Servidor - Docker Monitoring + Alertas (Compose + Telegram)

docker compose · docker events watcher · healthchecks · restart controlado · systemd hardening

Guía operativa (runbook) + scripts listos para copiar

Contenido

0. Resumen ejecutivo **4**

1. Qué incluye **5**

2. Diseño (decisiones + flujo) **6**

2.1. Flujo de eventos 6

2.2. Matriz de comportamiento 6

2.3. Restart policy vs watcher 6

3. Requisitos **8**

3.1. Paquetes 8

3.2. Usuario de ejecución (dedicado) 8

4. Instalación **9**

4.1. Camino A - Instalación vía Git (recomendada) 9

4.2. Camino B - Instalación manual (copy/paste) 9

4.2.1. Crear estructura 9

4.2.2. Archivo de entorno (secreto + config) 10

4.2.3. Script: docker-watch.sh 11

4.2.4. systemd service: docker-watch.service 18

5. Validación (sin tocar producción) **21**

5.1. Test "start" y "die" (exit code != 0) 21

5.2. Test "unhealthy" 21

5.3. Verificar logs del watcher 21

6. Heartbeat diario (opcional, recomendado) **22**

6.1. Script: docker-heartbeat.sh 22

6.2. Unit + Timer 23

7. Operación **25**

7.1. Ver estado 25

7.2. Logs y healthcheck 25

7.3. Logs del watcher 25

7.4. Ajustar "ruido" 25

8. Troubleshooting **26**

9. Checklist de auditoría (final) **30**

10. Desinstalación / rollback **31**

10.1. Detener y deshabilitar 31

10.2. Remover units y archivos 31

10.3. (Opcional) Remover usuario 31

0. Resumen ejecutivo

Este documento instala un **watcher de eventos Docker** que envía alertas por Telegram cuando un contenedor:

- inicia (start)
- se detiene o crashea (die)
- cambia de estado de salud (health_status: healthy/unhealthy)

En términos operativos (nivel empresa):

- **Mejora trazabilidad**: queda registro en Telegram + `journalctl`.
- **Evita "silencios"**: opcionalmente manda un heartbeat diario sin spam.

Política clave (diseño):

- **DIE => alerta + (opcional) intento controlado de start**, con cooldown y rate limit.

Ruta recomendada (15 minutos)

- Elegí instalación: **Git (recomendada)** en **4.1** o **Manual** en **4.2**.
- Configurá Telegram en `/opt/docker-watch/.env` (**4.2.2**).
- Activá el servicio: `systemctl enable --now docker-watch.service` (**4.2.4**).
- Probá con contenedores de test (sin tocar producción): **5**.
- (Opcional) activá heartbeat diario: **6**.

Riesgo del docker.sock

Acceder a `/var/run/docker.sock` equivale a privilegios muy altos (prácticamente root).
Este runbook mitiga con:

- hardening systemd
- filtrado por label (opcional)

1. Qué incluye

- `docker-watch.sh` (daemon):
- escucha `docker events` en tiempo real
- anti-spam por evento (TTL) + limpieza de estado
- mensajes Telegram robustos (HTML + escaping + validación de respuesta)
- restart controlado ante `die` (opcional)
- `docker-heartbeat.sh` (opcional):
- 1 mensaje diario con estado Docker + conteo de issues
- `docker-watch.service` + hardening systemd
- `docker-heartbeat.service` + `docker-heartbeat.timer` (si activás heartbeat)

2. Diseño (decisiones + flujo)

2.1. Flujo de eventos

1 `docker events` entrega eventos de contenedor

2 el watcher aplica:

- filtro por label (si está configurado)
- dedupe por TTL

1 arma mensaje con:

- servidor, contenedor, id corto
- stack/service de compose (si existen labels)
- exit code (si aplica)

1 envía Telegram con retry/backoff

2 (solo para `die`) decide si intenta `docker start` con rate limit

2.2. Matriz de comportamiento

Evento	Notifica	Acción automática	Recomendación
start	sí (configurable)	no	útil para despliegues; si hay mucho ruido, desactivar
die	sí	opcional: `docker start` (con cooldown y límite/h)	mantener activo para minimizar downtime
unhealthy	sí	no (solo alerta)	revisar logs + healthcheck; decidir manualmente
healthy	no (por default)	no	suele ser ruidoso; activar solo si es necesario

2.3. Restart policy vs watcher

- La **restart policy** de Docker/Compose es tu primera línea de defensa.
- El watcher agrega:
- visibilidad (alertas)
- "segundo intento" controlado (si decidís habilitarlo)

Nota: el watcher **no** reemplaza observabilidad completa (metrics, logs, tracing), pero cubre un 80/20 muy valioso para servidores con stacks Compose.

3. Requisitos

3.1. Paquetes

Código (bash)

```
sudo apt update
sudo apt install -y docker.io curl ca-certificates
sudo systemctl enable --now docker
```

3.2. Usuario de ejecución (dedicado)

Código (bash)

```
sudo useradd -r -s /usr/sbin/nologin dockwatch || true
sudo usermod -aG docker dockwatch
```

Logout/login

El grupo `docker` aplica tras re-login (o con `newgrp docker` si probás manual).

4. Instalación

Elegí un camino:

- **4.1 Git (recomendada):** trazabilidad + updates simples.
- **4.2 Manual (copy/paste):** sin depender de git ni instalador.

4.1. Camino A - Instalación vía Git (recomendada)

Qué asume este camino

Asume que tu repositorio incluye un `scripts/install.sh` que instala en `/opt/docker-watch` y registra units systemd. Si tu repo NO lo trae, usá el camino manual (4.2).

Código (bash)

```
sudo apt update
sudo apt install -y git curl
cd ~
git clone https://github.com/lucasborges2001/Docker.git
cd Docker
sudo ./scripts/install.sh
```

Luego configurá credenciales:

Código (bash)

```
sudo nano /opt/docker-watch/.env
sudo systemctl enable --now docker-watch.service
journalctl -u docker-watch.service -n 120 --no-pager
```

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

4.2.1. Crear estructura

Código (bash)

```
sudo mkdir -p /opt/docker-watch
sudo chmod 755 /opt/docker-watch
```

4.2.2. Archivo de entorno (secreto + config)

Código (bash)

```
sudo nano /opt/docker-watch/.env
```

Contenido mínimo:

Código (bash)

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

Opcionales recomendados:

Código (bash)

```
# Anti-spam: TTL (segundos) por tipo de evento
TTL_START_SEC="90"
TTL_DIE_SEC="60"
TTL_UNHEALTHY_SEC="180"
TTL_HEALTHY_SEC="600"
# Notificaciones (true/false)
NOTIFY_START="true"
NOTIFY_HEALTHY="false"
# Acción ante DIE (stop/crash)
AUTO_RESTART_ON_DIE="true"
RESTART_ONLY_ON_NONZERO_EXIT="true"
# Control de loops
RESTART_COOLDOWN_SEC="300"
MAX_RESTARTS_PER_HOUR="3"
# Filtrado (opcional): monitorear solo contenedores con label KEY=VALUE
# Ej: MONITOR_LABEL_KEY="monitoring" ; MONITOR_LABEL_VALUE="true"
MONITOR_LABEL_KEY=""
MONITOR_LABEL_VALUE=""
# Emojis en Telegram (true/false)
DOCKER_EMOJI="true"
```

Permisos:

Código (bash)

```
sudo chown dockwatch:dockwatch /opt/docker-watch/.env
sudo chmod 640 /opt/docker-watch/.env
```

Variable	Default	Qué controla
BOT_TOKEN / CHAT_ID	-	Destino Telegram
SERVER_LABEL	hostname	Nombre humano del servidor

TTL_*_SEC	ver arriba	Anti-spam por evento
NOTIFY_START	true	Avisar start
NOTIFY_HEALTHY	false	Avisar healthy (ruidoso)
AUTO_RESTART_ON_DIE	true	Intentar levantar si muere
RESTART_ONLY_ON_NONZERO_EXIT	true	No "revivir" paradas limpias (exit 0)
RESTART_COOLDOWN_SEC	300	Cooldown entre restarts
MAX_RESTARTS_PER_HOUR	3	Circuit breaker por contenedor
MONITOR_LABEL_*	vacío	Filtrar contenedores por label
DOCKER_EMOJI	true	Emojis en mensajes

4.2.3. Script: docker-watch.sh

Código (bash)

```
sudo nano /opt/docker-watch/docker-watch.sh
sudo chown dockwatch:dockwatch /opt/docker-watch/docker-watch.sh
sudo chmod 750 /opt/docker-watch/docker-watch.sh
```

Pegar:

Código (bash)

```
#!/usr/bin/env bash
set -euo pipefail
ENV_FILE="/opt/docker-watch/.env"
log_ok()   { echo "DOCKER_WATCH_OK $*"; }
log_fail() { echo "DOCKER_WATCH_FAIL $*"; }
retry_backoff() {
  local tries="${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 10 "$@"; }
escape_html() {
```

```
  sed -e 's/&/\&amp;/g' -e 's/</\&lt;/g' -e 's/>/\&gt;/g'
}
bool_is_true() { [[ "${1:-false}" == "true" ]]; }
emoji_on() { bool_is_true "${DOCKER_EMOJI:-true}"; }
# --- 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:-$(hostname)}"
# --- Runtime / state --------------------------------------------------------
----
# systemd crea /run/docker-watch via RuntimeDirectory; fallback si se ejecut
a manual.
STATE_DIR="/run/docker-watch"
if [[ ! -d "$STATE_DIR" || ! -w "$STATE_DIR" ]]; then
  STATE_DIR="/tmp/docker-watch"
  mkdir -p "$STATE_DIR" 2>/dev/null || true
fi
LOCK_FILE="$STATE_DIR/lock"
exec 9>"$LOCK_FILE"
flock -n 9 || { echo "DOCKER_WATCH_SKIP already_running"; exit 0; }
# TTLs
TTL_START_SEC="${TTL_START_SEC:-90}"
TTL_DIE_SEC="${TTL_DIE_SEC:-60}"
TTL_UNHEALTHY_SEC="${TTL_UNHEALTHY_SEC:-180}"
TTL_HEALTHY_SEC="${TTL_HEALTHY_SEC:-600}"
NOTIFY_START="${NOTIFY_START:-true}"
NOTIFY_HEALTHY="${NOTIFY_HEALTHY:-false}"
AUTO_RESTART_ON_DIE="${AUTO_RESTART_ON_DIE:-true}"
RESTART_ONLY_ON_NONZERO_EXIT="${RESTART_ONLY_ON_NONZERO_EXIT:-true}"
RESTART_COOLDOWN_SEC="${RESTART_COOLDOWN_SEC:-300}"
MAX_RESTARTS_PER_HOUR="${MAX_RESTARTS_PER_HOUR:-3}"
MONITOR_LABEL_KEY="${MONITOR_LABEL_KEY:-}"
MONITOR_LABEL_VALUE="${MONITOR_LABEL_VALUE:-}"
sanitize_key() {
  # shell-safe filename
  echo "$1" | tr -cs 'a-zA-Z0-9_.-@' '_'
}
should_send() {
  local key="$1" ttl="$2"
  local now; now="$(date +%s)"
  local f="$STATE_DIR/ttl_$(sanitize_key "$key")"
```

```
  if [[ -f "$f" ]]; then
    local last; last="$(cat "$f" 2>/dev/null || echo 0)"
    if [[ "$last" =~ ^[0-9]+$ ]] && (( now - last < ttl )); then
      return 1
    fi
  fi
  printf '%s\n' "$now" > "$f" 2>/dev/null || true
  return 0
}
cleanup_state() {
  # Limpieza simple: evita acumulación si el server vive meses.
  find "$STATE_DIR" -type f -name 'ttl_*' -mmin +4320 2>/dev/null | head -n
200 | xargs -r rm -f || true
  find "$STATE_DIR" -type f -name 'rst_*' -mmin +4320 2>/dev/null | head -n
200 | xargs -r rm -f || true
}
restart_allowed() {
  # Circuit breaker por contenedor (cid key)
  local cid_key="$1"
  local now; now="$(date +%s)"
  local lastf="$STATE_DIR/rst_last_${cid_key}"
  if [[ -f "$lastf" ]]; then
    local last; last="$(cat "$lastf" 2>/dev/null || echo 0)"
    if [[ "$last" =~ ^[0-9]+$ ]] && (( now - last < RESTART_COOLDOWN_SEC ));
 then
      return 1
    fi
  fi
  local listf="$STATE_DIR/rst_list_${cid_key}"
  touch "$listf" 2>/dev/null || true
  # Mantener solo timestamps de la última hora
  awk -v now="$now" '($1 ~ /^[0-9]+$/) && (now-$1 < 3600) {print $1}' "$list
f" 2>/dev/null > "${listf}.tmp" || true
  mv -f "${listf}.tmp" "$listf" 2>/dev/null || true
  local count; count="$(wc -l < "$listf" 2>/dev/null || echo 0)"
  if [[ "$count" =~ ^[0-9]+$ ]] && (( count >= MAX_RESTARTS_PER_HOUR )); the
n
    return 1
  fi
  printf '%s\n' "$now" >> "$listf" 2>/dev/null || true
  printf '%s\n' "$now" > "$lastf" 2>/dev/null || true
  return 0
}
send_telegram_html() {
  local msg="$1"
  local resp
  resp="$(
    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="${msg}"
  )" || return 1
  grep -q '"ok"[[:space:]]*:[[:space:]]*true' <<<"$resp"
}
match_filter() {
  # Filtra por label KEY=VALUE si está configurado (vacío => acepta todo).
  local label_k="$1" label_v="$2" want_k="$3" want_v="$4"
  [[ -z "$want_k" ]] && return 0
  [[ "$label_k" == "$want_k" && "$label_v" == "$want_v" ]] && return 0
  return 1
}
fmt_header() {
  local title="$1" label="$2"
  local esc_title esc_label esc_srv
  esc_title="$(printf '%s' "$title" | escape_html)"
  esc_label="$(printf '%s' "$label" | escape_html)"
  esc_srv="$(printf '%s' "$SERVER_LABEL" | escape_html)"
  printf '<b>%s</b>\n\n<b>Servidor:</b> %s\n<b>Contenedor:</b> %s\n' "$esc_t
itle" "$esc_srv" "$esc_label"
}
icon_ok="OK"
icon_warn="WARN"
icon_crit="ALERT"
if emoji_on; then
  icon_ok="OK"
  icon_warn="WARN"
  icon_crit="CRIT"
fi
log_ok "starting=1 state_dir=$STATE_DIR"
# --- Event loop ------------------------------------------------------------
----
docker events \
  --filter type=container \
  --filter event=start \
  --filter event=die \
  --filter event=health_status \
  --format '{{.Time}}||{{.Action}}||{{.Actor.Attributes.name}}||{{.Actor.ID}
}||{{.Actor.Attributes.exitCode}}||{{index .Actor.Attributes "com.docker.com
pose.project"}}||{{index .Actor.Attributes "com.docker.compose.service"}}||{
{index .Actor.Attributes "com.docker.compose.project.working_dir"}}||{{index
 .Actor.Attributes "'${MONITOR_LABEL_KEY:-}'"}}' \
| while IFS='||' read -r ts action name cid exitcode proj svc wdir label_val
; do
    cleanup_state
    name="${name:-unknown}"
    action="${action:-unknown}"
    cid="${cid:-}"
    short="${cid:0:12}"
```

```
    # Filtrado por label (si aplica)
    if [[ -n "${MONITOR_LABEL_KEY:-}" ]]; then
      if ! match_filter "${MONITOR_LABEL_KEY}" "${label_val:-}" "${MONITOR_L
ABEL_KEY}" "${MONITOR_LABEL_VALUE}"; then
        continue
      fi
    fi
    # Campos UX (compose)
    proj="${proj:-}"
    svc="${svc:-}"
    wdir="${wdir:-}"
    esc_name="$(printf '%s' "$name" | escape_html)"
    esc_short="$(printf '%s' "$short" | escape_html)"
    esc_proj="$(printf '%s' "$proj" | escape_html)"
    esc_svc="$(printf '%s' "$svc" | escape_html)"
    esc_ts="$(date --date="@$ts" "+%Y-%m-%d %H:%M:%S" 2>/dev/null | escape_h
tml || date | escape_html)"
    # Key por evento + contenedor (evita colisiones por recreación)
    cid_key="$(sanitize_key "${name}_${short}")"
    case "$action" in
      start)
        bool_is_true "$NOTIFY_START" || continue
        should_send "start_${cid_key}" "$TTL_START_SEC" || continue
        stack_line=""
        [[ -n "$proj" || -n "$svc" ]] && stack_line="• <b>Stack:</b> <code>$
{esc_proj}</code> · <b>Svc:</b> <code>${esc_svc}</code>"
        msg="$(cat <<MSG
$(fmt_header "${icon_ok} Container START" "$esc_name")
• <b>ID:</b> <code>${esc_short}</code>
${stack_line}
• <b>Hora:</b> <code>${esc_ts}</code>
MSG
)"
        retry_backoff 4 send_telegram_html "$msg" >/dev/null 2>&1 || true
        ;;
      die)
        should_send "die_${cid_key}" "$TTL_DIE_SEC" || continue
        esc_exit="$(printf '%s' "${exitcode:-N/D}" | escape_html)"
        stack_line=""
        [[ -n "$proj" || -n "$svc" ]] && stack_line="• <b>Stack:</b> <code>$
{esc_proj}</code> · <b>Svc:</b> <code>${esc_svc}</code>"
        base="$(cat <<MSG
$(fmt_header "${icon_crit} Container DIE" "$esc_name")
• <b>ID:</b> <code>${esc_short}</code>
• <b>ExitCode:</b> <code>${esc_exit}</code>
${stack_line}
• <b>Hora:</b> <code>${esc_ts}</code>
MSG
)"
        # Acción controlada: restart SOLO en die (stop/crash)
```

```
        do_restart="false"
        if bool_is_true "$AUTO_RESTART_ON_DIE"; then
          if bool_is_true "$RESTART_ONLY_ON_NONZERO_EXIT"; then
            [[ "${exitcode:-}" != "0" ]] && do_restart="true"
          else
            do_restart="true"
          fi
        fi
        # Primero avisamos el evento
        retry_backoff 4 send_telegram_html "$base" >/dev/null 2>&1 || true
        if [[ "$do_restart" == "true" ]]; then
          # Espera breve: si hay restart policy, puede volver solo.
          sleep 2
          running="$(docker inspect -f '{{.State.Running}}' "$cid" 2>/dev/nu
ll || echo false)"
          if [[ "$running" == "true" ]]; then
            msg="$(cat <<MSG
$(fmt_header "${icon_ok} Container RECOVERED" "$esc_name")
• <b>ID:</b> <code>${esc_short}</code>
• <b>Nota:</b> ya está <code>running</code> (restart policy u acción manual)
• <b>Hora:</b> <code>$(date | escape_html)</code>
MSG
)"
            retry_backoff 4 send_telegram_html "$msg" >/dev/null 2>&1 || tru
e
          else
            if restart_allowed "$cid_key"; then
              msg="$(cat <<MSG
$(fmt_header "${icon_warn} Attempt START after DIE" "$esc_name")
• <b>ID:</b> <code>${esc_short}</code>
• <b>Acción:</b> <code>docker start</code>
MSG
)"
              retry_backoff 4 send_telegram_html "$msg" >/dev/null 2>&1 || t
rue
              if docker start "$cid" >/dev/null 2>&1; then
                okmsg="$(cat <<MSG
$(fmt_header "${icon_ok} START executed" "$esc_name")
• <b>ID:</b> <code>${esc_short}</code>
• <b>Resultado:</b> comando ejecutado (verificar estabilidad)
• <b>Hora:</b> <code>$(date | escape_html)</code>
MSG
)"
                retry_backoff 4 send_telegram_html "$okmsg" >/dev/null 2>&1
|| true
              else
                failmsg="$(cat <<MSG
$(fmt_header "${icon_crit} START failed" "$esc_name")
• <b>ID:</b> <code>${esc_short}</code>
• <b>Resultado:</b> no se pudo iniciar
```

```
• <b>Sugerencia:</b> revisar logs: <code>docker logs ${esc_name} --tail 200<
/code>
MSG
)"
                retry_backoff 4 send_telegram_html "$failmsg" >/dev/null 2>&
1 || true
              fi
            else
              blockmsg="$(cat <<MSG
$(fmt_header "${icon_crit} Restart blocked (rate limit)" "$esc_name")
• <b>ID:</b> <code>${esc_short}</code>
• <b>Motivo:</b> cooldown o límite por hora alcanzado
• <b>Acción:</b> intervención requerida
MSG
)"
              retry_backoff 4 send_telegram_html "$blockmsg" >/dev/null 2>&1
 || true
            fi
          fi
        fi
        ;;
      health_status:unhealthy)
        should_send "unhealthy_${cid_key}" "$TTL_UNHEALTHY_SEC" || continue
        stack_line=""
        [[ -n "$proj" || -n "$svc" ]] && stack_line="• <b>Stack:</b> <code>$
{esc_proj}</code> · <b>Svc:</b> <code>${esc_svc}</code>"
        msg="$(cat <<MSG
$(fmt_header "${icon_warn} Container UNHEALTHY" "$esc_name")
• <b>ID:</b> <code>${esc_short}</code>
${stack_line}
• <b>Acción:</b> solo alerta (sin restart automático)
• <b>Sugerencia:</b> <code>docker ps</code> · <code>docker logs ${esc_name}
--tail 200</code>
• <b>Hora:</b> <code>${esc_ts}</code>
MSG
)"
        retry_backoff 4 send_telegram_html "$msg" >/dev/null 2>&1 || true
        ;;
      health_status:healthy)
        bool_is_true "$NOTIFY_HEALTHY" || continue
        should_send "healthy_${cid_key}" "$TTL_HEALTHY_SEC" || continue
        stack_line=""
        [[ -n "$proj" || -n "$svc" ]] && stack_line="• <b>Stack:</b> <code>$
{esc_proj}</code> · <b>Svc:</b> <code>${esc_svc}</code>"
        msg="$(cat <<MSG
$(fmt_header "${icon_ok} Container HEALTHY" "$esc_name")
• <b>ID:</b> <code>${esc_short}</code>
${stack_line}
• <b>Hora:</b> <code>${esc_ts}</code>
MSG
```

```
)"
        retry_backoff 4 send_telegram_html "$msg" >/dev/null 2>&1 || true
        ;;
      *)
        # Ignorar otros eventos para evitar ruido.
        ;;
    esac
  done
```

4.2.4. systemd service: docker-watch.service

Código (bash)

```
sudo nano /etc/systemd/system/docker-watch.service
```

Contenido:

Código (ini)

```
[Unit]
Description=Docker Events Watcher -> Telegram
After=docker.service network-online.target
Wants=docker.service network-online.target
[Service]
Type=simple
User=dockwatch
Group=dockwatch
ExecStart=/opt/docker-watch/docker-watch.sh
Restart=always
RestartSec=5
TimeoutStartSec=60
# /run/docker-watch con permisos correctos (usuario no-root)
RuntimeDirectory=docker-watch
RuntimeDirectoryMode=0750
# Hardening baseline
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
# Solo lo necesario para escribir state + hablar con el socket Docker
ReadWritePaths=/run/docker-watch /var/run/docker.sock
[Install]
WantedBy=multi-user.target
```

Activación:

Código (bash)

```
sudo systemctl daemon-reload
sudo systemctl enable --now docker-watch.service
journalctl -u docker-watch.service -n 120 --no-pager
```

docker.sock path

Si tu socket no está en `/var/run/docker.sock`, verificar con:

Código (bash)

```
ls -l /var/run/docker.sock /run/docker.sock 2>/dev/null || true
```

y ajustar `ReadWritePaths` en el unit.

5. Validación (sin tocar producción)

La forma más segura es probar con contenedores descartables.

5.1. Test "start" y "die" (exit code != 0)

Código (bash)

```
docker run -d --name dw-test alpine sh -c "sleep 9999"
docker kill -s SIGKILL dw-test   # genera die con exit code no-cero
```

Esperado:

- si `AUTO_RESTART_ON_DIE=true` y no está bloqueado por rate limit: intento de `docker start`

Cleanup:

Código (bash)

```
docker rm -f dw-test || true
```

5.2. Test "unhealthy"

Código (bash)

```
docker run -d --name dw-hc \
  --health-cmd='exit 1' --health-interval=5s --health-retries=1 \
  alpine sh -c "sleep 9999"
```

Esperado:

- **sin** reinicio automático

Cleanup:

Código (bash)

```
docker rm -f dw-hc || true
```

5.3. Verificar logs del watcher

Código (bash)

```
journalctl -u docker-watch.service -n 200 --no-pager
```

6. Heartbeat diario (opcional, recomendado)

6.1. Script: docker-heartbeat.sh

Código (bash)

```
sudo nano /opt/docker-watch/docker-heartbeat.sh
sudo chown dockwatch:dockwatch /opt/docker-watch/docker-heartbeat.sh
sudo chmod 750 /opt/docker-watch/docker-heartbeat.sh
```

Contenido:

Código (bash)

```
#!/usr/bin/env bash
set -euo pipefail
source /opt/docker-watch/.env
SERVER_LABEL="${SERVER_LABEL:-$(hostname)}"
escape_html() { sed -e 's/&/\&amp;/g' -e 's/</\&lt;/g' -e 's/>/\&gt;/g'; }
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
}
send_telegram_html() {
  local msg="$1"
  local resp
  resp="$(
    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="${msg}"
  )" || return 1
  grep -q '"ok"[[:space:]]*:[[:space:]]*true' <<<"$resp"
}
docker_ok="true"
systemctl is-active docker >/dev/null 2>&1 || docker_ok="false"
# Issues: exited / restarting / unhealthy
issues_list="$(
  docker ps -a --format '{{.Names}}||{{.Status}}' 2>/dev/null | awk -F'||' '
    {
      st=tolower($2);
      if (st ~ /^exited/ || st ~ /^restarting/ || st ~ /unhealthy/) {
```

```
        c++;
        if (c<=5) { printf "%s (%s)\n",$1,$2; }
      }
    }
    END { }
  ' || true
)"
issues_count="$(
  docker ps -a --format '{{.Status}}' 2>/dev/null | awk '
    { st=tolower($0); if (st ~ /^exited/ || st ~ /^restarting/ || st ~ /unhe
althy/) c++ }
    END{ print c+0 }
  ' || echo 0
)"
title="OK Heartbeat diario"
if [[ "$docker_ok" != "true" || "$issues_count" -gt 0 ]]; then
  title="WARN Heartbeat con alertas"
fi
esc_title="$(printf '%s' "$title" | escape_html)"
esc_srv="$(printf '%s' "$SERVER_LABEL" | escape_html)"
esc_issues="$(printf '%s' "$issues_count" | escape_html)"
list_block=""
if [[ -n "$issues_list" ]]; then
  list_block="<b>Top issues:</b>\n<pre>$(printf '%s' "$issues_list" | escape
_html)</pre>"
fi
msg="$(cat <<MSG
<b>${esc_title}</b>
<b>Servidor:</b> ${esc_srv}
<b>Docker:</b> <code>${docker_ok}</code>
<b>Issues:</b> <code>${esc_issues}</code>
${list_block}
<b>Hora:</b> <code>$(date | escape_html)</code>
MSG
)"
retry_backoff 4 send_telegram_html "$msg" >/dev/null 2>&1 || true
```

6.2. Unit + Timer

Código (bash)

```
sudo nano /etc/systemd/system/docker-heartbeat.service
```

Código (ini)

```ini
[Unit]
Description=Daily Docker Heartbeat -> Telegram
[Service]
Type=oneshot
User=dockwatch
Group=dockwatch
ExecStart=/opt/docker-watch/docker-heartbeat.sh
NoNewPrivileges=true
PrivateTmp=true
ProtectSystem=strict
ProtectHome=true
RestrictAddressFamilies=AF_INET AF_INET6 AF_UNIX
UMask=0077
ReadWritePaths=/var/run/docker.sock
```

Código (bash)

```bash
sudo nano /etc/systemd/system/docker-heartbeat.timer
```

Código (ini)

```ini
[Unit]
Description=Daily Docker Heartbeat Timer
[Timer]
OnCalendar=daily
Persistent=true
RandomizedDelaySec=600
[Install]
WantedBy=timers.target
```

Activar:

Código (bash)

```bash
sudo systemctl daemon-reload
sudo systemctl enable --now docker-heartbeat.timer
systemctl list-timers | grep docker-heartbeat || true
```

7. Operación

7.1. Ver estado

Código (bash)

```
docker ps
docker ps -a
docker inspect <container> --format '{{.State.Status}}'
```

7.2. Logs y healthcheck

Código (bash)

```
docker logs <container> --tail 200
docker inspect <container> --format '{{json .State.Health}}' | head
```

7.3. Logs del watcher

Código (bash)

```
journalctl -u docker-watch.service -n 200 --no-pager
```

7.4. Ajustar "ruido"

- Subir `TTL_*_SEC`
- `NOTIFY_START=false` si hay recreaciones frecuentes
- `NOTIFY_HEALTHY=false` (recomendado)

8. Troubleshooting

No llegan alertas

Checklist:

1. Logs del servicio:

Código (bash)

```
journalctl -u docker-watch.service -n 200 --no-pager
```

1. Socket Docker accesible:

Código (bash)

```
sudo -u dockwatch docker ps
ls -l /var/run/docker.sock
```

1. Red/DNS:

Código (bash)

```
getent hosts api.telegram.org
curl -I https://api.telegram.org
```

1. Credenciales:

- Revisar BOT_TOKEN y CHAT_ID en `/opt/docker-watch/.env`

UNHEALTHY no aparece

- Confirmar que el contenedor tiene HEALTHCHECK:
- Dockerfile: `HEALTHCHECK ...`
- Compose: `healthcheck: ...`

Restart bloqueado

- Subir `RESTART_COOLDOWN_SEC`
- Bajar `MAX_RESTARTS_PER_HOUR`
- O desactivar `AUTO_RESTART_ON_DIE` y depender solo de restart policy

9. Checklist de auditoría (final)

Checklist

- `/opt/docker-watch/.env` existe (640) y dueño dockwatch
- `/opt/docker-watch/docker-watch.sh` existe (750) y dueño dockwatch
- `docker-watch.service` enabled y running
- `/run/docker-watch/` existe post-start (RuntimeDirectory)
- Alertas llegan con tests de la sección 5
- Heartbeat timer (si aplica) activo: docker-heartbeat.timer
- Logs OK en: `journalctl -u docker-watch.service`

10. Desinstalación / rollback

10.1. Detener y deshabilitar

Código (bash)

```
sudo systemctl disable --now docker-watch.service
sudo systemctl disable --now docker-heartbeat.timer 2>/dev/null || true
sudo systemctl daemon-reload
```

10.2. Remover units y archivos

Código (bash)

```
sudo rm -f /etc/systemd/system/docker-watch.service
sudo rm -f /etc/systemd/system/docker-heartbeat.service
sudo rm -f /etc/systemd/system/docker-heartbeat.timer
sudo rm -rf /opt/docker-watch
```

10.3. (Opcional) Remover usuario

Código (bash)

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