Table of Contents

Set up Kotori behind a Caddy reverse proxy

Introduction

Caddy is a powerful, flexible web server that is able to automatically obtain and renew TLS certificates from Let's Encrypt; provide a secure, sensible TLS configuration by default; and provide a reverse proxy for other network services. It is noted for its simple configuration syntax. This page describes how to install Caddy on Ubuntu 18 and configure it as a reverse proxy for Kotori. These instructions should apply to any Linux version running systemd, as long as you can install Go v1.14 or newer on it.

Installation

Kotori is assumed to be installed and running.

Unfortunately, Caddy doesn't appear to be available to install as a .deb package in Ubuntu, so you'll need to build it from source. To do that, first install Go using `snap install go –classic`. Once that's installed, use to build `xcaddy`, which you'll use to build Caddy itself. Run `go get -u github.com/caddyserver/xcaddy/cmd/xcaddy` followed by `go build -o /usr/local/bin/xcaddy github.com/caddyserver/xcaddy/cmd/xcaddy`.

Finally, build Caddy itself. If you aren't adding any plugins (such as for DNS validation), run `xcaddy build –output /usr/bin/caddy`. If you're going to expose this system to the Internet, you won't need any plugins. If you're going to run this on a private network, you'll need to use DNS validation instead, and you'll need one of the plugins to do that. In that case, the build command would look like `xcaddy build –output /usr/bin/caddy –with github.com/caddy-dns/cloudflare` (or whichever other plugin you'd chosen).

The remainder of the installation process follows the Caddy documentation, and assumes you're the root user.

Create a user and group for Caddy: `groupadd –system caddy`, `useradd –system –gid caddy –create-home –home-dir /var/lib/caddy –shell /usr/sbin/nologin –comment “Caddy web server” caddy`.

Create `/etc/systemd/system/caddy.service` using your favorite text editor. Its contents should be:

```
# caddy.service
#
# For using Caddy with a config file.
#
# Make sure the ExecStart and ExecReload commands are correct
# for your installation.
#
```

```
# See https://caddyserver.com/docs/install for instructions.
#
# WARNING: This service does not use the --resume flag, so if you
# use the API to make changes, they will be overwritten by the
# Caddyfile next time the service is restarted. If you intend to
# use Caddy's API to configure it, add the --resume flag to the
# `caddy run` command or use the caddy-api.service file instead.

[Unit]
Description=Caddy
Documentation=https://caddyserver.com/docs/
After=network.target network-online.target
Requires=network-online.target

[Service]
User=caddy
Group=caddy
ExecStart=/usr/bin/caddy run --environ --config /etc/caddy/Caddyfile
ExecReload=/usr/bin/caddy reload --config /etc/caddy/Caddyfile
TimeoutStopSec=5s
LimitNOFILE=1048576
LimitNPROC=512
PrivateTmp=true
ProtectSystem=full
AmbientCapabilities=CAP_NET_BIND_SERVICE

[Install]
WantedBy=multi-user.target
```

Create a Caddyfile and set its ownership by running `mkdir /etc/caddy`, `touch /etc/caddy/Caddyfile`, and `chown -R caddy:caddy /etc/caddy/`.

Then activate this unit by running `systemctl daemon-reload` followed by `systemctl enable caddy`.

Configuration

Now that you have Caddy installed, you need to prepare a Caddyfile. This is Caddy's configuration file, the equivalent of Apache's httpd.conf or Nginx's nginx.conf–but much shorter, clearer, and simpler. Edit `/etc/caddy/Caddyfile` using your preferred text editor, and consult the relevant section below for its contents and discussion.

No certificate/TLS

If you just want to use Caddy to avoid needing to specify port 3000, the Caddyfile will be very simple:

```
*:80 {
        root * /usr/local/www/html
        file_server
        reverse_proxy localhost:3000
}
```

Save this, and you're ready to go.

Standalone Certificate

If you intend your Kotori installation to be open to the public Internet, start with this example:

```
{
        email admin@yourdomain.tld
    acme_ca https://acme-staging-v02.api.letsencrypt.org/directory
}

kotori.yourdomain.tld {
        root * /usr/local/www/html
        file_server
        reverse_proxy localhost:3000
}
```

The first block isn't strictly necessary, but it's a good idea to give Let's Encrypt your email address so you can be notified in the event your cert is about to expire–under normal circumstances you shouldn't get any traffic from this. The second directive tells Caddy to try to obtain your certificate from the staging server, rather than from Let's Encrypt's production servers. This is recommended in order to avoid running into the Let's Encrypt rate limits while testing things out. Once you're sure things are working properly, you can remove this line and run `systemctl reload caddy` to obtain a trusted certificate.

DNS Validation

I have Kotori running on my LAN, and don't intend to expose it to the public Internet, which means the Caddyfile above won't work. Instead, I tell Caddy to validate the certificate using DNS. I use Cloudflare for my DNS, and the instructions below will work without modification there. If you're using a different DNS host, you'll need to adjust things; consult the Caddy documentation for details. Your Caddyfile will look like this:

```
{
        email admin@yourdomain.tld
    acme_ca https://acme-staging-v02.api.letsencrypt.org/directory
}
```

```
kotori.yourdomain.tld {
        tls {
                dns cloudflare long_api_token
        }
        root * /usr/local/www/html
        file_server
        reverse_proxy localhost:3000
}
```

You'll notice this is identical to the one above, with the addition of the `tls` block. That block tells Caddy (1) to use DNS validation to obtain the certificate, (2) to use Cloudflare's DNS (which you'll need to have built into Caddy during the installation), and (3) what credential to use to authenticate to Cloudflare. Please note, for Cloudflare, this is an API **Token**, not an API **Key**. API tokens can be narrowly scoped, limiting the risk of their exposure. You'll need to create one through the Cloudflare interface. As above, this example uses the staging server to avoid exhausting the rate limits–you can remove that line once you're sure things are working properly.

Conclusion

Now that you've created the Caddyfile, start Caddy with `systemctl start caddy`. You should now be able to browse to the hostname you've chosen–Caddy will redirect to HTTPS, obtain and maintain the certificate, and have a secure TLS configuration.