fabricator Documentation

Release 1.0

T-Systems MMS

Nov 11, 2021

CONTENTS

1 Feature Highlight **3**

1.1 Introduction . . . 3
1.2 Getting Started . . . 4
1.3 Tutorial . . . 5
1.4 Quickstart . . . 16
1.5 Command-Line . . . 18

Fabricator - A tool that provides a scalable generic template for bootstrapping Hyperledger Fabric networks in production environments. It can be configured for various requirements with minimal effort.

CHAPTER

ONE

FEATURE HIGHLIGHT

- setup and bootstrap the Fabric network
- scale a running Fabric network such as adding new organizations, peers and orderers
- channel query, create, join, channel update
- easy chaincode deployment with a script ready to be used with Jenkins for chaincode CI/CD
- chaincode install, approve, instantiation, invoke and query
- Hyperledger Explorer support in order to monitor the network
- certificate reenrollment and revocation

Note: This project is under active development.

1.1 Introduction

The Hyperledger Fabric network can get too complicated to set up and scale. Organizations that want to participate in a consortium or in a permissioned network generally would want that bootstrapping and managing such a network should be a simple and hassle-free task. They would want to bring up the network and its components without understanding and being involved in depth of the technical details. Fabricator enables this.

1.1.1 What is Fabricator?

Fabricator is a tool that can setup and bootstrap the fabric network, bring up organizations and make them join the network with just a single command. It provides a generic template that contains script that are tailored in a way which makes it simple for organizations to configure the network according to their own needs. It enables scaling a running fabric network by allowing the dynamic addition of new network components, users and organizations. It is well suited for production environments since it uses the Fabric-CA server for certificate generation, rotation and revocation.

1.1.2 How does Fabricator works?

In order to understand how Fabricator works, it is essential to understand its design. Currently it is designed for 3 organizations and it can be easily scaled to more organizations.

Certificate authorities for organizations and orderers

In order to generate the certificates for the orderers of all organizations there is a single *Orderer CA Server* container running, called `ca-orderer.t-systems.com` . This is the first container that should be up when we bootstrap the network. This Orderer CA Server generates the crypto material and subsequently the certificates for the orderers of the first organization using the configuration present in the `fabric-ca-server-config.yaml` which is located inside the folder `fabric-orderer-ca`. The crypto material along with the certificates is also generated inside the folder `fabric-orderer-ca`. After this, `fabric-orderer-ca` is copied to every new organization, which means that the Orderer CA Server uses the crypto material of the first organization in order to generate the certificates for the orderers of each organization. This explains why we have a single Orderer CA Server for the orderers of all organizations. Each organization has their own CA server that is responsible to generate their certificates. Therefore, there is a CA Server container running for every organization and should be the first container that should be up when we bootstrap the organization. For example for an `organization 1` we have the *organization CA Server* container called `ca-org.organization1.t-systems.com`. Since in our current tutorial is for 3 organizations, this means there are 3 CA's for 3 organizations in a similar fashion as described above.

General design

Each organization has its own containers and can be deployed separately in their own machines. All the services/containers are divided in domains of their parent organization i.e. `peer.org1.example.com` , `peer.org2.example.com` etc except for orderers. Because of fabric design the orderers can not be divided in domains and they all belong to their own membership service provider i.e. OrdererMSP. Each organization has reservation of 10 orderers each. In our example Org1 has reservation from 0 to 9, Org2 has reservation from 10 to 19 and Org3 has reservation from 20 to 29 and so on and so forth. Orgs have their own base orderer (orderer0. for Org1, orderer1. for Org2 etc). The organizations join the system channel using their respective orderers and can also add their own local orderer nodes (maximum 10). Initially each organization has 1 orderer, 2 peers ; More peers and orderers can be dynamically added by following the steps below. We have provided demos to play around with this tool in 2 ways: Local setup: Setup a local 3 organization network on a single machine. Multi-machine setup: Setup a multi-organization network running on separate machines. A basic guide and tutorial for both these ways are explained below.

1.2 Getting Started

1.2.1 Prerequisites

- docker (install)
- docker-compose (install)
- curl (install)
- yq (install)

1.2.2 Download Fabric dependencies

Fabricator is compatible and fully tested with the Hyperledger Fabric network v2.0.1. Fabric Dependencies can be downloaded to start a new fabric network v2.0.1 from scratch using this command:

```
$ ./fabric-dependencies.sh -s -- <fabric_version> <fabric-ca_version> <thirdparty_
↪version>
$ ./fabric-dependencies.sh -s -- 2.0.1 1.4.6 0.4.18
```

More information about installing the Hyperledger Fabric samples and binaries can be found here.

1.3 Tutorial

1.3.1 Generate organizations

Before we go ahead creating either a local or multi-machine network, we have to first create individual scripts for each organization. To do this easily, we have created a script called generate-org.sh which copies the material from the `./fabric-template` folder and parametrizes it according to each organization.

Make sure you are present in the root directory of the repository:

```
└── ./blockchain-common-fabric-net
        ├── ./blockchain-common-fabric-net/deploy-chaincode.sh
        ├── ./blockchain-common-fabric-net/fabric-dependencies.sh
        ├── ./blockchain-common-fabric-net/fabric-template
        ├── ./blockchain-common-fabric-net/generate-org.sh
        ├── ./blockchain-common-fabric-net/local-startup.sh
        ├── ./blockchain-common-fabric-net/readme.md
        └── ./blockchain-common-fabric-net/remote-startup.sh
```

The command to create organization material can be run as:

```
$ ./generate-org.sh <ORG_NAME> <DOMAIN_NAME> <ORG_NUMBER> <ORG_EXPLORER_PORT>
```

After running the above command, all organization directories should be created inside the `./generated-orgs` directory:

```
└── ./blockchain-common-fabric-net
├── ./blockchain-common-fabric-net/generated-orgs
```

As an example we generate an organization called the MMS:

```
$ ./generate-org.sh MMS t-systems.com 1 7040
```

Therefore you should have created the `./MMS` directory along with the organization scripts that looks like this:

```
generated-orgs
└── generated-orgs/MMS
        ├── generated-orgs/MMS/base
        ├── generated-orgs/MMS/chaincode
        ├── generated-orgs/MMS/configtx.yaml
        ├── generated-orgs/MMS/connection-profile
        ├── generated-orgs/MMS/docker-compose-explorer.yaml
```

(continues on next page)

(continued from previous page)

```
├── generated-orgs/MMS/docker-compose-new-peer.yaml
├── generated-orgs/MMS/docker-compose-new-raft-orderer.yaml
├── generated-orgs/MMS/docker-compose-orderer-ca.yaml
├── generated-orgs/MMS/docker-compose-org-ca.yaml
├── generated-orgs/MMS/docker-compose.yaml
├── generated-orgs/MMS/explorer-config.json
├── generated-orgs/MMS/fabric-network.sh
├── generated-orgs/MMS/fabric-orderer-ca
├── generated-orgs/MMS/fabric-org-ca
├── generated-orgs/MMS/initial-configtx.yaml
└── generated-orgs/MMS/scripts
```

You can create as many organizations you want in the similar fashion as mentioned above.

Note: Once organzations have been generated, each organization can be manipulated using the `fabric-network.sh` script present inside each organzation folder as shown in the example above for the organization called **MMS**.

1.3.2 Adding organizations

This is a basic tutorial on setting up 3 organizations, use the following commands:

Note: The commands to add and publish a remote orderer can be run as given in the example below, where <ORDERER_NO> is the orderer number of the organization. Please refer to the **Command-Line** Section for more details.

Example:

```
./fabric-network.sh add-remote-orderer <ORDERER_NO>

./fabric-network.sh publish-remote-orderer <ORDERER_NO>
```

Organization 1

To read all the help regarding commands use: `./fabric-network.sh help.`

Generate the crypto material for *organization 1* and start it.

```
$ ./fabric-network.sh generate-crypto

$ ./fabric-network.sh up
```

Note: Before you bootstrap *organization 2* or any of the following organizations there is something you should know. There is only one *CA* (**Certificate Authority**) for the orderers of all organizations. Crypto material is generated after the **Fabric CA** for the orderer of the first organization (*organization 1*) runs. This crypto material is generated inside the folder `fabric-orderer-ca` of *organization 1* and should be copied to each *organization* as the orderers of all organizations need this crypto material in order to register and enroll themselves to the **Orderer CA** so that the required certificates needed to identify the orderer of each organization can be generated. However, every organizaton has its own **CA** that generates some crypto material stored in the `fabric-org-ca`. This crypto material is used to generate the MSP of each organization. *Peers*, *users* and *admins* of each organization can register and enroll themselves to the running

Organization CA that generates certificates for them. Since each organization has its own **CA**, so `fabric-org-ca` folder is present inside each organization at the time it is created.

Copy the `fabric-orderer-ca` folder from the root of *Organization 1* to the root of *Organization 2* and *Organization 3* . Once the crypto material for the orderer certificates is placed as required run following commands.

Organization 2

```
$ /fabric-network.sh generate-crypto
```

The above command generates 2 important files for bootstrapping (1) A json file i.e. `./channel-artifacts/Org2.json` that contains the Org2 MSP certificates required to join this Org to any channel at any time (2) A crt file i.e. `./channel-artifacts/orderer10.crt` that contains public certificates of Org2's base orderer required to add this base orderer into system channel to bootstrap As a next step copy the `./channel-artifacts/orderer10.crt` into channel-artifacts folder of Org1 so that it can bootstrap Org2's base orderer in system channel

```
$ ./fabric-network.sh add-remote-orderer 10
```

Since we are bootstrapping *Organization 2's base orderer* it is 10, it should be 20 for *organization 3* and so on.. As a result of this command for *base orderer 10* a genesis file will be generated in `channel-artifacts` folder i.e. `./channel-artifacts/orderer_genesis.pb` of *organization 1* This generates a genesis file would have been generated in `./channel-artifacts/orderer_genesis.pb` copy this file to `./Org2/channel-artifacts/orderer_genesis.pb`.

Next step: Copy the `channel-artifacts/orderer_genesis.pb` **of organization 1 into** `channel-artifacts/orderer_genesis.pb` **of the organization 2.**

Then run the following command to start the containers:

```
$ ./fabric-network.sh up
```

After the containers of *organization 2* are up, publish it's orderer details by running the following command:

```
$ ./fabric-network.sh publish-remote-orderer 10
```

The above command ensures that the base orderer of *organization 2* is published and now peers can contact this as an active orderer in the network. This command must be run after the containers of *organization 2* are up and running.

Organization 3

```
$ ./fabric-network.sh generate-crypto
```

This command generates 2 important files for bootstrapping (1) A json file i.e. `./channel-artifacts/Org3.json` that contains the Org3 MSP certificates required to join this Org to any channel at any time (2) A crt file i.e. `./channel-artifacts/orderer20.crt` that contains public certificates of Org3's base orderer required to add this base orderer into system channel to bootstrap As a next step copy the `./channel-artifacts/orderer20.crt` into channel-artifacts folder of Org2 so that it can bootstrap Org3's base orderer in system channel

To Bootstrap Org3 from Org2:

```
$ ./fabric-network.sh add-remote-orderer 20
```

In above command 20 refers to orderer number. Since we are bootstrapping Org3's base orderer it is 20, it should be 30 for Org4 and so on.. As a result of this command for base orderer20, a genesis file will be generated in `channel-artifacts` folder i.e. `channel-artifacts/orderer_genesis.pb`

Next step: Copy the `channel-artifacts/orderer_genesis.pb` **into** `channel-artifacts/orderer_genesis.pb` **of the Org3**

In the above step, a genesis file would have been generated in `Org2/channel-artifacts/orderer_genesis.pb` copy this file to `Org3/channel-artifacts/orderer_genesis.pb` and then run the following command to start the containers

```
$ ./fabric-network.sh up
```

After the containers of Org3 are up, publish it's orderer details by running the command in the following step:

```
$ ./fabric-network.sh publish-remote-orderer 20
```

The above command ensures that the base orderer of Org3 is published and now peers can contact this as an active orderer in the network. This command must be run after the containers of Org3 are up and running.

Note: You can follow the steps given above in this tutorial in the similar way to set up upto *N organizations*.

1.3.3 Channel Creation & Joining

This is a basic tutorial on creating and joining on a channel.

Note: This tutorial currently demonstrates an example of 3 organizations. However the same steps shall be followed for up to *N organizations*. The commands to *create and join a channel, add and sign the configuration of another organization* can be run as given below in the example.

Example:

To create a channel:

```
./fabric-network.sh create-channel <CHANNEL_PROFILE> <CHANNEL_NAME>
```

To join a peer to a channel:

```
./fabric-network.sh join-channel-peer <PEER_NO> <CHANNEL_NAME>
```

To add configuration of another organization:

```
./fabric-network.sh add-org-config <CHANNEL_NAME> <ORG_TO_BE_ADDED_NAME>
```

To sign configuration of another organization:

```
./fabric-network.sh add-org-sign <CHANNEL_NAME> <ORG_TO_BE_SIGNED_NAME>
```

Build channel configuration via configtx.yaml

A channel is created by building a channel transaction that specifies the initial configuration of the channel. The channel configuration has the information about the channel members(organizations), ordering nodes that can add new blocks to the channel and the policies that govern the channel updates. A channel is created by using the `configtx.yaml` file and the `configtxgen` tool. The `configtx.yaml` file contains the information that is required to build the channel configuration. The `configtxgen` tool reads the information in the `configtx.yaml` file and writes it in a special format that can be read by Fabric. In a channel, changes and updates are need to be approved by majority of channel members(organizations). How this approval would take place is in the policies defined inside the channel section of the `configtx.yaml`. As it can be seen above that we use the argument <CHANNEL_PROFILE> and <CHANNEL_NAME> in the commands to create and join a channel and to add a new channel member(organization) to the channel. The channel profiles are read by the `configtxgen` to build a channel configuration. Each profile uses YAML syntax to gather data from other sections of the file. The configtxgen tool uses this configuration to create a channel creation transaction for an applications channel, or to write the channel genesis block for a system channel.

A detailed documentation about the chanel configration using `configtx.yaml` file and the `configtxgen` tool. can be found here.

As shown below, we define and use a channel profile called `ChannelAll`.

```
Channel: &ChannelDefaults
    Policies:
        Readers:
            Type: ImplicitMeta
            Rule: "ANY Readers"
        Writers:
            Type: ImplicitMeta
            Rule: "ANY Writers"
        Admins:
            Type: ImplicitMeta
            Rule: "MAJORITY Admins"
    Capabilities:
        <<: *ChannelCapabilities

Profiles:

    OrdererGenesis:
        <<: *ChannelDefaults
        Orderer:
            <<: *OrdererDefaults
            OrdererType: etcdraft
            EtcdRaft:
                Consenters:
                - Host: orderer0.t-systems.com
                Port: 7050
                ClientTLSCert: crypto-config/ordererOrganizations/t-systems.com/orderers/
↪orderer0.t-systems.com/tls/server.crt
                ServerTLSCert: crypto-config/ordererOrganizations/t-systems.com/orderers/
↪orderer0.t-systems.com/tls/server.crt
            Addresses:
                - orderer0.t-systems.com:7050
            Organizations:
            - *OrdererOrg
            Capabilities:
```

(continues on next page)

(continued from previous page)

```
                <<: *OrdererCapabilities
        Consortiums:
            BaseConsortium:
                Organizations:
                    - *MMS
    ChannelAll:
        Consortium: BaseConsortium
        <<: *ChannelDefaults
        Capabilities:
            <<: *ChannelCapabilities
        Orderer:
            <<: *OrdererDefaults
            OrdererType: etcdraft
            EtcdRaft:
                Consenters:
                - Host: orderer0.t-systems.com
                Port: 7050
                ClientTLSCert: crypto-config/ordererOrganizations/t-systems.com/orderers/
↪orderer0.t-systems.com/tls/server.crt
                ServerTLSCert: crypto-config/ordererOrganizations/t-systems.com/orderers/
↪orderer0.t-systems.com/tls/server.crt
            Addresses:
                - orderer0.t-systems.com:7050
            Organizations:
            - *OrdererOrg
            Capabilities:
                <<: *OrdererCapabilities
        Application:
            <<: *ApplicationDefaults
            Organizations:
                - *MMS
            Capabilities:
                <<: *ApplicationCapabilities
```

Organization 1

To create a channel run following command. This commands creates the channel from the anchor peer i.e. of *oganization 1* and also make it join this new channel.

```
$ ./fabric-network.sh create-channel ChannelAll channelall
```

Organization 2

To Add Org 2 configuration in the channel copy `Org2.json` file in channel-artifacts folder of Org1 and run following commands. Usually `add-org-config` and `add-org-sign` are supposed to run from 2 different Orgs (depending upon channel configuration in `configtx.yaml`), but in this case since we only have Org1 currently added in this channel we can make an exception and add & sign both from Org1. But it would not be possible if there are more than 1 Orgs on a particular channel.

```
$ ./fabric-network.sh add-org-config channelall Org2

$ ./fabric-network.sh add-org-sign channelall Org2
```

The channel channelall has been updated with the new organization and new genesis block is now added in `./channel-artifacts/channelall.block` file Copy this file `./channel-artifacts/channelall.block` in new organization's channel-artifacts and join this channel from Org2

To join a channel by anchor peer i.e. peer0 of Org2 whose configuration is already added and signed, copy `.block` file from Org1 in `channel-artifacts` folder and run the following command. You can run this command for each peer of Org2 to join the channel by changing peer in the argument.

```
$ ./fabric-network.sh join-channel peer0 channelall
```

Organization 3

To Add Org 3 configuration in the channel copy `Org3.json` file in channel-artifacts folder of Org2 and run following commands. Now `add-org-config` and `add-org-sign` would from 2 different Orgs (depending upon channel configuration in `configtx.yaml`), as now there are more than 2 Orgs on channelall.

```
$ ./fabric-network.sh add-org-config channelall Org3
```

(1) The new organization configuration for this channel is exported in `channel-artifacts/Org3_update_in_envelope.pb` file

(2) Copy `channel-artifacts/Org3_update_in_envelope.pb` file in `channel-artifacts` folder of any other Org in this channel i.e. Org1

(3) run the command `./fabric-network add-org-sign` from any other organization on this channel to sign this configuration and commit to ledger i.e. Org1

To Sign Org 3 configuration added by Org2 in step 5, copy `channel-artifacts/Org3_update_in_envelope.pb` file in `channel-artifacts` folder and run following command

```
$ ./fabric-network.sh add-org-sign channelall Org3
```

The channel channelall has been updated with the new organization and new genesis block is now added in `./channel-artifacts/channelall.block` file. Copy this file `./channel-artifacts/channelall.block` in new organization's `channel-artifacts` folder and join this channel from anchor peer cli.

To join a channel by anchor peer i.e. peer0 of Org3 whose configuration is already added and signed, copy `.block` file from Org1 in `channel-artifacts` folder and run the following command. You can run this command for each peer of Org3 to join the channel by changing peer in the argument.

```
$ ./fabric-network.sh join-channel peer0 channelall
```

1.3.4 Adding Peers

The following commands can run from any organization as required:

To add a new peer

```
$ ./fabric-network.sh add-peer
```

Note: If need be, you can join this newer peer to the existing channel by changing the peer number using the command given below.

To join the newer peer to existing channel:

```
./fabric-network.sh join-channel-peer <PEER_NO> <CHANNEL_NAME>
```

For example if you want join *peer number 2* to the channel called *channelall*

```
$ ./fabric-network.sh join-channel-peer 2 channelall
```

1.3.5 Adding Orderers

The following commands can run from any organization as required:

To add a new orderer

```
$ ./fabric-network.sh add-local-orderer
```

Note: If need be, you can join this newer orderer to the existing channel using the command given below, by changing the orderer number as required.

To join the newer peer to existing channel:

```
./fabric-network.sh join-channel-orderer <ORDERER_NO> <CHANNEL_NAME>
```

For example if you want join *orderer number 1* to the channel called *channelall*

```
$ ./fabric-network.sh join-channel-orderer 1 channelall
```

1.3.6 Chaincode Deployment & Invocation

Our chaincode needs to be deployed on the channel by one participant and the others need to approve it before it can be committed. The number of participants whose approval is needed is dependent on the endorsement policy, it could be `ANY`, `MAJORITY` or `ALL`.

For our example, we are approving via every organization. The chaincode example that is being deployed is `fabcar` which needs to be present in the `chaincode` folder inside each organization. This `chaincode` folder is mounted in docker container.

To get help regarding the commands use: ./fabric-network.sh help

Organization 1

To package the chaincode

```
$ ./fabric-network.sh package-cc fabcar golang 1
```

To install the chaincode

```
$ ./fabric-network.sh install-cc fabcar
```

To query the installed chaincode [OPTIONAL]

```
$ ./fabric-network.sh query-installed-cc
```

Copy the package id you get here. It will be used in the the next commands

To approve a chaincode

```
$ ./fabric-network.sh query-installed-cc

Update the package id (long string) with the package id you get in previous command

$ ./fabric-network.sh approve-cc channelall fabcar 1␣
↪1:a413310bd764d0e4bfdbe988646b6081f6fcc80c865abd51a1cbc4b570a5feb2 1
```

To check commit readiness [OPTIONAL]

```
$ ./fabric-network.sh checkcommitreadiness-cc channelall fabcar 1 1 json
```

To commit a chaincode

```
$ ./fabric-network.sh commit-cc channelall fabcar 1 1
```

This command would fail if you haven't got required approvals from the organizations

To query a committed chaincode [OPTIONAL]

```
$ ./fabric-network.sh query-committed-cc channelall
```

To initialize a chaincode

```
$ ./fabric-network.sh init-cc channelall fabcar
```

To invoke the fabcar chaincode function

```
$ ./fabric-network.sh invoke-function-cc channelall fabcar initLedger
```

To query the fabcar chaincode function [OPTIONAL]

```
$ ./fabric-network.sh query-function-cc channelall fabcar queryAllCars
```

To invoke a fabcar chaincode function that changes the car owner

```
$ ./fabric-network.sh invoke-function-cc channelall fabcar changeCarOwner \"CAR9\",\
↪"XOXO\"
```

Invoke functions can be called from any organizations and all other orgs can see the state changes

To query whether the state change has been reflected [OPTIONAL]

```
$ ./fabric-network.sh query-function-cc channelall fabcar queryAllCars
```

Organization 2

To package the chaincode

```
$ ./fabric-network.sh package-cc fabcar golang 1
```

To install the chaincode

```
$ ./fabric-network.sh install-cc fabcar
```

To query the installed chaincode [OPTIONAL]

```
$ ./fabric-network.sh query-installed-cc
```

Copy the package id you get here. It will be used in the the next commands

To approve a chaincode

```
$ ./fabric-network.sh approve-cc channelall fabcar 1␣
↪1:a413310bd764d0e4bfdbe988646b6081f6fcc80c865abd51a1cbc4b570a5feb2 1
```

To check commit readiness [OPTIONAL]

```
$ ./fabric-network.sh checkcommitreadiness-cc channelall fabcar 1 1 json
```

Update the package id (long string) with the package id you get in previous command

To query a committed chaincode [OPTIONAL]

```
$ ./fabric-network.sh query-committed-cc channelall
```

To query the fabcar chaincode function [OPTIONAL]

```
$ ./fabric-network.sh query-function-cc channelall fabcar queryAllCars
```

To query whether the state change has been reflected [OPTIONAL]

```
$ ./fabric-network.sh query-function-cc channelall fabcar queryAllCars
```

Organization 3

To package the chaincode

```
$ ./fabric-network.sh package-cc fabcar golang 1
```

To install the chaincode

```
$ ./fabric-network.sh install-cc fabcar
```

To query the installed chaincode [OPTIONAL]

```
$ ./fabric-network.sh query-installed-cc
```

Use the package id you get here in the next commands.

To approve a chaincode

```
$ ./fabric-network.sh approve-cc channelall fabcar 1␣
↪1:a413310bd764d0e4bfdbe988646b6081f6fcc80c865abd51a1cbc4b570a5feb2 1
```

To check commit readiness [OPTIONAL]

```
$ ./fabric-network.sh checkcommitreadiness-cc channelall fabcar 1 1 json
```

Update the package id (long string) with the package id you get in previous command

To query a committed chaincode [OPTIONAL]

```
$ ./fabric-network.sh query-committed-cc channelall
```

To query the fabcar chaincode function [OPTIONAL]

```
$ ./fabric-network.sh query-function-cc channelall fabcar queryAllCars
```

To query whether the state change has been reflected [OPTIONAL]

```
$ ./fabric-network.sh query-function-cc channelall fabcar queryAllCars
```

1.3.7 Certificate reenrollment & revocation

A permissioned blockchain requires that an entity, be it a network user (client), an admin, or a network component (peers or orderers), must be identified and permissioned before accessing a consortium network. Every entity in the network is granted some digital certificates in order for them to be able to identify themselves as an identity in the network. These digital certificates are generated along with some other crypto material. There are two ways to generate crypto materials in Hyperledger Fabric: **Cryptogen and CA (Certificate Authority) Server**.

Fabricator uses the CA server, particularly the Hyperledger Fabric CA, in order to generate the crypto material. A detailed documentation about the Fabric CA can be found in Fabric CA User's Guide.

Note: With respect to the arguments <identity> and <identity_no> for the commands given below, an identity can be a *user, admin, peer* or an *orderer* for which you may want to reenroll or revoke certificates. Since Fabricator allows you to add as many *users, admins*, new *peers* you may want and up to *10 orderers* dynamically, the identity_no is to indicate which *identity* you want to address. For example: *peer 1* or *orderer 6*.

Renew enrollment certificate

In case if the enrollment certificate has been expired or compromised, it can be renewed using the following command:

```
$ ./fabric-network.sh reenroll-certificate <identity> <identity_no>
```

Revoke a certificate or identity

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. An identity or a certificate can be revoked using the following command:

```
$ ./fabric-network.sh revoke-certificate <identity> <identity_no>
```

1.3.8 Down & Cleanup

The following commands can run from any organization as required:

To bring down, cleanup and restore the initial state of configuration files please run the following command

```
$ ./fabric-network.sh down cleanup restore
```

1.4 Quickstart

Our example includes 3 organizations, each generates their crypto material individually stored in their independent folders (in case of local setup)or in their independent machines (in case of multi-machine setup).

Each organization joins the channel and deploy chaincode on it. We will make each organization join our network one by one.

The containers in this example can be deployed in single machine containing all 3 orgs or 3 distinct machines (1 for each org) which are in the same network.

1.4.1 Single machine for all Organizations

For single machine we need to open 3 terminals and change directory each terminal in the path of each organization. Then we need to create a bridge network using docker upon which this blockchain would be deployed.

The network name in this example is *fabric-template-network*. This can also be configured in the `fabric-network.sh` file

```
docker network create fabric-template-network
```

A demo of single machine for all Orgs is given in `local-startup.sh` file. This `local-startup.sh` file uses the individual `fabric-network.sh` files of each organization that was created by `generate-org` command described above. Before using `local-startup.sh` please do the following two things:

1.Make sure that the fabric-network.sh files have executable permissions

```
$ chmod +x generated-orgs/{ORG_NAME}/fabric-network.sh
```

2.Open the `local-startup.sh` file and at the top make sure to change the variables {Org1}, {Org2} and {Org3} according to the names of your organizations.

The actual shell tool `./fabric-network.sh` can also be well understood by reading the `local-startup.sh`

A basic network of 1 Orderer, 2 peers for each Org can be bootstrapped and hardcoded channels can be joined in this demo in simple steps.

Run the below command in order to run basic network with 3 organizations.

```
$ ./local-startup.sh bootstrap
```

Once 3 organizations have been bootstrapped then we can create and join the channels in which these organizations can participate. We have added hardcoded profiles `ChannelAll` and `ChannelAll2` in `configtx.yaml` files of organizations respectively and these channels can be created from their respective organizations and subsequently be joined by other organizations. In simple steps as demonstrated below.

```
$ ./local-startup.sh create-join-channel channelall

$ ./local-startup.sh create-join-channel channelall2
```

The channel profiles can be configured to have different rules upon which the channel will operate, these rules are very well explained in fabric docs. In our example the profiles `ChannelAll` and `ChannelAll2` have been configure to have a `"Majority"` consensus for Admin operations, hence in order to add a new organization to these channel we need majority organizations in the channel to sign and submit to ledger. In order to make it possible we have kept all 3 Orgs as ordering organizations.

As a counter example we have another channel profile in Org1 i.e. `ChannelAllANY`. This profiles configures the channel such that `"ANY"` organization can do Admin operations and hence the requirement of majority is relaxed. Since we have bootstrapped all 3 Orgs as ordering Orgs and the requirements of channel are relaxed so we can also use this profile to create channels in our on going example.

In short: `channelall`, `channelall2` => `Majority` organizations need to sign `channelallany` => `ANY` organization needs to sign

```
$ ./local-startup.sh create-join-channel channelallany
```

At this point, you have achieved to create a local network with 3 organizations joined by two channels.

If you do not wish to delve into multi-machine network right now, please skip to chaincode installation section to see how chaincode installation works. Otherwise, please carry on.

1.4.2 Organizations in a distributed setup

Now, we are moving forward to part 2 of this tutorial i.e. creation of a multi-machine network.

In this case we would have multiple machines where each machine would run the containers of 1 Org. So for 3 Orgs we would have 3 machines, depicting a realworld setting where each organization would run it's own nodes in it's own physical infrastructure. In order to communicate with each other, the machines need an overlay network. In our example, we are using a docker swarm overlay network for this purpose.

The machines need to join a docker swarm as manager for equal control. One organization would initialize the docker swarm using:

```
docker swarm init
```

This machine is the first manager of docker swarm. Running this command at base manager gives you exact command that can be used by other machines to join this docker swarm infrastructure.

```
docker swarm join-token manager
```

Now run the resulting command at each each machine for them to join this docker swarm as manager.

Once all the machines are docker swarm managers we can create a shared overlay network that all of them would use for this blockchain. Run this command from any of the machines.

```
docker network create --driver overlay --attachable {fabric-template-network}
```

This network can be verified at all machines using

```
docker network ls
```

Now that the *fabric-template-network* network has been set up at single machine/multiple machines we can bootstrap our fabric network.

1.4.3 Easy chaincode deployments with CI/CD

For easy chaincode deployments, we have created a script (`deploy-chaincode.sh`) to easily approve and deploy chaincode. The script has to be run once for each organization (in parallel). Only 1 organization would commit the chaincode (we call this organization `COMMITTER_OF_CHAINCODE`), all the organizations would only approve. This script was written to be used with Jenkins for chaincode CI/CD but can be used for local/remote installations as well by running the script with appropriate parameters.

If you want to do all the steps manually one by one yourself rather than using our script, please skip to the next section.

Please open 3 terminal instances and cd into the folder where we have `deploy-chaincode.sh` file.

```
$ ./deploy-chaincode.sh {COMMITTER_OF_CHAINCODE} {ORGANIZATION_RUNNING_SCRIPT} {CHANNEL_
↪NAME} {VERSION} {SEQUENCE}
```

For this example, please run the commands for the 3 orgs like this (either on the same machine or on different machines):

```
$ ./deploy-chaincode.sh {Org1} {Org1} channelall 1 1

$ ./deploy-chaincode.sh {Org1} {Org2} channelall 1 1

$ ./deploy-chaincode.sh {Org1} {Org3} channelall 1 1
```

Please make sure that the committer of chaincode is the same in all 3 orgs since this organization would be committing the chaincode, the rest will only approve.

1.5 Command-Line

Welcome to Fabric setup utility

The command works as shown below:

```
./fabric-network.sh COMMAND_NAME <ARGS>
```

1. To generate crypto material for this organization use:

   ```
   ./fabric-network.sh generate-crypto
   ```

2. To bring up the organization:

```
./fabric-network.sh up
```

3. To add config of another organization:

```
./fabric-network.sh add-org-config <CHANNEL_NAME> <ORG_TO_BE_ADDED_NAME>
```

4. To sign config of another organization:

```
./fabric-network.sh add-org-sign <CHANNEL_NAME> <ORG_TO_BE_SIGNED_NAME>
```

5. To create a channel:

```
./fabric-network.sh create-channel <CHANNEL_PROFILE> <CHANNEL_NAME>
```

6. To join a peer to a channel:

```
./fabric-network.sh join-channel-peer <PEER_NO> <CHANNEL_NAME>
```

7. To add another peer:

```
./fabric-network.sh add-peer
```

8. To add another local orderer of an organization:

```
./fabric-network.sh add-local-orderer
```

9. To add orderer to a channel:

```
./fabric-network.sh join-channel-orderer <ORDERER_NO> <CHANNEL_NAME>
```

10. To add remote orderer of another organization:

```
./fabric-network.sh add-remote-orderer <ORDERER_NO>
```

11. To publish remote orderer of another organization:

```
./fabric-network.sh publish-remote-orderer <ORDERER_NO>
```

12. To package a chaincode:

```
./fabric-network.sh package-cc <CHAINCODE_NAME> <CHAINCODE_LANGUAGE> <CHAINCODE_
↪LABEL>
```

13. To install a chaincode:

```
./fabric-network.sh install-cc <CHAINCODE_NAME>
```

14. To query whether a chaincode has installed:

```
./fabric-network.sh query-installed-cc
```

15. To approve a chaincode from your organization:

```
./fabric-network.sh approve-cc <CHANNEL_NAME> <CHAINCODE_NAME> <VERSION> <PACKAGE_
↪ID> <SEQUENCE>
```

16. To check commit-readiness of a chaincode:

```
./fabric-network.sh checkcommitreadiness-cc <CHANNEL_NAME> <CHAINCODE_NAME>
↪<VERSION> <SEQUENCE> <OUTPUT>
```

17. To commit a chaincode:

```
./fabric-network.sh commit-cc <CHANNEL_NAME> <CHAINCODE_NAME> <VERSION> <SEQUENCE>
```

18. To query committed chaincodes on a channel:

```
./fabric-network.sh query-committed-cc <CHANNEL_NAME>
```

19. To initialize a chaincode:

```
./fabric-network.sh init-cc <CHANNEL_NAME> <CHAINCODE_NAME>
```

20. To invoke a chaincode:

```
./fabric-network.sh invoke-function-cc <CHANNEL_NAME> <CHAINCODE_NAME> <FUNCTION>
↪<ARGS>
```

21. To query a chaincode:

```
./fabric-network.sh query-function-cc <CHANNEL_NAME> <CHAINCODE_NAME> <ARGS>
```

22. To start explorer:

```
./fabric-network.sh bootstrap-explorer
```

23. To down explorer:

```
./fabric-network.sh explorer-down
```

24. To display help:

```
./fabric-network.sh help
```

25. To shut down the organization and cleanup:

```
./fabric-network.sh down cleanup
```

26. To reenroll certificates

```
./fabric-network.sh reenroll-certificate <identity> <identity_no>
```

27. To revoke certificates

```
./fabric-network.sh revoke-certificate <identity> <identity_no>
```