Protect the Docker daemon socket

By Docker Inc

Published: 2024-09-10 · Archived: 2026-04-05 16:03:01 UTC

By default, Docker runs through a non-networked UNIX socket. It can also optionally communicate using SSH or a TLS (HTTPS) socket.

> The given `USERNAME` must have permissions to access the docker socket on the remote machine. Refer to manage Docker as a non-root user to learn how to give a non-root user access to the docker socket.

The following example creates a `docker context` to connect with a remote `dockerd` daemon on `host1.example.com` using SSH, and as the `docker-user` user on the remote machine:

After creating the context, use `docker context use` to switch the `docker` CLI to use it, and to connect to the remote engine:

Use the `default` context to switch back to the default (local) daemon:

Alternatively, use the `DOCKER_HOST` environment variable to temporarily switch the `docker` CLI to connect to the remote host using SSH. This does not require creating a context, and can be useful to create an ad-hoc connection with a different engine:

SSH Tips

For the best user experience with SSH, configure `~/.ssh/config` as follows to allow reusing a SSH connection for multiple invocations of the `docker` CLI:

If you need Docker to be reachable through HTTP rather than SSH in a safe manner, you can enable TLS (HTTPS) by specifying the `tlsverify` flag and pointing Docker's `tlscacert` flag to a trusted CA certificate.

In the daemon mode, it only allows connections from clients authenticated by a certificate signed by that CA. In the client mode, it only connects to servers with a certificate signed by that CA.

> Using TLS and managing a CA is an advanced topic. Familiarize yourself with OpenSSL, x509, and TLS before using it in production.

Create a CA, server and client keys with OpenSSL

> Replace all instances of `$HOST` in the following example with the DNS name of your Docker daemon's host.

First, on the Docker daemon's host machine, generate CA private and public keys:

Now that you have a CA, you can create a server key and certificate signing request (CSR). Make sure that "Common Name" matches the hostname you use to connect to Docker:

> Replace all instances of `$HOST` in the following example with the DNS name of your Docker daemon's host.

Next, we're going to sign the public key with our CA:

Since TLS connections can be made through IP address as well as DNS name, the IP addresses need to be specified when creating the certificate. For example, to allow connections using `10.10.10.20` and `127.0.0.1`:

Set the Docker daemon key's extended usage attributes to be used only for server authentication:

Now, generate the signed certificate:

Authorization plugins offer more fine-grained control to supplement authentication from mutual TLS. In addition to other information described in the above document, authorization plugins running on a Docker daemon receive the certificate information for connecting Docker clients.

For client authentication, create a client key and certificate signing request:

> For simplicity of the next couple of steps, you may perform this step on the Docker daemon's host machine as well.

To make the key suitable for client authentication, create a new extensions config file:

Now, generate the signed certificate:

After generating `cert.pem` and `server-cert.pem` you can safely remove the two certificate signing requests and extensions config files:

With a default `umask` of 022, your secret keys are *world-readable* and writable for you and your group.

To protect your keys from accidental damage, remove their write permissions. To make them only readable by you, change file modes as follows:

Certificates can be world-readable, but you might want to remove write access to prevent accidental damage:

Now you can make the Docker daemon only accept connections from clients providing a certificate trusted by your CA:

To connect to Docker and validate its certificate, provide your client keys, certificates and trusted CA:

> This step should be run on your Docker client machine. As such, you need to copy your CA certificate, your server certificate, and your client certificate to that machine.

> Replace all instances of `$HOST` in the following example with the DNS name of your Docker daemon's host.

> Docker over TLS should run on TCP port 2376.

> As shown in the example above, you don't need to run the `docker` client with `sudo` or the `docker` group when you use certificate authentication. That means anyone with the keys can give any instructions to your Docker daemon, giving them root access to the machine hosting the daemon. Guard these keys as you would a root password!

Secure by default

If you want to secure your Docker client connections by default, you can move the files to the `.docker` directory in your home directory --- and set the `DOCKER_HOST` and `DOCKER_TLS_VERIFY` variables as well (instead of passing `-H=tcp://$HOST:2376` and `--tlsverify` on every call).

Docker now connects securely by default:

```
$ docker ps
```

Other modes

If you don't want to have complete two-way authentication, you can run Docker in various other modes by mixing the flags.

Daemon modes

- `tlsverify`, `tlscacert`, `tlscert`, `tlskey` set: Authenticate clients
- `tls`, `tlscert`, `tlskey`: Do not authenticate clients

Client modes

- `tls`: Authenticate server based on public/default CA pool
- `tlsverify`, `tlscacert`: Authenticate server based on given CA
- `tls`, `tlscert`, `tlskey`: Authenticate with client certificate, do not authenticate server based on given CA
- `tlsverify`, `tlscacert`, `tlscert`, `tlskey`: Authenticate with client certificate and authenticate server based on given CA

If found, the client sends its client certificate, so you just need to drop your keys into `~/.docker/{ca,cert,key}.pem`. Alternatively, if you want to store your keys in another location, you can specify that location using the environment variable `DOCKER_CERT_PATH`.

Connecting to the secure Docker port using `curl`

To use `curl` to make test API requests, you need to use three extra command line flags:

- Using certificates for repository client verification

- Use trusted images

Source: https://docs.docker.com/engine/security/protect-access/