Servidor - Docker Monitoring + Alertas (Compose + Telegram)

docker compose · restart policies · healthchecks · docker events watcher · telegram

Guía operativa + scripts listos para copiar

Contenido

0. Resumen ejecutivo

Watcher de eventos Docker que envía alertas por Telegram cuando un contenedor:

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

En términos operativos (nivel empresa):

- Evita silencios: opcionalmente manda un heartbeat diario sin spam.
- Minimiza downtime (opcional): ante `die` puede intentar un `docker start` controlado.

Política clave:

- `die` => alerta + **opcional** intento controlado de `start` (cooldown + rate limit).

Ruta recomendada (15 minutos)

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

Riesgo del docker.sock

Acceder a `/var/run/docker.sock` equivale a privilegios muy altos (prácticamente root). Mitigaciones recomendadas: usuario dedicado + hardening systemd + filtros por label si aplica.

1. Qué incluye

- `docker-watch.sh` (daemon): escucha `docker events`, dedupe por TTL, alertas Telegram y restart controlado ante `die` (opcional).
- `docker-heartbeat.sh` (opcional): 1 mensaje diario con estado Docker + conteo de issues.
- Units systemd: `docker-watch.service`, y (si activás) `docker-heartbeat.service` + `docker-heartbeat.timer`.

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
3. arma un mensaje con servidor + contenedor + labels Compose (si existen) + exit code (si aplica)
4. envía Telegram con retry/backoff
5. (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` (cooldown + límite/h)	mantener activo para minimizar downtime
unhealthy	sí	no (solo alerta)	revisar logs + healthcheck; decidir manualmente
healthy	no (default)	no	suele ser ruidoso; activar solo si es necesario

2.3. Restart policy vs watcher

- La restart policy de Docker/Compose es la primera línea de defensa.
- El watcher agrega visibilidad (alertas) y un segundo intento controlado (si lo habilitás).

Compose + `.env` (convención)

Si tu stack está en `/srv/stack`:

- `/srv/stack/docker/docker-compose.yml`

Ejecutá desde el directorio que contiene `.env`:

cd /srv/stack docker compose -f docker/docker-compose.yml up -d

Esto mejora consistencia y hace que labels de Compose estén presentes (mejor UX en alertas).

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
```

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

4. Instalación

4.1. Camino A - Instalación vía Git (recomendada)

Qué asume este camino:

- 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:
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:

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

Opcionales recomendados:

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 MONITOR_LABEL_KEY="" MONITOR_LABEL_VALUE=""

Emojis en Telegram (true/false) DOCKER_EMOJI="true"

Permisos:

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

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:

#!/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

Procedimiento

```
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/&/\&/g' -e 's/</\</g' -e 's/>/\>/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 ejecuta 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

Procedimiento

```
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

Procedimiento

```
if [[ "$last" =~ ^[0-9]+$ ]] && (( now - last < RESTART_COOLDOWN_SEC )); the
n
  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 $1}' "$listf" 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)); then

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="$1" local resp resp="$(

Procedimiento

```
  "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="$2" local t esc_title esc_label esc_srv esc_title="$(printf '%s' "$title" | escape_html)" esc_label="$(printf '%s' "$label" | escape_html)" esc_srv="$(printf '%s' "$SERVER_LABEL" | escape_html)" printf '%s\n\nServidor: %s\nContenedor: %s\n' "$esc_title" "$esc_srv" "$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.compose.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

Procedimiento

```
name="${name:-unknown}"
action="${action:-unknown}"
cid="${cid:-}"
short="${cid:0:12}"
```

Procedimiento

```
# Filtrado por label (si aplica)
if [[ -n "${MONITOR_LABEL_KEY:-}" ]]; then
  if ! match_filter "${MONITOR_LABEL_KEY}" "${label_val:-}" "${MONITOR_LABEL_KEY}" "${MONITOR_LABEL_VALUE}"; then
    continue
  fi
fi
```

Procedimiento

```
# Campos UX (compose)
proj="${proj:-}"
svc="${svc:-}"
wdir="${wdir:-}"
```

Procedimiento

```
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_html || date | escape_html)"
```

Procedimiento

```
# Key por evento + contenedor (evita colisiones por recreación)
cid_key="$(sanitize_key "${name}_${short}")"
```

Procedimiento

```
case "$action" in
  start)
    bool_is_true "$NOTIFY_START" || continue
    should_send "start_${cid_key}" "$TTL_START_SEC" || continue
```

Procedimiento

```
    stack_line=""
    [[ -n "$proj" || -n "$svc" ]] && stack_line="• <b>Stack:</b> <code>${esc
_proj}</code> · <b>Svc:</b> <code>${esc_svc}</code>"
```

Procedimiento

```
    msg="$(cat <<EOF
```

$(fmt_header "${icon_ok} Container START" "$esc_name")

${stack_line}

EOF)"

Procedimiento

```
    ;;
```

Procedimiento

```
  die)
    should_send "die_${cid_key}" "$TTL_DIE_SEC" || continue
```

Procedimiento

```
    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>"
```

Procedimiento

```
    base="$(cat <<EOF
```

$(fmt_header "${icon_crit} Container DIE" "$esc_name")

- ExitCode: <code>${esc_exit}</code>

${stack_line}

EOF)"

Procedimiento

```
    # 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
```

Procedimiento

```
    # Primero avisamos el evento
    retry_backoff 4 send_telegram_html "$base" >/dev/null 2>&1 || true
```

Procedimiento

```
    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/null |
| echo false)"
```

Procedimiento

```
      if [[ "$running" == "true" ]]; then
        # Ya volvió (restart policy u operador)
        msg="$(cat <<EOF
```

$(fmt_header "${icon_ok} Container RECOVERED" "$esc_name")

- Nota: ya está <code>running</code> (restart policy u acción manual)
- Hora: <code>$(date | escape_html)</code>

EOF)"

Procedimiento

```
      else
        if restart_allowed "$cid_key"; then
          msg="$(cat <<EOF
```

$(fmt_header "${icon_warn} Attempt START after DIE" "$esc_name")

- Acción: <code>docker start</code>

EOF)"

Procedimiento

```
          if docker start "$cid" >/dev/null 2>&1; then
            okmsg="$(cat <<EOF
```

$(fmt_header "${icon_ok} START executed" "$esc_name")

- Resultado: comando ejecutado (verificar estabilidad)
- Hora: <code>$(date | escape_html)</code>

EOF)"

Procedimiento

```
          else
            failmsg="$(cat <<EOF
```

$(fmt_header "${icon_crit} START failed" "$esc_name")

- Resultado: no se pudo iniciar
- Sugerencia: revisar logs: <code>docker logs ${esc_name} --tail 200</code>

EOF)"

Procedimiento

```
          fi
        else
          blockmsg="$(cat <<EOF
```

$(fmt_header "${icon_crit} Restart blocked (rate limit)" "$esc_name")

- Motivo: cooldown o límite por hora alcanzado
- Acción: intervención requerida

EOF)"

Procedimiento

```
        fi
      fi
    fi
    ;;
```

Procedimiento

```
  health_status:unhealthy)
    should_send "unhealthy_${cid_key}" "$TTL_UNHEALTHY_SEC" || continue
```

Procedimiento

```
    stack_line=""
    [[ -n "$proj" || -n "$svc" ]] && stack_line="• <b>Stack:</b> <code>${esc
_proj}</code> · <b>Svc:</b> <code>${esc_svc}</code>"
```

Procedimiento

```
    msg="$(cat <<EOF
```

$(fmt_header "${icon_warn} Container UNHEALTHY" "$esc_name")

${stack_line}

- Sugerencia: <code>docker ps</code> · <code>docker logs ${esc_name} --tail 200</code>
- Hora: <code>${esc_ts}</code>

EOF)"

Procedimiento

```
    ;;
```

Procedimiento

```
  health_status:healthy)
    bool_is_true "$NOTIFY_HEALTHY" || continue
    should_send "healthy_${cid_key}" "$TTL_HEALTHY_SEC" || continue
```

Procedimiento

```
    stack_line=""
    [[ -n "$proj" || -n "$svc" ]] && stack_line="• <b>Stack:</b> <code>${esc
_proj}</code> · <b>Svc:</b> <code>${esc_svc}</code>"
```

Procedimiento

```
    msg="$(cat <<EOF
```

$(fmt_header "${icon_ok} Container HEALTHY" "$esc_name")

${stack_line}

EOF)"

Procedimiento

```
    ;;
```

Procedimiento

```
  *)
    # 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:

[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:

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

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

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

y ajustar `ReadWritePaths` en el unit.

5. Validación (sin tocar producción)

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
```

Cleanup:

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: alerta `unhealthy` y **sin** reinicio automático.

Cleanup:

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:

#!/usr/bin/env bash set -euo pipefail

source /opt/docker-watch/.env SERVER_LABEL="${SERVER_LABEL:-$(hostname)}"

escape_html() { sed -e 's/&/\&/g' -e 's/</\</g' -e 's/>/\>/g'; } retry_backoff() { local tries="${1:-6}"; shift local i=1 delay=1 while ((i <= tries)); do

Procedimiento

```
sleep "$delay"
delay=$((delay*2)); (( delay > 24 )) && delay=24
i=$((i+1))
```

done return 1 }

send_telegram_html() { local msg="$1" local resp resp="$(

Procedimiento

```
  "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'||' '

Procedimiento

```
  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 '

Procedimiento

```
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 "$issues_list"]]; then list_block="Top issues:\n<pre>$(printf '%s' "$issues_list" | escape_html)</pre>" fi

msg="$(cat <<EOF ${esc_title}

Servidor: ${esc_srv} Docker: <code>${docker_ok}</code> Issues: <code>${esc_issues}</code>

${list_block} Hora: <code>$(date | escape_html)</code> EOF)"

retry_backoff 4 send_telegram_html "$msg" >/dev/null 2>&1 || true

6.2. Unit + Timer

`docker-heartbeat.service`:

[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

`docker-heartbeat.timer`:

[Unit] Description=Daily Docker Heartbeat Timer

[Timer] OnCalendar=daily Persistent=true RandomizedDelaySec=600

[Install] WantedBy=timers.target

Activar:

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

7. Operación

7.1. Estado

Código (bash)

```
docker ps
docker ps -a
```

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
- mantener `NOTIFY_HEALTHY=false` salvo necesidad

8. Troubleshooting

8.1. No llegan alertas

Código (bash)

```
journalctl -u docker-watch.service -n 200 --no-pager
sudo -u dockwatch docker ps
getent hosts api.telegram.org
curl -I https://api.telegram.org
```

8.2. UNHEALTHY no aparece

- confirmar que el contenedor tiene `HEALTHCHECK` (Dockerfile o Compose).

8.3. Restart bloqueado

- ajustar `RESTART_COOLDOWN_SEC` y `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
```