hyperledger-fabric-ca Documentation

Release main

hyperledger

Aug 17, 2021

This build of the docs is from the “main” branch

CHAPTER 1

Fabric CA User's Guide

The Hyperledger Fabric CA is a Certificate Authority (CA) for Hyperledger Fabric.

It provides features such as:

- registration of identities, or connects to LDAP as the user registry
- issuance of Enrollment Certificates (ECerts)
- certificate renewal and revocation

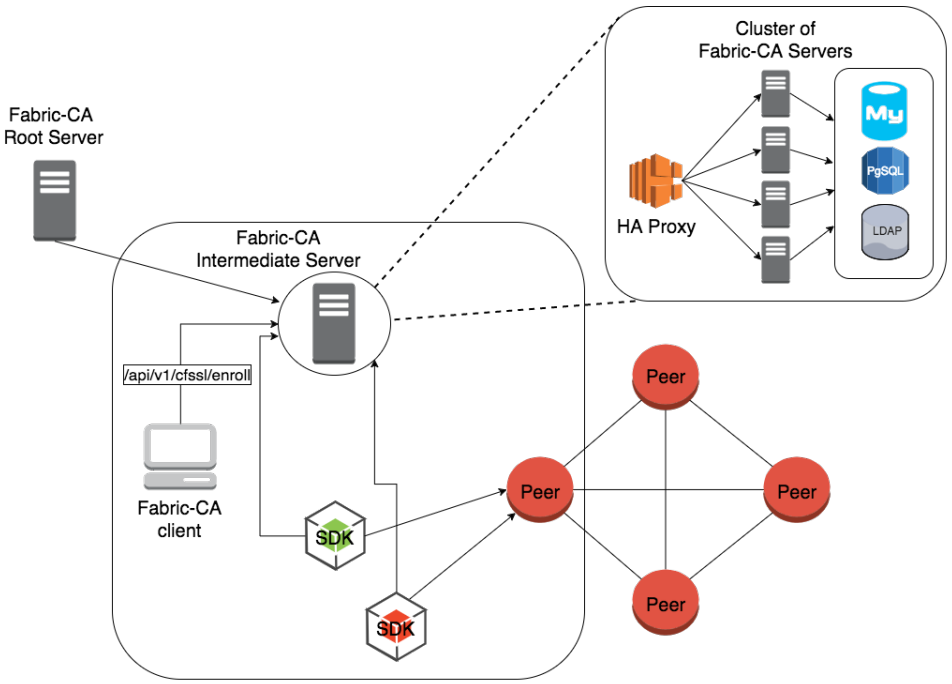
Hyperledger Fabric CA consists of both a server and a client component as described later in this document.

For developers interested in contributing to Hyperledger Fabric CA, see the Fabric CA repository for more information.

1.1 Table of Contents

1.2 Overview

The diagram below illustrates how the Hyperledger Fabric CA server fits into the overall Hyperledger Fabric architecture.



There are two ways of interacting with a Hyperledger Fabric CA server: via the Hyperledger Fabric CA client or through one of the Fabric SDKs. All communication to the Hyperledger Fabric CA server is via REST APIs. See *fabric-ca/swagger/swagger-fabric-ca.json* for the swagger documentation for these REST APIs. You may view this documentation via the Swagger online editor.

The Hyperledger Fabric CA client or SDK may connect to a server in a cluster of Hyperledger Fabric CA servers. This is illustrated in the top right section of the diagram. The client routes to an HA Proxy endpoint which load balances traffic to one of the fabric-ca-server cluster members.

All Hyperledger Fabric CA servers in a cluster share the same database for keeping track of identities and certificates. If LDAP is configured, the identity information is kept in LDAP rather than the database.

A server may contain multiple CAs. Each CA is either a root CA or an intermediate CA. Each intermediate CA has a parent CA which is either a root CA or another intermediate CA.

1.3 Getting Started

1.3.1 Prerequisites

- Go 1.10+ installation
- `GOPATH` environment variable is set correctly
- libtool and libtdhl-dev packages are installed

The following installs the libtool dependencies on Ubuntu:

```
sudo apt install libtool libltdl-dev
```

The following installs the libtool dependencies on MacOSX:

```
brew install libtool
```

Note: libtldl-dev is not necessary on MacOSX if you instal libtool via Homebrew

For more information on libtool, see https://www.gnu.org/software/libtool.

For more information on libltdl-dev, see https://www.gnu.org/software/libtool/manual/html_node/Using-libltdl.html.

1.3.2 Install

The following installs both the *fabric-ca-server* and *fabric-ca-client* binaries in $GOPATH/bin.

```
go get -u github.com/hyperledger/fabric-ca/cmd/...
```

Note: If you have already cloned the fabric-ca repository, make sure you are on the main branch before running the 'go get' command above. Otherwise, you might see the following error:

```
<gopath>/src/github.com/hyperledger/fabric-ca; git pull --ff-only
There is no tracking information for the current branch.
Please specify which branch you want to merge with.
See git-pull(1) for details.

    git pull <remote> <branch>

If you wish to set tracking information for this branch you can do so with:

    git branch --set-upstream-to=<remote>/<branch> tlsdoc

package github.com/hyperledger/fabric-ca/cmd/fabric-ca-client: exit status 1
```

1.3.3 Start Server Natively

The following starts the *fabric-ca-server* with default settings.

```
fabric-ca-server start -b admin:adminpw
```

The *-b* option provides the enrollment ID and secret for a bootstrap administrator; this is required if LDAP is not enabled with the "ldap.enabled" setting.

A default configuration file named *fabric-ca-server-config.yaml* is created in the local directory which can be customized.

1.3.4 Start Server via Docker

Docker Hub

Go to: https://hub.docker.com/r/hyperledger/fabric-ca/tags/

Find the tag that matches the architecture and version of fabric-ca that you want to pull.

Create a *docker-compose.yml* file like the one below. Change the *image* line to reflect the tag you found previously.

```
fabric-ca-server:
  image: hyperledger/fabric-ca:amd64-1.4.7
  container_name: fabric-ca-server
  ports:
    - "7054:7054"
  environment:
    - FABRIC_CA_HOME=/etc/hyperledger/fabric-ca-server
  volumes:
    - "./fabric-ca-server:/etc/hyperledger/fabric-ca-server"
  command: sh -c 'fabric-ca-server start -b admin:adminpw'
```

Open up a terminal in the same directory as the docker-compose.yml file and execute the following:

```
# docker-compose up -d
```

This will pull down the specified fabric-ca image in the Compose file if it does not already exist, and start an instance of the fabric-ca server.

Building Your Own Docker image

You can build and start the server via Docker Compose as shown below.

```
cd $GOPATH/src/github.com/hyperledger/fabric-ca
make docker
cd docker/server
docker-compose up -d
```

The hyperledger/fabric-ca Docker image contains both the fabric-ca-server and the fabric-ca-client.

```
# cd $GOPATH/src/github.com/hyperledger/fabric-ca
# FABRIC_CA_DYNAMIC_LINK=true make docker
# cd docker/server
# docker-compose up -d
```

1.3.5 Explore the Fabric CA CLI

This section simply provides the usage messages for the Fabric CA server and client for convenience. Additional usage information is provided in following sections.

The following links shows the Server Command Line and Client Command Line.

Note: Note that command line options that are string slices (lists) can be specified either by specifying the option with comma-separated list elements or by specifying the option multiple times, each with a string value that make up the list. For example, to specify `host1` and `host2` for the `csr.hosts` option, you can either pass `--csr.hosts 'host1,host2'` or `--csr.hosts host1 --csr.hosts host2`. When using the former format, please make sure there are no space before or after any commas.

Back to Top

1.3.6 Configuration Settings

The Fabric CA provides 3 ways to configure settings on the Fabric CA server and client. The precedence order is:

1. CLI flags
2. Environment variables
3. Configuration file

In the remainder of this document, we refer to making changes to configuration files. However, configuration file changes can be overridden through environment variables or CLI flags.

For example, if we have the following in the client configuration file:

```
tls:
  # Enable TLS (default: false)
  enabled: false

  # TLS for the client's listenting port (default: false)
  certfiles:
  client:
    certfile: cert.pem
    keyfile:
```

The following environment variable may be used to override the `cert.pem` setting in the configuration file:

```
export FABRIC_CA_CLIENT_TLS_CLIENT_CERTFILE=cert2.pem
```

If we wanted to override both the environment variable and configuration file, we can use a command line flag.

```
fabric-ca-client enroll --tls.client.certfile cert3.pem
```

The same approach applies to fabric-ca-server, except instead of using `FABRIC_CA_CLIENT` as the prefix to environment variables, `FABRIC_CA_SERVER` is used.

A word on file paths

All the properties in the Fabric CA server and client configuration file that specify file names support both relative and absolute paths. Relative paths are relative to the config directory, where the configuration file is located. For example, if the config directory is `~/config` and the tls section is as shown below, the Fabric CA server or client will look for the `root.pem` file in the `~/config` directory, `cert.pem` file in the `~/config/certs` directory and the `key.pem` file in the `/abs/path` directory

```
tls:
  enabled: true
  certfiles:
    - root.pem
  client:
    certfile: certs/cert.pem
    keyfile: /abs/path/key.pem
```

Back to Top

1.4 Fabric CA Server

This section describes the Fabric CA server.

You may initialize the Fabric CA server before starting it. This provides an opportunity for you to generate a default configuration file that can be reviewed and customized before starting the server.

The Fabric CA server's home directory is determined as follows:

- if the –home command line option is set, use its value
- otherwise, if the `FABRIC_CA_SERVER_HOME` environment variable is set, use its value
- otherwise, if `FABRIC_CA_HOME` environment variable is set, use its value
- otherwise, if the `CA_CFG_PATH` environment variable is set, use its value
- otherwise, use current working directory

For the remainder of this server section, we assume that you have set the `FABRIC_CA_HOME` environment variable to `$HOME/fabric-ca/server`.

The instructions below assume that the server configuration file exists in the server's home directory.

1.4.1 Initializing the server

Initialize the Fabric CA server as follows:

```
fabric-ca-server init -b admin:adminpw
```

The `-b` (bootstrap identity) option is required for initialization when LDAP is disabled. At least one bootstrap identity is required to start the Fabric CA server; this identity is the server administrator.

The server configuration file contains a Certificate Signing Request (CSR) section that can be configured. The following is a sample CSR.

```
cn: fabric-ca-server
names:
   - C: US
     ST: "North Carolina"
     L:
     O: Hyperledger
     OU: Fabric
hosts:
  - host1.example.com
  - localhost
ca:
   expiry: 131400h
   pathlength: 1
```

All of the fields above pertain to the X.509 signing key and certificate which is generated by the `fabric-ca-server init`. This corresponds to the `ca.certfile` and `ca.keyfile` files in the server's configuration file. The fields are as follows:

- **cn** is the Common Name
- **O** is the organization name
- **OU** is the organizational unit
- **L** is the location or city
- **ST** is the state
- **C** is the country

If custom values for the CSR are required, you may customize the configuration file, delete the files specified by the `ca.certfile` and `ca.keyfile` configuration items, and then run the `fabric-ca-server init -b admin:adminpw` command again.

The `fabric-ca-server init` command generates a self-signed CA certificate unless the `-u <parent-fabric-ca-server-URL>` option is specified. If the `-u` is specified, the server's CA certificate is signed by the parent Fabric CA server. In order to authenticate to the parent Fabric CA server, the URL must be of the form `<scheme>://<enrollmentID>:<secret>@<host>:<port>` , where <enrollmentID> and <secret> correspond to an identity with an 'hf.IntermediateCA' attribute whose value equals 'true'. The `fabric-ca-server init` command also generates a default configuration file named **fabric-ca-server-config.yaml** in the server's home directory.

If you want the Fabric CA server to use a CA signing certificate and key file which you provide, you must place your files in the location referenced by `ca.certfile` and `ca.keyfile` respectively. Both files must be PEM-encoded and must not be encrypted. More specifically, the contents of the CA certificate file must begin with `-----BEGIN CERTIFICATE-----` and the contents of the key file must begin with `-----BEGIN PRIVATE KEY-----` and not `-----BEGIN ENCRYPTED PRIVATE KEY-----` .

Algorithms and key sizes

The CSR can be customized to generate X.509 certificates and keys that support Elliptic Curve (ECDSA). The following setting is an example of the implementation of Elliptic Curve Digital Signature Algorithm (ECDSA) with curve `prime256v1` and signature algorithm `ecdsa-with-SHA256` :

```
key:
   algo: ecdsa
   size: 256
```

The choice of algorithm and key size are based on security needs.

Elliptic Curve (ECDSA) offers the following key size options:

size	ASN1 OID	Signature Algorithm
256	prime256v1	ecdsa-with-SHA256
384	secp384r1	ecdsa-with-SHA384

1.4.2 Starting the server

Start the Fabric CA server as follows:

```
fabric-ca-server start -b <admin>:<adminpw>
```

If the server has not been previously initialized, it will initialize itself as it starts for the first time. During this initialization, the server will generate the ca-cert.pem and ca-key.pem files if they don't yet exist and will also create a default configuration file if it does not exist. See the *Initialize the Fabric CA server* section.

Unless the Fabric CA server is configured to use LDAP, it must be configured with at least one pre-registered bootstrap identity to enable you to register and enroll other identities. The `-b` option specifies the name and password for a bootstrap identity.

To cause the Fabric CA server to listen on `https` rather than `http` , set `tls.enabled` to `true` .

SECURITY WARNING: The Fabric CA server should always be started with TLS enabled (`tls.enabled` set to true). Failure to do so leaves the server vulnerable to an attacker with access to network traffic.

To limit the number of times that the same secret (or password) can be used for enrollment, set the `registry.maxenrollments` in the configuration file to the appropriate value. If you set the value to 1, the Fabric CA server allows passwords to only be used once for a particular enrollment ID. If you set the value to -1, the Fabric CA server places no limit on the number of times that a secret can be reused for enrollment. The default value is -1. Setting the value to 0, the Fabric CA server will disable enrollment for all identities and registration of identities will not be allowed.

The Fabric CA server should now be listening on port 7054.

You may skip to the *Fabric CA Client* section if you do not want to configure the Fabric CA server to run in a cluster or to use LDAP.

1.4.3 Configuring the database

This section describes how to configure the Fabric CA server to connect to PostgreSQL or MySQL databases. The default database is SQLite and the default database file is `fabric-ca-server.db` in the Fabric CA server's home directory.

If you don't care about running the Fabric CA server in a cluster, you may skip this section; otherwise, you must configure either PostgreSQL or MySQL as described below. Fabric CA supports the following database versions in a cluster setup:

- PostgreSQL: 9.5.5 or later
- MySQL: 5.7 or later

PostgreSQL

The following sample may be added to the server's configuration file in order to connect to a PostgreSQL database. Be sure to customize the various values appropriately. There are limitations on what characters are allowed in the database name. Please refer to the following Postgres documentation for more information: https://www.postgresql.org/docs/current/static/sql-syntax-lexical.html#SQL-SYNTAX-IDENTIFIERS

```
db:
  type: postgres
  datasource: host=localhost port=5432 user=Username password=Password dbname=fabric_
↪ca sslmode=verify-full
```

Specifying *sslmode* configures the type of SSL authentication. Valid values for sslmode are:

Mode	Description
disable	No SSL
require	Always SSL (skip verification)
verify-ca	Always SSL (verify that the certificate presented by the server was signed by a trusted CA)
verify-full	Same as verify-ca AND verify that the certificate presented by the server was signed by a trusted CA and the server hostname matches the one in the certificate

If you would like to use TLS, then the `db.tls` section in the Fabric CA server configuration file must be specified. If SSL client authentication is enabled on the PostgreSQL server, then the client certificate and key file must also be specified in the `db.tls.client` section. The following is an example of the `db.tls` section:

```
db:
  ...
  tls:
      enabled: true
      certfiles:
        - db-server-cert.pem
      client:
            certfile: db-client-cert.pem
            keyfile: db-client-key.pem
```

certfiles - A list of PEM-encoded trusted root certificate files.
certfile and **keyfile** - PEM-encoded certificate and key files that are used by the Fabric CA server to communicate securely with the PostgreSQL server

PostgreSQL SSL Configuration

Basic instructions for configuring SSL on the PostgreSQL server:

1. In postgresql.conf, uncomment SSL and set to "on" (SSL=on)
2. Place certificate and key files in the PostgreSQL data directory.

Instructions for generating self-signed certificates for: https://www.postgresql.org/docs/9.5/static/ssl-tcp.html

Note: Self-signed certificates are for testing purposes and should not be used in a production environment

PostgreSQL Server - Require Client Certificates

1. Place certificates of the certificate authorities (CAs) you trust in the file root.crt in the PostgreSQL data directory
2. In postgresql.conf, set "ssl_ca_file" to point to the root cert of the client (CA cert)
3. Set the clientcert parameter to 1 on the appropriate hostssl line(s) in pg_hba.conf.

For more details on configuring SSL on the PostgreSQL server, please refer to the following PostgreSQL documentation: https://www.postgresql.org/docs/9.4/static/libpq-ssl.html

MySQL

The following sample may be added to the Fabric CA server configuration file in order to connect to a MySQL database. Be sure to customize the various values appropriately. There are limitations on what characters are allowed in the database name. Please refer to the following MySQL documentation for more information: https://dev.mysql.com/doc/refman/5.7/en/identifiers.html

On MySQL 5.7.X, certain modes affect whether the server permits '0000-00-00' as a valid date. It might be necessary to relax the modes that MySQL server uses. We want to allow the server to be able to accept zero date values.

In my.cnf, find the configuration option *sql_mode* and remove *NO_ZERO_DATE* if present. Restart MySQL server after making this change.

Please refer to the following MySQL documentation on different modes available and select the appropriate settings for the specific version of MySQL that is being used.

https://dev.mysql.com/doc/refman/5.7/en/sql-mode.html

```
db:
  type: mysql
  datasource: root:rootpw@tcp(localhost:3306)/fabric_ca?parseTime=true&tls=custom
```

If connecting over TLS to the MySQL server, the `db.tls.client` section is also required as described in the **PostgreSQL** section above.

MySQL SSL Configuration

Basic instructions for configuring SSL on MySQL server:

1. Open or create my.cnf file for the server. Add or uncomment the lines below in the [mysqld] section. These should point to the key and certificates for the server, and the root CA cert.

   Instructions on creating server and client-side certificates: http://dev.mysql.com/doc/refman/5.7/en/creating-ssl-files-using-openssl.html

   [mysqld] ssl-ca=ca-cert.pem ssl-cert=server-cert.pem ssl-key=server-key.pem

   Can run the following query to confirm SSL has been enabled.

   mysql> SHOW GLOBAL VARIABLES LIKE 'have_%ssl';

   Should see:

Variable_name	Value
have_openssl	YES
have_ssl	YES

2. After the server-side SSL configuration is finished, the next step is to create a user who has a privilege to access the MySQL server over SSL. For that, log in to the MySQL server, and type:

   mysql> GRANT ALL PRIVILEGES ON . TO 'ssluser'@'%' IDENTIFIED BY 'password' REQUIRE SSL;
   mysql> FLUSH PRIVILEGES;

   If you want to give a specific IP address from which the user will access the server change the '%' to the specific IP address.

MySQL Server - Require Client Certificates

Options for secure connections are similar to those used on the server side.

- ssl-ca identifies the Certificate Authority (CA) certificate. This option, if used, must specify the same certificate used by the server.
- ssl-cert identifies MySQL server's certificate.
- ssl-key identifies MySQL server's private key.

Suppose that you want to connect using an account that has no special encryption requirements or was created using a GRANT statement that includes the REQUIRE SSL option. As a recommended set of secure-connection options, start the MySQL server with at least –ssl-cert and –ssl-key options. Then set the `db.tls.certfiles` property in the server configuration file and start the Fabric CA server.

To require that a client certificate also be specified, create the account using the REQUIRE X509 option. Then the client must also specify proper client key and certificate files; otherwise, the MySQL server will reject the connection. To specify client key and certificate files for the Fabric CA server, set the `db.tls.client.certfile` , and `db.tls.client.keyfile` configuration properties.

1.4.4 Configuring LDAP

The Fabric CA server can be configured to read from an LDAP server.

In particular, the Fabric CA server may connect to an LDAP server to do the following:

- authenticate an identity prior to enrollment
- retrieve an identity's attribute values which are used for authorization.

Modify the LDAP section of the Fabric CA server's configuration file to configure the server to connect to an LDAP server.

```
ldap:
   # Enables or disables the LDAP client (default: false)
   enabled: false
   # The URL of the LDAP server
   url: <scheme>://<adminDN>:<adminPassword>@<host>:<port>/<base>
   userfilter: <filter>
   attribute:
      # 'names' is an array of strings that identify the specific attributes
      # which are requested from the LDAP server.
      names: <LDAPAttrs>
      # The 'converters' section is used to convert LDAP attribute values
      # to fabric CA attribute values.
      #
      # For example, the following converts an LDAP 'uid' attribute
      # whose value begins with 'revoker' to a fabric CA attribute
      # named "hf.Revoker" with a value of "true" (because the expression
      # evaluates to true).
      #    converters:
      #       - name: hf.Revoker
      #         value: attr("uid") =~ "revoker*"
      #
      # As another example, assume a user has an LDAP attribute named
      # 'member' which has multiple values of "dn1", "dn2", and "dn3".
      # Further assume the following configuration.
      #    converters:
      #       - name: myAttr
      #         value: map(attr("member"),"groups")
      #    maps:
      #       groups:
      #          - name: dn1
      #            value: client
      #          - name: dn2
      #            value: peer
      # The value of the user's 'myAttr' attribute is then computed to be
      # "client,peer,dn3".  This is because the value of 'attr("member")' is
      # "dn1,dn2,dn3", and the call to 'map' with a 2nd argument of
      # "group" replaces "dn1" with "client" and "dn2" with "peer".
      converters:
        - name: <fcaAttrName>
          value: <fcaExpr>
      maps:
        <mapName>:
            - name: <from>
              value: <to>
```

Where:

- `scheme` is one of *ldap* or *ldaps*;
- `adminDN` is the distinquished name of the admin user;
- `pass` is the password of the admin user;
- `host` is the hostname or IP address of the LDAP server;
- `port` is the optional port number, where default 389 for *ldap* and 636 for *ldaps*;
- `base` is the optional root of the LDAP tree to use for searches;
- `filter` is a filter to use when searching to convert a login user name to a distinguished name. For example, a value of `(uid=%s)` searches for LDAP entries with the value of a `uid` attribute whose value is the login user name. Similarly, `(email=%s)` may be used to login with an email address.
- `LDAPAttrs` is an array of LDAP attribute names to request from the LDAP server on a user's behalf;
- the attribute.converters section is used to convert LDAP attributes to fabric CA attributes, where * `fcaAttrName` is the name of a fabric CA attribute; * `fcaExpr` is an expression whose evaluated value is assigned to the fabric CA attribute. For example, suppose that <LDAPAttrs> is ["uid"], <fcaAttrName> is 'hf.Revoker', and <fcaExpr> is 'attr("uid") =~ "revoker*"'. This means that an attribute named "uid" is requested from the LDAP server on a user's behalf. The user is then given a value of 'true' for the 'hf.Revoker' attribute if the value of the user's 'uid' LDAP attribute begins with 'revoker'; otherwise, the user is given a value of 'false' for the 'hf.Revoker' attribute.
- the attribute.maps section is used to map LDAP response values. The typical use case is to map a distinguished name associated with an LDAP group to an identity type.

The LDAP expression language uses the govaluate package as described at https://github.com/Knetic/govaluate/blob/master/MANUAL.md. This defines operators such as "=~" and literals such as "revoker*", which is a regular expression. The LDAP-specific variables and functions which extend the base govaluate language are as follows:

- `DN` is a variable equal to the user's distinguished name.
- `affiliation` is a variable equal to the user's affiliation.
- `attr` is a function which takes 1 or 2 arguments. The 1st argument is an LDAP attribute name. The 2nd argument is a separator string which is used to join multiple values into a single string; the default separator string is ",". The `attr` function always returns a value of type 'string'.
- `map` is a function which takes 2 arguments. The 1st argument is any string. The second argument is the name of a map which is used to perform string substitution on the string from the 1st argument.
- `if` is a function which takes a 3 arguments where the first argument must resolve to a boolean value. If it evaluates to true, the second argument is returned; otherwise, the third argument is returned.

For example, the following expression evaluates to true if the user has a distinguished name ending in "O=org1,C=US", or if the user has an affiliation beginning with "org1.dept2." and also has the "admin" attribute of "true".

DN =~ "*O=org1,C=US" || (affiliation =~ "org1.dept2.*" && attr('admin') = 'true')

NOTE: Since the `attr` function always returns a value of type 'string', numeric operators may not be used to construct expressions. For example, the following is NOT a valid expression:

```
value: attr("gidNumber") >= 10000 && attr("gidNumber") < 10006
```

Alternatively, a regular expression enclosed in quotes as shown below may be used to return an equivalent result:

```
value: attr("gidNumber") =~ "1000[0-5]$" || attr("mail") == "root@example.com"
```

The following is a sample configuration section for the default setting for the OpenLDAP server whose Docker image is at `https://github.com/osixia/docker-openldap`.

```
ldap:
   enabled: true
   url: ldap://cn=admin,dc=example,dc=org:admin@localhost:10389/dc=example,dc=org
   userfilter: (uid=%s)
```

See `FABRIC_CA/scripts/run-ldap-tests` for a script which starts an OpenLDAP Docker image, configures it, runs the LDAP tests in `FABRIC_CA/cli/server/ldap/ldap_test.go`, and stops the OpenLDAP server.

When LDAP is configured, enrollment works as follows:

- The Fabric CA client or client SDK sends an enrollment request with a basic authorization header.
- The Fabric CA server receives the enrollment request, decodes the identity name and password in the authorization header, looks up the DN (Distinguished Name) associated with the identity name using the "userfilter" from the configuration file, and then attempts an LDAP bind with the identity's password. If the LDAP bind is successful, the enrollment processing is authorized and can proceed.

1.4.5 Setting up a cluster

You may use any IP sprayer to load balance to a cluster of Fabric CA servers. This section provides an example of how to set up Haproxy to route to a Fabric CA server cluster. Be sure to change hostname and port to reflect the settings of your Fabric CA servers.

haproxy.conf

```
global
      maxconn 4096
      daemon

defaults
      mode http
      maxconn 2000
      timeout connect 5000
      timeout client 50000
      timeout server 50000

listen http-in
      bind *:7054
      balance roundrobin
      server server1 hostname1:port
      server server2 hostname2:port
      server server3 hostname3:port
```

Note: If using TLS, need to use `mode tcp`.

1.4.6 Setting up multiple CAs

The fabric-ca server by default consists of a single default CA. However, additional CAs can be added to a single server by using *cafiles* or *cacount* configuration options. Each additional CA will have its own home directory.

cacount:

The *cacount* provides a quick way to start X number of default additional CAs. The home directory will be relative to the server directory. With this option, the directory structure will be as follows:

```
--<Server Home>
  |--ca
    |--ca1
    |--ca2
```

Each additional CA will get a default configuration file generated in it's home directory, within the configuration file it will contain a unique CA name.

For example, the following command will start 2 default CA instances:

```
fabric-ca-server start -b admin:adminpw --cacount 2
```

cafiles:

If absolute paths are not provided when using the cafiles configuration option, the CA home directory will be relative to the server directory.

To use this option, CA configuration files must have already been generated and configured for each CA that is to be started. Each configuration file must have a unique CA name and Common Name (CN), otherwise the server will fail to start as these names must be unique. The CA configuration files will override any default CA configuration, and any missing options in the CA configuration files will be replaced by the values from the default CA.

The precedence order will be as follows:

1. CA Configuration file
2. Default CA CLI flags
3. Default CA Environment variables
4. Default CA Configuration file

A CA configuration file must contain at least the following:

```
ca:
# Name of this CA
name: <CANAME>

csr:
  cn: <COMMONNAME>
```

You may configure your directory structure as follows:

```
--<Server Home>
  |--ca
    |--ca1
      |-- fabric-ca-config.yaml
    |--ca2
      |-- fabric-ca-config.yaml
```

For example, the following command will start two customized CA instances:

```
fabric-ca-server start -b admin:adminpw --cafiles ca/ca1/fabric-ca-config.yaml
--cafiles ca/ca2/fabric-ca-config.yaml
```

1.4.7 Enrolling an intermediate CA

In order to create a CA signing certificate for an intermediate CA, the intermediate CA must enroll with a parent CA in the same way that a fabric-ca-client enrolls with a CA. This is done by using the -u option to specify the URL of the parent CA and the enrollment ID and secret as shown below. The identity associated with this enrollment ID must have an attribute with a name of "hf.IntermediateCA" and a value of "true". The CN (or Common Name) of the issued certificate will be set to the enrollment ID. An error will occur if an intermediate CA tries to explicitly specify a CN value.

```
fabric-ca-server start -b admin:adminpw -u http://<enrollmentID>:<secret>@
↪<parentserver>:<parentport>
```

For other intermediate CA flags see *Fabric CA server's configuration file format* section.

1.4.8 Upgrading the server

The Fabric CA server must be upgraded before upgrading the Fabric CA client. Prior to upgrade, it is suggested that the current database be backed up:

- If using sqlite3, backup the current database file (which is named fabric-ca-server.db by default).
- For other database types, use the appropriate backup/replication mechanism.

To upgrade a single instance of Fabric CA server:

1. Stop the fabric-ca-server process.
2. Ensure the current database is backed up.
3. Replace previous fabric-ca-server binary with the upgraded version.
4. Launch the fabric-ca-server process.
5. Verify the fabric-ca-server process is available with the following command where <host> is the hostname on which the server was started:

   ```
   fabric-ca-client getcainfo -u http://<host>:7054
   ```

Upgrading a cluster:

To upgrade a cluster of fabric-ca-server instances using either a MySQL or Postgres database, perform the following procedure. We assume that you are using haproxy to load balance to two fabric-ca-server cluster members on host1 and host2, respectively, both listening on port 7054. After this procedure, you will be load balancing to upgraded fabric-ca-server cluster members on host3 and host4 respectively, both listening on port 7054.

In order to monitor the changes using haproxy stats, enable statistics collection. Add the following lines to the global section of the haproxy configuration file:

```
stats socket /var/run/haproxy.sock mode 666 level operator
stats timeout 2m
```

Restart haproxy to pick up the changes:

```
# haproxy -f <configfile> -st $(pgrep haproxy)
```

To display summary information from the haproxy "show stat" command, the following function may prove useful for parsing the copious amount of CSV data returned:

```
haProxyShowStats() {
   echo "show stat" | nc -U /var/run/haproxy.sock |sed '1s/^# *//'|
      awk -F',' -v fmt="%4s %12s %10s %6s %6s %4s %4s\n" '
         { if (NR==1) for (i=1;i<=NF;i++) f[tolower($i)]=i }
         { printf fmt, $f["sid"],$f["pxname"],$f["svname"],$f["status"],
                       $f["weight"],$f["act"],$f["bck"] }'
}
```

1. Initially your haproxy configuration file is similar to the following:

   ```
   server server1 host1:7054 check
   server server2 host2:7054 check
   ```

   Change this configuration to the following:

   ```
   server server1 host1:7054 check backup
   server server2 host2:7054 check backup
   server server3 host3:7054 check
   server server4 host4:7054 check
   ```

2. Restart the HA proxy with the new configuration as follows:

   ```
   haproxy -f <configfile> -st $(pgrep haproxy)
   ```

   `"haProxyShowStats"` will now reflect the modified configuration, with two active, older-version backup servers and two (yet to be started) upgraded servers:

   ```
   sid   pxname      svname  status  weig  act  bck
     1   fabric-cas  server3   DOWN     1    1    0
     2   fabric-cas  server4   DOWN     1    1    0
     3   fabric-cas  server1     UP     1    0    1
     4   fabric-cas  server2     UP     1    0    1
   ```

3. Install upgraded binaries of fabric-ca-server on host3 and host4. The new upgraded servers on host3 and host4 should be configured to use the same database as their older counterparts on host1 and host2. After starting the upgraded servers, the database will be automatically migrated. The haproxy will forward all new traffic to the upgraded servers, since they are not configured as backup servers. Verify using the `"fabric-ca-client getcainfo"` command that your cluster is still functioning appropriately before proceeding. Also, `"haProxyShowStats"` should now reflect that all servers are active, similar to the following:

   ```
   sid   pxname      svname  status  weig  act  bck
     1   fabric-cas  server3    UP     1    1    0
     2   fabric-cas  server4    UP     1    1    0
     3   fabric-cas  server1    UP     1    0    1
     4   fabric-cas  server2    UP     1    0    1
   ```

4. Stop the old servers on host1 and host2. Verify using the `"fabric-ca-client getcainfo"` command that your new cluster is still functioning appropriately before proceeding. Then remove the older server backup configuration from the haproxy configuration file, so that it looks similar to the following:

   ```
   server server3 host3:7054 check
   server server4 host4:7054 check
   ```

5. Restart the HA proxy with the new configuration as follows:

```
haproxy -f <configfile> -st $(pgrep haproxy)
```

"haProxyShowStats" will now reflect the modified configuration, with two active servers which have been upgraded to the new version:

```
sid   pxname      svname  status  weig  act  bck
  1   fabric-cas  server3   UP       1    1    0
  2   fabric-cas  server4   UP       1    1    0
```

Back to Top

1.4.9 Operations Service

The CA Server hosts an HTTP server that offers a RESTful "operations" API. This API is intended to be used by operators, not administrators or "users" of the network.

The API exposes the following capabilities:

Prometheus target for operational metrics (when configured)

Configuring the Operations Service

The operations service requires two basic pieces of configuration:

The **address** and **port** to listen on. The **TLS certificates** and **keys** to use for authentication and encryption. Note, **these certificates should be generated by a separate and dedicated CA**. Do not use a CA that has generated certificates for any organizations in any channels.

The CA server can be configurated in the operations section of server's configuration file:

```
operations:
  # host and port for the operations server
  listenAddress: 127.0.0.1:9443

  # TLS configuration for the operations endpoint
  tls:
    # TLS enabled
    enabled: true

    # path to PEM encoded server certificate for the operations server
    cert:
      file: tls/server.crt

    # path to PEM encoded server key for the operations server
    key:
      file: tls/server.key

    # require client certificate authentication to access all resources
    clientAuthRequired: false

    # paths to PEM encoded ca certificates to trust for client authentication
    clientRootCAs:
      files: []
```

The listenAddress key defines the host and port that the operation server will listen on. If the server should listen on all addresses, the host portion can be omitted.

The `tls` section is used to indicate whether or not TLS is enabled for the operations service, the location of the service's certificate and private key, and the locations of certificate authority root certificates that should be trusted for client authentication. When `clientAuthRequired` is `true`, clients will be required to provide a certificate for authentication.

Operations Security

As the operations service is focused on operations and intentionally unrelated to the Fabric network, it does not use the Membership Services Provider for access control. Instead, the operations service relies entirely on mutual TLS with client certificate authentication.

It is highly recommended to enable mutual TLS by setting the value of `clientAuthRequired` to `true` in production environments. With this configuration, clients are required to provide a valid certificate for authentication. If the client does not provide a certificate or the service cannot verify the client's certificate, the request is rejected. Note that if `clientAuthRequired` is set to `false`, clients do not need to provide a certificate; if they do, however, and the service cannot verify the certificate, then the request will be rejected.

When TLS is disabled, authorization is bypassed and any client that can connect to the operations endpoint will be able to use the API.

Metrics

The Fabric CA exposes metrics that can provide insight into the behavior of the system. Operators and administrators can use this information to better understand how the system is performing over time.

Configuring Metrics

Fabric CA provides two ways to expose metrics: a **pull** model based on Prometheus and a **push** model based on StatsD.

Prometheus

A typical Prometheus deployment scrapes metrics by requesting them from an HTTP endpoint exposed by instrumented targets. As Prometheus is responsible for requesting the metrics, it is considered a pull system.

When configured, a Fabric CA Server will present a `/metrics` resource on the operations service. To enable Prometheus, set the provider value in the server's configuration file to `prometheus`.

```
metrics:
  provider: prometheus
```

StatsD

StatsD is a simple statistics aggregation daemon. Metrics are sent to a `statsd` daemon where they are collected, aggregated, and pushed to a backend for visualization and alerting. As this model requires instrumented processes to send metrics data to StatsD, this is considered a push system.

The CA Server can be configured to send metrics to StatsD by setting the metrics provider to `statsd` in the `metrics` section in servers' configuration filel. The `statsd` subsection must also be configured with the address of the StatsD daemon, the network type to use (`tcp` or `udp`), and how often to send the metrics. An optional `prefix` may be specified to help differentiate the source of the metrics — for example, differentiating metrics coming from separate servers — that would be prepended to all generated metrics.

```
metrics:
  provider: statsd
  statsd:
    network: udp
    address: 127.0.0.1:8125
    writeInterval: 10s
    prefix: server-0
```

For a look at the different metrics that are generated, check out metrics_reference.

Back to Top

1.5 Fabric CA Client

This section describes how to use the fabric-ca-client command.

The Fabric CA client's home directory is determined as follows:

- if the –home command line option is set, use its value
- otherwise, if the `FABRIC_CA_CLIENT_HOME` environment variable is set, use its value
- otherwise, if the `FABRIC_CA_HOME` environment variable is set, use its value
- otherwise, if the `CA_CFG_PATH` environment variable is set, use its value
- otherwise, use `$HOME/.fabric-ca-client`

The instructions below assume that the client configuration file exists in the client's home directory.

1.5.1 Enrolling the bootstrap identity

First, if needed, customize the CSR (Certificate Signing Request) section in the client configuration file. Note that `csr.cn` field must be set to the ID of the bootstrap identity. Default CSR values are shown below:

```
csr:
  cn: <<enrollment ID>>
  key:
    algo: ecdsa
    size: 256
  names:
    - C: US
      ST: North Carolina
      L:
      O: Hyperledger Fabric
      OU: Fabric CA
  hosts:
   - <<hostname of the fabric-ca-client>>
  ca:
    pathlen:
    pathlenzero:
    expiry:
```

See *CSR fields* for description of the fields.

Then run `fabric-ca-client enroll` command to enroll the identity. For example, following command enrolls an identity whose ID is **admin** and password is **adminpw** by calling Fabric CA server that is running locally at 7054 port.

```
export FABRIC_CA_CLIENT_HOME=$HOME/fabric-ca/clients/admin
fabric-ca-client enroll -u http://admin:adminpw@localhost:7054
```

The enroll command stores an enrollment certificate (ECert), corresponding private key and CA certificate chain PEM files in the subdirectories of the Fabric CA client's `msp` directory. You will see messages indicating where the PEM files are stored.

1.5.2 Registering a new identity

The identity performing the register request must be currently enrolled, and must also have the proper authority to register the type of the identity that is being registered.

In particular, three authorization checks are made by the Fabric CA server during registration as follows:

1. The registrar (i.e. the invoker) must have the "hf.Registrar.Roles" attribute with a comma-separated list of values where one of the values equals the type of identity being registered; for example, if the registrar has the "hf.Registrar.Roles" attribute with a value of "peer", the registrar can register identities of type peer, but not client, admin, or orderer.
2. The affiliation of the registrar must be equal to or a prefix of the affiliation of the identity being registered. For example, an registrar with an affiliation of "a.b" may register an identity with an affiliation of "a.b.c" but may not register an identity with an affiliation of "a.c". If root affiliation is required for an identity, then the affiliation request should be a dot (".") and the registrar must also have root affiliation. If no affiliation is specified in the registration request, the identity being registered will be given the affiliation of the registrar.
3. The registrar can register an identity with attributes if all of the following conditions are satisfied:
   - Registrar can register Fabric CA reserved attributes that have the prefix 'hf.' only if the registrar possesses the attribute and it is part of the value of the hf.Registrar.Attributes' attribute. Furthermore, if the attribute is of type list then the value of attribute being registered must be equal to or a subset of the value that the registrar has. If the attribute is of type boolean, the registrar can register the attribute only if the registrar's value for the attribute is 'true'.
   - Registering custom attributes (i.e. any attribute whose name does not begin with 'hf.') requires that the registrar has the 'hf.Registar.Attributes' attribute with the value of the attribute or pattern being registered. The only supported pattern is a string with a "*" at the end. For example, "a.b.*" is a pattern which matches all attribute names beginning with "a.b.". For example, if the registrar has hf.Registrar.Attributes=orgAdmin, then the only attribute which the registrar can add or remove from an identity is the 'orgAdmin' attribute.
   - If the requested attribute name is 'hf.Registrar.Attributes', an additional check is performed to see if the requested values for this attribute are equal to or a subset of the registrar's values for 'hf.Registrar.Attributes'. For this to be true, each requested value must match a value in the registrar's value for 'hf.Registrar.Attributes' attribute. For example, if the registrar's value for 'hf.Registrar.Attributes' is 'a.b.*, x.y.z' and the requested attribute value is 'a.b.c, x.y.z', it is valid because 'a.b.c' matches 'a.b.*' and 'x.y.z' matches the registrar's 'x.y.z' value.

Examples:

Valid Scenarios:

1. If the registrar has the attribute 'hf.Registrar.Attributes = a.b.*, x.y.z' and is registering attribute 'a.b.c', it is valid 'a.b.c' matches 'a.b.*'.
2. If the registrar has the attribute 'hf.Registrar.Attributes = a.b.*, x.y.z' and is registering attribute 'x.y.z', it is valid because 'x.y.z' matches the registrar's 'x.y.z' value.
3. If the registrar has the attribute 'hf.Registrar.Attributes = a.b.*, x.y.z' and the requested attribute value is 'a.b.c, x.y.z', it is valid because 'a.b.c' matches 'a.b.*' and 'x.y.z' matches the registrar's 'x.y.z' value.

4. If the registrar has the attribute 'hf.Registrar.Roles = peer,client,admin,orderer' and the requested attribute value is 'peer', 'peer,client,admin,orderer', or 'client,admin' it is valid because the requested value is equal to or a subset of the registrar's value.

Invalid Scenarios:

1. If the registrar has the attribute 'hf.Registrar.Attributes = a.b.*, x.y.z' and is registering attribute 'hf.Registar.Attributes = a.b.c, x.y.*', it is invalid because requested attribute 'x.y.*' is not a pattern owned by the registrar. The value 'x.y.*' is a superset of 'x.y.z'.
2. If the registrar has the attribute 'hf.Registrar.Attributes = a.b.*, x.y.z' and is registering attribute 'hf.Registar.Attributes = a.b.c, x.y.z, attr1', it is invalid because the registrar's 'hf.Registrar.Attributes' attribute values do not contain 'attr1'.
3. If the registrar has the attribute 'hf.Registrar.Attributes = a.b.*, x.y.z' and is registering attribute 'a.b', it is invalid because the value 'a.b' is not contained in 'a.b.*'.
4. If the registrar has the attribute 'hf.Registrar.Attributes = a.b.*, x.y.z' and is registering attribute 'x.y', it is invalid because 'x.y' is not contained by 'x.y.z'.
5. If the registrar has the attribute 'hf.Registrar.Roles = peer' and the requested attribute value is 'peer,client', it is invalid because the registrar does not have the client role in its value of hf.Registrar.Roles attribute.
6. If the registrar has the attribute 'hf.Revoker = false' and the requested attribute value is 'true', it is invalid because the hf.Revoker attribute is a boolean attribute and the registrar's value for the attribute is not 'true'.

The table below lists all the attributes that can be registered for an identity. The names of attributes are case sensitive.

Name	Type	Description
hf.Registrar.Roles	List	List of roles that the registrar is allowed to manage
hf.Registrar.DelegateRoles	List	List of roles that the registrar is allowed to give to a registree for its 'hf.Registrar.Roles' attribute
hf.Registrar.Attributes	List	List of attributes that registrar is allowed to register
hf.GenCRL	Boolean	Identity is able to generate CRL if attribute value is true
hf.Revoker	Boolean	Identity is able to revoke an identity and/or certificates if attribute value is true
hf.AffiliationMgr	Boolean	Identity is able to manage affiliations if attribute value is true
hf.IntermediateCA	Boolean	Identity is able to enroll as an intermediate CA if attribute value is true

Note: When registering an identity, you specify an array of attribute names and values. If the array specifies multiple array elements with the same name, only the last element is currently used. In other words, multi-valued attributes are not currently supported.

The following command uses the **admin** identity's credentials to register a new identity with an enrollment id of "admin2", an affiliation of "org1.department1", an attribute named "hf.Revoker" with a value of "true", and an attribute named "admin" with a value of "true". The ":ecert" suffix means that by default the "admin" attribute and its value will be inserted into the identity's enrollment certificate, which can then be used to make access control decisions.

```
export FABRIC_CA_CLIENT_HOME=$HOME/fabric-ca/clients/admin
fabric-ca-client register --id.name admin2 --id.affiliation org1.department1 --id.
↪attrs 'hf.Revoker=true,admin=true:ecert'
```

The password, also known as the enrollment secret, is printed. This password is required to enroll the identity. This allows an administrator to register an identity and give the enrollment ID and the secret to someone else to enroll the identity.

Multiple attributes can be specified as part of the –id.attrs flag, each attribute must be comma separated. For an attribute value that contains a comma, the attribute must be encapsulated in double quotes. See example below.

```
fabric-ca-client register -d --id.name admin2 --id.affiliation org1.department1 --id.
↪attrs '"hf.Registrar.Roles=peer,client",hf.Revoker=true'
```

or

```
fabric-ca-client register -d --id.name admin2 --id.affiliation org1.department1 --id.
↪attrs '"hf.Registrar.Roles=peer,client"' --id.attrs hf.Revoker=true
```

You may set default values for any of the fields used in the register command by editing the client's configuration file. For example, suppose the configuration file contains the following:

```
id:
  name:
  type: client
  affiliation: org1.department1
  maxenrollments: -1
  attributes:
    - name: hf.Revoker
      value: true
    - name: anotherAttrName
      value: anotherAttrValue
```

The following command would then register a new identity with an enrollment id of "admin3" which it takes from the command line, and the remainder is taken from the configuration file including the identity type: "client", affiliation: "org1.department1", and two attributes: "hf.Revoker" and "anotherAttrName".

```
export FABRIC_CA_CLIENT_HOME=$HOME/fabric-ca/clients/admin
fabric-ca-client register --id.name admin3
```

To register an identity with multiple attributes requires specifying all attribute names and values in the configuration file as shown above.

Setting *maxenrollments* to 0 or leaving it out from the configuration will result in the identity being registered to use the CA's max enrollment value. Furthermore, the max enrollment value for an identity being registered cannot exceed the CA's max enrollment value. For example, if the CA's max enrollment value is 5. Any new identity must have a value less than or equal to 5, and also can't set it to -1 (infinite enrollments).

Next, let's register a peer identity which will be used to enroll the peer in the following section. The following command registers the **peer1** identity. Note that we choose to specify our own password (or secret) rather than letting the server generate one for us.

```
export FABRIC_CA_CLIENT_HOME=$HOME/fabric-ca/clients/admin
fabric-ca-client register --id.name peer1 --id.type peer --id.affiliation org1.
↪department1 --id.secret peer1pw
```

Note that affiliations are case sensitive except for the non-leaf affiliations that are specified in the server configuration file, which are always stored in lower case. For example, if the affiliations section of the server configuration file looks like this:

```
affiliations:
  BU1:
    Department1:
      - Team1
  BU2:
    - Department2
    - Department3
```

BU1, *Department1*, *BU2* are stored in lower case. This is because Fabric CA uses Viper to read configuration. Viper treats map keys as case insensitive and always returns lowercase value. To register an identity with *Team1* affiliation, *bu1.department1.Team1* would need to be specified to the *–id.affiliation* flag as shown below:

```
export FABRIC_CA_CLIENT_HOME=$HOME/fabric-ca/clients/admin
fabric-ca-client register --id.name client1 --id.type client --id.affiliation bu1.
↪department1.Team1
```

1.5.3 Enrolling a peer identity

Now that you have successfully registered a peer identity, you may now enroll the peer given the enrollment ID and secret (i.e. the *password* from the previous section). This is similar to enrolling the bootstrap identity except that we also demonstrate how to use the "-M" option to populate the Hyperledger Fabric MSP (Membership Service Provider) directory structure.

The following command enrolls peer1. Be sure to replace the value of the "-M" option with the path to your peer's MSP directory which is the 'mspConfigPath' setting in the peer's core.yaml file. You may also set the FABRIC_CA_CLIENT_HOME to the home directory of your peer.

```
export FABRIC_CA_CLIENT_HOME=$HOME/fabric-ca/clients/peer1
fabric-ca-client enroll -u http://peer1:peer1pw@localhost:7054 -M $FABRIC_CA_CLIENT_
↪HOME/msp
```

Enrolling an orderer is the same, except the path to the MSP directory is the 'LocalMSPDir' setting in your orderer's orderer.yaml file.

All enrollment certificates issued by the fabric-ca-server have organizational units (or "OUs" for short) as follows:

1. The root of the OU hierarchy equals the identity type
2. An OU is added for each component of the identity's affiliation

For example, if an identity is of type *peer* and its affiliation is *department1.team1*, the identity's OU hierarchy (from leaf to root) is *OU=team1, OU=department1, OU=peer*.

1.5.4 Getting Identity Mixer credential

Identity Mixer (Idemix) is a cryptographic protocol suite for privacy-preserving authentication and transfer of certified attributes. Idemix allows clients to authenticate with verifiers without the involvement of the issuer (CA) and selectively disclose only those attributes that are required by the verifier and can do so without being linkable across their transactions.

Fabric CA server can issue Idemix credentials in addition to X509 certificates. An Idemix credential can be requested by sending the request to the `/api/v1/idemix/credential` API endpoint. For more information on this and other Fabric CA server API endpoints, please refer to swagger-fabric-ca.json.

The Idemix credential issuance is a two step process. First, send a request with an empty body to the `/api/v1/idemix/credential` API endpoint to get a nonce and CA's Idemix public key. Second, create a credential request using the nonce and CA's Idemix public key and send another request with the credential request in the body to the `/api/v1/idemix/credential` API endpoint to get an Idemix credential, Credential Revocation Information (CRI), and attribute names and values. Currently, only three attributes are supported:

- **OU** - organization unit of the identity. The value of this attribute is set to identity's affiliation. For example, if identity's affiliation is *dept1.unit1*, then OU attribute is set to *dept1.unit1*
- **IsAdmin** - if the identity is an admin or not. The value of this attribute is set to the value of *isAdmin* registration attribute.

- **EnrollmentID** - enrollment ID of the identity

You can refer to the *handleIdemixEnroll* function in https://github.com/hyperledger/fabric-ca/blob/main/lib/client.go for reference implementation of the two step process for getting Idemix credential.

The `/api/v1/idemix/credential` API endpoint accepts both basic and token authorization headers. The basic authorization header should contain User's registration ID and password. If the identity already has X509 enrollment certificate, it can also be used to create a token authorization header.

Note that Hyperledger Fabric will support clients to sign transactions with both X509 and Idemix credentials, but will only support X509 credentials for peer and orderer identities. As before, applications can use a Fabric SDK to send requests to the Fabric CA server. SDKs hide the complexity associated with creating authorization header and request payload, and with processing the response.

1.6 Getting Idemix CRI (Certificate Revocation Information)

An Idemix CRI (Credential Revocation Information) is similar in purpose to an X509 CRL (Certificate Revocation List): to revoke what was previously issued. However, there are some differences.

In X509, the issuer revokes an end user's certificate and its ID is included in the CRL. The verifier checks to see if the user's certificate is in the CRL and if so, returns an authorization failure. The end user is not involved in this revocation process, other than receiving an authorization error from a verifier.

In Idemix, the end user is involved. The issuer revokes an end user's credential similar to X509 and evidence of this revocation is recorded in the CRI. The CRI is given to the end user (aka "prover"). The end user then generates a proof that their credential has not been revoked according to the CRI. The end user gives this proof to the verifier who verifies the proof according to the CRI. For verification to succeed, the version of the CRI (known as the "epoch") used by the end user and verifier must be same. The latest CRI can be requested by sending a request to `/api/v1/idemix/cri` API endpoint.

The version of the CRI is incremented when an enroll request is received by the fabric-ca-server and there are no revocation handles remaining in the revocation handle pool. In this case, the fabric-ca-server must generate a new pool of revocation handles which increments the epoch of the CRI. The number of revocation handles in the revocation handle pool is configurable via the `idemix.rhpoolsize` server configuration property.

1.6.1 Reenrolling an identity

Suppose your enrollment certificate is about to expire or has been compromised. You can issue the reenroll command to renew your enrollment certificate as follows.

```
export FABRIC_CA_CLIENT_HOME=$HOME/fabric-ca/clients/peer1
fabric-ca-client reenroll
```

1.6.2 Revoking a certificate or identity

An identity or a certificate can be revoked. Revoking an identity will revoke all the certificates owned by the identity and will also prevent the identity from getting any new certificates. Revoking a certificate will invalidate a single certificate.

In order to revoke a certificate or an identity, the calling identity must have the `hf.Revoker` and `hf.Registrar.Roles` attribute. The revoking identity can only revoke a certificate or an identity that has an affiliation that is equal to or prefixed by the revoking identity's affiliation. Furthermore, the revoker can only revoke identities with types that are listed in the revoker's `hf.Registrar.Roles` attribute.

For example, a revoker with affiliation **orgs.org1** and 'hf.Registrar.Roles=peer,client' attribute can revoke either a **peer** or **client** type identity affiliated with **orgs.org1** or **orgs.org1.department1** but can't revoke an identity affiliated with **orgs.org2** or of any other type.

The following command disables an identity and revokes all of the certificates associated with the identity. All future requests received by the Fabric CA server from this identity will be rejected.

```
fabric-ca-client revoke -e <enrollment_id> -r <reason>
```

The following are the supported reasons that can be specified using `-r` flag:

1. unspecified
2. keycompromise
3. cacompromise
4. affiliationchanged
5. superseded
6. cessationofoperation
7. certificatehold
8. removefromcrl
9. privilegewithdrawn
10. aacompromise

For example, the bootstrap admin who is associated with root of the affiliation tree can revoke **peer1**'s identity as follows:

```
export FABRIC_CA_CLIENT_HOME=$HOME/fabric-ca/clients/admin
fabric-ca-client revoke -e peer1
```

An enrollment certificate that belongs to an identity can be revoked by specifying its AKI (Authority Key Identifier) and serial number as follows:

```
fabric-ca-client revoke -a xxx -s yyy -r <reason>
```

For example, you can get the AKI and the serial number of a certificate using the openssl command and pass them to the `revoke` command to revoke the said certificate as follows:

```
serial=$(openssl x509 -in userecert.pem -serial -noout | cut -d "=" -f 2)
aki=$(openssl x509 -in userecert.pem -text | awk '/keyid/ {gsub(/ *keyid:|:/,"",$1);
↪print tolower($0)}')
fabric-ca-client revoke -s $serial -a $aki -r affiliationchange
```

The *–gencrl* flag can be used to generate a CRL (Certificate Revocation List) that contains all the revoked certificates. For example, following command will revoke the identity **peer1**, generates a CRL and stores it in the **<msp folder>/crls/crl.pem** file.

```
fabric-ca-client revoke -e peer1 --gencrl
```

A CRL can also be generated using the *gencrl* command. Refer to the *Generating a CRL (Certificate Revocation List)* section for more information on the *gencrl* command.

1.6.3 Generating a CRL (Certificate Revocation List)

After a certificate is revoked in the Fabric CA server, the appropriate MSPs in Hyperledger Fabric must also be updated. This includes both local MSPs of the peers as well as MSPs in the appropriate channel configuration blocks. To do this, PEM encoded CRL (certificate revocation list) file must be placed in the *crls* folder of the MSP. The `fabric-ca-client gencrl` command can be used to generate a CRL. Any identity with `hf.GenCRL` attribute can create a CRL that contains serial numbers of all certificates that were revoked during a certain period. The created CRL is stored in the *<msp folder>/crls/crl.pem* file.

The following command will create a CRL containing all the revoked certificates (expired and unexpired) and store the CRL in the *~/msp/crls/crl.pem* file.

```
export FABRIC_CA_CLIENT_HOME=~/clientconfig
fabric-ca-client gencrl -M ~/msp
```

The next command will create a CRL containing all certificates (expired and unexpired) that were revoked after 2017-09-13T16:39:57-08:00 (specified by the *–revokedafter* flag) and before 2017-09-21T16:39:57-08:00 (specified by the *–revokedbefore* flag) and store the CRL in the *~/msp/crls/crl.pem* file.

```
export FABRIC_CA_CLIENT_HOME=~/clientconfig
fabric-ca-client gencrl --caname "" --revokedafter 2017-09-13T16:39:57-08:00 --
↪revokedbefore 2017-09-21T16:39:57-08:00 -M ~/msp
```

The *–caname* flag specifies the name of the CA to which this request is sent. In this example, the gencrl request is sent to the default CA.

The *–revokedafter* and *–revokedbefore* flags specify the lower and upper boundaries of a time period. The generated CRL will contain certificates that were revoked in this time period. The values must be UTC timestamps specified in RFC3339 format. The *–revokedafter* timestamp cannot be greater than the *–revokedbefore* timestamp.

By default, 'Next Update' date of the CRL is set to next day. The *crl.expiry* CA configuration property can be used to specify a custom value.

The gencrl command will also accept *–expireafter* and *–expirebefore* flags that can be used to generate a CRL with revoked certificates that expire during the period specified by these flags. For example, the following command will generate a CRL that contains certificates that were revoked after 2017-09-13T16:39:57-08:00 and before 2017-09-21T16:39:57-08:00, and that expire after 2017-09-13T16:39:57-08:00 and before 2018-09-13T16:39:57-08:00

```
export FABRIC_CA_CLIENT_HOME=~/clientconfig
fabric-ca-client gencrl --caname "" --expireafter 2017-09-13T16:39:57-08:00 --
↪expirebefore 2018-09-13T16:39:57-08:00  --revokedafter 2017-09-13T16:39:57-08:00 --
↪revokedbefore 2017-09-21T16:39:57-08:00 -M ~/msp
```

1.6.4 Enabling TLS

This section describes in more detail how to configure TLS for a Fabric CA client.

The following sections may be configured in the `fabric-ca-client-config.yaml`.

```
tls:
  # Enable TLS (default: false)
  enabled: true
  certfiles:
    - root.pem
  client:
    certfile: tls_client-cert.pem
    keyfile: tls_client-key.pem
```

The **certfiles** option is the set of root certificates trusted by the client. This will typically just be the root Fabric CA server's certificate found in the server's home directory in the **ca-cert.pem** file.

The **client** option is required only if mutual TLS is configured on the server.

1.6.5 Attribute-Based Access Control

Access control decisions can be made by chaincode (and by the Hyperledger Fabric runtime) based upon an identity's attributes. This is called **Attribute-Based Access Control**, or **ABAC** for short.

In order to make this possible, an identity's enrollment certificate (ECert) may contain one or more attribute name and value. The chaincode then extracts an attribute's value to make an access control decision.

For example, suppose that you are developing application *app1* and want a particular chaincode operation to be accessible only by app1 administrators. Your chaincode could verify that the caller's certificate (which was issued by a CA trusted for the channel) contains an attribute named *app1Admin* with a value of *true*. Of course the name of the attribute can be anything and the value need not be a boolean value.

So how do you get an enrollment certificate with an attribute? There are two methods:

1. When you register an identity, you can specify that an enrollment certificate issued for the identity should by default contain an attribute. This behavior can be overridden at enrollment time, but this is useful for establishing default behavior and, assuming registration occurs outside of your application, does not require any application change.

   The following shows how to register *user1* with two attributes: *app1Admin* and *email*. The ":ecert" suffix causes the *appAdmin* attribute to be inserted into user1's enrollment certificate by default, when the user does not explicitly request attributes at enrollment time. The *email* attribute is not added to the enrollment certificate by default.

```
fabric-ca-client register --id.name user1 --id.secret user1pw --id.type client --id.
↪affiliation org1 --id.attrs 'app1Admin=true:ecert,email=user1@gmail.com'
```

2. When you enroll an identity, you may explicitly request that one or more attributes be added to the certificate. For each attribute requested, you may specify whether the attribute is optional or not. If it is not requested optionally and the identity does not possess the attribute, an error will occur.

   The following shows how to enroll *user1* with the *email* attribute, without the *app1Admin* attribute, and optionally with the *phone* attribute (if the user possesses the *phone* attribute).

```
fabric-ca-client enroll -u http://user1:user1pw@localhost:7054 --enrollment.attrs
↪"email,phone:opt"
```

The table below shows the three attributes which are automatically registered for every identity.

Attribute Name	Attribute Value
hf.EnrollmentID	The enrollment ID of the identity
hf.Type	The type of the identity
hf.Affiliation	The affiliation of the identity

To add any of the above attributes **by default** to a certificate, you must explicitly register the attribute with the ":ecert" specification. For example, the following registers identity 'user1' so that the 'hf.Affiliation' attribute will be added to an enrollment certificate if no specific attributes are requested at enrollment time. Note that the value of the affiliation (which is 'org1') must be the same in both the '–id.affiliation' and the '–id.attrs' flags.

```
fabric-ca-client register --id.name user1 --id.secret user1pw --id.type client --id.
↪affiliation org1 --id.attrs 'hf.Affiliation=org1:ecert'
```

For information on the chaincode library API for Attribute-Based Access Control, see https://github.com/hyperledger/fabric-chaincode-go/blob/master/pkg/cid/README.md

1.6.6 Dynamic Server Configuration Update

This section describes how to use fabric-ca-client to dynamically update portions of the fabric-ca-server's configuration without restarting the server.

All commands in this section require that you first be enrolled by executing the *fabric-ca-client enroll* command.

Dynamically updating identities

This section describes how to use fabric-ca-client to dynamically update identities.

An authorization failure will occur if the client identity does not satisfy all of the following:

- The client identity must possess the "hf.Registrar.Roles" attribute with a comma-separated list of values where one of the values equals the type of identity being updated; for example, if the client's identity has the "hf.Registrar.Roles" attribute with a value of "client", the client can update identities of type 'client', but not 'peer'.
- The affiliation of the client's identity must be equal to or a prefix of the affiliation of the identity being updated. For example, a client with an affiliation of "a.b" may update an identity with an affiliation of "a.b.c" but may not update an identity with an affiliation of "a.c". If root affiliation is required for an identity, then the update request should specify a dot (".") for the affiliation and the client must also have root affiliation.

The following shows how to add, modify, and remove an affiliation.

Getting Identity Information

A caller may retrieve information on a identity from the fabric-ca server as long as the caller meets the authorization requirements highlighted in the section above. The following command shows how to get an identity.

```
fabric-ca-client identity list --id user1
```

A caller may also request to retrieve information on all identities that it is authorized to see by issuing the following command.

```
fabric-ca-client identity list
```

Adding an identity

The following adds a new identity for 'user1'. Adding a new identity performs the same action as registering an identity via the 'fabric-ca-client register' command. There are two available methods for adding a new identity. The first method is via the *–json* flag where you describe the identity in a JSON string.

```
fabric-ca-client identity add user1 --json '{"secret": "user1pw", "type": "client",
↪"affiliation": "org1", "max_enrollments": 1, "attrs": [{"name": "hf.Revoker", "value
↪": "true"}]}'
```

The following adds a user with root affiliation. Note that an affiliation name of "." means the root affiliation.

```
fabric-ca-client identity add user1 --json '{"secret": "user1pw", "type": "client",
↪"affiliation": ".", "max_enrollments": 1, "attrs": [{"name": "hf.Revoker", "value":
↪"true"}]}'
```

The second method for adding an identity is to use direct flags. See the example below for adding 'user1'.

```
fabric-ca-client identity add user1 --secret user1pw --type client --affiliation . --
↪maxenrollments 1 --attrs hf.Revoker=true
```

The table below lists all the fields of an identity and whether they are required or optional, and any default values they might have.

Fields	Required	Default Value
ID	Yes	
Secret	No	
Affiliation	No	Caller's Affiliation
Type	No	client
Maxenrollments	No	0
Attributes	No	

Modifying an identity

There are two available methods for modifying an existing identity. The first method is via the *–json* flag where you describe the modifications in to an identity in a JSON string. Multiple modifications can be made in a single request. Any element of an identity that is not modified will retain its original value.

NOTE: A maxenrollments value of "-2" specifies that the CA's max enrollment setting is to be used.

The command below make multiple modification to an identity using the –json flag.

```
fabric-ca-client identity modify user1 --json '{"secret": "newPassword", "affiliation
↪": ".", "attrs": [{"name": "hf.Regisrar.Roles", "value": "peer,client"},{"name":
↪"hf.Revoker", "value": "true"}]}'
```

The commands below make modifications using direct flags. The following updates the enrollment secret (or password) for identity 'user1' to 'newsecret'.

```
fabric-ca-client identity modify user1 --secret newsecret
```

The following updates the affiliation of identity 'user1' to 'org2'.

```
fabric-ca-client identity modify user1 --affiliation org2
```

The following updates the type of identity 'user1' to 'peer'.

```
fabric-ca-client identity modify user1 --type peer
```

The following updates the maxenrollments of identity 'user1' to 5.

```
fabric-ca-client identity modify user1 --maxenrollments 5
```

By specifying a maxenrollments value of '-2', the following causes identity 'user1' to use the CA's max enrollment setting.

```
fabric-ca-client identity modify user1 --maxenrollments -2
```

The following sets the value of the 'hf.Revoker' attribute for identity 'user1' to 'false'. If the identity has other attributes, they are not changed. If the identity did not previously possess the 'hf.Revoker' attribute, the attribute is added to the identity. An attribute may also be removed by specifying no value for the attribute.

```
fabric-ca-client identity modify user1 --attrs hf.Revoker=false
```

The following removes the 'hf.Revoker' attribute for user 'user1'.

```
fabric-ca-client identity modify user1 --attrs hf.Revoker=
```

The following demonstrates that multiple options may be used in a single *fabric-ca-client identity modify* command. In this case, both the secret and the type are updated for user 'user1'.

```
fabric-ca-client identity modify user1 --secret newpass --type peer
```

Removing an identity

The following removes identity 'user1' and also revokes any certificates associated with the 'user1' identity.

```
fabric-ca-client identity remove user1
```

Note: Removal of identities is disabled in the fabric-ca-server by default, but may be enabled by starting the fabric-ca-server with the *–cfg.identities.allowremove* option.

Dynamically updating affiliations

This section describes how to use fabric-ca-client to dynamically update affiliations. The following shows how to add, modify, remove, and list an affiliation.

Adding an affiliation

An authorization failure will occur if the client identity does not satisfy all of the following:

- The client identity must possess the attribute 'hf.AffiliationMgr' with a value of 'true'.
- The affiliation of the client identity must be hierarchically above the affiliation being updated. For example, if the client's affiliation is "a.b", the client may add affiliation "a.b.c" but not "a" or "a.b".

The following adds a new affiliation named 'org1.dept1'.

```
fabric-ca-client affiliation add org1.dept1
```

Modifying an affiliation

An authorization failure will occur if the client identity does not satisfy all of the following:

- The client identity must possess the attribute 'hf.AffiliationMgr' with a value of 'true'.
- The affiliation of the client identity must be hierarchically above the affiliation being updated. For example, if the client's affiliation is "a.b", the client may add affiliation "a.b.c" but not "a" or "a.b".
- If the '–force' option is true and there are identities which must be modified, the client identity must also be authorized to modify the identity.

The following renames the 'org2' affiliation to 'org3'. It also renames any sub affiliations (e.g. 'org2.department1' is renamed to 'org3.department1').

```
fabric-ca-client affiliation modify org2 --name org3
```

If there are identities that are affected by the renaming of an affiliation, it will result in an error unless the '–force' option is used. Using the '–force' option will update the affiliation of identities that are affected to use the new affiliation name.

```
fabric-ca-client affiliation modify org1 --name org2 --force
```

Removing an affiliation

An authorization failure will occur if the client identity does not satisfy all of the following:

- The client identity must possess the attribute 'hf.AffiliationMgr' with a value of 'true'.
- The affiliation of the client identity must be hierarchically above the affiliation being updated. For example, if the client's affiliation is "a.b", the client may remove affiliation "a.b.c" but not "a" or "a.b".
- If the '–force' option is true and there are identities which must be modified, the client identity must also be authorized to modify the identity.

The following removes affiliation 'org2' and also any sub affiliations. For example, if 'org2.dept1' is an affiliation below 'org2', it is also removed.

```
fabric-ca-client affiliation remove org2
```

If there are identities that are affected by the removing of an affiliation, it will result in an error unless the '–force' option is used. Using the '–force' option will also remove all identities that are associated with that affiliation, and the certificates associated with any of these identities.

Note: Removal of affiliations is disabled in the fabric-ca-server by default, but may be enabled by starting the fabric-ca-server with the *–cfg.affiliations.allowremove* option.

Listing affiliation information

An authorization failure will occur if the client identity does not satisfy all of the following:

- The client identity must possess the attribute 'hf.AffiliationMgr' with a value of 'true'.
- Affiliation of the client identity must be equal to or be hierarchically above the affiliation being updated. For example, if the client's affiliation is "a.b", the client may get affiliation information on "a.b" or "a.b.c" but not "a" or "a.c".

The following command shows how to get a specific affiliation.

```
fabric-ca-client affiliation list --affiliation org2.dept1
```

A caller may also request to retrieve information on all affiliations that it is authorized to see by issuing the following command.

```
fabric-ca-client affiliation list
```

1.6.7 Manage Certificates

This section describes how to use fabric-ca-client to manage certificates.

Listing certificate information

The certificates that are visible to a caller include:

- Those certificates which belong to the caller
- If the caller possesses the `hf.Registrar.Roles` attribute or the `hf.Revoker` attribute with a value of `true` , all certificates which belong to identities in and below the caller's affiliation. For example, if the client's affiliation is `a.b` , the client may get certificates for identities who's affiliation is `a.b` or `a.b.c` but not `a` or `a.c` .

If executing a list command that requests certificates of more than one identity, only certificates of identities with an affiliation that is equal to or hierarchically below the caller's affiliation will be listed.

The certificates which will be listed may be filtered based on ID, AKI, serial number, expiration time, revocation time, notrevoked, and notexpired flags.

- `id` : List certificates for this enrollment ID
- `serial` : List certificates that have this serial number
- `aki` : List certificates that have this AKI
- `expiration` : List certificates that have expiration dates that fall within this expiration time
- `revocation` : List certificates that were revoked within this revocation time
- `notrevoked` : List certificates that have not yet been revoked
- `notexpired` : List certificates that have not yet expired

You can use flags `notexpired` and `notrevoked` as filters to exclude revoked certificates and/or expired certificates from the result set. For example, if you only care about certificates that have expired but have not been revoked you can use the `expiration` and `notrevoked` flags to get back such results. An example of this case is provided below.

Time should be specified based on RFC3339. For instance, to list certificates that have expirations between March 1, 2018 at 1:00 PM and June 15, 2018 at 2:00 AM, the input time string would look like 2018-03-01T13:00:00z and 2018-06-15T02:00:00z. If time is not a concern, and only the dates matter, then the time part can be left off and then the strings become 2018-03-01 and 2018-06-15.

The string `now` may be used to denote the current time and the empty string to denote any time. For example, `now::` denotes a time range from now to any time in the future, and `::now` denotes a time range from any time in the past until now.

The following command shows how to list certificates using various filters.

List all certificates:

```
fabric-ca-client certificate list
```

List all certificates by id:

```
fabric-ca-client certificate list --id admin
```

List certificate by serial and aki:

```
fabric-ca-client certificate list --serial 1234 --aki 1234
```

List certificate by id and serial/aki:

```
fabric-ca-client certificate list --id admin --serial 1234 --aki 1234
```

List certificates that are neither revoker nor expired by id:

```
fabric-ca-client certificate list --id admin --notrevoked --notexpired
```

List all certificates that have not been revoked for an id (admin):

```
fabric-ca-client certificate list --id admin --notrevoked
```

List all certificates have not expired for an id (admin):

The "–notexpired" flag is equivalent to "–expiration now::", which means certificates will expire some time in the future.

```
fabric-ca-client certificate list --id admin --notexpired
```

List all certificates that were revoked between a time range for an id (admin):

```
fabric-ca-client certificate list --id admin --revocation 2018-01-01T01:30:00z::2018-
↪01-30T05:00:00z
```

List all certificates that were revoked between a time range but have not expired for an id (admin):

```
fabric-ca-client certificate list --id admin --revocation 2018-01-01::2018-01-30 --
↪notexpired
```

List all revoked certificates using duration (revoked between 30 days and 15 days ago) for an id (admin):

```
fabric-ca-client certificate list --id admin --revocation -30d::-15d
```

List all revoked certificates before a time

```
fabric-ca-client certificate list --revocation ::2018-01-30
```

List all revoked certificates after a time

```
fabric-ca-client certificate list --revocation 2018-01-30::
```

List all revoked certificates before now and after a certain date

```
fabric-ca-client certificate list --id admin --revocation 2018-01-30::now
```

List all certificate that expired between a time range but have not been revoked for an id (admin):

```
fabric-ca-client certificate list --id admin --expiration 2018-01-01::2018-01-30 --
↪notrevoked
```

List all expired certificates using duration (expired between 30 days and 15 days ago) for an id (admin):

```
fabric-ca-client certificate list --expiration -30d::-15d
```

List all certificates that have expired or will expire before a certain time

```
fabric-ca-client certificate list --expiration ::2058-01-30
```

List all certificates that have expired or will expire after a certain time

```
fabric-ca-client certificate list --expiration 2018-01-30::
```

List all expired certificates before now and after a certain date

```
fabric-ca-client certificate list --expiration 2018-01-30::now
```

List certificates expiring in the next 10 days:

```
fabric-ca-client certificate list --id admin --expiration ::+10d --notrevoked
```

The list certificate command can also be used to store certificates on the file system. This is a convenient way to populate the admins folder in an MSP, The "-store" flag points to the location on the file system to store the certificates.

Configure an identity to be an admin, by storing certificates for an identity in the MSP:

```
export FABRIC_CA_CLIENT_HOME=/tmp/clientHome
fabric-ca-client certificate list --id admin --store msp/admincerts
```

1.6.8 Contact specific CA instance

When a server is running multiple CA instances, requests can be directed to a specific CA. By default, if no CA name is specified in the client request the request will be directed to the default CA on the fabric-ca server. A CA name can be specified on the command line of a client command using the `caname` filter as follows:

```
fabric-ca-client enroll -u http://admin:adminpw@localhost:7054 --caname <caname>
```

Back to Top

1.7 Configuring an HSM

By default, the Fabric CA server and client store private keys in a PEM-encoded file, but they can also be configured to store private keys in an HSM (Hardware Security Module) via PKCS11 APIs. This behavior is configured in the BCCSP (BlockChain Crypto Service Provider) section of the server's or client's configuration file. Currently, Fabric only supports the PKCS11 standard to communicate with an HSM.

1.7.1 Example

The following example demonstrates how to configure the Fabric CA server or client to use a software version of PKCS11 called SoftHSM (see https://github.com/opendnssec/SoftHSMv2).

After installing SoftHSM, make sure to set your SOFTHSM2_CONF environment variable to point to the location where the SoftHSM2 configuration file is stored. The config file looks like

```
directories.tokendir = /tmp/
objectstore.backend = file
log.level = INFO
```

Create a token, label it "ForFabric", set the pin to '98765432' (refer to SoftHSM documentation).

You can use both the config file and environment variables to configure BCCSP. For example, set the bccsp section of Fabric CA server configuration file as follows. Note that the default field's value is PKCS11.

```
###############################################################################
# BCCSP (BlockChain Crypto Service Provider) section is used to select which
# crypto library implementation to use
```

```
##############################################################################
bccsp:
  default: PKCS11
  pkcs11:
    Library: /usr/local/Cellar/softhsm/2.1.0/lib/softhsm/libsofthsm2.so
    Pin: 98765432
    Label: ForFabric
    hash: SHA2
    security: 256
    filekeystore:
      # The directory used for the software file-based keystore
      keystore: msp/keystore
```

And you can override relevant fields via environment variables as follows:

```
FABRIC_CA_SERVER_BCCSP_DEFAULT=PKCS11
FABRIC_CA_SERVER_BCCSP_PKCS11_LIBRARY=/usr/local/Cellar/softhsm/2.1.0/lib/softhsm/
↪libsofthsm2.so
FABRIC_CA_SERVER_BCCSP_PKCS11_PIN=98765432
FABRIC_CA_SERVER_BCCSP_PKCS11_LABEL=ForFabric
```

The prebuilt Hyperledger Fabric Docker images are not enabled to use PKCS11. If you are deploying the Fabric CA using Docker, you need to build your own image and enable PKCS11 using the following command:

```
make docker GO_TAGS=pkcs11
```

You also need to ensure that the PKCS11 library is available to be used by the CA by installing it or mounting it inside the container. If you are deploying your Fabric CA using Docker Compose, you can update your Compose files to mount the SoftHSM library and configuration file inside the container using volumes. As an example, you would add the following environment and volumes variables to your Docker Compose file:

```
environment:
   - SOFTHSM2_CONF=/etc/hyperledger/fabric/config.file
volumes:
   - /home/softhsm/config.file:/etc/hyperledger/fabric/config.file
   - /usr/local/Cellar/softhsm/2.1.0/lib/softhsm/libsofthsm2.so:/etc/hyperledger/
↪fabric/libsofthsm2.so
```

Back to Top

1.8 File Formats

1.8.1 Fabric CA server's configuration file format

A default configuration file is created in the server's home directory (see *Fabric CA Server* section for more info). The following link shows a sample Server configuration file.

1.8.2 Fabric CA client's configuration file format

A default configuration file is created in the client's home directory (see *Fabric CA Client* section for more info). The following link shows a sample Client configuration file.

Back to Top

1.9 Troubleshooting

1. If you see a `Killed: 9` error on OSX when trying to execute `fabric-ca-client` or `fabric-ca-server`, there is a long thread describing this problem at https://github.com/golang/go/issues/19734. The short answer is that to work around this issue, you can run the following command:

```
# sudo ln -s /usr/bin/true /usr/local/bin/dsymutil
```

2. The error `[ERROR] No certificates found for provided serial and aki` will occur if the following sequence of events occurs:

   (a) You issue a *fabric-ca-client enroll* command, creating an enrollment certificate (i.e. an ECert). This stores a copy of the ECert in the fabric-ca-server's database.

   (b) The fabric-ca-server's database is deleted and recreated, thus losing the ECert from step 'a'. For example, this may happen if you stop and restart a Docker container hosting the fabric-ca-server, but your fabric-ca-server is using the default sqlite database and the database file is not stored on a volume and is therefore not persistent.

   (c) You issue a *fabric-ca-client register* command or any other command which tries to use the ECert from step 'a'. In this case, since the database no longer contains the ECert, the `[ERROR] No certificates found for provided serial and aki` will occur.

   To resolve this error, you must enroll again by repeating step 'a'. This will issue a new ECert which will be stored in the current database.

3. When sending multiple parallel requests to a Fabric CA Server cluster that uses shared sqlite3 databases, the server occasionally returns a 'database locked' error. This is most probably because the database transaction timed out while waiting for database lock (held by another cluster member) to be released. This is an invalid configuration because sqlite is an embedded database, which means the Fabric CA server cluster must share the same file via a shared file system, which introduces a SPoF (single point of failure), which contradicts the purpose of cluster topology. The best practice is to use either Postgres or MySQL databases in a cluster topology.

4. Suppose an error similar to `Failed to deserialize creator identity,err The supplied identity is not valid,Verify() returned x509: certificate signed by unknown authority` is returned by a peer or orderer when using an enrollment certificate issued by the Fabric CA Server. This indicates that the signing CA certificate used by the Fabric CA Server to issue certificates does not match a certificate in the *cacerts* or *intermediatecerts* folder of the MSP used to make authorization checks.

   The MSP which is used to make authorization checks depends on which operation you were performing when the error occurred. For example, if you were trying to install chaincode on a peer, the local MSP on the file system of the peer is used; otherwise, if you were performing some channel specific operation such as instantiating chaincode on a specific channel, the MSP in the genesis block or the most recent configuration block of the channel is used.

   To confirm that this is the problem, compare the AKI (Authority Key Identifier) of the enrollment certificate to the SKI (Subject Key Identifier) of the certificate(s) in the *cacerts* and *intermediatecerts* folder of appropriate MSP. The command *openssl x509 -in <PEM-file> -noout -text | grep -A1 "Authority Key Identifier"* will display the AKI and *openssl x509 -in <PEM-file> -noout -text | grep -A1 "Subject Key Identifier"* will display the SKI. If they are not equal, you have confirmed that this is the cause of the error.

   This can happen for multiple reasons including:

   (a) You used *cryptogen* to generate your key material but did not start *fabric-ca-server* with the signing key and certificate generated by *cryptogen*.

   To resolve (assuming *FABRIC_CA_SERVER_HOME* is set to the home directory of your *fabric-ca-server*):

   i. Stop *fabric-ca-server*.

ii. Copy *crypto-config/peerOrganizations/<orgName>/ca/*pem* to *$FABRIC_CA_SERVER_HOME/ca-cert.pem*.

iii. Copy *crypto-config/peerOrganizations/<orgName>/ca/*_sk* to *$FABRIC_CA_SERVER_HOME/msp/keystore/*.

iv. Start *fabric-ca-server*.

v. Delete any previously issued enrollment certificates and get new certificates by enrolling again.

(b) You deleted and recreated the CA signing key and certificate used by the Fabric CA Server after generating the genesis block. This can happen if the Fabric CA Server is running in a Docker container, the container was restarted, and its home directory is not on a volume mount. In this case, the Fabric CA Server will create a new CA signing key and certificate.

Assuming that you can not recover the original CA signing key, the only way to recover from this scenario is to update the certificate in the *cacerts* (or *intermediatecerts*) of the appropriate MSPs to the new CA certificate.

CHAPTER 2

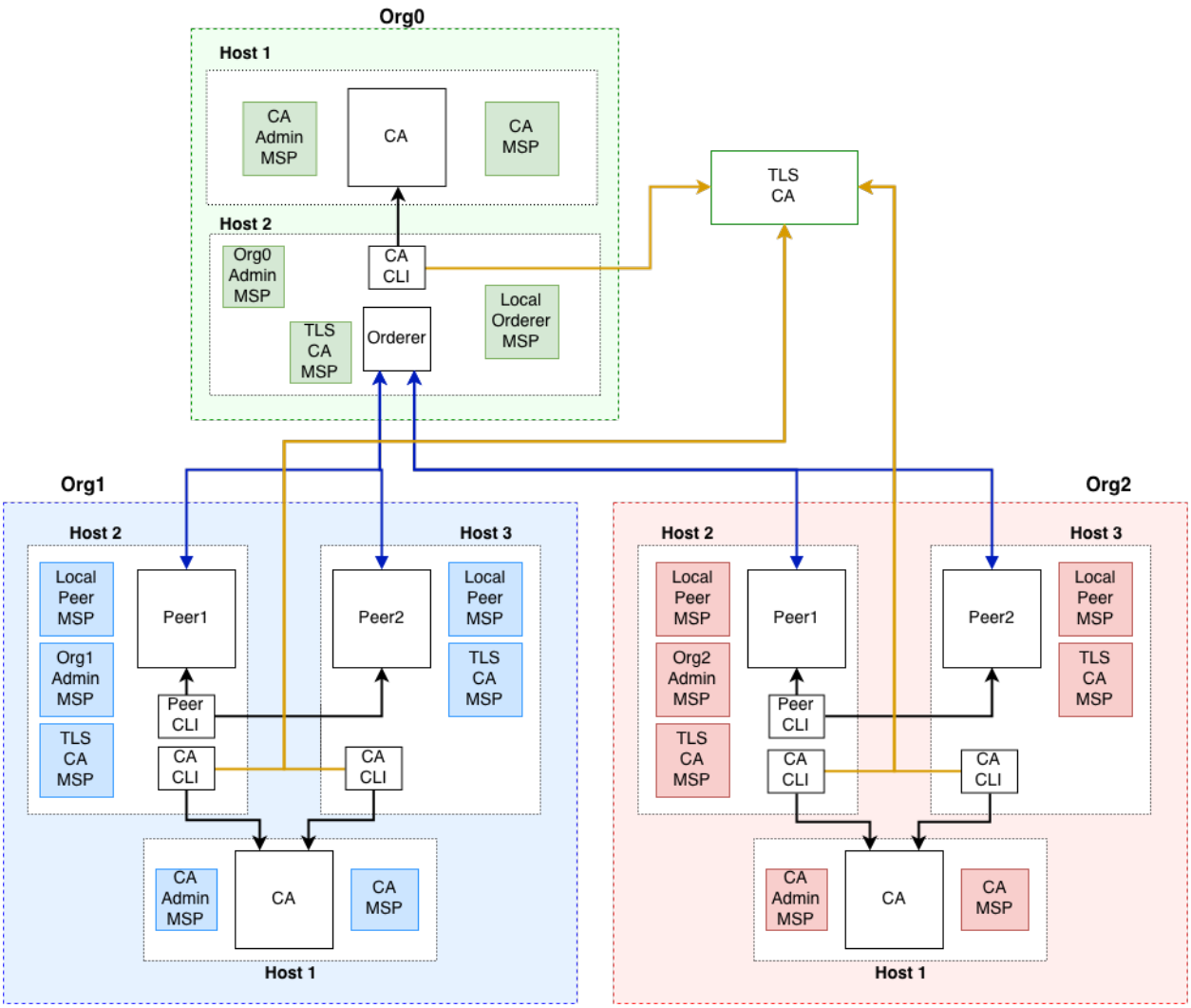
Fabric CA Operations Guide

This guide will illustrate how to use Fabric CA to setup a Fabric network. All identities that participate on a Hyperledger Fabric blockchain network must be authorized. This authorization is provided in the form of cryptographic material that is verified against trusted authorities.

In this guide, you will see the process for setting up a blockchain network that includes two organizations, each with two peers and an orderer. You'll see how to generate cryptographic material for orderers, peers, administrators, and end users so that private keys never leave the host or container where they are generated.

2.1 Topology

In this example, we will look at how to setup up an orderer, peers, and CAs across three organizations. The topology of this deployment can be seen in the image below:



This example will simulate a deployment using docker containers. The containers will be treated as if they are running on different host machines. This is done so that you can see which assets need to be exchanged out-of-band between the parties involved in the network.

The network configuration for docker assumes that all containers are running in the same network. If your deployment is spread across different networks, the example will need to be adjusted to work with your network configurations.

The documentation below breaks down the docker-compose file to talk about individual components. To see the entire docker-compose, click here.

2.2 Setup CAs

2.2.1 Download fabric-ca-client binary

For each host that needs to acquire cryptographic material, you will need to have the fabric-ca-client binary available on the host machine. The client will be used to connect to the Fabric CA server container.

To download the fabric-ca-client binary, browse to this repository and select the latest binary for your machine.

Note: This example is using version 1.4.0 of fabric-ca-client.

2.2.2 Setup TLS CA

A TLS CA is used to issue TLS certificates. These certificates are required in order to secure the communication between various processes.

In order to simplify this example, all organizations will use the same TLS CA and TLS mutual authentication is disabled.

Note: In a production environment, you will probably use your organization's CA to get TLS certificates. You will have to transfer out-of-band your CA's certificate with organizations that will validate your TLS certificates. Thus, unlike this example, each organization would have its own TLS CA.

A docker service, such as the one below can be used to a launch a Fabric TLS CA container.

```
ca-tls:
  container_name: ca-tls
  image: hyperledger/fabric-ca
  command: sh -c 'fabric-ca-server start -d -b tls-ca-admin:tls-ca-adminpw --port 7052
↪'
  environment:
    - FABRIC_CA_SERVER_HOME=/tmp/hyperledger/fabric-ca/crypto
    - FABRIC_CA_SERVER_TLS_ENABLED=true
    - FABRIC_CA_SERVER_CSR_CN=ca-tls
    - FABRIC_CA_SERVER_CSR_HOSTS=0.0.0.0
    - FABRIC_CA_SERVER_DEBUG=true
  volumes:
    - /tmp/hyperledger/tls/ca:/tmp/hyperledger/fabric-ca
  networks:
    - fabric-ca
  ports:
    - 7052:7052
```

This container can be started using the following docker command.

```
docker-compose up ca-tls
```

On a successful launch of the container, you will see the following line in the CA container's log.

```
[INFO] Listening on https://0.0.0.0:7052
```

At this point the TLS CA server is on a listening on a secure socket, and can start issuing TLS certificates.

Enroll TLS CA's Admin

Before you can start using the CA client, you must acquire the signing certificate for the CA's TLS certificate. This is a required step before you can connect using TLS.

In our example, you would need to acquire the file located at `/tmp/hyperledger/tls-ca/crypto/ca-cert.pem` on the machine running the TLS CA server and copy this file over to the host where you will be running the CA client binary. This certificate, also known as the TLS CA's signing certificate is going to be used to validate the TLS

certificate of the CA. Once the certificate has been copied over to the CA client's host machine, you can start issuing commands using the CA.

The TLS CA's signing certificate will need to be available on each host that will run commands against the TLS CA.

The TLS CA server was started with a bootstrap identity which has full admin privileges for the server. One of the key abilities of the admin is the ability to register new identities. The administrator for this CA will use the Fabric CA client to register four new identities with the CA, one for each peer and one for the orderer. These identities will be used to get TLS certificates for peers and orderers.

You will issue the commands below to enroll the TLS CA admin and then register identities. We assume the trusted root certificate for the TLS CA has been copied to `/tmp/hyperledger/tls-ca/crypto/tls-ca-cert.pem` on all host machines that will communicate with this CA via the fabric-ca-client.

```
export FABRIC_CA_CLIENT_TLS_CERTFILES=/tmp/hyperledger/tls-ca/crypto/tls-ca-cert.pem
export FABRIC_CA_CLIENT_HOME=/tmp/hyperledger/tls-ca/admin
fabric-ca-client enroll -d -u https://tls-ca-admin:tls-ca-adminpw@0.0.0.0:7052
fabric-ca-client register -d --id.name peer1-org1 --id.secret peer1PW --id.type peer -
↪u https://0.0.0.0:7052
fabric-ca-client register -d --id.name peer2-org1 --id.secret peer2PW --id.type peer -
↪u https://0.0.0.0:7052
fabric-ca-client register -d --id.name peer1-org2 --id.secret peer1PW --id.type peer -
↪u https://0.0.0.0:7052
fabric-ca-client register -d --id.name peer2-org2 --id.secret peer2PW --id.type peer -
↪u https://0.0.0.0:7052
fabric-ca-client register -d --id.name orderer1-org0 --id.secret ordererPW --id.type␣
↪orderer -u https://0.0.0.0:7052
```

Note: If the path of the environment variable FABRIC_CA_CLIENT_TLS_CERTFILES is not an absolute path, it will be parsed as relative to the client's home directory.

With the identities registered on the TLS CA, we can move forward to setting up the each organization's network. Anytime we need to get TLS certificates for a node in an organization, we will refer to this CA.

Setup Orderer Org CA

Each organization must have its own Certificate Authority (CA) for issuing enrollment certificates. The CA will issue the certificates for each of the peers and clients in the organization.

Your CA creates the identities that belong to your organization and issue each identity a public and private key. These keys are what allow all of your nodes and applications to sign and verify their actions. Any identity signed by your CA will be understood by other members of the network to identify the components that belong to your organization.

An administrator for Org0 will launch a Fabric CA docker container, which will be used by Org0 to issue cryptographic material for identities in Org0.

A docker service such as the one below can be used to a launch a Fabric CA container.

```
rca-org0:
   container_name: rca-org0
   image: hyperledger/fabric-ca
   command: /bin/bash -c 'fabric-ca-server start -d -b rca-org0-admin:rca-org0-
↪adminpw --port 7053'
   environment:
      - FABRIC_CA_SERVER_HOME=/tmp/hyperledger/fabric-ca/crypto
      - FABRIC_CA_SERVER_TLS_ENABLED=true
      - FABRIC_CA_SERVER_CSR_CN=rca-org0
```

```
      - FABRIC_CA_SERVER_CSR_HOSTS=0.0.0.0
      - FABRIC_CA_SERVER_DEBUG=true
   volumes:
      - /tmp/hyperledger/org0/ca:/tmp/hyperledger/fabric-ca
   networks:
      - fabric-ca
   ports:
      - 7053:7053
```

On a successful launch of the container, you will see the following line in the CA container's log.

```
[INFO] Listening on https://0.0.0.0:7053
```

At this point the CA server is listening on a secure socket, and can start issuing cryptographic material.

2.2.3 Enroll Orderer Org's CA Admin

You will issue the commands below to enroll the CA admin and then register both of Org0's identities.

In the commands below, we will assume the trusted root certificate for the CA's TLS certificate has been copied to `/tmp/hyperledger/org0/ca/crypto/ca-cert.pem` on the host machine where the fabric-ca-client binary is present. If the client binary is located on a different host, you will need to get the signing certificate through an out-of-band process.

The following identities will be registered:

- Orderer (orderer1-org0)
- Orderer admin (admin-org0)

```
export FABRIC_CA_CLIENT_TLS_CERTFILES=/tmp/hyperledger/org0/ca/crypto/ca-cert.pem
export FABRIC_CA_CLIENT_HOME=/tmp/hyperledger/org0/ca/admin
fabric-ca-client enroll -d -u https://rca-org0-admin:rca-org0-adminpw@0.0.0.0:7053
fabric-ca-client register -d --id.name orderer1-org0 --id.secret ordererpw --id.type␣
↪orderer -u https://0.0.0.0:7053
fabric-ca-client register -d --id.name admin-org0 --id.secret org0adminpw --id.type␣
↪admin --id.attrs "hf.Registrar.Roles=client,hf.Registrar.Attributes=*,hf.
↪Revoker=true,hf.GenCRL=true,admin=true:ecert,abac.init=true:ecert" -u https://0.0.0.
↪0:7053
```

The enroll command you executed above, would have populated the `/tmp/hyperledger/org0/ca/admin` directory with the cryptographic material issued form the CA. You will see files such as the ones below:

```
admin
-- fabric-ca-client-config.yaml
-- msp
   -- IssuerPublicKey
   -- IssuerRevocationPublicKey
   -- cacerts
   |   -- 0-0-0-0-7053.pem
   -- keystore
   |   -- 60b6a16b8b5ba3fc3113c522cce86a724d7eb92d6c3961cfd9afbd27bf11c37f_sk
   -- signcerts
   |   -- cert.pem
   -- user
```

The `fabric-ca-client-config.yaml` is a file that is generated by the CA client, this file contains the configuration of the CA client. There are three other important files to note. First one is

`0-0-0-0-7053.pem` , this is the public certificate of the CA that issued the certificate for this identity. Second is `60b6a16b8b5ba3fc3113c522cce86a724d7eb92d6c3961cfd9afbd27bf11c37f_sk` , this is the private key that was generated by the client. The name of this file is variable and will be different every time a key is generated. The last item is `cert.pem` , this is the certificate of the admin that was signed and issued by the CA.

Setup Org1's CA

The same set of steps you performed for Org0 apply to Org1's CA.

An administrator for Org1 will launch a Fabric CA docker container, which will be used by Org1 to issue cryptographic material for identities in Org1.

A docker service, such as the one below can be used to a launch a Fabric CA container.

```
rca-org1:
   container_name: rca-org1
   image: hyperledger/fabric-ca
   command: /bin/bash -c 'fabric-ca-server start -d -b rca-org1-admin:rca-org1-adminpw
↪'
   environment:
      - FABRIC_CA_SERVER_HOME=/tmp/hyperledger/fabric-ca/crypto
      - FABRIC_CA_SERVER_TLS_ENABLED=true
      - FABRIC_CA_SERVER_CSR_CN=rca-org1
      - FABRIC_CA_SERVER_CSR_HOSTS=0.0.0.0
      - FABRIC_CA_SERVER_DEBUG=true
   volumes:
      - /tmp/hyperledger/org1/ca:/tmp/hyperledger/fabric-ca
   networks:
      - fabric-ca
   ports:
      - 7054:7054
```

On a successful launch of the container, you will see the following line in the CA container's log.

```
[INFO] Listening on https://0.0.0.0:7054
```

At this point the CA server is listening on a secure socket, and can start issuing cryptographic material.

2.2.4 Enroll Org1's CA Admin

You will issue the commands below to enroll the CA admin and then register both of Org1's identities.

The following identities are being registered:

- Peer 1 (peer1-org1)
- Peer 2 (peer2-org1)
- Admin (admin1-org1)
- End user (user-org1)

In the commands below, we will assume the trusted root certificate for the CA's TLS certificate has been copied to `/tmp/hyperledger/org1/ca/crypto/ca-cert.pem` on the host machine where the fabric-ca-client binary is present. If the client's binary is located on a different host, you will need to get the signing certificate through an out-of-band process.

```
export FABRIC_CA_CLIENT_TLS_CERTFILES=/tmp/hyperledger/org1/ca/crypto/ca-cert.pem
export FABRIC_CA_CLIENT_HOME=/tmp/hyperledger/org1/ca/admin
fabric-ca-client enroll -d -u https://rca-org1-admin:rca-org1-adminpw@0.0.0.0:7054
fabric-ca-client register -d --id.name peer1-org1 --id.secret peer1PW --id.type peer -
↪u https://0.0.0.0:7054
fabric-ca-client register -d --id.name peer2-org1 --id.secret peer2PW --id.type peer -
↪u https://0.0.0.0:7054
fabric-ca-client register -d --id.name admin-org1 --id.secret org1AdminPW --id.type␣
↪user -u https://0.0.0.0:7054
fabric-ca-client register -d --id.name user-org1 --id.secret org1UserPW --id.type␣
↪user -u https://0.0.0.0:7054
```

Setup Org2's CA

The same set of steps that you followed for Org1 apply to Org2. So, we will quickly go through the set of steps that the administrator for Org2 will perform.

A docker service, such as the one below can be used to a launch a Fabric CA for Org2.

```
rca-org2:
  container_name: rca-org2
  image: hyperledger/fabric-ca
  command: /bin/bash -c 'fabric-ca-server start -d -b rca-org2-admin:rca-org2-adminpw␣
↪--port 7055'
  environment:
    - FABRIC_CA_SERVER_HOME=/tmp/hyperledger/fabric-ca/crypto
    - FABRIC_CA_SERVER_TLS_ENABLED=true
    - FABRIC_CA_SERVER_CSR_CN=rca-org2
    - FABRIC_CA_SERVER_CSR_HOSTS=0.0.0.0
    - FABRIC_CA_SERVER_DEBUG=true
  volumes:
    - /tmp/hyperledger/org2/ca:/tmp/hyperledger/fabric-ca
  networks:
    - fabric-ca
  ports:
    - 7055:7055
```

On a successful launch of the container, you will see the following line in the CA container's log.

```
[INFO] Listening on https://0.0.0.0:7055
```

At this point the CA server is listening on a secure socket, and can start issuing cryptographic material.

2.2.5 Enrolling Org2's CA Admin

You will issue the commands below to get the CA admin enrolled and all peer related identities registered. In the commands below, we will assume the trusted root certificate of CA's TLS certificate has been copied to `/tmp/hyperledger/org2/ca/crypto/ca-cert.pem`.

```
export FABRIC_CA_CLIENT_TLS_CERTFILES=/tmp/hyperledger/org2/ca/crypto/ca-cert.pem
export FABRIC_CA_CLIENT_HOME=/tmp/hyperledger/org2/ca/admin
fabric-ca-client enroll -d -u https://rca-org2-admin:rca-org2-adminpw@0.0.0.0:7055
fabric-ca-client register -d --id.name peer1-org2 --id.secret peer1PW --id.type peer -
↪u https://0.0.0.0:7055
fabric-ca-client register -d --id.name peer2-org2 --id.secret peer2PW --id.type peer -
↪u https://0.0.0.0:7055
```

```
fabric-ca-client register -d --id.name admin-org2 --id.secret org2AdminPW --id.type␣
↪user -u https://0.0.0.0:7055
fabric-ca-client register -d --id.name user-org2 --id.secret org2UserPW --id.type␣
↪user -u https://0.0.0.0:7055
```

2.3 Setup Peers

Once the CAs are up and running, we can start enrolling peers.

2.3.1 Setup Org1's Peers

An administrator for Org1 will enroll the peers with its CA and then launch the peer docker containers. Before you can start up a peer, you will need to enroll the peer identities with the CA to get the MSP that the peer will use. This is known as the local peer MSP.

Enroll Peer1

If the host machine running Peer1 does not have the fabric-ca-client binary, refer to the instructions above on to download the binary.

In the command below, we will assume the trusted root certificate of Org1 has been copied to `/tmp/hyperledger/org1/peer1/assets/ca/org1-ca-cert.pem` on Peer1's host machine. Acquiring of the signing certificate is an out of band process.

```
export FABRIC_CA_CLIENT_HOME=/tmp/hyperledger/org1/peer1
export FABRIC_CA_CLIENT_TLS_CERTFILES=/tmp/hyperledger/org1/peer1/assets/ca/org1-ca-
↪cert.pem
export FABRIC_CA_CLIENT_MSPDIR=msp
fabric-ca-client enroll -d -u https://peer1-org1:peer1PW@0.0.0.0:7054
```

Next step is to get the TLS cryptographic material for the peer. This requires another enrollment, but this time you will enroll against the `tls` profile on the TLS CA. You will also need to provide the address of the Peer1's host machine in the enrollment request as the input to the `csr.hosts` flag. In the command below, we will assume the certificate of the TLS CA has been copied to `/tmp/hyperledger/org1/peer1/assets/tls-ca/tls-ca-cert.pem` on Peer1's host machine.

```
export FABRIC_CA_CLIENT_MSPDIR=tls-msp
export FABRIC_CA_CLIENT_TLS_CERTFILES=/tmp/hyperledger/org1/peer1/assets/tls-ca/tls-
↪ca-cert.pem
fabric-ca-client enroll -d -u https://peer1-org1:peer1PW@0.0.0.0:7052 --enrollment.
↪profile tls --csr.hosts peer1-org1
```

Go to path `/tmp/hyperledger/org1/peer1/tls-msp/keystore` and change the name of the key to `key.pem`. This will make it easy to be able to refer to in later steps.

At this point, you will have two MSP directories. One MSP contains peer's enrollment certificate and the other has the peer's TLS certificate. However, there needs to be an additional folder added in the enrollment MSP directory, and this is the `admincerts` folder. This folder will contain certificate(s) for the administrator of Org1. We will talk more about this when we enroll Org1's admin a little further down.

Enroll Peer2

You will perform similar commands for Peer2. In the commands below, we will assume the trusted root certificate of Org1 has been copied to `/tmp/hyperledger/org1/peer2/assets/ca/org1-ca-cert.pem` on Peer2's host machine.

```
export FABRIC_CA_CLIENT_HOME=/tmp/hyperledger/org1/peer2
export FABRIC_CA_CLIENT_TLS_CERTFILES=/tmp/hyperledger/org1/peer2/assets/ca/org1-ca-
↪cert.pem
export FABRIC_CA_CLIENT_MSPDIR=msp
fabric-ca-client enroll -d -u https://peer2-org1:peer2PW@0.0.0.0:7054
```

Next step is to get the TLS cryptographic material for the peer. This requires another enrollment, but this time you will enroll against the `tls` profile on the TLS CA. You will also need to provide the address of the Peer2's host machine in the enrollment request as the input to the `csr.hosts` flag. In the command below, we will assume the certificate of the TLS CA has been copied to `/tmp/hyperledger/org1/peer2/assets/tls-ca/tls-ca-cert.pem` on Peer2's host machine.

```
export FABRIC_CA_CLIENT_MSPDIR=tls-msp
export FABRIC_CA_CLIENT_TLS_CERTFILES=/tmp/hyperledger/org1/peer2/assets/tls-ca/tls-
↪ca-cert.pem
fabric-ca-client enroll -d -u https://peer2-org1:peer2PW@0.0.0.0:7052 --enrollment.
↪profile tls --csr.hosts peer2-org1
```

Go to path `/tmp/hyperledger/org1/peer2/tls-msp/keystore` and change the name of the key to `key.pem`. This will make it easy to be able to refer to in later steps.

At this point, you will have two MSP directories. One MSP contains peer's enrollment certificate and the other has the peer's TLS certificate. You will add the `admincerts` folder to the enrollment MSP once the admin has been enrolled.

Enroll Org1's Admin

At this point, both peers have been enrolled. Now, you will enroll Org1's admin identity. The admin identity is responsible for activities such as installing and instantiating chaincode. The steps below will enroll the admin. In the commands below, we will assume that they are being executed on Peer1's host machine.

```
export FABRIC_CA_CLIENT_HOME=/tmp/hyperledger/org1/admin
export FABRIC_CA_CLIENT_TLS_CERTFILES=/tmp/hyperledger/org1/peer1/assets/ca/org1-ca-
↪cert.pem
export FABRIC_CA_CLIENT_MSPDIR=msp
fabric-ca-client enroll -d -u https://admin-org1:org1AdminPW@0.0.0.0:7054
```

After enrollment, you should have an admin MSP. You will copy the certificate from this MSP and move it to the Peer1's MSP in the `admincerts` folder. You will need to disseminate this admin certificate to other peers in the org, and it will need to go in to the `admincerts` folder of each peers' MSP.

The command below is only for Peer1, the exchange of the admin certificate to Peer2 will happen out-of-band.

```
mkdir /tmp/hyperledger/org1/peer1/msp/admincerts
cp /tmp/hyperledger/org1/admin/msp/signcerts/cert.pem /tmp/hyperledger/org1/peer1/msp/
↪admincerts/org1-admin-cert.pem
```

If the `admincerts` folder is missing from the peer's local MSP, the peer will fail to start up.

Launch Org1's Peers

Once we have enrolled all the peers and org admin, we have the necessary MSPs to start the peers.

A docker service, such as the one below can be used to a launch a container for Peer1.

```
peer1-org1:
  container_name: peer1-org1
  image: hyperledger/fabric-peer
  environment:
    - CORE_PEER_ID=peer1-org1
    - CORE_PEER_ADDRESS=peer1-org1:7051
    - CORE_PEER_LOCALMSPID=org1MSP
    - CORE_PEER_MSPCONFIGPATH=/tmp/hyperledger/org1/peer1/msp
    - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
    - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=guide_fabric-ca
    - FABRIC_LOGGING_SPEC=debug
    - CORE_PEER_TLS_ENABLED=true
    - CORE_PEER_TLS_CERT_FILE=/tmp/hyperledger/org1/peer1/tls-msp/signcerts/cert.pem
    - CORE_PEER_TLS_KEY_FILE=/tmp/hyperledger/org1/peer1/tls-msp/keystore/key.pem
    - CORE_PEER_TLS_ROOTCERT_FILE=/tmp/hyperledger/org1/peer1/tls-msp/tlscacerts/tls-
↪0-0-0-0-7052.pem
    - CORE_PEER_GOSSIP_USELEADERELECTION=true
    - CORE_PEER_GOSSIP_ORGLEADER=false
    - CORE_PEER_GOSSIP_EXTERNALENDPOINT=peer1-org1:7051
    - CORE_PEER_GOSSIP_SKIPHANDSHAKE=true
  working_dir: /opt/gopath/src/github.com/hyperledger/fabric/org1/peer1
  volumes:
    - /var/run:/host/var/run
    - /tmp/hyperledger/org1/peer1:/tmp/hyperledger/org1/peer1
  networks:
    - fabric-ca
```

Launching the peer service will bring up a peer container, and in the logs you will see the following line:

```
serve -> INFO 020 Started peer with ID=[name:"peer1-org1" ], network ID=[dev],␣
↪address=[peer1-org1:7051]
```

A docker service, such as the one below can be used to a launch a container for Peer2.

```
peer2-org1:
  container_name: peer2-org1
  image: hyperledger/fabric-peer
  environment:
    - CORE_PEER_ID=peer2-org1
    - CORE_PEER_ADDRESS=peer2-org1:7051
    - CORE_PEER_LOCALMSPID=org1MSP
    - CORE_PEER_MSPCONFIGPATH=/tmp/hyperledger/org1/peer2/msp
    - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
    - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=guide_fabric-ca
    - FABRIC_LOGGING_SPEC=grpc=debug:info
    - CORE_PEER_TLS_ENABLED=true
    - CORE_PEER_TLS_CERT_FILE=/tmp/hyperledger/org1/peer2/tls-msp/signcerts/cert.pem
    - CORE_PEER_TLS_KEY_FILE=/tmp/hyperledger/org1/peer2/tls-msp/keystore/key.pem
    - CORE_PEER_TLS_ROOTCERT_FILE=/tmp/hyperledger/org1/peer2/tls-msp/tlscacerts/tls-
↪0-0-0-0-7052.pem
    - CORE_PEER_GOSSIP_USELEADERELECTION=true
    - CORE_PEER_GOSSIP_ORGLEADER=false
    - CORE_PEER_GOSSIP_EXTERNALENDPOINT=peer2-org1:7051
```

```
    - CORE_PEER_GOSSIP_SKIPHANDSHAKE=true
    - CORE_PEER_GOSSIP_BOOTSTRAP=peer1-org1:7051
  working_dir: /opt/gopath/src/github.com/hyperledger/fabric/org1/peer2
  volumes:
    - /var/run:/host/var/run
    - /tmp/hyperledger/org1/peer2:/tmp/hyperledger/org1/peer2
  networks:
    - fabric-ca
```

Launching the peer service will bring up a peer container, and in the logs you will see the following line:

```
serve -> INFO 020 Started peer with ID=[name:"peer2-org1" ], network ID=[dev],␣
↪address=[peer2-org1:7051]
```

2.3.2 Setup Org2's Peers

An administrator for Org2 will use the CA's bootstrap identity to enroll the peers with the CA and then launch the peer docker containers.

Enroll Peer1

You will issue the commands below to enroll Peer1. In the commands below, we will assume the trusted root certificate of Org2 is available at `/tmp/hyperledger/org2/peer1/assets/ca/org2-ca-cert.pem` on Peer1's host machine.

```
export FABRIC_CA_CLIENT_HOME=/tmp/hyperledger/org2/peer1
export FABRIC_CA_CLIENT_TLS_CERTFILES=/tmp/hyperledger/org2/peer1/assets/ca/org2-ca-
↪cert.pem
export FABRIC_CA_CLIENT_MSPDIR=msp
fabric-ca-client enroll -d -u https://peer1-org2:peer1PW@0.0.0.0:7055
```

Next, you will get the TLS certificate. In the command below, we will assume the certificate of the TLS CA has been copied to `/tmp/hyperledger/org2/peer1/assets/tls-ca/tls-ca-cert.pem` on Peer1's host machine.

```
export FABRIC_CA_CLIENT_MSPDIR=tls-msp
export FABRIC_CA_CLIENT_TLS_CERTFILES=/tmp/hyperledger/org2/peer1/assets/tls-ca/tls-
↪ca-cert.pem
fabric-ca-client enroll -d -u https://peer1-org2:peer1PW@0.0.0.0:7052 --enrollment.
↪profile tls --csr.hosts peer1-org2
```

Go to path `/tmp/hyperledger/org2/peer1/tls-msp/keystore` and change the name of the key to `key.pem`.

Enroll Peer2

You will issue the commands below to get Peer2 enrolled. In the commands below, we will assume the trusted root certificate of Org2 is available at `/tmp/hyperledger/org2/peer2/tls/org2-ca-cert.pem` on Peer2's host machine.

```
export FABRIC_CA_CLIENT_HOME=/tmp/hyperledger/org2/peer2
export FABRIC_CA_CLIENT_TLS_CERTFILES=/tmp/hyperledger/org2/peer2/assets/ca/org2-ca-
↪cert.pem
```

```
export FABRIC_CA_CLIENT_MSPDIR=msp
fabric-ca-client enroll -d -u https://peer2-org2:peer2PW@0.0.0.0:7055
```

Next, you will get the TLS certificate. In the command below, we will assume the certificate of the TLS CA has been copied to `/tmp/hyperledger/org2/peer2/assets/tls-ca/tls-ca-cert.pem` on Peer2's host machine.

```
export FABRIC_CA_CLIENT_MSPDIR=tls-msp
export FABRIC_CA_CLIENT_TLS_CERTFILES=/tmp/hyperledger/org2/peer2/assets/tls-ca/tls-
↪ca-cert.pem
fabric-ca-client enroll -d -u https://peer2-org2:peer2PW@0.0.0.0:7052 --enrollment.
↪profile tls --csr.hosts peer2-org2
```

Go to path `/tmp/hyperledger/org2/peer2/tls-msp/keystore` and change the name of the key to `key.pem`.

Enroll Org2's Admin

At this point, you will have two MSP directories. One MSP contains your enrollment certificate and the other has your TLS certificate. However, there needs be one additional folder added in the enrollment MSP directory, and this is the `admincerts` folder. This folder will contain certificates for the administrator of Org2. The steps below will enroll the admin. In the commands below, we will assume that they are being executed on Peer1's host machine.

```
export FABRIC_CA_CLIENT_HOME=/tmp/hyperledger/org2/admin
export FABRIC_CA_CLIENT_TLS_CERTFILES=/tmp/hyperledger/org2/peer1/assets/ca/org2-ca-
↪cert.pem
export FABRIC_CA_CLIENT_MSPDIR=msp
fabric-ca-client enroll -d -u https://admin-org2:org2AdminPW@0.0.0.0:7055
```

After enrollment, you should have an admin MSP. You will copy the certificate from this MSP and move it to the peer MSP under the `admincerts` folder. The commands below are only for Peer1, the exchange of admin cert to peer2 will happen out-of-band.

```
mkdir /tmp/hyperledger/org2/peer1/msp/admincerts
cp /tmp/hyperledger/org2/admin/msp/signcerts/cert.pem /tmp/hyperledger/org2/peer1/msp/
↪admincerts/org2-admin-cert.pem
```

If the `admincerts` folder is missing from the peer's local MSP, the peer will fail to start up.

Launch Org2's Peers

Once we have enrolled all the peers and admin, we have the necessary MSPs to start the peers.

A docker service, such as the one below can be used to a launch a container for the peer1.

```
peer1-org2:
  container_name: peer1-org2
  image: hyperledger/fabric-peer
  environment:
    - CORE_PEER_ID=peer1-org2
    - CORE_PEER_ADDRESS=peer1-org2:7051
    - CORE_PEER_LOCALMSPID=org2MSP
    - CORE_PEER_MSPCONFIGPATH=/tmp/hyperledger/org2/peer1/msp
    - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
    - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=guide_fabric-ca
```

```
    - FABRIC_LOGGING_SPEC=debug
    - CORE_PEER_TLS_ENABLED=true
    - CORE_PEER_TLS_CERT_FILE=/tmp/hyperledger/org2/peer1/tls-msp/signcerts/cert.pem
    - CORE_PEER_TLS_KEY_FILE=/tmp/hyperledger/org2/peer1/tls-msp/keystore/key.pem
    - CORE_PEER_TLS_ROOTCERT_FILE=/tmp/hyperledger/org2/peer1/tls-msp/tlscacerts/tls-
↪0-0-0-0-7052.pem
    - CORE_PEER_GOSSIP_USELEADERELECTION=true
    - CORE_PEER_GOSSIP_ORGLEADER=false
    - CORE_PEER_GOSSIP_EXTERNALENDPOINT=peer1-org2:7051
    - CORE_PEER_GOSSIP_SKIPHANDSHAKE=true
  working_dir: /opt/gopath/src/github.com/hyperledger/fabric/org2/peer1
  volumes:
    - /var/run:/host/var/run
    - /tmp/hyperledger/org2/peer1:/tmp/hyperledger/org2/peer1
  networks:
    - fabric-ca
```

Launching the peer service will bring up a peer container, and in the logs you will see the following line:

```
serve -> INFO 020 Started peer with ID=[name:"peer1-org2" ], network ID=[dev],␣
↪address=[peer1-org2:7051]
```

A docker service, such as the one below can be used to a launch a container for the peer1.

```
peer2-org2:
  container_name: peer2-org2
  image: hyperledger/fabric-peer
  environment:
    - CORE_PEER_ID=peer2-org2
    - CORE_PEER_ADDRESS=peer2-org2:7051
    - CORE_PEER_LOCALMSPID=org2MSP
    - CORE_PEER_MSPCONFIGPATH=/tmp/hyperledger/org2/peer2/msp
    - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
    - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=guide_fabric-ca
    - FABRIC_LOGGING_SPEC=debug
    - CORE_PEER_TLS_ENABLED=true
    - CORE_PEER_TLS_CERT_FILE=/tmp/hyperledger/org2/peer2/tls-msp/signcerts/cert.pem
    - CORE_PEER_TLS_KEY_FILE=/tmp/hyperledger/org2/peer2/tls-msp/keystore/key.pem
    - CORE_PEER_TLS_ROOTCERT_FILE=/tmp/hyperledger/org2/peer2/tls-msp/tlscacerts/tls-
↪0-0-0-0-7052.pem
    - CORE_PEER_GOSSIP_USELEADERELECTION=true
    - CORE_PEER_GOSSIP_ORGLEADER=false
    - CORE_PEER_GOSSIP_EXTERNALENDPOINT=peer2-org2:7051
    - CORE_PEER_GOSSIP_SKIPHANDSHAKE=true
    - CORE_PEER_GOSSIP_BOOTSTRAP=peer1-org2:7051
  working_dir: /opt/gopath/src/github.com/hyperledger/fabric/org2/peer2
  volumes:
    - /var/run:/host/var/run
    - /tmp/hyperledger/org2/peer2:/tmp/hyperledger/org2/peer2
  networks:
    - fabric-ca
```

Launching the peer service will bring up a peer container, and in the logs you will see the following line:

```
serve -> INFO 020 Started peer with ID=[name:"peer2-org2" ], network ID=[dev],␣
↪address=[peer2-org2:7052]
```

2.4 Setup Orderer

The last thing we need to setup is the orderer. We need to take a couple of actions before we can start up the orderer.

2.4.1 Enroll Orderer

Before starting the orderer, you will need to enroll the orderer's identity with a CA to get the MSP that the orderer will use. This is known as the local orderer MSP.

If the host machine does not have the fabric-ca-client binary, please refer to the instructions above on to download the binary.

You will issue the commands below to get the orderer enrolled. In the commands below, we will assume the trusted root certificates for Org0 is available in `/tmp/hyperledger/org0/orderer/assets/ca/org0-ca-cert.pem` on the orderer's host machine.

```
export FABRIC_CA_CLIENT_HOME=/tmp/hyperledger/org0/orderer
export FABRIC_CA_CLIENT_TLS_CERTFILES=/tmp/hyperledger/org0/orderer/assets/ca/org0-ca-
↪cert.pem
fabric-ca-client enroll -d -u https://orderer1-org0:ordererpw@0.0.0.0:7053
```

Next, you will get the TLS certificate. In the command below, we will assume the certificate of the TLS CA has been copied to `/tmp/hyperledger/org0/orderer/assets/tls-ca/tls-ca-cert.pem` on Orderer's host machine.

```
export FABRIC_CA_CLIENT_MSPDIR=tls-msp
export FABRIC_CA_CLIENT_TLS_CERTFILES=/tmp/hyperledger/org0/orderer/assets/tls-ca/tls-
↪ca-cert.pem
fabric-ca-client enroll -d -u https://orderer1-org0:ordererPW@0.0.0.0:7052 --
↪enrollment.profile tls --csr.hosts orderer1-org0
```

Go to path `/tmp/hyperledger/org0/orderer/tls-msp/keystore` and change the name of the key to `key.pem`.

At this point, you will have two MSP directories. One MSP contains your enrollment certificate and the other has your TLS certificate. However, there needs be one additional folder added in the enrollment MSP directory, this is the `admincerts` folder. This folder will contain certificates for the administrator of peer 1. Now, you will enroll the Org0's admin identity by issuing the commands below.

2.4.2 Enroll Org0's Admin

The command below assumes that this is being executed on the orderer's host machine.

```
export FABRIC_CA_CLIENT_HOME=/tmp/hyperledger/org0/admin
export FABRIC_CA_CLIENT_TLS_CERTFILES=/tmp/hyperledger/org0/orderer/assets/ca/org0-ca-
↪cert.pem
export FABRIC_CA_CLIENT_MSPDIR=msp
fabric-ca-client enroll -d -u https://admin-org0:org0adminpw@0.0.0.0:7053
```

After enrollment, you should have an msp folder at `/tmp/hyperledger/org0/admin`. You will copy the certificate from this MSP and move it to the orderer's MSP under the `admincerts` folder.

```
mkdir /tmp/hyperledger/org0/orderer/msp/admincerts
cp /tmp/hyperledger/org0/admin/msp/signcerts/cert.pem /tmp/hyperledger/org0/orderer/
↪msp/admincerts/orderer-admin-cert.pem
```

2.4.3 Create Genesis Block and Channel Transaction

The orderer requires a genesis block that it uses to bootstrap itself. You can find more information in the Hyperledger Fabric documentation

In documentation below, you'll find a snippet of `configtx.yaml` that is written for this specific deployment. For the full `configtx.yaml` , click here.

On the orderer's host machine, we need to collect the MSPs for all the organizations. The `organization` section in the `configtx.yaml` looks like:

```
Organizations:

- &org0

   Name: org0

   ID: org0MSP

   MSPDir: /tmp/hyperledger/org0/msp

- &org1

   Name: org1

   ID: org1MSP

   MSPDir: /tmp/hyperledger/org1/msp

   AnchorPeers:
      - Host: peer1-org1
        Port: 7051

- &org2

   Name: org2

   ID: org2MSP

   MSPDir: /tmp/hyperledger/org2/msp

   AnchorPeers:
      - Host: peer1-org2
        Port: 7051
```

The MSP for Org0 will contain the trusted root certificate of Org0, the certificate of the Org0's admin identity, and the trusted root certificate of the TLS CA. The MSP folder structure can be seen below.

```
/tmp/hyperledger/org0/msp
-- admincerts
|   -- admin-org0-cert.pem
-- cacerts
|   -- org0-ca-cert.pem
-- tlscacerts
|   -- tls-ca-cert.pem
-- users
```

The pattern is the same for all organization. The MSP folder structure for Org1 would like:

```
/tmp/hyperledger/org1/msp
-- admincerts
|   -- admin-org1-cert.pem
-- cacerts
|   -- org1-ca-cert.pem
-- tlscacerts
|   -- tls-ca-cert.pem
-- users
```

The MSP folder structure for Org2 would like:

```
/tmp/hyperledger/org2/msp
-- admincerts
|   -- admin-org2-cert.pem
-- cacerts
|   -- org2-ca-cert.pem
-- tlscacerts
|   -- tls-ca-cert.pem
-- users
```

Once all these MSPs are present on the orderer's host machine you will execute the following commands from the directory in which `configtx.yaml` is present:

```
configtxgen -profile OrgsOrdererGenesis -outputBlock /tmp/hyperledger/org0/orderer/
↪genesis.block -channelID syschannel
configtxgen -profile OrgsChannel -outputCreateChannelTx /tmp/hyperledger/org0/orderer/
↪channel.tx -channelID mychannel
```

This will generate two artifacts, `genesis.block` and `channel.tx` , which will be used in later steps.

Commands for gathering certificates

The Fabric CA client has a couple commands that are useful in acquiring the certificates for the orderer genesis and peer MSP setup.

The first command is the *fabric-ca-client certificate* command. This command can be used to get certificates for the admincerts folder. For more information on how to use this command , please refer to: listing certificate information

The second command is the *fabric-ca-client getcainfo* command. This command can be used to gather certificates for the *cacerts* and *tlscacerts* folders. The *getcainfo* command returns back the certificate of the CA.

2.4.4 Mutual TLS

Endpoints can be secured using Mutual TLS as well. If the CA, Peer, or Orderer are using mutual TLS then the client must also present a TLS certificate that will be verified by the server.

Mutual TLS requires the client to acquire a TLS certificate that it will present to the server. Acquiring a TLS certificate can be done via a TLS certificate authority that does have mutual TLS enabled. Once the client has acquired a TLS certificate, then it can start communication with mutual TLS enabled servers as long as the trusted TLS authority on the server is the same as issuing authority for the client's TLS certificate.

2.4.5 Launch Orderer

Once you have created the genesis block and the channel transaction, you can define an orderer service that points to the genesis.block created above.

```
orderer1-org0:
  container_name: orderer1-org0
  image: hyperledger/fabric-orderer
  environment:
    - ORDERER_HOME=/tmp/hyperledger/orderer
    - ORDERER_HOST=orderer1-org0
    - ORDERER_GENERAL_LISTENADDRESS=0.0.0.0
    - ORDERER_GENERAL_GENESISMETHOD=file
    - ORDERER_GENERAL_GENESISFILE=/tmp/hyperledger/org0/orderer/genesis.block
    - ORDERER_GENERAL_LOCALMSPID=org0MSP
    - ORDERER_GENERAL_LOCALMSPDIR=/tmp/hyperledger/org0/orderer/msp
    - ORDERER_GENERAL_TLS_ENABLED=true
    - ORDERER_GENERAL_TLS_CERTIFICATE=/tmp/hyperledger/org0/orderer/tls-msp/signcerts/
↪cert.pem
    - ORDERER_GENERAL_TLS_PRIVATEKEY=/tmp/hyperledger/org0/orderer/tls-msp/keystore/
↪key.pem
    - ORDERER_GENERAL_TLS_ROOTCAS=[/tmp/hyperledger/org0/orderer/tls-msp/tlscacerts/
↪tls-0-0-0-0-7052.pem]
    - ORDERER_GENERAL_LOGLEVEL=debug
    - ORDERER_DEBUG_BROADCASTTRACEDIR=data/logs
  volumes:
    - /tmp/hyperledger/org0/orderer:/tmp/hyperledger/org0/orderer/
  networks:
    - fabric-ca
```

Launching the orderer service will bring up an orderer container, and in the logs you will see the following line:

```
UTC [orderer/common/server] Start -> INFO 0b8 Beginning to serve requests
```

2.5 Create CLI Containers

Communication with peers requires a CLI container, the container contains the appropriate binaries that will allow you to issue peer related commands. You will create a CLI container for each org. In this example, we launch a CLI container in the same host machine as Peer1 for each org.

2.5.1 Launch Org1's CLI

```
cli-org1:
   container_name: cli-org1
   image: hyperledger/fabric-tools
   tty: true
   stdin_open: true
   environment:
     - GOPATH=/opt/gopath
     - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
     - FABRIC_LOGGING_SPEC=DEBUG
     - CORE_PEER_ID=cli-org1
     - CORE_PEER_ADDRESS=peer1-org1:7051
     - CORE_PEER_LOCALMSPID=org1MSP
```

```
    - CORE_PEER_TLS_ENABLED=true
    - CORE_PEER_TLS_ROOTCERT_FILE=/tmp/hyperledger/org1/peer1/tls-msp/tlscacerts/tls-
↪0-0-0-0-7052.pem
    - CORE_PEER_MSPCONFIGPATH=/tmp/hyperledger/org1/peer1/msp
  working_dir: /opt/gopath/src/github.com/hyperledger/fabric/org1
  command: sh
  volumes:
    - /tmp/hyperledger/org1/peer1:/tmp/hyperledger/org1/peer1
    - /tmp/hyperledger/org1/peer1/assets/chaincode:/opt/gopath/src/github.com/
↪hyperledger/fabric-samples/chaincode
    - /tmp/hyperledger/org1/admin:/tmp/hyperledger/org1/admin
  networks:
    - fabric-ca
```

2.5.2 Launch Org2's CLI

```
cli-org2:
  container_name: cli-org2
  image: hyperledger/fabric-tools
  tty: true
  stdin_open: true
  environment:
    - GOPATH=/opt/gopath
    - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
    - FABRIC_LOGGING_SPEC=DEBUG
    - CORE_PEER_ID=cli-org2
    - CORE_PEER_ADDRESS=peer1-org2:7051
    - CORE_PEER_LOCALMSPID=org2MSP
    - CORE_PEER_TLS_ENABLED=true
    - CORE_PEER_TLS_ROOTCERT_FILE=/tmp/hyperledger/org2/peer1/tls-msp/tlscacerts/tls-
↪0-0-0-0-7052.pem
    - CORE_PEER_MSPCONFIGPATH=/tmp/hyperledger/org2/peer1/msp
  working_dir: /opt/gopath/src/github.com/hyperledger/fabric/org2
  command: sh
  volumes:
    - /tmp/hyperledger/org2/peer1:/tmp/hyperledger/org2/peer1
    - /tmp/hyperledger/org1/peer1/assets/chaincode:/opt/gopath/src/github.com/
↪hyperledger/fabric-samples/chaincode
    - /tmp/hyperledger/org2/admin:/tmp/hyperledger/org2/admin
  networks:
    - fabric-ca
```

2.6 Create and Join Channel

2.6.1 Org1

With the CLI containers up and running, you can now issue commands to create and join a channel. We are going to use Peer1 to create the channel. In the host machine of Peer1, you will execute:

```
docker exec -it cli-org1 sh
```

This command will bring you inside the CLI container and open up a terminal. From here, you will execute the following commands using the admin MSP:

```
export CORE_PEER_MSPCONFIGPATH=/tmp/hyperledger/org1/admin/msp
peer channel create -c mychannel -f /tmp/hyperledger/org1/peer1/assets/channel.tx -o␣
↪orderer1-org0:7050 --outputBlock /tmp/hyperledger/org1/peer1/assets/mychannel.block␣
↪--tls --cafile /tmp/hyperledger/org1/peer1/tls-msp/tlscacerts/tls-0-0-0-0-7052.pem
```

The `channel.tx` is an artifact that was generated by running the `configtxgen` command on the orderer. This artifact needs to be transferred to Peer1's host machine out-of-band from the orderer. The command above will generate `mychannel.block` on Peer1 at the specified output path `/tmp/hyperledger/org1/peer1/assets/mychannel.block`, which will be used by all peers in the network that wish to join the channel. This `mychannel.block` will be need to transferred to all peers in both Org1 and Org2 out-of-band.

The next commands you are going to run is to have Peer1 and Peer2 in join the channel.

```
export CORE_PEER_MSPCONFIGPATH=/tmp/hyperledger/org1/admin/msp
export CORE_PEER_ADDRESS=peer1-org1:7051
peer channel join -b /tmp/hyperledger/org1/peer1/assets/mychannel.block

export CORE_PEER_ADDRESS=peer2-org1:7051
peer channel join -b /tmp/hyperledger/org1/peer1/assets/mychannel.block
```

2.6.2 Org2

Run the following command to enter the CLI docker container.

```
docker exec -it cli-org2 sh
```

In Org2, you only need to have the peers join the channel. Peers in Org2 do not need to create the channel, this was already done by Org1. From inside the Org2 CLI container, you will execute the following commands using the admin MSP:

```
export CORE_PEER_MSPCONFIGPATH=/tmp/hyperledger/org2/admin/msp
export CORE_PEER_ADDRESS=peer1-org2:7051
peer channel join -b /tmp/hyperledger/org2/peer1/assets/mychannel.block

export CORE_PEER_ADDRESS=peer2-org2:7051
peer channel join -b /tmp/hyperledger/org2/peer1/assets/mychannel.block
```

2.7 Install and Instantiate Chaincode

Download this chaincode from Github to the local file system on Peer1 in both orgs.

2.7.1 Org1

On Peer1, you are going to install chaincode. The command assumes that the chaincode that needs to be installed is available inside the GOPATH. In this example we will assume the chaincode is located at `/opt/gopath/src/github.com/hyperledger/fabric-samples/chaincode/abac/go` with the GOPATH being `/opt/gopath`. From Org1's CLI container, you will execute the following command:

```
export CORE_PEER_ADDRESS=peer1-org1:7051
export CORE_PEER_MSPCONFIGPATH=/tmp/hyperledger/org1/admin/msp
peer chaincode install -n mycc -v 1.0 -p github.com/hyperledger/fabric-samples/
↪chaincode/abac/go
```

The same set of steps will be followed for peer2.

```
export CORE_PEER_ADDRESS=peer2-org1:7051
export CORE_PEER_MSPCONFIGPATH=/tmp/hyperledger/org1/admin/msp
peer chaincode install -n mycc -v 1.0 -p github.com/hyperledger/fabric-samples/
↪chaincode/abac/go
```

2.7.2 Org2

On Peer1, you are going to perform the same steps as Org1. The command assumes that the chaincode that needs to be installed is available at `/opt/gopath/src/github.com/hyperledger/org2/peer1/assets/chaincode/abac/go` . From Org2's CLI container, you will execute the following command:

```
export CORE_PEER_ADDRESS=peer1-org2:7051
export CORE_PEER_MSPCONFIGPATH=/tmp/hyperledger/org2/admin/msp
peer chaincode install -n mycc -v 1.0 -p github.com/hyperledger/fabric-samples/
↪chaincode/abac/go
```

The same set of steps will be followed for peer2.

```
export CORE_PEER_ADDRESS=peer2-org2:7051
export CORE_PEER_MSPCONFIGPATH=/tmp/hyperledger/org2/admin/msp
peer chaincode install -n mycc -v 1.0 -p github.com/hyperledger/fabric-samples/
↪chaincode/abac/go
```

The next step is going to be to instantiate the chaincode. This done by executing:

```
peer chaincode instantiate -C mychannel -n mycc -v 1.0 -c '{"Args":["init","a","100",
↪"b","200"]}' -o orderer1-org0:7050 --tls --cafile /tmp/hyperledger/org2/peer1/tls-
↪msp/tlscacerts/tls-0-0-0-0-7052.pem
```

2.8 Invoke and Query Chaincode

From Org1's CLI container, execute:

```
export CORE_PEER_ADDRESS=peer1-org1:7051
export CORE_PEER_MSPCONFIGPATH=/tmp/hyperledger/org1/admin/msp
peer chaincode query -C mychannel -n mycc -c '{"Args":["query","a"]}'
```

This should return a value of `100` .

From Org2's CLI container, execute:

```
export CORE_PEER_ADDRESS=peer1-org2:7051
export CORE_PEER_MSPCONFIGPATH=/tmp/hyperledger/org2/admin/msp
peer chaincode invoke -C mychannel -n mycc -c '{"Args":["invoke","a","b","10"]}' --
↪tls --cafile /tmp/hyperledger/org2/peer1/tls-msp/tlscacerts/tls-0-0-0-0-7052.pem
```

This is going to subtract 10 from value of `a` and move it to `b` . Now, if you query by running:

```
peer chaincode query -C mychannel -n mycc -c '{"Args":["query","a"]}'
```

This should return a value of `90` .

This concludes the Operations Guide for Fabric CA.

CHAPTER 3

Fabric CA Deployment Guide

This guide will illustrate how to setup a Fabric CA for a production network using the Fabric CA binaries. After you have mastered deploying and running a CA by using these binaries, it is likely you will want to use the Fabric CA image instead, for example in a Kubernetes or Docker deployment. For now though, the purpose of this guide is to teach you how to properly use the binaries. Then the process can be extended to other environments.

The first topic introduces you to planning for a CA and deciding on the CA topology that is required for your organization. Next, you should review the checklist for a production CA server to understand the most common configuration parameters for a CA and how they interact with each other. Finally, the CA deployment steps walk you through the process of configuring a TLS CA, an organization CA, and optionally, an intermediate CA for your production network. When this configuration is complete, you are ready to use the organization CA, or intermediate CA if you create one, to register and enroll the identities for your organization.

3.1 Planning for a CA

Audience: Architects, network operators, users setting up a production Fabric network and are familiar with Transport Layer Security (TLS), Public Key Infrastructure (PKI) and Membership Service Providers (MSPs).

These deployment instructions provide guidance for how to deploy a CA for a Production network. If you need to quickly stand up a network for education or testing purposes, check out the Fabric test network. While the Fabric CA server remains a preferred and tested certificate authority for Hyperledger Fabric, you can instead use certificates from a non-Fabric CA with your Fabric network; however, the scope of this deployment guide is focused on using a Fabric CA. It focuses on the most important configuration parameters you need to consider and provides best practices for configuring a CA.

You may already be familiar with the Fabric CA User's Guide and the Operations Guide. This topic is intended to inform your decisions before deploying a CA and provide guidance on how to configure the CA parameters based on those decisions. You may still need to reference those topics when you make your decisions.

Recall that a Fabric CA performs the following functions on a blockchain network:

- Registration of identities, or connect to a Lightweight Directory Access Protocol (LDAP) as the user registry.
- Issuance of Enrollment Certificates (ECerts). Enrollment is a process whereby the Fabric CA issues a certificate key-pair, comprised of a signing certificate and a private key that forms the identity. The private and public keys are first generated locally by the Fabric CA client, and then the public key is sent to the CA which returns an encoded certificate, the signing certificate.
- Certificate renewal and revocation.

You have the opportunity to customize the behavior of these functions. The first time the CA is started, it looks for a fabric-ca-server-config.yaml file which contains the CA configuration parameters. If the file is not there, a default one

is created for you. Before you deploy your CA, this topic provides guidance around the parameters in that file and the decisions you need to make in order to customize the CA according to your use case.

3.1.1 What CA topology will you use on your network?

The topology of CAs on your network can vary depending on how many organizations participate on the network and how you prefer to administer your CAs.

How many CAs are required?

Before configuring a CA, you need to understand how many CAs are required for your network. If you read the Deployment Process Overview, you'll recall that it is recommended that you deploy two CAs per organization, an organization CA and a TLS CA. TLS communications are required for any production network to secure communications between nodes in the organization. Thus, it is the TLS CA that issues those TLS certificates. The organization CA on the other hand is used to generate organization and node identities. Also, because this is a *distributed ledger*, the ordering service should not be part of the same organization as the peers, so you will need separate organizations (and therefore CAs) for your peer organizations and ordering service organization. When multiple organizations contribute nodes to an ordering service, each ordering node would have its own organization CA. All of this separation is crucial for distributed management of the ordering service and channels and defeats the ability of a bad actor to disrupt the network.

Why is a separate TLS server recommended?

A separate TLS server provides an independent chain of trust just for securing communications. Most organizations prefer that the TLS communications are secured by separate crypto material – from a different root, either from a separate Fabric TLS CA or another external Certificate Authority.

One option that Fabric provides is the ability to configure a dual-headed CA, a single CA that under the covers includes an organization's identity enrollment CA, hereafter called organization CA, and a TLS CA. They operate on the same CA node and port but are addressable by a different CA name. The `cafiles` parameter discussed in more detail later in this topic allows each CA to have its own configuration but is beneficial when you want both CAs to share the same backend database. Thus with this option, conveniently, when the organization CA is deployed, the TLS CA is automatically deployed for you along side it.

It is also worth noting that from a functional perspective there is no difference between a Fabric organization CA and a Fabric TLS CA. The difference lies in types of certificates they generate.

When would I want an intermediate CA?

Intermediate CAs are optional. For added security, organizations can deploy a chain of CAs known as intermediate CAs. An intermediate CA has their root certificate issued by a parent CA (root CA) or another intermediate authority (that becomes the parent CA), which establishes a "chain of trust" for any certificate that is issued by any CA in the chain. Therefore, having one or more intermediate CAs allows you to protect your root of trust. This ability to track back to the root CA not only allows the function of CAs to scale while still providing security — allowing organizations that consume certificates to use intermediate CAs with confidence — it limits the exposure of the root CA, which, if compromised, would endanger the entire chain of trust. If an intermediate CA is compromised, on the other hand, there will be a much smaller exposure. A key benefit is that after the intermediate CAs are up and running, the root CA can be effectively turned off, limiting its vulnerability even more.

Another reason to include an intermediate CA would be when you have a very large organization with multiple departments and you don't want a single CA to generate certificates for all of the departments. Intermediate CAs provide a mechanism for scoping the certificates that a CA manages to a smaller department or sub-group. Intermediate CAs

are not required, but they mitigate risk and scope certificate management. This pattern incurs significant overheard if there will only be a small number of members in the organization. As an alternative to deploying multiple intermediate CAs, you can configure a CA with `affiliations` (similar to departments) instead. More on this later when we talk about `affiliations` .

Also, in situations where it is acceptable for one TLS CA to be used to issue certificates to secure communications for an entire organization, it would be reasonable for intermediate CAs to use the same TLS CA as the root CA rather than having their own dedicated TLS CA.

In what order should the CAs be deployed?

If a dual-headed CA is not configured with an organization CA and TLS CA, then the TLS CA is deployed separately, and needs to be deployed *before* the organization CA in order to generate the TLS certificate for the organization CA. After the TLS CA is deployed, then the organization CA can be deployed followed by any intermediate CAs if required.

3.1.2 Deciding on a user registry

Because the Fabric CA controls the identities for an organization, you need to decide on your user registry before you configure your CA. The Fabric CA server can be configured to use a database as the user registry or it can be configured to read from an Lightweight Directory Access Protocol (LDAP) server. LDAP is an industry standard for server data storage and retrieval where information is represented as a hierarchical tree. The LDAP protocol is used to lookup data on the server, in this context the data is users and groups, and the server is therefore a user repository. When an LDAP server will be your user repository, you will need to provide the connection and configuration details. After LDAP is configured for the CA, it authenticates an identity against the LDAP user registry prior to generating the certificates for that user, a process known as `enrollment` . Additionally, user attributes retrieved from the LDAP registry are useful for making access control decisions in smart contracts. See Configuring LDAP if you want to learn more about LDAP considerations.

This deployment guide demonstrates the process for configuring the user registry with a database.

3.1.3 A note about configuration

There are three ways to configure settings on the Fabric CA server and client. The precedence order for overriding the default settings is:

1. Use the Fabric CA server CLI commands.
2. Use environment variables to override configuration file settings.
3. Modify the configuration file.

This order means that from a code perspective, any flags passed on a Fabric CA server CLI command will override an environment variable if it exists for the same setting, as well as the default value of the setting in the configuration file. Likewise, environment variables can be used to override settings in the configuration file. However, use of environment variables to modify configuration settings is discouraged because the changes are not persisted and can lead to problems later when they do not get set or are not set to what they should be. It's important to understand the parameters in the configuration file and their dependencies on other parameter settings in the file. Blindly overriding one setting using an environment variable could affect the functionality of another setting. Therefore, the recommendation is that before starting the server, you **make the modifications to the settings in the configuration file** to become familiar with the available settings and how they work.

Note that some configuration settings are stored in the CA database which means that after a CA is started, overriding the settings can no longer be performed by editing the configuration file or by setting environment variables. Affected

parameters are noted throughout these instructions. In these cases, the modifications are required to be made by using the Fabric CA server CLI commands and have the added benefit of not requiring a server restart.

3.2 Checklist for a production CA server

As you prepare to build a production Fabric CA server, you need to consider configuration of the following fields in the fabric-ca-server-config.yaml file. When you initialize the CA server, this file is generated for you so that you can customize it before actually starting the server.

This checklist covers key configuration parameters for setting up a production network. Of course you can always refer to the fabric-ca-server-config.yaml file for additional parameters or more information. The list of parameters in this topic includes:

- *ca*
- *tls*
- *cafiles*
- *intermediate ca*
- *port*
- *user registry*
- *registry database*
- *LDAP*
- *affiliations*
- *csr*
- *signing*
- *bccsp*
- *cors*
- *cfg*
- *operations*
- *metrics*

3.2.1 ca

```
ca:
  # Name of this CA
  name:
  # Key file (is only used to import a private key into BCCSP)
  keyfile:
  # Certificate file (default: ca-cert.pem)
  certfile:
  # Chain file
  chainfile:
```

Start by giving your CA a name. Often the name indicates the organization that this CA will serve. Or, if this is a TLS CA you might want to indicate that in the name. This `ca.name` is used when targeting this server for requests by the Fabric CA Client `--caname` parameter. If the crypto material for this CA is generated elsewhere, you can

provide the name of the files along with the fully qualified path or relative path to where they reside. The `keyfile` parameter is the private key and `certfile` is the public key.

The `chainfile` parameter applies only to intermediate CAs. When starting up an intermediate CA, the CA will create the chainfile at the location specified by this parameter. The file will contain the certificates of the trusted chain starting with the root cert plus any intermediate certificates.

3.2.2 tls

```
tls:
  # Enable TLS (default: false)
  enabled: true
  # TLS for the server's listening port
  certfile:
  keyfile:
  clientauth:
    type: noclientcert
    certfiles:
```

Configure this section to enable TLS communications for the CA. After TLS is enabled, all nodes that transact with the CA will also need to enable TLS.

- **`tls.enabled`** : For a secure production environment, TLS should be enabled for secure communications between nodes by setting `enabled: true` in the `tls` section of the config file. (Note that it is disabled by default which may be acceptable for a test network but for production it needs to be enabled.) This setting will configure `server-side TLS` , meaning that TLS will guarantee the identity of the *server* to the client and provides a two-way encrypted channel between them.
- **`tls.certfile`** : Every CA needs to register and enroll with its TLS CA before it can transact securely with other nodes in the organization. Therefore, before you can deploy an organization CA or an intermediate CA, you must first deploy the TLS CA and then register and enroll the organization CA bootstrap identity with the TLS CA to generate the organziation CA's TLS signed certificate. When you use the Fabric CA client to generate the certificates, the TLS signed cert is generated in the `signedcerts` folder of the specified msp directory of the `FABRIC_CA_CLIENT_HOME` folder. For example: `/msp/signcerts/cert.pem` . Then, in this `tls.certfile` field, provide the name and location of the generated TLS signed certificate. If this is the root TLS CA, this field can be blank.
- **`tls.keyfile`** : Similar to the `tls.certfile` , provide the name and location of the generated TLS private key for this CA. For example: `/msp/keystore/87bf5eff47d33b13d7aee81032b0e8e1e0ffc7a6571400493a7c_sk` . If you are using an HSM or if this is a root TLS CA, this field will be blank.

Note: Minimally in a production network, `server-side TLS` should be enabled.

If server-side TLS is sufficient for your needs, you are done with this section. If you need to use mutual TLS in your network then you need to configure the following two additional fields. (Mutual TLS is disabled by default.)

- **`tls.clientauth.type`** : If the server additionally needs to authenticate the identity of the *client*, then **mutual TLS** (mTLS) is required. When mTLS is configured, the client is required to send its certificate during a TLS handshake. To configure your CA for mTLS, set the `clientauth.type` to `RequireAndVerifyClientCert` .
- **`tls.clientauth.certfiles`** : For mTLS only, provide the PEM-encoded list of root certificate authorities that the server uses when verifying client certificates. Specify the certificates in a dashed yaml list.

3.2.3 cafiles

```
cafiles:
```

As mentioned in the topic on Planning for a CA, the `cafiles` parameter can be used to configure a dual-headed CA – a single CA that under the covers includes both an organization CA and a TLS CA. This usage pattern can be for convenience, allowing each CA to maintain its own configuration but still share the same back-end user database. In the `cafiles` parameter, enter the path to the `fabric-ca-server-config.yaml` of the second CA server, for example the TLS CA. The configuration of the secondary CA can contain all of the same elements as are found in the primary CA server config file except `port` and `tls` sections.

If this is not a desired configuration for your CA, you can leave the value of this parameter blank.

3.2.4 intermediate CA

```
intermediate:
  parentserver:
    url:
    caname:

  enrollment:
    hosts:
    profile:
    label:

  tls:
    certfiles:
    client:
      certfile:
      keyfile:
```

Intermediate CAs are not required, but to reduce the risk of your organization (root) CA becoming compromised, you may want to include one or more intermediate CAs in your network.

Important: Before setting up an intermediate CA, you need to verify the value of the `csr.ca.pathlength` parameter in the parent CA. When set to `0` , the organization CA can issue intermediate CA certificates, but these intermediate CAs may not in turn enroll other intermediate CAs. If you want your intermediate CA to be able to enroll other intermediate CAs, the root ca `csr.ca.pathlength` needs to be set to `1` . And if you want those intermediate CAs to enroll other intermediate CAs, the root ca `csr.ca.pathlength` would need to be set to `2` .

- **`parentserver.url`** : Specify the parent server url in the format `https://<PARENT-CA-ENROLL-ID>:<PARENT-CA-SECRET>@<PARENT-CA-URL>:<PARENT-CA-PORT>` .
- **`parentserver.caname`** : Enter `ca.name` of the parent CA server.
- **`enrollment.profile:`** : Enter the value of the `signing.profile` for the parent CA. Normally this would be `ca` .
- **`tls.certfiles`** : Enter the location and name of the TLS CA signing cert, `ca-cert.pem` file. For example, `tls/ca-cert.pem` . This location is relative to where the server configuration .yaml file exists.

In addition to editing this `intermediate` section, you also need to edit the following sections of the configuration .yaml file for this intermediate CA:

- `csr` - Ensure that the `csr.cn` field is blank.
- `port` - Be sure to set a unique port for the intermediate CA.

- `signing` - Verify that `isca` set to `true` , and `maxpathlen` is set to greater than `0` in the root CA only if the intermediate CA will serve as a parent CA to other intermediate CAs, otherwise it should be set to `0` . See the *signing* parameter.

3.2.5 port

```
port:
```

Each CA must be running on its own unique port and obviously must not conflict with any other service running on that port. You need to decide what ports you want to use for your CAs ahead of time and configure that port in the .yaml file.

3.2.6 user registry

```
registry:
  # Maximum number of times a password/secret can be reused for enrollment
  # (default: -1, which means there is no limit)
  maxenrollments: -1

  # Contains identity information which is used when LDAP is disabled
  identities:
     - name: <<<adminUserName>>>
       pass: <<<adminPassword>>>
       type: client
       affiliation: ""
       attrs:
          hf.Registrar.Roles: "*"
          hf.Registrar.DelegateRoles: "*"
          hf.Revoker: true
          hf.IntermediateCA: true
          hf.GenCRL: true
          hf.Registrar.Attributes: "*"
          hf.AffiliationMgr: true
```

If you are not using an LDAP user registry, then this section along with the associated registry database `db:` section are required. This section can be used to register a list of users with the CA when the server is started. Note that it simply registers the users and does not generate enrollment certificates for them. You use the Fabric CA client to generate the associated enrollment certificates.

- `maxenrollments` : Used to restrict the number of times certificates can be generated for a user using the enroll ID and secret. The reenroll command can be used to get certificates without any limitation.
- `identities` : This section defines the list of users and their associated attributes to be registered when the CA is started. After you run the `fabric-ca-server init` command, the `<<<adminUserName>>>` and `<<<adminPassword>>>` here are replaced with the CA server bootstrap identity user and password specified with the `-b` option on the command.
- `identities.type` : For Fabric, the list of valid types is `client` , `peer` , `admin` , `orderer` , and `member` .
- `affiliation` : Select the affiliation to be associated with the user specified by the associated `name:` parameter. The list of possible affiliations is defined in the `affiliations:` section.
- `attrs` : The list of roles included above would be for "admin" users, meaning they can register and enroll other users. If you are registering a non-admin user, you would not give them these permissions. The `hf` attributes

associated with an identity affect that identity's ability to register other users. You should review the topic on Registering a new identity to understand the patterns that are required.

When the user is subsequently "enrolled", the `type` , `affiliation` , and `attrs` are visible inside the user's signing certificate and are used by policies to enforce authorization. Recall that `enrollment` is a process whereby the Fabric CA issues a certificate key-pair, comprised of a signed cert and a private key that forms the identity. The private and public keys are first generated locally by the Fabric CA client, and the public key is then sent to the CA which returns an encoded certificate, the signing certificate.

After a user has been registered, you can use the `Identity` command to modify the properties of the user.

3.2.7 registry database

```
db:
  type: sqlite3
  datasource: fabric-ca-server.db
  tls:
      enabled: false
      certfiles:
      client:
        certfile:
        keyfile:
```

The Fabric CA stores user identities, affiliations, credentials, and public certificates in a database. Use this section to specify the type of database to be used to store the CA data. Fabric supports three database **types**:

- `sqlite` (SQLite Version 3)
- `postgres` (PostgresSQL)
- `mysql` (MySQL)

If you are running the database in a cluster, you must choose `postgres` or `mysql` as the database type.

If LDAP is being used as the user registry (designated by `ldap.enabled:true`), then this section is ignored.

3.2.8 LDAP

```
ldap:
   # Enables or disables the LDAP client (default: false)
   # If this is set to true, the "registry" section is ignored.
   enabled: false
   # The URL of the LDAP server
   url: ldap://<adminDN>:<adminPassword>@<host>:<port>/<base>
   # TLS configuration for the client connection to the LDAP server
   tls:
      certfiles:
      client:
         certfile:
         keyfile:
   # Attribute related configuration for mapping from LDAP entries to Fabric CA␣
↪attributes
   attribute:
      # 'names' is an array of strings containing the LDAP attribute names which are
      # requested from the LDAP server for an LDAP identity's entry
      names: ['uid','member']
      # The 'converters' section is used to convert an LDAP entry to the value of
```

```
      # a fabric CA attribute.
      # For example, the following converts an LDAP 'uid' attribute
      # whose value begins with 'revoker' to a fabric CA attribute
      # named "hf.Revoker" with a value of "true" (because the boolean expression
      # evaluates to true).
      #    converters:
      #       - name: hf.Revoker
      #         value: attr("uid") =~ "revoker*"
      converters:
         - name:
           value:
      # The 'maps' section contains named maps which may be referenced by the 'map'
      # function in the 'converters' section to map LDAP responses to arbitrary␣
↪values.
      # For example, assume a user has an LDAP attribute named 'member' which has␣
↪multiple
      # values which are each a distinguished name (i.e. a DN). For simplicity,␣
↪assume the
      # values of the 'member' attribute are 'dn1', 'dn2', and 'dn3'.
      # Further assume the following configuration.
      #    converters:
      #       - name: hf.Registrar.Roles
      #         value: map(attr("member"),"groups")
      #    maps:
      #       groups:
      #          - name: dn1
      #            value: peer
      #          - name: dn2
      #            value: client
      # The value of the user's 'hf.Registrar.Roles' attribute is then computed to be
      # "peer,client,dn3".  This is because the value of 'attr("member")' is
      # "dn1,dn2,dn3", and the call to 'map' with a 2nd argument of
      # "group" replaces "dn1" with "peer" and "dn2" with "client".
      maps:
         groups:
            - name:
              value:
```

If an LDAP registry is configured, all settings in the *registry* section are ignored.

3.2.9 affiliations

```
affiliations:
   org1:
      - department1
      - department2
   org2:
      - department1
```

Affiliations are useful to designate sub-departments for organizations. They can then be referenced from a policy definition, for example when you might want to have transactions endorsed by a peer who is not simply a member of ORG1, but is a member of ORG1.MANUFACTURING. Note that the affiliation of the registrar must be equal to or a prefix of the affiliation of the identity being registered. If you are considering using affiliations you should review the topic on Registering a new identity for requirements. To learn more about how affiliations are used in an MSP, see the MSP Key concept topic on Organizational Units (OUs) and MSPs.

The default affiliations listed above are added to the Fabric CA database the first time the server is started. If you prefer not to have these affiliations on your server, you need to edit this config file and remove or replace them *before* you start the server for the first time. Otherwise, you must use the Fabric CA client Affiliation command to modify the list of affiliations. By default, affiliations cannot be removed from the configuration, rather that feature has to be explicitly enabled. See the *cfg* section for instructions on configuring the ability to allow removal of affiliations.

3.2.10 csr (certificate signing request)

```
csr:
   cn: <<<COMMONNAME>>>
   keyrequest:
     algo: ecdsa
     size: 256
   names:
      - C: US
        ST: "North Carolina"
        L:
        O: Hyperledger
        OU: Fabric
   hosts:
     - <<<MYHOST>>>
     - localhost
   ca:
      expiry: 131400h
      pathlength: <<<PATHLENGTH>>>
```

The CSR section controls the creation of the root CA certificate. Therefore, if you want to customize any values, it is recommended to configure this section before you start your server for the first time. The values you specify here will be included in the signing certificates that are generated. If you customize values for the CSR after you start the server, you need to delete the `ca.cert` file and `ca.key` files and then run the `fabric-ca-server start` command again.

- **csr.cn**: This field must be set to the ID of the CA bootstrap identity and can be left blank. It defaults to the CA server bootstrap identity.
- **csr.keyrequest**: Use these values ff you want to customize the crypto algorithm and key sizes.
- **csr.names**: Specify the values you want to use for the certificate Issuer, visible in the signing certificate.
- **csr.hosts**: Provide the host names that the server will be running on.
- **csr.expiry**: Specify when this CA's root certificate expires. The default value `131400h` is 15 years.
- **csr.pathlength**: If you will have an intermediate CA, set this value to `1` in the root CA. If this is an intermediate CA the value would be `0` unless it will serve as a parent CA for another intermediate CA.

3.2.11 signing

```
signing:
    default:
      usage:
        - digital signature
      expiry: 8760h
    profiles:
      ca:
         usage:
           - cert sign
```

```
          - crl sign
        expiry: 43800h
        caconstraint:
          isca: true
          maxpathlen: 0
          maxpathlenzero: true
    tls:
       usage:
          - signing
          - key encipherment
          - server auth
          - client auth
          - key agreement
       expiry: 8760h
```

The defaults in this section are normally sufficient for a production server. However, you might want to modify the default expiration of the generated organization CA and TLS certificates. Note that this is different than the `expiry` specified in the `csr` section for the CA root certificate.

If this is a TLS CA, it is recommended that you remove the `ca` section from `profiles:` since a TLS CA should only be issuing TLS certificates.

If you plan to have more than one level of intermediate CAs, then you must set `maxpathlen` to greater than `0` in the configuration .yaml file for the root CA. This field represents the maximum number of non-self-issued intermediate certificates that can follow this certificate in a certificate chain. For example, if you plan to have intermediate CAs under this root CA, the `maxpathlen` can be set to `0`. But if you want your intermediate CA to serve as a parent CA to another intermediate CA, then `maxpathlen` should be set to `1`.

To enforce a `maxpathlen` of `0`, you need to also set `maxpathlenzero` to true. If `maxpathlen` is greater than `0`, `maxpathlenzero` should be set to `false`.

3.2.12 bccsp

```
bccsp:
    default: SW
    sw:
        hash: SHA2
        security: 256
        filekeystore:
            # The directory used for the software file-based keystore
            keystore: msp/keystore
```

The information in this section controls where the private key for the CA is stored. The configuration above causes the private key to be stored on the file system of the CA server in the `msp/keystore` folder. If you plan to use a Hardware Security Module (HSM) the configuration is different. When you configure HSM for a CA, the CA private key is generated by and stored in the HSM instead of the `msp/keystore` folder. An example of the HSM configuration for softHSM would be similar to:

```
bccsp:
  default: PKCS11
  pkcs11:
    Library: /etc/hyperledger/fabric/libsofthsm2.so
    Pin: 71811222
    Label: fabric
    hash: SHA2
```

```
    security: 256
    Immutable: false
```

3.2.13 cors

```
cors:
    enabled: false
    origins:
      - "*"
```

Cross-Origin Resource Sharing (CORS) can be configured to use additional HTTP headers to tell browsers to give a web application running at one origin access to selected resources from a different origin. The `origins` parameter contains a list of domains that are allowed to access the resources.

3.2.14 cfg

```
cfg:
  affiliations:
    allowremove: false
  identities:
    allowremove: false
```

These two parameters are not listed in the sample configuration file, but are important to understand. With the default configuration set to false, you will be unable to remove affiliations or identities without a server restart. If you anticipate the need to remove affiliations or identities from your production environment without a server restart, both of these fields should be set to `true` before starting your server. Note that after the server is started, affiliations and identities can only be modified by using the Fabric CA client CLI commands.

3.2.15 operations

```
operations:
    # host and port for the operations server
    listenAddress: 127.0.0.1:9443

    # TLS configuration for the operations endpoint
    tls:
        # TLS enabled
        enabled: false

        # path to PEM encoded server certificate for the operations server
        cert:
            file:

        # path to PEM encoded server key for the operations server
        key:
            file:

        # require client certificate authentication to access all resources
        clientAuthRequired: false

        # paths to PEM encoded ca certificates to trust for client authentication
```

```
        clientRootCAs:
            files: []
```

The operations service can be used for health monitoring of the CA and relies on mutual TLS for communication with the node. Therefore, you need to set `operations.tls.clientAuthRequired` to `true`. When this is set to `true`, clients attempting to ascertain the health of the node are required to provide a valid certificate for authentication. If the client does not provide a certificate or the service cannot verify the client's certificate, the request is rejected. This means that the clients will need to register with the TLS CA and provide their TLS signing certificate on the requests.

In the case where two CAs are running on the same machine, you need to modify the `listenAddress:` for the second CA to use a different port. Otherwise, when you start the second CA, it will fail to start, reporting that `the bind address is already in use.`

3.2.16 metrics

```
metrics:
    # statsd, prometheus, or disabled
    provider: disabled

    # statsd configuration
    statsd:
        # network type: tcp or udp
        network: udp

        # statsd server address
        address: 127.0.0.1:8125

        # the interval at which locally cached counters and gauges are pushed
        # to statsd; timings are pushed immediately
        writeInterval: 10s

        # prefix is prepended to all emitted statsd metrics
        prefix: server
```

If you want to monitor the metrics for the CA, choose your metrics provider:

- **provider**: `Statsd` is a push model, `Prometheus` is a pull model. Because Prometheus is a pull model there is not any configuration required from the Fabric CA server side. Rather, Prometheus sends requests to the operations URL to poll for metrics. Available metrics.

3.2.17 Next steps

After deciding on your CA configuration, you are ready to deploy your CAs. Follow instructions in the next CA Deployment steps topic to start your CA.

3.3 CA Deployment steps

3.3.1 Download the binaries

The Fabric CA server and CA client binaries can be downloaded from github. Scroll down to **Assets** and select the latest binary for your machine type. The .zip file contains both the CA server and the CA client binaries. After you

have mastered deploying and running a CA by using these binaries, it is likely you will want to use the Fabric CA image instead, for example in a Kubernetes or Docker deployment. For now though, the purpose of this topic is to teach you how to properly use the binaries.

Server binary file

In this topic, we use the server binaries to deploy three different types of CAs: the TLS CA, an organization CA, and optionally an intermediate CA. The TLS CA issues certificates that secure communicates between all the nodes in the organization. The organization CA issues identity certificates. If you decide to include an intermediate CA, the organization CA serves as the root CA or parent server for the intermediate CA. If you have not already, you should review the topic on Planning for a CA to understand the purpose of each type of CA and their differences. We run the TLS CA, organization CA, and intermediate CAs from different folders. We will copy the CA server binary to each folder.

Client binary file

Likewise, we will copy the Fabric CA client binary to its own directory. Having the CA client in its own folder facilitates certificate management, especially when you need to interact with multiple CAs. When you issue a command from the CA client against a CA server, you can target a specific CA by modifying the CA server URL on the request. Therefore, only a single Fabric CA client binary is required and can be used to transact with multiple CAs. More on using the Fabric CA client below.

3.3.2 Fabric CA client

Before deploying a Fabric CA server, you need to understand the role of the Fabric CA client. While you can use the Fabric SDKs to interact with your CA, it is recommended that you **use the Fabric CA client to register and enroll node admin identities**. The instructions provided in this topic assume a single Fabric CA client is being used. Registering an identity, or user, is the process by which the enroll id and secret is added to the CA database "user registry". If you are using LDAP server for your user registry, then the register step is not required because the identities already exist in the LDAP database. After a user is registered you can use the Fabric CA client to "enroll" the identity which is the process that generates the certificates the identity needs to transact as part of the organization. When you submit an enrollment request, the private and public keys are first generated locally by the Fabric CA client, and then the public key is sent to the CA which returns an encoded "signed certificate".

Because you will use a single CA client to submit register and enrollment requests to multiple CAs, certificate management is critically important when using the CA client. A best practice therefore is to create sub-folders for each CA server that the CA client will interact with, to store the generated certificates.

- Create a sub-folder in order to connect to each CA server, such as `/tls-ca` or `/org1-ca` or `/int-ca` . This folder can be under the Fabric CA client or anywhere that the CA client can access the path. For purposes of these instructions, these folders reside inside the `fabric-ca-client` directory. For example:

```
mkdir fabric-ca-client
cd fabric-ca-client
mkdir tls-ca org1-ca int-ca
```

  Tip: While you can run the Fabric CA client binary from any folder you prefer, for ease of following these instructions we will refer to it in its own directory named `fabric-ca-client` .

- Copy the Fabric CA client binary into the `fabric-ca-client` folder.
- Because TLS communications are enabled on a production network, the TLS CA for the organization is responsible for generating certificates that secure communications between all nodes in the organization. Therefore, every time the Fabric CA client transacts with a CA server in that organization, it needs to provide the TLS

CA "root certificate" to secure the client-server communication. For example, when the Fabric CA client issues a register or enroll request to the CA server, the client request includes that root certificate to perform an SSL handshake. The TLS CA root certificate, named `ca-cert.pem`, is generated on the TLS CA after TLS is enabled in the server config .yaml file. To enable TLS communications for your CA client, you need a `tls-root-cert` sub-folder to store the root certificate. Later in this topic, we will copy the root certificate into this folder.

```
mkdir tls-root-cert
```

The resulting folder structure resembles:

```
fabric-ca-client
  -- int-ca
  -- org1-ca
  -- tls-ca
  -- tls-root-cert
```

Important: If your Fabric CA client will transact with CAs from multiple organizations that are secured by different TLS servers, then you would need to either create different `tls-root-cert` folders to hold the TLS CA root certificate for each organization or simply name them differently inside the folder to differentiate them. Since our Fabric CA client will only be transacting with CA servers in the same organization, all of which are secured by the same TLS CA, we will only have a single root certificate in this folder.

You can use environment variables or flags on the CLI commands to specify the location of certificates and the Fabric CA client binary:

- `FABRIC_CA_CLIENT_HOME` - Specify the fully qualified path to where Fabric CA client binary resides.
- `FABRIC_CA_CLIENT_TLS_CERTFILES` - Specify the location and name of the TLS CA root certificate. If the path of the environment variable `FABRIC_CA_CLIENT_TLS_CERTFILES` is not an absolute path, it will be parsed as relative to the Fabric CA client's home directory as specified by `FABRIC_CA_CLIENT_HOME`. Throughout these instructions, we use the `--tls.certfiles` flag on the commands instead to specify the location of the TLS CA root certificate.
- `FABRIC_CA_CLIENT_MSPDIR` - While you can use this environment variable to specify the name of the folder where the certificates are located, because the client communicates with multiple CAs, *a better option is to explicitly pass the `--mspdir` flag on the register and enroll commands to specify the location.* If not specified on the command, the location defaults to `$FABRIC_CA_CLIENT_HOME/msp` which will be problematic if the Fabric CA client transacts with multiple CA servers in the organization.

Tip: The first time you issue an `enroll` command from the CA client, if the `fabric-ca-client-config.yaml` does not already exist in the `$FABRIC_CA_CLIENT_HOME` directory, it is generated. When you customize the values in this file, they are used automatically by the CA client and do not have to be passed on the command line on a subsequent `enroll` command.

The usage of a single Fabric CA client to interact with multiple CA servers is used throughout these instructions but is not necessarily a required pattern. Another alternative is to have a single Fabric CA client for *each* CA server. In that case, the Fabric CA client connection settings to the server are generated and stored in the `fabric-ca-client-config.yaml` file when the initial enroll command is issued for the CA server admin.

3.3.3 Submitting transactions from the CLI

Two sets of CLI commands are included with the CA server and CA client binary files:

- Use the Fabric CA **server** CLI commands to deploy and update the CA server.
- Use the Fabric CA **client** CLI commands to submit requests to your CA server after it is set up, such as registering, enrolling or revoking identities.

We will use both of these CLI commands throughout this topic.

3.3.4 What order should I deploy the CAs?

Assuming you are not deploying a dual-headed CA that contains both a TLS CA and an organization CA together, you would deploy the CAs in the following order:

1. **Deploy the TLS CA**

   Because TLS communication is required in a Production network, TLS must be enabled on each CA, peer, and ordering node. While the example configuration in the CA Operations Guide shares a single TLS CA across all organizations, the recommended configuration for production is to deploy a TLS CA for each organization. The TLS CA issues the TLS certificates that secure communications between all the nodes on the network. Therefore, it needs to be deployed first to generate the TLS certificates for the TLS handshake that occurs between the nodes.

2. **Deploy the organization CA**

   This is the organization's identity enrollment CA and is used to register and enroll the identities that will participate in the network from this organization.

3. **Deploy the intermediate CA (Optional)**

   If you decide to include an intermediate CA in your network, the intermediate CA's parent server (the associated root CA) must be deployed before any intermediate CAs.

3.3.5 Deploy the TLS CA

Regardless of whether you are setting up a TLS CA, an organization CA or an intermediate CA, the process follows the same overall steps. The differences will be in the modifications you make to the CA server configuration .yaml file. The following steps provide an overview of the process:

- *Step one: Initialize the CA server*
- *Step two: Modify the CA server configuration*
- *Step three: Delete the CA server certificates*
- *Step four: Start the CA server*
- *Step five: Enroll bootstrap user with TLS CA*

When you deploy any node, you have three options for your TLS configuration:

- No TLS. *Not recommended for a production network.*
- Server-side TLS.
- Mutual TLS.

This process will configure a CA with server-side TLS enabled which is recommended for production networks. Mutual TLS is disabled by default. If you need to use mutual TLS, refer to the TLS configuration settings.

Before you begin

You should have already downloaded and copied the Fabric CA server binary `fabric-ca-server` to a clean directory on your machine. For purposes of these instructions, we put the binary in its own folder named `fabric-ca-server-tls`.

```
mkdir fabric-ca-server-tls
```

Copy the `fabric-ca-server` binary into this folder.

Initialize the TLS CA server

The first step to deploy a CA server is to "initialize" it. Run the following CA server CLI command to initialize the server by specifying the admin user id and password for the CA:

```
./fabric-ca-server init -b <ADMIN_USER>:<ADMIN_PWD>
```

For example:

```
cd fabric-ca-server-tls
./fabric-ca-server init -b tls-admin:tls-adminpw
```

The `-b` (bootstrap identity) flag bootstraps the admin username and password to the CA server which effectively "registers" the CA admin user with the server for you, so an explicit Fabric CA client CLI `register` command is not required for the bootstrapped user. All CA users need to be "registered" and then "enrolled" with the CA, except for this CA admin identity which is implicitly registered by using the `-b` flag. The registration process inserts the user into the CA database. The `-b` option is not required for initialization when LDAP will be configured.

Note: This example is for illustration purposes only. Obviously, in a production environment you would never use `tls-admin` and `tls-adminpw` as the bootstrap username and password. Be sure that you record the admin id and password that you specify. They are required later when you issue register and enroll commands against the CA. It can help to use a meaningful id to differentiate which server you are transacting with and follow secure password practices.

What does the CA server `init` command do?

The `init` command does not actually start the server but generates the required metadata if it does not already exist for the server:

- Sets the default the CA Home directory (referred to as `FABRIC_CA_HOME` in these instructions) to where the `fabric-ca-server init` command is run.
- Generates the default configuration file `fabric-ca-server-config.yaml` that is used as a template for your server configuration in the `FABRIC_CA_HOME` directory. We refer to this file throughout these instructions as the "configuration .yaml" file.
- Creates the TLS CA root signed certificate file `ca-cert.pem`, if it does not already exist in the CA Home directory. This is the **self-signed root certificate**, meaning it is generated and signed by the TLS CA itself and does not come from another source. This certificate is the public key that must be shared with all clients that want to transact with any node in the organization. When any client or node submits a transaction to another node, it must include this certificate as part of the transaction.
- Generates the CA server private key and stores it in the `FABRIC_CA_HOME` directory under `/msp/keystore`.
- Initializes a default SQLite database for the server although you can modify the database setting in the configuration .yaml file to use the supported database of your choice. Every time the server is started, it loads the data from this database. If you later switch to a different database such as PostgreSQL or MySQL, and the identities defined in the `registry.identites` section of the configuration .yaml file don't exist in that database, they will be registered.

- Bootstraps the CA server administrator, specified by the `-b` flag parameters `<ADMIN_USER>` and `<ADMIN_PWD>`, onto the server. When the CA server is subsequently started, the admin user is registered with the admin attributes provided in the configuration .yaml file `registry` section. If this CA will be used to register other users with any of those attributes, then the CA admin user needs to possess those attributes. In other words, the registrar must have the `hf.Registrar.Roles` attributes before it can register another identity with any of those attributes. Therefore, if this CA admin will be used to register the admin identity for an Intermediate CA, then this CA admin must have the `hf.IntermediateCA` set to `true` even though this may not be an intermediate CA server. The default settings already include these attributes.

Important: When you modify settings in the configuration .yaml file and restart the server, the **previously issued certificates are not replaced**. If you want the certificates to be regenerated when the server is started, you need to delete them and run the `fabric-ca-server start` command. For example, if you modify the `csr` values after you start the server, you need to delete the previously generated certificates, and then run the `fabric-ca-server start` command. Be aware though, that when you restart the CA server using the new signed certificate and private key, all previously issued certificates will no longer be able to authenticate with the CA.

Modify the TLS CA server configuration

Now that you have initialized your server, you can edit the generated `fabric-ca-server-config.yaml` file to modify the default configuration settings for your use case according to the Checklist for a production CA server.

At a minimum you should do the following:

- `port` - Enter the port that you want to use for this server. These instructions use `7054`, but you can choose your port.
- `tls.enabled` - Recall that TLS is disabled in the default configuration file. Since this is a production server, enable it by setting this value to `true`. Setting this value to `true` causes the TLS signed certificate `tls-cert.pem` file to be generated when the server is started in the next step. The `tls-cert.pem` is the certificate the server will present to the client during a TLS handshake, which the client will then verify using the TLS CA's `ca-cert.pem`.
- `ca.name` - Give the CA a name by editing the parameter, for example `tls-ca`.
- `csr.hosts` - Update this parameter to include this hostname and ip address where this server is running, if it is different than what is already in this file.
- `signing.profiles.ca` - Since this is a TLS CA that will not issue CA certificates, the `ca` profiles section can be removed. The `signing.profiles` block should only contain `tls` profile.
- `operations.listenAddress:` - In the unlikely case that there is another node running on this host and port, then you need to update this parameter to use a different port.

Delete the TLS CA server certificates

Before starting the server, if you modified any of the values in the `csr` block of the configuration .yaml file, you need to delete the `fabric-ca-server-tls/ca-cert.pem` file and the entire `fabric-ca-server-tls/msp` folder. These certificates will be re-generated when you start the CA server in the next step.

Start the TLS CA server

Run the following command to start the CA server:

```
./fabric-ca-server start
```

When the server starts successfully you will see something similar to:

```
[INFO] Listening on https://0.0.0.0:7054
```

Because you have enabled TLS communications, notice that the TLS signed certificate `tls-cert.pem` file is generated under the `FABRIC_CA_HOME` location.

Tip: The CA `ADMIN_USER` and `ADMIN_PWD` that were set on the `init` command cannot be overridden with the `-b` flag on this `start` command. When you need to modify the CA admin password, use the Fabric CA client identity command.

Optional flags:

- `-d` - If you want to run the server in DEBUG mode which facilitates problem diagnosis, you can include the `-d` flag on the start command. However, in general it is not recommended to run a server with debug enabled as this will cause the server to perform slower.
- `-p` - If you want the server to run on a port different than what is specified in the configuration .yaml file, you can override the existing port.

Enroll bootstrap user with TLS CA

Now that your TLS CA is configured and before you can deploy any other nodes for your organization, you need to enroll the bootstrap (admin) user of the TLS CA. Since the CA server is up and running, instead of using the **Fabric CA server CLI commands** we now use the **Fabric CA client CLI commands** to submit an enrollment request to the server.

Performed by using the Fabric CA client, the enrollment process is used to generate the certificate and private key pair which forms the node identity. You should have already setup the required folders in the *Fabric CA client* section.

The folder structure that we are using for these Fabric CA client commands is:

```
fabric-ca-client
  -- tls-ca
  -- tls-root-cert
```

These folders are used by the Fabric CA client to:

- Store the certificates that are issued when the Fabric CA client enroll command is run against the TLS CA server to enroll the TLS CA bootstrap identity. (**tls-ca** folder)
- Know where the TLS CA root certificate resides that allows the Fabric CA client to communicate with the TLS CA server. (**tls-root-cert** folder)

1. Copy the TLS CA root certificate file `fabric-ca-server-tls/ca-cert.pem`, that was generated when the TLS CA server was started, to the `fabric-ca-client/tls-root-cert/tls-ca-cert.pem` folder. Notice the file name is changed to `tls-ca-cert.pem` to make it clear this is the root certificate from the TLS CA. **Important:** This TLS CA root certificate will need to be available on each client system that will run commands against the TLS CA.
2. The Fabric CA Client also needs to know where Fabric CA client binary is located. The `FABRIC_CA_CLIENT_HOME` environment variable is used to set the location.

   ```
   export FABRIC_CA_CLIENT_HOME=<FULLY-QUALIFIED-PATH-TO-FABRIC-CA-BINARY>
   ```

   For example, if you are in the `fabric-ca-client` folder you can use:

   ```
   export FABRIC_CA_CLIENT_HOME=$PWD
   ```

3. You are ready to use the Fabric CA client CLI to enroll the TLS CA admin user. Run the command:

```
./fabric-ca-client enroll -d -u https://<ADMIN>:<ADMIN-PWD>@<CA-URL>:<PORT> --tls.
↪certfiles <RELATIVE-PATH-TO-TLS-CERT> --enrollment.profile tls --csr.hosts '<CA_
↪HOSTNAME>' --mspdir tls-ca/tlsadmin/msp
```

Replace:

- `<ADMIN>` - with the TLS CA admin specified on the `init` command.
- `<ADMIN-PWD>` - with the TLS CA admin password specified on the `init` command.
- `<CA-URL>` - with the hostname specified in the `csr` section of the TLS CA configuration .yaml file.
- `<PORT>` - with the port that the TLS CA is listening on.
- `<RELATIVE-PATH-TO-TLS-CERT>` - with the path and name of the root TLS certificate file that you copied from your TLS CA. This path is relative to `FABRIC_CA_CLIENT_HOME` . If you are following the folder structure in this tutorial it would be `tls-root-cert/tls-ca-cert.pem` .
- `<CA_HOSTNAME>` - with a comma-separated list of host names for which the certificate should be valid. If not specified, the default value from the `fabric-ca-client-config.yaml` is used. You can specify a wildcard for the domain. For example, when you include the flag `--csr.hosts 'host1,*.example.com'` it means that the hostname `host1` is recognized as well as any host from the `example.com` domain. These values are inserted into the generated certificate Subject Alternative Name (SAN) attribute. The value specified here corresponds to the `csr.hosts` parameter that you specified for the CA server.

For example:

```
./fabric-ca-client enroll -d -u https://tls-admin:tls-adminpw@my-machine.example.
↪com:7054 --tls.certfiles tls-root-cert/tls-ca-cert.pem --enrollment.profile tls␣
↪--csr.hosts 'host1,*.example.com' --mspdir tls-ca/tlsadmin/msp
```

In this case, the `-d` parameter runs the client in DEBUG mode which is useful for debugging enrollment failures.

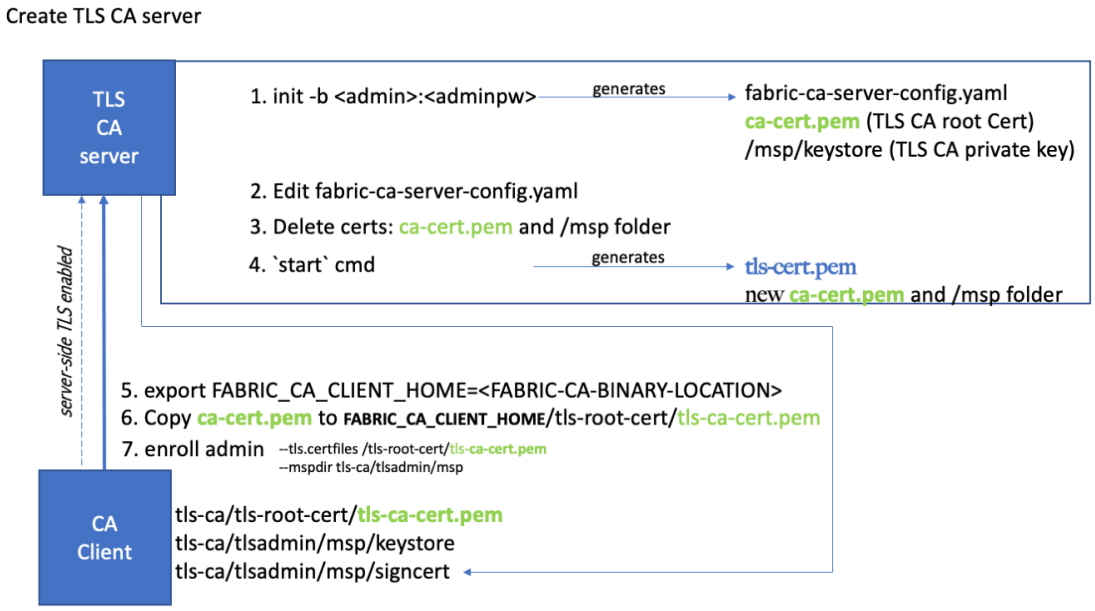
Notice the `--mspdir` flag is used on the command to designate where to store the TLS CA admin certificates that are generated by the enroll command.

The `--enrollment.profile tls` flag is specified because we are enrolling against the TLS CA. Use of this flag means that the enrollment is performed according to the `usage` and `expiry` settings of the TLS profile that is defined in the `signing` section of the configuration .yaml file. **Note:** If you removed the `signing.profiles.ca` block from the TLS CA configuration .yaml file, you could omit the `--enrollment.profile tls` flag.

When this command completes successfully, the `fabric-ca-client/tls-ca/tlsadmin/msp` folder is generated and contains the signed cert and private key for the TLS CA admin identity. If the enroll command fails for some reason, to avoid confusion later, you should remove the generated private key from the `fabric-ca-client/tls-ca/admin/msp/keystore` folder before reattempting the enroll command. We will reference this crypto material later when it is required to register other identities with the TLS CA.

Tip: After you issue this first `enroll` command from the Fabric CA client, examine the contents of the generated `fabric-ca-client/fabric-ca-client-config.yaml` file to become familiar with the default settings that are used by the Fabric CA client. Because we are using a single Fabric CA client to interact with multiple CA servers, we need to use the `-u` flags on the client CLI commands to target the correct CA server. In conjunction, the `--mspdir` flag indicates the location of the cryptographic material to use on a `register` command or where to store the generated certificates on an `enroll` command.

The following diagram is a conceptual summary of the steps you perform to create a TLS CA server and enroll the bootstrap identity using the Fabric CA client:



Register and enroll the organization CA bootstrap identity with the TLS CA

The TLS CA server was started with a bootstrap identity which has full admin privileges for the server. One of the key abilities of the admin is the ability to register new identities. Each node in the organization that transacts on the network needs to register with the TLS CA. Therefore, before we set up the organization CA, we need to use the TLS CA to register and enroll the organization CA bootstrap identity to get its TLS certificate and private key. The following command registers the organization CA bootstrap identity `rcaadmin` and `rcaadminpw` with the TLS CA.

```
./fabric-ca-client register -d --id.name rcaadmin --id.secret rcaadminpw -u https://
↪my-machine.example.com:7054  --tls.certfiles tls-root-cert/tls-ca-cert.pem --mspdir␣
↪tls-ca/tlsadmin/msp
```

Notice that the `--mspdir` flag on the command points to the location of TLS CA admin msp certificates that we generated in the previous step. This crypto material is required to be able to register other users with the TLS CA.

Next, we need to enroll the `rcaadmin` user to generate the TLS certificates for the identity. In this case, we use the `--mspdir` flag on the enroll command to designate where the generated organization CA TLS certificates should be stored for the `rcaadmin` user. Because these certificates are for a different identity, it is a best practice to put them in their own folder. Therefore, instead of generating them in the default `msp` folder, we will put them in a new folder named `rcaadmin` that resides along side the `tlsadmin` folder.

```
./fabric-ca-client enroll -d -u https://rcaadmin:rcaadminpw@my-machine.example.com:
↪7054 --tls.certfiles tls-root-cert/tls-ca-cert.pem --enrollment.profile tls --csr.
↪hosts 'host1,*.example.com' --mspdir tls-ca/rcaadmin/msp
```

In this case, the `--mspdir` flag works a little differently. For the enroll command, the `--mspdir` flag indicates where to store the generated certificates for the `rcaadmin` identity.

Important: The organization CA TLS signed certificate is generated under `fabric-ca-client/tls-ca/rcaadmin/msp/signcert` and the private key is available under `fabric-ca-client/tls-ca/rcaadmin/msp/keystore` . When you deploy the organization CA

you will need to point to the location of these two files in the `tls` section of the CA configuration .yaml file. For ease of reference, you can rename the file in the `keystore` folder to `key.pem`.

(Optional) Register and enroll the Intermediate CA admin with the TLS CA

Similarly, if you are planning to have an intermediate CA that can issue certificates on behalf of the organization CA, you should also register and enroll the intermediate CA admin user now as well. The following command registers the intermediate CA admin id `icaadmin` and `icaadminpw` with the TLS CA. You can use any values you choose for the identity name and password.

```
./fabric-ca-client register -d --id.name icaadmin --id.secret icaadminpw -u https://
↪my-machine.example.com:7054  --tls.certfiles tls-root-cert/tls-ca-cert.pem --mspdir␣
↪tls-ca/tlsadmin/msp
```

Again, the `--mspdir` flag on the register command points to the location of TLS CA admin msp certificates that are required to be able to register other users with the TLS CA.

Now would also be a good time to generate the intermediate CA TLS certificates for the `icaadmin` user by enrolling the user. For the enroll command, we use the `--mspdir` flag to designate where the generated intermediate CA TLS certificates should be stored for the `icaadmin` user. In this case we put them into a new folder named `icaadmin/msp` along side the `tlsadmin` folder.

```
./fabric-ca-client enroll -d -u https://icaadmin:icaadminpw@my-machine.example.com:
↪7054 --tls.certfiles tls-root-cert/tls-ca-cert.pem --enrollment.profile tls --csr.
↪hosts 'host1,*.example.com' --mspdir tls-ca/icaadmin/msp
```

Important: The intermediate CA TLS signed certificate is generated under `fabric-ca-client/tls-ca/icaadmin/signcert` and the private key is available under `fabric-ca-client/tls-ca/icaadmin/keystore`. When you deploy the intermediate CA you will need to refer to these two files in the `tls` section of the intermediate CA configuration .yaml file. For ease of reference you can rename the file in the `keystore` folder to `key.pem`.

The resulting folder structure resembles:

```
fabric-ca-client
  -- tls-ca
    -- tlsadmin
      -- msp
    -- rcaadmin
      -- msp
    -- icaadmin
      -- msp
    -- tls-root-cert
      -- tls-ca-cert.pem
```

Tip: After you have registered all your nodes with the TLS CA, it can be safely turned off.

3.3.6 Deploy an organization CA

The deployment process overview describes the need for both an organization CA and a TLS CA for every organization. The TLS CA issues the TLS certificates that allow for secure transactions within the organization. The organization CA, also referred to as the "enrollment CA" or the "eCert CA" is used to issue identities for the organization. You deployed the TLS CA in the previous set of steps, now we are ready to deploy the organization CA. Later in this topic you can optionally create an intermediate CA; therefore, this CA serves as the "root CA" in that chain of trust.

Because you've already registered and enrolled your organization CA bootstrap identity `rcaadmin` with the TLS CA in the previous step, you are ready to deploy the CA following the same pattern of steps that were used when you deployed the TLS CA.

Before you begin

- Copy the Fabric CA server binary `fabric-ca-server` to a new directory on your machine. For purposes of these instructions, we put the binary in its own folder named `fabric-ca-server-org1`.

```
mkdir fabric-ca-server-org1
```

  Now, copy the `fabric-ca-server` binary into this folder.

- Using the following commands, copy the organization CA TLS certificate and key pair that you generated in the previous step to a location that can be accessed by this CA server, for example `fabric-ca-server-org1/tls`. These are the `fabric-ca-client/tls-ca/rcaadmin/msp/signcerts/cert.pem` and `fabric-ca-client/tls-ca/rcaadmin/msp/keystore/` files that were generated by the enroll command.

  Note: The following commands assume that:

  - The generated private key under `fabric-ca-client/tls-ca/rcaadmin/msp/keystore/` was renamed to `key.pem`.
  - The `fabric-ca-client` and `fabric-ca-server-org1` folders are at the same level in your file structure.

```
cd fabric-ca-server-org1
mkdir tls
cp ../fabric-ca-client/tls-ca/rcaadmin/msp/signcerts/cert.pem tls && cp ../fabric-
↪ca-client/tls-ca/rcaadmin/msp/keystore/key.pem tls
```

  The resulting folder structure is similar to the following diagram. (Some folders and files have been omitted for clarity):

```
fabric-ca-client
  -- tls-ca
        -- rcaadmin
          -- msp
             -- IssuerPublicKey
             -- IssuerRevocationPublicKey
             -- cacerts
             -- keystore
                 -- key.pem
             -- signcerts
                 -- cert.pem
fabric-ca-server-org1
  -- tls
    -- cert.pem
    -- key.pem
```

Initialize the CA server

Run the command to initialize the server, specifying a new admin user id and password for the CA. We use the same identity `rcaadmin` that we registered with the TLS CA in the previous set of steps as the bootstrap identity of the organization CA. Run this command from the `fabric-ca-server-org1` folder.

```
./fabric-ca-server init -b <ADMIN_USER>:<ADMIN_PWD>
```

For example:

```
./fabric-ca-server init -b rcaadmin:rcaadminpw
```

Modify the CA server configuration

As we did with the TLS CA, we need to edit the generated `fabric-ca-server-config.yaml` file for the organization CA to modify the default configuration settings for your use case according to the Checklist for a production CA server.

At a minimum, you should edit the following fields:

- `port` - Enter the port that you want to use for this server. These instructions use `7055`, but you can choose your port.
- `tls.enabled` - Enable TLS by setting this value to `true`.
- `tls.certfile` and `tls.keystore` - Enter the relative path and filenames for the TLS CA signed certificate and private key that were generated when the bootstrap admin for this CA was enrolled with the TLS CA. The signed certificate, `cert.pem`, was generated using the Fabric CA client and can be found under `fabric-ca-client/tls-ca/rcaadmin/msp/signcerts/cert.pem`. The private key is located under `fabric-ca-client/tls-ca/rcaadmin/msp/keystore`. The specified path name is relative to `FABRIC_CA_CLIENT_HOME` therefore if you are following the folder structure that is used throughout these instructions you can simply specify `tls/cert.pem` for the `tls.certfile` and `tls/key.pem` for the `tls.keystore` or you can specify the fully qualified path name.
- `ca.name` - Give the organization CA a name by specifying a value in this parameter, for example `org1-ca`.
- `csr.hosts` - Update this parameter to include this hostname and ip address where this server is running if it is different than what is already in the file.
- `operations.listenAddress:` - If there is another CA running on this host, then you need to update this parameter to use a different port.
- `csr.ca.pathlength`: This field is used to limit CA certificate hierarchy. Setting this value to `1` for the root CA means the root CA can issue intermediate CA certificates, but these intermediate CAs cannot in turn issue other **CA** certificates. In other words the intermediate CA cannot enroll other intermediate CAs, but it can issue enrollment certificates for users. The default value is `1`.
- `signing.profiles.ca.caconstraint.maxpathlen` - This field represents the maximum number of non-self-issued intermediate certificates that can follow this certificate in a certificate chain. **If this will be a parent server for an intermediate CA, and you want that intermediate CA to act as a parent CA for another intermediate CA, this root CA needs to set this value to greater than 0 in the configuration .yaml file.** See the instructions for the signing section. The default value is `0`.
- `operations.listenAddress:` - In the unlikely case that there is another node running on this host and port, then you need to update this parameter to use a different port.

Delete the CA server certificates

Before starting the server, if you modified any of the values in the `csr` block of the configuration .yaml file, you need to delete the `fabric-ca-server-org1/ca-cert.pem` file and the entire `fabric-ca-server-org1/msp` folder. These certificates will be re-generated based on the new settings in the configuration .yaml file when you start the CA server in the next step.

Start the CA server

Run the following command to start the CA server:

```
./fabric-ca-server start
```

Enroll the CA admin

The final step for deploying the CA is to enroll the CA admin bootstrap identity which generates the node signed certificate and private key. The key-pair is required for this admin identity to be able to enroll other identities. Again we will use the Fabric CA client CLI to enroll the admin. You should have already setup the required folders in the *Fabric CA client* section.

The folder structure we are using for these commands is:

```
fabric-ca-client
  -- org1-ca
  -- tls-root-cert
```

These folders are used by the Fabric CA client to:

- Store the certificates that are issued when the Fabric CA client enroll command is run against the TLS CA server. (**org1-ca** folder)
- Know where the TLS certificate resides that allows the Fabric CA client to communicate with the TLS CA server. (**tls-root-cert** folder)

1. When you previously used the Fabric CA client to generate certificates for the TLS CA, you designated the value of the `FABRIC_CA_CLIENT_HOME` . Assuming that is still set you can proceed to the next step. Otherwise, you should be in the directory where the Fabric CA client binary resides and run the command:

   ```
   export FABRIC_CA_CLIENT_HOME=$PWD
   ```

2. Now you can use the Fabric CA client to generate the CA admin certificate and private key. You need this certificate and private key to be able to issue identities using this CA. We use the `--mspdir` flag on the enroll command to designate where to store the generated certificates. Run the command:

   ```
   ./fabric-ca-client enroll -d -u https://<ADMIN>:<ADMIN-PWD>@<CA-URL>:<PORT> --tls.
   ↪certfiles <RELATIVE-PATH-TO-TLS-CERT> --csr.hosts '<CA_HOSTNAME>' --mspdir org1-
   ↪ca/rcaadmin/msp
   ```

   Replace:

   - `<ADMIN>` - with the organization CA admin specified on the `init` command.
   - `<ADMIN-PWD>` - with the organization CA admin password specified on the `init` command.
   - `<CA-URL>` - with the hostname specified in the `csr` section of the organization CA configuration .yaml file.
   - `<PORT>` - with the port that the organization CA is listening on.
   - `<RELATIVE-PATH-TO-TLS-CERT>` - with the path to the tls-ca-cert.pem file that you copied from your TLS CA. This is the path relative to `FABRIC_CA_CLIENT_HOME` .
   - `<CA_HOSTNAME>` - with a comma-separated list of host names for which the certificate should be valid. If not specified, the default value from the `fabric-ca-client-config.yaml` is used. If a host name is dynamic you can specify a wildcard for the domain. For example, when you include the flag

`--csr.hosts 'host1,*.example.com'` it means that the hostname `host1` is recognized as well as any host from the `example.com` domain.

In this case, the `-d` parameter runs the client in DEBUG mode which is useful for debugging command failures.

For example:

```
./fabric-ca-client enroll -d -u https://rcaadmin:rcaadminpw@my-machine.example.
↪com:7055 --tls.certfiles tls-root-cert/tls-ca-cert.pem --csr.hosts 'host1,*.
↪example.com' --mspdir org1-ca/rcaadmin/msp
```

When this command runs, the enroll command creates the `fabric-ca-client/org1-ca/rcaadmin/msp` folder and contains the signed cert and private key for the organization CA and looks similar to:

```
-- msp
    -- cacerts
        -- my-machine-example-com-7055.pem
    -- keystore
        -- 60b6a16b8b5ba3fc3113c522cce86a724d7eb92d6c3961cfd9afbd27bf11c37f_sk
    -- signcerts
        -- cert.pem
    -- user
    -- IssuerPublicKey
    -- IssuerRevocationPublicKey
```

Where:

- `my-machine-example-com-7055.pem` is the **Organization CA root certificate**.
- `60b6a16b8b5ba3fc3113c522cce86a724d7eb92d6c3961cfd9afbd27bf11c37f_sk` is the **private key** for the organization CA admin identity. This key needs to be protected and should not be shared with anyone. It is required to be able to register and enroll other identities with this CA. Feel free to rename this file to something easier to reference, for example `org1-key.pem`.
- `cert.pem` is the CA admin identity **signed certificate**.

3. (Optional) Register the intermediate CA bootstrap identity with the organization (root) CA.

   If you plan to deploy an intermediate CA, you must register the intermediate CA bootstrap identity with its root CA in order to form the chain of trust. Recall that you already registered the `icaadmin` identity with the TLS CA. You also need to register the same identity with the (root) organization CA. And because this will be an intermediate CA, you must include the `hf.IntermediateCA=true` attribute. (Run this command from the same terminal window where you enrolled the organization CA admin in the previous step.)

   ```
   ./fabric-ca-client register -u https://my-machine.example.com:7055  --id.name␣
   ↪icaadmin --id.secret icaadminpw --id.attrs '"hf.Registrar.Roles=user,admin","hf.
   ↪Revoker=true","hf.IntermediateCA=true"' --tls.certfiles tls-root-cert/tls-ca-
   ↪cert.pem --mspdir org1-ca/rcaadmin/msp
   ```

   The `--mspdir` flag on the register command points to the crypto material for the organization CA admin that we enrolled in the previous step and is authorized to register new users. We don't enroll the `icaadmin` identity with the organization CA. Rather this intermediate CA admin identity is enrolled later against the intermediate CA.

3.3.7 (Optional) Deploy an intermediate CA

Intermediate CAs form a chain a trust with the organization root CA and can be used to direct enrollment requests for a specific organization to a single CA as well as protect the root of trust by shutting down the root CA. Thus, when intermediate CAs are used to process all of the enrollment requests, the root CA can be turned off.

Note: This section assumes that you have already *registered and enrolled* the `icaadmin` identity with the TLS CA as well as the parent organization CA (step 3 immediately preceding this section).

Before you begin

- Copy the Fabric CA server binary `fabric-ca-server` to a new directory on your machine. For purposes of these instructions, we put the binary in its own folder named `fabric-ca-server-int-ca`.

```
mkdir fabric-ca-server-int-ca
```

  Copy the `fabric-ca-server` binary into this folder.

- Use the following commands to copy the CA admin TLS certificate and key pair that you generated in the previous step to a location that can be accessed by this CA server, for example `fabric-ca-server-int-ca/tls`. These are the `fabric-ca-client/tls-ca/icaadmin/msp/signcerts/cert.pem` and `fabric-ca-client/tls-ca/icaadmin/msp/keystore/` files that were generated by the enroll command.

  Note: The following commands assume that:

  - The generated private key under `fabric-ca-client/tls-ca/icaadmin/keystore/` is renamed to `key.pem`.
  - The `fabric-ca-client` and `fabric-ca-server-int-ca` folders are at the same level in your file structure.

```
cd fabric-ca-server-int-ca
mkdir tls
cp ../fabric-ca-client/tls-ca/icaadmin/msp/signcerts/cert.pem tls && cp ../fabric-
↪ca-client/tls-ca/icaadmin/msp/keystore/key.pem tls
```

- Because TLS communications are enabled, the intermediate CA needs the TLS CA root certificate to be able to securely communicate with the parent organization CA. Therefore, you need to copy the `fabric-ca-server-tls/ca-cert.pem`, that was generated when the TLS CA server was initialized, to the `tls` folder. Notice the file name is changed to `tls-ca-cert.pem` to make it clear this is the root certificate from the TLS CA.

```
cp ../fabric-ca-server-tls/ca-cert.pem tls/tls-ca-cert.pem
```

The resulting folder structure is similar to the following structure. (Some folders and files have been omitted for clarity):

```
fabric-ca-client
  -- tls-ca
        -- icaadmin
          -- msp
             -- cacerts
             -- keystore
                 -- key.pem
             -- signcerts
                 -- cert.pem
             -- tlscacerts
             -- user
             -- IssuerPublicKey
             -- IssuerRevocationPublicKey
fabric-ca-server-int-ca
  -- tls
```

```
-- tls-ca-cert.pem
-- cert.pem
-- key.pem
```

Because you have already deployed the parent organization (root) CA, you can use the following steps to create the intermediate CA:

1. From the intermediate CA home directory, initialize the CA by running the `init` command and bootstrapping the `icaadmin` id that you already registered with the TLS CA and parent organization CA. For example:

   ```
   ./fabric-ca-server init -b icaadmin:icaadminpw
   ```

2. Modify the `fabric-ca-server-config.yaml` file.

   - `port` : Specify a unique port for this server. These instructions use `7056` , but you can choose your port.
   - `tls.enabled` : Must be set to `true` .
   - `tls.certfile` and `tls.keystore` : Enter the path and filename for the TLS CA signed certificate and private key. These are the certificate and private key files that you created when you enrolled the `icaadmin` user with the TLS CA. The signed certificate, `cert.pem` can be found under `fabric-ca-client/tls-ca/icaadmin/msp/signcerts/cert.pem` . The private key is located under `fabric-ca-client/tls-ca/icaadmin/msp/keystore` . The specified path name is relative to `FABRIC_CA_CLIENT_HOME` therefore if you are following the folder structure that is used throughout these instructions you can simply specify `tls/cert.pem` for the `tls.certfile` and `tls/key.pem` for the `tls.keystore` or you can specify the fully qualified path name.
   - `ca` : Specify a name for the ca. For example `ica` .
   - `signing.profiles.ca.caconstraint.maxpathlen` : Set this value to 0, meaning there are no more intermediate CA's under this one. The default value is `0` .
   - `csr.cn` : The common name must be blank for intermediate CAs.
   - `csr.ca.pathlength` : Set this value to 0.
   - `intermediate.parentserver.url` : Enter the value of the parent server URL, in the form `https://<ROOT-CA-ADMIN>:<ROOT-CA-ADMIN-PW>@<CA-URL>:<PORT>` , for example `https://rcaadmin:rcaadminpw@my-machine.example.com:7055` .
   - `intermediate.parentserver.caname` : Enter the value of the parent server `caname` from the parent organization CA server configuration .yaml file. In this tutorial, we named that CA, `org1-ca` .
   - `intermediate.enrollment.hosts` : Enter the host name that the intermediate CA server is listening on.
   - `intermediate.enrollment.profile` : Enter the name of the signing profile from `signing.profile` section to use when issuing certificates. Normally this value is `ca` .
   - `intermediate.tls.certfiles` : Enter the path and file name to the TLS CA root `tls-ca-cert.pem` file. If you are following the folder structure that is used throughout these instructions you can simply specify `tls/tls-ca-cert.pem` .
   - `operations.listenAddress` : If another CA is running on the same host, you need to specify a unique port.

3. **Important:** You must delete the intermediate CA `fabric-ca-server-int-ca/ca-cert.pem` and `fabric-ca-server-int-ca/msp` folders in order for them to be regenerated with the intermediate CA settings.

4. Start the intermediate CA server. Because the intermediate CA bootstrap identity is enrolled with the parent organization (root) CA when the server is started, **ensure that the parent organziation CA is running before you start the intermediate CA**.

```
./fabric-ca-server start
```

As this is an intermediate CA server, notice that a `ca-chain.pem` file is generated. This file contains the certificate chain, and includes the intermediate CA `ca-cert.pem` as well as the root CA `ca-cert.pem`.

Enroll the Intermediate CA admin

The final step for deploying the intermediate CA is to enroll the intermediate CA admin to generate the node signed certificate and private key which is required for the identity to be able to enroll other identities. You should have already setup the required folders in the *Fabric CA client* section.

The folder structure we are using for these commands is

```
fabric-ca-client
  -- int-ca
  -- tls-root-cert
```

These folders are used by the Fabric CA client to:

- Store the certificates that are issued when the Fabric CA client enroll command is run against the TLS CA server. (**int-ca** folder)
- Know where the TLS certificate resides that allows the Fabric CA client to communicate with the TLS CA server. (**tls-root-cert** folder)

1. When you previously used the Fabric CA client to generate certificates for the TLS CA and organization CA, you designated the value of the `FABRIC_CA_CLIENT_HOME`. Assuming that is still set, you can proceed to the next step. Otherwise, you should be in the directory where the Fabric CA client binary resides and run the command:

```
export FABRIC_CA_CLIENT_HOME=$PWD
```

2. Now you can use the Fabric CA client to generate the CA admin certificate and private key. You need this certificate and private key to be able to issue identities using this CA. We use the `--mspdir` flag on the enroll command to designate where to store the generated certificates. Run the command:

```
./fabric-ca-client enroll -d -u https://<ADMIN>:<ADMIN-PWD>@<CA-URL>:<PORT> --tls.
↪certfiles <RELATIVE-PATH-TO-TLS-CERT> --csr.hosts '<CA_HOSTNAME>' --mspdir int-ca/
↪icaadmin/msp
```

Replace:

- `<ADMIN>` - with the intermediate CA admin specified on the `init` command.
- `<ADMIN-PWD>` - with the intermediate CA admin password specified on the `init` command.
- `<CA-URL>` - with the hostname specified in the `csr` section of the intermediate CA configuration .yaml file.
- `<PORT>` - with the port that the intermediate CA is listening on.
- `<RELATIVE-PATH-TO-TLS-CERT>` - with the path to the tls-ca-cert.pem file that you copied from your TLS CA. This is the path relative to `FABRIC_CA_CLIENT_HOME`.
- `<CA_HOSTNAME>` - with a comma-separated list of host names for which the certificate should be valid. If not specified, the default value from the `fabric-ca-client-config.yaml` is used. If a host name is dynamic you can specify a wildcard for the domain. For example, when you include the flag `--csr.hosts`

`'host1,*.example.com'` it means that the hostname `host1` is recognized as well as any host from the `example.com` domain.

For example:

```
./fabric-ca-client enroll -d -u https://icaadmin:icaadminpw@my-machine.example.com:
↪7056 --tls.certfiles tls-root-cert/tls-ca-cert.pem --csr.hosts 'host1,*.example.com
↪' --mspdir int-ca/icaadmin/msp
```

When the enroll command runs, it creates the `fabric-ca-client/int-ca/icaadmin/msp` folder and contains the signed cert and private key for the intermediate CA. Notice the `/intermediatecerts` folder is also created and populated with the intermediate CA certificate which connects this intermediate CA to the root CA.

Tip: After the intermediate CA is successfully deployed and you can register and enroll identities, then you can safely turn off the parent server root CA, the organization CA.

3.3.8 Next steps

Minimally, you should now have a TLS CA and an organization CA configured for your organization. You can now use the Fabric CA client to register and enroll node admin identities, node identities, and organization identities with the TLS CA to generate their TLS certificates required for server side TLS communications. Likewise you will also need to register and enroll the same node admins and users with the organization CA to generate their enrollment certificates and MSPs. See Use the CA to create identities and MSPs for more information. If you did configure an intermediate CA, you can now use that CA to register and enroll identities for the organization instead of the root CA.

Tip: When you subsequently use the Fabric CA client to register identities with the intermediate CA, ensure that you specify the `--mspdir int-ca/icaadmin/msp` on the register command.

3.3.9 Troubleshooting CA Deployment

Fabric CA client `enroll` command fails

Problem: When running an enroll command with the Fabric CA client CLI, it fails with:

```
Error: Failed to read config file at '/Users/mwp/.fabric-ca-client/fabric-ca-client-
↪config.yaml': While parsing config: yaml: line 42: mapping values are not allowed␣
↪in this context
```

Solution:

This error occurs when the `FABRIC_CA_CLIENT_HOME` is not set. Ensure that you have set the `FABRIC_CA_CLIENT_HOME` environment variable to point to the location of the Fabric CA client binary. Navigate to the folder where the `fabric-ca-client.exe` binary file resides and run the command:

```
export FABRIC_CA_CLIENT_HOME=$PWD
```

Note that when `FABRIC_CA_CLIENT_HOME` is set and an enrollment command fails, it is recommended that should delete the generated `FABRIC_CA_CLIENT_HOME/msp` folder and the generated `fabric-ca-client.yaml` file to avoid confusion before reattempting the enrollment command.

Intermediate CA server fails to start

Problem: The intermediate CA server fails to start with the error:

```
Error: Response from server: Error Code: 0 - Certificate signing failure: {"code":
↪5300,"message":"Policy violation request"}
```

You may also see the associated error on the root CA:

```
[ERROR] local signer certificate disallows CA MaxPathLen extending
[INFO] 9.27.117.220:49864 POST /enroll 500 0 "Certificate signing failure: {"code":
↪5300,"message":"Policy violation request"}"
```

Solution: The values of the `signing.profiles.ca.caconstraint.maxpathlen` and the `csr.ca.pathlength` fields in the intermediate CA configuration .yaml file need to be set to 0.

Starting the intermediate CA fails

Problem: When you start the intermediate CA it fails with an error:

```
Post https://host1.com:7060/enroll: x509: certificate signed by unknown authority
```

And the Root organization CA, has the error:

```
TLS handshake error from 192.168.1.134:63094: remote error: tls: bad certificate
```

Solution: This problem occurs during enrollment of the intermediate CA admin user with the root CA when the intermediate CA server is started. To resolve this problem, make sure that the TLS certificate that is specified in the `intermediate.tls.certfiles` section of the intermediate CA `fabric-ca-server-config.yaml` file points to the TLS CA root certificate. If you are following these instructions it will be `tls/tls-ca-cert.pem`.

Enrolling the intermediate CA admin user fails

Problem: When you start the intermediate CA and the process fails with the error:

```
Error: Response from server: Error Code: 0 - Chain file does not exist at /fabric-ca-
↪server-int-ca/ca-chain.pem
```

Solution:

Because you modified the `csr` block of the intermediate CA configuration file, you need to delete the intermediate CA, `ca-cert.pem` file and the `/msp` folder before you start the intermediate CA server.

3.4 Registering and enrolling identities with a CA

Audience: organization administrators, node administrators

If you've read our topics on identity and Membership Service Provider (MSP) you're aware that in Hyperledger Fabric, Certificate Authorities are used to generate the identities assigned to admins, nodes, and users (client applications). While any Certificate Authority that can generate x.509 certificates can be used to create the public/private key pair that constitutes an identity, the Fabric CA can additionally generate the local and organizational MSP folder structures that are required by Hyperledger Fabric.

In this topic, we'll show a "happy path" for using the Fabric CA to generate identities and MSPs. Note that you do not have to use the Fabric CA to register and enroll identities. However, if you use a different CA, you will need to create the relevant identities and MSPs that Fabric uses to build organizations, client identities, and nodes. We will show examples of those MSPs below.

3.4.1 Overview of registration and enrollment

While it is possible for the admin of a CA to create an identity and give the public/private key pair to a user out of band, this process would give the CA admin access to the private key of every user. Such an arrangement violates basic security procedures regarding the security of private keys, which should not be exposed for any reason.

As a result, CA admins **register** users, a process in which the CA admin gives an enroll ID and secret (these are similar to a username and password) to an identity and assigns it a role and any required attributes. The CA admin then gives this enroll ID and secret to the ultimate user of the identity. The user can then execute a Fabric CA client **enroll** command using this enroll ID and secret, returning the public/private key pair containing the role and attributes assigned by the CA admin.

This process preserves both the integrity of the CA (because only CA admins can register users and assign roles and affiliations) and private keys (since only the user of an identity will have access to them).

While admin identities only need to be registered and enrolled with an "organization CA" that generates identity certificates for both admins and nodes alike, nodes must also be registered and enrolled with a TLS CA. This will create a public/private TLS key pair that nodes use to sign and encrypt their communications. If the TLS CA has been created only using a "TLS" profile, the commands to register and enroll an identity with the organization CA are identical to those to register and enroll with the TLS CA. If you are using a CA that contains both profiles, you will have to specify the TLS profile when communicating with the CA. For more information about creating a CA that can only function as a TLS CA, check out the CA deployment guide.

3.4.2 Before you begin

In this tutorial, we will assume that a CA server has been configured and set up using the CA setup instructions. We will also show both the commands and variables used when registering an identity (a task handled by a CA admin or an identity with CA `registrar` rights), and when "enrolling" an identity (a task handled by the user of the identity).

In either case, the `fabric-ca-client` must be set up, as it is used to make calls to the CA server where an identity is registered and enrolled. If you are operating in a production environment, you should have TLS enabled, and will need to provide TLS certificates to secure your communications with the CA. The TLS certificates will need to come from the TLS CA you spin up alongside the "organization" CA you use to generate identities for nodes, admins, and clients, and this TLS CA is the same CA you will use to generate certificates (since nodes use TLS to communicate with each other).

Decide on the structure of your folders and certificates

Whether you are running in a test or production environment, it is critically important that you maintain a consistent and coherent structure for managing your folders and certificates. While it is not a strict Fabric necessity to use the same patterns everywhere (as long as your paths are correct whenever you reference them, for example when bootstrapping a node, Fabric can consume them), certificate pathing errors are among the most common errors faced by Fabric users. Forethought and consistency can dramatically reduce these issues.

Note that production deployments might include structures we won't show here. For example, it might include a folder for **gateways**, allowing an admin to easily see which organizations and nodes and clients are associated with each network. Similarly, you might see a **smart contracts** folder containing the smart contracts associated with a network.

The method for organizing your folders and certificates we describe here is not mandatory, but you will find it helpful, as it is consistent with the rest of this topic as well as with the CA deployment guide. Most importantly, it organizes your structures around the **organization** that owns and manages them. While it might be natural to think of your deployments as being organized around physical structures like peers and ordering nodes, it is actually the organization that is the centralizing figure. Especially since not all network participants will necessarily own a node.

The structures and methods presented here also represent best practices, preventing cases where, for example, Fabric expects an MSP folder to be called `msp` and, if a different name is used, the name of the folder will have to be changed in the relevant YAML file.

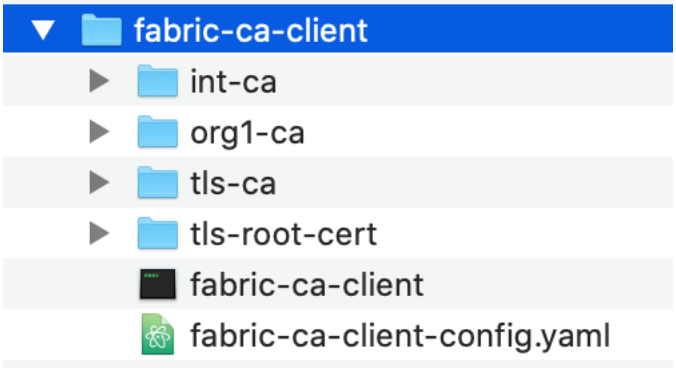
Folder structure for operating the Fabric CA client

While a consistent structure is important for the certificates you will get back from a CA (which you will use when creating nodes and acting as an admin), it is also important for those who will be using a single Fabric CA client to connect to multiple CAs as an admin. This is because unlike organization admins, in which a single identity can be used as the admin of as many nodes as you need, each CA necessarily has a separate admin that is registered when it is bootstrapped and later enrolled.

This is why if you are connecting to different CA servers as an admin from the same CA client, when you use the `--mspdir` flag, you also **must** include the `-u` flag to target the correct CA server. This will allow you to specify the correct CA admin credentials for the CA you are connecting to.

If you will only be using a single CA client to target a single CA server (which will more often be the case for users who will be admins of organizations or nodes), you have the option of specifying the CA server in the YAML file of the CA client.

If you've followed the process described in the CA deployment guide, you should have a set of folders associated with your Fabric CA client that look similar to the following:



The figure above shows the structure of folders associated with using a single Fabric CA client to connect to multiple CA servers.

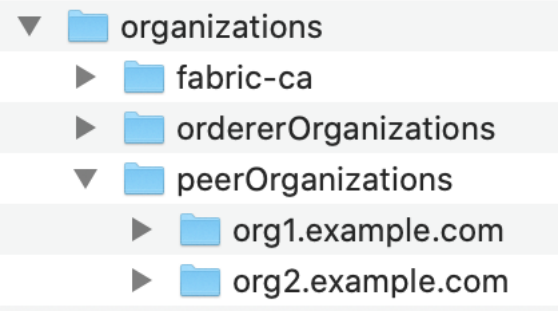
As you can see, each CA server has a separate folder underneath the `fabric-ca-client` folder. Inside each of these CA folders is an `msp` folder that contains the public/private key pair for the identity of the admin of that CA. This is the `--mspdir` you must specify when administrating the CA (for example, when registering an identity).

If you are not a CA admin, but rather have a Fabric CA client only for the purpose of enrolling with an organization CA and a TLS CA, it is still a best practice to use a single Fabric CA client. This CA client will still need TLS certificates (which can be obtained using the process described in the CA deployment guide), but you will not need to point to CA admin certificates since you are not acting as a CA admin. Instead, the enroll ID and secret given to you by the CA admin that registered the identity allows you to interact with a particular CA server and receive the necessary certificates.

Folder structure for your org and node admin identities

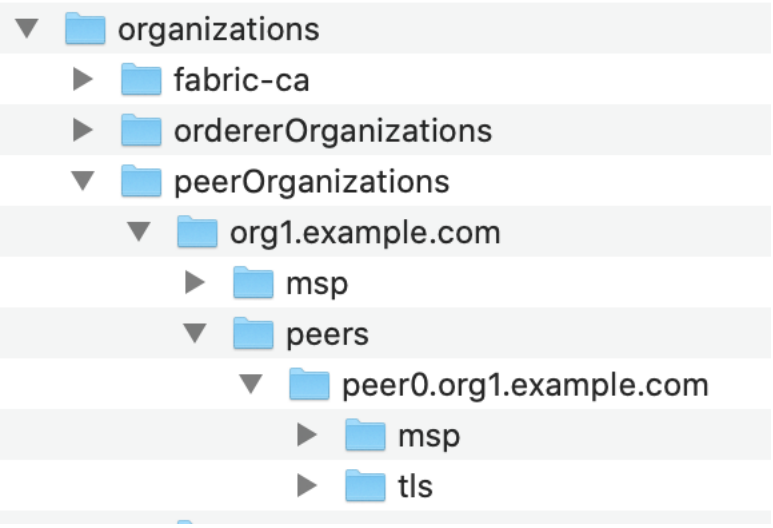
While the way you organize the folders of the CAs you operate using the Fabric CA client is determined in large part by the multiple CAs a typical CA admin will interact with, the organizational method you use to organize your organization MSPs will be determined in part by how many organizations you anticipate creating and administering.

For example, in the Fabric test network, both peer organizations and orderer organizations are created. As a result, the scripts associated with the network create a folder called `organizations` , which contains an `ordererOrganization` and a `peerOrganization` folder. Each of these folders contains a folder for each organization, which contains both an MSP for that organization and a folder for each node owned by those organizations.



The figure above shows the structure of the organizations managed by an administrator.

Even if you don't plan to create an orderer organization, this kind of structure provides the highest level of long term flexibility for your deployment. If you create a new peer, for example, you will know to create a folder at `organizations/<name of org>/<name of new peer>` . This `<name of new peer>` folder will be the location for the local MSP of the peer (generated when the peer identity is enrolled) and for the certificates generated through enrollment with the TLS CA. Similarly, the location of the MSP of the organization the peer belongs to can reference the `msp` folder of the organization (which includes both the `config.yaml` file if Node OUs are being used as well as the public certificate of the admin of the organization, which in many cases will be the admin of the peer).



The figure above shows the subfolders inside of a peer owned by the organization. Note the `msp` folder here under the `peers` folder. This is the local MSP of the peer, not a duplicate of the `org1.example.com` MSP.

It is the best practice to create these folders before enrolling identities and then referencing them when issuing the enroll command through the `--mspdir` flag. Note that while the –mspdir flag is used to specify where the MSP of the CA admin is during **registration**, it is used instead during **enrollment** to specify the location on the filesystem where the folders and certificates returned by the CA will be stored.

NodeOUs

In previous versions of Fabric, identities only had two types: `client` and `peer`. The `peer` type was used for both peers and ordering nodes, while the `client` type was used for both clients (applications) and admins, with the placement of a `client` type in a special `admincerts` folder making the identity an admin within a particular context.

Now it is possible, and recommended, to encode not just `peer` or `client`, but also `orderer` or `admin` roles into the certificates generated by a CA using NodeOUs. This embeds the role an identity has within the certificate.

Note that an identity can only have one of these roles, and that to enable these roles you must copy the relevant stanzas into a file called `config.yaml`. This `config.yaml` file is used by Fabric in different ways. In a channel MSP, it is used to verify that the `admin` of an organization has a role of `admin` (this replaces the use of an `admincerts` folder which was used in older versions of Fabric). In the local MSP of a node, it is used to verify the `admin` role of the node admin and the `peer` or `orderer` role of the node itself.

Note that you can name this `msp` folder anything you want — `msp` is the default folder name used by the Fabric CA client. If you choose another name, for example, `org1msp`, then you will have to reference this folder using the `--mspdir` flag when enrolling the identity. Programmatically, you can use a command similar to this to copy `config.yaml` to the correct `msp` folder for the identity you've enrolled.

```
echo 'NodeOUs:
 Enable: true
 ClientOUIdentifier:
   Certificate: cacerts/localhost-7054-ca-org1.pem
   OrganizationalUnitIdentifier: client
 PeerOUIdentifier:
   Certificate: cacerts/localhost-7054-ca-org1.pem
   OrganizationalUnitIdentifier: peer
 AdminOUIdentifier:
   Certificate: cacerts/localhost-7054-ca-org1.pem
   OrganizationalUnitIdentifier: admin
 OrdererOUIdentifier:
   Certificate: cacerts/localhost-7054-ca-org1.pem
   OrganizationalUnitIdentifier: orderer' > path to msp>/msp/config.yaml
```

Or you can manually copy the Node OU material into the `config.yaml` file for the `msp` folder:

```
NodeOUs:
  Enable: true
  ClientOUIdentifier:
    Certificate: cacerts/<root CA cert for this org>.pem
    OrganizationalUnitIdentifier: client
  PeerOUIdentifier:
    Certificate: cacerts/<root CA cert for this org>.pem
    OrganizationalUnitIdentifier: peer
  AdminOUIdentifier:
    Certificate: cacerts/<root CA cert for this org>.pem
    OrganizationalUnitIdentifier: admin
  OrdererOUIdentifier:
    Certificate: cacerts/<root CA cert for this org>.pem
    OrganizationalUnitIdentifier: orderer
```

In a production scenario, it is assumed that users will be creating only one organization. However, it is a good practice to establish a separate folder structure for this organization and then create a structure underneath this organization for your `msp` (defining the organization) and your nodes (which will have a local MSP and TLS sections).

If you are creating an orderer, you obviously do not need to copy the `PeerOUIdentifier` into your `config.yaml` file (or vice versa), but for the sake of simplicity you might want to use the entire section — ex-

tra stanzas do no harm, and they allow the same `config.yaml` to be used for multiple types of nodes and identities associated with an organization.

3.4.3 Register an identity

While identities that will be used by admins (or other users) and identities used by nodes have different purposes, they are fundamentally all just **identities**: public/private key pairs in which the public key is known to others and the private key is used to sign, generating an output which can be verified to have come from the private key even though the private key itself is never exposed.

As discussed above, an identity is first registered with a CA by a CA admin. This identity is then enrolled by the user of the identity. If you are using the Fabric CA client, this registration command looks like this (regardless of the type of identity you are enrolling, and the type of CA):

```
./fabric-ca-client register -d --id.name <ID_NAME> --id.secret <ID_SECRET> -u <CA_URL>
↪ --mspdir <CA_ADMIN> --id.type <ID_TYPE> --id.attrs $ID_ATTRIBUTE --tls.certfiles
↪<TLSCERT>
```

Where the variables are the following:

- `ID_NAME` : The enroll ID of the identity. This name will be given to the user out of band, who will use it when enrolling.
- `ID_SECRET` : The secret (similar to a password) for the identity. This secret will also be given along to the user along with the enroll ID to use when enrolling.
- `CA_URL` : The URL of the CA, followed by the port `7054` (unless the default port has been changed).
- `CA_ADMIN` : The path to the location of the certificates for the admin of the CA.
- `ID_TYPE` : The type (or role) of the identity. There are four possible types: `peer` , `orderer` , `admin` , and `client` (used for applications). This type must be linked to the relevant *NodeOU*. If NodeOUs are not being used, you can ignore the type and `--id.type` flag.
- `ID_ATTRIBUTE` : Any attributes specified for this identity. For more information about attributes, check out Attribute based access control. These attributes can also be added as a JSON array, therefore the `$ID_ATTRIBUTE` is not meant to represent a single attribute but any and all attributes, which should be placed in the register command after the `--id.attrs` flag.
- `TLSCERT` : The relative path to your the TLS CA root signed certificate (generated when creating the TLS CA).

Note that the `-d` flag enables debug mode, which is useful for debugging if the registration fails.

Here is a sample register command for an admin identity:

```
./fabric-ca-client register -d --id.name org1admin --id.secret org1adminpw -u https://
↪example.com:7054 --mspdir ./org1-ca/msp --id.type admin --tls.certfiles ../tls/tls-
↪ca-cert.pem --csr.hosts 'host1,*.example.com'
```

After the identity has been successfully registered, the CA admin would give the enroll ID (`org1admin`) and enroll secret (`org1adminpw`) to the user who will enroll as an admin.

If you are creating the certificates needed for a node, make sure to register and enroll with the TLS CA associated with the organization as well.

3.4.4 Enroll an identity

Once the enrollment CA has been set up and identities have been registered, the CA admin will need to contact the user who will be enrolling out of band to give them the enroll ID and secret they used when registering the identity.

Then, using this ID and secret, the user can enroll the identity using their own copy of the Fabric CA client to contact the relevant CA (which will be either an organization CA, used to create admin and node identities, or a TLS CA, used to generate the TLS certificates that nodes need). Note that if TLS has been enabled, this user will need to acquire the TLS CA root signed cert to include when enrolling.

While it's possible to enroll a node identity before enrolling an admin, it makes more sense to enroll an admin first and establish your organization's MSP before enrolling nodes (whether it's a peer or an ordering node). You certainly need to enroll an admin identity and place its certificate in the local MSP of a node before starting the node.

The command looks like this:

```
./fabric-ca-client enroll -u https://<ENROLL_ID>:<ENROLL_SECRET><@CA_URL>:<PORT> --
↪mspdir <MSP_FOLDER> --csr.hosts <CSR_HOSTNAME> --tls.certfiles $TLS_CERT
```

With these variables:

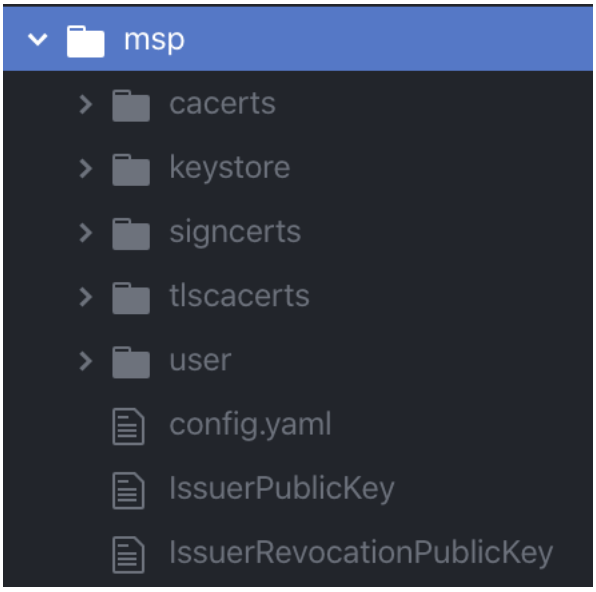
- `ENROLL_ID` : The enroll ID that was specified when registering this identity. This will have to be communicated to the user of this identity out of band.
- `ENROLL_SECRET` : The enroll secret that was specified when registering this identity. This will have to be communicated to the user of this identity out of band.
- `CA_URL` : The URL of the CA, including the port (which is 7054 by default). If you have configured two CAs at the same location, you will also have to specify a CA name following a `--caname` flag, but in this tutorial we assume you are using a configuration of CAs as specified in the [CA deployment tutorial].
- `PORT` : The port utilized by the CA you are enrolling with.
- `MSP_FOLDER` : The path to the MSP (the local MSP, if enrolling a node, or the org MSP, if enrolling an admin) on the filesystem. If you do not specify the `-mspdir` flag to specify a location, the certificates will be placed in a folder called `msp` at your current location (if this folder does not already exist, it will be created).
- `CSR_HOSTNAME` : Only relevant to node identities, this will encode the domain name of a node. For example, MagnetoCorp might choose a hostname of `peer0.mgntoorg.magnetocorp.com`.
- `TLS_CERT` : The relative path to the TLS CA root signed certificate of the TLS CA associated with this organization.

Here is an example enroll command corresponding to the example register command we used earlier:

```
./fabric-ca-client enroll -u https://org1admin:org1adminpw@example.com:7054 --mspdir .
↪/org1.example.com/msp --csr.hosts 'org1,*.example.com' --tls.certfiles ../tls/tls-
↪ca-cert.pem
```

Unlike a typical CA, in which an enrollment command will return only the public/private key pair, the Fabric CA returns a folder structure called an MSP. This MSP can then be used to create a structure that can be consumed by Fabric when creating nodes or adding organizations to a channel. In the case of enrolling an admin, the MSP forms the basis of an organization. In the case of enrolling a node identity, it forms the basis for the local MSP for the node. Note that this folder structure will also be returned by the TLS CA. However, only the relevant TLS certificates are needed.

Here is a sample of the MSP that will be returned after your enroll the identity:



The figure above shows the subfolders returned by an enrollment.

In certificate naming, it is helpful to use a convention that will help you keep track of whether you are referencing a public certificate or a private key. Given that both have the `.pem` extension, consider the following convention for naming public certs and private keys:

- Rename a public cert from `cert.pem` (which is the default name the Fabric CA will give a public cert) to something meaningful. For example, the public cert of an admin of "Org1" could be given a name like `org1-admin-cert.pem`.
- Rename a private key from `94u498f9r9fr98t49t345545345_sk` to something meaningful like `org1-admin-key.pem`.

In this convention, the last word in the name before appending the `.pem` extension would be either `cert` or `key` to help you remember which is which.

3.4.5 Create an MSP from an enrolled identity

As we have noted, enrolling an identity with the Fabric CA generates output that includes not just public/private key pairs, but a number of related folders and certificates that Fabric networks need to consume.

However, that does not mean that these folders can simply be dropped into a channel configuration (to join an org to a channel) or into the local configuration of a node (to create a local MSP). In the case of creating an org MSP that can be added to a channel, you will need to remove the private key of the admin. In the case of a local MSP, you will need to add the public certificate of an admin.

For more information about the folders and certificates that are needed in both an org MSP (also known as a "channel MSP", since it is added to a channel) and the local MSP of a node, check out MSP structure.

Create the org MSP needed to add an org to a channel

The organizations in a Fabric network don't exist in a physical sense the way nodes do. Rather, they exist as a structure of folders and certificates on the configuration of a channel. These certificates identify the relevant root CA, intermediate CA (if one was used), TLS CA, and at least one admin identity. As you recall from the membership topic and the registration and enrollment steps above, these folders and certificates are returned by the Fabric CA client

when enrolling an admin identity, which is why the act of enrolling an admin and the act of "creating an organization" are closely related.

Here is a sample of the folder structure you need to create when you want to add an organization to a channel (the structure might vary slightly depending on the method you use to add an organization to a channel, but whatever the method, these are the files and folders you will need):

```
<location of msp>/msp
-- config.yaml
-- cacerts
|   -- cacert.crt
-- intermediatecerts
|   -- cacert.crt
-- tlscacerts
|   -- tlsca.<org-domain>.pem
-- tlsintermediatecerts
    -- tlsca.<org-domain>.pem
```

Where the folders and certificates are:

- `cacerts` : the root certificate (`ca-cert.pem`) of the organization CA where the identity of the admin was registered and enrolled.
- `intermediatecerts` : the root certificate of an intermediate CA, if one was used.
- `tlscacerts` : the root certificate (`ca-cert.pem`) of the TLS CA that has issued certificates to the nodes associated with this organization.
- `tlsintermediatecerts` : the root certificate of the intermediate TLS CA, if one was used.

Note that while the certificates themselves can be named anything you want, you should not change the name of the folders themselves, as Fabric expects to consume folders with certain names.

See *NodeOUs* for instructions on how to generate the `config.yaml` file for this organization. In older versions of Fabric, the `config.yaml` file would not have been here and an additional folder, `admincerts` , would be needed, in which certificates identifying the admin of this organization would be placed. This is no longer necessary thanks to Node OUs. **Any identity given a Node OU of `admin` by the CA listed in `config.yaml` can administer the organization**.

Create the local MSP of a node

While the MSP of an organization serves as the representation of the organization on a channel configuration, the local MSP of a node is a logical collection of parameters that is used, along with other parameters, as part of the creation of a node.

As has been noted above, nodes must be bootstrapped both with enrollment certificates (the public/private key pair that identifies a node) and the TLS certificates that encrypt the communication layer between nodes. This "bootstrapping" happens by listing the location of these certificates in the relevant YAML configuration file that is referenced when creating the node. This means that **the local MSP of a node must be created before the node itself can be created**. Note that the enrollment certificates for the node are specified by listing the location of the MSP that contains them, while TLS certificates are identified through the absolute path to the location of each certificate.

For reference, here is a sample peer configuration file.

And here is a sample ordering node configuration file.

Note that these configuration files ask for the location of the relevant local MSP folder. For the peer, this is defined through the `mspConfigPath` . For the orderer, it is the `LocalMSPDir` . The folders found in this location will be used to define the local MSP of the node, including the private key the node will use when signing its actions as well as the public key of at least one admin of the node.

The TLS certificates, on the other hand, are defined individually, rather than pointing to a folder, and can be found in the `TLS settings` section of the YAML. This means that TLS certificates do not need to be kept in a strict folder structure like the local MSP (relevant in particular to users who will be using an external CA to generate TLS certificates — use the sample YAML files as a guide to what these certificates are used for). When you enroll a node with the TLS CA, the generated TLS public key can be found in the `/signcerts` folder and the TLS private key can be found in the `/keystore` folder. When you stand up a node that is enabled for TLS, you need to point to these files from the relevant fields in the YAML config file.

As with all of the configuration parameters in the YAML file of a node, you have the option to specify the `msp` folder and TLS certificate locations either in the YAML itself or through the use of environment variables.

If you are using a containerized solution for running your network (which for obvious reasons is a popular choice), **it is a best practice to mount these folders (volumes) external to the container where the node itself is running. This will allow the certificates to be used to create a new node should the node container go down, become corrupted, or is restarted.**

For a look at a sample local MSP, check out MSP structure. Note that you will not receive all of these certificates back simply by enrolling a peer identity. You will need, for example, to create the `users` subfolder and put the public certificate of the identity that will be administering the node in the folder prior to bootstrapping. You will also need an operations certificate (depending on the configuration of your network, this might come from a separate operations CA). For more information about the operations service, check out The Operations Service.

Here is a sample local MSP as it might look when the node has been enrolled and the additional fields have been added:

```
localmsp
  -- config.yaml
  -- cacerts
      -- <root CA public cert>.pem
  -- intermediatecerts
      -- <intermediate CA public cert>.pem
  -- keystore
      -- <node private cert>.pem
  -- signcerts
      -- <node public cert>.pem
  -- tlscacerts
      -- tlsca.<org-domain>.pem
  -- tlsintermediatecerts
      -- tlsca.<org-domain>.pem
  -- operationscerts
      -- operationcert.pem
```

Where the folders and certificates are:

- `cacerts` : the root cert of the organization CA where the identity of the admin was registered and enrolled.
- `intermediatecerts` : the root cert of an intermediate CA, if one was used.
- `keystore` : the private key of the node. This is the key the node uses to sign its communications.
- `signcerts` : the public key of the node. This certificate is presented to nodes making incoming communications, allowing the node initiating a communication to know that it is talking to the correct node.
- `tlscacerts` : the root cert of the TLS CA that has issued certificates to the CAs or nodes associated with this organization.
- `tlsintermediatecerts` : the root cert of the intermediate TLS CA, if one was used.
- `operationscerts` : the certificate needed for interaction with the operations service.

Note that while the certificates themselves can be named anything you want, you should not change the name of the folders themselves, as Fabric expects to consume folders with certain names.

Just as Node OUs make it no longer necessary to include a certificate of an admin in the organization MSP, it is not necessary to include the public certificate of a node admin to administer the node. **Any identity given a Node OU of `admin` by the CA listed in `config.yaml` can administer any of the nodes owned by that organization without needing to place the public certificate of that admin in the organization MSP or the local MSP**.