Attributes`, `localKeyID`, `subject=` e `issuer=`. Se ven así:

```
Bag Attributes
    localKeyID: 87 F1 CE 34 18 B5 C4 41 A9 9C 20 9E FC 39 D6 08 67 FC B8 0A
subject=CN=Titular Test
issuer=CN=CA Test
-----BEGIN CERTIFICATE-----
```

Esas líneas **deben quedar fuera** de lo que envíes a la API. No es un detalle estético: la API valida que el valor empiece exactamente con `-----BEGIN CERTIFICATE-----` y, si no, rechaza la petición con el error *"El certificado no tiene el formato PEM correcto"*.

`openssl pkcs12` no tiene ninguna opción para omitir esos metadatos, así que los comandos pasan la salida por `sed`, que conserva únicamente lo que va desde `-----BEGIN` hasta `-----END`.

El prefijo `LC_ALL=C` que verás en esos comandos hace que `sed` trate la entrada como bytes sueltos. Es necesario porque los metadatos incluyen el nombre del titular y, si lleva acentos o ñ en una codificación distinta de UTF-8, el `sed` de macOS y de los BSD aborta con `RE error: illegal byte sequence`. En Linux con GNU `sed` el prefijo es inofensivo, así que puedes usar el mismo comando en cualquier sistema.

Si trabajas en Windows sin `sed` (fuera de Git Bash o WSL), ejecuta el comando sin la tubería y luego borra a mano, en un editor de texto, todo lo que esté antes de la primera línea `-----BEGIN`.

La conversión a una línea

La API no recibe el PEM multilínea tal cual: espera el valor en **una sola línea**, con los saltos representados como los dos caracteres `\` y `n`. Así se ve el resultado:

```
-----BEGIN CERTIFICATE-----\nMIIC7zCCAdegAwIBAgIUC3UApaj1...\n-----END CERTIFICATE----
-\n
```

Por eso los comandos terminan con un `awk` que hace esa conversión: cada uno genera un único archivo, ya listo para pegar sin retoques, tanto si armas la petición por código como si escribes el JSON a mano.

Si OpenSSL falla

- **`Mac verify error: invalid password?`** — la contraseña no corresponde al archivo. Es la causa más frecuente; revísala antes de buscar otra explicación.
- **`unsupported`** o **`RC2-40-CBC`** — falta la opción `-legacy` en el comando.
- **`sed: RE error: illegal byte sequence`** — falta el prefijo `LC_ALL=C` antes de `sed`. Ocurre en macOS y BSD cuando el nombre del titular en los metadatos del archivo trae acentos o ñ en una codificación distinta de UTF-8.
- **`No such file or directory`** — revisa la ruta del archivo. Si el nombre tiene espacios, enciérralo en comillas.

`could not load the shared library` al usar `-legacy`

El módulo del proveedor `legacy` no está disponible. Comprueba si tu instalación lo tiene:

```
openssl list -providers -provider legacy
```

Si responde `status: active`, el proveedor está bien y el problema es otro. Si falla, el módulo no está instalado o el `openssl` de tu `PATH` no es el mismo al que pertenecen los módulos instalados, algo habitual cuando hay más de un OpenSSL en el sistema. Revisa dónde los busca con:

```
openssl version -m
```

Como alternativa, en vez de `-legacy` puedes cargar los proveedores de forma explícita:

```
openssl pkcs12 -provider legacy -provider default -in firma.p12 -nokeys \
  | LC_ALL=C sed -n '/-----BEGIN/,/-----END/p' \
  | LC_ALL=C awk '{printf "%s%s", $0, "\\n"}' > cert-data.txt
```

Tienen que ir los dos: si indicas solo `-provider legacy`, OpenSSL deja de cargar el proveedor por defecto y falla con `Error verifying PKCS12 MAC; no PKCS12KDF support`.

Cuida tu clave privada

- Es una credencial secreta: quien la tenga puede firmar en tu nombre.
- No la subas a repositorios ni la pegues en canales de chat o tickets.
- Guárdala en variables de entorno o en un gestor de secretos, nunca en el código fuente.
- El archivo `pkey-data.txt` queda **sin cifrar** en tu disco. Bórralo cuando ya no lo necesites, o restringe sus permisos.

Last updated on 21/08/2026