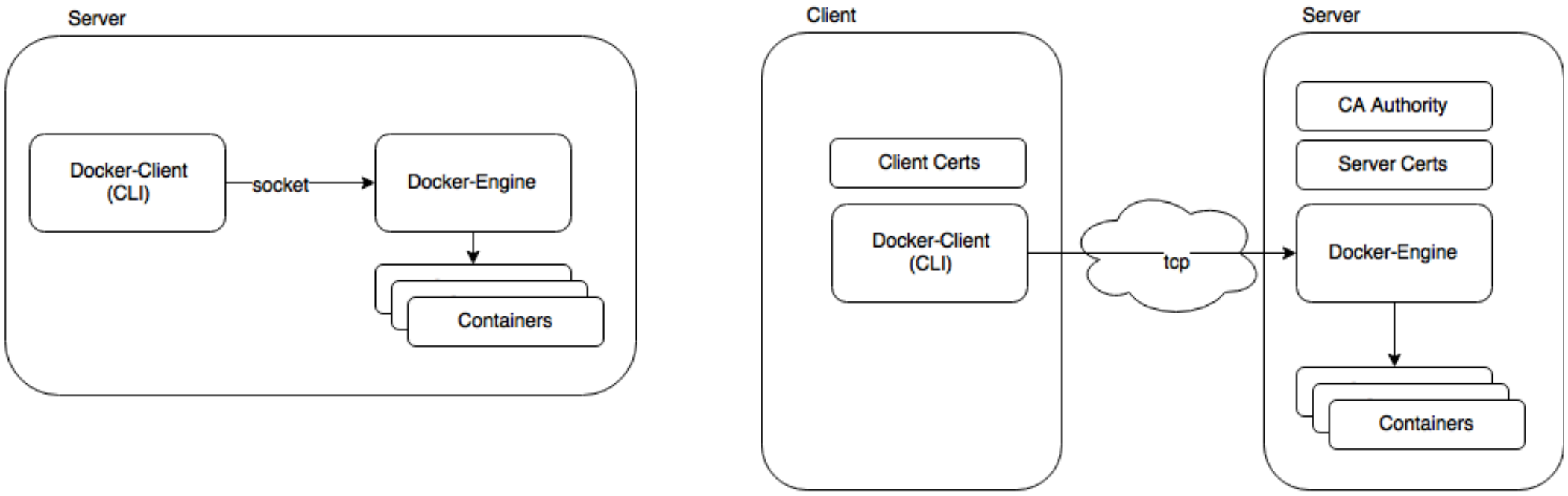
Securing Docker engine with TLS.

30 Mar 2016

Introduction

By default, when you install and start Docker on Linux, Docker-Client (cli) access Docker-Engine on local host with unix `socket`. Docker-Engine allows any process to access api from local host. Yes, that's right Docker-Engine is nothing more than small web server exposing REST API with root access to Linux kernel functionality (cgroups, kernel namespaces, etc.). In this post I will show how to access Docker-Engine over network (tcp) and how to secure this communication using TLS (client / server certificates).



What is TLS ?

> The TLS protocol allows client-server applications to communicate across a network in a way designed to prevent eavesdropping and tampering (3).

Steps :

- Generate CA (authority) certificates.
- Generate Server side certs.
- Generate Client side certs.
- Configure Docker-Engine to listen on tcp and respect TLS.
- Test Docker-Client using newly generated certs.

Generate CA (authority) certificates.

We need CA certs to sign client and server side certs. On Docker-Engine server execute :

```
openssl genrsa -out ~/.docker/ca-key.pem 2048
openssl req -x509 -new -nodes -days 10000 \
  -subj '/CN=docker-CA' \
  -key ~/.docker/ca-key.pem \
  -out ~/.docker/ca.pem
```

This will produce : `ca-key.pem` and `ca.pem` files. We will store certs in user home `~/.docker/` please ensure this folder exists `make -p ~/.docker/`. Also ensure `openssl` package is installed on your server : `yum install openssl`.

Generate Server side certs.

Server side certs will be used by Docker-engine daemon. Lets start by generating key cert.:

```
openssl genrsa -out ~/.docker/key.pem 2048
```

Next we need to generate request to sign cert. Because its important to know certificate subject (server DNS, IP, etc.) we will use `subjectAltName` which will give us flexibility to set multiple subject names. Lets create `openssl` config file :

`nano ~/.docker/openssl.cnf`

```
[req]
req_extensions = v3_req
distinguished_name = req_distinguished_name
[req_distinguished_name]
[ v3_req ]
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
extendedKeyUsage = serverAuth, clientAuth
subjectAltName = @alt_names

[alt_names]
IP.1 = 123.123.123.123
IP.2 = 0.0.0.0
```

Its important to replace `IP.1` with your real server IP where Docker-Engine will run. With second line (`IP.2`) we allow to use certs in connections from local host (for debug purposes).

Now lets generate request itself :

```
openssl req -new \
  -subj '/CN=docker-server' \
  -key ~/.docker/key.pem \
  -config ~/.docker/openssl.cnf \
  -out ~/.docker/cert.csr
```

And sign server cert with CA cert + CA key and sign request :

```
openssl x509 -req \
  -in ~/.docker/cert.csr \
  -CA ~/.docker/ca.pem \
  -CAkey ~/.docker/ca-key.pem \
  -extfile ~/.docker/openssl.cnf \
  -out ~/.docker/cert.pem \
  -days 365 -extensions v3_req -CAcreateserial
```

In the end we are interested to have 3 cert files on Docker-Engine server :

1. `cert.pem` - Server certificate
2. `key.pem` - Server key file
3. `ca.pem` - CA certificate.

Generate Client side certs.

Before generating certs on Docker-client side please ensure you copy CA certs and CA key file (`~/.docker/ca.pem` , `~/.docker/ca-key.pem`) to client instance . Also ensure `openssl` is installed.

Same way as with server first generate key file :

```
openssl genrsa -out ~/.docker/key.pem 2048
```

Next create openssl config file :

`nano ~/.docker/openssl.cnf`

```
[req]
req_extensions = v3_req
distinguished_name = req_distinguished_name
[req distinguished name]
[ v3_req ]
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
extendedKeyUsage = serverAuth, clientAuth
```

And create certificate sign file

```
openssl req -new -subj '/CN=docker-client' \
  -key ~/.docker/key.pem \
  -config ~/.docker/openssl.cnf \
  -out ~/.docker/cert.csr
```

Last step is to sign client certificate :

```
openssl x509 -req -CAcreateserial \
  -days 365 -extensions v3_req \
  -CA ~/.docker/ca.pem \
  -CAkey ~/.docker/ca-key.pem \
  -in ~/.docker/cert.csr \
  -extfile ~/.docker/openssl.cnf \
  -out ~/.docker/cert.pem
```

In the end on client machine we have following files :

1. `cert.pem` - Client certificate.
2. `key.pem` - Client key file.
3. `ca.pem` - CA certificate.

Configure Docker-Engine to listen on tcp and respect TLS.

Now when certs are ready lets switch back to instance with Docker-Engine. We need to instruct Docker-Engine daemon to listen on `tcp` port `2376` and verify TLS cert. Docker daemon configuration depends on docker version, for example in docker `1.9.0` we can set config in `/etc/sysconfig/docker` file :

`sudo nano /etc/sysconfig/docker`

```
OPTIONS=" --default-ulimit nofile=1024:4096
-H unix:///var/run/docker.sock -H 0.0.0.0:2376 --tlsverify \
--tlscacert=~/.docker/ca.pem --tlscert=~/.docker/cert.pem \
--tlskey=~/.docker/key.pem"
```

Please notice sentence `-H unix:///var/run/docker.sock` , by this we will allow to make socket connections from local host. Last thing, restart daemon :

```
/etc/init.d/docker restart
```

And check docker logs for any error message:

```
less /var/log/docker
```

Test Docker-Client using newly generated certs.

Last piece, we need to test can we reach Docker-Engine from Docker-Client. Lets switch back to client instance and instruct Docker-client (CLI) to use remote Docker-Engine:

```
docker --tlsverify --tlskey=key.pem --tlscacert=ca.pem \
  --tlscert=cert.pem -H=123.123.123.123:2376 version
```

Lets ask Docker-Engine to return server version :

```
client$ docker --tlsverify --tlskey=key.pem \
  --tlscacert=ca.pem --tlscert=cert.pem \
  -H=123.123.123.123:2376 version
Client:
 Version:      1.9.0
 API version:  1.21
 Go version:   go1.4.3
 Git commit:   76d6bc9
 Built:        Tue Nov  3 19:20:09 UTC 2015
 OS/Arch:      darwin/amd64

Server:
 Version:      1.9.1
 API version:  1.21
 Go version:   go1.4.2
 Git commit:   a34a1d5/1.9.1
 Built:
 OS/Arch:      linux/amd64
```

As we can see client and server versions are different. We can simplify process of typing same commands over and over by setting env. variables.

```
export DOCKER_CERT_PATH=~/.docker/
export DOCKER_HOST=tcp://123.123.123.123:2376
export DOCKER_TLS_VERIFY=1

client$ dockerversion
```