Raspberry Pi 4 ADS-B Flight Feeder with Incus Containers

A guide to building a Raspberry Pi 4 home server running Debian 13 (Trixie) on an encrypted external SSD, using Incus containers to run ADS-B flight feeder services.

Presented to the vicpimakers group.

> **Note**: These instructions cover presentations one and two. Sections on security hardening, stability improvements, and Cloudflare tunnel remote access are not covered here and could be the topics of a future presentation. These instructions also supersede the previous Debian 12 (Bookworm) and LVM-based approach presented earlier.

Hardware

- Raspberry Pi 4 (8GB)
- External USB SSD (Samsung 128GB) — connected directly to one of the Pi's blue USB 3.0 ports (not through a hub)
- USB RTL-SDR dongle (generic RTL2838 TV tuner) for ADS-B reception on 1090 MHz

---

Overview

- Debian 13 (Trixie), arm64, installed via `debootstrap` from a Raspberry Pi OS Lite (64-bit) SD card
- Boots entirely from the external SSD — no SD card required after install
- Full disk encryption using LUKS2
- Remote LUKS unlock via Dropbear SSH in the initramfs
- Incus containers for isolated feeder services
- One decoder container shares ADS-B data with two feeder containers

Minimum System Requirements

These instructions are written specifically for the **Raspberry Pi 4**. The Pi 4 is the minimum recommended hardware for this setup because:

- It introduced native USB 3.0, allowing the Pi to boot directly from an external SSD without SD card tricks or OTP fuse programming
- It has a proper 64-bit ARM processor well-suited to running Incus containers
- It supports up to 8 GB RAM, giving comfortable headroom for multiple containers

RAM: 2 GB minimum, 4 GB recommended. These instructions were written using an 8 GB model.

Pi 5: Should work with minor adjustments but has not been tested with these instructions.

Earlier models (Pi 2, Pi 3): Flight feeders alone run perfectly well on older hardware — the ADS-B workload is very light. It is the combination of encrypted SSD booting and Incus containerisation that makes the Pi 4 the practical minimum for these specific instructions.

Why Debian 13 (Trixie)?

Earlier guides for this type of setup recommended Debian 12 (Bookworm) because Trixie was still in the "testing" phase and not considered stable enough for a production setup. That's no longer the case — Trixie was officially released in August 2025 and is now at version 13.5 with several rounds of security and bug-fix updates behind it. It ships kernel 6.12 LTS and has a support lifecycle running to 2030. Debian 12 reached end of full support in June 2026, so new builds should use Trixie.

Why no LVM?

The previous version of these instructions used LVM (Logical Volume Manager) on top of the LUKS2 encrypted partition, which allowed flexible resizing of volumes. However, LVM adds complexity and creates problems when trying to use Incus's native storage backends — Incus wants to manage a raw block device or volume group directly, which conflicts with nesting inside an existing LVM setup. For this use case (a small number of fixed-purpose containers), LVM's flexibility isn't needed. The simpler approach of a dedicated encrypted btrfs partition for Incus is cleaner, easier to understand, and works correctly with Incus out of the box.

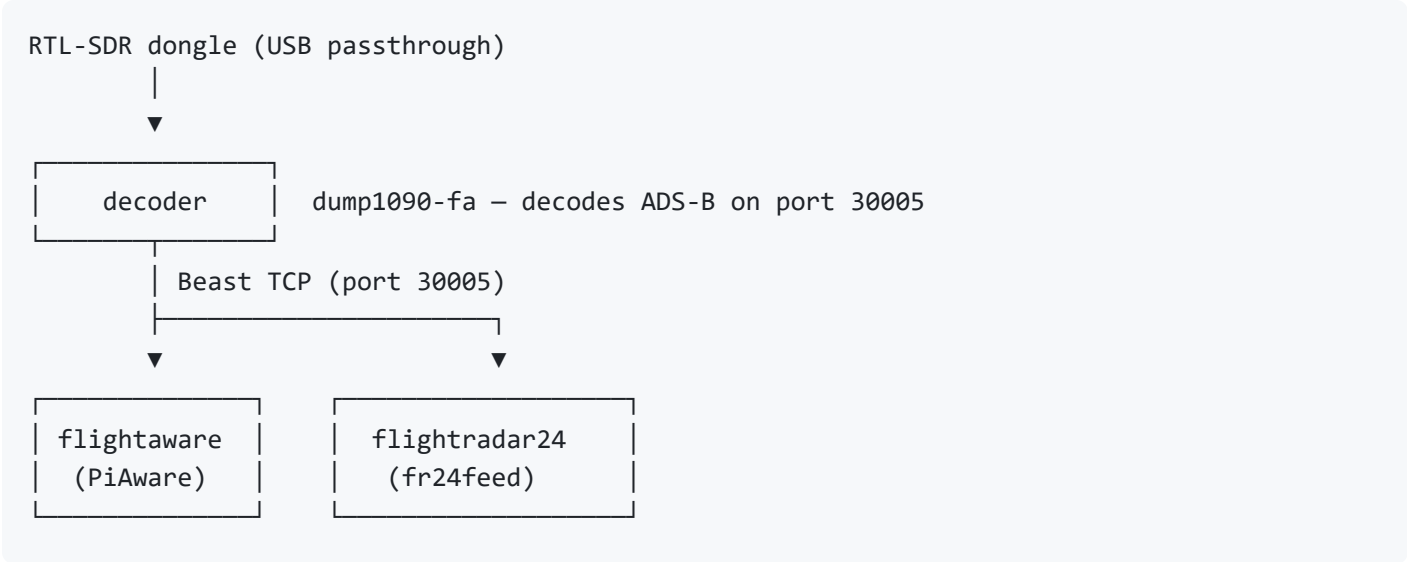
Disk Layout

Partition	Size	Type	Purpose
`sda1`	1 GiB	FAT32	Boot partition
`sda2`	32 GiB	LUKS2 → ext4	Root filesystem, unlocked by passphrase via Dropbear
`sda3`	~86 GiB	LUKS2 → btrfs	Incus storage, auto-unlocked via keyfile on root

A few notes on these choices:

- **`sda1` is 1 GiB** rather than the more common 512 MiB because Debian's `raspi-firmware` package installs firmware files for multiple Pi models, DTBs, the kernel, and the initramfs (which includes Dropbear) into the boot partition. 512 MiB is too tight once all of this is in place and leaves no room for kernel updates.
- **`sda2` is 32 GiB** — an arbitrary comfortable choice. The practical minimum for a Debian system running Incus with a few small containers is around 8–10 GiB, but with no LVM to resize later, it's worth being generous.
- **Swapfile on the root partition** rather than a separate swap partition — this means swap is automatically encrypted as part of `sda2` without any additional configuration.

Container Architecture



One container per feeder — isolation means a broken or misconfigured feeder doesn't affect the others. Each container can be restarted, rebuilt, or replaced independently.

btrfs storage backend for Incus — chosen out of familiarity from other Incus deployments. It works correctly with Incus out of the box and supports container snapshots if needed in the future.

Dropbear for remote unlock — the Pi is located remotely. After a power failure it needs to be unlockable without physically connecting a keyboard and monitor.

Part Zero: Prepare the SD Card

Use Raspberry Pi Imager to write Raspberry Pi OS Lite (64-bit) to a MicroSD card.

Key settings:

- **Hostname**: `vicpimakers-demo` (or your preferred hostname)
- **Username**: `gordo`
- **Locale**: `America/Vancouver`, keyboard `us`
- **SSH**: enabled with password authentication
- **Wi-Fi**: leave blank (use wired ethernet)

Boot the Pi from the SD card with keyboard, monitor, and network connected — but **do not attach the SSD yet**.

Part One: Install Debian 13 (Trixie) on the External SSD

1.1 Prepare the SD Card Environment

Log in and switch to root:

```
sudo -i
```

Update the system and EEPROM to ensure USB boot support:

```
apt update && apt full-upgrade -y
rpi-eeprom-update -a
```

Reboot if prompted, then install required tools:

```
apt install -y cryptsetup lvm2 debootstrap btrfs-progs
```

1.2 Attach the SSD and Partition It

Attach the external SSD and confirm it's visible:

```
lsblk -S -o name,model,size,fstype,mountpoint
```

You should see your SSD as `sda` with no filesystem. If the SSD has previous data, wipe it:

```
wipefs -a /dev/sda
```

Create the partition table:

```
parted -s /dev/sda mklabel gpt

# Boot partition (1 GiB)
parted -s /dev/sda mkpart primary fat32 1MiB 1025MiB
parted -s /dev/sda set 1 boot on
mkfs.vfat -F32 /dev/sda1

# Root partition (32 GiB)
parted -s /dev/sda mkpart primary 1025MiB 33793MiB

# Incus partition (remaining space)
parted -s /dev/sda mkpart primary 33793MiB 100%
```

Verify the layout:

```
parted /dev/sda print
```

Expected output:

```
Number  Start   End     Size    File system  Name     Flags
 1      1049kB  1075MB  1074MB  fat32        primary  boot, esp
 2      1075MB  35.4GB  34.4GB               primary
 3      35.4GB  128GB   92.6GB               primary
```

> **Note**: The size of `sda3` will vary depending on the size of your SSD. It will occupy all remaining space after `sda1` and `sda2`.

1.3 Set Up Encryption

Encrypt the root partition (`sda2`) with LUKS2:

```
cryptsetup luksFormat --type luks2 --cipher aes-xts-plain64 --key-size 512 --hash sha256
/dev/sda2
```

Open the encrypted partition and format it as ext4:

```
cryptsetup open /dev/sda2 cryptroot
mkfs.ext4 /dev/mapper/cryptroot
```

Encrypt the Incus partition (`sda3`) with a passphrase, then immediately create and enrol the keyfile:

```
cryptsetup luksFormat --type luks2 --cipher aes-xts-plain64 --key-size 512 --hash sha256
/dev/sda3
```

Open the encrypted partition and format it as btrfs:

```
cryptsetup open /dev/sda3 cryptincus
mkfs.btrfs /dev/mapper/cryptincus
```

Verify the layout:

```
lsblk -o name,size,fstype,mountpoint
```

Expected output:

```
sda            119.2G
├─sda1             1G vfat
├─sda2            32G crypto_LUKS
│ └─cryptroot     32G ext4
└─sda3          86.2G crypto_LUKS
  └─cryptincus  86.2G btrfs
```

1.4 Bootstrap Debian Trixie

Mount the root and boot partitions:

```
mkdir -p /mnt/debian
mount /dev/mapper/cryptroot /mnt/debian
mkdir -p /mnt/debian/boot/firmware
mount -t vfat /dev/sda1 /mnt/debian/boot/firmware
```

Bootstrap Debian 13 (Trixie) for ARM64:

```
debootstrap --arch=arm64 --include=locales,vim trixie /mnt/debian
http://deb.debian.org/debian
```

This will take a few minutes. Look for `I: Base system installed successfully.` at the end.

Now that `/mnt/debian` exists, create the keyfile for `sda3` and enrol it. You will be prompted for the `sda3` passphrase:

```
mkdir -p /mnt/debian/etc/keys
chmod 700 /mnt/debian/etc/keys
dd if=/dev/urandom bs=512 count=4 of=/mnt/debian/etc/keys/incus.key
chmod 600 /mnt/debian/etc/keys/incus.key
cryptsetup luksAddKey /dev/sda3 /mnt/debian/etc/keys/incus.key
```

After this the passphrase is no longer needed — all future unlocking uses the keyfile.

1.5 Enter the Chroot

Set up the chroot mounts:

```
mount --bind /dev /mnt/debian/dev
mount --bind /dev/pts /mnt/debian/dev/pts
mount -t proc proc /mnt/debian/proc
mount -t sysfs sys /mnt/debian/sys
mount -t tmpfs tmpfs /mnt/debian/run
chroot /mnt/debian /bin/bash
```

1.6 Configure Locale, Users, and Hostname

```
sed -i 's/^# *\(en_CA.UTF-8 UTF-8\)/\1/' /etc/locale.gen
sed -i 's/^# *\(en_GB.UTF-8 UTF-8\)/\1/' /etc/locale.gen
locale-gen
update-locale LANG=en_CA.UTF-8 LC_ALL=en_CA.UTF-8 LANGUAGE="en_CA:en_GB"
```

Set the root password:

```
passwd
```

Create the user account:

```
useradd -m gordo
passwd gordo
usermod -aG sudo gordo
chsh -s /bin/bash gordo
```

Set hostname:

```
echo "pi4-ywh" | tee /etc/hostname
echo "127.0.1.1 pi4-ywh" | tee -a /etc/hosts
```

1.7 Configure Networking

> **Note**: Debian Trixie on Raspberry Pi 4 uses `eth0` as the interface name (with `end0` as an alias). This is a Raspberry Pi convention — `raspi-firmware` writes `net.ifnames=0` into `cmdline.txt`, which disables predictable network interface naming and keeps the familiar `eth0` name used across all Raspberry Pi OS releases. We configure the network using `eth0` to match.

```
tee /etc/systemd/network/10-eth0.network > /dev/null <<EOF
[Match]
Name=eth0

[Network]
Address=192.168.8.21/24
Gateway=192.168.8.1
DNS=149.112.121.10
DNS=149.112.122.10
EOF
```

> **Note**: The DNS servers `149.112.121.10` and `149.112.122.10` are CIRA Canadian Shield (Private tier) — a free, privacy-focused recursive DNS service operated by the Canadian Internet Registration Authority (CIRA). It blocks malware and phishing domains and keeps DNS queries within Canada. A fitting choice for a Canadian project.

Enable `systemd-networkd` to start on boot:

```
systemctl enable systemd-networkd
```

1.8 Configure apt Sources

> **Note**: `mirror.it.ubc.ca` is the University of British Columbia's Debian mirror, hosted in Vancouver.

```
rm /etc/apt/sources.list

tee /etc/apt/sources.list.d/debian.sources > /dev/null <<EOF
Types: deb
URIs: http://mirror.it.ubc.ca/debian
Suites: trixie trixie-updates
Components: main contrib non-free non-free-firmware
Signed-By: /usr/share/keyrings/debian-archive-keyring.gpg

Types: deb
URIs: http://security.debian.org/debian-security
Suites: trixie-security
Components: main contrib non-free non-free-firmware
Signed-By: /usr/share/keyrings/debian-archive-keyring.gpg
EOF

apt update && apt full-upgrade -y
```

1.9 Install Required Packages

```
apt install -y \
    linux-image-arm64 \
    firmware-brcm80211 \
    firmware-misc-nonfree \
    firmware-linux \
    btrfs-progs \
    cryptsetup \
    cryptsetup-initramfs \
    curl \
    dropbear-initramfs \
    initramfs-tools \
    openssh-server \
    python3 \
    raspi-firmware \
    sudo \
    systemd-cryptsetup \
    systemd-resolved \
    systemd-timesyncd \
    ca-certificates \
    usbutils
```

Select your keyboard layout when prompted. Ignore the `dropbear` and `crypttab` warnings — these are addressed in the next steps.

1.10 Capture UUIDs

```
BOOT_UUID=$(blkid -s UUID -o value /dev/sda1)
SDA2_UUID=$(blkid -s UUID -o value /dev/sda2)
SDA3_UUID=$(blkid -s UUID -o value /dev/sda3)
ROOT_UUID=$(blkid -s UUID -o value /dev/mapper/cryptroot)

echo "BOOT: $BOOT_UUID"
echo "SDA2: $SDA2_UUID"
echo "SDA3: $SDA3_UUID"
echo "ROOT: $ROOT_UUID"
```

Verify all four UUIDs are populated before proceeding.

1.11 Configure fstab and crypttab

```
tee /etc/fstab > /dev/null <<EOF
UUID=${ROOT_UUID}  /               ext4   defaults,noatime  0 1
UUID=${BOOT_UUID}  /boot/firmware  vfat   defaults          0 2
/swapfile          none            swap   sw                0 0
EOF
```

```
tee /etc/crypttab > /dev/null <<EOF
cryptroot   UUID=${SDA2_UUID}  none                luks
cryptincus  UUID=${SDA3_UUID}  /etc/keys/incus.key  luks
EOF
```

Add the initramfs crypttab hook — this ensures only `cryptroot` is included in the initramfs (not `cryptincus`, which is unlocked later by systemd):

```
tee /etc/initramfs-tools/hooks/make-crypttab > /dev/null <<'EOF'
#!/bin/sh
PREREQ=""
prereqs() { echo "$PREREQ"; }
case "$1" in prereqs) prereqs; exit 0;; esac
. /usr/share/initramfs-tools/hook-functions
grep '^cryptroot' /etc/crypttab > "${DESTDIR}/cryptroot/crypttab"
exit 0
EOF
chmod +x /etc/initramfs-tools/hooks/make-crypttab
```

1.12 Configure Dropbear for Remote LUKS Unlock

Add network drivers to the initramfs:

```
tee -a /etc/initramfs-tools/modules > /dev/null <<EOF
bcmgenet
raspberrypi-cpufreq
EOF
```

Configure the initramfs network — note `eth0` to match `net.ifnames=0`:

```
sed -i 's/^DEVICE=.*/DEVICE=eth0/' /etc/initramfs-tools/initramfs.conf
echo "IP=192.168.8.21::192.168.8.1:255.255.255.0:pi4-ywh-dropbear:eth0:none" >>
/etc/initramfs-tools/initramfs.conf
```

Configure Dropbear options:

```
echo 'DROPBEAR_OPTIONS="-E -F -I 180 -j -k -p 2222 -s -c cryptroot-unlock"' | tee -a
/etc/dropbear/initramfs/dropbear.conf
```

Add your Dropbear public key. First generate the keypair on your **client**:

```
ssh-keygen -t ed25519 -a 100 -f ~/.ssh/pi4-ywh-dropbear -C "dropbear"
cat ~/.ssh/pi4-ywh-dropbear.pub
```

Then on the **server** (inside the chroot), paste your public key:

```
echo "ssh-ed25519 <your-public-key> dropbear" | tee
/etc/dropbear/initramfs/authorized_keys
chmod 600 /etc/dropbear/initramfs/authorized_keys
```

Add an entry to `~/.ssh/config` on your **client** for convenience:

```
Host dropbear
    Hostname 192.168.8.21
    Port 2222
    User root
    IdentityFile ~/.ssh/pi4-ywh-dropbear
    IdentitiesOnly yes
```

1.13 Configure Boot Parameters

Prevent video blanking and disable Wi-Fi:

> **Note**: The Pi's onboard Wi-Fi is disabled because the Wi-Fi firmware (`brcmfmac`) and the `wpa_supplicant` authentication daemon are not available in the initramfs at the Dropbear stage. Without them, the Pi cannot connect to a wireless network early enough to accept the LUKS unlock passphrase remotely. Ethernet is required for Dropbear to work. If you don't have a wired network available, a travel router configured as a wireless-to-ethernet bridge is a practical workaround.

```
tee -a /etc/default/raspi-firmware > /dev/null <<EOF

# prevent video blanking on a Raspberry Pi 4
hdmi_force_hotplug=1
hdmi_group=1
hdmi_mode=16

# disable Wi-Fi at boot
dtoverlay=disable-wifi
EOF
```

Set the kernel command line via `raspi-extra-cmdline` so it persists across kernel updates:

```
echo "cryptdevice=UUID=${SDA2_UUID}:cryptroot" | tee /etc/default/raspi-extra-cmdline
```

Create the Incus mount point:

```
mkdir -p /var/lib/incus
```

Add the fstab entry for the Incus volume:

```
echo "/dev/mapper/cryptincus  /var/lib/incus  btrfs  defaults,x-systemd.requires=systemd-
cryptsetup@cryptincus.service  0 0" >> /etc/fstab
```

Create the swapfile now, inside the chroot, so it exists on first boot:

```
dd if=/dev/zero of=/swapfile bs=1M count=4096
chmod 600 /swapfile
mkswap /swapfile
```

1.14 Build the initramfs

```
update-initramfs -u
```

Verify the critical components are present:

```
lsinitramfs $(ls /boot/initrd.img-*) | grep -E
'authorized_keys|dropbear|crypttab|bcmgenet'
```

Verify the key is actually populated (not empty):

```
unmkinitramfs $(ls /boot/initrd.img-*) /tmp/initrd
find /tmp/initrd -name authorized_keys -exec cat {} \;
```

You should see your public key printed. If the file is empty, re-run the `tee` command to write the key and rebuild the initramfs.

1.15 Exit the Chroot and Boot

```
exit
```

```
sync
umount /mnt/debian/boot/firmware
umount /mnt/debian/dev/pts
umount /mnt/debian/dev
umount /mnt/debian/proc
umount /mnt/debian/sys
umount /mnt/debian/run
umount /mnt/debian
cryptsetup close cryptincus
cryptsetup close cryptroot
poweroff
```

Wait for the green LED to stop flashing before disconnecting power.

Remove the SD card, then power on. The Pi should boot into the Dropbear initramfs.

1.16 First Boot

Wait 30-60 seconds after powering on, then connect via Dropbear from your client:

```
ssh dropbear
```

Type your LUKS passphrase for `sda2` when prompted. The connection will drop and the Pi will continue booting into Debian.

> **Note**: On this first boot only, you will also be prompted for the `sda3` passphrase on the console — systemd needs it to unlock `cryptincus` before the keyfile mechanism takes over on subsequent boots. Have a keyboard and monitor connected for this step. After the post-boot configuration is complete and you reboot, `sda3` will unlock automatically via the keyfile with no manual intervention required.

SSH into the fully booted system:

```
ssh gordo@192.168.8.21
```

1.17 Post-Boot Configuration

Switch to root:

```
sudo -i
```

Verify the keyfile is in place and works:

```
ls -la /etc/keys/incus.key
cryptsetup open /dev/sda3 cryptincus --key-file /etc/keys/incus.key
cryptsetup close cryptincus
```

Unlock and mount `cryptincus` manually for this first session only — subsequent boots will handle this automatically via crypttab and fstab:

```
systemctl daemon-reload
systemctl start systemd-cryptsetup@cryptincus
mount /dev/mapper/cryptincus /var/lib/incus
```

Rebuild the initramfs to pick up the updated crypttab:

```
update-initramfs -u
```

Reboot to verify everything comes up automatically:

```
reboot
```

After unlocking via Dropbear and booting, verify:

```
sudo -i
systemctl --failed
lsblk -o name,size,fstype,mountpoint
swapon --show
```

Expected: 0 failed units, `cryptincus` open and mounted at `/var/lib/incus`, swapfile active.

Part Two: Install and Configure Incus

2.1 Install Incus

```
sudo apt install -y incus
```

Add your user to the `incus-admin` group:

```
sudo usermod -aG incus-admin gordo
```

Log out and back in for the group membership to take effect, then verify:

```
groups
```

2.2 Initial Incus Configuration

```
sudo incus admin init
```

Answer the prompts as follows:

Prompt	Answer
Use clustering?	`no`
Configure a new storage pool?	`yes`
Name of the new storage pool	`pool0`
Storage backend	`btrfs`
Create a new btrfs subvolume under /var/lib/incus?	`yes`
Create a new local network bridge?	`yes`
Bridge name	`incusbr0` (default)
IPv4 address	`auto` (default)
IPv6 address	`none`
Available over the network?	`no`
Automatic image updates?	`yes`
Print YAML preseed?	`no`

Verify the setup:

```
incus info
incus storage list
incus network list
```

Part Three: Configure the Host for RTL-SDR

3.1 Blacklist DVB-T Kernel Modules

Prevent the host from grabbing the RTL-SDR dongle before the container can use it:

```
sudo tee /etc/modprobe.d/blacklist-rtl.conf > /dev/null <<EOF
blacklist dvb_usb_rtl28xxu
blacklist rtl2832
blacklist rtl2830
EOF

sudo modprobe -r dvb_usb_rtl28xxu rtl2832 rtl2830 2>/dev/null
```

3.2 Add udev Rule for RTL-SDR Permissions

```
sudo tee /etc/udev/rules.d/rtl-sdr.rules > /dev/null <<EOF
SUBSYSTEM=="usb", ATTRS{idVendor}=="0bda", ATTRS{idProduct}=="2838", MODE="0664",
GROUP="plugdev"
EOF

sudo udevadm control --reload-rules
sudo udevadm trigger --subsystem-match=usb --action=add
```

Verify the permissions:

```
ls -la /dev/bus/usb/001/004
```

Expected: `crw-rw-r-- 1 root plugdev`

Part Four: Create and Configure the Decoder Container

4.1 Launch the Container

```
incus launch images:debian/13 decoder
```

4.2 Add USB Passthrough

The RTL-SDR dongle has vendor ID `0bda` and product ID `2838`:

```
incus config device add decoder rtlsdr usb vendorid=0bda productid=2838
incus config device set decoder rtlsdr gid 46
incus config device set decoder rtlsdr mode 0664
```

The `gid 46` corresponds to the `plugdev` group.

Restart the container to apply the device config:

```
incus restart decoder
```

4.3 Disable IPv6

```
incus exec decoder -- bash -c "echo 'net.ipv6.conf.all.disable_ipv6 = 1' >
/etc/sysctl.d/99-disable-ipv6.conf && sysctl -p /etc/sysctl.d/99-disable-ipv6.conf"
```

4.4 Install dump1090-fa

```
incus exec decoder -- /bin/bash
```

Inside the container:

```
apt update && apt install -y wget usbutils

wget https://www.flightaware.com/adsb/piaware/files/packages/pool/piaware/f/flightaware-
apt-repository/flightaware-apt-repository_1.3_all.deb
dpkg -i flightaware-apt-repository_1.3_all.deb
apt update
apt install -y dump1090-fa
```

Add the `dump1090` user to the `plugdev` group:

```
usermod -aG plugdev dump1090
systemctl restart dump1090-fa
systemctl status dump1090-fa
```

Verify it's receiving signals (after a minute or two):

```
cat /run/dump1090-fa/stats.json | python3 -m json.tool | grep -A5 '"last1min"'
```

Exit the container:

```
exit
```

4.5 Enable Autostart

```
incus config set decoder boot.autostart true
```

Part Five: Create and Configure the FlightAware Container

5.1 Launch the Container

```
incus launch images:debian/13 flightaware
```

5.2 Disable IPv6

```
incus exec flightaware -- bash -c "echo 'net.ipv6.conf.all.disable_ipv6 = 1' >
/etc/sysctl.d/99-disable-ipv6.conf && sysctl -p /etc/sysctl.d/99-disable-ipv6.conf"
```

5.3 Install PiAware

```
incus exec flightaware -- /bin/bash
```

Inside the container:

```
apt update && apt install -y wget

wget https://www.flightaware.com/adsb/piaware/files/packages/pool/piaware/f/flightaware-
apt-repository/flightaware-apt-repository_1.3_all.deb
dpkg -i flightaware-apt-repository_1.3_all.deb
apt update
apt install -y piaware
```

5.4 Configure PiAware

```
piaware-config receiver-type other
piaware-config receiver-host decoder
piaware-config receiver-port 30005
```

Restart and check status:

```
systemctl restart piaware
journalctl -u piaware -n 20
```

Look for:

```
encrypted session established with FlightAware
logged in to FlightAware as user guest
Started faup1090 ... to connect to the ADS-B data program at decoder/30005
```

> **Note**: After installing, claim your feeder using the following command to generate a direct claim URL:
>
> ```
> echo "https://www.flightaware.com/adsb/piaware/claim/$(incus exec flightaware -- cat /var/cache/piaware/feeder_id)"
> ```
>
> Open the URL in your browser to associate the feeder with your FlightAware account.

Exit the container:

```
exit
```

5.5 Enable Autostart

```
incus config set flightaware boot.autostart true
```

Part Six: Create and Configure the FlightRadar24 Container

6.1 Launch the Container

```
incus launch images:debian/13 flightradar24
```

6.2 Disable IPv6

```
incus exec flightradar24 -- bash -c "echo 'net.ipv6.conf.all.disable_ipv6 = 1' > /etc/sysctl.d/99-disable-ipv6.conf && sysctl -p /etc/sysctl.d/99-disable-ipv6.conf"
```

6.3 Install fr24feed

> **Note**: FR24's apt repository uses a SHA1 signing key which Debian Trixie rejects as insecure since February 2026. Use the official install script instead, which always installs the latest version.

```
incus exec flightradar24 -- /bin/bash
```

Inside the container:

```
apt update && apt install -y wget
wget -qO- https://fr24.com/install.sh | bash -s
```

> **Note**: The install will produce many lines of red `Permission denied: Failed to write 'change' to /sys/devices/...` errors. These are harmless — fr24feed's post-install script tries to trigger udev events which are correctly blocked inside an LXC container. The install has succeeded as long as you see `Setting up fr24feed` near the top of the output.

6.4 Configure fr24feed

Run the signup wizard:

```
fr24feed --signup
```

Key answers:

- **Receiver type**: `Beast` (option for network Beast feed)
- **Beast host**: `decoder`
- **Beast port**: `30005`
- **MLAT**: `no` (feeding multiple networks)
- **Basestation feed on port 30003**: `no`

After signup, verify the config:

```
cat /etc/fr24feed.ini
```

Expected:

```
receiver="beast-tcp"
fr24key="<your-key>"
host="decoder:30005"
bs="no"
raw="no"
mlat="no"
mlat-without-gps="no"
```

Enable and start the service:

```
systemctl enable --now fr24feed
```

Check status:

```
systemctl status fr24feed
journalctl -u fr24feed -n 10 --no-pager
```

Look for:

```
[reader][i]Connected to the receiver, configuring
[reader][i]Configured, processing messages
[feed][n]connected via UDP
[feed][n]working
```

> **Note**: If fr24feed cannot connect to `decoder` and the logs show `Could not connect to tcp://decoder:30005`, the IPv6 disable setting may not have taken effect yet. Exit the container and restart it:
>
> ```
> exit
> incus restart flightradar24
> ```
>
> Then check the logs again — fr24feed should connect automatically on restart.

Exit the container:

```
exit
```

6.5 Enable Autostart

```
incus config set flightradar24 boot.autostart true
```

Part Seven: Verify the Complete Setup

Check all containers are running:

```
incus list
```

Check all services are healthy:

```
incus exec decoder -- systemctl status dump1090-fa --no-pager
incus exec flightaware -- systemctl status piaware --no-pager
incus exec flightradar24 -- systemctl status fr24feed --no-pager
```

Check for any failed systemd units on the host:

```
systemctl --failed
```

Do a full reboot to verify everything comes up automatically:

```
sudo reboot
```

After unlocking via Dropbear and booting, all three containers should start automatically and all services should connect without intervention.