Servidor - Boot Report por Telegram (Ubuntu + systemd)

Boot health report · anti-duplicados · backoff · hardening · troubleshooting

Guía operativa + script listo para copiar

Contenido

0. Resumen ejecutivo

Un **reporte técnico al arranque** (boot) enviado por Telegram, pensado para servers Ubuntu con systemd. La idea es que **un reboot deje evidencia**: "arrancó" y "arrancó sano".

En términos operativos (nivel empresa):

- **Reducción de incertidumbre**: confirmás que el server levantó y con qué estado.
- **Detección temprana**: si el boot arranca degradado, te enterás al instante.

Incluye (en el mensaje):

- Red: IP privada + IP pública (con fallback y retry)
- Salud: CPU load normalizado por cores, RAM, Disco, Inodos, Temperatura (best-effort)
- Señales de riesgo: unidades systemd fallidas (si existieran), updates pendientes
- Modo "solo alertas" (opcional)

Y a nivel operación:

- Retries con backoff para red/DNS/Telegram
- systemd hardening (baseline + extras opcionales)
- Logs claros en `journalctl` con tags: `BOOT_REPORT_OK / FAIL / SKIP`

Ruta recomendada (10 minutos)

- **Instalación rápida (Git):** ver **3.1**.
- **Configurar credenciales (Telegram):** ver **5**.
- **Probar sin reiniciar:** ver **4.1**.
- **Validar con reboot:** ver **4.2**.

Resultados esperados

- Tras cada reboot, recibís un mensaje Telegram con el estado del server.
- Si la red tarda en levantar, el script reintenta y evita falsos negativos.
- Si el servicio se dispara 2 veces, no duplica mensajes.
- Si Telegram falla, systemd reintenta (Restart=on-failure) sin "spamear".
- Los logs quedan en `journalctl -u boot-report.service`.

1. Diseño

1.1. Componentes

- `/opt/boot-report/.env` - secretos + configuración (propietario: bootreport)
- `/opt/boot-report/boot-report.sh` - script principal (oneshot)
- `/etc/systemd/system/boot-report.service` - unit systemd

1.2. Flujo (alto nivel)

1. systemd espera `network-online.target`
2. crea `/run/boot-report/` (RuntimeDirectory)
3. ejecuta `boot-report.sh` como usuario dedicado
4. el script toma lock (flock) + chequea "stamp" (anti-duplicados)
5. recolecta métricas
6. valida conectividad mínima (ruta + DNS)
7. envía Telegram (con retry/backoff y validación `"ok":true`)
8. escribe "stamp" solo si el envío fue OK
9. loguea OK/FAIL/SKIP a journal

1.3. Anti-duplicados (lo importante)

- **Lock**: evita concurrencia (dos ejecuciones simultáneas).
- **Stamp**: evita que un restart re-envíe el mismo boot dentro de una ventana de tiempo.

Esto resuelve el problema típico de locks en `/run` cuando el servicio corre con usuario no-root: systemd crea un runtime dir con permisos correctos.

1.4. Señales, umbrales y acción sugerida

Señal	Qué significa	WARN / CRIT	Acción sugerida
CPU load %	Load normalizado por cores	70 / 100	Revisar procesos + top/htop; si persiste, plan de capacity
RAM %	Memoria usada	70 / 85	Identificar leak; revisar cache/buffers; swap
Disco % (/)	Uso de filesystem raíz	70 / 85	Limpiar logs/caché; crecer disco; revisar rotación
Inodos % (/)	Cantidad de archivos vs inodos	70 / 85	Buscar directorios con muchos files; rotación; limpiar tmp
Temp (°C)	Temperatura (best-effort)	70 / 85	Revisar ventilación/carga; si N/D, validar sensores
systemd failed	Units en failed	>0	`systemctl --failed` + `journalctl -xe`
Updates	Paquetes pendientes	>0	Programar ventana de mantenimiento; aplicar parches

2. Requisitos

2.1. Paquetes

Código (bash)

```
sudo apt update
sudo apt install -y curl ca-certificates lm-sensors
```

2.2. (Opcional) detección de sensores

Código (bash)

```
sudo sensors-detect --auto || true
```

Temperatura
La temperatura es "best effort". Si no hay sensores, el reporte imprime TEMP=N/D.

2.3. Recomendado: wait-online real

`network-online.target` depende de un servicio "wait-online". Activá el que corresponda:

- Si usás NetworkManager:

Código (bash)

```
sudo systemctl enable --now NetworkManager-wait-online.service
```

- Si usás systemd-networkd:

Código (bash)

```
sudo systemctl enable --now systemd-networkd-wait-online.service
```

Antes de empezar (Telegram)
Si todavía no tenés `BOT_TOKEN` y `CHAT_ID`, saltá a la sección **5**. Para avanzar con la instalación podés dejar placeholders en `.env` y completarlos al final.

3. Instalación

Elegí un camino:

- **Camino A (Git) - recomendado:** más reproducible y fácil de actualizar.
- **Camino B (manual):** copy/paste, sin depender de git en el servidor.

3.1. Camino A - Instalación vía Git (recomendada)

Camino A (Git) - recomendado para producción

- Instalación reproducible (mismo layout, permisos y unit).
- Actualización simple: `git pull` + `install.sh` (no pisa tu `.env`).
- Mejor auditoría: cambios quedan registrados en commits.

Repositorio (Boot):

3.1.1. Requisitos

Código (bash)

```
sudo apt update
sudo apt install -y git curl
```

(Optativo: temperatura)

sudo apt install -y lm-sensors sudo sensors-detect --auto || true

3.1.2. Clonar e instalar

Código (bash)

```
cd ~
git clone https://github.com/lucasborges2001/Boot.git boot-report
cd boot-report
sudo ./scripts/install.sh
```

3.1.3. Configurar credenciales (Telegram)

Código (bash)

```
sudo nano /opt/boot-report/.env
```

Obligatorios:

- `CHAT_ID`

3.1.4. Probar y diagnosticar

Código (bash)

```
sudo systemctl start boot-report.service
journalctl -u boot-report.service -b --no-pager
```

3.1.5. Actualizar (pull + reinstalar)

Código (bash)

```
cd ~/boot-report
git pull
sudo ./scripts/install.sh
```

Idempotencia

El instalador **no pisa** `/opt/boot-report/.env` si ya existe.

3.2. Camino B - Instalación manual (copy/paste)

Nota

Si vas por este camino, asegurate de completar `BOT_TOKEN` y `CHAT_ID` (ver **5**).

3.2.1. Crear estructura

Código (bash)

```
sudo mkdir -p /opt/boot-report
sudo chmod 755 /opt/boot-report
```

3.2.2. Crear usuario dedicado

Código (bash)

```
sudo useradd -r -s /usr/sbin/nologin bootreport || true
```

3.2.3. Crear archivo de entorno (secreto + config)

Código (bash)

```
sudo nano /opt/boot-report/.env
```

Contenido mínimo:

Código (bash)

```
BOT_TOKEN="PEGAR_TOKEN_TELEGRAM"
CHAT_ID="PEGAR_CHAT_ID"
SERVER_LABEL="prod-web-01"
```

Opcionales recomendados:

Código (bash)

```
# Si es true: si NO hay issues, NO envía mensaje.
BOOT_ALERTS_ONLY="false"
# Emojis en Telegram (true/false)
BOOT_EMOJI="true"
# Ventana anti-duplicados (segundos) para "stamp" post-éxito
LOCK_TTL_SEC="900"
# Thresholds (porcentaje)
WARN_LOAD_PCT="70"
CRIT_LOAD_PCT="100"
WARN_RAM_PCT="70"
CRIT_RAM_PCT="85"
WARN_DISK_PCT="70"
CRIT_DISK_PCT="85"
# Thresholds (temperatura en °C)
WARN_TEMP_C="70"
CRIT_TEMP_C="85"
```

Permisos:

Código (bash)

```
sudo chown bootreport:bootreport /opt/boot-report/.env
sudo chmod 640 /opt/boot-report/.env
```

Seguridad del token

No publiques `BOT_TOKEN`. Es equivalente a credencial de escritura del bot.

Variable	Default	Qué controla	Ejemplo
BOT_TOKEN	(obligatorio)	Token del bot	"123:ABC..."
CHAT_ID	(obligatorio)	Chat destino (user o grupo)	"123456" / "-100..."
SERVER_LABEL	hostname	Etiqueta humana del server	"prod-db-02"
BOOT_ALERTS_ONLY	false	Enviar solo si hay alertas	true
BOOT_EMOJI	true	Emojis en mensaje	false

LOCK_TTL_SEC	900	Ventana anti-duplicados	600
WARN_ / *CRIT_*	ver arriba	Thresholds	85

3.2.4. Script principal: boot-report.sh

Código (bash)

```
sudo nano /opt/boot-report/boot-report.sh
sudo chown bootreport:bootreport /opt/boot-report/boot-report.sh
sudo chmod 750 /opt/boot-report/boot-report.sh
```

Pegar el script:

Código (bash)

```
#!/usr/bin/env bash
set -euo pipefail
ENV_FILE="/opt/boot-report/.env"
log_ok() { echo "BOOT_REPORT_OK $*"; }
log_fail() { echo "BOOT_REPORT_FAIL $*"; }
# --- Utils ----------------------------------------------------------------
---
retry_backoff() {
 local tries="${1:-6}"; shift
 local i=1 delay=1
 while (( i <= tries )); do
 if "$@"; then return 0; fi
 sleep "$delay"
 delay=$((delay*2)); (( delay > 24 )) && delay=24
 i=$((i+1))
 done
 return 1
}
curl_quiet() { curl -fsS --max-time 7 "$@"; }
escape_html() {
 # Escapa &, <, > para Telegram parse_mode=HTML
 sed -e 's/&/\&amp;/g' -e 's/</\&lt;/g' -e 's/>/\&gt;/g'
}
badge_pct() {
 # echo: "OK" | "WARN" | "CRIT" (con o sin emoji)
 local val="$1" warn="$2" crit="$3"
 local ok="OK" wa="WARN" cr="CRIT"
 if [[ "${BOOT_EMOJI:-true}" == "true" ]]; then
 ok="OK"; wa="WARN"; cr="CRIT"
 fi
 if (( val < warn )); then echo "$ok"
 elif (( val < crit )); then echo "$wa"
 else echo "$cr"
```

```
 fi
}
badge_temp() {
 local val="$1" warn="${WARN_TEMP_C:-70}" crit="${CRIT_TEMP_C:-85}"
 badge_pct "$val" "$warn" "$crit"
}
net_ready() {
 # Ruta + DNS: suficiente para evitar "network-online" falso.
 ip route get 1.1.1.1 >/dev/null 2>&1 || return 1
 getent hosts api.telegram.org >/dev/null 2>&1 || return 1
}
get_ip_pub() {
 local ip
 ip="$(curl_quiet https://api.ipify.org || true)"
 [[ -z "$ip" ]] && ip="$(curl_quiet ifconfig.me || true)"
 [[ -n "$ip" ]] || return 1
 echo "$ip"
}
send_telegram() {
 # Requiere: BOT_TOKEN, CHAT_ID, MSG (HTML)
 local resp
 resp="$(
 curl -fsS --max-time 12 -X POST \
 "https://api.telegram.org/bot${BOT_TOKEN}/sendMessage" \
 -d chat_id="${CHAT_ID}" \
 -d parse_mode="HTML" \
 -d disable_web_page_preview="true" \
 --data-urlencode text="${MSG}"
 )" || return 1
 # Telegram puede responder 200 pero ok=false; validamos.
 grep -q '"ok"[[:space:]]*:[[:space:]]*true' <<<"$resp"
}
# --- Load env ------------------------------------------------------------
---
if [[ ! -f "$ENV_FILE" ]]; then
 log_fail "missing_env_file=$ENV_FILE"
 exit 1
fi
# shellcheck disable=SC1090
set -a
source "$ENV_FILE"
set +a
: "${BOT_TOKEN:?missing BOT_TOKEN in $ENV_FILE}"
: "${CHAT_ID:?missing CHAT_ID in $ENV_FILE}"
SERVER_LABEL="${SERVER_LABEL:-$(hostname)}"
# --- Anti-duplicados -----------------------------------------------------
----
RUNDIR="${BOOT_REPORT_RUNDIR:-/run/boot-report}"
LOCK_TTL_SEC="${LOCK_TTL_SEC:-900}"
# Si se corre fuera de systemd con RuntimeDirectory, intentamos fallback.
```

```
if [[ ! -d "$RUNDIR" || ! -w "$RUNDIR" ]]; then
 RUNDIR="/tmp/boot-report"
 mkdir -p "$RUNDIR" 2>/dev/null || true
fi
LOCK_FILE="$RUNDIR/lock"
SENT_STAMP="$RUNDIR/sent"
exec 9>"$LOCK_FILE"
if ! flock -n 9; then
 echo "BOOT_REPORT_SKIP already_running"
 exit 0
fi
now_epoch="$(date +%s)"
if [[ -f "$SENT_STAMP" ]]; then
 last_epoch="$(cat "$SENT_STAMP" 2>/dev/null || echo 0)"
 if [[ "$last_epoch" =~ ^[0-9]+$ ]] && (( now_epoch - last_epoch < LOCK_TTL_
SEC )); then
 echo "BOOT_REPORT_SKIP already_sent age=$((now_epoch-last_epoch))s"
 exit 0
 fi
fi
# --- Recolectar datos ----------------------------------------------------
-----
HOST="$(hostname)"
HOST_ESC="$(printf '%s' "$HOST" | escape_html)"
LABEL_ESC="$(printf '%s' "$SERVER_LABEL" | escape_html)"
UPTIME="$(uptime -p 2>/dev/null || uptime)"
UPTIME_ESC="$(printf '%s' "$UPTIME" | escape_html)"
KERNEL="$(uname -r)"
KERNEL_ESC="$(printf '%s' "$KERNEL" | escape_html)"
DATE_TXT="$(date)"
DATE_ESC="$(printf '%s' "$DATE_TXT" | escape_html)"
IP_PRIV="$(hostname -I 2>/dev/null | awk '{print $1}' || true)"
[[ -n "$IP_PRIV" ]] || IP_PRIV="N/D"
IP_PRIV_ESC="$(printf '%s' "$IP_PRIV" | escape_html)"
# Espera corta por red "real"
retry_backoff 6 net_ready >/dev/null 2>&1 || true
IP_PUB="N/D"
IP_PUB="$(retry_backoff 6 get_ip_pub || echo "N/D")"
IP_PUB_ESC="$(printf '%s' "$IP_PUB" | escape_html)"
CORES="$(nproc 2>/dev/null || echo 1)"
LOAD="$(awk '{print $1}' /proc/loadavg 2>/dev/null || echo 0)"
LOAD_PCT="$(awk -v l="$LOAD" -v c="$CORES" 'BEGIN{ if(c<=0)c=1; printf "%.0f
", (l/c)*100 }')"
RAM_USED_PCT="$(free 2>/dev/null | awk '/Mem:/ {printf "%.0f", $3/$2*100}' |
| echo 0)"
DISK_USED_PCT="$(df / 2>/dev/null | awk 'NR==2 {gsub("%","",$5); print $5}'
|| echo 0)"
INODE_USED_PCT="$(df -i / 2>/dev/null | awk 'NR==2 {gsub("%","",$5); print $
5}' || echo 0)"
UPDATES="$(apt list --upgradable 2>/dev/null | grep -vc Listing || true)"
```

```
TEMP_TXT="N/D"
TEMP_BADGE="N/D"
if command -v sensors >/dev/null 2>&1; then
 t="$(sensors 2>/dev/null | awk '
 /Package id 0:/ {gsub(/[+°C]/,"",$4); print int($4); exit}
 /Tctl:/ {gsub(/[+°C]/,"",$2); print int($2); exit}
 ' || true)"
 if [[ -n "${t:-}" ]]; then
 TEMP_TXT="${t}°C"
 TEMP_BADGE="$(badge_temp "$t")"
 fi
fi
FAILED_UNITS_COUNT="N/D"
FAILED_UNITS_PREVIEW=""
if systemctl --failed --no-legend --plain >/dev/null 2>&1; then
 FAILED_UNITS_COUNT="$(systemctl --failed --no-legend --plain | wc -l | awk
'{print $1}')"
 # Preview: primeras 3 unit names
 FAILED_UNITS_PREVIEW="$(systemctl --failed --no-legend --plain | awk '{prin
t $1}' | head -n 3 | paste -sd ', ' -)"
fi
NTP_SYNC="N/D"
if command -v timedatectl >/dev/null 2>&1; then
 NTP_SYNC="$(timedatectl show -p NTPSynchronized --value 2>/dev/null || echo
 N/D)"
fi
# --- Detectar issues ----------------------------------------------------
----
WARN_LOAD_PCT="${WARN_LOAD_PCT:-70}"
CRIT_LOAD_PCT="${CRIT_LOAD_PCT:-100}"
WARN_RAM_PCT="${WARN_RAM_PCT:-70}"
CRIT_RAM_PCT="${CRIT_RAM_PCT:-85}"
WARN_DISK_PCT="${WARN_DISK_PCT:-70}"
CRIT_DISK_PCT="${CRIT_DISK_PCT:-85}"
LOAD_BADGE="$(badge_pct "$LOAD_PCT" "$WARN_LOAD_PCT" "$CRIT_LOAD_PCT")"
RAM_BADGE="$(badge_pct "$RAM_USED_PCT" "$WARN_RAM_PCT" "$CRIT_RAM_PCT")"
DISK_BADGE="$(badge_pct "$DISK_USED_PCT" "$WARN_DISK_PCT" "$CRIT_DISK_PCT")"
INODE_BADGE="$(badge_pct "$INODE_USED_PCT" "$WARN_DISK_PCT" "$CRIT_DISK_PCT"
)"
HAS_ISSUES="false"
ISSUES=()
(( LOAD_PCT >= CRIT_LOAD_PCT )) && HAS_ISSUES="true" && ISSUES+=("CPU")
(( RAM_USED_PCT >= CRIT_RAM_PCT )) && HAS_ISSUES="true" && ISSUES+=("RAM")
(( DISK_USED_PCT >= CRIT_DISK_PCT )) && HAS_ISSUES="true" && ISSUES+=("DISK"
)
(( INODE_USED_PCT >= CRIT_DISK_PCT )) && HAS_ISSUES="true" && ISSUES+=("INOD
ES")
if [[ "$FAILED_UNITS_COUNT" != "N/D" && "$FAILED_UNITS_COUNT" -gt 0 ]]; then
 HAS_ISSUES="true"
 ISSUES+=("SYSTEMD_FAILED")
```

```
fi
TITLE="Servidor iniciado"
if [[ "$HAS_ISSUES" == "true" ]]; then
 TITLE="ALERTA: servidor iniciado con issues"
fi
TITLE_ESC="$(printf '%s' "$TITLE" | escape_html)"
# Solo alertas (opcional)
if [[ "${BOOT_ALERTS_ONLY:-false}" == "true" && "$HAS_ISSUES" != "true" ]];
then
 echo "BOOT_REPORT_SKIP no_issues alerts_only=true"
 exit 0
fi
ISSUES_TXT="none"
if [[ "${#ISSUES[@]}" -gt 0 ]]; then
 ISSUES_TXT="$(printf '%s' "${ISSUES[*]}")"
fi
ISSUES_ESC="$(printf '%s' "$ISSUES_TXT" | escape_html)"
FAILED_LINE=""
if [[ "$FAILED_UNITS_COUNT" != "N/D" && "$FAILED_UNITS_COUNT" -gt 0 ]]; then
 preview_esc="$(printf '%s' "$FAILED_UNITS_PREVIEW" | escape_html)"
 FAILED_LINE="• <b>systemd failed:</b> <code>${FAILED_UNITS_COUNT}</code> ($
{preview_esc})"
fi
# --- Mensaje ----------------------------------------------------------
---
MSG="$(cat <<EOF
<b>${TITLE_ESC}</b>
<b>Servidor:</b> ${LABEL_ESC}
<b>Host:</b> ${HOST_ESC}
<b>Uptime:</b> ${UPTIME_ESC}
<b>Red</b>
• Privada: <code>${IP_PRIV_ESC}</code>
• Pública: <code>${IP_PUB_ESC}</code>
<b>Salud</b>
• CPU load: ${LOAD_BADGE} <code>${LOAD}</code> (${LOAD_PCT}% / ${CORES} core
s)
• RAM: ${RAM_BADGE} <code>${RAM_USED_PCT}%</code>
• Disco (/): ${DISK_BADGE} <code>${DISK_USED_PCT}%</code>
• Inodos (/): ${INODE_BADGE} <code>${INODE_USED_PCT}%</code>
• Temp: <code>${TEMP_TXT}</code> ${TEMP_BADGE}
• Updates: <code>${UPDATES}</code>
${FAILED_LINE}
<b>Estado</b>
• Issues: <code>${ISSUES_ESC}</code>
• NTP sync: <code>${NTP_SYNC}</code>
<b>Sistema</b>
• Kernel: <code>${KERNEL_ESC}</code>
• Fecha: <code>${DATE_ESC}</code>
EOF
)"
```

```
# --- Envío ----------------------------------------------------------------
---
if retry_backoff 6 send_telegram; then
 printf '%s\n' "$now_epoch" > "$SENT_STAMP" 2>/dev/null || true
 log_ok "sent=1 issues=${HAS_ISSUES} list=${ISSUES_TXT} disk=${DISK_USED_PCT
}% ram=${RAM_USED_PCT}% load_pct=${LOAD_PCT}%"
 exit 0
else
 log_fail "sent=0 issues=${HAS_ISSUES} list=${ISSUES_TXT} reason=telegram_se
nd_failed"
 exit 1
fi
```

3.2.5. Unit systemd: boot-report.service

Código (bash)

```
sudo nano /etc/systemd/system/boot-report.service
```

Contenido:

Código (ini)

```
[Unit]
Description=Telegram Boot Report
After=network-online.target
Wants=network-online.target
# Evita que systemd "se rinda" rápido si hubo varios fallos (opcional):
StartLimitIntervalSec=0
[Service]
Type=oneshot
User=bootreport
Group=bootreport
ExecStart=/opt/boot-report/boot-report.sh
TimeoutStartSec=240
# Runtime dir en /run con permisos correctos para user no-root
RuntimeDirectory=boot-report
RuntimeDirectoryMode=0750
# Reintento si falla Telegram/red
Restart=on-failure
RestartSec=30
# Hardening (baseline compatible con curl + lectura de sistema)
NoNewPrivileges=true
PrivateTmp=true
ProtectSystem=strict
ProtectHome=true
ProtectKernelTunables=true
ProtectKernelModules=true
ProtectControlGroups=true
LockPersonality=true
```

```
MemoryDenyWriteExecute=true
RestrictSUIDSGID=true
RestrictNamespaces=true
RestrictRealtime=true
RestrictAddressFamilies=AF_INET AF_INET6 AF_UNIX
UMask=0077
SystemCallArchitectures=native
SystemCallFilter=@system-service @network-io
# Opcional: si tu version de systemd lo soporta (y no rompe sensors)
# ProtectKernelLogs=true
# ProtectProc=invisible
# ProcSubset=pid
# PrivateDevices=true
[Install]
WantedBy=multi-user.target
```

Activación:

Código (bash)

```
sudo systemctl daemon-reload
sudo systemctl enable boot-report.service
```

4. Pruebas (sin reiniciar y con reinicio)

4.1. Prueba manual

Código (bash)

```
sudo systemctl start boot-report.service
journalctl -u boot-report.service -n 120 --no-pager
```

4.2. Prueba final (reboot)

Código (bash)

```
sudo reboot
```

5. Token y CHAT_ID (Telegram)

5.1. Crear bot y token

- En Telegram abrir @BotFather
- /newbot
- guardar BOT_TOKEN

5.2. Obtener CHAT_ID (chat privado)

1 Mandale un mensaje al bot ("hola")

2 En el servidor:

Código (bash)

```
curl -s "https://api.telegram.org/bot<BOT_TOKEN>/getUpdates"
```

Buscar:

Código (json)

```
"chat":{"id":123456789, ...}
```

Ese número es el CHAT_ID.

Grupos

Si el destino es un grupo, el chat_id suele ser negativo y el bot debe estar agregado.

6. Operación

6.1. Ver logs

Código (bash)

```
journalctl -u boot-report.service -n 200 --no-pager
```

Tags típicos:

- `BOOT_REPORT_FAIL ...`
- `BOOT_REPORT_SKIP ...`

6.2. Cambiar label del servidor

Código (bash)

```
sudo nano /opt/boot-report/.env
# SERVER_LABEL="prod-db-02"
sudo systemctl restart boot-report.service
```

7. Troubleshooting (con rutas de diagnóstico)

No llega Telegram

Checklist rápido:

1. Logs:

Código (bash)

```
journalctl -u boot-report.service -n 200 --no-pager
```

1. Red/DNS desde el server:

Código (bash)

```
ip route get 1.1.1.1
getent hosts api.telegram.org
curl -I https://api.telegram.org
```

1. Token y chat:

- Revisar `BOT_TOKEN` y `CHAT_ID`
- Si es grupo, `CHAT_ID` negativo y bot agregado

IP pública N/D

- Puede fallar por DNS o salida bloqueada
- El script reintenta con backoff; si persiste, revisar firewall/routing/DNS

El servicio corre 2 veces / duplica mensajes

- usa `RuntimeDirectory=/run/boot-report` + `flock` + `stamp` post-éxito.
- Si querés ampliar la ventana anti-duplicados: subir `LOCK_TTL_SEC` (ej. 1800).

8. Checklist de auditoría (final)

Checklist

- /opt/boot-report/.env existe (640) y dueño bootreport
- /opt/boot-report/boot-report.sh existe (750) y dueño bootreport
- boot-report.service enabled
- RuntimeDirectory activo: /run/boot-report/ existe post-boot
- Mensaje llega manualmente (systemctl start)
- Mensaje llega post-reboot
- Logs OK/FAIL/SKIP visibles en journalctl

9. Desinstalación / rollback

9.1. Detener y deshabilitar el servicio

Código (bash)

```
sudo systemctl disable --now boot-report.service
sudo systemctl daemon-reload
```

9.2. Remover archivos (deja logs en journal)

Código (bash)

```
sudo rm -f /etc/systemd/system/boot-report.service
sudo rm -rf /opt/boot-report
```

9.3. (Opcional) Remover el usuario dedicado

Código (bash)

```
sudo userdel bootreport 2>/dev/null || true
```

Nota

Si estás usando el **Camino A (Git)**, el repositorio clonado en tu home (ej. `~/boot-report`) no se borra con los pasos anteriores.