Docs Menu

Configure popular ACME clients to use a private CA with the ACME protocol

The Automated Certificate Management Environment (ACME) protocol radically simplifies TLS deployment. With ACME, endpoints can obtain TLS certificates on their own, automatically. `step-ca` works with any ACME-compliant (specifically, ACMEv2; RFC8555) client.

About this tutorial

- Learn how to configure popular ACME clients to get certificates from `step-ca` .
- Examples include copy/paste code blocks and specific commands for nginx, certbot, and more.
- When complete, you will have a fully functioning ACME configuration using a private certificate authority.
- Estimated effort: Reading time ~7 mins, Lab time ~20 to 60 mins.

If you run into any issues, please let us know in GitHub Discussions or on our Discord server.

Requirements

- **Open source -** For `step-ca` , this tutorial assumes you have initialized and started up an instance using the steps in Getting Started. Additionally, you'll need to configure your CA with an ACME provisioner. Run `step ca provisioner add acme --type ACME` , and restart your CA.
- **Smallstep Certificate Manager -** this tutorial assumes you have created a hosted or linked authority and are running a local ACME Registration Authority.

Overview

Here are the most common configuration parameters for any ACME client:

DIRECTORY URL

Most ACME clients connect to Let's Encrypt's CA by default. To connect to a private CA, you need to point the client your ACME Directory URL.

A single instance of `step-ca` can have multiple ACME provisioners, each with their own ACME Directory URL. The URL follows this form:

```
https://{ca-host}/acme/{provisioner-name}/directory
```

For example, an ACME provisioner named `ACME` on the host `ca.internal` has the directory URL:

```
https://ca.internal/acme/ACME/directory
```

ACME CHALLENGE TYPE

You'll need to select the ACME challenge type.

CA CERTIFICATE

Communication between an ACME client and server uses HTTPS. Many clients will validate the server's TLS certificate using the public root certificates in your system's default trust store. Some clients will let you pass a CA certificate bundle into the client.

Clients will validate the server's HTTPS certificate using the public root certificates in your system's default trust store. When you're connecting to Let's Encrypt, it's a public certificate authority and its root certificate is already in your system's default trust store. Your internal root certificate isn't, so HTTPS connections from ACME clients to `step-ca` will fail.

There are two ways to address this challenge. Either:

- Explicitly configure your ACME client to trust `step-ca` 's root certificate, or
- Add the `step-ca` root certificate to your system's default trust store.

`step` provides a helper command to do the latter:

```
step certificate install
```

If you are using your certificate authority for TLS in production, explicitly configuring your ACME client to only trust your root certificate is a better option. You will see how this method works with an example below. You can find several other examples here.

If you are simulating Let's Encrypt in pre-production, installing your root certificate is a more realistic simulation of production. Once your root certificate is installed, no additional client configuration is necessary.

> ⚠ **A Word of Caution**
>
> Adding a root certificate to your system's default trust store is a global operation. Certificates issued by your CA will be trusted everywhere, including in many web browsers.

RENEWAL PERIOD

With most ACME clients, you can configure how often you want to renew your certificates. Choose a renewal period that is two-thirds of the entire certificate's lifetime, so that you'll have enough time to fix any renewal issues before it's too late.

Popular ACME Clients

- Certbot
- acme.sh
- lego
- win-acme
- Caddy v2
- NGINX
- Apache
- Node.js
- Golang
- Python

- Traefik
- Certify The Web

CERTBOT

`certbot` is the granddaddy of all ACME clients. Built and supported by the EFF, it's the standard-bearer for production-grade command-line ACME.

To get a certificate from `step-ca` using `certbot` you need to:

1. Point `certbot` at your ACME directory URL using the `--server` flag
2. Tell `certbot` to trust your root certificate using the `REQUESTS_CA_BUNDLE` environment variable

For example:

```
sudo REQUESTS_CA_BUNDLE=$(step path)/certs/root_ca.crt \
    certbot certonly -n --standalone -d foo.internal \
    --server https://ca.internal/acme/acme/directory
```

`sudo` is required in `certbot`'s *standalone* mode so it can listen on port 80 to complete the `http-01` challenge. If you already have a webserver running you can use *webroot* mode instead. With the appropriate plugin `certbot` also supports the `dns-01` challenge for most popular DNS providers. Deeper integrations with nginx and apache can even configure your server to use HTTPS automatically (we'll set this up ourselves later). All of this works with `step-ca`.

You can renew all of the certificates you've installed using `cerbot` by running:

```
sudo REQUESTS_CA_BUNDLE=$(step path)/certs/root_ca.crt certbot renew
```

You can automate renewal with a simple `cron` entry:

```
*/15 * * * * root REQUESTS_CA_BUNDLE=$(step path)/certs/root_ca.crt ce
```

The `certbot` packages for some Linux distributions will create a `cron` entry or systemd timer like this for you. This entry won't work with `step-ca` because it doesn't set the `REQUESTS_CA_BUNDLE` environment variable. You'll need to manually tweak it to do so.

More subtly, `certbot` 's default renewal job is tuned for Let's Encrypt's 90 day certificate lifetimes: it's run every 12 hours, with actual renewals occurring for certificates within 30 days of expiry. By default, `step-ca` issues certificates with *much shorter* 24 hour lifetimes. The `cron` entry above accounts for this by running `certbot renew` every 15 minutes. You'll also want to configure your domain to only renew certificates when they're within a few hours of expiry by adding a line like:

```
renew_before_expiry = 8 hours
```

to the top of your renewal configuration (e.g., in `/etc/letsencrypt/renewal/foo.internal.conf`).

ACME.SH

acme.sh is another popular command-line ACME client. It's written completely in shell (`bash` , `dash` , and `sh` compatible) with very few dependencies.

To get a certificate from `step-ca` using `acme.sh` you need to:

1. Point `acme.sh` at your ACME directory URL using the `--server` flag
2. Tell `acme.sh` to trust your root certificate using the `--ca-bundle` flag

For example:

```
sudo acme.sh --issue --standalone -d foo.internal \
    --server https://ca.internal/acme/acme/directory \
    --ca-bundle $(step path)/certs/root_ca.crt \
    --fullchain-file foo.crt \
    --key-file foo.key
```

Like `certbot` , `acme.sh` can solve the `http-01` challenge in *standalone* mode and *webroot* mode. It can also solve the `dns-01` challenge for many DNS providers.

Renewals are slightly easier since `acme.sh` remembers to use the right root certificate. It can also remember how long you'd like to wait before renewing a certificate. Unfortunately, the duration is specified in days (via the `--days` flag) which is too coarse for `step-ca` 's default 24 hour certificate lifetimes. So the easiest way to schedule renewals with `acme.sh` is to force them at a reasonable frequency, like every 8 hours, via cron:

```
0 */8 * * * root "/home/<user>/.acme.sh"/acme.sh --cron --home "/home/<
```

LEGO

lego is another popular command-line ACME client. It's written completely in Go and works on all platforms (Windows, Linux, Mac).

To get a certificate from `step-ca` using `lego` you need to:

1. Point `lego` at your ACME directory URL using the `--server` flag
2. Tell `lego` to trust your root certificate using the `LEGO_CA_CERTIFICATES` environment variable

For example:

```
sudo LEGO_CA_CERTIFICATES="$(step path)/certs/root_ca.crt" \
  lego --email="you@example.com" -d foo.internal \
  -s https://ca.internal/acme/acme/directory --http run
```

Like `certbot`, `lego` can solve the `http-01` challenge in *standalone* mode and *webroot* mode. It can also solve the `dns-01` challenge for many DNS providers.

You can renew the certificates you've installed using `lego` by running:

```
sudo LEGO_CA_CERTIFICATES="$(step path)/certs/root_ca.crt" lego --emai
```

You can automate renewal with a simple `cron` entry:

```
*/15 * * * * root LEGO_CA_CERTIFICATES="$(step path)/certs/root_ca.crt'
```

WIN-ACME

win-acme (`wacs.exe`) is a popular ACME client for Windows.

To use `win-acme` with `step-ca`, you'll need to do the following:

- Add your root CA certificate (`root_ca.crt`) to the Windows trust store.

- Change the ACMEv2 endpoint used by `win-acme` (in the `settings.json` file that comes with the program) to point to your CA's ACME provisioner (eg. `https://ca.internal/acme/acme/directory`). Or pass the `--baseuri` flag with your ACME provisioner's endpoint.
- We recommend using the `tls-alpn-01` challenge type to prove ownership.

CADDY V2

Caddy is an HTTP/2 web server with *automatic HTTPS* powered by an integrated ACME client. In addition to serving static websites, Caddy is commonly used as a TLS-terminating API gateway proxy.

Caddy comes with its own ACME server and by default it will generate an internal CA and issue certificates to itself. But, you can configure Caddy to use a local `step-ca` instance to obtain certificates.

Here's a `Caddyfile` global config block. Add this to the top of your `Caddyfile` to get certificates from `ca.internal` for all configured domains:

```
{
  email carl@smallstep.com
  acme_ca https://ca.internal/acme/acme/directory
  acme_ca_root <step path>/root_ca.crt
}
```

Here's a `Caddyfile` that will use `ca.internal` only to get a certificate for `foo.internal`:

```
foo.internal

root * /var/www
tls carl@smallstep.com {
  ca https://ca.internal/acme/acme/directory
  ca_root <step path>/certs/root_ca.crt
}
```

Replace `<step path>` with the output of the `step path` command.

Now run `caddy` to start serving HTTPS!

```
$ sudo caddy start
```

Check your work with curl:

```
$ curl https://foo.internal --cacert $(step path)/certs/root_ca.crt

Hello, TLS!
```

Caddy will automatically renew its certificates after ⅔ of the validity period elapses.

NGINX

[NGINX][https://nginx.com/] doesn't support ACME natively, but there are two options:

- The njs-acme module allows for automatic generation and renewal of TLS certificates for NGINX using ACME.
- You can use a command-line ACME client to get certificates for NGINX.

Using the njs-acme module

See njs-acme for full documentation.

Using a command-line ACME client

Here's an example `nginx.conf` that runs NGINX in a common configuration where it terminates TLS and proxies to a back-end server listening on local loopback:

```
server {
  listen 443 ssl;
  server_name foo.internal;
  ssl_certificate /path/to/foo.crt;
  ssl_certificate_key /path/to/foo.key;
  location / {
    proxy_pass http://127.0.0.1:8000
  }
}
```

With this code, you are telling NGINX to listen on port 443 using TLS, with a certificate and private key stored on disk. Other resources provide a more thorough explanation of NGINX's various TLS configuration options.

We can start an HTTP server using python and check our work with curl:

```
$ echo "Hello TLS!" > index.html

$ python -m SimpleHTTPServer 8000 &

$ curl https://foo.internal --cacert $(step path)/certs/root_ca.crt

Hello TLS!
```

NGINX only reads certificates once, only at startup. When you renew the certificate on disk, NGINX won't notice. After each renewal you'll need to run the following command:

```
nginx -s reload
```

You can use the `--exec` flag on the `step ca renew` command to do this automatically:

```
step ca renew --daemon --exec "nginx -s reload" \
    /path/to/foo.crt \
    /path/to/foo.key
```

If you're using certbot, check out the `--post-hook` flag to do the same thing. If you're using acme.sh, check out the `--reloadcmd` flag.

APACHE

Apache httpd has integrated ACME support, via mod_md. Or you can deploy certificates to Apache using an external ACME client, such as certbot.

Here's an example Apache configuration, using certificates issued by `step-ca` through certbot:

```
<VirtualHost *:443>
  ServerName foo.internal
  DocumentRoot /home/mmalone/www
  SSLEngine on
  SSLCertificateFile /etc/letsencrypt/live/foo.internal/fullchain.pem
  SSLCertificateKeyFile /etc/letsencrypt/live/foo.internal/privkey.pem
</VirtualHost>
```

Start Apache and check your work with curl:

```
$ curl --cacert $(step path)/certs/root_ca.crt https://foo.internal

Hello TLS
```

Apache needs to be signaled after certificates are renewed by running the following command:

```
apachectl graceful
```

NODE

Publish Lab's acme-client is an excellent ACMEv2 client written in Node.js. Take a look at an example of how easy it is to obtain a certificate and serve HTTPS in JavaScript: https://gist.github.com/mmalone/f3c33a2381ffa3d67e86c6d5ad3042c9

Most importantly, to make things work:

- Point the ACME client at your ACME directory URL
- Tell the ACME client to trust your CA by configuring the HTTP client to verify certificates using your root certificate

To install dependencies and start the server run:

```
npm install node-acme-client
node acme.js
```

Then check your work with curl:

```
$ curl https://foo.internal:11443 \
    --cacert $(step path)/certs/root_ca.crt

Hello, TLS
```

This server supports optional client authentication using certificates and checks if the client authenticated in the handler:

```
$ curl https://foo.internal:11443 \
    --cacert $(step path)/certs/root_ca.crt \
    --cert mike.crt \
```

```
    --key mike.key

Hello, mike@smallstep.com
```

GOLANG

lego is an ACME client CLI and library written in Go. You can use it to obtain a certificate from `step-ca` programmatically. You can find an example of this code here: https://gist.github.com/ldez/e975a1026b704e55f1d1f85143b377b7

Essentially, the steps involved are:

- Point lego at your ACME directory URL by setting `lego.Config.CADirUrl`
- Tell lego to trust your CA by configuring an `http.Client` that trusts your root certificate and telling lego to use it

Fetch the required dependencies and start the server:

```
$ go run acme.go
```

Then test with curl:

```
$ curl https://foo.internal:5443 \
    --cacert $(step path)/certs/root_ca.crt

Hello, TLS!
```

The server is configured to verify client certificates if they are sent. That means the server is configured to support mutual TLS. The handler checks whether a client certificate was provided, and responds with a personalized greeting if one was.

You can get a client certificate from `step-ca` using an OAuth/OIDC provisioner:

```
$ step ca certificate mike@example.com mike.crt mike.key

✔ Provisioner: Google (OIDC) [client: <redacted>.apps.googleusercontent.com]
✔ CA: https://ca.internal
✔ Certificate: mike.crt
✔ Private Key: mike.key
```

And test mutual TLS out with curl:

```
$ curl https://foo.internal:5443 \
  --cacert $(step path)/certs/root_ca.crt \
  --cert mike.crt \
  --key mike.key

Hello, mike@example.com!
```

With a few tweaks to this code you can implement robust access control.

There are other good options for programmatic ACME in Go. The certmagic package builds on lego and offers higher level, easier to use abstractions. The x/crypto/acme package is lower level and offers more control, but it currently implements a pre-standardization draft version of ACME that doesn't work with `step-ca`.

PYTHON

certbot is written in Python and exposes its acme module as a standalone package. You can find an example of obtaining a certificate and serving HTTPS in Python here: https://gist.github.com/mmalone/12f5422b2ec68e64e9d11eae0c6ca47d

Make sure that you:

- Point the ACME client at your ACME Directory URL
- Tell the ACME client to trust your CA by configuring the injected HTTP client to verify certificates using your root certificate

To install dependencies and start the server, run:

```
pip install acme
pip install pem
python https.py
```

Then check your work with curl:

```
$ curl https://foo.internal:10443 \
    --cacert $(step path)/certs/root_ca.crt

Hello, TLS!
```

Like the Go example above, this server also supports mutual TLS and checks if the client authenticated in the handler:

```
$ curl https://foo.internal:10443 \
  --cacert $(step path)/certs/root_ca.crt \
  --cert mike.crt \
  --key mike.key

Hello, mike@smallstep.com!
```

TRAEFIK

Traefik is a modern reverse-proxy with integrated support for ACME. It's designed primarily to handle ingress for a compute cluster, dynamically routing traffic to microservices and web applications.

Traefik v2

It's easy to get a certificate from `step-ca` in Traefik v2, using the `tls-alpn-01` ACME challenge type.

Most importantly, Traefik will need to trust your root CA certificate. Either use the `LEGO_CA_CERTIFICATES` environment variable to provide the full path to your `root_ca.crt` when running `traefik`, or install your root certificate in your system's default trust store by running `step certificate install root_ca.crt`.

In your Traefik static configuration, you'll need to add a `certificatesResolvers` block:

```
[certificatesResolvers]
  [certificatesResolvers.myresolver]
    [certificatesResolvers.myresolver.acme]
      caServer = "https://step-ca.internal/acme/acme/directory"
      email = "carl@smallstep.com"
      storage = "acme.json"
      certificatesDuration = 24
      tlsChallenge = true
```

Then, when you add routers to your dynamic configuration for HTTPS traffic, you need to set `tls` and `tls.certresolver`:

```
[http]
  [http.routers]
    [http.routers.router1]
```

```
...
[http.routers.router1.tls]
  certResolver = "myresolver"
```

If you're running Traefik inside a Docker container, you can get your root CA certificate and add it to the container's trust store by running the following:

```
$ step ca bootstrap --ca-url "${CA_URL}" --fingerprint "${CA_FINGERPRI
$ update-ca-certificates
```

Traefik v1

To get a certificate from `step-ca` to Traefik v1 you need to:

- Point Traefik at your ACME directory URL using the `caServer` directive in your configuration file
- Tell Traefik to trust your root certificate using the LEGO_CA_CERTIFICATES environment variable

Here's an example `traefik.toml` file that configures Traefik to terminate TLS and proxy to a service listening on *localhost*:

```
defaultEntryPoints = ["http", "https"]
[entryPoints]
  [entryPoints.http]
  address = ":80"
  [entryPoints.https]
  address = ":443"
    [entryPoints.https.tls]
[acme]
storage = "acme.json"
caServer = "https://ca.internal/acme/acme/directory"
entryPoint = "https"
[acme.httpChallenge]
entryPoint = "http"
[[acme.domains]]
main = "foo.internal"
[file]
[frontends]
  [frontends.foo]
  backend = "foo"
[backends]
  [backends.foo]
```

```
[backends.foo.servers.server0]
  url = "http://127.0.0.1:8000"
```

Start Traefik by running:

```
LEGO_CA_CERTIFICATES=$(step path)/certs/root_ca.crt traefik
```

Start an HTTP server for Traefik to proxy to, and test with curl:

```
$ echo "Hello TLS!" > index.html
$ python -m SimpleHTTPServer 8000 &
$ curl https://foo.internal --cacert $(step path)/certs/root_ca.crt

Hello TLS!
```

CERTIFY THE WEB

Certify The Web is a popular ACME Certificate Manager for Windows. It provides a full UI for managing thousands of certificates, supports a wide range of built in deployment tasks and integrates with many DNS API providers. Commercial licensing and support is also available.

To use with `step-ca`:

- Add your CA root certificate to *Local Machine* > *Trusted Certificate Authorities* and your CA intermediate to *Local Machine* > *Intermediate Certification Authorities*. This will make your endpoint certificate (and the other ACME certificates you issue from your CA) trusted on this machine.
- Add your `step-ca` instance details as a new Certificate Authority under *Settings* > *Certificate Authorities*. You can set the Production and Staging API urls either to the same directory endpoint or point them to different instances if you are operating a split staging and production configuration.
- Add a CA account for your new CA under *Settings* > *Certificate Authorities* > *New Account*, selecting your new CA from the list.
- Select *New Certificate* to begin ordering a new certificate from your CA. Make sure to set your CA preference under Certificate > Advanced > Certificate Authority (or you can set this as a global setting). You can use HTTP or DNS validation. Select *Request Certificate* to perform your certificate order. Subsequent renewals are automatic.

- By default the certificate will be added to the local machine certificate store. It can also be automatically deployed to IIS sites on the same machine, or you can use Deployment Tasks to push certificates to secrets vaults or to remote machines via SFTP (windows or linux etc) or to UNC shares etc.

Subscribe to updates

Unsubscribe anytime, see Privacy Policy

Your email

Learn

Blog

Try for free

Register for demo

Products

Certificate Manager

Smallstep SSH

ACME Registration Authority

Integrations

Documentation

Certificate Manager

Smallstep SSH

step-ca

Tutorials

Step command reference

Open Source

step-ca

Step CLI

About

About

Support

Status

Careers

Privacy

Terms of use

Privacy Policy

Privacy Center

Security

© 2025 Smallstep Labs, Inc. All rights reserved