# Practical 4

**Aim: Deploy an asset-transfer app using block chain. Learn app development within a Hyperledger Fabric network.**

### Installing dependencies

```
sudo apt install jq
```

### Start the network and deploy the smart contract

```
cd fabric-samples/test-network
./network.sh down
```

```
→ web3 cd fabric-samples/test-network
```

```
→ test-network git:(v2.2.0) ./network.sh down
Stopping network
Stopping cli                      ... done
Stopping peer0.org2.example.com ... done
Stopping peer0.org1.example.com ... done
```

```
./network.sh up createChannel -ca
```

```
→ test-network git:(v2.2.0) ./network.sh up createChannel -ca
Creating channel 'mychannel'.
If network is not up, starting nodes with CLI timeout of '5' tries and CLI de
d using database 'leveldb with crypto from 'Certificate Authorities'
Bringing up network
LOCAL_VERSION=2.2.0
DOCKER_IMAGE_VERSION=2.2.0
CA_LOCAL_VERSION=v1.5.7
CA_DOCKER_IMAGE_VERSION=v1.5.7
Generating certificates using Fabric CA
Creating network "fabric_test" with the default driver
Creating ca_org2    ... done
```

You can then use the test network script to deploy the *asset-transfer-abac smart* contract to a channel on the network:

```
./network.sh deployCC -ccn abac -ccp ../asset-transfer-abac/chaincode-go/ -ccl go
```

```
→ test-network git:(v2.2.0) ./network.sh deployCC -ccn abac -ccp ../asset-transfer-abac/chaincode-go/ -ccl go
deploying chaincode on channel 'mychannel'
executing with the following
- CHANNEL_NAME: mychannel
- CC_NAME: abac
- CC_SRC_PATH: ../asset-transfer-abac/chaincode-go/
- CC_SRC_LANGUAGE: go
- CC_VERSION: 1.0
- CC_SEQUENCE: 1
```

**Register identities with attributes**

We can use the one of the test network Certificate Authorities to register and enroll identities with the attribute of **abac.creator=true**. First, we need to set the following environment variables in order to use the Fabric CA client.

```
export PATH=${PWD}/../bin:${PWD}:$PATH
export FABRIC_CFG_PATH=$PWD/../config/
```

We will create the identities using the Org1 CA. Set the Fabric CA client home to the MSP of the Org1 CA admin:

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

There are two ways to generate certificates with attributes added. We will use both methods and create two identities in the process. The first method is to specify that the attribute be added to the certificate by default when the identity is registered. The following command will register an identity named creator1 with the attribute of **abac.creator=true**.

```
→ test-network git:(v2.2.0) export PATH=${PWD}/../bin:${PWD}:$PATH
→ test-network git:(v2.2.0) export FABRIC_CFG_PATH=$PWD/../config/
→ test-network git:(v2.2.0) export FABRIC_CA_CLIENT_HOME=${PWD}/organizations/peerOrganizations/org1.example.com/
```

```
fabric-ca-client register --id.name creator1 --id.secret
creator1pw --id.type client --id.affiliation org1 --id.attrs
'abac.creator=true:ecert' --tls.certfiles
"${PWD}/organizations/fabric-ca/org1/tls-cert.pem"
```

```
→ test-network git:(v2.2.0) fabric-ca-client register --id.name creator1 --id.secret creator1pw --id.type client
--id.affiliation org1 --id.attrs 'abac.creator=true:ecert' --tls.certfiles "${PWD}/organizations/fabric-ca/org1/tl
s-cert.pem"
2024/02/15 10:53:16 [INFO] Configuration file location: /home/hackerman/web3/fabric-samples/test-network/organizat
ions/peerOrganizations/org1.example.com/fabric-ca-client-config.yaml
2024/02/15 10:53:16 [INFO] TLS Enabled
2024/02/15 10:53:16 [INFO] TLS Enabled
Password: creator1pw
```

The **ecert** suffix adds the attribute to the certificate automatically when the identity is enrolled. As a result, the following enroll command will contain the attribute that was provided in the registration command.

```
fabric-ca-client enroll -u
https://creator1:creator1pw@localhost:7054 --caname ca-org1 -M
"${PWD}/organizations/peerOrganizations/org1.example.com/users/cre
```

```
ator1@org1.example.com/msp" --tls.certfiles
"${PWD}/organizations/fabric-ca/org1/tls-cert.pem"
```

```
→ test-network git:(v2.2.0) fabric-ca-client enroll -u https://creator1:creator1pw@localhost:7054 --caname ca-org
1 -M "${PWD}/organizations/peerOrganizations/org1.example.com/users/creator1@org1.example.com/msp" --tls.certfiles
 "${PWD}/organizations/fabric-ca/org1/tls-cert.pem"
2024/02/15 10:53:36 [INFO] TLS Enabled
2024/02/15 10:53:36 [INFO] generating key: &{A:ecdsa S:256}
2024/02/15 10:53:36 [INFO] encoded CSR
2024/02/15 10:53:36 [INFO] Stored client certificate at /home/hackerman/web3/fabric-samples/test-network/organizat
ions/peerOrganizations/org1.example.com/users/creator1@org1.example.com/msp/signcerts/cert.pem
2024/02/15 10:53:36 [INFO] Stored root CA certificate at /home/hackerman/web3/fabric-samples/test-network/organiza
tions/peerOrganizations/org1.example.com/users/creator1@org1.example.com/msp/cacerts/localhost-7054-ca-org1.pem
2024/02/15 10:53:36 [INFO] Stored Issuer public key at /home/hackerman/web3/fabric-samples/test-network/organizati
ons/peerOrganizations/org1.example.com/users/creator1@org1.example.com/msp/IssuerPublicKey
2024/02/15 10:53:36 [INFO] Stored Issuer revocation public key at /home/hackerman/web3/fabric-samples/test-network
/organizations/peerOrganizations/org1.example.com/users/creator1@org1.example.com/msp/IssuerRevocationPublicKey
→ test-network git:(v2.2.0)
```

Now that we have enrolled the identity, run the command below to copy the Node OU configuration file into the creator1 MSP folder.

```
cp
"${PWD}/organizations/peerOrganizations/org1.example.com/msp/confi
g.yaml"
"${PWD}/organizations/peerOrganizations/org1.example.com/users/cre
ator1@org1.example.com/msp/config.yaml"
```

The second method is to request that the attribute be added upon enrollment. The following command will register an identity named creator2 with the same **abac.creator** attribute.

```
fabric-ca-client register --id.name creator2 --id.secret
creator2pw --id.type client --id.affiliation org1 --id.attrs
'abac.creator=true:' --tls.certfiles
"${PWD}/organizations/fabric-ca/org1/tls-cert.pem"
```

```
→ test-network git:(v2.2.0) cp "${PWD}/organizations/peerOrganizations/org1.example.com/msp/config.yaml" "${PWD}/
organizations/peerOrganizations/org1.example.com/users/creator1@org1.example.com/msp/config.yaml"
→ test-network git:(v2.2.0) fabric-ca-client register --id.name creator2 --id.secret creator2pw --id.type client
--id.affiliation org1 --id.attrs 'abac.creator=true:' --tls.certfiles "${PWD}/organizations/fabric-ca/org1/tls-cer
t.pem"
2024/02/15 10:54:00 [INFO] Configuration file location: /home/hackerman/web3/fabric-samples/test-network/organizat
ions/peerOrganizations/org1.example.com/fabric-ca-client-config.yaml
2024/02/15 10:54:00 [INFO] TLS Enabled
2024/02/15 10:54:00 [INFO] TLS Enabled
Password: creator2pw
→ test-network git:(v2.2.0)
```

The following enroll command will add the attribute to the certificate:

```
fabric-ca-client enroll -u
https://creator2:creator2pw@localhost:7054 --caname ca-org1
```

```
--enrollment.attrs "abac.creator" -M
"${PWD}/organizations/peerOrganizations/org1.example.com/users/cre
ator2@org1.example.com/msp" --tls.certfiles
"${PWD}/organizations/fabric-ca/org1/tls-cert.pem"
```

```
→ test-network git:(v2.2.0) fabric-ca-client enroll -u https://creator2:creator2pw@localhost:7054 --caname ca-org
1 --enrollment.attrs "abac.creator" -M "${PWD}/organizations/peerOrganizations/org1.example.com/users/creator2@org
1.example.com/msp" --tls.certfiles "${PWD}/organizations/fabric-ca/org1/tls-cert.pem"
2024/02/15 10:54:20 [INFO] TLS Enabled
2024/02/15 10:54:20 [INFO] generating key: &{A:ecdsa S:256}
2024/02/15 10:54:20 [INFO] encoded CSR
2024/02/15 10:54:20 [INFO] Stored client certificate at /home/hackerman/web3/fabric-samples/test-network/organizat
ions/peerOrganizations/org1.example.com/users/creator2@org1.example.com/msp/signcerts/cert.pem
2024/02/15 10:54:20 [INFO] Stored root CA certificate at /home/hackerman/web3/fabric-samples/test-network/organiza
tions/peerOrganizations/org1.example.com/users/creator2@org1.example.com/msp/cacerts/localhost-7054-ca-org1.pem
2024/02/15 10:54:20 [INFO] Stored Issuer public key at /home/hackerman/web3/fabric-samples/test-network/organizati
ons/peerOrganizations/org1.example.com/users/creator2@org1.example.com/msp/IssuerPublicKey
2024/02/15 10:54:20 [INFO] Stored Issuer revocation public key at /home/hackerman/web3/fabric-samples/test-network
/organizations/peerOrganizations/org1.example.com/users/creator2@org1.example.com/msp/IssuerRevocationPublicKey
→ test-network git:(v2.2.0)
```

Run the command below to copy the Node OU configuration file into the creator2 MSP folder.

```
cp
"${PWD}/organizations/peerOrganizations/org1.example.com/msp/confi
g.yaml"
"${PWD}/organizations/peerOrganizations/org1.example.com/users/cre
ator2@org1.example.com/msp/config.yaml"
```

```
→ test-network git:(v2.2.0) cp "${PWD}/organizations/peerOrganizations/org1.example.com/msp/config.yaml" "${PWD}/
organizations/peerOrganizations/org1.example.com/users/creator2@org1.example.com/msp/config.yaml"
→ test-network git:(v2.2.0)
```

### Create an asset

You can use either identity with the **abac.creator=true** attribute to create an asset using the **asset-transfer-abac** smart contract. We will set the following environment variables to use the first identity that was generated, creator1:

```
export CORE_PEER_TLS_ENABLED=true
export CORE_PEER_LOCALMSPID="Org1MSP"
export
CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org1.ex
ample.com/users/creator1@org1.example.com/msp
export
CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/peerOrganizations/org
1.example.com/peers/peer0.org1.example.com/tls/ca.crt
export CORE_PEER_ADDRESS=localhost:7051
export TARGET_TLS_OPTIONS=(-o localhost:7050
```

```
--ordererTLSHostnameOverride orderer.example.com --tls --cafile
"${PWD}/organizations/ordererOrganizations/example.com/orderers/ordere
r.example.com/msp/tlscacerts/tlsca.example.com-cert.pem"
--peerAddresses localhost:7051 --tlsRootCertFiles
"${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.o
rg1.example.com/tls/ca.crt" --peerAddresses localhost:9051
--tlsRootCertFiles
"${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.o
rg2.example.com/tls/ca.crt")
```

```
→ test-network git:(v2.2.0) export CORE_PEER_TLS_ENABLED=true
→ test-network git:(v2.2.0) export CORE_PEER_LOCALMSPID="Org1MSP"
→ test-network git:(v2.2.0) export CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org1.example.co
m/users/creator1@org1.example.com/msp
→ test-network git:(v2.2.0) export CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/peerOrganizations/org1.exampl
e.com/peers/peer0.org1.example.com/tls/ca.crt
→ test-network git:(v2.2.0) export CORE_PEER_ADDRESS=localhost:7051
→ test-network git:(v2.2.0) export TARGET_TLS_OPTIONS=(-o localhost:7050 --ordererTLSHostnameOverride orderer.exa
mple.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tl
scacerts/tlsca.example.com-cert.pem" --peerAddresses localhost:7051 --tlsRootCertFiles "${PWD}/organizations/peerO
rganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt" --peerAddresses localhost:9051 --tlsRootCer
tFiles "${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt")

→ test-network git:(v2.2.0)
```

Run the following command to create Asset1:

```
peer chaincode invoke "${TARGET_TLS_OPTIONS[@]}" -C mychannel -n abac -c
'{"function":"CreateAsset","Args":["Asset1","blue","20","100"]}'
```

```
→ test-network git:(v2.2.0) peer chaincode invoke "${TARGET_TLS_OPTIONS[@]}" -C mychannel -n abac -c '{"function"
:"CreateAsset","Args":["Asset1","blue","20","100"]}'
2024-02-15 10:56:19.843 IST [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 001 Chaincode invoke successful. result:
 status:200
→ test-network git:(v2.2.0)
```

You can use the command below to query the asset on the ledger:

```
peer chaincode query -C mychannel -n abac -c
'{"function":"ReadAsset","Args":["Asset1"]}'
```

The result will list the creator1 identity as the asset owner.

```
→ test-network git:(v2.2.0) peer chaincode query -C mychannel -n abac -c '{"function":"ReadAsset","Args":["Asset1
"]}'
{"ID":"Asset1","color":"blue","size":20,"owner":"x509::CN=creator1,OU=org1+OU=client,O=Hyperledger,ST=North Caroli
na,C=US::CN=ca.org1.example.com,O=org1.example.com,L=Durham,ST=North Carolina,C=US","appraisedValue":100}
→ test-network git:(v2.2.0)
```

### Transfer the asset

As the owner of Asset1, the creator1 identity has the ability to transfer the asset to another owner. In order to transfer the asset, the owner needs to provide the name and issuer of the new owner to the **TransferAsset** function. The **asset-transfer-abac** smart contract has a **GetSubmittingClientIdentity** function that allows users to retrieve their certificate information and provide it to the asset owner out of band (we omit this step). Issue the command below to transfer Asset1 to the user1 identity from Org1 that was created when the test network was deployed:

```
export RECIPIENT="x509::CN=user1,OU=client,O=Hyperledger,ST=North
Carolina,C=US::CN=ca.org1.example.com,O=org1.example.com,L=Durham,
ST=North Carolina,C=US"

peer chaincode invoke "${TARGET_TLS_OPTIONS[@]}" -C mychannel -n
abac -c
'{"function":"TransferAsset","Args":["Asset1","'"$RECIPIENT"'"]}'
```

```
→ test-network git:(v2.2.0) export RECIPIENT="x509::CN=user1,OU=client,O=Hyperledger,ST=North Carolina,C=US::CN=c
a.org1.example.com,O=org1.example.com,L=Durham,ST=North Carolina,C=US"
→ test-network git:(v2.2.0) peer chaincode invoke "${TARGET_TLS_OPTIONS[@]}" -C mychannel -n abac -c '{"function"
:"TransferAsset","Args":["Asset1","'"$RECIPIENT"'"]}'
2024-02-15 10:57:13.212 IST [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 001 Chaincode invoke successful. result:
 status:200
→ test-network git:(v2.2.0)
```

Query the ledger to verify that the asset has a new owner:

```
peer chaincode query -C mychannel -n abac -c
'{"function":"ReadAsset","Args":["Asset1"]}'
```

We can see that Asset1 with is now owned by User1:

```
→ test-network git:(v2.2.0) peer chaincode query -C mychannel -n abac -c '{"function":"ReadAsset","Args":["Asset1
"]}'
{"ID":"Asset1","color":"blue","size":20,"owner":"x509::CN=user1,OU=client,O=Hyperledger,ST=North Carolina,C=US::CN
=ca.org1.example.com,O=org1.example.com,L=Durham,ST=North Carolina,C=US","appraisedValue":100}
→ test-network git:(v2.2.0)
```

### Update the asset

Now that the asset has been transferred, the new owner can update the asset properties. The smart contract uses the **GetID()** API to ensure that the update is being submitted by the asset owner. To demonstrate the difference between identity and attribute based access control, lets try to update the asset using the creator1 identity first:

```
peer chaincode invoke "${TARGET_TLS_OPTIONS[@]}" -C mychannel -n
abac -c
'{"function":"UpdateAsset","Args":["Asset1","green","20","100"]}'
```

Even though creator1 can create new assets, the smart contract detects that the transaction was not submitted by the identity that owns the asset, user1. The command returns the following error:

Error: endorsement failure during invoke. response: status:500 message:"submitting client not authorized to update asset, does not own asset"

```
→ test-network git:(v2.2.0) peer chaincode invoke "${TARGET_TLS_OPTIONS[@]}" -C mychannel -n abac -c '{"function"
:"UpdateAsset","Args":["Asset1","green","20","100"]}'
Error: endorsement failure during invoke. response: status:500 message:"submitting client not authorized to update
 asset, does not own asset"
→ test-network git:(v2.2.0)
```

Run the following command to operate as the asset owner by setting the MSP path to User1:

```
export
CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org
1.example.com/users/User1@org1.example.com/msp
```

```
→ test-network git:(v2.2.0) export CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org1.example.co
m/users/User1@org1.example.com/msp
→ test-network git:(v2.2.0)
```

We can now update the asset. Run the following command to change the asset color from blue to green. All other aspects of the asset will remain unchanged.

```
peer chaincode invoke "${TARGET_TLS_OPTIONS[@]}" -C mychannel -n
abac -c
'{"function":"UpdateAsset","Args":["Asset1","green","20","100"]}'
```

```
→ test-network git:(v2.2.0) peer chaincode invoke "${TARGET_TLS_OPTIONS[@]}" -C mychannel -n abac -c '{"function"
:"UpdateAsset","Args":["Asset1","green","20","100"]}'
2024-02-15 10:58:29.739 IST [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 001 Chaincode invoke successful. result:
 status:200
→ test-network git:(v2.2.0)
```

Run the query command again to verify that the asset has changed color:

```
peer chaincode query -C mychannel -n abac -c
```

```
'{"function":"ReadAsset","Args":["Asset1"]}'
```

The result will display that Asset1 is now green:

```
→ test-network git:(v2.2.0) peer chaincode query -C mychannel -n abac -c '{"function":"ReadAsset","Args":["Asset1
"]}'
{"ID":"Asset1","color":"green","size":20,"owner":"x509::CN=user1,OU=client,O=Hyperledger,ST=North Carolina,C=US::C
N=ca.org1.example.com,O=org1.example.com,L=Durham,ST=North Carolina,C=US","appraisedValue":100}
→ test-network git:(v2.2.0)
```

**Delete the asset**

The owner also has the ability to delete the asset. Run the following command to remove Asset1 from the ledger:

```
peer chaincode invoke "${TARGET_TLS_OPTIONS[@]}" -C mychannel -n
abac -c '{"function":"DeleteAsset","Args":["Asset1"]}'
```

```
→ test-network git:(v2.2.0) peer chaincode invoke "${TARGET_TLS_OPTIONS[@]}" -C mychannel -n abac -c '{"function"
:"DeleteAsset","Args":["Asset1"]}'
2024-02-15 10:59:09.373 IST [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 001 Chaincode invoke successful. result:
 status:200
→ test-network git:(v2.2.0)
```

If you query the ledger once more, you will see that Asset1 no longer exists:

```
peer chaincode query -C mychannel -n abac -c
'{"function":"ReadAsset","Args":["Asset1"]}'
```

```
→ test-network git:(v2.2.0) peer chaincode query -C mychannel -n abac -c '{"function":"ReadAsset","Args":["Asset1
"]}'
Error: endorsement failure during query. response: status:500 message:"the asset Asset1 does not exist"
→ test-network git:(v2.2.0)
```

While we are operating as User1, we can demonstrate attribute based access control by trying to create an asset using an identity without the **abac.creator=true** attribute. Run the following command to try to create Asset1 as User1:

```
peer chaincode invoke "${TARGET_TLS_OPTIONS[@]}" -C mychannel -n
abac -c
'{"function":"CreateAsset","Args":["Asset2","red","20","100"]}'
```

The smart contract will return the following error:

Error: endorsement failure during invoke. response: status:500 message:"submitting client not authorized to create asset, does not have abac.creator role"

```
→ test-network git:(v2.2.0) peer chaincode invoke "${TARGET_TLS_OPTIONS[@]}" -C mychannel -n abac -c '{"function"
:"CreateAsset","Args":["Asset2","red","20","100"]}'
Error: endorsement failure during invoke. response: status:500 message:"submitting client not authorized to create
 asset, does not have abac.creator role"
→ test-network git:(v2.2.0)
```

### Clean up

When you are finished, you can run the following command to bring down the test network:

```
./network.sh down
```

```
→ test-network git:(v2.2.0) ./network.sh down
Stopping network
Stopping cli                      ... done
Stopping peer0.org2.example.com ... done
Stopping peer0.org1.example.com ... done
Stopping orderer.example.com    ... done
Stopping ca_org1                ...
Stopping ca_orderer             ...
Stopping ca_org2                ...
```