ULTIMATE DNS SHIELD

Build Your Own Private, Ad-Free DNS Server

with a Raspberry Pi 4

Pi-hole + Unbound + Docker

Cherif Jebali
2026 Edition

License — CC BY-NC 4.0

This guide is released under the Creative Commons Attribution-NonCommercial 4.0 International License (CC BY-NC 4.0).

You are free to:

Share and redistribute this guide in any format or medium.

Adapt, remix, and build upon this material.

Under the following terms:

Attribution — You must give appropriate credit to the original author (Cherif Jebali) and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the author endorses you or your use.

NonCommercial — You may not use this material for commercial purposes. This includes selling, reselling, or including it in a paid product or service.

Full license text: https://creativecommons.org/licenses/by-nc/4.0/

Disclaimer

The information in this guide is provided in good faith for educational purposes. Network configurations vary by hardware and ISP. The author accepts no liability for misconfiguration, data loss, or security incidents arising from the use of this guide. Always maintain a backup connection and test changes in a controlled environment before applying them to a production network.

Table of Contents

CHAPTER 1

Introduction

Before writing a single command, you need to understand what you're protecting yourself from — and why a Raspberry Pi is the right tool for the job.

The Hidden Cost of "Free" DNS

Every time your browser, phone, or smart TV connects to a website, it first needs to translate a human-readable domain name (like *google.com*) into a machine-readable IP address. This translation is handled by the **Domain Name System (DNS)** — the internet's phonebook.

By default, most home networks rely on DNS resolvers provided by Google (8.8.8.8), Cloudflare (1.1.1.1), or their ISP. These services are fast, free, and reliable. But they come at a price: **every domain you visit is logged, analyzed, and monetized.** Your browsing habits, app usage patterns, and even your sleep schedule can be inferred from DNS traffic alone.

In 2013, Edward Snowden's revelations confirmed that intelligence agencies were already tapping into commercial DNS infrastructure at scale. Since then, nothing fundamental has changed — the same data collection happens today, only more efficiently. DNS-over-HTTPS (DoH) shifts the problem rather than solving it: you're still trusting a third party, just a different one.

The only real solution is to run your own recursive DNS resolver — one that queries the authoritative root nameservers directly, without routing your queries through any intermediary.

What Is a Recursive Resolver?

When you type *example.com* in your browser, a forwarding resolver (like Google's) simply passes your query to another server that already knows the answer. Your query is logged on the way in and on the way out.

A **recursive resolver** works differently. It starts at the root of the DNS hierarchy and walks the tree itself:

- "Who is responsible for *.com*?" > asks a root nameserver
- "Who handles *example.com*?" > asks the .com TLD server
- "What is the IP of *example.com*?" > asks the domain's own nameserver

No third-party resolver ever sees your query. **This is what Unbound does in this guide.** Pi-hole sits in front of it, acting as a sinkhole that blocks known ad and tracker domains before they even reach Unbound.

What You Will Build

By the end of this guide, your home network will have a fully self-hosted DNS stack running on a Raspberry Pi 4:

Component	Role	IP
Pi-hole	DNS sinkhole — blocks ads & trackers, listens on port 53	172.20.0.3 (Docker internal)
Unbound	Recursive resolver — queries root servers directly	172.20.0.2 (Docker internal)
Raspberry Pi 4	Host machine — accessible on your LAN	192.168.0.x (your router reserves this)

The two Docker containers communicate over an isolated bridge network called `pihole_net`. Unbound is **never directly reachable from your LAN** — only Pi-hole exposes port 53 to the outside. This "defense in depth" design means that even if Pi-hole were compromised, the resolver layer would remain isolated.

Prerequisites

This guide assumes:

- You own a Raspberry Pi 4 (2GB RAM or more recommended)
- You have a microSD card (32 GB, Class 10 or A1 rated)
- You can access your home router's admin panel to change DNS settings
- You are comfortable opening a terminal and typing commands
- You have about 2–3 hours available for the initial setup

> **[i] No Linux experience?**
> Every command in this guide is explained in plain English. You will understand *why* you are typing something, not just *what* to type. The goal is not only a working setup — it's a setup you can maintain and troubleshoot.

CHAPTER 2

Hardware Guide

Choosing the right hardware ensures your DNS server is reliable 24/7 — because a DNS server that drops offline leaves your entire network without internet access.

Why Hardware Matters for a DNS Server

Unlike a desktop computer you only use a few hours a day, your DNS server will run **continuously**, handling every DNS query from every device on your network. A typical household with smartphones, laptops, smart TVs, and IoT devices can generate thousands of DNS queries per hour.

The Raspberry Pi 4 is the ideal hardware for this role: low power consumption (~3–7 W), fanless operation, and enough processing power to handle your entire household's DNS traffic without breaking a sweat.

Hardware Checklist

Item	Minimum	Recommended
Raspberry Pi board	RPi 3B+ (1 GB RAM)	RPi 4 (2 GB+ RAM)
microSD card	16 GB Class 10	32 GB A1 / A2 rated
Power supply	5V / 2.5A micro-USB	Official RPi 4 PSU (5V/3A USB-C)
Network connection	Wi-Fi (not ideal)	Ethernet cable (CAT5e or better)
Case / enclosure	None (bare board)	Any passive-cooled case
SD card reader	Required for flashing	USB 3.0 reader preferred

Why Ethernet Over Wi-Fi

A DNS server is on the critical path of every network connection your devices make. If it is unreachable for even a few hundred milliseconds, web pages stall, apps freeze, and streaming buffers. Wi-Fi introduces three reliability problems that ethernet avoids:

- **Packet loss:** Wi-Fi drops packets under interference. DNS queries use UDP — dropped packets mean query timeouts visible as slow page loads.
- **Jitter:** Wireless latency varies with signal quality. A consistent 1 ms wired response is always better than an unpredictable 5–50 ms wireless one.
- **Association drops:** The Pi may temporarily lose its Wi-Fi association during heavy traffic, causing a full DNS outage for all devices.

> **[i] SD Card Longevity**
> Your Pi will write Pi-hole's query log to the SD card continuously. Standard SD cards wear out from write cycles. Use an A1 or A2 rated card, or better yet — configure Pi-hole to reduce logging frequency in its settings (Settings > System > Privacy). This significantly extends the card's life.

About the SD Card Capacity

Pi-hole's database and blocklists can grow over time. Starting with a 32 GB card gives you plenty of headroom. Keep an eye on disk usage with:

```
# Check disk usage on the root partition
df -h /
```

A healthy setup should use less than 20–30% of a 32 GB card. If you see usage climbing above 80%, check Pi-hole's database size under Settings > System and consider adjusting log retention.

CHAPTER 3

Installing Raspberry Pi OS

A clean, minimal OS is the foundation of a secure server. We install Raspberry Pi OS Lite — no desktop, no bloat, just a lean Debian base.

Step 1 — Download Raspberry Pi Imager

The official Raspberry Pi Imager is the simplest and most reliable way to flash your SD card. Download it from:

```
https://www.raspberrypi.com/software/
```

Available for Windows, macOS, and Linux. Install it, insert your microSD card via a card reader, and launch the app.

Step 2 — Select OS and Configure

In Raspberry Pi Imager, click **"Choose OS"** > **"Raspberry Pi OS (other)"** → **"Raspberry Pi OS Lite (64-bit)"**. This is the minimal, server-grade image with no graphical interface — exactly what we want.

> **[!] Why 64-bit?**
> Docker on ARM works best with 64-bit. The mvance/unbound-rpi image and pihole/pihole images both have optimized 64-bit (aarch64) builds that perform better and receive longer support.

Before clicking "Write", click the **gear icon** (or press Ctrl+Shift+X) to open the **Advanced Options**. This is where you configure the Pi for headless operation — no monitor or keyboard required:

Setting	Value
Set hostname	e.g. `pihole` or `raspberrypi`
Enable SSH	Checked — Use password authentication (for initial setup only)
Set username	Choose a non-obvious username (not "pi")

Set password	Strong password — we will disable password SSH login later
Configure Wi-Fi	Leave empty if using ethernet (recommended)
Set locale	Your timezone and keyboard layout

Step 3 — Flash and First Boot

Click "Write" and confirm. The imager will download the image (if needed), write it, and verify the flash. This takes 5–10 minutes.

Insert the SD card into the Pi, connect the ethernet cable, then plug in the power supply. The Pi will boot in ~45–60 seconds on first start.

Step 4 — Find Your Pi on the Network

Your router has automatically assigned an IP to the Pi via DHCP. You can find it several ways:

- **Hostname resolution (easiest):** Raspberry Pi OS enables mDNS by default. From any computer on the same network, simply ping the hostname you set during flashing:

```
# Use the default hostname:
ping raspberrypi.local

# Or with the custom hostname you chose during flashing:
ping YOUR_HOSTNAME.local

# The IP address shown in the reply is your Pi's current address
```

- **Router admin panel:** Log into your router and look in the DHCP client list — your Pi will appear with its hostname and the assigned IP address.

Step 5 — First SSH Connection

Connect to your Pi from your computer:

```
ssh username@YOUR_PI_IP

# You will see a fingerprint prompt — type yes and press Enter
# Then enter the password you set during flashing
```

Step 6 — Reserve a Static IP on Your Router

A DNS server **must have a stable, predictable IP address**. If the Pi's IP changes, all your devices lose DNS service until you update the router settings.

In your router's admin panel, find the DHCP settings and create a "static DHCP lease" or "DHCP reservation" that permanently assigns a fixed IP to your Pi's MAC address. The Pi's MAC address is shown in the DHCP client list or with:

```
ip link show eth0

# Look for the line starting with "link/ether"
# Example output:
# link/ether dc:a6:32:xx:xx:xx brd ff:ff:ff:ff:ff:ff
# ^^^^^^^^^^^^^^^^ this is your MAC address
```

Step 7 — Update the System

Before installing anything, update all packages to their latest versions. This closes known security vulnerabilities in the base OS:

```
# Update the package list
sudo apt update

# Upgrade all installed packages
sudo apt upgrade -y

# Install prerequisites needed for Docker installation
sudo apt install -y curl ca-certificates gnupg
```

> **[i] Reboot after upgrade**
> If the kernel or systemd was updated, reboot with `sudo reboot` and reconnect after ~30 seconds.

CHAPTER 4

Securing SSH

SSH is the only remote access method to your server. Locking it down properly is not optional — it's the minimum baseline for any internet-connected Linux machine.

The Problem with Password Authentication

The default SSH setup allows anyone who knows your username and password to log in. Automated bots continuously scan the internet for open SSH ports and attempt credential stuffing with common passwords. Even on a local network, a compromised device could attack your Pi.

The solution is **SSH key-based authentication**: instead of a password, your identity is proven by a cryptographic key pair. The private key never leaves your computer. The Pi only stores your public key. No password, no brute force attack surface.

Step 1 — Generate an SSH Key Pair (on your computer, not the Pi)

Run this on your personal laptop or desktop — **not on the Pi**. We use Ed25519, a modern elliptic curve algorithm that is both faster and more secure than the older RSA-2048:

```
# On Linux / macOS:
ssh-keygen -t ed25519 -C "pihole-admin"

# On Windows (PowerShell or Windows Terminal):
ssh-keygen -t ed25519 -C "pihole-admin"
```

When prompted, you can set a passphrase to protect the key file itself (recommended for extra security) or press Enter for no passphrase.

This creates two files:

- `~/.ssh/id_ed25519` — your **private** key. Never share or copy this file anywhere.
- `~/.ssh/id_ed25519.pub` — your **public** key. This is what goes on the Pi.

Step 2 — Copy the Public Key to Your Pi

```
ssh-copy-id -i ~/.ssh/id_ed25519.pub username@YOUR_PI_IP

# Enter your password one last time when prompted
# This is the last time you will ever need it
```

If `ssh-copy-id` is not available (e.g. on Windows), do this manually:

```
# Run this in one command:
cat ~/.ssh/id_ed25519.pub | ssh username@YOUR_PI_IP \
  "mkdir -p ~/.ssh && chmod 700 ~/.ssh && \
   cat >> ~/.ssh/authorized_keys && \
   chmod 600 ~/.ssh/authorized_keys"
```

Step 3 — Disable Password Authentication

On the Pi, open the SSH daemon configuration:

```
sudo nano /etc/ssh/sshd_config
```

Find and set — or add — these exact lines:

```
PubkeyAuthentication yes

PasswordAuthentication no

PermitRootLogin no

AuthorizedKeysFile .ssh/authorized_keys
```

Save the file (Ctrl+O, Enter, Ctrl+X in nano), then restart SSH:

```
sudo systemctl restart ssh
```

> **[x] Do not close your current session yet!**
> Open a **new terminal window** and verify you can log in with your key before closing the current session. If something is misconfigured, you still have access to fix it from the existing session.

Step 4 — Verify Key Authentication

In the new terminal:

```
ssh -i ~/.ssh/id_ed25519 username@YOUR_PI_IP

# You should be logged in without being asked for a password
```

To confirm password auth is truly disabled, try:

```
ssh -o PreferredAuthentications=password username@YOUR_PI_IP

# Expected output:
# Permission denied (publickey)
```

> **[v] Your SSH is now hardened**
> From this point on, only someone with physical access to your private key file can authenticate to your Pi. Brute force attacks against SSH are no longer a concern.

Step 5 — Simplify Future Connections (Optional)

Create an SSH config entry on your computer to avoid typing the full command each time:

```
# Edit ~/.ssh/config (create it if it doesn't exist):
nano ~/.ssh/config

# Add:
Host pihole
    HostName YOUR_PI_IP
    User username
    IdentityFile ~/.ssh/id_ed25519
```

You can now connect with simply:

```
ssh pihole
```

CHAPTER 5

Installing Docker

Docker is the engine that runs Pi-hole and Unbound as isolated, reproducible containers. Once you understand why we use it, you'll never want to go back to installing services directly on the OS.

Why Docker for This Setup?

You could install Pi-hole and Unbound directly on the OS using their native installers. Many guides do exactly that. Here's why we don't:

- **Isolation:** Pi-hole and Unbound each run in their own filesystem namespace. A bug or vulnerability in one cannot directly affect the other or the host OS.
- **Reproducibility:** Your entire setup is defined in a single `docker-compose.yml` file. To rebuild from scratch on a new SD card, you copy one file and run one command.
- **Clean updates:** Updating Pi-hole means pulling a new image and restarting the container. No package manager conflicts, no leftover config files.
- **Custom networking:** Docker lets us create an isolated bridge network where Unbound is only reachable from Pi-hole — not from the internet or your LAN.
- **Rollback:** If an update breaks something, you can pin the previous image version in your compose file and redeploy in seconds.

Installation — Official Method

Always install Docker using the **official Docker repository**, not the version in Debian's package manager. The Debian-packaged Docker is often outdated and missing features.

```
# 1. Install dependencies
sudo apt install -y ca-certificates curl gnupg

# 2. Create the keyring directory
sudo install -m 0755 -d /etc/apt/keyrings

# 3. Add Docker's official GPG key
curl -fsSL https://download.docker.com/linux/debian/gpg \
  | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
sudo chmod a+r /etc/apt/keyrings/docker.gpg

# 4. Add the Docker repository
echo \
  "deb [arch=$(dpkg --print-architecture) \
  signed-by=/etc/apt/keyrings/docker.gpg] \
  https://download.docker.com/linux/debian \
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" \
  | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

# 5. Install Docker Engine
sudo apt update
sudo apt install -y docker-ce docker-ce-cli containerd.io \
  docker-buildx-plugin docker-compose-plugin
```

Post-Installation Steps

By default, only `root` can run Docker commands. Add your user to the `docker` group to manage containers without prefixing every command with `sudo`:

```
# Add yourself to the docker group
sudo usermod -aG docker $USER

# Apply the new group membership (or log out and back in)
newgrp docker
```

Verify the installation works:

```
docker run hello-world
# You should see "Hello from Docker!" — if so, Docker is working correctly

# Check Docker Compose is available:
docker compose version
```

[i] Docker Compose v2

This guide uses the modern plugin-style `docker compose` (with a space), not the older `docker-compose` (with a dash). The plugin version is installed automatically with Docker Engine from the official repo.

Docker Concepts You Need to Know

Term	What it means
`Image`	A read-only template (like a recipe). `pihole/pihole:latest` is an image.
`Container`	A running instance of an image. You can run multiple containers from the same image.
`Volume`	A directory on the host that is mounted inside the container. Data in volumes persists even when the container is recreated.
`Network`	An isolated virtual network connecting containers. Our `pihole_net` is a custom bridge network.
`docker-compose.yml`	A YAML file that describes your entire multi-container setup: images, volumes, networks, ports, and environment variables.

CHAPTER 6

Pi-hole + Unbound

This is the core of the guide. We will deploy two Docker containers on an isolated network, configure Unbound as a recursive resolver, and point Pi-hole to use it as its upstream DNS.

Architecture Deep Dive

Before running any command, it is worth understanding exactly what happens when a device on your network resolves a domain name after this setup:

1. Your phone asks: "What is the IP of *spotify.com*?"
2. The query arrives at **Pi-hole** on port 53 of your Pi (YOUR_PI_IP)
3. Pi-hole checks its blocklist. *spotify.com* is not an ad domain — the query passes.
4. **Pi-hole forwards** the query to Unbound at 172.20.0.2:5335 (internal Docker network)
5. **Unbound** starts recursive resolution: asks a root nameserver > gets TLD > gets authoritative server
6. Unbound returns the IP to Pi-hole, which caches it and returns it to your phone
7. Your phone connects directly to Spotify's servers. No DNS intermediary ever saw the query.

The critical insight is step 5: **Unbound talks directly to authoritative nameservers.** There is no Google, no Cloudflare, no ISP DNS in this chain. Your queries are resolved from source.

Directory Structure

Create the directory structure before writing any files:

```
mkdir -p ~/services/pihole/etc-pihole
mkdir -p ~/services/pihole/etc-unbound
cd ~/services/pihole
```

The docker-compose.yml — Explained Line by Line

Create the compose file:

```
nano ~/services/pihole/docker-compose.yml
```

The file defines two services communicating over a private Docker network. Here is each block explained:

Service 1 — Unbound (the recursive resolver)

```
services:
  unbound:
    container_name: unbound
    image: mvance/unbound-rpi:latest
    # ARM-optimized image for Raspberry Pi
    volumes:
      - ./etc-unbound/unbound.conf:/opt/unbound/etc/unbound/unbound.conf:ro
    # Mount our config file as read-only inside the container
    networks:
      pihole_net:
        ipv4_address: 172.20.0.2
    # Fixed internal IP — Pi-hole will always find Unbound at this address
    restart: unless-stopped
    # Automatically restart on crash or system reboot
```

Service 2 — Pi-hole (the DNS sinkhole)

```
  pihole:
    container_name: pihole
    image: pihole/pihole:latest
    depends_on:
      - unbound
    # Wait for Unbound to be ready before starting
    networks:
      pihole_net:
        ipv4_address: 172.20.0.3
    ports:
      - "53:53/tcp"
      - "53:53/udp"
    # Expose DNS port to your LAN (TCP + UDP)
      - "80:80/tcp"
      - "443:443/tcp"
    # Pi-hole web admin interface
    environment:
      - TZ=Europe/Paris
    # Replace with your own timezone
      - FTLCONF_webserver_api_password=YOUR_STRONG_PASSWORD
      - FTLCONF_dns_listeningMode=all
    # Listen on all interfaces inside the container
    volumes:
      - ./etc-pihole:/etc/pihole
    # Persistent storage for config and query database
    restart: unless-stopped

networks:
  pihole_net:
    driver: bridge
    ipam:
      config:
        - subnet: 172.20.0.0/24
    # Private subnet — Unbound is never reachable from your LAN directly
```

> **[i] About FTLCONF_dns_listeningMode**
>
> Setting this to `all` tells Pi-hole's FTL daemon to listen on all interfaces inside the container — including the Docker bridge. Without this, Pi-hole would only listen on `localhost` and would not accept queries from your network.

The unbound.conf -- Explained

Create the Unbound configuration file:

```
nano ~/services/pihole/etc-unbound/unbound.conf
```

```
server:
    interface: 0.0.0.0
    port: 5335 # Pi-hole forwards queries here

    do-ip4: yes
    do-ip6: yes
    do-udp: yes
    do-tcp: yes

    # -- Access control --
    access-control: 0.0.0.0/0 refuse # deny all by default
    access-control: 127.0.0.0/8 allow # localhost
    access-control: 172.20.0.0/24 allow # Docker bridge only

    # -- TCP tuning --
    incoming-num-tcp: 100
    outgoing-num-tcp: 100
    tcp-idle-timeout: 30000

    edns-buffer-size: 1232
    verbosity: 1

    # -- Privacy --
    hide-identity: yes
    hide-version: yes

    domain-insecure: "verteiltesysteme.net"
```

Every directive in this file has a deliberate purpose. Here are the most important ones:

- **access-control:** The most critical security setting. The first rule refuses all connections by default. The second and third rules then open access only to localhost and our Docker subnet. This means Unbound is *completely unreachable* from your LAN — only Pi-hole can query it.
- **edns-buffer-size: 1232:** DNS responses that exceed the EDNS buffer size get fragmented into multiple packets. Fragmented UDP packets are prone to being dropped by some routers and firewalls. 1232 bytes is the safe recommended value from RIPE NCC's research (it fits within a standard 1280-byte IPv6 minimum MTU).
- **hide-identity / hide-version:** By default, Unbound answers queries like "CHAOS TXT id.server" with its hostname and version number. This information helps attackers target known vulnerabilities. These two settings silence that.

- **incoming/outgoing-num-tcp: 100:** Increases the number of simultaneous TCP connections Unbound can handle. Relevant if many devices make DNS queries simultaneously — which they do during peak hours.
- **domain-insecure: "verteiltesysteme.net":** Tells Unbound to skip DNSSEC validation for this specific domain. This is used to allow the standard DNSSEC test queries (sigfail / sigok) to work as expected in Chapter 7.

Deployment

Start both containers:

```
cd ~/services/pihole
docker compose up -d

# Verify both are running:
docker ps

# Check Pi-hole logs for errors:
docker logs pihole --tail 30
```

Configuring Pi-hole's Upstream DNS

Pi-hole needs to know that it should forward queries to Unbound. Open Pi-hole's web interface at `http://YOUR_PI_IP/admin` and log in with your password.

Navigate to: **Settings > DNS**

In the "Upstream DNS Servers" section, uncheck all default servers and enter a custom one:

```
Custom 1 (IPv4): 172.20.0.2#5335
```

This tells Pi-hole to forward all non-blocked queries to Unbound at its Docker-internal IP on port 5335. Make sure to enable **DNSSEC** while you're in the DNS settings.

> **[i] Why port 5335?**
> Port 53 is the standard DNS port, but it's already used by Pi-hole itself. Unbound listens on port 5335 inside the Docker network to avoid the conflict. Since the containers communicate over the internal `pihole_net` network, this port is never exposed to your LAN.

CHAPTER 7

Configuring Your Router

The final step to route all your network traffic through Pi-hole: tell your router to hand out your Pi's IP as the DNS server for every device.

How DHCP DNS Distribution Works

When a device joins your network, your router's DHCP server gives it an IP address. As part of this process, the router also tells the device which DNS server to use. By changing this setting from your ISP's DNS to your Pi's IP, **every device on your network automatically starts using Pi-hole** — no configuration needed on individual devices.

> **[!] Router interfaces vary**
> Every router manufacturer uses a different admin interface. The concepts are identical across all routers, but the exact menu names differ. Look for sections named: "DHCP Settings", "LAN Setup", "DNS Server", or "Network Settings". Consult your router's manual if you can't find it.

Step-by-Step: Pointing DNS to Your Pi

1. Open your router admin panel:

```
http://192.168.0.1 (most common)
http://192.168.1.1 (alternative)
# Or check the label on your router
```

2. Find the DHCP or DNS settings. Look for fields like:

• "Primary DNS Server" or "DNS Server 1"

• "Secondary DNS Server" or "DNS Server 2"

3. Set the values:

```
Primary DNS: YOUR_PI_IP < Your Raspberry Pi's IP
Secondary DNS: 9.9.9.9 < Quad9 (fallback if Pi is offline)
```

> **[i] Why set a secondary DNS?**
> If your Pi reboots for an update or loses power, all DNS queries would fail without a fallback. Quad9 (9.9.9.9) is a privacy-respecting alternative to Google that also blocks known malicious domains. Your devices switch to it automatically if Pi-hole is unreachable.

4. Save the settings. Reconnect your devices or wait for DHCP leases to renew (usually within 24 hours, or immediately after reconnecting to Wi-Fi).

Testing Ad Blocking

From any computer on your network, test that Pi-hole is blocking ads:

```
# Check that your DNS is now served by Pi-hole:
nslookup google.com
# The "Server:" line should show YOUR_PI_IP

# Test ad blocking — this domain is in all major blocklists:
nslookup doubleclick.net
# Should return 0.0.0.0 (blocked)

# Or with dig:
dig @YOUR_PI_IP doubleclick.net
# Look for "0.0.0.0" in the ANSWER SECTION
```

Verifying DNSSEC is Working

DNSSEC prevents DNS cache poisoning by cryptographically signing DNS responses. Two test domains exist specifically for verifying DNSSEC validation:

```
# This domain has a DELIBERATELY BROKEN DNSSEC signature:
dig @YOUR_PI_IP sigfail.verteiltesysteme.net
# Expected result: SERVFAIL — Unbound correctly rejects the invalid signature

# This domain has a VALID DNSSEC signature:
dig @YOUR_PI_IP sigok.verteiltesysteme.net
# Expected result: NOERROR with a valid IP address
```

If `sigfail` returns `SERVFAIL` and `sigok` returns `NOERROR`, your DNSSEC validation is working perfectly.

Verifying Recursive Resolution

To confirm Unbound is resolving queries recursively (not forwarding to a third party), check Pi-hole's query log. In the web interface, go to **Queries** and look at any resolved query. The "Forwarded to" column should show `172.20.0.2#5335` — your local Unbound instance.

> **[v] Setup complete!**
> At this point, your entire network's DNS traffic is being resolved locally, with ads and trackers blocked at the DNS level. No query leaves your network to a third-party resolver.

CHAPTER 8

Additional Security Hardening

Pi-hole and Unbound are now running. This chapter adds the final layers of defense: a firewall, intrusion prevention, and automatic security updates.

UFW — Uncomplicated Firewall

UFW provides a simple interface to `iptables`, Linux's built-in packet filtering system. Without a firewall, any port your Pi opens is accessible to any device on your network (and potentially beyond). With UFW, you define exactly what is allowed — everything else is dropped.

Install UFW:

```
sudo apt install -y ufw
```

Configure the rules — **do this before enabling UFW**:

```
# Set default policies: block all incoming, allow all outgoing
sudo ufw default deny incoming
sudo ufw default allow outgoing

# Allow SSH (if you skip this, you will lock yourself out!)
sudo ufw allow 22/tcp

# Allow DNS queries from your local network only
sudo ufw allow from YOUR_NETWORK/24 to any port 53

# Allow Pi-hole web interface from local network only
sudo ufw allow from YOUR_NETWORK/24 to any port 80
sudo ufw allow from YOUR_NETWORK/24 to any port 443

# Enable the firewall
sudo ufw enable

# Verify all rules are correct
sudo ufw status verbose
```

> **[x] Enable SSH rule BEFORE running ufw enable**
> If you forget to add the SSH rule and then enable UFW, you will be locked out of your Pi remotely. The only recovery is physical access (keyboard + monitor). Always verify with `sudo ufw status` before enabling.

After enabling, your rule set should look like this:

```
To Action From
-- ------ ----
22/tcp ALLOW IN Anywhere
53 ALLOW IN YOUR_NETWORK/24
80/tcp ALLOW IN YOUR_NETWORK/24
443/tcp ALLOW IN YOUR_NETWORK/24
```

Fail2Ban — Intrusion Prevention

Fail2Ban monitors log files for repeated failed authentication attempts and automatically bans the offending IP address. Even on a local network, this adds a layer of protection against lateral movement if another device is compromised.

Fail2Ban should already be installed. Verify and check the SSH jail status:

```
sudo fail2ban-client status
# Should show: Jail list: sshd

sudo fail2ban-client status sshd
# Shows current banned IPs and statistics
```

The default configuration is functional, but we can tighten it. Create a local override file (never edit the main config — it gets overwritten on updates):

```
sudo nano /etc/fail2ban/jail.local
```

Add this content:

```
[DEFAULT]
# Ban duration: 1 hour
bantime = 1h

# Time window to count failures
findtime = 10m

# Number of failures before banning
maxretry = 3

[sshd]
enabled = true
port = ssh
logpath = %(sshd_log)s
backend = %(sshd_backend)s
maxretry = 3
bantime = 2h
```

```
# Apply the new configuration:
sudo systemctl restart fail2ban
sudo fail2ban-client status sshd
```

Automatic Security Updates

New kernel and library vulnerabilities are discovered regularly. The `unattended-upgrades` package automatically installs security updates without requiring manual intervention:

```
sudo apt install -y unattended-upgrades
sudo dpkg-reconfigure --priority=low unattended-upgrades
# Select "Yes" when prompted

# Verify it's enabled:
sudo systemctl status unattended-upgrades
```

> **[i] Docker images are not covered by unattended-upgrades**
> Automatic OS updates cover Debian packages, not Docker images. You need to manually update Pi-hole and Unbound containers periodically. See the Bonus chapter for the update procedure.

Final Security Checklist

Check	Verify with
SSH key auth only	`PasswordAuthentication no in /etc/ssh/sshd_config`
Root login disabled	`PermitRootLogin no in /etc/ssh/sshd_config`
UFW enabled	`sudo ufw status shows "Status: active"`
Fail2Ban running	`sudo fail2ban-client status shows sshd jail`
Auto-updates active	`systemctl status unattended-upgrades`
DNS on fixed IP	`Router DHCP reservation for Pi's MAC address`
Pi-hole password set	`Strong password in docker-compose.yml`

CHAPTER 9

Bonus: Blocklists & Maintenance

Your setup is running. This chapter covers everything you need to maintain it, improve it, and recover from failure.

Recommended Blocklists

Pi-hole's effectiveness depends entirely on the quality of its blocklists. The default list is a good start, but the community maintains much more comprehensive ones. Here are the best, tested for low false-positive rates:

Steven Black's Unified Hosts

The gold standard. Merges multiple well-maintained lists. ~200,000 entries covering ads, malware, and trackers.

```
https://raw.githubusercontent.com/StevenBlack/hosts/master/hosts
```

Hagezi Multi Pro

Aggressive but well-maintained. Excellent tracker coverage with very few false positives. Highly recommended.

```
https://raw.githubusercontent.com/hagezi/dns-blocklists/main/hosts/pro.txt
```

OISD Big

Large comprehensive list specifically tuned for minimal false positives. Good for households with smart home devices.

```
https://big.oisd.nl/domains
```

To add a list: Pi-hole web interface > **Adlists** > paste the URL > Add > then go to **Tools > Update Gravity** to download and apply all lists.

> **[!] More is not always better**
> Adding too many blocklists can block legitimate services your household uses. Start with Steven Black + Hagezi Pro. If something breaks, use Pi-hole's "Whitelist" to add exceptions, or check which list is blocking a domain under Tools > Querylog.

Testing Commands

A toolkit of commands to verify your setup is working at every layer:

```
# -- Basic connectivity --
# Is Pi-hole responding?
dig @YOUR_PI_IP google.com

# -- Ad blocking --
# Should return 0.0.0.0 (blocked)
dig @YOUR_PI_IP doubleclick.net
dig @YOUR_PI_IP googleadservices.com

# -- DNSSEC validation --
# Should return SERVFAIL (broken signature correctly rejected)
dig @YOUR_PI_IP sigfail.verteiltesysteme.net
# Should return NOERROR (valid signature accepted)
dig @YOUR_PI_IP sigok.verteiltesysteme.net

# -- Verify recursive resolution --
# Trace the full resolution path from root
dig @YOUR_PI_IP example.com +trace

# -- Container health --
docker ps # Check container status
docker logs pihole --tail 50 # Pi-hole logs
docker logs unbound --tail 50 # Unbound logs
docker stats --no-stream # CPU and memory usage
```

Fallback DNS: What Happens if the Pi Goes Down?

Your secondary DNS on the router (9.9.9.9) serves as automatic fallback. Most operating systems will switch to the secondary DNS after 2–5 seconds of timeout on the primary. During this time:

- Ad blocking is disabled (Quad9 doesn't block ads)
- DNS privacy is temporarily reduced (queries go to Quad9)
- All devices retain internet connectivity

For a more resilient setup, you could run a second Pi-hole instance on another machine as a true redundant DNS server — but for most home users, the Quad9 fallback is a good balance between reliability and simplicity.

Keeping Everything Updated

Docker images do not update automatically. A monthly update routine keeps you protected against Pi-hole and Unbound vulnerabilities:

```
cd ~/services/pihole

# Pull latest images
docker compose pull

# Recreate containers with new images (zero-downtime if Pi-hole restarts fast)
docker compose up -d

# Clean up old images to reclaim disk space
docker image prune -f

# Verify both containers are healthy
docker ps
```

Useful Maintenance Commands

```
# Restart Pi-hole only (e.g. after config change)
docker compose restart pihole

# Completely stop and remove containers (data in volumes is preserved)
docker compose down

# Check how much disk Pi-hole's database is using
du -sh ~/services/pihole/etc-pihole/

# View Pi-hole statistics from the terminal
docker exec pihole pihole -c

# Flush the DNS cache
docker exec pihole pihole restartdns

# Check system resource usage
htop
free -h
df -h
```

Congratulations

You now have a production-grade, self-hosted DNS infrastructure running on a €60 computer. Every device on your network resolves DNS privately, blocks ads and trackers at the network level, and validates DNSSEC signatures — without trusting any third-party resolver.

This is not just a home lab experiment. This is the same architecture used by security-conscious organizations: two isolated services, custom networking, hardened OS, and automated recovery. You built it yourself, and you understand every component of it.

> **[v] What to explore next**
>
> Consider adding a VPN server (WireGuard) to your Pi so you can benefit from Pi-hole's ad blocking even when you're on a mobile network. Pair it with a monitoring tool like Grafana to visualize your DNS traffic over time.