# MeshNet “The Boss” — Broker JSONL + Bot de Telegram + Pasarela APRS (Docker)

Este repositorio contiene un *stack* Docker listo para usar que integra:

* **Broker JSONL** para Meshtastic (v5.x) — exporta eventos, backlog y tareas.
* **Bot de Telegram** con comandos de red (vecinos, traceroute, telemetría, enviar, programar, APRS, etc.).
* **Pasarela APRS** (Meshtastic ⇄ APRS KISS y APRS-IS opcional), con filtros por etiqueta [CHx].

Objetivo: que cualquier usuario pueda clonar, rellenar su .env y levantar los contenedores en **dos minutos**.

## 🧱 Requisitos

* **Docker** y **Docker Compose v2** (o docker compose integrado).
* Un **nodo Meshtastic** accesible por TCP (normalmente en IP\_DEL\_NODO:4403).
* (Opcional) Un **TNC KISS por TCP** (ej. Direwolf o Soundmodem) en el host: host.docker.internal:8100 en Windows/macOS o 127.0.0.1:8100 en Linux.
* (Opcional) Credenciales de **APRS-IS** (indicativo con SSID y *passcode*) para subir posiciones etiquetadas.
* Un **bot de Telegram** (Token) y, opcionalmente, lista de administradores.

## 🚀 Puesta en marcha rápida

# 1) Clona el repositorio  
git clone https://github.com/tu-usuario/the-boss-docker.git  
cd the-boss-docker  
  
# 2) Crea tu fichero de entorno  
cp .env-example.txt .env  
# 3) Edita .env con tus valores (ver tabla más abajo)  
  
# 4) Levanta los servicios principales  
docker compose up -d broker bot aprs  
  
# 5) Ver logs  
docker compose logs -f broker  
docker compose logs -f bot  
docker compose logs -f aprs

Consejo: Si vas a usar **Direwolf**/**Soundmodem** en el host, arráncalo primero y verifica que el puerto TCP (p.ej. 8100) está escuchando.

## ⚙️ Variables de entorno (.env)

Crea un archivo .env en la raíz (puedes partir de .env-example.txt). Mínimo, ajusta estos campos:

| Clave | Descripción | Ejemplo |
| --- | --- | --- |
| MESHTASTIC\_HOST | IP/host del nodo Meshtastic (TCPInterface, normalmente 4403) | 192.168.1.201 |
| BROKER\_PORT | Puerto **JSONL** del broker hacia clientes (bot/APRS) | 8765 |
| BACKLOG\_PORT | Puerto **backlog/ctrl** del broker (UDP/TCP) | 8766 |
| TELEGRAM\_TOKEN | Token del bot de Telegram | 123456:ABC... |
| ADMIN\_IDS | Lista de IDs (coma/; separada) con rol administrador | 1111,2222 |
| KISS\_HOST | Host del TNC KISS TCP | host.docker.internal (Windows/macOS) / 127.0.0.1 (Linux) |
| KISS\_PORT | Puerto del TNC KISS TCP | 8100 |
| MESHTASTIC\_CH | Canal lógico por defecto para inyección desde APRS si no hay etiqueta | 0 |
| BOT\_START\_DELAY | Segundos que el bot espera al iniciar (permitir enlazar el nodo) | 90 |

**Parámetros APRS opcionales (si subes a APRS‑IS):**

| Clave | Descripción | Ejemplo |
| --- | --- | --- |
| APRSIS\_USER | Indicativo-SSID con el que subir a APRS‑IS | EB2XXX-10 |
| APRSIS\_PASSCODE | *Passcode* asociado a tu indicativo | 12345 |
| APRSIS\_HOST | Servidor APRS‑IS | rotate.aprs2.net |
| APRSIS\_PORT | Puerto APRS‑IS | 14580 |
| APRSIS\_FILTER | Filtro APRS-IS opcional | m/50 |

**Ajustes KISS (10 ms/unidad):** KISS\_TXDELAY=30 (300 ms), KISS\_PERSIST=200, KISS\_SLOTTIME=10, KISS\_TXTAIL=3.

**Control y red del broker (avanzado):**

* BROKER\_HOST / BROKER\_CTRL\_HOST: cómo se conectan bot/APRS al broker dentro de Docker. Por defecto, el compose los resuelve por nombre de servicio.
* DISABLE\_BOT\_TCP=1: evita doble sesión TCP del **bot** al nodo cuando ya existe la del **broker**.
* En la pasarela APRS se usa network\_mode: "service:broker" para **compartir la red** del broker y enlazarlo por 127.0.0.1.

**Windows/macOS:** usa host.docker.internal para que el contenedor alcance el TNC KISS del host.

**Linux:** usa 127.0.0.1 solo si compartes *network namespace* con el broker; si no, mapea el puerto del host (-p 8100:8100).

## 🧩 Servicios y puertos

* **broker**
  + Expone JSONL en :8765 (por defecto) y **backlog/ctrl** en :8766.
  + Lee del nodo Meshtastic por TCP (MESHTASTIC\_HOST:4403).
  + Persiste posiciones y tareas en ./bot\_data (volumen mapeado).
* **bot**
  + Habla con el broker (BROKER\_HOST:8765) y con backlog/ctrl (:8766).
  + Comandos principales: /start, /menu, /ver\_nodos, /vecinos, /traceroute, /telemetria, /enviar, /enviar\_ack, /programar, /en, /manana, /tareas, /position, /position\_mapa, /cobertura, /aprs, /aprs\_on, /aprs\_off, /estado, /reconectar.
  + Usa BOT\_START\_DELAY para dar tiempo a que el broker enlace con el nodo.
* **aprs**
  + **KISS TCP** hacia tu TNC: KISS\_HOST:KISS\_PORT.
  + **Control UDP** (desde el bot) en 127.0.0.1:9464 (compartiendo red con broker).
  + **Broker JSONL** en 127.0.0.1:8765 (compartiendo red con broker).
  + Sube a **APRS‑IS** si APRSIS\_USER y APRSIS\_PASSCODE están definidos.
  + **Reinyecta a malla SOLO** tramas que lleven [CHx] o [CANAL x] en el comentario.

## 🗂 Estructura de volúmenes y datos

* ./bot\_data/positions.jsonl y positions\_last.json — últimas posiciones.
* ./bot\_data/scheduled\_tasks.jsonl — planificador de mensajes.
* ./bot\_data/maps/ — salidas de cobertura (HTML/KML) si generas mapas desde el bot.

Puedes montar bot\_data como volumen para persistir datos entre reinicios.

## 📦 Ejemplos de docker compose

Los servicios están definidos para que:

* bot y aprs **dependan** de broker.
* aprs use network\_mode: "service:broker" (misma pila de red); así puede hablar con broker por 127.0.0.1.
* Variables del .env prevalezcan sobre valores del YAML.

Si no deseas APRS, puedes levantar solo broker y bot.

## 🔐 Seguridad / buenas prácticas

* El *token* de Telegram y el *passcode* de APRS‑IS **no deben** enviarse a git; guárdalos solo en .env.
* Usa **IDs de admin** reales para limitar comandos avanzados.
* Mapea puertos de broker solo dentro de tu red local a no ser que necesites acceso externo.

## 🧪 Pruebas rápidas

1. **Bot operativo**

* En Telegram: /estado → debe listar latencia y servicios.
* /ver\_nodos → muestra nodos; /vecinos → directos; /traceroute !id.

1. **APRS**

* Con TNC activo, desde el bot: /aprs 0 Hola APRS ⇒ deberías ver la trama en el TNC.
* Para uplink APRS‑IS: define APRSIS\_USER y APRSIS\_PASSCODE; solo suben **posiciones** con [CHx].

1. **Programación**

* /en 5 canal 0 Recordatorio ⇒ mensaje en 5 minutos por canal 0.
* /tareas para revisar estado.

## 🛠 Solución de problemas

* **El bot no “responde” inmediatamente**: respeta BOT\_START\_DELAY para dar tiempo a que el broker enlace con el nodo.
* **El APRS no transmite**: verifica KISS\_HOST:KISS\_PORT, que el TNC acepte KISS por TCP y que el contenedor pueda llegar (Windows/macOS → host.docker.internal).
* **No quiero reinyectar todo APRS a la malla**: la pasarela **solo** reinyecta si hay etiqueta [CHx] en el comentario ([CANAL x] también válido).
* **Duplicados**: el sistema hace *de‑dup* básico en APRS y gestiona ACKs por aplicación para minimizar repeticiones.
* **Heartbeat del SDK**: el broker incluye *guards* para proteger sendHeartbeat y evitar olores a *loopback*.

## 📋 Licencia y contribuciones

* PRs y *issues* bienvenidos.
* Incluye logs recortados y tu .env sin secretos cuando abras incidencias.

## 🗺 Hoja de ruta

* CSV/JSON de métricas enriquecidas por nodo.
* Endpoints adicionales del backlog.
* Plantillas para despliegue en Raspberry Pi y Synology.

### Créditos

* Meshtastic® y su comunidad.
* aprslib / Direwolf / Soundmodem.
* Usuarios que han testeado y aportado ideas a “The Boss”.

## 📜 Comandos del bot: guía completa

Todos los comandos funcionan en chats privados con el bot y en grupos donde esté presente. Los ejemplos muestran el **mensaje que envías a Telegram** y un **resumen de lo que hace**.

Notas generales: - Si el comando acepta !id o alias, el alias debe existir en el fichero de nodos (o haber sido visto recientemente por el broker). - Cuando procede, el bot **pausa** momentáneamente la sesión del broker para ejecutar CLI y luego **reanuda** (evita duplicar conexiones al 4403). - La mayoría de listados aceptan límites (max\_n) y timeout para esperar datos del pool.

### 🧭 /menu y /start

Muestra el menú contextual oficial (Telegram **SetMyCommands**) según tu rol (admin/usuario) y un resumen rápido del sistema. - **Ejemplo:** - Tú: /start - Bot: «Bienvenido… usa /menu para ver opciones».

### 🆘 /ayuda

Ayuda corta con enlaces y recordatorio de los comandos más usados.

### 🛰️ /estado

Resumen del estado del sistema: latencia de respuesta del nodo, estado del broker, bot y APRS. - **Ejemplo:** - Tú: /estado - Bot: «Broker OK (JSONL :8765, CTRL :8766) • Nodo enlazado • APRS: KISS conectado…»

### 📡 /ver\_nodos [max\_n] [timeout]

Lee los **últimos nodos** del **pool persistente** (no abre sesión nueva). Orden por recencia; muestra alias, !id, SNR y “visto hace”. - **Ejemplos:** - /ver\_nodos → top recientes. - /ver\_nodos 30 4 → hasta 30 nodos, esperando hasta 4 s al pool.

### 🤝 /vecinos [max\_n] [hops\_max]

Lista **vecinos** vistos (recientes) con su número de **saltos (hops)**, SNR y recencia. Usa broker/pool; no abre TCP nuevo. - **Ejemplos:** - /vecinos → directos por defecto (hops 0) o configuración actual. - /vecinos 20 2 → hasta 20 nodos con **hops ≤ 2**. - Alias/SNR y “visto hace” aparecen en salida.

### 🛰️🍞 /traceroute <!id|alias> [timeout]

Ejecuta **traceroute** hacia un nodo. El bot **pausa** el broker, lanza CLI meshtastic --traceroute, parsea los saltos y **reanuda** el broker. - **Ejemplos:** - /traceroute !06c756f0 → muestra cadena de saltos. - /traceroute Zgz\_Romareda 35 → con timeout 35 s.

### 📶 /telemetria [!id|alias] [mins|max\_n] [timeout]

* **Sin destino**: listado rápido de **métricas en vivo** del pool (orden por recencia). max\_n limita filas.
* **Con destino** (!id/alias): mezcla **en vivo + histórico** (FETCH\_TELEMETRY en broker) en una ventana de mins (por defecto 30).
* **Campos** comunes: SNR, RSSI, batería/voltaje, temperatura, airmon, etc.
* **Ejemplos:**
  + /telemetria → top métricas recientes.
  + /telemetria !06c756f0 20 4 → histórico 20 min, timeout 4 s.

### ✉️ /enviar canal <n> <texto> y /enviar <número|!id|alias> <texto>

Envío rápido por **canal** (broadcast) o **unicast** por !id/alias. - Usa **nodos.txt** / pool (sin refrescar por API) para evitar múltiples conexiones. - Reintento resiliente 1 vez si hay reconexión de pool. - Broadcast: **sin ACK**; Unicast: sin ACK (evita duplicados). El adapter añade feedback local si hay packet\_id. - **Ejemplos:** - /enviar canal 0 Hola red → broadcast por canal 0. - /enviar !ea0a8638 Prueba directa → unicast por !id. - /enviar Zgz\_Romareda Mensaje → unicast por alias.

### ✅ /enviar\_ack <número|!id|alias> <texto>

Como /enviar unicast pero solicitando **ACK** de aplicación. El bot reporta confirmación si llega.

### ⏱️ /programar, /en <min> canal <n> <texto>, /manana <hora> canal <n> <texto>

Planificador de envíos diferidos y tareas. - /en 5 canal 0 Recordatorio → en 5 minutos. - /manana 09:30 canal 0 Buenos días → mañana a las 09:30. - /programar → flujo guiado. - /tareas → lista tareas con estados (pending, sent, etc.). - /cancelar\_tarea <uuid> → cancela.

### 👂 /escuchar y /parar\_escucha

Pone al nodo en **modo escucha** un tiempo/condiciones definidas, y reporta nodos entrantes por consola/Telegram. Útil para descubrir vecinos.

### 🌐 /canales

Muestra/gestiona canal lógico por defecto y ayudas para **[CHx]**.

### 📍 /position y /position\_mapa

* /position → posición actual/conocida, última hora y !id.
* /position\_mapa → genera/enlaza mapa HTML/KML en ./bot\_data/maps/.

### 🗺️ /cobertura [opciones]

Genera **mapas de cobertura** (HTML/KML) a partir de posiciones/vistas conocidas. Archivos quedan en ./bot\_data/maps/.

### 🔌 /reconectar

Ordena al broker **reconectar** con el nodo (fuerza limpieza de cooldown si aplica).

### 📊 /estadistica *(solo admin)*

Muestra estadísticas de uso del bot por usuarios/fechas.

### 🪪 /lora

Información resumida del enlace LoRa y parámetros relevantes.

### 📡 APRS: /aprs, /aprs\_on, /aprs\_off, /aprs\_status

**Puente APRS ⇄ Mesh** con etiqueta obligatoria para inyección a la malla. - **Formatos admitidos** en /aprs: - /aprs canal N <texto> → broadcast a **canal N** y salida por APRS KISS. - /aprs N <texto> → atajo del anterior. - /aprs <CALL|broadcast>: <texto> [canal N] → compat dirigido o broadcast. - **Troceo**: si el payload supera APRS\_MAX\_LEN (≈67), se divide en varias tramas. - **Reinyección a malla**: **solo** si el comentario contiene [CHx] o [CANAL x]. - **APRS‑IS**: si defines APRSIS\_USER+APRSIS\_PASSCODE, se suben **posiciones** etiquetadas. - **Ejemplos:** - /aprs canal 0 [CH0] Saludo → emite por KISS y etiqueta para malla. - /aprs EB2EAS-11: Mensaje a estación → dirigido. - /aprs\_status → estado de KISS/APRS‑IS. - /aprs\_on / /aprs\_off → habilita/inhabilita uplink a APRS‑IS.

### 🔒 Permisos y roles

* **Usuarios**: acceso a consultas estándar y envíos por canal.
* **Admins**: comandos de gestión (p.ej. /estadistica, /reconectar, cancelación de tareas) y opciones avanzadas del menú.

### ✅ Buenas prácticas

* Evita spam de /traceroute: usa timeouts razonables (20–35 s) y recuerda que pausa/reanuda la sesión.
* Para **APRS**, configura bien KISS (host.docker.internal:8100 en Windows/macOS) y etiqueta [CHx] para reinyectar a la malla.
* Define BOT\_START\_DELAY (p.ej. 90 s) para que el bot espere a que el broker enlace con el nodo al arrancar.