(/os/docs/latest)

Generate self-signed certificates

If you build Container Linux cluster on top of public networks it is recommended to enable encryption for Container Linux services to prevent traffic interception and man-in-the-middle attacks. For these purposes you have to use Certificate Authority (CA), private keys and certificates signed by CA. Let's use cfssl (https://github.com/cloudflare/cfssl) and walk through the whole process to create all these components.

NOTE: We will use basic procedure here. If your configuration requires advanced security options, please refer to official cfssl (https://github.com/cloudflare/cfssl) documentation.

Download cfssl

CloudFlare's distributes cfssl (https://github.com/cloudflare/cfssl) source code on github page and binaries on cfssl website (https://pkg.cfssl.org/).

Our documentation assumes that you will run cfssl (https://github.com/cloudflare/cfssl) on your local x86_64 Linux host.

```
mkdir ~/bin
curl -s -L -o ~/bin/cfssl https://pkg.cfssl.org/R1.2/cfssl_linux-amd64
curl -s -L -o ~/bin/cfssljson https://pkg.cfssl.org/R1.2/cfssljson_linux-amd64
chmod +x ~/bin/{cfssl,cfssljson}
export PATH=$PATH:~/bin
```

Initialize a certificate authority

First of all we have to save default `cfssl` options for future substitutions:

```
mkdir ~/cfssl
cd ~/cfssl
cfssl print-defaults config > ca-config.json
cfssl print-defaults csr > ca-csr.json
```

Certificate types which are used inside Container Linux

- **client certificate** is used to authenticate client by server. For example `etcdctl` , `etcd proxy` , `fleetctl` or `docker` clients.
- **server certificate** is used by server and verified by client for server identity. For example `docker` server or `kube-apiserver` .
- **peer certificate** is used by etcd cluster members as they communicate with each other in both ways.

Configure CA options

Now we can configure signing options inside `ca-config.json` config file. Default options contain following preconfigured fields:

- profiles: **www** with `server auth` (TLS Web Server Authentication) X509 V3 extension and **client** with `client auth` (TLS Web Client Authentication) X509 V3 extension.
- expiry: with `8760h` default value (or 365 days)

For compliance let's rename **www** profile into **server**, create additional **peer** profile with both `server auth` and `client auth` extensions, and set expiry to 43800h (5 years):

```
{
    "signing": {
        "default": {
            "expiry": "43800h"
        },
        "profiles": {
            "server": {
                "expiry": "43800h",
                "usages": [
                    "signing",
                    "key encipherment",
                    "server auth"
                ]
            },
            "client": {
                "expiry": "43800h",
                "usages": [
                    "signing",
                    "key encipherment",
                    "client auth"
                ]
            },
            "peer": {
                "expiry": "43800h",
                "usages": [
                    "signing",
                    "key encipherment",
                    "server auth",
                    "client auth"
                ]
            }
        }
    }
}
```

You can also modify `ca-csr.json` Certificate Signing Request (CSR):

```
{
    "CN": "My own CA",
    "key": {
        "algo": "rsa",
        "size": 2048
    },
    "names": [
        {
            "C": "US",
            "L": "CA",
            "O": "My Company Name",
            "ST": "San Francisco",
            "OU": "Org Unit 1",
            "OU": "Org Unit 2"
        }
    ]
}
```

And generate CA with defined options:

```
cfssl gencert -initca ca-csr.json | cfssljson -bare ca -
```

You'll get following files:

```
ca-key.pem
ca.csr
ca.pem
```

- Please keep `ca-key.pem` file in safe. This key allows to create any kind of certificates within your CA.
- ***.csr** files are not used in our example.

Generate server certificate

```
cfssl print-defaults csr > server.json
```

Most important values for server certificate are **Common Name (CN)** and **hosts**. We have to substitute them, for example:

```
...
    "CN": "coreos1",
    "hosts": [
        "192.168.122.68",
        "ext.example.com",
        "coreos1.local",
        "coreos1"
    ],
...
```

Now we are ready to generate server certificate and private key:

```
cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -profile=server server.json | cfssl
json -bare server
```

Or without CSR json file:

```
echo '{"CN":"coreos1","hosts":[""],"key":{"algo":"rsa","size":2048}}' | cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -profile=server -hostname="192.168.122.68,ext.example.com,coreos1.local,coreos1" - | cfssljson -bare server
```

You'll get following files:

```
server-key.pem
server.csr
server.pem
```

Generate peer certificate

```
cfssl print-defaults csr > member1.json
```

Substitute CN and hosts values, for example:

```
...
    "CN": "member1",
    "hosts": [
        "192.168.122.101",
        "ext.example.com",
        "member1.local",
        "member1"
    ],
...
```

Now we are ready to generate member1 certificate and private key:

```
cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -profile=peer member1.json | cfssljson -bare member1
```

Or without CSR json file:

```
echo '{"CN":"member1","hosts":[""],"key":{"algo":"rsa","size":2048}}' | cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -profile=peer -hostname="192.168.122.101,ext.example.com,member1.local,member1" - | cfssljson -bare member1
```

You'll get following files:

```
member1-key.pem
member1.csr
member1.pem
```

Repeat these steps for each `etcd` member hostname.

Generate client certificate

```
cfssl print-defaults csr > client.json
```

For client certificate we can ignore **hosts** values and set only **Common Name (CN)** to **client** value:

```
...
    "CN": "client",
    "hosts": [""],
...
```

Generate client certificate:

```
cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -profile=client client.json | cfssljson -bare client
```

Or without CSR json file:

```
echo '{"CN":"client","hosts":[""],"key":{"algo":"rsa","size":2048}}' | cfssl gencert -ca=ca.pem -ca-key=ca-key.pem -config=ca-config.json -profile=client - | cfssljson -bare client
```

You'll get following files:

```
client-key.pem
client.csr
client.pem
```

TLDR

Download binaries

```
mkdir ~/bin
curl -s -L -o ~/bin/cfssl https://pkg.cfssl.org/R1.2/cfssl_linux-amd64
curl -s -L -o ~/bin/cfssljson https://pkg.cfssl.org/R1.2/cfssljson_linux-amd64
chmod +x ~/bin/{cfssl,cfssljson}
export PATH=$PATH:~/bin
```

Create directory to store certificates:

```
mkdir ~/cfssl
cd ~/cfssl
```

Generate CA and certificates

```
echo '{"CN":"CA","key":{"algo":"rsa","size":2048}}' | cfssl gencert -initca - | cfssljson -bare ca -
echo '{"signing":{"default":{"expiry":"43800h","usages":["signing","key encipherment","server auth","client auth"]}}}' > ca-config.json
export ADDRESS=192.168.122.68,ext1.example.com,coreos1.local,coreos1
export NAME=server
echo '{"CN":"'$NAME'","hosts":[""],"key":{"algo":"rsa","size":2048}}' | cfssl gencert -config=ca-config.json -ca=ca.pem -ca-key=ca-key.pem -hostname="$ADDRESS" - | cfssljson -bare $NAME
export ADDRESS=
export NAME=client
echo '{"CN":"'$NAME'","hosts":[""],"key":{"algo":"rsa","size":2048}}' | cfssl gencert -config=ca-config.json -ca=ca.pem -ca-key=ca-key.pem -hostname="$ADDRESS" - | cfssljson -bare $NAME
```

Verify data

```
openssl x509 -in ca.pem -text -noout
openssl x509 -in server.pem -text -noout
openssl x509 -in client.pem -text -noout
```

Things to know

- Don't put your `ca-key.pem` into a Container Linux Config, it is recommended to store it in safe place. This key allows to generate as much certificates as possible.
- Keep **key** files in safe. Don't forget to set proper file permissions, i.e. `chmod 0600 server-key.pem`.
- Certificates in this **TLDR** example have both `server auth` and `client auth` X509 V3 extensions and you can use them with servers and clients' authentication.
- You are free to generate keys and certificates for wildcard `*` address as well. They will work on any machine. It will simplify certificates routine but increase security risks.

More information

For another examples, check out these documents:

Custom Certificate Authorities (adding-certificate-authorities.html) etcd Security Model (/etcd/docs/latest/op-guide/security.html)