# Caddy

# Bloquear países en Caddy con MaxMind

## Introducción

Bloqueo de IPs por territorios directamente en Caddy v2, sin depender de Cloudflare. Esto corta accesos no deseados antes de que lleguen al backend, devolviendo un 403 en la misma capa del proxy.

En este caso, el ejemplo corresponde a lo que he aplicado en mi propia configuración: se usa para aquellos dominios que no están protegidos por Cloudflare, como capa extra de defensa junto con CrowdSec.

## Características

- Se aplica en Caddy antes de Authentik, file_server o reverse_proxy.
- Snippet reutilizable en cualquier dominio.
- Actualización automática de bases de MaxMind.
- Permite bypass para LAN/loopback y CGNAT.
- Refuerzo adicional en dominios sin Cloudflare.

## Requisitos previos

- Caddy compilado con plugin `github.com/porech/caddy-maxmind-geolocation` usando `xcaddy`.
- Cuenta gratuita en MaxMind con licencia.
- Paquete `geoipupdate` instalado y configurado.

## Desarrollo / pasos

### 1. Instalar MaxMind

Instalación de la herramienta:

```
sudo apt install geoipupdate
```

Actualizar manualmente la base de datos:

```
sudo geoipupdate -v
```

El archivo `GeoIP.conf` se obtiene al generar una clave en la web de MaxMind. En mi caso, borré el contenido original y lo sustituí por el archivo que me dio MaxMind directamente al crear la clave.

Las bases se descargan en:

```
/var/lib/GeoIP/GeoLite2-{Country,City,ASN}.mmdb
```

# 2. Verificar ruta en Caddy

El matcher usará la base de países:

```
/var/lib/GeoIP/GeoLite2-Country.mmdb
```

Caddy no necesita reinicio al actualizar el archivo.

# 3. Snippet de geobloqueo

```
(geoblock_matcher) {
	@geoblock {
		# No geobloquees LAN/CGNAT/loopback
		not remote_ip private_ranges 127.0.0.1 ::1

		# Bloquea si NO está en la lista permitida
		not maxmind_geolocation {
			db_path "/var/lib/GeoIP/GeoLite2-Country.mmdb"
			allow_countries
			AD AL AM AT AX BA BE BG BY CH CY CZ DE DK EE ES FI FO FR
  			GB GI GR HR HU IE IM IS IT JE LI LT LU LV MC MD ME MK MT NL NO PL PT RO
			RS SE SI SK SM UA VA XK
			CA GL PM US AG AI AW BB BL BM BS BZ CR CU CW DM DO GD GP GT HN HT JM KN
			KY LC MF MQ MS NI PA PR SX SV TC TT VC VG VI MX
			AR BO BR CL CO EC FK GF GY PE PY SR UY VE AQ
		}
	}
```

```
    respond @geoblock 403
}
```

### 4. Uso en un sitio

```
midominio.tld {
    import tls_estricto
    import seguridad_basica
    import log_json_global
    import geoblock_matcher
    route {
        reverse_proxy 192.168.1.3:8080
    }
}
```

---

## Errores comunes o decisiones importantes

- Los `@matchers` no pueden ir en global, solo dentro de un sitio o importados con snippet.
- El plugin no soporta continentes, solo países.
- Sin la exclusión `remote_ip private_ranges`, se bloquea la LAN.
- Validar siempre con `caddy validate` antes de recargar.
- Si el dominio va tras un proxy/CDN, configurar `servers { trusted_proxies ... }` para ver la IP real.

---

## Resumen breve

- Caddy con plugin MaxMind.
- Bases MMDB en `/var/lib/GeoIP/`.
- Snippet `geoblock_matcher` con lista de países permitidos.
- Importar snippet dentro de cada sitio.
- Validar y probar con VPN de otros países.
- En mi caso, aplicado solo a dominios sin Cloudflare, junto con CrowdSec.

---

## Referencias

- Repositorio del plugin caddy-maxmind-geolocation
- Usar binario de Caddy customizado

- Documentación oficial MaxMind GeoIP2

# Caddyfile con variables de entorno

Para evitar dejar tokens y credenciales a la vista en el `Caddyfile`, se pueden cargar como variables de entorno desde systemd y luego referenciarlas dentro de la configuración.

## Definir variables en systemd

Se añaden en drop-ins bajo `/etc/systemd/system/caddy.service.d/`:

```
# /etc/systemd/system/caddy.service.d/cloudflare.conf
[Service]
Environment=CLOUDFLARE_AUTH_TOKEN=A_ZYXS-LjFdbxxxxxxx42cmb7KH-iM8xxxxxxxxx

# /etc/systemd/system/caddy.service.d/openwebui.conf
[Service]
Environment="OPENWEBUI_APP_TOKEN=V9APMi3w..."
```

Después:

```
systemctl daemon-reload
systemctl restart caddy
```

## Usar las variables en el Caddyfile

Caddy permite referenciar cualquier variable de entorno con `{env.VARIABLE}`:

```
# Opciones globales
{
  admin 127.0.0.1:2019
  email admin@example.org
  acme_dns cloudflare {env.CLOUDFLARE_AUTH_TOKEN}
```

```
}

# Matcher compuesto: cabecera + path de API o WS
@conduit_combined {
  header X-App-Token {env.OPENWEBUI_APP_TOKEN}
  path /api/* /ws*
}
```

---

# Validar cambios antes de reiniciar

Para asegurarse de que el `Caddyfile` es válido y que las variables se pasan correctamente:

```
caddy fmt Caddyfile --overwrite

sudo env CLOUDFLARE_AUTH_TOKEN=$CLOUDFLARE_AUTH_TOKEN \
           OPENWEBUI_APP_TOKEN=$OPENWEBUI_APP_TOKEN \
           caddy validate /etc/caddy/Caddyfile
```

Si se usan otras variables, deben adaptarse al comando anterior.

---

# Verificar que systemd exporta las variables

```
systemctl show caddy | grep Environment
```

Deberían aparecer las variables declaradas en los drop-ins. Si no, revisar el nombre del archivo y volver a recargar/reiniciar.

---

# Resumen

- Variables sensibles → en drop-ins de systemd, no en el Caddyfile.
- Referencia en Caddyfile → `{env.NOMBRE}`.
- Recargar y reiniciar para aplicar cambios.
- Validar con `caddy fmt` + `caddy validate` pasando las variables al entorno.
- Comprobar con `systemctl show` si se están cargando.

# Directivas de Caddy (uso personal)

Apunte rápido para tener recogidas las directivas que uso hoy en día. Si añado más en el futuro, irán aquí.

## Recordatorio: cómo importar snippets

En Caddy los snippets se definen entre paréntesis y luego se importan con `import`. **Ventaja**: evitan repetir configuraciones y mantienen el Caddyfile más limpio. Si un snippet cambia, la modificación se aplica automáticamente en todos los sitios donde se importe.

Ejemplo usando la directiva más corta (TLS estricto), con un sitio definido para mostrar la estructura completa:

```
(tls_estricto) {
    tls {
        protocols tls1.3
    }
}

# Ejemplo ficticio de uso de imports
blog.ejemplo.org {
    import tls_estricto
    import seguridad_basica
    import log_json_global

    reverse_proxy http://10.0.0.5:8080 {
        import ip_headers
    }
}
```

### Authentik

Snippet para preautenticación y proxy de assets. Asegura que todas las rutas pasen por Authentik y copia claims útiles.

```
(authentik) {
    reverse_proxy /outpost.goauthentik.io/* http://IP_INTERNO_AUTHENTIK:PUERTO

    forward_auth http://IP_INTERNO_AUTHENTIK:PUERTO {
        uri /outpost.goauthentik.io/auth/caddy

        copy_headers {
            X-Authentik-Username
            X-Authentik-Email
            X-Authentik-Groups
            X-Authentik-Name
            X-Authentik-Uid
        }

        header_up X-Real-IP {client_ip}
        header_up X-Client-IP {client_ip}
    }
}
```

---

## TLS estricto

Snippet para forzar que todo vaya con TLS 1.3. Evita negociar versiones anteriores.

```
(tls_estricto) {
    tls {
        protocols tls1.3
    }
}
```

---

## Seguridad básica

Cabeceras básicas de seguridad que suelo meter para endurecer la configuración.

```
(seguridad_basica) {
    header {
        X-Frame-Options "SAMEORIGIN"
```

```
        X-Content-Type-Options "nosniff"
        Referrer-Policy "strict-origin-when-cross-origin"
        Strict-Transport-Security "max-age=31536000; includeSubDomains; preload"
    }
}
```

- **X-Frame-Options: SAMEORIGIN** → Evita que embeban la web en iframes externos.
- **X-Content-Type-Options: nosniff** → Bloquea detección automática de tipo.
- **Referrer-Policy: strict-origin-when-cross-origin** → Controla qué se envía en Referer.
- **Strict-Transport-Security** → Obliga HTTPS y permite preload en navegadores.

---

## IP real en headers

Útil para que los servicios backend reciban la IP del cliente en lugar de la del proxy.

```
(ip_headers) {
    header_up X-Real-IP {client_ip}
    header_up X-Client-IP {client_ip}
    header_up -X-Forwarded-Port
    header_up X-Forwarded-Port 443
}
```

---

## Registro de logs en JSON

Genera logs en formato JSON. En mi caso lo uso para CrowdSec, pero puede adaptarse a cualquier otra necesidad (centralizar logs, enviar a un SIEM, etc.).

```
(log_json_global) {
    log {
        output file /var/log/caddy/crowdsec-global.json
        format json
        level INFO
    }
}
```

---

# Errores comunes

- Poner `import` fuera del bloque del sitio → no funciona, siempre debe ir dentro.

- Reutilizar el mismo nombre en dos snippets → Caddy solo acepta uno, el último pisa al anterior.
- Olvidar importar un snippet en un sitio concreto → el sitio queda sin la configuración esperada (ej. sin TLS estricto o sin logs).
- Colocar `import` después de una directiva que dependa de él → puede dar lugar a comportamientos inesperados.

# Implementación de Cloudflare en Caddy

Caddy gestiona certificados HTTPS de forma automática, pero si usas **Cloudflare como DNS**, puedes mejorar la seguridad y ocultar la IP de tu servidor. Para eso, puedes configurar el **ACME DNS Challenge**, que permite validar tus dominios usando Cloudflare en lugar del típico challenge HTTP.

---

## Requisitos previos

- Dominio gestionado en Cloudflare.
- Caddy instalado.
- Un API Token de Cloudflare con permisos DNS sobre tu dominio.

---

### Paso 1: Crear el API Token en Cloudflare

1. Accede a Cloudflare Dashboard.
2. En tu perfil, entra a **API Tokens**.
3. Crea un token con:
   - **Zone:DNS:Edit** (limitado al dominio en cuestión).
4. Guarda el token en un sitio seguro.

---

### Paso 2: Añadir el token al entorno

Guarda el token como variable de entorno para evitar escribirlo directamente en el `Caddyfile`.

```
# En ~/.bashrc o ~/.zshrc
export CLOUDFLARE_AUTH_TOKEN=tu_token_aqui
source ~/.bashrc  # o ~/.zshrc
```

---

### Paso 3: Configurar el `Caddyfile`

Bloque global con soporte para DNS Challenge usando Cloudflare:

```
{
    acme_dns cloudflare
    email admin@example.com
}

example.com {
    reverse_proxy http://192.168.1.10:8080
}

subdomain.example.com {
    reverse_proxy http://192.168.1.10:8081
}
```

> El token se toma de la variable de entorno `CLOUDFLARE_AUTH_TOKEN`.

---

## Paso 4: Validar y recargar Caddy

```
caddy validate --config /etc/caddy/Caddyfile
sudo systemctl reload caddy
```

---

# Beneficios de usar Cloudflare con Caddy

- Ocultas la IP real del servidor.
- Añades mitigación contra ataques DDoS.
- Puedes aprovechar el cacheo de contenido estático.

---

# Detalles adicionales

- **Modo SSL en Cloudflare**: activa "Full (Strict)" para que Cloudflare acepte los certificados de Caddy.
- **Compilar Caddy con soporte DNS personalizado**: si no usas un binario precompilado, consulta esta guía: Usar binario de Caddy customizado