PRACTICAL PORTS

Jail-based DNS AdBlocking Tutorial

BY BENEDICT REUSCHLING

One of the things that drew me to FreeBSD early on is its ability to easily run services. Services can be system services that come with the operating system (the simplest example would be the ssh daemon), or through third-party software installed via pkg or ports. The process is the same for both: you add a line to /etc/rc.conf to enable it (either via sysrc or "service ... enable") to run the next time the system boots. Then there is usually a configuration file to make settings that customize the service to your needs. Typically, this requires entering which IP address or DNS hostname to listen to, a network port, and some specifics of the software. From then on, it's either starting the service directly (using "service ... start"), or at the next reboot, in case it requires loading kernel modules that kldload can't load for the running system (which is rarely the case these days).

This is straightforward, keeps the configuration of all system services in one convenient location, and is easy to repeat once you've done it for one or two services. Of course, running a whole set of services on the FreeBSD host system works perfectly fine—until it gets more complicated. Running different versions of the same software side by side is perfectly reasonable and not too uncommon. This is done for testing purposes—checking to see if an upgrade works as intended or if certain software still requires an older version as a dependency. One example is trying to run different versions of the PostgreSQL database next to each other. In this case, both pkg and ports check for the versions or will cancel the install operation with a message saying that certain binaries will be put into the same place and hence overwrite each other. This is an undesirable situation, and the user has to make a decision for one or the other version because they can't coexist next to each other.

Of course, running a whole set of services on the FreeBSD host system works perfectly fine—until it gets more complicated.

That is, unless virtualization or containerization is involved. This allows process isolation in separate execution environments using various methods to permit multiple such systems to run on the same hardware. Virtualization adds an extra layer over the operating system that permits the installation of the same or a different operating system with simulated hardware. Containers or jails do this by isolating processes using chroot(8). We'll focus on the latter here, as it is more lightweight in terms of resource use and getting something running fairly quickly.

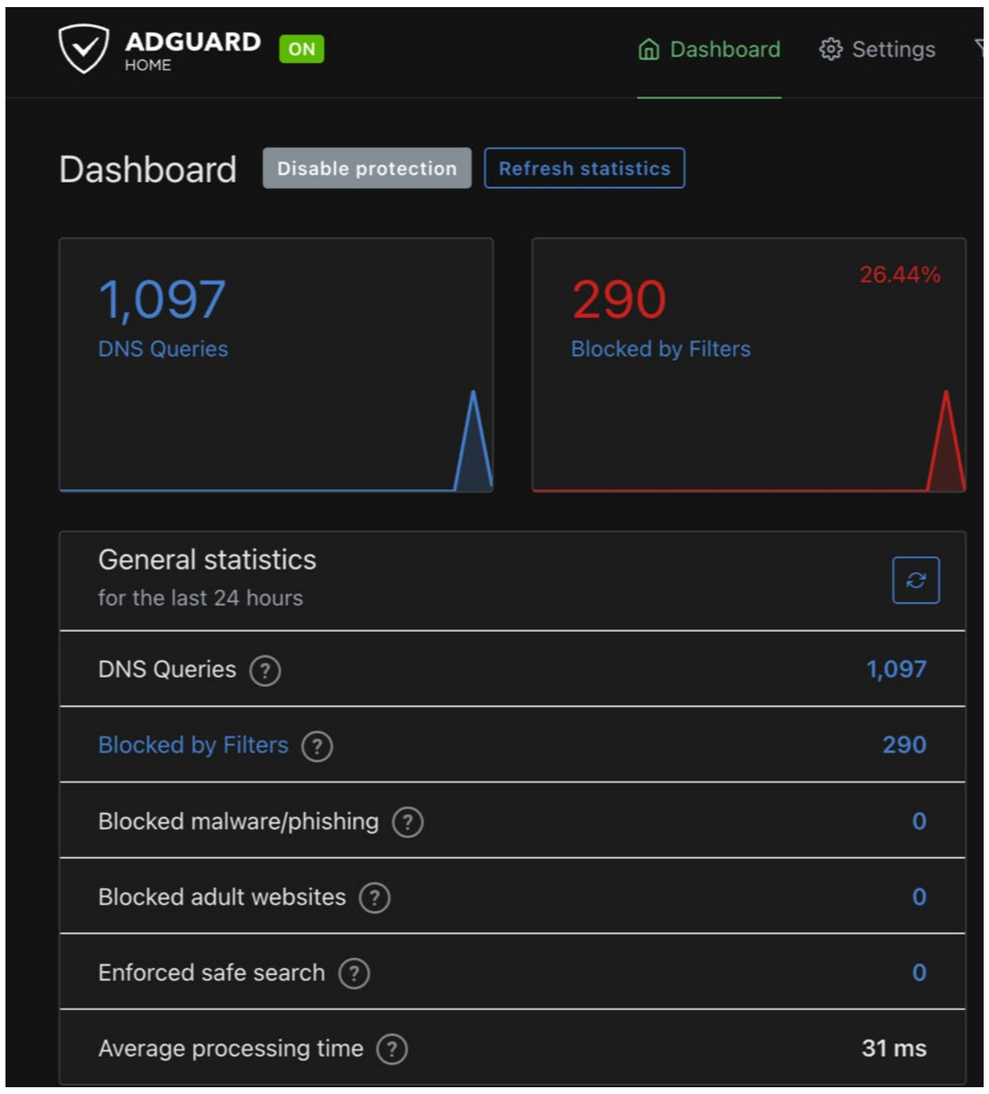
The benefit of this isolation is not only that various, different versions can run side by side, it also allows separation for security reasons. When an application is running in a jail container, the processes inside can't access the host system by default. The application discovers all the usual devices (like networking), directory structure, and files in the right places, but in reality, it is a separate environment that mimics the host system behavior and layout. When such a jail becomes compromised somehow, it is easy to stop it without affecting services in other jails or the host system. Access to them is strictly prohibited and isolates any intruders into that particular jail cell.

That also makes it easy to migrate a system to another host by stopping, copying the jail's directory structure to the new location, and starting there again (with a few local modifications like the new IP address, for example). Backup and restore is done in the same way. Multiple such jails are typically managed by jail management software that takes care of creating, modifying, and removing jails.

When an application is running in a jail container, the processes inside can't access the host system by default.

One such jail manager is called Bastille which is written entirely as a shell script. We're taking a closer look at Bastille in this article by going through the process of setting up the host system for it, creating a jail, and starting a service within it based on a template. Such templates allow for sharing configurations in a central repository so as to apply them without the need to know about the inner workings of the service. That way, complicated situations are easy to set up for people who want to get something running quickly.

In this article, we're deploying a service called AdGuard by AdGuard Software Limited. With a running AdGuard service in a network, clients connecting their DNS resolution to it can filter out advertisements from web browsing activities. This helps avoid tracking and user profile building by advertisers, and also helps pages load more quickly since they don't have to transfer ads next to the content the user wants to see. AdGuard does this using filter lists and DNS sinkholing. Based on the filter lists, a known advertisement site gets blocked by AdGuard sending an invalid address response back before it renders the ad in the browser. There are various ways to use the AdGuard service—as a browser extension for an individual device, desktop applications, or by running it as a recursive DNS resolver. Note that AdGuard does not completely protect against all forms of advertisements (especially the ones dynamically embedded in video sites) but does a good job of removing a big piece of them from web pages.



We're starting out with a Raspberry Pi, because this service runs pretty much all the time, and we want a low power usage footprint. I have an RPI3 here, but other devices (including full fledges servers) capable of running FreeBSD work just the same. Install the operating system, apply the latest security patches and lock down remote access using SSH keys.

Environment Setup

I have an old 32GB SSD connected to the Raspberry Pi, which will do most of the I/O heavy operations with a single disk ZFS pool instead of the slower compact flash card. I am running FreeBSD 13.2 at the time of writing this article and I'm fairly confident that future versions will work just as well or with only minor adjustments.

```
# pkg install bastille git-lite
```

Bastille, being a shell script, is fairly quick to install and has no extra dependencies. It may not be as full-blown in some of the functionality that other jail managers have, but it nevertheless works. Git is necessary to clone the adguard home template (and others) from Bastille's GitLab repository.

Next, we create a PF configuration for bastille in /etc/pf.conf like the following:

```
ext_if="ue0" ## <- change ue0 to match host interface

set block-policy return
scrub in on $ext_if all fragment reassemble
set skip on lo

table <jails> persist
nat on $ext_if from <jails> to any -> ($ext_if:0)
rdr-anchor "rdr/*"

block in all
pass out quick keep state
```

```
pass in inet proto tcp from any to any port ssh flags S/SA keep state
pass in inet proto tcp from any to any port bootps flags S/SA keep state
pass in inet proto tcp from any to any port {9100,9124} flags S/SA modulate state
```

Make sure to change the ext_if line at the top in the interface that you are using. On my RPI, the network cable is connected to ue0, so I entered that. The pf.conf will create a table for our jail traffic (using NAT). There are a couple of options that bastille supports for networking, which makes it flexible enough for both an office and home network as well as one provided by a hosting service. They are described in detail here: https://docs.bastillebsd.org/en/latest/chapters/networking.html

I will be using a VNET-based jail as I have an IP address available on my local network. After editing the configuration file, we add an entry to start PF and the pf logging device together with other services upon boot. Bastille should start as well, and we list the name of the jail that we will create for AdGuard (my naming schemes are both legendary and boring):

```
# sysrc pf_enable=YES
# sysrc pflog_enable=YES
# sysrc bastille_enable=yes
# sysrc bastlle_list="adguard"
```

It is good practice to check your firewall ruleset for errors before starting the firewall. Use

```
# pfctl -nvf /etc/pf.conf
```

for such a check. It will echo the whole ruleset upon success or any errors you might have. Note that it can't check for logical errors like blocking your SSH port 22 which may be the only remote way to connect. Fortunately, there is already a rule present to let SSH traffic pass.

Once the check has run, start the PF service, and begin filtering traffic:

```
# service pf start
# service pflog start
```

Expect your SSH connection to drop, so keep a separate terminal open that you can directly access in case you lock yourself out.

Upon reconnect, we need to edit a couple more configuration files. A VNET-enabled jail needs an entry in /etc/devfs.rules (NOT .conf), which may not exist on a fresh install. Simply create it and add the following rules:

```
[bastille_vnet=13]
add path 'bpf*' unhide
```

That enables bastille to see the traffic on the VNET interface and connect the jail to the outside world. This may be a layman's description of what is going on. Luckily, we need not worry about it too much (…maybe I need to on my next networking exam).

Another file we have to visit is /etc/sysctl.conf, which needs the following lines:

```
sysctl net.inet.ip.forwarding=1
sysctl net.link.bridge.pfil_bridge=0
sysctl net.link.bridge.pfil_onlyip=0
sysctl net.link.bridge.pfil_member=0
```

When Bastille runs, it will dynamically create a bridge for us between the RPI's external interface (ue0) and the jail's network interface (vtnet). These two interfaces are connected by a

virtual cable, with one end in the host's interface and the other in the jail, exchanging traffic over it.

Apply those changes also to the running system without having to reboot by issuing:

```
# sysctl -f /etc/sysctl.conf
```

I was totally baffled when I had finished the setup and restarted the PI only to find that the jail could not access the network anymore. A lot of head scratching later, I learned from this exchange

```
https://www.mail-archive.com/freebsd-net@freebsd.org/msg64577.html
```

that this needed an extra line in /boot/loader.conf in FreeBSD 13. This may drive you crazy, so before going insane, add this one to it to make it work for future reboots:

```
if_bridge_load="YES"
```

The bridge interface is properly loaded, and that also causes sysctls to appear, so that sysctl.conf can change them from their default value of 1 to 0. Be that as it may, we visit one last file before we're done.

The Bastille configuration file is located in /usr/local/etc/bastille/bastille.conf. You can either edit it directly (it's well commented) or use sysrc if you don't mind typing a lot. Since I'm running this on a ZFS pool connected to my Raspberry, I set bastille_zfs_enable to give it the name of my pool.

```
# sysrc -f /usr/local/etc/bastille/bastille.conf bastille_zfs_enable=YES
# sysrc -f /usr/local/etc/bastille/bastille.conf bastille_zfs_zpool=rpi3
```

Change the name of your pool in case yours is named differently at the bastille_zfs_zpool line. One option I also changed is the bastille_network_gateway="" option. I entered my default gateway address because I had some trouble down the road getting the jails to resolve names. You may or may not need to set this, but in case you do experience problems, revisit this option and see if that resolves the problem.

Bootstrapping Bastille

Now that all settings are in place, it is time to let Bastille create its dataset structure on the pool we assigned to it. It will download a base FreeBSD 13.2 release and update it with any patches that were published afterwards. Issue the following command and wait until it finishes:

```
# bastille bootstrap 13.2-RELEASE update
Bootstrapping FreeBSD distfiles...
/usr/local/bastille/cache/13.2-RELEASE/MANIFES         782  B 1670 kBps    00s
/usr/local/bastille/cache/13.2-RELEASE/base.tx         168 MB 6526 kBps    26s
Validated checksum for 13.2-RELEASE: base.txz
MANIFEST: 7d1b032a480647a73d6d7331139268a45e628c9f5ae52d22b110db65fdcb30ff
DOWNLOAD: 7d1b032a480647a73d6d7331139268a45e628c9f5ae52d22b110db65fdcb30ff
Extracting FreeBSD 13.2-RELEASE base.txz.

Bootstrap successful.
See ‘bastille -help’ for available commands.

src component not installed, skipped
```

```
Looking up update.FreeBSD.org mirrors... 2 mirrors found.
Fetching metadata signature for 13.2-RELEASE from update2.freebsd.org... done.
Fetching metadata index... done.
Inspecting system... done.
Preparing to download files... done.
The following files will be updated as part of updating to
13.2-RELEASE-p1:
/bin/freebsd-version
/usr/lib/libpam.a
/usr/lib/pam_krb5.so.6
/usr/share/locale/zh_CN.GB18030/LC_COLLATE
/usr/share/locale/zh_CN.GB18030/LC_CTYPE
/usr/share/man/man8/pam_krb5.8.gz
Installing updates...
Restarting sshd after upgrade
Performing sanity check on sshd configuration.
Stopping sshd.
Waiting for PIDS: 1063.
Performing sanity check on sshd configuration.
Starting sshd.
Scanning /usr/local/bastille/releases/13.2-RELEASE/usr/share/certs/blacklisted for certificates...
Scanning /usr/local/bastille/releases/13.2-RELEASE/usr/share/certs/trusted for certificates...
 done.
```

My pool grew these datasets after the bootstrap operation:

```
# zfs list -r rpi3/bastille
NAME                                 USED  AVAIL  REFER  MOUNTPOINT
rpi3                                 621M  28.0G    24K  /rpi3
rpi3/bastille                        584M  28.0G    26K  /usr/local/bastille
rpi3/bastille/backups                 24K  28.0G    24K  /usr/local/bastille/backups
rpi3/bastille/cache                  169M  28.0G    24K  /usr/local/bastille/cache
rpi3/bastille/cache/13.2-RELEASE     169M  28.0G   169M  /usr/local/bastille/cache/13.2-RELEASE
rpi3/bastille/jails                   24K  28.0G    24K  /usr/local/bastille/jails
rpi3/bastille/logs                    24K  28.0G    24K  /var/log/bastille
rpi3/bastille/releases               414M  28.0G    24K  /usr/local/bastille/releases
rpi3/bastille/releases/13.2-RELEASE  414M  28.0G   414M  /usr/local/bastille/releases/13.2-RELEASE
rpi3/bastille/templates               24K  28.0G    24K  /usr/local/bastille/templates
```

Let's run another bootstrap operation, this one is for the template that will provide us with AdGuard Home.

```
# bastille bootstrap https://gitlab.com/bastillebsd-templates/adguardhome
warning: redirecting to https://gitlab.com/bastillebsd-templates/adguardhome.git/
Already up to date.
Detected Bastillefile hook.
[Bastillefile]:
PKG ca_root_nss adguardhome
CP usr /
SYSRC adguardhome_enable=YES
SERVICE adguardhome start
RDR tcp 80 80
RDR udp 53 53

Template ready to use.
```

That was quick. Bastille has its own template language which you can see in the capitalized commands like PKG, CP, etc. They have the same functionality as their system equivalents in lowercase. With those, instructions are run in the jail to set up a service in the proper order. They're mostly self-explaining. The two RDR commands at the end will redirect network ports from the host system into the jail. All other ports are still firewalled, so only port 80 is connected from the host to the jail (and back), as well as DNS port 53. Check back in your /etc/pf.conf for the rdr-anchor "rdr/*" line. This is what makes it so flexible. Instead of opening the port for all jails, each one can open the ports it needs and keep others closed.

It is high time to create and start our first Bastille jail now. Since we're using VNET, we need to pass the -V option to the bastille create command, along with a name for the jail, the release to run, followed by the IP address on the local network assigned to the jail and the hosts network interface for the bridging. Combined, the command is:

```
# bastille create -V adguard 13.2-RELEASE 192.168.2.55 ue0
Valid: (192.168.2.55).
Valid: (ue0).

Creating a thinjail...

[adguard]:
e0a_bastille0
e0b_bastille0
adguard: created

[adguard]:
Applying template: default/vnet...
[adguard]:
Applying template: default/base...
[adguard]:
[adguard]: 0

[adguard]:
syslogd_flags: -s -> -ss

[adguard]:
sendmail_enable: NO -> NO

[adguard]:
sendmail_submit_enable: YES -> NO

[adguard]:
sendmail_outbound_enable: YES -> NO

[adguard]:
sendmail_msp_queue_enable: YES -> NO

[adguard]:
cron_flags:  -> -J 60

[adguard]:
/etc/resolv.conf -> /usr/local/bastille/jails/adguard/root/etc/resolv.conf
```

```
Template applied: default/base

No value provided for arg: GATEWAY6
[adguard]:
ifconfig_e0b_bastille0_name:  -> vnet0

[adguard]:
ifconfig_vnet0:  -> inet 192.168.2.55

[adguard]:
defaultrouter: NO -> 192.168.2.1
[adguard]: 0

[adguard]:
[adguard]: 0

Template applied: default/vnet

[adguard]:
adguard: removed
no IP address found for -

[adguard]:
e0a_bastille0
e0b_bastille0
adguard: created
```

You can see both ends of the virtual network cable I wrote about before: e0a_bastile0 and e0b_bastile0 form the connection between the host system and the jail. Check the ifconfig output on your host for a new bridge created from the jail's traffic.

The settings that are applied to the jail during its creation are fairly standard and mostly disable services we won't use anyway. After the jail was created, two more datasets exist on my pool that hold all the jail data:

```
# zfs list|grep adguard
rpi3/bastille/jails/adguard          2.36M  28.0G     26.5K  /usr/local/bastille/jails/adguard
rpi3/bastille/jails/adguard/root     2.34M  28.0G     2.34M  /usr/local/bastille/jails/adguard/
root
```

This forms the / filesystem for the jail and follows other jail manager layouts. To copy files into or out of the jail, simply use the prefix /usr/local/bastille/jails/adguard/root for the jail /-directory.

The jls command will list the bastille jail with it's settings:

```
# bastille list -a
 JID      State  IP Address    Published Ports      Hostname  Release           Path
 adguard  Up     192.168.2.55  -                    adguard   13.2-RELEASE-p1 /usr/local/bastille/
jails/adguard/root
```

At this point, the jail is alive and running. The only thing missing is the adguard home installation. Since we had bootstrapped that template earlier, we can apply it to the jail with this command:

```
#  bastille template adguard bastillebsd-templates/adguardhome
bastille template adguard bastillebsd-templates/adguardhome
[adguard]:
Applying template: bastillebsd-templates/adguardhome...
[adguard]:
Bootstrapping pkg from pkg+http://pkg.FreeBSD.org/FreeBSD:13:aarch64/quarterly, please wait...
Verifying signature with trusted certificate pkg.freebsd.org.2013102301... done
[adguard] Installing pkg-1.19.1_1...
[adguard] Extracting pkg-1.19.1_1: 100%
Updating FreeBSD repository catalogue...
[adguard] Fetching meta.conf: 100%    163 B   0.2kB/s    00:01
[adguard] Fetching packagesite.pkg: 100%    6 MiB   6.5MB/s    00:01
Processing entries: 100%
FreeBSD repository update completed. 31664 packages processed.
All repositories are up to date.
Updating database digests format: 100%
The following 2 package(s) will be affected (of 0 checked):

New packages to be INSTALLED:
        adguardhome: 0.107.22_5
        ca_root_nss: 3.89

Number of packages to be installed: 2

The process will require 41 MiB more space.
7 MiB to be downloaded.
[adguard] [1/2] Fetching adguardhome-0.107.22_5.pkg: 100%    6 MiB   6.7MB/s    00:01
[adguard] [2/2] Fetching ca_root_nss-3.89.pkg: 100%  266 KiB 272.1kB/s    00:01
Checking integrity... done (0 conflicting)
[adguard] [1/2] Installing ca_root_nss-3.89...
[adguard] [1/2] Extracting ca_root_nss-3.89: 100%
[adguard] [2/2] Installing adguardhome-0.107.22_5...
[adguard] [2/2] Extracting adguardhome-0.107.22_5: 100%
=====
Message from ca_root_nss-3.89:

-
FreeBSD does not, and can not warrant that the certification authorities
whose certificates are included in this package have in any way been
audited for trustworthiness or RFC 3647 compliance.

Assessment and verification of trust is the complete responsibility of the
system administrator.


This package installs symlinks to support root certificates discovery by
default for software that uses OpenSSL.

This enables SSL Certificate Verification by client software without manual
intervention.

If you prefer to do this manually, replace the following symlinks with
either an empty file or your site-local certificate bundle.
```

```
  * /etc/ssl/cert.pem
  * /usr/local/etc/ssl/cert.pem
  * /usr/local/openssl/cert.pem
=====
Message from adguardhome-0.107.22_5:

-
You installed AdGuardHome: Network-wide ads & trackers blocking DNS server.

In order to use it please start the service ‘adguardhome’ and
then access the URL http://0.0.0.0:3000/ in your favorite browser.

[adguard]:
/usr/local/bastille/templates/bastillebsd-templates/adguardhome/usr -> /usr/local/bastille/jails/
adguard/root/usr
/usr/local/bastille/templates/bastillebsd-templates/adguardhome/usr/local -> /usr/local/bastille/
jails/adguard/root/usr/local
/usr/local/bastille/templates/bastillebsd-templates/adguardhome/usr/local/bin -> /usr/local/bas-
tille/jails/adguard/root/usr/local/bin
/usr/local/bastille/templates/bastillebsd-templates/adguardhome/usr/local/bin/AdGuardHome.yaml ->
/usr/local/bastille/jails/adguard/root/usr/local/bin/AdGuardHome.yaml

[adguard]:
adguardhome_enable:  -> YES

[adguard]:
moving old config /usr/local/bin/AdGuardHome.yaml to the new location /usr/local/etc/AdGuardHome.
yaml
Starting adguardhome.

stdin:2: syntax error
pfctl: Syntax error in config file: pf rules not loaded
tcp 80 80
stdin:2: syntax error
pfctl: Syntax error in config file: pf rules not loaded
udp 53 53
Template applied: bastillebsd-templates/adguardhome
```

All it had to do was execute the instructions in the template (PKG, CP, etc.) in the jail. It is also a good test to see if networking is properly set up. If not, the jail won't be able to fetch the packages from the repository. The pfctl warnings at the end worried me a little, but it did work in spite of them.

Full of anticipation, I opened a browser as instructed by one of the messages on screen and pointed it to the jail IP address. And surely, a login screen for AdGuard Home presented itself. But where are the credentials? I checked back on the Bastille template website https://gitlab.com/bastillebsd-templates and found nothing that worked. The Bastille blog post mentioned AdGuard as the user, but the password did not work for it. So, I had to create my own, which is better security anyway, as default passwords are easily scanned for by bad actors.

I opened a console in the jail using this command:

```
# bastille console adguard
```

In the jail, I find that AdGuard put its configuration file under /usr/local/etc/AdGuard-Home.yaml. Near the top, I found this section:

```
users:
  - name: adguard
    password: some password not in clear text
```

Exiting again, I needed a way to create a BCrypt password. The htpasswd utility can do that, so I installed the apache24 web server which includes this:

```
# pkg install apache24
```

After running the "refresh" command, I could run the htpasswd utility. Checking its man page, I had to construct a command line that looked like this:

```
htpasswd -Bnb adguard BastilleBSD!
```

I provided the -B option to create a BCrypt password followed by a user this password should apply to (we have this already in the config file, but maybe you want another user or multiple ones), followed by the password in clear text. Yes, this is not a secure way, as this ends up in your shell history. But did I say anywhere in this tutorial that it is production ready? Exactly, I didn't. Dutifully, htpasswd spit the resulting password on the command line, which I copied and pasted into AdGuards config file.

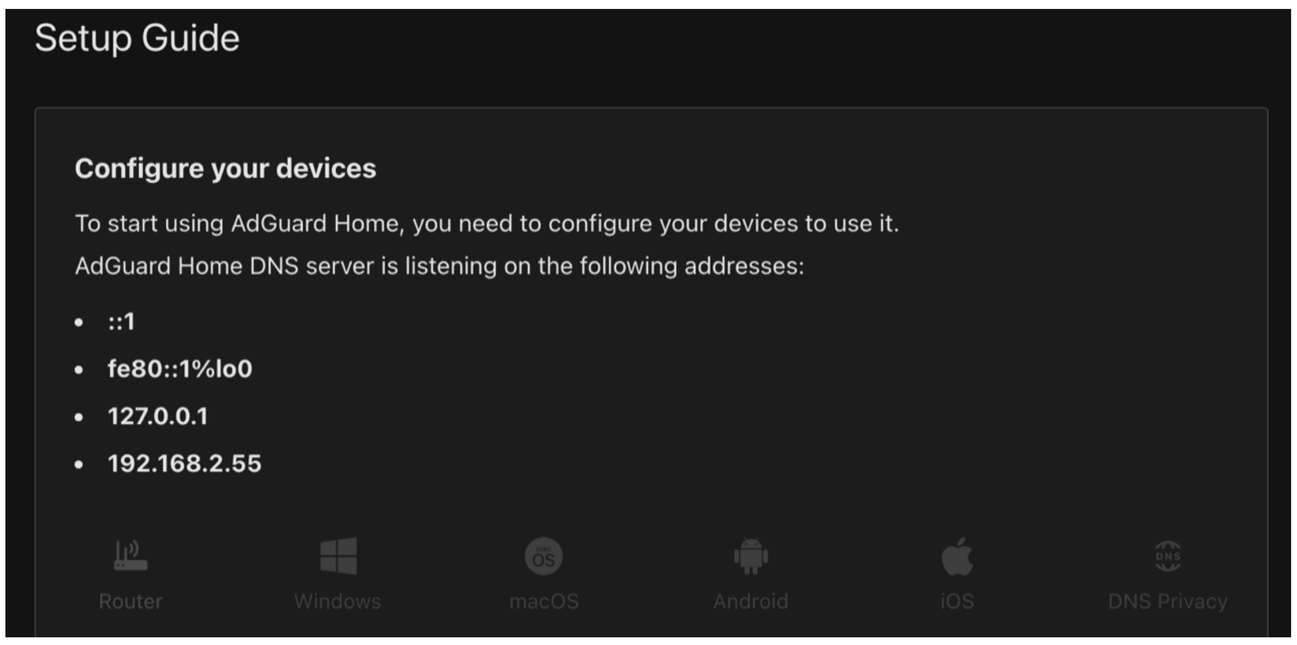
Then I ran

```
# service adguardhome restart
```

(still in my jail, mind you) to restart the service and apply the new settings. The other settings in the file are documented in the AdGuard Home Wiki: https://github.com/AdguardTeam/AdGuardHome/wiki/Configuration

Refreshing my web browser, I entered my new credentials and was redirected to the main AdGuard Dashboard. Success!

At the top, there is a Setup Guide that shows what needs to be done to use your new AdGuard service for either your router (to cover the whole network) and various devices, describing it for both mobile and desktop operating systems. Neat!



After I did that on my mobile phone—for testing purposes—and surfed the web a little, I saw statistics appearing in the Dashboard. That shows our setup is working and that we should rename the Internet to SnooperNet. Pretty much all sites track you in some way or display ads for your viewing unpleasure. The Raspberry Pi could handle the load and I

tweaked the number of connections in the ratelimit parameter of the AdGuardHome.yaml.

You can find the logs that AdGuard writes for the service in the jail's /var/log/adguard-home.log directory.

Wrap

That wraps up this tutorial. I found AdGuard to be well documented and easy to get started with, thanks to the work of the template creator. I'm already enjoying leaving fewer tracks on the web and seeing fewer ads online. The nice thing about it being a DNS service is that any device on your network can use it: PC, laptop, smartphone, tablets, TVs, IoT devices, and the neighbor's smart cat flap for all I know.

Bastille may have required a bit of configuration up front, but after that, it's a straightforward process to create jails. Maybe you'll find other services that you want to run on the Bastille template website: https://gitlab.com/bastillebsd-templates?

BENEDICT REUSCHLING is a documentation committer in the FreeBSD project and member of the documentation engineering team. In the past, he served on the FreeBSD core team for two terms. He administers a big data cluster at the University of Applied Sciences, Darmstadt, Germany. He's also teaching a course "Unix for Developers" for undergraduates. Benedict is one of the hosts of the weekly bsdnow.tv podcast.