# Hyperledger Fabric Roles/Identities:

**Guide to Hyperledger Fabric deployment:** [**https://media.readthedocs.org/pdf/hyperledger-fabric/latest/hyperledger-fabric.pdf**](https://media.readthedocs.org/pdf/hyperledger-fabric/latest/hyperledger-fabric.pdf) **(\*\*\*Good resource to follow on actual deployment sequence with ownership of components clearly explained\*\*\* Page 25, section 2.4.2)**

## Deployment Roles in Hyperledger Fabric:

Network config

Network

Network Admin(s)

Channel 1

Channel Admin(s)

Org 1 Admin(s)

Member(s)

Org 2 Admin(s)

Member(s)

Channel 2

Channel Admin(s)

Org 1 Admin(s)

Member(s)

Org 2 Admin(s)

Member(s)

Channel config

Channel config

**Roles:**

* Network Admin – authorized to
  + Run the orderer(s)
  + Create and update the Network configuration
  + Add new orgs to network
  + Allow creation of new channels/consortia between orgs

Not authorized to

* + Participate in channel affairs, unless part of the channel structure
  + Read/write to ledger on peers, unless part of the channel structure
* Channel Admin – authorized to
  + Add peers and join them to the channel
  + Install chaincode on the peers (these peers can see the logic/code and endorse them)
  + Instantiate chaincode on the channel (logical view of chaincode)
  + Decide endorsement policy for that channel
  + Create/register and modify member identities

Not authorized to

* + Change orderer or overall network settings
  + Add new orgs to the channel
* Member – authorized to
  + Invoke and query chaincode
  + Endorse proposals
  + Commit transactions
* CA Admin (??)

**Description**

**Owner/Network Admin** - The entity which runs the Orderer is deemed to be the owner of the network with Admin rights to configure the network. This entity could grant equal rights to other entities, so that they may also run nodes on the network. Other entities without explicit permissions may not modify this global Network Configuration. So only these entities could create a consortium of orgs.

**CA** – Each org typically has its own CA; the CA serves to provide identities (in the form of certificates) which are used to authenticate the various components as well as for signing transaction proposals during endorsement.

_images/fabric-ca.png

The diagram explains Fabric CA hierarchy. There are two ways of interacting with a Hyperledger Fabric CA server: via the Hyperledger Fabric CA client or through one of the Fabric SDKs. All communication to the Hyperledger Fabric CA server is via REST APIs.

**Channel/Consortium Admin** – Any one of the global network admins could provide another entity the permissions to create a consortium or channel. Once a channel is created, only the entities within that channel may configure it, i.e., global network admin does not have permissions within the channel, if it is not a member of that channel. When an org is given permissions to form a channel, it can add Peers within that channel; these peers maintain the ledger state exclusive for that channel. The identities issued by the CA of that org serve to bind the peers of the same org to the corresponding channel.

**System Channel** – Special channel used by the Ordering service.

## Identity in Fabric:

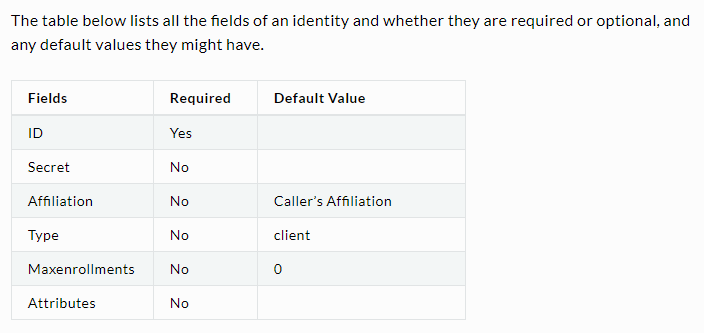
A [digital identity](https://hyperledger-fabric.readthedocs.io/en/release-1.2/identity/identity.html?highlight=principal) has some additional attributes that Fabric uses to determine permissions, and it gives the union of an identity and the associated attributes a special name — **principal**. **Principal**s are just like userIDs or groupIDs, but a little more flexible because they can include a wide range of properties of an actor’s identity.

A principal is described in terms of the MSP that is tasked to validate the identity of the signer and of the role that the signer has within that MSP. Four roles are supported: **member**, **admin**, **client**, and **peer**. Principals are described as MSP.ROLE, where MSP is the MSP ID that is required, and ROLE is one of the four strings member, admin, client and peer. Examples of valid principals are 'Org0.admin' (any administrator of the Org0 MSP) or 'Org1.member' (any member of the Org1 MSP), 'Org1.client' (any client of the Org1 MSP), and 'Org1.peer' (any peer of the Org1 MSP).

At the application/chaincode level, the following users are required for the basic ‘***fabcar***’ sample network (uses Fabric Node SDK to invoke and query the ledger using Chaincode):

1. ***Admin*** – An admin user (with enrollment Secret: adminpw) is already registered with the CA. This admin’s certificates are further used to enroll other users as needed.
2. ***User1*** - will be the identity we use when querying and updating the ledger.

Firstly, the Fabric CA client enrolls the admin user and then registers the ***User1*** identity as in the following link: <http://hyperledger-fabric-ca.readthedocs.io/en/latest/users-guide.html#adding-an-identity>



An example command to register an identity is as follows:

fabric**-**ca**-**client identity add user1 **--**json '{"secret": "user1pw", "type": "user", "affiliation": "org1", "max\_enrollments": 1, "attrs": [{"name": "**hf.Registrar.Roles**", "value": "peer,client"}, {"name": "hf.Revoker", "value": "true"}]}'

The attributes include fields like hf.Registrar.Roles (which tie back into the 4 possible roles of a principal)

## Types of Peers:

Peers can be one or more of the following types:

* [Endorsing peers](https://hyperledger-fabric.readthedocs.io/en/release-1.2/endorsement-policies.html) –

In addition to performing the role of peer, endorsement peers also simulate all proposed transactions to check for code determinism and code stability. Every peer with a smart contract can be an endorsing peer if it has a smart contract installed.

Endorsement policies are used to instruct a peer on how to decide whether a transaction is properly endorsed, including whether there is an appropriate number of endorsements coming from the correct peers.

The policy can be specified at instantiate time using the -P switch, followed by the policy.

For example:

peer chaincode instantiate **-**C **<**channelid**>** **-**n mycc **-**P "AND('Org1.member', 'Org2.member')"

This command deploys chaincode mycc with the policy AND('Org1.member', 'Org2.member') which would require that a member of both Org1 and Org2 sign the transaction.

* [Leading peers](https://hyperledger-fabric.readthedocs.io/en/release-1.2/gossip.html?highlight=gossip%20protocol#leader-election) –

The leader election mechanism is used to **elect** one peer per each organization which will maintain connection with ordering service and initiate distribution of newly arrived blocks across peers of its own organization. Leveraging leader election provides system with ability to efficiently utilize bandwidth of the ordering service. There are two possible operation modes for leader election module:

1. **Static** – system administrator manually configures one peer in the organization to be the leader, e.g. one to maintain open connection with the ordering service. See <https://hyperledger-fabric.readthedocs.io/en/release-1.2/gossip.html#static-leader-election>
2. **Dynamic** – peers execute a leader election procedure to select one peer in an organization to become leader, pull blocks from the ordering service, and disseminate blocks to the other peers in the organization. See <https://hyperledger-fabric.readthedocs.io/en/release-1.2/gossip.html#dynamic-leader-election>

* Anchor peers – When creating a channel, the anchor peers for each org are updated via “channel configuration transactions”; see Anchor Peers attribute at <https://hyperledger-fabric.readthedocs.io/en/release-1.2/config_update.html#editing-a-config>

[Anchor peers](https://hyperledger-fabric.readthedocs.io/en/release-1.2/gossip.html?highlight=gossip%20protocol#anchor-peers) are used to facilitate gossip communication between peers from **different** organizations. In order for cross-org gossip to work, peers from one org need to know **at least one address** of a peer from other orgs (from this peer, it can find out about all of the peers in that org). This address is the anchor peer, and it’s defined in the channel configuration.

Each organization that has a peer will have at least one of its peers (though it can be more than one) defined in the channel configuration as the anchor peer. Note that the anchor peer does not need to be the same peer as the leader peer.

Service discovery: <https://hyperledger-fabric.readthedocs.io/en/release-1.2/discovery-overview.html>

## Upgrading your Network:

<https://hyperledger-fabric.readthedocs.io/en/release-1.2/upgrading_your_network_tutorial.html>

At a high level, the upgrade process to move from v1.1 to v1.2 will perform the following steps:

1. Back up the ledger and MSPs.
2. Upgrade the orderer binaries to Fabric v1.2.
3. Upgrade the peer binaries to Fabric v1.2.
4. Enable the new v1.2 capability.

In production environments, the orderers and peers can simultaneously be upgraded on a rolling basis. In other words, you can upgrade the binaries in any order, without bringing down the network. Because BYFN uses a “SOLO” ordering service (one orderer), our script brings down the entire network. But this is not necessary in a production environment. Orderers and peers may be upgraded in parallel.

However, it is important to make sure that enabling capabilities will not create issues with the versions of orderers and peers that are currently running. For v1.2, the new capability is in the application group, which governs peer related functionalities, and as a result does not conflict with the ordering service.