Monitorización de Internet con MySpeed, myspeed-api, n8n y Telegram

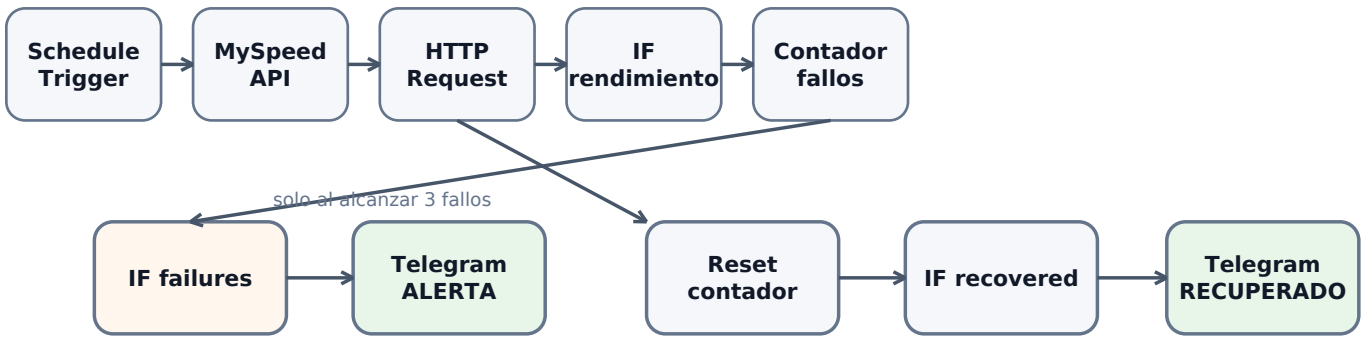
Guía técnica completa — instalación, configuración, pruebas y funcionamiento

Objetivo: construir una monitorización local y gratuita que ejecute una prueba de velocidad periódicamente, evalúe el rendimiento, acumule fallos consecutivos y envíe una notificación de Telegram únicamente cuando se detecte una degradación sostenida. Cuando la conexión vuelve a la normalidad, el sistema puede enviar una notificación de recuperación.

> Nota: esta guía excluye completamente la integración con Microsoft Teams. Está basada en el procedimiento y la configuración trabajados en el proyecto: Ubuntu Server, Docker/Compose, MySpeed, myspeed-api, n8n y Telegram.

1. Arquitectura general

La solución se divide en varios componentes independientes. MySpeed realiza las pruebas y almacena sus resultados; myspeed-api expone el último resultado mediante HTTP; n8n consulta ese endpoint y decide qué hacer; Telegram actúa como canal de notificación.



Internet normal → no Telegram. 3 fallos consecutivos → alerta. Recuperación → aviso y reinicio del estado.

Flujo lógico resumido

1	Schedule Trigger	n8n inicia una ejecución según el intervalo configurado.
2	HTTP Request	n8n consulta myspeed-api y recibe download, upload, ping, error y otros datos.
3	IF	Se determina si el resultado es anormal. En la configuración final usada: download < 10 Mbps, upload < 10 Mbps o ping > 100 ms.
4	Contador fallos	Cada resultado anormal incrementa el contador persistente. El umbral es 3 fallos consecutivos.
5	IF failures + Telegram	Solo cuando se alcanza el tercer fallo y todavía no se ha alertado, se envía la alerta.

6	Reset contador	Cuando el resultado vuelve a ser normal, se reinicia el contador y se conserva el estado anterior para saber si hubo una alerta activa.
7	IF recovered + Telegram	Si había una alerta activa y la conexión se recupera, se envía el mensaje de recuperación.

2. Requisitos y preparación

Servidor Linux (en el proyecto se utilizó Ubuntu Server), acceso SSH o consola, Docker, Docker Compose y acceso a la red local. El diseño está pensado para ejecutarse localmente sin depender de servicios cloud de pago.

Se recomienda reservar una IP fija para el servidor o una reserva DHCP. En el proyecto el servidor fue accesible en la red local y n8n se utilizó mediante el puerto 5678.

3. Instalación de Docker y Docker Compose

En Ubuntu Server, instala Docker Engine y el complemento de Compose. Una vez instalado, verifica que ambos comandos funcionan.

```
docker --version
docker compose version
```

Comprueba además que el servicio de Docker está activo:

```
sudo systemctl status docker
sudo systemctl enable docker
```

Prueba básica:

```
sudo docker run hello-world
```

Si el contenedor de prueba termina correctamente, Docker está operativo.

4. Portainer (opcional, pero utilizado en el entorno)

Portainer permite administrar los contenedores desde una interfaz web. No es imprescindible para que el sistema funcione; todos los componentes pueden administrarse con Docker Compose.

```
docker volume create portainer_data

docker run -d \
-p 8000:8000 \
-p 9443:9443 \
--name portainer \
--restart=always \
-v /var/run/docker.sock:/var/run/docker.sock \
-v portainer_data:/data \
portainer/portainer-ce:lts
```

Después, abre el puerto 9443 del servidor para acceder a Portainer.

5. Instalación de MySpeed

MySpeed es el componente que ejecuta y almacena las pruebas de velocidad. En el proyecto se utilizó la imagen germannewsmaker/myspeed.

Una forma sencilla de mantenerlo es mediante Docker Compose. Crea un directorio para el proyecto:

```
mkdir -p ~/myspeed
cd ~/myspeed
```

Ejemplo de compose:

```
services:
myspeed:
image: germannewsmaker/myspeed:latest
container_name: myspeed
restart: unless-stopped
ports:
- "5216:5216"
volumes:
- ./data:/myspeed/data
```

Levanta el servicio:

```
docker compose up -d
docker ps
```

Comprueba la interfaz web de MySpeed desde un navegador usando la IP del servidor y el puerto publicado. El objetivo es confirmar primero que MySpeed ejecuta pruebas y guarda resultados antes de conectar n8n.

6. Instalación de myspeed-api

myspeed-api se utilizó como una pequeña API intermedia para exponer el último resultado de MySpeed a n8n. En el proyecto se construyó con Docker mediante un Dockerfile y docker-compose.yml.

La estructura utilizada fue similar a:

```
myspeed-api/
■■■ Dockerfile
■■■ docker-compose.yml
■■■ app/
```

Un punto importante durante la instalación fue la sensibilidad a mayúsculas/minúsculas del nombre del Dockerfile. En Linux debe coincidir exactamente con el nombre que se utiliza en el contexto de build.

Construye y levanta el servicio:

```
docker compose up -d --build
docker ps
```

Verifica el endpoint de salud y el endpoint del último resultado. En el proyecto se comprobó mediante curl.

```
curl http://192.168.1.25:8080/health
curl http://192.168.1.25:8080/latest
```

El endpoint /latest debe devolver datos como id, ping, download, upload, error, type, time y created. Esos campos son los que posteriormente consume n8n.

Ejemplo de respuesta esperada:

```
{
"id": 52,
"ping": 10,
"download": 942.82,
"upload": 779.25,
"error": null,
"type": "auto",
"time": 30,
"created": "2026-08-13T10:00:40.808Z"
}
```

7. Instalación de n8n con Docker

En el proyecto se utilizó n8n mediante Docker Compose, con la imagen docker.n8n.io/n8nio/n8n:latest. La persistencia es importante: el directorio/volumen de n8n debe mantenerse para conservar credenciales, workflows y configuraciones.

```
mkdir -p ~/n8n
cd ~/n8n
```

Ejemplo de compose base:

```
services:
n8n:
image: docker.n8n.io/n8nio/n8n:latest
container_name: n8n
restart: unless-stopped
ports:
```

```
- "5678:5678"
volumes:
- ./n8n_data:/home/node/.n8n

docker compose up -d
docker ps
docker logs -f n8n
```

Accede a n8n desde el navegador mediante http://IP_DEL_SERVIDOR:5678. En el entorno utilizado se trabajó con n8n 2.34.4.

8. Crear el workflow «Monitor Internet - MySpeed»

El workflow final contiene ocho elementos principales: Schedule Trigger, HTTP Request, IF, Contador fallos, IF failures, Telegram de alerta, Reset contador, IF recovered y Telegram de recuperación.

8.1 Schedule Trigger

> Para las pruebas se utilizó un intervalo corto. En la captura final, el trigger estaba configurado para ejecutar cada 5 minutos. Una vez validado el sistema, puedes aumentar el intervalo según el nivel de monitorización que quieras.

Importante: n8n no necesita que ejecutes manualmente el workflow cada vez. Cuando el workflow está activo/publicado, el Schedule Trigger inicia las ejecuciones automáticamente.

8.2 HTTP Request

Configura una petición GET contra el endpoint /latest de myspeed-api. En el entorno del proyecto el endpoint era similar a:

```
GET http://192.168.1.25:8080/latest
```

Ejecuta el nodo y confirma que devuelve un único item con los campos download, upload y ping.

8.3 IF — detectar rendimiento anormal

El IF recibe directamente el JSON de HTTP Request. La condición final utilizada en el proyecto fue:

```
{{ $json.download < 10 ||
$json.upload < 10 ||
$json.ping > 100 }}
```

Con esta lógica, un resultado se considera anormal si la descarga baja de 10 Mbps, la subida baja de 10 Mbps o el ping supera 100 ms. Si ninguna condición se cumple, el resultado se considera normal.

8.4 Contador de fallos

El contador utiliza los datos estáticos globales del workflow para mantener el estado entre ejecuciones.

```
const staticData = $getWorkflowStaticData('global');

if (!staticData.failures) {
staticData.failures = 0;
}

staticData.failures++;

const threshold = 3;

const shouldAlert =
staticData.failures >= threshold &&
staticData.alerting !== true;

if (shouldAlert) {
staticData.alerting = true;
}

return [{
```

```
json: {
...$json,
failures: staticData.failures,
alert: shouldAlert
}
}];
```

El comportamiento es deliberadamente conservador: el primer fallo incrementa el contador a 1, el segundo a 2 y el tercero activa la alerta. Mientras alerting permanezca activo, no se vuelven a enviar alertas en cada ejecución.

8.5 IF failures

Este IF comprueba el campo generado por el contador:

```
{{ $json.alert }}
```

Operador: is true. Solo la salida True debe conectarse al nodo de Telegram de alerta.

8.6 Telegram — alerta

Crea/configura la credencial de Telegram en n8n y selecciona el bot y chat de destino. El mensaje puede incluir download, upload, ping y el número de fallos consecutivos.

Una prueba manual del nodo Send a text message debe producir inmediatamente el mensaje en Telegram. Esta prueba confirma que la credencial y el chat están correctamente configurados.

8.7 Reset contador

La rama False del IF principal indica que la conexión ha vuelto a valores normales. El nodo utilizado para gestionar la recuperación es:

```
const staticData = $getWorkflowStaticData('global');

// Guardamos el estado anterior
const wasAlerting = staticData.alerting === true;

// Reiniciamos el contador
staticData.failures = 0;
staticData.alerting = false;

return [{
json: {
...$json,
recovered: wasAlerting
}
}];
```

La clave está en guardar wasAlerting antes de reiniciar el estado. Así se puede diferenciar entre una recuperación real y una ejecución normal sin alerta previa.

8.8 IF recovered

Este nodo evalúa:

```
{{ $json.recovered }}
```

Operador: is true. Solo cuando existía una alerta activa y posteriormente el resultado vuelve a ser normal se envía el mensaje de recuperación.

9. Pruebas del sistema

> No conviene probar todo únicamente ejecutando nodos aislados. La persistencia del contador depende de ejecuciones reales del workflow. Para una prueba completa, utiliza el workflow activo y deja que el Schedule Trigger genere las ejecuciones.

Prueba 1 — conexión normal

Con valores como download ≈ 900 Mbps, upload ≈ 700–800 Mbps y ping ≈ 10 ms, el IF debe ir por la rama normal. No debe llegar ningún mensaje de alerta a Telegram.

Prueba 2 — tres fallos consecutivos

Fuerza o espera resultados que cumplan la condición anormal durante tres ejecuciones consecutivas. Debes observar aproximadamente:

```
Ejecución 1 → failures = 1 → alert = false
Ejecución 2 → failures = 2 → alert = false
Ejecución 3 → failures = 3 → alert = true → Telegram
```

Después de la tercera ejecución debe llegar la alerta. La cuarta ejecución, si continúa el problema, no debería generar otra alerta porque alerting ya está en true.

Prueba 3 — recuperación

Cuando el siguiente resultado vuelva a ser normal, la rama de recuperación debe ejecutar Reset contador. Si había una alerta activa, recovered será true y Telegram enviará el aviso de recuperación. El contador vuelve a 0.

Secuencia esperada:

```
Normal → Normal → Fallo(1) → Fallo(2) → Fallo(3)
→ ALERTA TELEGRAM → Fallo(4) sin nueva alerta
→ Normal → RECUPERACIÓN TELEGRAM → contador = 0
```

10. Comprobaciones y diagnóstico

Si Telegram funciona al pulsar «Execute step» pero no llega automáticamente, comprueba que el IF realmente esté enviando datos por la salida True y que el workflow esté activo. El botón Execute step prueba un nodo; no sustituye a una ejecución completa disparada por el Schedule Trigger.

Si el contador no conserva el valor entre ejecuciones, no pruebes únicamente el nodo de código de forma aislada. Ejecuta el workflow completo mediante el Schedule Trigger y revisa la pestaña de Executions. También verifica que el workflow esté guardado y activo.

Si /latest no responde, revisa:

• docker ps — comprobar que myspeed, myspeed-api y n8n están levantados.

• docker logs — revisar errores de arranque.

• curl http://192.168.1.25:8080/health — comprobar la API.

• curl http://192.168.1.25:8080/latest — comprobar que existe un resultado.

• Comprobar conectividad entre el contenedor de n8n y myspeed-api.

11. Comportamiento final del sistema

La intención del diseño es que Telegram no se convierta en una máquina de spam. En condiciones normales no recibes mensajes. Si aparece una degradación puntual, el contador la registra pero espera confirmación. Solo después de tres fallos consecutivos se genera la alerta.

Una vez alertado, el sistema queda en estado alerting = true. Los siguientes fallos no generan nuevas alertas. Cuando la conexión vuelve a la normalidad, Reset contador detecta que había una alerta activa, genera recovered = true, permite enviar el aviso de recuperación y limpia el estado.

12. Checklist de puesta en producción

☐ Docker instalado y servicio activo.

☐ MySpeed ejecutando pruebas correctamente.

☐ myspeed-api responde en /health y /latest.

☐ n8n accesible y con almacenamiento persistente.

☐ Workflow «Monitor Internet - MySpeed» guardado.

☐ Schedule Trigger configurado con el intervalo deseado.

☐ HTTP Request devuelve los datos de MySpeed.

☐ IF principal usa la condición de rendimiento definida.

☐ Contador de fallos conserva el estado entre ejecuciones.

☐ IF failures evalúa {{$json.alert}} como true.

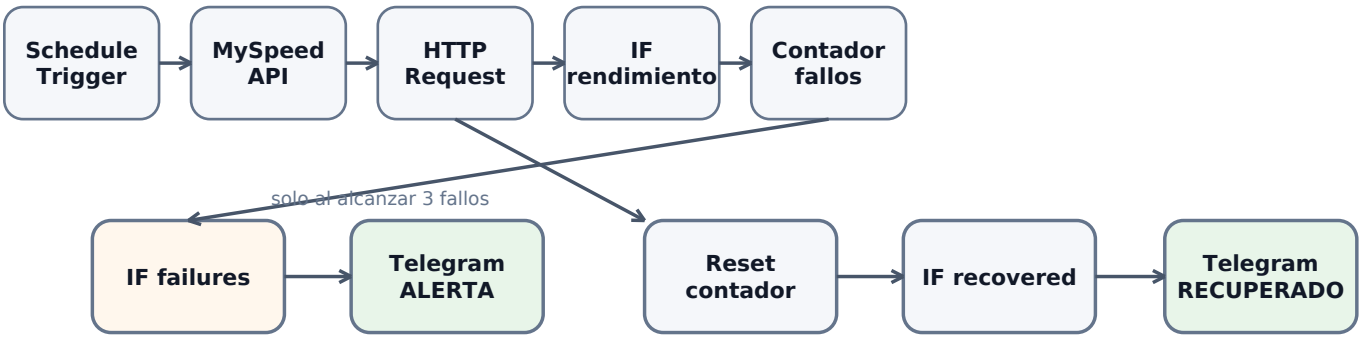
☐ Telegram envía correctamente una prueba manual.

☐ Reset contador y IF recovered están conectados.

☐ El workflow está activo para que Schedule Trigger lo ejecute automáticamente.

13. Diagrama rápido para publicar en un artículo

Puedes reutilizar esta sección como resumen visual en WordPress o LinkedIn. La idea clave es separar la recogida de datos, la evaluación y la gestión del estado.



Internet normal → no Telegram. 3 fallos consecutivos → alerta. Recuperación → aviso y reinicio del estado.

Resumen en una frase: MySpeed mide → myspeed-api expone → n8n analiza → el contador confirma → Telegram avisa.

14. Resultado conseguido

Con esta arquitectura se obtiene una monitorización local, automatizada y basada en Docker, sin necesidad de contratar un servicio externo de monitorización. El sistema permite detectar degradaciones sostenidas, evitar falsas alarmas por un único resultado extraño y avisar también cuando la conexión se recupera.

Fin de la guía.