# Setting up the Hyperledger Fabric network

This lab manual will provide an in-depth setup of how to build a fabric network from scratch. We will try to set up a network similar to **test-network** available in **fabric-samples**. The fabric-samples consist of a collection of tested examples of Hyperledger Fabric applications. The test-network is a sample network provided by Hyperledger Fabric, which comes with a 2-org single peer consortium with a single orderer.

The fabric-samples can be downloaded using the following commands:

```
curl -sSLO https://raw.githubusercontent.com/hyperledger/fabric/main/scripts/install-fabric.sh && chmod +x install-fabric.sh
```

```
./install-fabric.sh -f 2.5.4 -c 1.5.7
```

Copy the binaries to usr/local/bin

```
sudo cp fabric-samples/bin/* /usr/local/bin
```

The test-network architecture is given below:

No of Orgs: 2

Org1 - 1 peer (peer0.org1.example.com)

Org2 - 1 peer (peer0.org2.example.com)

Ordering Service: 1 orderer  (orderer. example.com)

Database Type - CouchDB

Certificate Authority : Separate CA for org1, org2 and orderer.

We will be trying to set up a network similar to test-network. Before we move on to an advanced view of how the network is being set up, let's brief the steps involved in the process.

1. Starting the network
    a. Crypto material generation (fabric-ca, CA for each org)
    b. Bringing up the components (2 peers, 2 CouchDB, 1 orderer)
    c. Generate the genesis block
    d. Creating channel
    e. Joining the peers to the created channel
    f. Anchor peer update
2. Deploying the chaincode
    a. Package the chaincode
    b. Install the packaged chaincode to selected peers
    c. Approve chaincode with chaincode definition
    d. Commit the chaincode to the channel

## Creating the folder structure

Create a folder **Fabric-network**, all the configurations and docker-compose files will be inside this folder.

```
mkdir Fabric-network

cd Fabric-network
```

## Generating the crypto materials using fabric-ca

Fabric-ca is a certificate authority used by Hyperledger Fabric to provide and manage identities.

There are two binaries provided for fabric-ca:

**fabric-ca-server** - Uses the fabric-ca-server-config.yaml to start. Acts as the ca server.

**fabric-ca-client** - A client to interact with the ca server to register, enroll and revoke identities.

In our case, we will use different ca's for the organizations. We will be specifying the services for the ca server inside a docker-compose-ca.yaml file. Inside the **Fabric-network** folder, create a folder named **docker**, to keep all docker-compose files.

**mkdir docker**

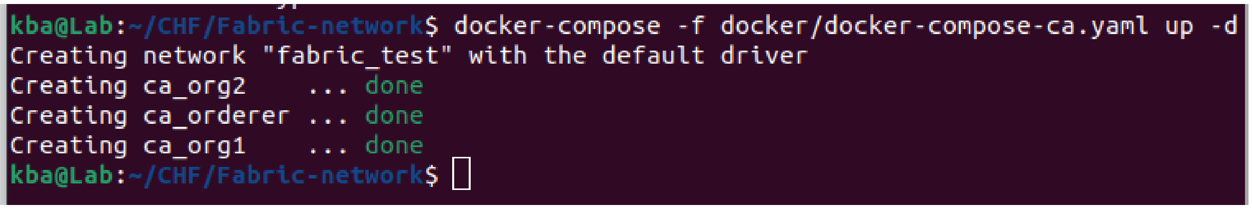
Within the docker folder, create a file named **docker-compose-ca.yaml**.  Here we will specify the 3 organization ca's as services under the fabric_test network. (Refer: fabric-samples/test-network/compose/compose-ca.yaml). Add the following to the file.

```
networks:
  test:
    name: fabric_test

services:

  ca_org1:
    image: hyperledger/fabric-ca:latest
    labels:
      service: hyperledger-fabric
    environment:
      - FABRIC_CA_HOME=/etc/hyperledger/fabric-ca-server
      - FABRIC_CA_SERVER_CA_NAME=ca-org1
      - FABRIC_CA_SERVER_TLS_ENABLED=true
      - FABRIC_CA_SERVER_PORT=7054
      - FABRIC_CA_SERVER_OPERATIONS_LISTENADDRESS=0.0.0.0:17054
    ports:
      - "7054:7054"
      - "17054:17054"
    command: sh -c 'fabric-ca-server start -b admin:adminpw -d'
    volumes:
      -
../organizations/fabric-ca/org1:/etc/hyperledger/fabric-ca-server
    container_name: ca_org1
    networks:
      - test

  ca_org2:
    image: hyperledger/fabric-ca:latest
    labels:
      service: hyperledger-fabric
    environment:
      - FABRIC_CA_HOME=/etc/hyperledger/fabric-ca-server
      - FABRIC_CA_SERVER_CA_NAME=ca-org2
      - FABRIC_CA_SERVER_TLS_ENABLED=true
      - FABRIC_CA_SERVER_PORT=8054
      - FABRIC_CA_SERVER_OPERATIONS_LISTENADDRESS=0.0.0.0:18054
    ports:
      - "8054:8054"
      - "18054:18054"
    command: sh -c 'fabric-ca-server start -b admin:adminpw -d'
    volumes:
      -
../organizations/fabric-ca/org2:/etc/hyperledger/fabric-ca-server
    container_name: ca_org2
    networks:
      - test

  ca_orderer:
    image: hyperledger/fabric-ca:latest
    labels:
      service: hyperledger-fabric
    environment:
      - FABRIC_CA_HOME=/etc/hyperledger/fabric-ca-server
      - FABRIC_CA_SERVER_CA_NAME=ca-orderer
      - FABRIC_CA_SERVER_TLS_ENABLED=true
      - FABRIC_CA_SERVER_PORT=9054
      - FABRIC_CA_SERVER_OPERATIONS_LISTENADDRESS=0.0.0.0:19054
    ports:
      - "9054:9054"
      - "19054:19054"
    command: sh -c 'fabric-ca-server start -b admin:adminpw -d'
    volumes:
      -
../organizations/fabric-ca/ordererOrg:/etc/hyperledger/fabric-ca-serv
er
    container_name: ca_orderer
    networks:
      - test
```

Remember to pass the appropriate certificate folder in the volumes to avoid errors. Observing the env and volume mappings, the FABRIC_CA_HOME env is the location where the fabric-ca-server-config.yaml for a particular organization lies, and it is set to **/etc/hyperledger/fabric-ca-server.** The **fabric-ca-server start** command is used to start the fabric-ca-server.

To start the docker-compose file, you can use the following command (Note: Execute the command from Fabric-network folder)

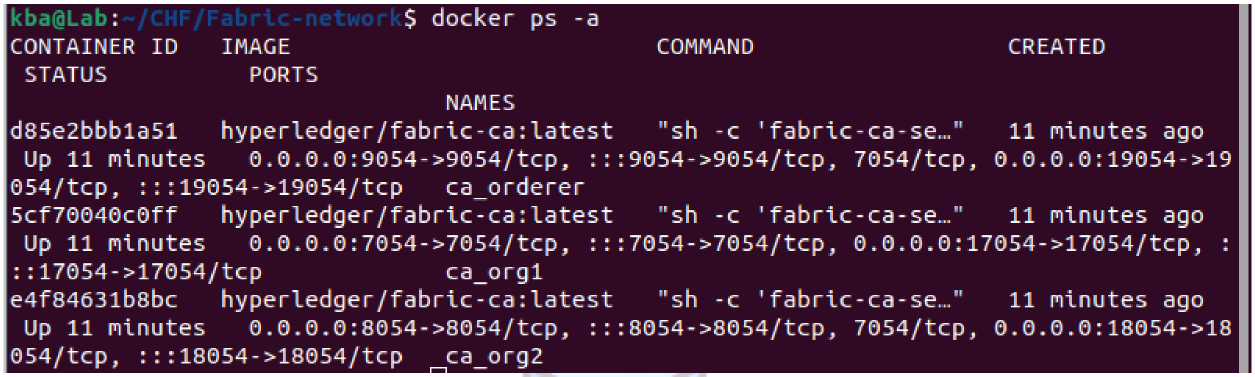
**docker compose -f docker/docker-compose-ca.yaml up -d**



```
kba@Lab:~/CHF/Fabric-network$ docker-compose -f docker/docker-compose-ca.yaml up -d
Creating network "fabric_test" with the default driver
Creating ca_org2     ... done
Creating ca_orderer ... done
Creating ca_org1     ... done
kba@Lab:~/CHF/Fabric-network$ 
```

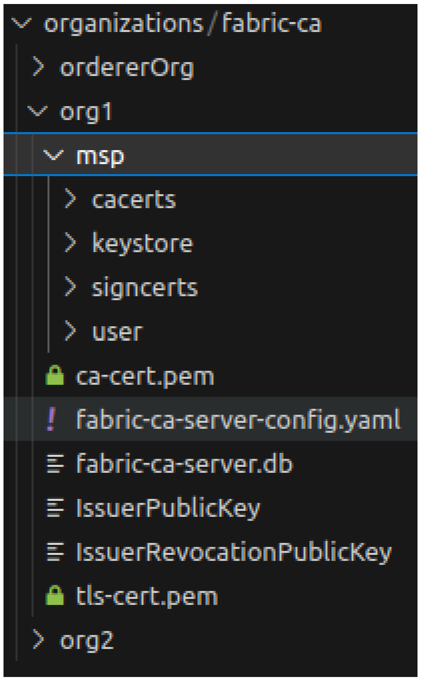
After executing the command, we can see that the ca server containers are up and running. You can check it using the command

**docker ps -a**



```
kba@Lab:~/CHF/Fabric-network$ docker ps -a
CONTAINER ID   IMAGE                               COMMAND                    CREATED
 STATUS          PORTS
                              NAMES
d85e2bbb1a51   hyperledger/fabric-ca:latest   "sh -c 'fabric-ca-se…"   11 minutes ago
 Up 11 minutes   0.0.0.0:9054->9054/tcp, :::9054->9054/tcp, 7054/tcp, 0.0.0.0:19054->19
054/tcp, :::19054->19054/tcp   ca_orderer
5cf70040c0ff   hyperledger/fabric-ca:latest   "sh -c 'fabric-ca-se…"   11 minutes ago
 Up 11 minutes   0.0.0.0:7054->7054/tcp, :::7054->7054/tcp, 0.0.0.0:17054->17054/tcp, :
::17054->17054/tcp             ca_org1
e4f84631b8bc   hyperledger/fabric-ca:latest   "sh -c 'fabric-ca-se…"   11 minutes ago
 Up 11 minutes   0.0.0.0:8054->8054/tcp, :::8054->8054/tcp, 7054/tcp, 0.0.0.0:18054->18
054/tcp, :::18054->18054/tcp   ca_org2
```

Also, each organization's fabric-ca-server-config file, certificates and other related files will be available under the **organizations/fabric-ca** folder.



```
organizations / fabric-ca
  ordererOrg
  org1
    msp
      cacerts
      keystore
      signcerts
      user
    ca-cert.pem
    fabric-ca-server-config.yaml
    fabric-ca-server.db
    IssuerPublicKey
    IssuerRevocationPublicKey
    tls-cert.pem
  org2
```

If the organizations folder is locked, then unlock it using:

**sudo chmod -R 777 organizations/**

After we start the ca, it’s time to generate the required identities using the fabric-ca-server. For adding new users to the network, fabric-ca uses two steps:

**Register** - The fabric-ca-server will register a new user and share the enrollment id and secret with the user.

**Enroll** - The user needs to enroll themselves using the fabric-ca-client using the provided enrollment id and secret by interacting with the fabric-ca-server.

Let’s write a script file for registering and enrolling the various identities and organizing the certificates in particular folders. Create a file **registerEnroll.sh** in Fabric-network folder and add the following (Refer: fabric-samples/test-network/organizations/fabric-ca/registerEnroll.sh)

```
#!/bin/bash
```

```
function createOrg1() {
  echo "Enrolling the CA admin"
  mkdir -p organizations/peerOrganizations/org1.example.com/

  export
FABRIC_CA_CLIENT_HOME=${PWD}/organizations/peerOrganizations/org1.exa
mple.com/

  set -x
  fabric-ca-client enroll -u https://admin:adminpw@localhost:7054
--caname ca-org1 --tls.certfiles
"${PWD}/organizations/fabric-ca/org1/ca-cert.pem"
  { set +x; } 2>/dev/null

  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' >
"${PWD}/organizations/peerOrganizations/org1.example.com/msp/config.y
aml"

  # Since the CA serves as both the organization CA and TLS CA, copy
the org's root cert that was generated by CA startup into the org
level ca and tlsca directories

  # Copy org1's CA cert to org1's /msp/tlscacerts directory (for use
in the channel MSP definition)
  mkdir -p
```

```
"${PWD}/organizations/peerOrganizations/org1.example.com/msp/tlscacer
ts"
  cp "${PWD}/organizations/fabric-ca/org1/ca-cert.pem"
"${PWD}/organizations/peerOrganizations/org1.example.com/msp/tlscacer
ts/ca.crt"

  # Copy org1's CA cert to org1's /tlsca directory (for use by
clients)
  mkdir -p
"${PWD}/organizations/peerOrganizations/org1.example.com/tlsca"
  cp "${PWD}/organizations/fabric-ca/org1/ca-cert.pem"
"${PWD}/organizations/peerOrganizations/org1.example.com/tlsca/tlsca.
org1.example.com-cert.pem"

  # Copy org1's CA cert to org1's /ca directory (for use by clients)
  mkdir -p
"${PWD}/organizations/peerOrganizations/org1.example.com/ca"
  cp "${PWD}/organizations/fabric-ca/org1/ca-cert.pem"
"${PWD}/organizations/peerOrganizations/org1.example.com/ca/ca.org1.e
xample.com-cert.pem"

  echo "Registering peer0"
  set -x
  fabric-ca-client register --caname ca-org1 --id.name peer0
--id.secret peer0pw --id.type peer --tls.certfiles
"${PWD}/organizations/fabric-ca/org1/ca-cert.pem"
  { set +x; } 2>/dev/null

  echo "Registering user"
  set -x
  fabric-ca-client register --caname ca-org1 --id.name user1
--id.secret user1pw --id.type client --tls.certfiles
"${PWD}/organizations/fabric-ca/org1/ca-cert.pem"
  { set +x; } 2>/dev/null

  echo "Registering the org admin"
  set -x
  fabric-ca-client register --caname ca-org1 --id.name org1admin
```

```
--id.secret org1adminpw --id.type admin --tls.certfiles
"${PWD}/organizations/fabric-ca/org1/ca-cert.pem"
  { set +x; } 2>/dev/null

  echo "Generating the peer0 msp"
  set -x
  fabric-ca-client enroll -u https://peer0:peer0pw@localhost:7054
--caname ca-org1 -M
"${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.
org1.example.com/msp" --tls.certfiles
"${PWD}/organizations/fabric-ca/org1/ca-cert.pem"
  { set +x; } 2>/dev/null

  cp
"${PWD}/organizations/peerOrganizations/org1.example.com/msp/config.y
aml"
"${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.
org1.example.com/msp/config.yaml"

  echo "Generating the peer0-tls certificates, use --csr.hosts to
specify Subject Alternative Names"
  set -x
  fabric-ca-client enroll -u https://peer0:peer0pw@localhost:7054
--caname ca-org1 -M
"${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.
org1.example.com/tls" --enrollment.profile tls --csr.hosts
peer0.org1.example.com --csr.hosts localhost --tls.certfiles
"${PWD}/organizations/fabric-ca/org1/ca-cert.pem"
  { set +x; } 2>/dev/null

  # Copy the tls CA cert, server cert, server keystore to well known
file names in the peer's tls directory that are referenced by peer
startup config
  cp
"${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.
org1.example.com/tls/tlscacerts/"*
"${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.
org1.example.com/tls/ca.crt"
```

```
  cp
"${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.
org1.example.com/tls/signcerts/"*
"${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.
org1.example.com/tls/server.crt"
  cp
"${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.
org1.example.com/tls/keystore/"*
"${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.
org1.example.com/tls/server.key"

  echo "Generating the user msp"
  set -x
  fabric-ca-client enroll -u https://user1:user1pw@localhost:7054
--caname ca-org1 -M
"${PWD}/organizations/peerOrganizations/org1.example.com/users/User1@
org1.example.com/msp" --tls.certfiles
"${PWD}/organizations/fabric-ca/org1/ca-cert.pem"
  { set +x; } 2>/dev/null

  cp
"${PWD}/organizations/peerOrganizations/org1.example.com/msp/config.y
aml"
"${PWD}/organizations/peerOrganizations/org1.example.com/users/User1@
org1.example.com/msp/config.yaml"

  echo "Generating the org admin msp"
  set -x
  fabric-ca-client enroll -u
https://org1admin:org1adminpw@localhost:7054 --caname ca-org1 -M
"${PWD}/organizations/peerOrganizations/org1.example.com/users/Admin@
org1.example.com/msp" --tls.certfiles
"${PWD}/organizations/fabric-ca/org1/ca-cert.pem"
  { set +x; } 2>/dev/null

  cp
"${PWD}/organizations/peerOrganizations/org1.example.com/msp/config.y
aml"
```

```
"${PWD}/organizations/peerOrganizations/org1.example.com/users/Admin@
org1.example.com/msp/config.yaml"
}

function createOrg2() {
  echo "Enrolling the CA admin"
  mkdir -p organizations/peerOrganizations/org2.example.com/

  export
FABRIC_CA_CLIENT_HOME=${PWD}/organizations/peerOrganizations/org2.exa
mple.com/

  set -x
  fabric-ca-client enroll -u https://admin:adminpw@localhost:8054
--caname ca-org2 --tls.certfiles
"${PWD}/organizations/fabric-ca/org2/ca-cert.pem"
  { set +x; } 2>/dev/null

  echo 'NodeOUs:
  Enable: true
  ClientOUIdentifier:
    Certificate: cacerts/localhost-8054-ca-org2.pem
    OrganizationalUnitIdentifier: client
  PeerOUIdentifier:
    Certificate: cacerts/localhost-8054-ca-org2.pem
    OrganizationalUnitIdentifier: peer
  AdminOUIdentifier:
    Certificate: cacerts/localhost-8054-ca-org2.pem
    OrganizationalUnitIdentifier: admin
  OrdererOUIdentifier:
    Certificate: cacerts/localhost-8054-ca-org2.pem
    OrganizationalUnitIdentifier: orderer' >
"${PWD}/organizations/peerOrganizations/org2.example.com/msp/config.y
aml"

  # Since the CA serves as both the organization CA and TLS CA, copy
the org's root cert that was generated by CA startup into the org
level ca and tlsca directories
```

```
  # Copy org2's CA cert to org2's /msp/tlscacerts directory (for use
in the channel MSP definition)
  mkdir -p
"${PWD}/organizations/peerOrganizations/org2.example.com/msp/tlscacer
ts"
  cp "${PWD}/organizations/fabric-ca/org2/ca-cert.pem"
"${PWD}/organizations/peerOrganizations/org2.example.com/msp/tlscacer
ts/ca.crt"

  # Copy org2's CA cert to org2's /tlsca directory (for use by
clients)
  mkdir -p
"${PWD}/organizations/peerOrganizations/org2.example.com/tlsca"
  cp "${PWD}/organizations/fabric-ca/org2/ca-cert.pem"
"${PWD}/organizations/peerOrganizations/org2.example.com/tlsca/tlsca.
org2.example.com-cert.pem"

  # Copy org2's CA cert to org2's /ca directory (for use by clients)
  mkdir -p
"${PWD}/organizations/peerOrganizations/org2.example.com/ca"
  cp "${PWD}/organizations/fabric-ca/org2/ca-cert.pem"
"${PWD}/organizations/peerOrganizations/org2.example.com/ca/ca.org2.e
xample.com-cert.pem"

  echo "Registering peer0"
  set -x
  fabric-ca-client register --caname ca-org2 --id.name peer0
--id.secret peer0pw --id.type peer --tls.certfiles
"${PWD}/organizations/fabric-ca/org2/ca-cert.pem"
  { set +x; } 2>/dev/null

  echo "Registering user"
  set -x
  fabric-ca-client register --caname ca-org2 --id.name user1
--id.secret user1pw --id.type client --tls.certfiles
"${PWD}/organizations/fabric-ca/org2/ca-cert.pem"
  { set +x; } 2>/dev/null
  echo "Registering the org admin"
  set -x
  fabric-ca-client register --caname ca-org2 --id.name org2admin
--id.secret org2adminpw --id.type admin --tls.certfiles
"${PWD}/organizations/fabric-ca/org2/ca-cert.pem"
  { set +x; } 2>/dev/null

  echo "Generating the peer0 msp"
  set -x
  fabric-ca-client enroll -u https://peer0:peer0pw@localhost:8054
--caname ca-org2 -M
"${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.
org2.example.com/msp" --tls.certfiles
"${PWD}/organizations/fabric-ca/org2/ca-cert.pem"
  { set +x; } 2>/dev/null

  cp
"${PWD}/organizations/peerOrganizations/org2.example.com/msp/config.y
aml"
"${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.
org2.example.com/msp/config.yaml"

  echo "Generating the peer0-tls certificates, use --csr.hosts to
specify Subject Alternative Names"
  set -x
  fabric-ca-client enroll -u https://peer0:peer0pw@localhost:8054
--caname ca-org2 -M
"${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.
org2.example.com/tls" --enrollment.profile tls --csr.hosts
peer0.org2.example.com --csr.hosts localhost --tls.certfiles
"${PWD}/organizations/fabric-ca/org2/ca-cert.pem"
  { set +x; } 2>/dev/null

  # Copy the tls CA cert, server cert, server keystore to well known
file names in the peer's tls directory that are referenced by peer
startup config
  cp
```

```
"${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.
org2.example.com/tls/tlscacerts/"*
"${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.
org2.example.com/tls/ca.crt"
  cp
"${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.
org2.example.com/tls/signcerts/"*
"${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.
org2.example.com/tls/server.crt"
  cp
"${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.
org2.example.com/tls/keystore/"*
"${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.
org2.example.com/tls/server.key"

  echo "Generating the user msp"
  set -x
  fabric-ca-client enroll -u https://user1:user1pw@localhost:8054
--caname ca-org2 -M
"${PWD}/organizations/peerOrganizations/org2.example.com/users/User1@
org2.example.com/msp" --tls.certfiles
"${PWD}/organizations/fabric-ca/org2/ca-cert.pem"
  { set +x; } 2>/dev/null

  cp
"${PWD}/organizations/peerOrganizations/org2.example.com/msp/config.y
aml"
"${PWD}/organizations/peerOrganizations/org2.example.com/users/User1@
org2.example.com/msp/config.yaml"

  echo "Generating the org admin msp"
  set -x
  fabric-ca-client enroll -u
https://org2admin:org2adminpw@localhost:8054 --caname ca-org2 -M
"${PWD}/organizations/peerOrganizations/org2.example.com/users/Admin@
org2.example.com/msp" --tls.certfiles
"${PWD}/organizations/fabric-ca/org2/ca-cert.pem"
  { set +x; } 2>/dev/null
  cp
"${PWD}/organizations/peerOrganizations/org2.example.com/msp/config.y
aml"
"${PWD}/organizations/peerOrganizations/org2.example.com/users/Admin@
org2.example.com/msp/config.yaml"
}

function createOrderer() {
  echo "Enrolling the CA admin"
  mkdir -p organizations/ordererOrganizations/example.com

  export
FABRIC_CA_CLIENT_HOME=${PWD}/organizations/ordererOrganizations/examp
le.com

  set -x
  fabric-ca-client enroll -u https://admin:adminpw@localhost:9054
--caname ca-orderer --tls.certfiles
"${PWD}/organizations/fabric-ca/ordererOrg/ca-cert.pem"
  { set +x; } 2>/dev/null

  echo 'NodeOUs:
  Enable: true
  ClientOUIdentifier:
    Certificate: cacerts/localhost-9054-ca-orderer.pem
    OrganizationalUnitIdentifier: client
  PeerOUIdentifier:
    Certificate: cacerts/localhost-9054-ca-orderer.pem
    OrganizationalUnitIdentifier: peer
  AdminOUIdentifier:
    Certificate: cacerts/localhost-9054-ca-orderer.pem
    OrganizationalUnitIdentifier: admin
  OrdererOUIdentifier:
    Certificate: cacerts/localhost-9054-ca-orderer.pem
    OrganizationalUnitIdentifier: orderer' >
"${PWD}/organizations/ordererOrganizations/example.com/msp/config.yam
l"
```

```
  # Since the CA serves as both the organization CA and TLS CA, copy the org's root cert that was generated by CA startup into the org level ca and tlsca directories

  # Copy orderer org's CA cert to orderer org's /msp/tlscacerts directory (for use in the channel MSP definition)
  mkdir -p "${PWD}/organizations/ordererOrganizations/example.com/msp/tlscacerts"
  cp "${PWD}/organizations/fabric-ca/ordererOrg/ca-cert.pem" "${PWD}/organizations/ordererOrganizations/example.com/msp/tlscacerts/tlsca.example.com-cert.pem"

  # Copy orderer org's CA cert to orderer org's /tlsca directory (for use by clients)
  mkdir -p "${PWD}/organizations/ordererOrganizations/example.com/tlsca"
  cp "${PWD}/organizations/fabric-ca/ordererOrg/ca-cert.pem" "${PWD}/organizations/ordererOrganizations/example.com/tlsca/tlsca.example.com-cert.pem"

  echo "Registering orderer"
  set -x
  fabric-ca-client register --caname ca-orderer --id.name orderer --id.secret ordererpw --id.type orderer --tls.certfiles "${PWD}/organizations/fabric-ca/ordererOrg/ca-cert.pem"
  { set +x; } 2>/dev/null

  echo "Registering the orderer admin"
  set -x
  fabric-ca-client register --caname ca-orderer --id.name ordererAdmin --id.secret ordererAdminpw --id.type admin --tls.certfiles "${PWD}/organizations/fabric-ca/ordererOrg/ca-cert.pem"
  { set +x; } 2>/dev/null

  echo "Generating the orderer msp"
  set -x
  fabric-ca-client enroll -u https://orderer:ordererpw@localhost:9054
--caname ca-orderer -M
"${PWD}/organizations/ordererOrganizations/example.com/orderers/order
er.example.com/msp" --tls.certfiles
"${PWD}/organizations/fabric-ca/ordererOrg/ca-cert.pem"
  { set +x; } 2>/dev/null

  cp
"${PWD}/organizations/ordererOrganizations/example.com/msp/config.yam
l"
"${PWD}/organizations/ordererOrganizations/example.com/orderers/order
er.example.com/msp/config.yaml"

  echo "Generating the orderer-tls certificates, use --csr.hosts to
specify Subject Alternative Names"
  set -x
  fabric-ca-client enroll -u https://orderer:ordererpw@localhost:9054
--caname ca-orderer -M
"${PWD}/organizations/ordererOrganizations/example.com/orderers/order
er.example.com/tls" --enrollment.profile tls --csr.hosts
orderer.example.com --csr.hosts localhost --tls.certfiles
"${PWD}/organizations/fabric-ca/ordererOrg/ca-cert.pem"
  { set +x; } 2>/dev/null

  # Copy the tls CA cert, server cert, server keystore to well known
file names in the orderer's tls directory that are referenced by
orderer startup config
  cp
"${PWD}/organizations/ordererOrganizations/example.com/orderers/order
er.example.com/tls/tlscacerts/"*
"${PWD}/organizations/ordererOrganizations/example.com/orderers/order
er.example.com/tls/ca.crt"
  cp
"${PWD}/organizations/ordererOrganizations/example.com/orderers/order
er.example.com/tls/signcerts/"*
"${PWD}/organizations/ordererOrganizations/example.com/orderers/order
er.example.com/tls/server.crt"
```

```
  cp
"${PWD}/organizations/ordererOrganizations/example.com/orderers/order
er.example.com/tls/keystore/"*
"${PWD}/organizations/ordererOrganizations/example.com/orderers/order
er.example.com/tls/server.key"

  # Copy orderer org's CA cert to orderer's /msp/tlscacerts directory
(for use in the orderer MSP definition)
  mkdir -p
"${PWD}/organizations/ordererOrganizations/example.com/orderers/order
er.example.com/msp/tlscacerts"
  cp
"${PWD}/organizations/ordererOrganizations/example.com/orderers/order
er.example.com/tls/tlscacerts/"*
"${PWD}/organizations/ordererOrganizations/example.com/orderers/order
er.example.com/msp/tlscacerts/tlsca.example.com-cert.pem"

  echo "Generating the admin msp"
  set -x
  fabric-ca-client enroll -u
https://ordererAdmin:ordererAdminpw@localhost:9054 --caname
ca-orderer -M
"${PWD}/organizations/ordererOrganizations/example.com/users/Admin@ex
ample.com/msp" --tls.certfiles
"${PWD}/organizations/fabric-ca/ordererOrg/ca-cert.pem"
  { set +x; } 2>/dev/null

  cp
"${PWD}/organizations/ordererOrganizations/example.com/msp/config.yam
l"
"${PWD}/organizations/ordererOrganizations/example.com/users/Admin@ex
ample.com/msp/config.yaml"
}

createOrg1
createOrg2
createOrderer
```

The registerEnroll.sh script file consists of the commands for the following:

1. Create appropriate folders for each organization so we can specify each org's folder while enrolling.

   Eg:
   ```
   mkdir -p organizations/peerOrganizations/org1.example.com/
   ```

2. Specify the FABRIC_CA_CLIENT_HOME variable so that enrolling a user will generate the required identities on the specified folder.

   Eg:
   ```
   export
   FABRIC_CA_CLIENT_HOME=${PWD}/organizations/peerOrganizations/org1.example.com/
   ```

3. Enroll the CA admin for each organization using the **fabric-ca-client enroll** command. The command will enroll the admin and place the certificates on the path of FABRIC_CA_CLIENT_HOME.

   Eg:
   ```
   fabric-ca-client enroll -u https://admin:adminpw@localhost:7054
   --caname ca-org1 --tls.certfiles
   "${PWD}/organizations/fabric-ca/org1/ca-cert.pem"
   ```

   In the command, -u refers to the URL of the fabric-ca-server with enrollment-id and enrollment secret, --tls.certfiles contains the path of the tls cert files (if TLS is enabled).

4. Define the NodeOUs for each organization by generating a config.yaml inside the MSP folder.
5. Register the peer0, user and org admin for each organization using the **fabric-ca-client register** command. During the registration process, we will pass the enrollment id (id.name) and secret(id.secret).

Eg: fabric-ca-client register --caname ca-org1 --id.name peer0 --id.secret peer0pw --id.type peer --tls.certfiles "${PWD}/organizations/fabric-ca/org1/ca-cert.pem"

6. Generate the msp folder with the necessary certificates for peer0, user and org admin using the **fabric-ca-client enroll** command.

   Eg: fabric-ca-client enroll -u https://peer0:peer0pw@localhost:7054 --caname ca-org1 -M "${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/msp" --tls.certfiles "${PWD}/organizations/fabric-ca/org1/ca-cert.pem"

   Here, -M specifies the path where the certificates need to be stored.

7. Generate the tls certificates for peer0 using the **fabric-ca-client enroll** command.

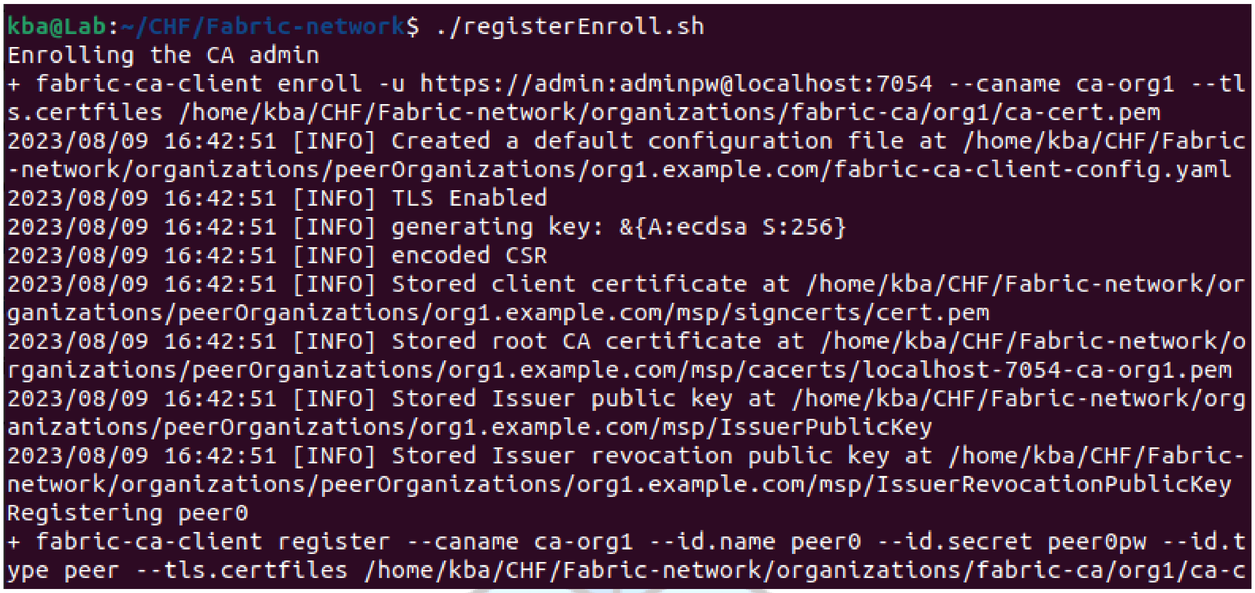
   Eg: fabric-ca-client enroll -u https://peer0:peer0pw@localhost:7054 --caname ca-org1 -M "${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls" --enrollment.profile tls --csr.hosts peer0.org1.example.com --csr.hosts localhost --tls.certfiles "${PWD}/organizations/fabric-ca/org1/ca-cert.pem"
8. Copy the nodeOus config.yaml file to the admin MSP.
9. Categorize the certificates by creating a separate folder for tls, ca, tls ca certs using the **mkdir** and **cp** command.

Now run the **registerEnroll.sh** file. First, make the script file executable using the command:

**chmod +x registerEnroll.sh**

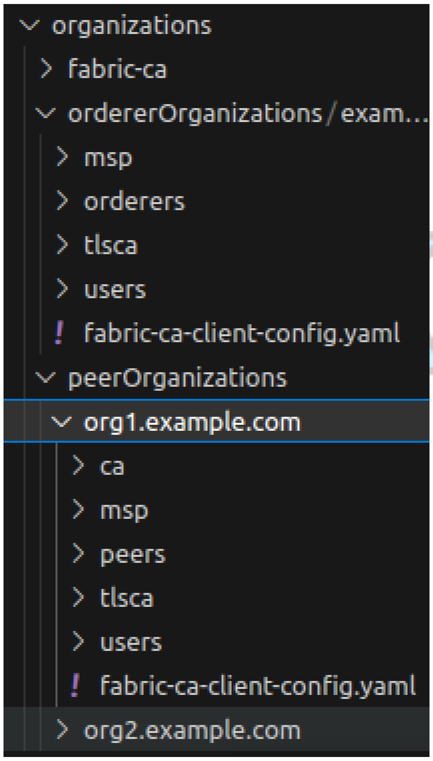
Run the script file using the command:

### ./registerEnroll.sh



```
kba@Lab:~/CHF/Fabric-network$ ./registerEnroll.sh
Enrolling the CA admin
+ fabric-ca-client enroll -u https://admin:adminpw@localhost:7054 --caname ca-org1 --tl
s.certfiles /home/kba/CHF/Fabric-network/organizations/fabric-ca/org1/ca-cert.pem
2023/08/09 16:42:51 [INFO] Created a default configuration file at /home/kba/CHF/Fabric
-network/organizations/peerOrganizations/org1.example.com/fabric-ca-client-config.yaml
2023/08/09 16:42:51 [INFO] TLS Enabled
2023/08/09 16:42:51 [INFO] generating key: &{A:ecdsa S:256}
2023/08/09 16:42:51 [INFO] encoded CSR
2023/08/09 16:42:51 [INFO] Stored client certificate at /home/kba/CHF/Fabric-network/or
ganizations/peerOrganizations/org1.example.com/msp/signcerts/cert.pem
2023/08/09 16:42:51 [INFO] Stored root CA certificate at /home/kba/CHF/Fabric-network/o
rganizations/peerOrganizations/org1.example.com/msp/cacerts/localhost-7054-ca-org1.pem
2023/08/09 16:42:51 [INFO] Stored Issuer public key at /home/kba/CHF/Fabric-network/org
anizations/peerOrganizations/org1.example.com/msp/IssuerPublicKey
2023/08/09 16:42:51 [INFO] Stored Issuer revocation public key at /home/kba/CHF/Fabric-
network/organizations/peerOrganizations/org1.example.com/msp/IssuerRevocationPublicKey
Registering peer0
+ fabric-ca-client register --caname ca-org1 --id.name peer0 --id.secret peer0pw --id.t
ype peer --tls.certfiles /home/kba/CHF/Fabric-network/organizations/fabric-ca/org1/ca-c
```

It creates the respective folders and certificates within the organizations folder.



```
∨ organizations
  > fabric-ca
  ∨ ordererOrganizations / exam...
    > msp
    > orderers
    > tlsca
    > users
    ! fabric-ca-client-config.yaml
  ∨ peerOrganizations
    ∨ org1.example.com
      > ca
      > msp
      > peers
      > tlsca
      > users
      ! fabric-ca-client-config.yaml
    > org2.example.com
```

## Bringing up the components

Since we are trying to start a fabric network, it is necessary to simulate multiple hosts running different components such as peer, orderer, ca etc. We will be using docker for this purpose, but to manage(start, stop, restart, remove) these containers we will be using docker-compose. We will write a docker-compose file to start all the necessary components.

The docker-compose file will have 6 services.

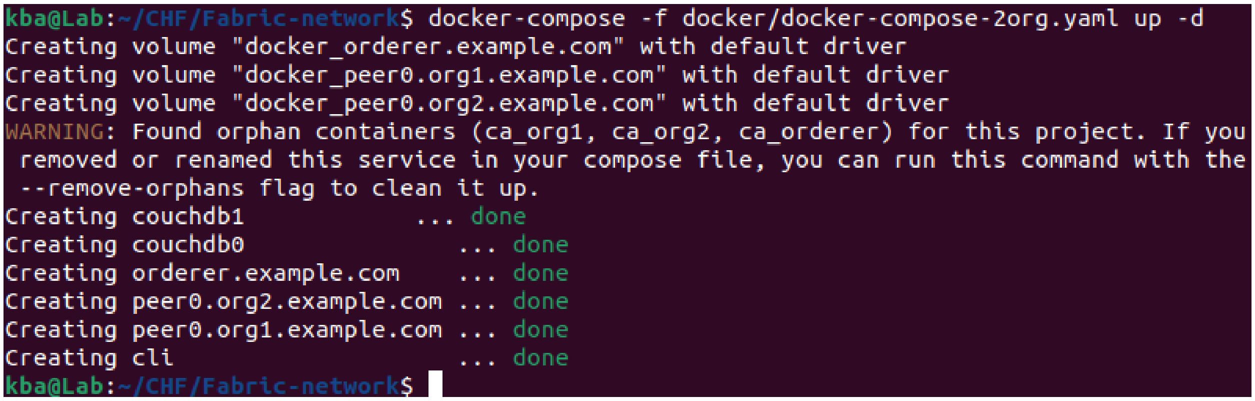
- 2 organization peers ( 1 peer for each org)
- 2 couch db ( 1 for each peers)
- A single node orderer
- A cli

Inside the docker folder, create a file **docker-compose-2org.yaml** and add the following. (Refer: fabric-samples/test-network/compose/compose-test-net.yaml, fabric-samples/test-network/compose/compose-couch.yaml)

```
version: '3.7'

volumes:
  orderer.example.com:
  peer0.org1.example.com:
  peer0.org2.example.com:

networks:
  test:
    name: fabric_test

services:

  orderer.example.com:
    container_name: orderer.example.com
    image: hyperledger/fabric-orderer:latest
    labels:
      service: hyperledger-fabric
    environment:
      - FABRIC_LOGGING_SPEC=INFO
      - ORDERER_GENERAL_LISTENADDRESS=0.0.0.0
      - ORDERER_GENERAL_LISTENPORT=7050
      - ORDERER_GENERAL_LOCALMSPID=OrdererMSP
      - ORDERER_GENERAL_LOCALMSPDIR=/var/hyperledger/orderer/msp
      # enabled TLS
      - ORDERER_GENERAL_TLS_ENABLED=true
      -
ORDERER_GENERAL_TLS_PRIVATEKEY=/var/hyperledger/orderer/tls/server.ke
y
      -
ORDERER_GENERAL_TLS_CERTIFICATE=/var/hyperledger/orderer/tls/server.c
rt
      -
ORDERER_GENERAL_TLS_ROOTCAS=[/var/hyperledger/orderer/tls/ca.crt]
      -
ORDERER_GENERAL_CLUSTER_CLIENTCERTIFICATE=/var/hyperledger/orderer/tl
s/server.crt
      -
ORDERER_GENERAL_CLUSTER_CLIENTPRIVATEKEY=/var/hyperledger/orderer/tls
/server.key
      -
ORDERER_GENERAL_CLUSTER_ROOTCAS=[/var/hyperledger/orderer/tls/ca.crt]
      - ORDERER_GENERAL_BOOTSTRAPMETHOD=none
      - ORDERER_CHANNELPARTICIPATION_ENABLED=true
      - ORDERER_ADMIN_TLS_ENABLED=true
      -
ORDERER_ADMIN_TLS_CERTIFICATE=/var/hyperledger/orderer/tls/server.crt
      -
ORDERER_ADMIN_TLS_PRIVATEKEY=/var/hyperledger/orderer/tls/server.key
      -
ORDERER_ADMIN_TLS_ROOTCAS=[/var/hyperledger/orderer/tls/ca.crt]
      -
ORDERER_ADMIN_TLS_CLIENTROOTCAS=[/var/hyperledger/orderer/tls/ca.crt]
      - ORDERER_ADMIN_LISTENADDRESS=0.0.0.0:7053
      - ORDERER_OPERATIONS_LISTENADDRESS=orderer.example.com:9443
      - ORDERER_METRICS_PROVIDER=prometheus
    working_dir: /root
    command: orderer
    volumes:
      -
../organizations/ordererOrganizations/example.com/orderers/orderer.ex
ample.com/msp:/var/hyperledger/orderer/msp
      -
../organizations/ordererOrganizations/example.com/orderers/orderer.ex
ample.com/tls/:/var/hyperledger/orderer/tls
      - orderer.example.com:/var/hyperledger/production/orderer
    ports:
      - 7050:7050
      - 7053:7053
      - 9443:9443
    networks:
      - test

  couchdb0:
    container_name: couchdb0
    image: couchdb:3.3.2
    labels:
      service: hyperledger-fabric
    environment:
      - COUCHDB_USER=admin
      - COUCHDB_PASSWORD=adminpw
    ports:
      - "5984:5984"
    networks:
      - test

  peer0.org1.example.com:
    container_name: peer0.org1.example.com
    image: hyperledger/fabric-peer:latest
    labels:
      service: hyperledger-fabric
    environment:
      - FABRIC_LOGGING_SPEC=INFO
      #- FABRIC_LOGGING_SPEC=DEBUG
      - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
      - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=fabric_test
      - CORE_PEER_TLS_ENABLED=true
      - CORE_PEER_PROFILE_ENABLED=false
      -
CORE_PEER_TLS_CERT_FILE=/etc/hyperledger/fabric/tls/server.crt
      - CORE_PEER_TLS_KEY_FILE=/etc/hyperledger/fabric/tls/server.key
      -
CORE_PEER_TLS_ROOTCERT_FILE=/etc/hyperledger/fabric/tls/ca.crt
      # Peer specific variables
      - CORE_PEER_ID=peer0.org1.example.com
      - CORE_PEER_ADDRESS=peer0.org1.example.com:7051
      - CORE_PEER_LISTENADDRESS=0.0.0.0:7051
      - CORE_PEER_CHAINCODEADDRESS=peer0.org1.example.com:7052
      - CORE_PEER_CHAINCODELISTENADDRESS=0.0.0.0:7052
      - CORE_PEER_GOSSIP_BOOTSTRAP=peer0.org1.example.com:7051
      - CORE_PEER_GOSSIP_EXTERNALENDPOINT=peer0.org1.example.com:7051
      - CORE_PEER_LOCALMSPID=Org1MSP
      - CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/fabric/msp
      - CORE_OPERATIONS_LISTENADDRESS=peer0.org1.example.com:9444
      - CORE_METRICS_PROVIDER=prometheus
      -
CHAINCODE_AS_A_SERVICE_BUILDER_CONFIG={"peername":"peer0org1"}
      - CORE_CHAINCODE_EXECUTETIMEOUT=300s
      - CORE_LEDGER_STATE_STATEDATABASE=CouchDB
      - CORE_LEDGER_STATE_COUCHDBCONFIG_COUCHDBADDRESS=couchdb0:5984
      - CORE_LEDGER_STATE_COUCHDBCONFIG_USERNAME=admin
      - CORE_LEDGER_STATE_COUCHDBCONFIG_PASSWORD=adminpw
    volumes:
      - /var/run/docker.sock:/host/var/run/docker.sock
      -
../organizations/peerOrganizations/org1.example.com/peers/peer0.org1.
example.com:/etc/hyperledger/fabric
      - peer0.org1.example.com:/var/hyperledger/production
    working_dir: /root
    command: peer node start
    ports:
      - 7051:7051
      - 9444:9444
    depends_on:
      - couchdb0
    networks:
      - test

  couchdb1:
    container_name: couchdb1
    image: couchdb:3.3.2
    labels:
      service: hyperledger-fabric
    environment:
      - COUCHDB_USER=admin
      - COUCHDB_PASSWORD=adminpw
    ports:
      - "7984:5984"
    networks:
      - test

  peer0.org2.example.com:
    container_name: peer0.org2.example.com
    image: hyperledger/fabric-peer:latest
    labels:
      service: hyperledger-fabric
    environment:
      - FABRIC_LOGGING_SPEC=INFO
      #- FABRIC_LOGGING_SPEC=DEBUG
      - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
      - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=fabric_test
      - CORE_PEER_TLS_ENABLED=true
      - CORE_PEER_PROFILE_ENABLED=false
      -
CORE_PEER_TLS_CERT_FILE=/etc/hyperledger/fabric/tls/server.crt
      - CORE_PEER_TLS_KEY_FILE=/etc/hyperledger/fabric/tls/server.key
      -
CORE_PEER_TLS_ROOTCERT_FILE=/etc/hyperledger/fabric/tls/ca.crt
      # Peer specific variables
      - CORE_PEER_ID=peer0.org2.example.com
      - CORE_PEER_ADDRESS=peer0.org2.example.com:9051
      - CORE_PEER_LISTENADDRESS=0.0.0.0:9051
      - CORE_PEER_CHAINCODEADDRESS=peer0.org2.example.com:9052
      - CORE_PEER_CHAINCODELISTENADDRESS=0.0.0.0:9052
      - CORE_PEER_GOSSIP_EXTERNALENDPOINT=peer0.org2.example.com:9051
      - CORE_PEER_GOSSIP_BOOTSTRAP=peer0.org2.example.com:9051
      - CORE_PEER_LOCALMSPID=Org2MSP
      - CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/fabric/msp
      - CORE_OPERATIONS_LISTENADDRESS=peer0.org2.example.com:9445
      - CORE_METRICS_PROVIDER=prometheus
      -
CHAINCODE_AS_A_SERVICE_BUILDER_CONFIG={"peername":"peer0org2"}
      - CORE_CHAINCODE_EXECUTETIMEOUT=300s
      - CORE_LEDGER_STATE_STATEDATABASE=CouchDB
      - CORE_LEDGER_STATE_COUCHDBCONFIG_COUCHDBADDRESS=couchdb1:5984
      - CORE_LEDGER_STATE_COUCHDBCONFIG_USERNAME=admin
      - CORE_LEDGER_STATE_COUCHDBCONFIG_PASSWORD=adminpw
    volumes:
      - /var/run/docker.sock:/host/var/run/docker.sock
      -
../organizations/peerOrganizations/org2.example.com/peers/peer0.org2.
example.com:/etc/hyperledger/fabric
      - peer0.org2.example.com:/var/hyperledger/production
    working_dir: /root
    command: peer node start
    ports:
      - 9051:9051
      - 9445:9445
    depends_on:
      - couchdb1
    networks:
      - test
```

Now run the docker-compose file. For that, execute the following command from the Fabric-network folder.

**docker compose -f docker/docker-compose-2org.yaml up -d**



```
kba@Lab:~/CHF/Fabric-network$ docker-compose -f docker/docker-compose-2org.yaml up -d
Creating volume "docker_orderer.example.com" with default driver
Creating volume "docker_peer0.org1.example.com" with default driver
Creating volume "docker_peer0.org2.example.com" with default driver
WARNING: Found orphan containers (ca_org1, ca_org2, ca_orderer) for this project. If you
 removed or renamed this service in your compose file, you can run this command with the
 --remove-orphans flag to clean it up.
Creating couchdb1             ... done
Creating couchdb0                ... done
Creating orderer.example.com     ... done
Creating peer0.org2.example.com ... done
Creating peer0.org1.example.com ... done
Creating cli                     ... done
kba@Lab:~/CHF/Fabric-network$ 
```

This will start all the containers using the certificates we mapped in the docker-compose file.

## Generating the channel artifacts

In Hyperledger Fabric, privacy can be maintained through channels, and peers can join channels to be part of a group of organizations.

For creating a channel, we need to follow the below steps,

- Generate the **configtx.yaml** file, which contains the network policies, channel capabilities and profiles.
- Generate the genesis block of the channel by passing a profile available in the configtx.yaml
- Create the channel using the **osnadmin** command.

Create a folder **config** to keep the configuration.

**mkdir config**

Create **configtx.yaml** file within the **config** folder and add the following. (Refer: fabric-samples/test-network/configtx/configtx.yaml)

```
Organizations:
  - &OrdererOrg
    Name: OrdererOrg

    ID: OrdererMSP

```

```
      MSPDir: ../organizations/ordererOrganizations/example.com/msp

      Policies:
        Readers:
          Type: Signature
          Rule: "OR('OrdererMSP.member')"
        Writers:
          Type: Signature
          Rule: "OR('OrdererMSP.member')"
        Admins:
          Type: Signature
          Rule: "OR('OrdererMSP.admin')"

      OrdererEndpoints:
        - orderer.example.com:7050

    - &Org1
      Name: Org1MSP

      ID: Org1MSP

      MSPDir: ../organizations/peerOrganizations/org1.example.com/msp

      Policies:
        Readers:
          Type: Signature
          Rule: "OR('Org1MSP.admin', 'Org1MSP.peer', 'Org1MSP.client')"
        Writers:
          Type: Signature
          Rule: "OR('Org1MSP.admin', 'Org1MSP.client')"
        Admins:
          Type: Signature
          Rule: "OR('Org1MSP.admin')"
        Endorsement:
          Type: Signature
          Rule: "OR('Org1MSP.peer')"

    - &Org2
      Name: Org2MSP

      ID: Org2MSP

      MSPDir: ../organizations/peerOrganizations/org2.example.com/msp

      Policies:
        Readers:
          Type: Signature
          Rule: "OR('Org2MSP.admin', 'Org2MSP.peer', 'Org2MSP.client')"
        Writers:
          Type: Signature
          Rule: "OR('Org2MSP.admin', 'Org2MSP.client')"
        Admins:
          Type: Signature
          Rule: "OR('Org2MSP.admin')"
        Endorsement:
          Type: Signature
          Rule: "OR('Org2MSP.peer')"

  Capabilities:
    Channel: &ChannelCapabilities
      V2_0: true

    Orderer: &OrdererCapabilities
      V2_0: true

    Application: &ApplicationCapabilities
      V2_5: true

  Application: &ApplicationDefaults
    Organizations:

    Policies:
      Readers:
        Type: ImplicitMeta
        Rule: "ANY Readers"
      Writers:
        Type: ImplicitMeta
        Rule: "ANY Writers"
      Admins:
        Type: ImplicitMeta
        Rule: "MAJORITY Admins"
      LifecycleEndorsement:
        Type: ImplicitMeta
        Rule: "MAJORITY Endorsement"
      Endorsement:
        Type: ImplicitMeta
        Rule: "MAJORITY Endorsement"

    Capabilities:
      <<: *ApplicationCapabilities

  Orderer: &OrdererDefaults
    OrdererType: etcdraft
    Addresses:
      - orderer.example.com:7050

    BatchTimeout: 2s

    BatchSize:
      MaxMessageCount: 10

      PreferredMaxBytes: 512 KB

    Organizations:

    Policies:
      Readers:
        Type: ImplicitMeta
        Rule: "ANY Readers"
      Writers:
        Type: ImplicitMeta
        Rule: "ANY Writers"
      Admins:
        Type: ImplicitMeta
        Rule: "MAJORITY Admins"
      BlockValidation:
        Type: ImplicitMeta
        Rule: "ANY Writers"

  Channel: &ChannelDefaults
    Policies:
      Readers:
        Type: ImplicitMeta
        Rule: "ANY Readers"
      Writers:
        Type: ImplicitMeta
        Rule: "ANY Writers"
      Admins:
        Type: ImplicitMeta
        Rule: "MAJORITY Admins"

    Capabilities:
      <<: *ChannelCapabilities

  Profiles:
    ChannelUsingRaft:
      <<: *ChannelDefaults
      Orderer:
        <<: *OrdererDefaults
        OrdererType: etcdraft
        EtcdRaft:
          Consenters:
            - Host: orderer.example.com
              Port: 7050
              ClientTLSCert:
  ../organizations/ordererOrganizations/example.com/orderers/orderer.ex
  ample.com/tls/server.crt
              ServerTLSCert:
  ../organizations/ordererOrganizations/example.com/orderers/orderer.ex
  ample.com/tls/server.crt
        Organizations:
          - *OrdererOrg
        Capabilities: *OrdererCapabilities
      Application:
        <<: *ApplicationDefaults
        Organizations:
          - *Org1
          - *Org2
        Capabilities: *ApplicationCapabilities
```

The configtx.yaml file has multiple sections like Organizations, Capabilities, Channel, Application, and Profiles.

Once you configure the configtx.yaml file, we can now generate the genesis block using **configtxgen** binary by passing this config and the profile specified inside this file. First, set the FABRIC_CFG_PATH environment variable to the directory's path that contains the configtx.yaml file. For that, execute the following command:

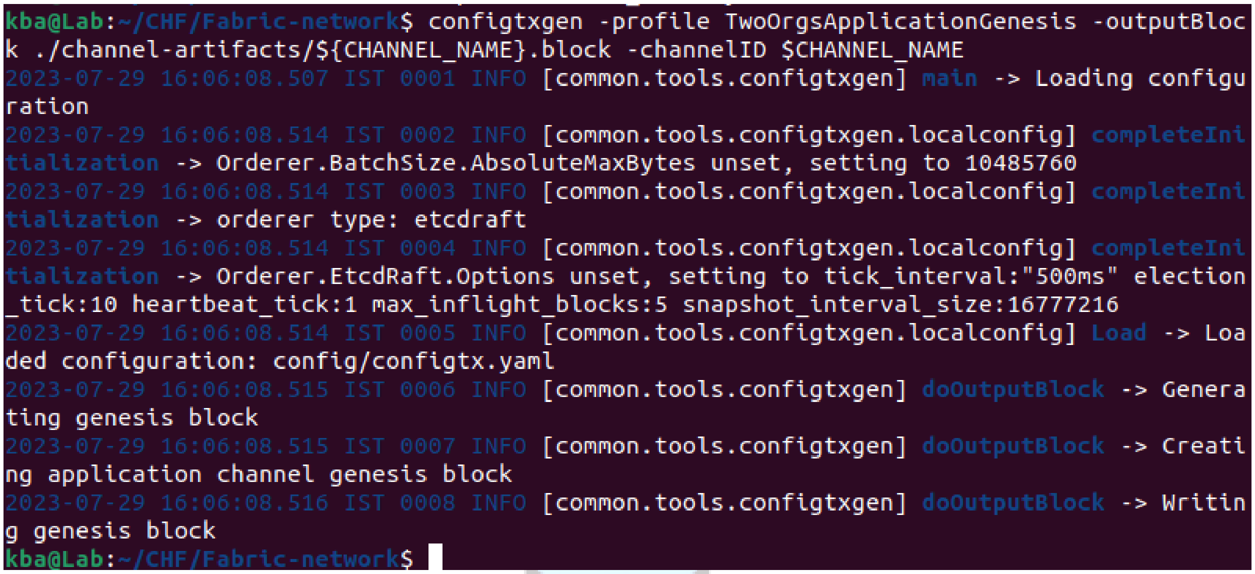
**export FABRIC_CFG_PATH=./config**

Define the channel name as an environment variable so it can be reused. For that, execute the following command:

**export CHANNEL_NAME=mychannel**

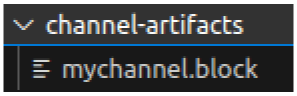
Generate the genesis block for the channel using the following command:

**configtxgen -profile ChannelUsingRaft -outputBlock ./channel-artifacts/${CHANNEL_NAME}.block -channelID $CHANNEL_NAME**



```
kba@Lab:~/CHF/Fabric-network$ configtxgen -profile TwoOrgsApplicationGenesis -outputBloc
k ./channel-artifacts/${CHANNEL_NAME}.block -channelID $CHANNEL_NAME
2023-07-29 16:06:08.507 IST 0001 INFO [common.tools.configtxgen] main -> Loading configu
ration
2023-07-29 16:06:08.514 IST 0002 INFO [common.tools.configtxgen.localconfig] completeIni
tialization -> Orderer.BatchSize.AbsoluteMaxBytes unset, setting to 10485760
2023-07-29 16:06:08.514 IST 0003 INFO [common.tools.configtxgen.localconfig] completeIni
tialization -> orderer type: etcdraft
2023-07-29 16:06:08.514 IST 0004 INFO [common.tools.configtxgen.localconfig] completeIni
tialization -> Orderer.EtcdRaft.Options unset, setting to tick_interval:"500ms" election
_tick:10 heartbeat_tick:1 max_inflight_blocks:5 snapshot_interval_size:16777216
2023-07-29 16:06:08.514 IST 0005 INFO [common.tools.configtxgen.localconfig] Load -> Loa
ded configuration: config/configtx.yaml
2023-07-29 16:06:08.515 IST 0006 INFO [common.tools.configtxgen] doOutputBlock -> Genera
ting genesis block
2023-07-29 16:06:08.515 IST 0007 INFO [common.tools.configtxgen] doOutputBlock -> Creati
ng application channel genesis block
2023-07-29 16:06:08.516 IST 0008 INFO [common.tools.configtxgen] doOutputBlock -> Writin
g genesis block
kba@Lab:~/CHF/Fabric-network$
```

It creates the **mychannel.block** in channel-artifacts folder.



```
∨ channel-artifacts
  ≡ mychannel.block
```

## Add the orderer to the channel

Now, join the ordering node to the channel using the **osnadmin channel join** command. First set the following environment variables.

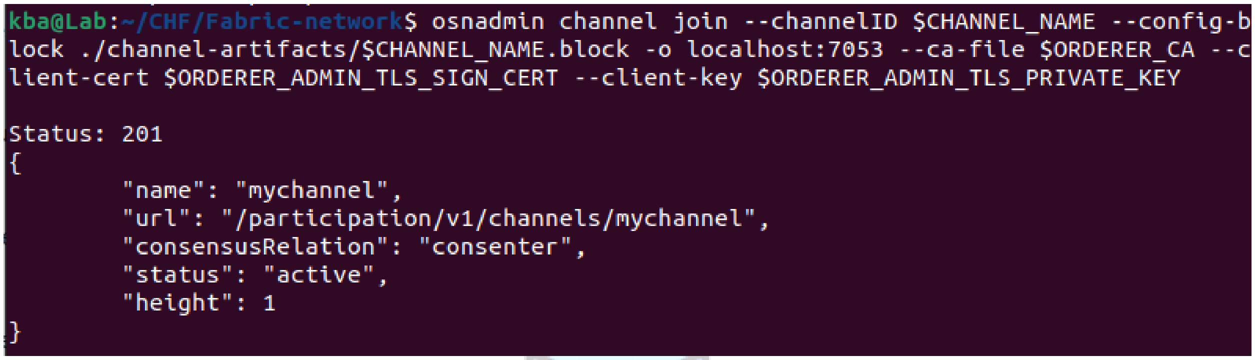
**export ORDERER_CA=./organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem**

**export ORDERER_ADMIN_TLS_SIGN_CERT=./organizations/ordererOrganizations/example.com/orderers/orderer.example.com/tls/server.crt**

**export ORDERER_ADMIN_TLS_PRIVATE_KEY=./organizations/ordererOrganizations/example.com/orderers/orderer.example.com/tls/server.key**

Execute the following command to join the orderer to channel.

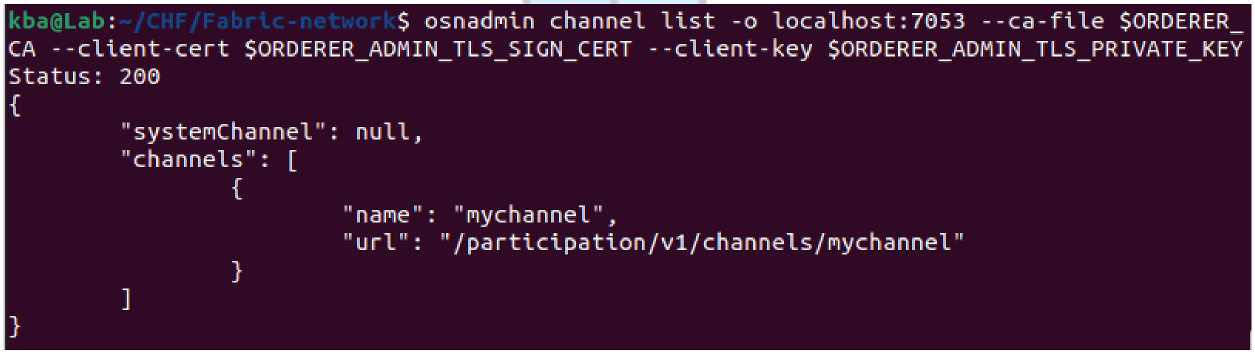
**osnadmin channel join --channelID $CHANNEL_NAME --config-block ./channel-artifacts/$CHANNEL_NAME.block -o localhost:7053 --ca-file $ORDERER_CA --client-cert $ORDERER_ADMIN_TLS_SIGN_CERT --client-key $ORDERER_ADMIN_TLS_PRIVATE_KEY**



```
kba@Lab:~/CHF/Fabric-network$ osnadmin channel join --channelID $CHANNEL_NAME --config-b
lock ./channel-artifacts/$CHANNEL_NAME.block -o localhost:7053 --ca-file $ORDERER_CA --c
lient-cert $ORDERER_ADMIN_TLS_SIGN_CERT --client-key $ORDERER_ADMIN_TLS_PRIVATE_KEY

Status: 201
{
        "name": "mychannel",
        "url": "/participation/v1/channels/mychannel",
        "consensusRelation": "consenter",
        "status": "active",
        "height": 1
}
```

Execute the following command to view the details of any channel on ordering node.

**osnadmin channel list -o localhost:7053 --ca-file $ORDERER_CA --client-cert $ORDERER_ADMIN_TLS_SIGN_CERT --client-key $ORDERER_ADMIN_TLS_PRIVATE_KEY**



```
kba@Lab:~/CHF/Fabric-network$ osnadmin channel list -o localhost:7053 --ca-file $ORDERER_
CA --client-cert $ORDERER_ADMIN_TLS_SIGN_CERT --client-key $ORDERER_ADMIN_TLS_PRIVATE_KEY
Status: 200
{
        "systemChannel": null,
        "channels": [
                {
                        "name": "mychannel",
                        "url": "/participation/v1/channels/mychannel"
                }
        ]
}
```

## Join peers to the channel

We need to use the **peer channel join** command to join each peer to the channel. To execute peer commands, we need to provide core.yaml.

Create a folder **peercfg**, to keep the core.yaml file.

**mkdir peercfg**

Create a **core.yaml** with in the **peercfg** folder and add the following (Refer: fabric-samples/config/core.yaml)

```
peer:
    id: jdoe
    networkId: dev
    listenAddress: 0.0.0.0:7051
    address: 0.0.0.0:7051
    addressAutoDetect: false
    gateway:
        enabled: true
        endorsementTimeout: 30s
        broadcastTimeout: 30s
        dialTimeout: 2m

    keepalive:
        interval: 7200s
        timeout: 20s
        minInterval: 60s
        client:
            interval: 60s
            timeout: 20s
        deliveryClient:
            interval: 60s
            timeout: 20s
    gossip:
        bootstrap: 127.0.0.1:7051
        useLeaderElection: false
        orgLeader: true
        membershipTrackerInterval: 5s
        endpoint:
        maxBlockCountToStore: 10
        maxPropagationBurstLatency: 10ms
        maxPropagationBurstSize: 10
        propagateIterations: 1
        propagatePeerNum: 3
        pullInterval: 4s
        pullPeerNum: 3
        requestStateInfoInterval: 4s
        publishStateInfoInterval: 4s
        stateInfoRetentionInterval:
        publishCertPeriod: 10s
        skipBlockVerification: false
        dialTimeout: 3s
        connTimeout: 2s
        recvBuffSize: 20
        sendBuffSize: 200
        digestWaitTime: 1s
        requestWaitTime: 1500ms
        responseWaitTime: 2s
        aliveTimeInterval: 5s
        aliveExpirationTimeout: 25s
        reconnectInterval: 25s
        maxConnectionAttempts: 120
        msgExpirationFactor: 20
        externalEndpoint:
        election:
            startupGracePeriod: 15s
            membershipSampleInterval: 1s
            leaderAliveThreshold: 10s
            leaderElectionDuration: 5s
        pvtData:
            pullRetryThreshold: 60s
            transientstoreMaxBlockRetention: 1000
            pushAckTimeout: 3s
            btlPullMargin: 10
            reconcileBatchSize: 10
            reconcileSleepInterval: 1m
            reconciliationEnabled: true
            skipPullingInvalidTransactionsDuringCommit: false
            implicitCollectionDisseminationPolicy:
                requiredPeerCount: 0
                maxPeerCount: 1
```

```
    state:
        enabled: false
        checkInterval: 10s
        responseTimeout: 3s
        batchSize: 10
        blockBufferSize: 20
        maxRetries: 3

tls:
    enabled: false
    clientAuthRequired: false
    cert:
        file: tls/server.crt
    key:
        file: tls/server.key
    rootcert:
        file: tls/ca.crt
    clientRootCAs:
        files:
            - tls/ca.crt
    clientKey:
        file:
    clientCert:
        file:
authentication:
    timewindow: 15m
fileSystemPath: /var/hyperledger/production

BCCSP:
    Default: SW
    SW:
        Hash: SHA2
        Security: 256
        FileKeyStore:
            KeyStore:
    PKCS11:
        Library:
        Label:
        Pin:
        Hash:
        Security:
        SoftwareVerify:
        Immutable:
        AltID:
        KeyIds:

mspConfigPath: msp
localMspId: SampleOrg
client:
    connTimeout: 3s
deliveryclient:
    blockGossipEnabled: true
    reconnectTotalTimeThreshold: 3600s
    connTimeout: 3s
    reConnectBackoffThreshold: 3600s
    addressOverrides:
localMspType: bccsp
profile:
    enabled: false
    listenAddress: 0.0.0.0:6060
handlers:
    authFilters:
        - name: DefaultAuth
        - name: ExpirationCheck
    decorators:
        - name: DefaultDecorator
    endorsers:
        escc:
            name: DefaultEndorsement
            library:
    validators:
        vscc:
            name: DefaultValidation
            library:
```

```
    validatorPoolSize:

    discovery:
        enabled: true
        authCacheEnabled: true
        authCacheMaxSize: 1000
        authCachePurgeRetentionRatio: 0.75
        orgMembersAllowedAccess: false

    limits:
        concurrency:
            endorserService: 2500
            deliverService: 2500
            gatewayService: 500

    maxRecvMsgSize: 104857600
    maxSendMsgSize: 104857600

vm:
    endpoint: unix:///var/run/docker.sock
    docker:
        tls:
            enabled: false
            ca:
                file: docker/ca.crt
            cert:
                file: docker/tls.crt
            key:
                file: docker/tls.key
        attachStdout: false

        hostConfig:
            NetworkMode: host
            Dns:
            LogConfig:
                Type: json-file
                Config:
                    max-size: "50m"
                    max-file: "5"
            Memory: 2147483648

chaincode:
    id:
        path:
        name:
    builder: $(DOCKER_NS)/fabric-ccenv:$(TWO_DIGIT_VERSION)
    pull: false
    golang:
        runtime: $(DOCKER_NS)/fabric-baseos:$(TWO_DIGIT_VERSION)
        dynamicLink: false
    java:
        runtime: $(DOCKER_NS)/fabric-javaenv:$(TWO_DIGIT_VERSION)
    node:
        runtime: $(DOCKER_NS)/fabric-nodeenv:$(TWO_DIGIT_VERSION)

    externalBuilders:
        - name: ccaas_builder
          path: /opt/hyperledger/ccaas_builder
          propagateEnvironment:
              - CHAINCODE_AS_A_SERVICE_BUILDER_CONFIG

    installTimeout: 300s
    startuptimeout: 300s
    executetimeout: 30s
    mode: net
    keepalive: 0

    system:
        _lifecycle: enable
        cscc: enable
        lscc: enable
        qscc: enable

    logging:
        level: info
        shim: warning
        format: "%{color}%{time:2006-01-02 15:04:05.000 MST}
[%{module}] %{shortfunc} -> %{level:.4s} %{id:03x}%{color:reset}
%{message}"

ledger:
    blockchain:
    state:
        stateDatabase: goleveldb
        totalQueryLimit: 100000
        couchDBConfig:
            couchDBAddress: 127.0.0.1:5984
            username:
            password:
            maxRetries: 3
            maxRetriesOnStartup: 10
            requestTimeout: 35s
            internalQueryLimit: 1000
            maxBatchUpdateSize: 1000
            createGlobalChangesDB: false
            cacheSize: 64
    history:
        enableHistoryDatabase: true
    pvtdataStore:
        collElgProcMaxDbBatchSize: 5000
        collElgProcDbBatchesInterval: 1000
        deprioritizedDataReconcilerInterval: 60m
        purgeInterval: 100
        purgedKeyAuditLogging: true

    snapshots:
        rootDir: /var/hyperledger/production/snapshots
operations:
    listenAddress: 127.0.0.1:9443

    tls:
        enabled: false
        cert:
            file:
```

```
        key:
            file:
        clientAuthRequired: false
        clientRootCAs:
            files: []
metrics:
    provider: disabled
    statsd:
        network: udp
        address: 127.0.0.1:8125
        writeInterval: 10s
        prefix:
```

Now, open another command terminal (without closing the existing command terminal)  within the Fabric-network folder. This command terminal is meant for submitting transactions as Org1.

**Note**: Hereafter, if any commands need to be executed in this terminal, we will mention them as **peer0_Org1 terminal**. If any commands need to be executed in the other terminal, it will be mentioned as the **host terminal**.

Execute the following commands for setting the environment variables.

##### Execute the command in **peer0_Org1** terminal #####

**export FABRIC_CFG_PATH=./peercfg**

**export CHANNEL_NAME=mychannel**

**export CORE_PEER_LOCALMSPID=Org1MSP**

**export CORE_PEER_TLS_ENABLED=true**

**export CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt**

**export CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp**

**export CORE_PEER_ADDRESS=localhost:7051**

**export ORDERER_CA=${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem**

**export ORG1_PEER_TLSROOTCERT=${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt**

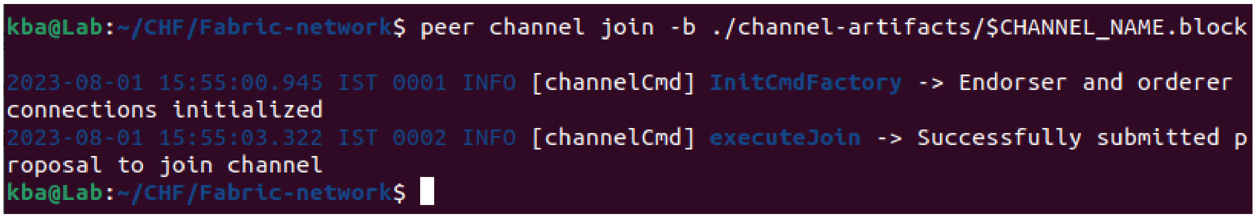
**export ORG2_PEER_TLSROOTCERT=${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt**

### Joining peer0 of org1 to channel

Join the peer to the channel using the **peer channel join** command. The command uses the path of genesis block(mychannel.block) to join the channel:

##### Execute the command in **peer0_Org1** terminal #####

**peer channel join -b ./channel-artifacts/$CHANNEL_NAME.block**

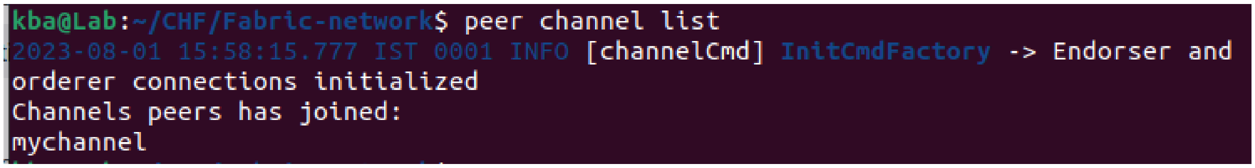


```
kba@Lab:~/CHF/Fabric-network$ peer channel join -b ./channel-artifacts/$CHANNEL_NAME.block

2023-08-01 15:55:00.945 IST 0001 INFO [channelCmd] InitCmdFactory -> Endorser and orderer
connections initialized
2023-08-01 15:55:03.322 IST 0002 INFO [channelCmd] executeJoin -> Successfully submitted p
roposal to join channel
kba@Lab:~/CHF/Fabric-network$ 
```

The Org1 peer has successfully joined the channel. Confirm it by the peer channel list command.

##### Execute the command in **peer0_Org1** terminal #####

**peer channel list**



```
kba@Lab:~/CHF/Fabric-network$ peer channel list
2023-08-01 15:58:15.777 IST 0001 INFO [channelCmd] InitCmdFactory -> Endorser and
orderer connections initialized
Channels peers has joined:
mychannel
```

### Joining peer0 of org2 to channel

Now, let's join peer0.org2.example.com to this channel. So open another command terminal (without closing the existing two command terminals) and set the environment variables related to Org2.

**Note**: Hereafter, if any commands need to be executed in this terminal, we will mention them as **peer0_Org2 terminal**.

Execute the following commands for setting the environment variables.

##### Execute the command in **peer0_Org2** terminal #####

**export FABRIC_CFG_PATH=./peercfg**

**export CHANNEL_NAME=mychannel**

**export CORE_PEER_LOCALMSPID=Org2MSP**

**export CORE_PEER_TLS_ENABLED=true**

**export CORE_PEER_ADDRESS=localhost:9051**

**export CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt**

**export CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp**

**export ORDERER_CA=${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem**

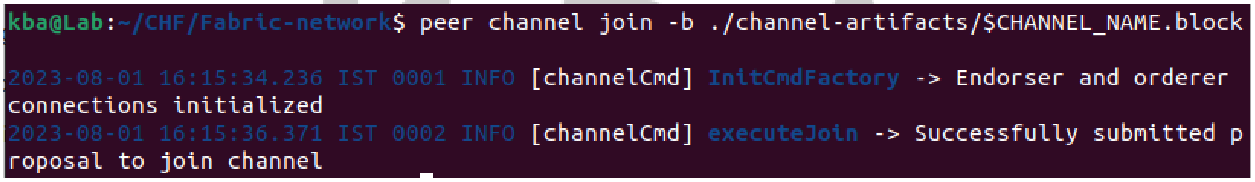
**export ORG1_PEER_TLSROOTCERT=${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt**

**export ORG2_PEER_TLSROOTCERT=${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt**

Now we can execute the command for joining  peer0.org2.example.com to the channel:

##### Execute the command in **peer0_Org2** terminal #####

**peer channel join -b ./channel-artifacts/$CHANNEL_NAME.block**

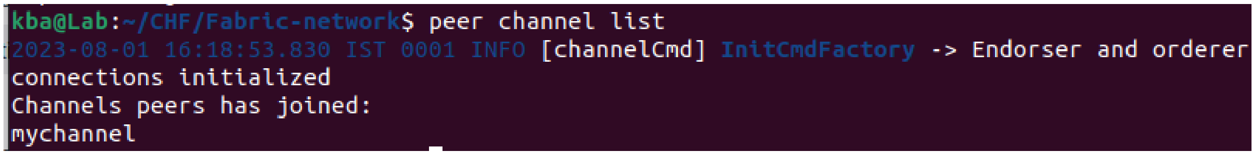


```
kba@Lab:~/CHF/Fabric-network$ peer channel join -b ./channel-artifacts/$CHANNEL_NAME.block

2023-08-01 16:15:34.236 IST 0001 INFO [channelCmd] InitCmdFactory -> Endorser and orderer
connections initialized
2023-08-01 16:15:36.371 IST 0002 INFO [channelCmd] executeJoin -> Successfully submitted p
roposal to join channel
```

Now, peer0.org2.example.com joined to mychannel. Confirm it by the **peer channel list** command.

##### Execute the command in **peer0_Org2** terminal #####

**peer channel list**



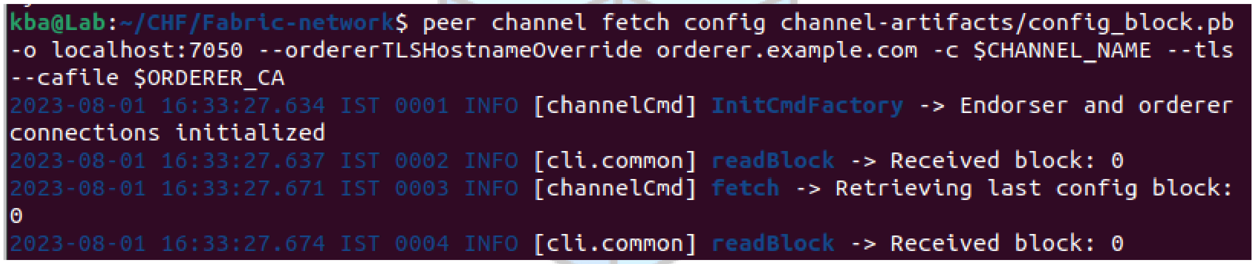
```
kba@Lab:~/CHF/Fabric-network$ peer channel list
2023-08-01 16:18:53.830 IST 0001 INFO [channelCmd] InitCmdFactory -> Endorser and orderer
connections initialized
Channels peers has joined:
mychannel
```

## Anchor Peer Update

After an organization has joined their peers to the channel, they should select at least one of their peers to become an anchor peer. Execute the following commands for setting the anchor peer for org1.

##### Execute the command in **peer0_Org1** terminal #####

**peer channel fetch config channel-artifacts/config_block.pb -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com -c $CHANNEL_NAME --tls --cafile $ORDERER_CA**



```
kba@Lab:~/CHF/Fabric-network$ peer channel fetch config channel-artifacts/config_block.pb
-o localhost:7050 --ordererTLSHostnameOverride orderer.example.com -c $CHANNEL_NAME --tls
--cafile $ORDERER_CA
2023-08-01 16:33:27.634 IST 0001 INFO [channelCmd] InitCmdFactory -> Endorser and orderer
connections initialized
2023-08-01 16:33:27.637 IST 0002 INFO [cli.common] readBlock -> Received block: 0
2023-08-01 16:33:27.671 IST 0003 INFO [channelCmd] fetch -> Retrieving last config block:
0
2023-08-01 16:33:27.674 IST 0004 INFO [cli.common] readBlock -> Received block: 0
```

**cd channel-artifacts**

**configtxlator proto_decode --input config_block.pb --type common.Block --output config_block.json**

**jq '.data.data[0].payload.data.config' config_block.json > config.json**

**cp config.json config_copy.json**

**jq '.channel_group.groups.Application.groups.Org1MSP.values += {"AnchorPeers":{"mod_policy": "Admins","value":{"anchor_peers": [{"host": "peer0.org1.example.com","port": 7051}]},"version": "0"}}' config_copy.json > modified_config.json**

**configtxlator proto_encode --input config.json --type common.Config --output**

```
config.pb
```

```
configtxlator proto_encode --input modified_config.json --type common.Config
--output modified_config.pb
```

```
configtxlator compute_update --channel_id ${CHANNEL_NAME} --original
config.pb --updated modified_config.pb --output config_update.pb
```

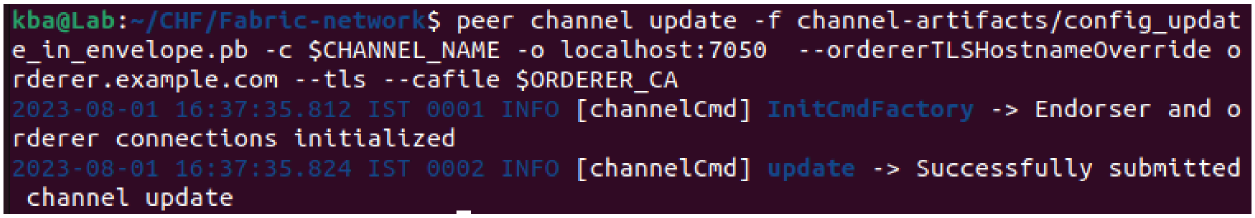
```
configtxlator proto_decode --input config_update.pb --type
common.ConfigUpdate --output config_update.json
```

```
echo
'{"payload":{"header":{"channel_header":{"channel_id":"'$CHANNEL_NAME'",
"type":2}},"data":{"config_update":'$(cat config_update.json)'}}}' | jq . >
config_update_in_envelope.json
```

```
configtxlator proto_encode --input config_update_in_envelope.json --type
common.Envelope --output config_update_in_envelope.pb
```

```
cd ..
```

```
peer channel update -f channel-artifacts/config_update_in_envelope.pb -c
$CHANNEL_NAME -o localhost:7050  --ordererTLSHostnameOverride
orderer.example.com --tls --cafile $ORDERER_CA
```

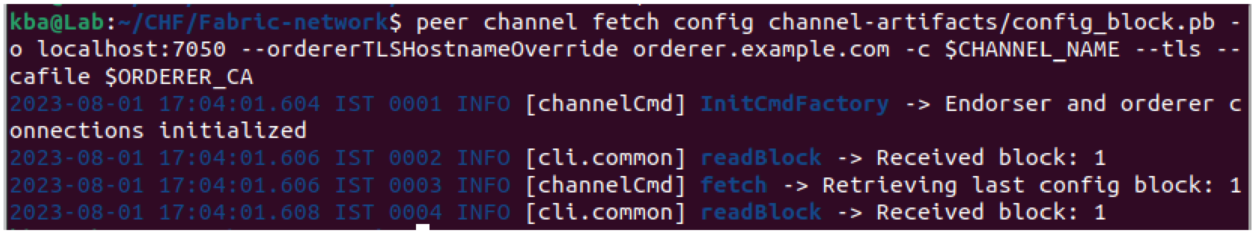


```
kba@Lab:~/CHF/Fabric-network$ peer channel update -f channel-artifacts/config_updat
e_in_envelope.pb -c $CHANNEL_NAME -o localhost:7050  --ordererTLSHostnameOverride o
rderer.example.com --tls --cafile $ORDERER_CA
2023-08-01 16:37:35.812 IST 0001 INFO [channelCmd] InitCmdFactory -> Endorser and o
rderer connections initialized
2023-08-01 16:37:35.824 IST 0002 INFO [channelCmd] update -> Successfully submitted
 channel update
```

Execute the following commands for setting the anchor peer for org2.

##### Execute the command in **peer0_Org2** terminal #####

```
peer channel fetch config channel-artifacts/config_block.pb -o localhost:7050
--ordererTLSHostnameOverride orderer.example.com -c $CHANNEL_NAME
--tls --cafile $ORDERER_CA
```



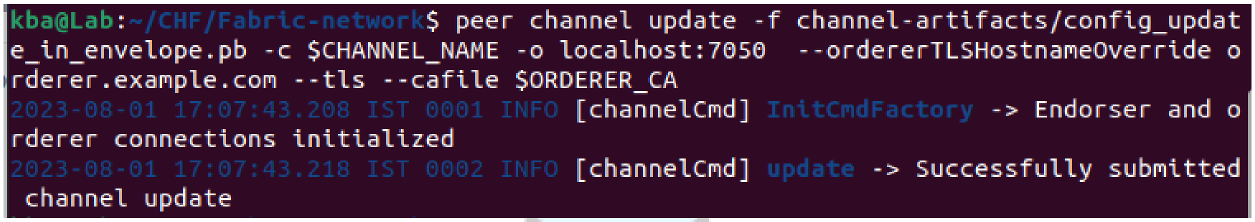
```
kba@Lab:~/CHF/Fabric-network$ peer channel fetch config channel-artifacts/config_block.pb -
o localhost:7050 --ordererTLSHostnameOverride orderer.example.com -c $CHANNEL_NAME --tls --
cafile $ORDERER_CA
2023-08-01 17:04:01.604 IST 0001 INFO [channelCmd] InitCmdFactory -> Endorser and orderer c
onnections initialized
2023-08-01 17:04:01.606 IST 0002 INFO [cli.common] readBlock -> Received block: 1
2023-08-01 17:04:01.606 IST 0003 INFO [channelCmd] fetch -> Retrieving last config block: 1
2023-08-01 17:04:01.608 IST 0004 INFO [cli.common] readBlock -> Received block: 1
```

```
cd channel-artifacts

configtxlator proto_decode --input config_block.pb --type common.Block
--output config_block.json

jq '.data.data[0].payload.data.config' config_block.json > config.json

cp config.json config_copy.json

jq '.channel_group.groups.Application.groups.Org2MSP.values +=
{"AnchorPeers":{"mod_policy": "Admins","value":{"anchor_peers": [{"host":
"peer0.org2.example.com","port": 9051}]},"version": "0"}}' config_copy.json >
modified_config.json

configtxlator proto_encode --input config.json --type common.Config --output
config.pb

configtxlator proto_encode --input modified_config.json --type common.Config
--output modified_config.pb

configtxlator compute_update --channel_id $CHANNEL_NAME --original
config.pb --updated modified_config.pb --output config_update.pb

configtxlator proto_decode --input config_update.pb --type
common.ConfigUpdate --output config_update.json

echo
'{"payload":{"header":{"channel_header":{"channel_id":"'$CHANNEL_NAME'",
"type":2}},"data":{"config_update":'$(cat config_update.json)'}}}' | jq . >
config_update_in_envelope.json

configtxlator proto_encode --input config_update_in_envelope.json --type
```

**common.Envelope --output config_update_in_envelope.pb**

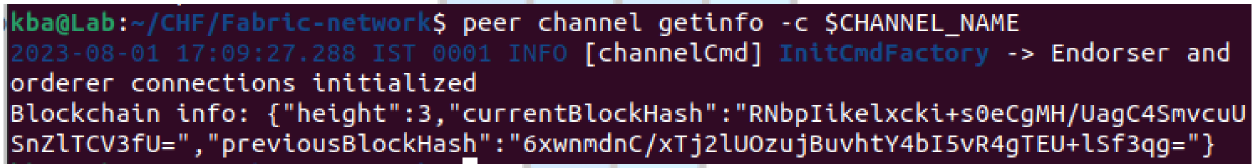
**cd ..**

**peer channel update -f channel-artifacts/config_update_in_envelope.pb -c $CHANNEL_NAME -o localhost:7050  --ordererTLSHostnameOverride orderer.example.com --tls --cafile $ORDERER_CA**



```
kba@Lab:~/CHF/Fabric-network$ peer channel update -f channel-artifacts/config_updat
e_in_envelope.pb -c $CHANNEL_NAME -o localhost:7050  --ordererTLSHostnameOverride o
rderer.example.com --tls --cafile $ORDERER_CA
2023-08-01 17:07:43.208 IST 0001 INFO [channelCmd] InitCmdFactory -> Endorser and o
rderer connections initialized
2023-08-01 17:07:43.218 IST 0002 INFO [channelCmd] update -> Successfully submitted
 channel update
```

**peer channel getinfo -c $CHANNEL_NAME**



```
kba@Lab:~/CHF/Fabric-network$ peer channel getinfo -c $CHANNEL_NAME
2023-08-01 17:09:27.288 IST 0001 INFO [channelCmd] InitCmdFactory -> Endorser and
orderer connections initialized
Blockchain info: {"height":3,"currentBlockHash":"RNbpIikelxcki+s0eCgMH/UagC4SmvcuU
SnZlTCV3fU=","previousBlockHash":"6xwnmdnC/xTj2lUOzujBuvhtY4bI5vR4gTEU+lSf3qg="}
```

Now the anchor peers are updated.

# Deploying a chaincode

Since we will be using the lifecycle methods to deploy a chaincode, we basically need to complete 4 steps to successfully deploy a chaincode.

- Package Chaincode
- Install Chaincode
- Approve chaincode
- Commit Chaincode

Here, we will be deploying the KBA-Automobile chaincode available in the same location of Fabric-network. If you are following a different path then edit the following command accordingly.

## Package the Chaincode

To package the chaincode we will be using the **peer lifecycle chaincode package** command, along with the label.

#####Execute the command in **peer0_Org1** terminal#####

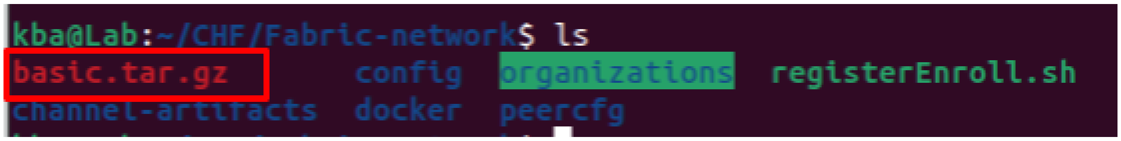
**peer lifecycle chaincode package KBA-Automobile.tar.gz --path ../Chaincode/ --lang golang --label kbaautomobile_1.0**

The above command creates a package named **KBA-Automobile.tar.gz** in your current directory. The **--lang** flag specifies the chaincode language (e.g. go , node, java ) and the **--path** flag should point to the location of your smart contract code. The path must be a fully qualified path or a path relative to your current working directory. The **--label** flag specifies a chaincode label that identifies your chaincode after it is installed. The package will be available in the current folder. Check it by using the **ls** command.

#####Execute the command in **peer0_Org1** terminal#####

**ls**

It will lists KBA-Automobile.tar.gz



```
kba@Lab:~/CHF/Fabric-network$ ls
basic.tar.gz        config  organizations  registerEnroll.sh
channel-artifacts   docker  peercfg
```

## Installing Chaincode

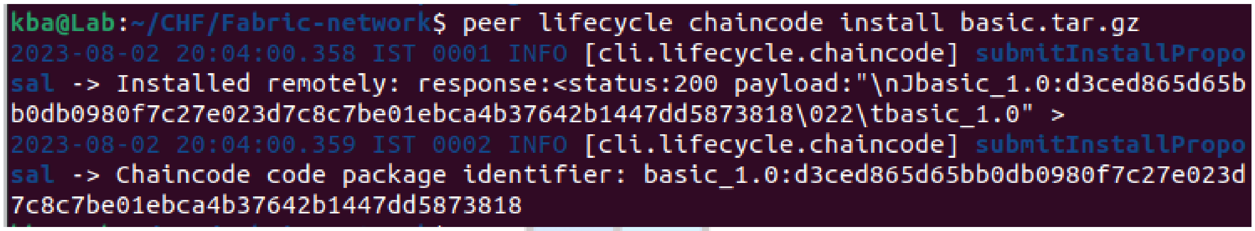
The chaincode must be installed on all the peers to successfully invoke any functions from a particular peer. The chaincode is installed using the **peer lifecycle**

**chaincode install** command.

### Install Chaincode on peer0 of Org1

#####Execute the command in **peer0_Org1** terminal#####

**peer lifecycle chaincode install KBA-Automobile.tar.gz**



```
kba@Lab:~/CHF/Fabric-network$ peer lifecycle chaincode install basic.tar.gz
2023-08-02 20:04:00.358 IST 0001 INFO [cli.lifecycle.chaincode] submitInstallPropo
sal -> Installed remotely: response:<status:200 payload:"\nJbasic_1.0:d3ced865d65b
b0db0980f7c27e023d7c8c7be01ebca4b37642b1447dd5873818\022\tbasic_1.0" >
2023-08-02 20:04:00.359 IST 0002 INFO [cli.lifecycle.chaincode] submitInstallPropo
sal -> Chaincode code package identifier: basic_1.0:d3ced865d65bb0db0980f7c27e023d
7c8c7be01ebca4b37642b1447dd5873818
```

The above command may take some time to give output. It will return the **package identifier**(Package ID). You can query the peer and get a list of the installed chaincode packages using the following command.

#####Execute the command in **peer0_Org1** terminal#####

**peer lifecycle chaincode queryinstalled**

### Install Chaincode on peer0 of Org2

#####Execute the command in **peer0_Org2** terminal#####

**peer lifecycle chaincode install KBA-Automobile.tar.gz**

**peer lifecycle chaincode queryinstalled**

## Approve Chaincode

After installing the chaincode package, you need to approve a chaincode definition for the organization. The chaincode is approved using the **peer lifecycle chaincode approveformyorg** command.

### Approve chaincode for peer0 of Org1

Set the package ID as environment variable.

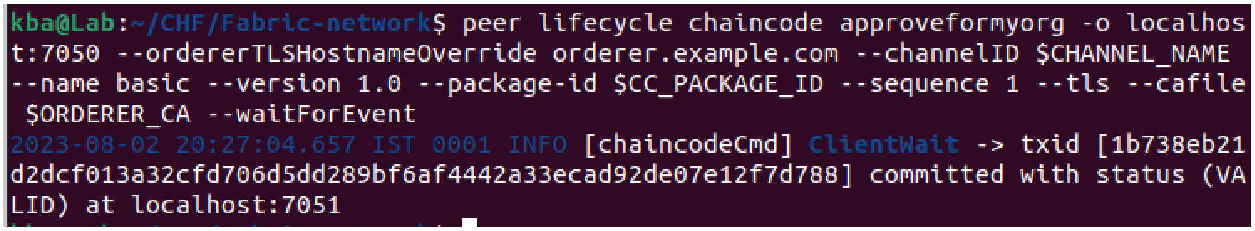
#####Execute the command in **peer0_Org1** terminal#####

**export CC_PACKAGE_ID=$(peer lifecycle chaincode calculatepackageid KBA-Automobile.tar.gz)**

Then execute the **peer lifecycle chaincode approveformyorg** command.

#####Execute the command in **peer0_Org1** terminal#####

**peer lifecycle chaincode approveformyorg -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --channelID $CHANNEL_NAME --name kbaautomobile --version 1.0 --package-id $CC_PACKAGE_ID --sequence 1 --collections-config ../Chaincode/collection.json --tls --cafile $ORDERER_CA --waitForEvent**



```
kba@Lab:~/CHF/Fabric-network$ peer lifecycle chaincode approveformyorg -o localhos
t:7050 --ordererTLSHostnameOverride orderer.example.com --channelID $CHANNEL_NAME
--name basic --version 1.0 --package-id $CC_PACKAGE_ID --sequence 1 --tls --cafile
 $ORDERER_CA --waitForEvent
2023-08-02 20:27:04.657 IST 0001 INFO [chaincodeCmd] ClientWait -> txid [1b738eb21
d2dcf013a32cfd706d5dd289bf6af4442a33ecad92de07e12f7d788] committed with status (VA
LID) at localhost:7051
```

Here, **sequence** parameter is used to track the number of times a chaincode has been defined or updated. Initially the sequence number is 1. When this chaincode is upgraded, the sequence number will be incremented to 2.

(**Note**: To define new endorsement policy use the flag **--signature-policy**

Eg: --signature-policy "OR('Org1MSP.peer', 'Org2MSP.peer')"

To implement PDC use the flag **--collections-config**

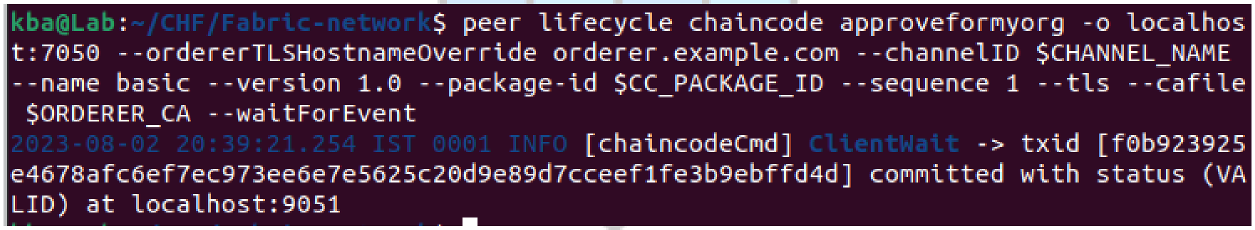
Eg: --collections-config ../Chaincode/collection.json )

**Approve chaincode for peer0 of Org2**

#####Execute the command in **peer0_Org2** terminal#####

**export CC_PACKAGE_ID=$(peer lifecycle chaincode calculatepackageid KBA-Automobile.tar.gz)**

**peer lifecycle chaincode approveformyorg -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --channelID $CHANNEL_NAME --name kbaautomobile --version 1.0 --package-id $CC_PACKAGE_ID --sequence 1 --collections-config ../Chaincode/collection.json --tls --cafile $ORDERER_CA --waitForEvent**



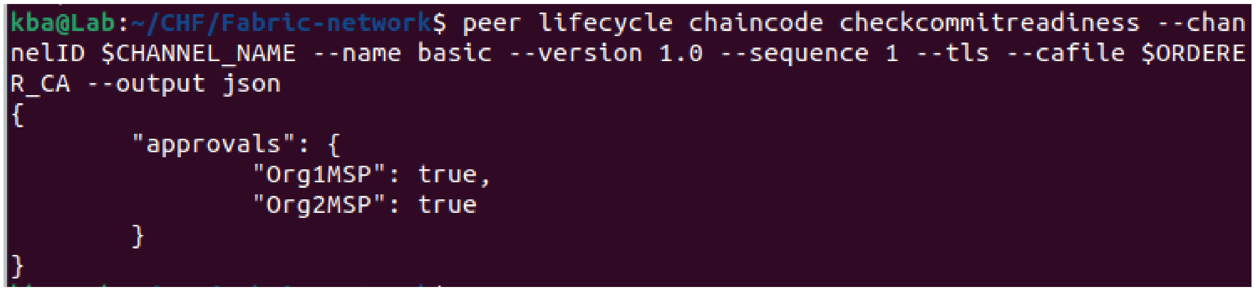
```
kba@Lab:~/CHF/Fabric-network$ peer lifecycle chaincode approveformyorg -o localhos
t:7050 --ordererTLSHostnameOverride orderer.example.com --channelID $CHANNEL_NAME
--name basic --version 1.0 --package-id $CC_PACKAGE_ID --sequence 1 --tls --cafile
 $ORDERER_CA --waitForEvent
2023-08-02 20:39:21.254 IST 0001 INFO [chaincodeCmd] ClientWait -> txid [f0b923925
e4678afc6ef7ec973ee6e7e5625c20d9e89d7cceef1fe3b9ebffd4d] committed with status (VA
LID) at localhost:9051
```

## Commit Chaincode

After a sufficient number of organizations have approved a chaincode definition, one organization can commit the chaincode definition to the channel. The **peer lifecycle chaincode checkcommitreadiness** command is used to check whether channel members have approved the same chaincode definition:

#####Execute the command in **peer0_Org1** terminal#####

**peer lifecycle chaincode checkcommitreadiness --channelID $CHANNEL_NAME --name kbaautomobile --version 1.0 --sequence 1 --tls --cafile $ORDERER_CA --output json**

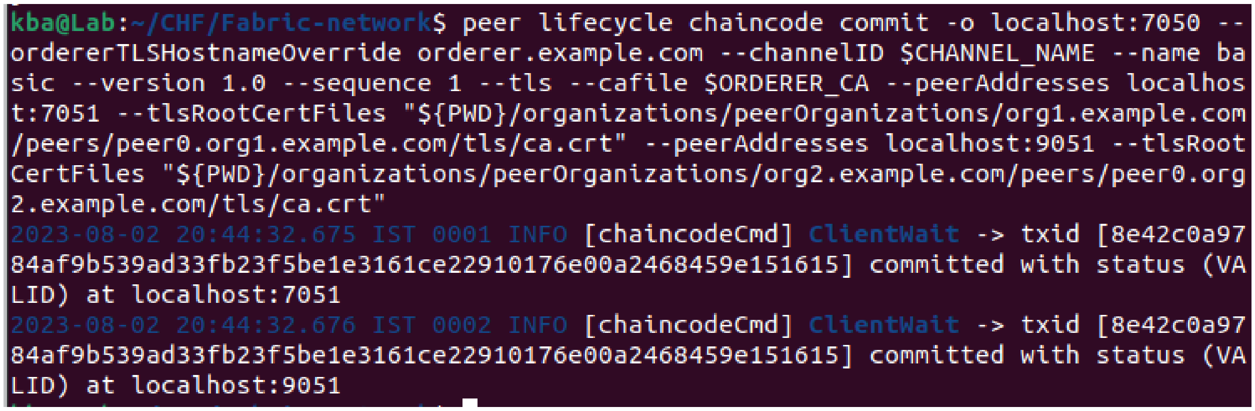


```
kba@Lab:~/CHF/Fabric-network$ peer lifecycle chaincode checkcommitreadiness --chan
nelID $CHANNEL_NAME --name basic --version 1.0 --sequence 1 --tls --cafile $ORDERE
R_CA --output json
{
        "approvals": {
                "Org1MSP": true,
                "Org2MSP": true
        }
}
```

Now, execute the **peer lifecycle chaincode commit** command.

#####Execute the command in peer0_Org1 terminal#####

**peer lifecycle chaincode commit -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --channelID $CHANNEL_NAME --name kbaautomobile --version 1.0 --sequence 1 --collections-config ../Chaincode/collection.json --tls --cafile $ORDERER_CA --peerAddresses localhost:7051 --tlsRootCertFiles $ORG1_PEER_TLSROOTCERT --peerAddresses localhost:9051 --tlsRootCertFiles $ORG2_PEER_TLSROOTCERT**



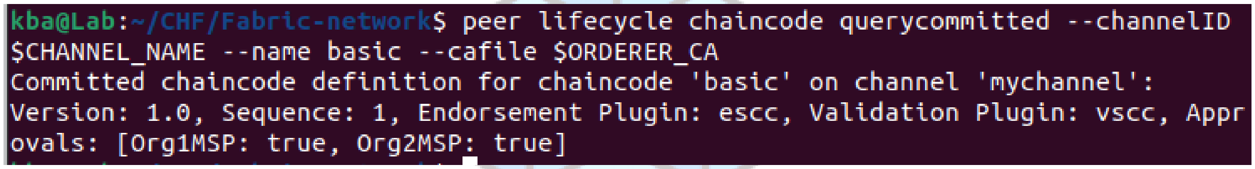
```
kba@Lab:~/CHF/Fabric-network$ peer lifecycle chaincode commit -o localhost:7050 --
ordererTLSHostnameOverride orderer.example.com --channelID $CHANNEL_NAME --name ba
sic --version 1.0 --sequence 1 --tls --cafile $ORDERER_CA --peerAddresses localhos
t:7051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org1.example.com
/peers/peer0.org1.example.com/tls/ca.crt" --peerAddresses localhost:9051 --tlsRoot
CertFiles "${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org
2.example.com/tls/ca.crt"
2023-08-02 20:44:32.675 IST 0001 INFO [chaincodeCmd] ClientWait -> txid [8e42c0a97
84af9b539ad33fb23f5be1e3161ce22910176e00a2468459e151615] committed with status (VA
LID) at localhost:7051
2023-08-02 20:44:32.676 IST 0002 INFO [chaincodeCmd] ClientWait -> txid [8e42c0a97
84af9b539ad33fb23f5be1e3161ce22910176e00a2468459e151615] committed with status (VA
LID) at localhost:9051
```

The transaction above uses the **--peerAddresses** flag to target peer0.org1.example.com from Org1 and peer0.org2.example.com from Org2. The commit transaction is submitted to the peers joined to the channel to query the chaincode definition that was approved by the organization that operates the peer. The command needs to target the peers from a sufficient number of organizations to satisfy the policy for deploying a chaincode. Because the approval is distributed within each organization, you can target any peer that belongs to a channel member.

We can now use the **peer lifecycle chaincode querycommitted** command to confirm that the chaincode definition was committed to our channel:

#####Execute the command in **peer0_Org1** terminal#####

**peer lifecycle chaincode querycommitted --channelID $CHANNEL_NAME --name kbaautomobile --cafile $ORDERER_CA**



```
kba@Lab:~/CHF/Fabric-network$ peer lifecycle chaincode querycommitted --channelID
$CHANNEL_NAME --name basic --cafile $ORDERER_CA
Committed chaincode definition for chaincode 'basic' on channel 'mychannel':
Version: 1.0, Sequence: 1, Endorsement Plugin: escc, Validation Plugin: vscc, Appr
ovals: [Org1MSP: true, Org2MSP: true]
```

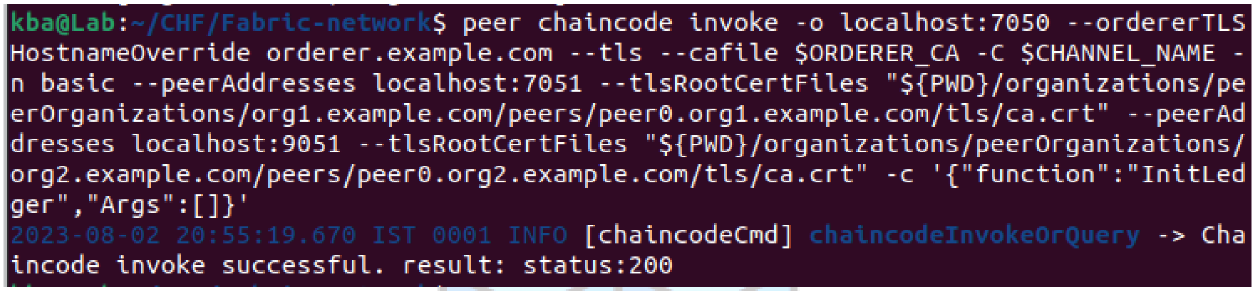
## Invoking the Chaincode

### Invoke Chaincode As Org1 Peer

After the chaincode definition has been committed to a channel, the chaincode will start on the peers joined to the channel where the chaincode was installed. The asset-transfer (basic) chaincode is now ready to be invoked by client applications. Use the following command to create an initial set of assets on the ledger. Note that the invoke command needs to target a sufficient number of peers to meet the chaincode endorsement policy. For invoking, we will be using the **peer chaincode invoke** command.

#####Execute the command in **peer0_Org1** terminal#####

**peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile $ORDERER_CA -C $CHANNEL_NAME -n kbaautomobile --peerAddresses localhost:7051 --tlsRootCertFiles $ORG1_PEER_TLSROOTCERT --peerAddresses localhost:9051 --tlsRootCertFiles $ORG2_PEER_TLSROOTCERT -c '{"function":"CreateCar","Args":["Car-11", "Tata", "Nexon", "White", "Factory-1", "22/07/2024"]}'**



```
kba@Lab:~/CHF/Fabric-network$ peer chaincode invoke -o localhost:7050 --ordererTLS
HostnameOverride orderer.example.com --tls --cafile $ORDERER_CA -C $CHANNEL_NAME -
n basic --peerAddresses localhost:7051 --tlsRootCertFiles "${PWD}/organizations/pe
erOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt" --peerAd
dresses localhost:9051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/
org2.example.com/peers/peer0.org2.example.com/tls/ca.crt" -c '{"function":"InitLed
ger","Args":[]}'
2023-08-02 20:55:19.670 IST 0001 INFO [chaincodeCmd] chaincodeInvokeOrQuery -> Cha
incode invoke successful. result: status:200
```

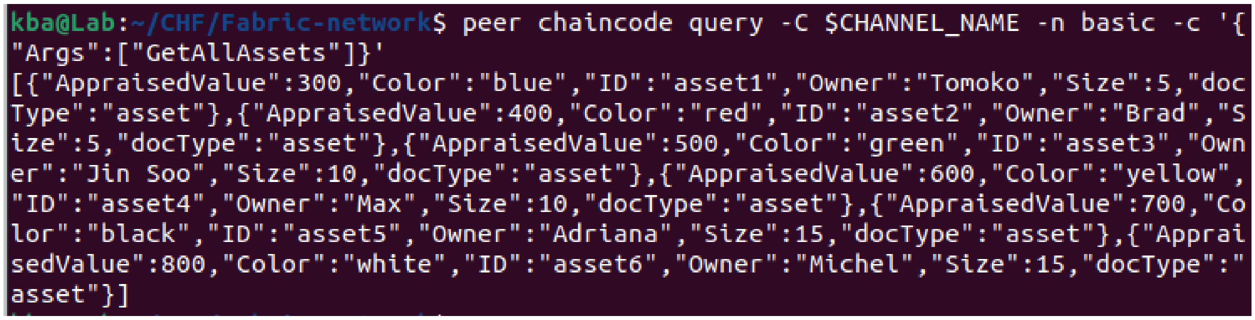
## Querying the chaincode

For querying we will be using the **peer chaincode query** command.

### Query Chaincode as peer0 of Org1

#####Execute the command in **peer0_Org1** terminal#####

**peer chaincode query -C $CHANNEL_NAME -n kbaautomobile -c '{"Args":["GetAllCars"]}'**



```
kba@Lab:~/CHF/Fabric-network$ peer chaincode query -C $CHANNEL_NAME -n basic -c '{
"Args":["GetAllAssets"]}'
[{"AppraisedValue":300,"Color":"blue","ID":"asset1","Owner":"Tomoko","Size":5,"doc
Type":"asset"},{"AppraisedValue":400,"Color":"red","ID":"asset2","Owner":"Brad","S
ize":5,"docType":"asset"},{"AppraisedValue":500,"Color":"green","ID":"asset3","Own
er":"Jin Soo","Size":10,"docType":"asset"},{"AppraisedValue":600,"Color":"yellow",
"ID":"asset4","Owner":"Max","Size":10,"docType":"asset"},{"AppraisedValue":700,"Co
lor":"black","ID":"asset5","Owner":"Adriana","Size":15,"docType":"asset"},{"Apprai
sedValue":800,"Color":"white","ID":"asset6","Owner":"Michel","Size":15,"docType":"
asset"}]
```

### Query Chaincode as peer0 of Org2

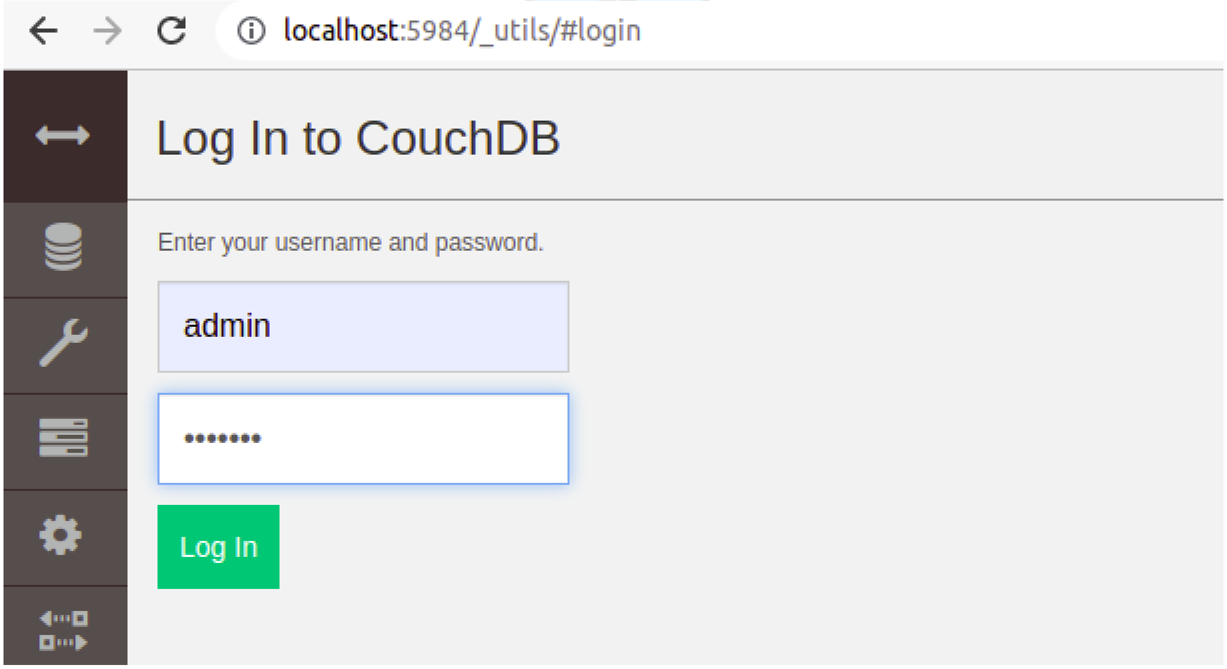
#####Execute the command in **peer0_Org2** terminal#####

```
peer chaincode query -C $CHANNEL_NAME -n kbaautomobile -c
'{"Args":["GetAllCars"]}'
```

You will get the same result.

## Viewing the data in CouchDB

In a browser, give the url as http://localhost:5984/_utils . You can give either 5984 or 7984 as port number. Give the username as **admin** and password as **adminpw**, which were set in the couchDb service in docker-compose file.



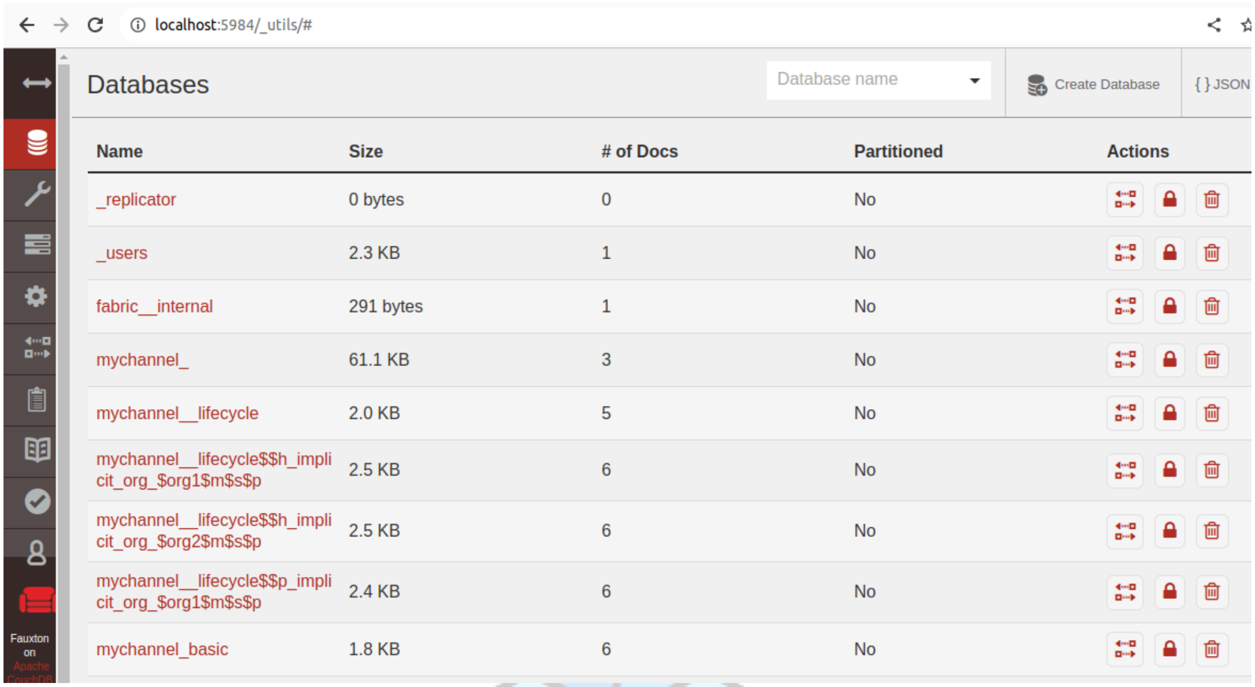
localhost:5984/_utils/#login

Log In to CouchDB

Enter your username and password.

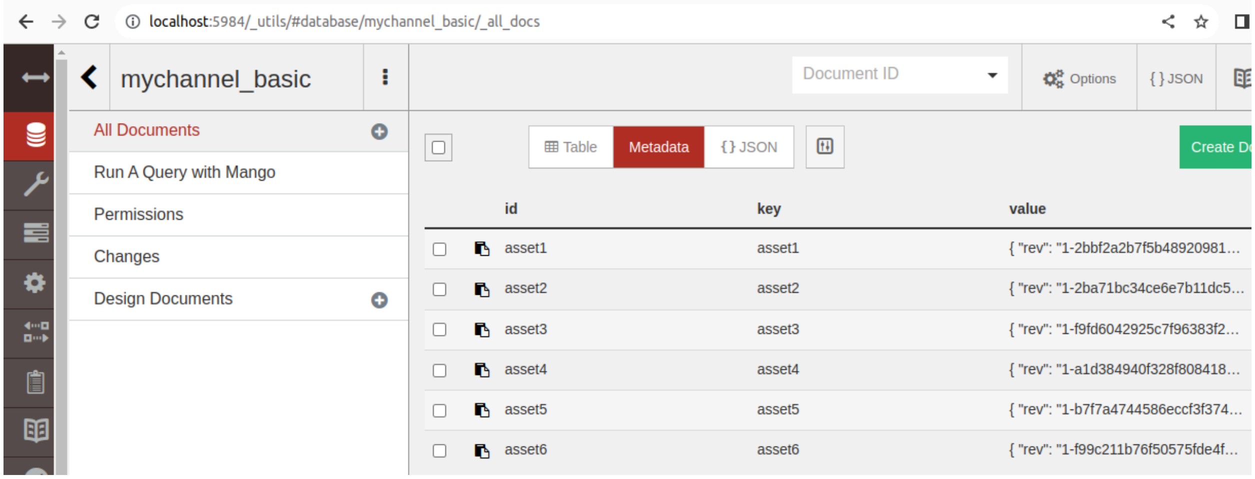
admin

•••••••



| Name | Size | # of Docs | Partitioned | Actions |
|---|---|---|---|---|
| _replicator | 0 bytes | 0 | No | |
| _users | 2.3 KB | 1 | No | |
| fabric__internal | 291 bytes | 1 | No | |
| mychannel_ | 61.1 KB | 3 | No | |
| mychannel__lifecycle | 2.0 KB | 5 | No | |
| mychannel__lifecycle$$h_implicit_org_$org1$m$s$p | 2.5 KB | 6 | No | |
| mychannel__lifecycle$$h_implicit_org_$org2$m$s$p | 2.5 KB | 6 | No | |
| mychannel__lifecycle$$p_implicit_org_$org1$m$s$p | 2.4 KB | 6 | No | |
| mychannel_basic | 1.8 KB | 6 | No | |

Click on **mychannel_basic**, to view the data.



| | id | key | value |
|---|---|---|---|
| □ | asset1 | asset1 | { "rev": "1-2bbf2a2b7f5b48920981… |
| □ | asset2 | asset2 | { "rev": "1-2ba71bc34ce6e7b11dc5… |
| □ | asset3 | asset3 | { "rev": "1-f9fd6042925c7f96383f2… |
| □ | asset4 | asset4 | { "rev": "1-a1d384940f328f808418… |
| □ | asset5 | asset5 | { "rev": "1-b7f7a4744586eccf3f374… |
| □ | asset6 | asset6 | { "rev": "1-f99c211b76f50575fde4f… |

# Bring down the network

Execute the following commands to stop the network.

Note: Execute the following commands within the **Fabric-network** folder.

#####Execute the command in **host** terminal#####

```
docker-compose -f docker/docker-compose-2org.yaml down

docker-compose -f docker/docker-compose-ca.yaml down

docker volume rm $(docker volume ls -q)
```

Remove all the crypto material generated for the network.

```
#####Execute the command in host terminal#####

sudo rm -rf channel-artifacts/

sudo rm -rf organizations/

sudo rm KBA-Automobile.tar.gz
```

Now, when you try the **docker ps** command, if any containers are listed, then execute the following commands.

```
docker rm $(docker container ls -q) --force

docker system prune

docker volume prune

docker network prune
```