Components of Hyperledger Fabric

By Coding-Bootcamps.com

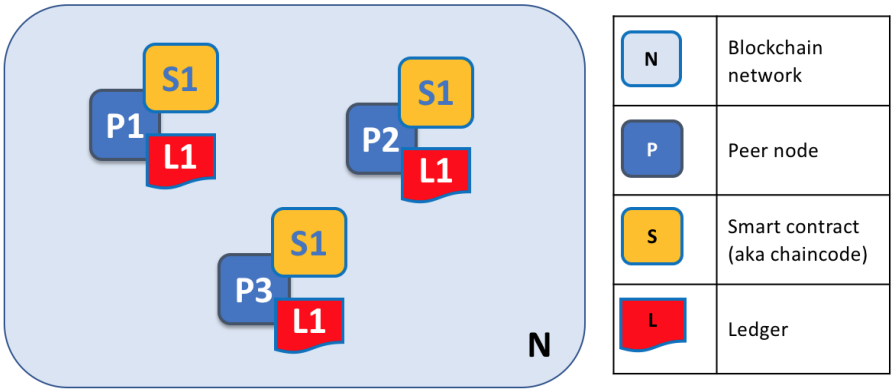
Here is the list of main Hyperledger Fabric components:

1. **Peer**
2. **Ordering Service**
3. **Fabric CA (Credential Authority)**
4. **Fabric Ledger and Chaincodes**
5. **Channel**
6. **Policies**
7. **Membership Services Provider (MSP)**

1. Peer

A blockchain network is comprised primarily of a set of *peer nodes* (or, simply, *peers*). Peers are a fundamental element of the network because they host ledgers and smart contracts. Recall that a ledger immutably records all the transactions generated by smart contracts (which in Hyperledger Fabric are contained in a *chaincode*). Smart contracts and ledgers are used to encapsulate the shared *processes* and shared *information* in a network, respectively. These aspects of a peer make them a good starting point to understand a Fabric network.

Other elements of the blockchain network are of course important: ledgers and smart contracts, orderers, policies, channels, applications, organizations, identities, and membership, and we will be covering them in our Hyperledger Fabric System Admin course in detail. In this course, we only focus on peers, and their relationship to those other elements in a Fabric network will be covered in our Hyperledger Fabric System Admin course.



A blockchain network is comprised of peer nodes, each of which can hold copies of ledgers and copies of smart contracts. In this example, the network N consists of peers P1, P2 and P3, each of which maintain their own instance of the distributed ledger L1. P1, P2 and P3 use the same chaincode, S1, to access their copy of that distributed ledger.

Peers can be created, started, stopped, reconfigured, and even deleted. They expose a set of APIs that enable administrators and applications to interact with the services that they provide.

2. Ordering service

The ordering service, or orderer, builds a shared communication channel between clients and peers. It atomically broadcasts messages containing transactions to peers to be committed. All peers receive the same block of transactions in the same order. The ordering service orders transactions on a first come, first serve basis for all channels on the network. After the transaction is ordered, the records of the committed are grouped and assigned as part of the block for that communication channel.

Note: The orderer and transaction flow are converged in depth in our Hyperledger Fabric for System Admin course.

3. Fabric CA

It's because Certificate Authority or CAs are so important that Fabric provides a built-in CA component to allow you to create CAs in the blockchain networks you form. This component — known as **Fabric CA** is a private root CA provider capable of managing digital identities of Fabric participants that have the form of X.509 certificates. Because Fabric CA is a custom CA targeting the Root CA needs of Fabric, it is inherently not capable of providing SSL certificates for general/automatic use in browsers. However, because some CA must be used to manage identity (even in a test environment), Fabric CA can be used to provide and manage certificates.

Note: The concepts such as digital identity, public key infrastructure, digital certificate and authentication are converged in depth in our Hyperledger Fabric for System Admin course.

4. Fabric Ledger and Chaincode

The ledger is the sequenced, immutable, entire historical record of all state transactions. All the transactions that are performed will be added to the ledger. We can find the entire transaction history for each channel. A Fabric blockchain ledger has two types of data—**a world state** and **a blockchain transaction ledger**. The world state, which is stored in a database, **LevelDB**, is the default database. The state data is mutable. It has a version number that keeps incrementally updated when the state changes. On the other hand, the ledger data is immutable, and is stored as a file. It records transaction data block information, which contains a sequence of transactions. Each block's header includes a hash of the block's transactions, as discussed before in our intro to blockchain technology session.

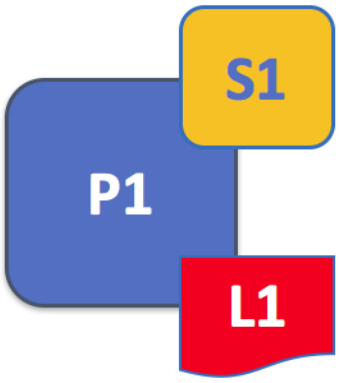
So far we understand that the ledger is the sequenced, tamper-resistant record of all state transitions in the Fabric while State transitions are a result of chaincode invocations ('transactions') submitted by participating parties. Each transaction results in a set of asset key-value pairs that are committed to the ledger as creates, updates, or deletes.

The ledger is comprised of a blockchain ('chain') to store the immutable, sequenced record in blocks, as well as a state database to maintain current Fabric state. There is one ledger per channel. Each peer maintains a copy of the ledger for each channel of which they are a member.

Some features of a Fabric ledger:

- Query and update ledger using key-based lookups, range queries, and composite key queries
- Read-only queries using a rich query language (if using CouchDB as state database)
- Read-only history queries — Query ledger history for a key, enabling data provenance scenarios
- Transactions consist of the versions of keys/values that were read in chaincode (read set) and keys/values that were written in chaincode (write set)
- Transactions contain signatures of every endorsing peer and are submitted to ordering service
- Transactions are ordered into blocks and are "delivered" from an ordering service to peers on a channel
- Peers validate transactions against endorsement policies and enforce the policies
- Prior to appending a block, a versioning check is performed to ensure that states for assets that were read have not changed since chaincode execution time
- There is immutability once a transaction is validated and committed
- A channel's ledger contains a configuration block defining policies, access control lists, and other pertinent information
- Channels contain Membership Service Provider instances allowing for crypto materials to be derived from different certificate authorities

Let's look at a peer in a little more detail. We can see that it's the peer that hosts both the ledger and chaincode. More accurately, **the peer actually hosts *instances* of the ledger, and *instances* of chaincode**. Note that this provides a deliberate redundancy in a Fabric network — it avoids single points of failure.

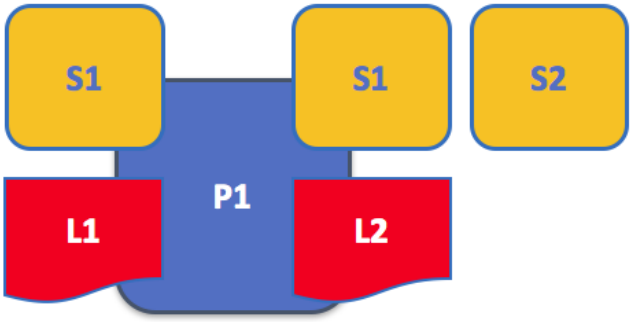


A peer hosts instances of ledgers and instances of chaincodes. In this example, P1 hosts an instance of ledger L1 and an instance of chaincode S1. There can be many ledgers and chaincodes hosted on an individual peer.

Because a peer is a *host* for ledgers and chaincodes, applications and administrators must interact with a peer if they want to access these resources. That's why peers are considered the most fundamental building blocks of a Fabric network. When a peer is first created, it has neither ledgers nor chaincodes. We'll see later how ledgers get created, and how chaincodes get installed, on peers.

Multiple Ledgers

A peer is able to host more than one ledger, which is helpful because it allows for a flexible system design. The simplest configuration is for a peer to manage a single ledger, but it's absolutely appropriate for a peer to host two or more ledgers when required.



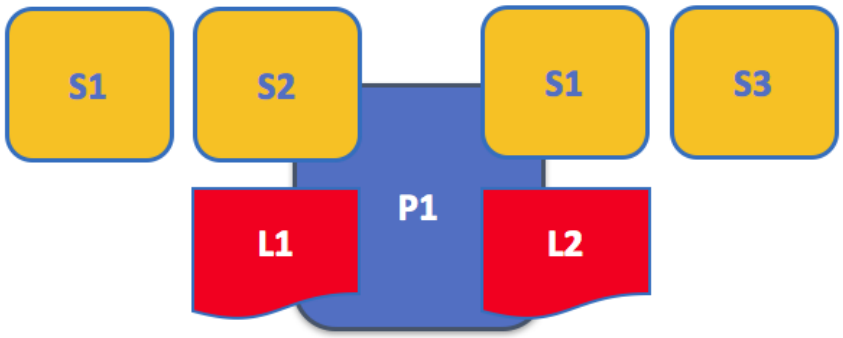
A peer hosting multiple ledgers. Peers host one or more ledgers, and each ledger has zero or more chaincodes that apply to them. In this example, we can see that the peer P1 hosts ledgers L1 and L2. Ledger L1 is accessed using chaincode S1. Ledger L2 on the other hand can be accessed using chaincodes S1 and S2.

Although it is perfectly possible for a peer to host a ledger instance without hosting any chaincodes which access that ledger, it's rare that peers are configured this way. The vast majority of peers will have at least one chaincode installed on it which can query or update the peer's ledger instances. It's worth

mentioning in passing that, whether or not users have installed chaincodes for use by external applications, peers also have special **system chaincodes** that are always present. These are not discussed in detail in this course.

Multiple Chaincodes

There isn't a fixed relationship between the number of ledgers a peer has and the number of chaincodes that can access that ledger. A peer might have many chaincodes and many ledgers available to it.



An example of a peer hosting multiple chaincodes. Each ledger can have many chaincodes which access it. In this example, we can see that peer P1 hosts ledgers L1 and L2, where L1 is accessed by chaincodes S1 and S2, and L2 is accessed by S1 and S3. We can see that S1 can access both L1 and L2.

We'll see in the next section why the concept of **channels** in Fabric is important when hosting multiple ledgers or multiple chaincodes on a peer.

5. Channel

A Hyperledger Fabric channel is a private "subnet" of communication between two or more specific network members, for the purpose of conducting private and confidential transactions. Here are the major characteristics of a Fabric channel.

- **A channel is defined by members (organizations), anchor peers per member, the shared ledger, chaincode application(s) and the ordering service node(s)**.
- Each transaction on the network is executed on a channel, where each party must be authenticated and authorized to transact on that channel.
- Each peer that joins a channel, has its own identity given by a membership services provider (MSP), which authenticates each peer to its channel peers and services.

To create a new channel, the client SDK calls configuration system chaincode and references properties such as anchor peers, and members (organizations). This request creates a genesis block for the channel ledger, which stores configuration information about the channel policies, members and anchor

peers. When adding a new member to an existing channel, either this genesis block, or if applicable, a more recent reconfiguration block, is shared with the new member.

The election of a leading peer for each member on a channel determines which peer communicates with the ordering service on behalf of the member. If no leader is identified, an algorithm can be used to identify the leader. The consensus service orders transactions and delivers them, in a block, to each leading peer, which then distributes the block to its member peers, and across the channel, using the gossip protocol.

Although any anchor peer can belong to multiple channels, and therefore maintain multiple ledgers, **no ledger data can pass from one channel to another**. This separation of ledgers, by channel, is defined and implemented by configuration chaincode, the identity membership service and the gossip data dissemination protocol. The dissemination of data, which includes information on transactions, ledger state and channel membership, is restricted to peers with verifiable membership on the channel. This isolation of peers and ledger data, by channel, allows network members that require private and confidential transactions to coexist with business competitors and other restricted members, on the same blockchain network.

Note: The channel and its detailed configurations are converged in depth in our Hyperledger Fabric for System Admin course.

6. Policies

At its most basic level, a policy is a set of rules that define the structure for how decisions are made and specific outcomes are reached. To that end, policies typically describe a **who** and a **what**, such as the access or rights that an individual has over an **asset**. We can see that policies are used throughout our daily lives to protect assets of value to us, from car rentals, health, our homes, and many more.

For example, an insurance policy defines the conditions, terms, limits, and expiration under which an insurance payout will be made. The policy is agreed to by the policy holder and the insurance company, and defines the rights and responsibilities of each party.

Whereas an insurance policy is put in place for risk management, in Hyperledger Fabric, policies are the mechanism for infrastructure management. **Fabric policies represent how members come to agreement on accepting or rejecting changes to the network, a channel, or a smart contract**. Policies are agreed to by the consortium members when a network is originally configured, but they can also be modified as the network evolves. For example, they describe the criteria for adding or removing members from a channel, change how blocks are formed, or specify the number of organizations required to endorse a smart contract. All of these actions are described by a policy which defines who can perform the action. Simply put, everything you want to do on a Fabric network is controlled by a policy.

Note: The Fabric policies are converged in depth in our Hyperledger Fabric for System Admin course.

7. Membership Service Provider (MSP)

Because Fabric is a permissioned network, blockchain participants need a way to prove their identity to the rest of the network in order to transact on the network. As we covered in our Hyperledger Fabric for System Admin course, a Public Key Infrastructure (PKI) can provide verifiable identities through a chain of trust. How is that chain of trust used by the blockchain network?

Certificate Authorities issue identities by generating a public and private key which forms a key-pair that can be used to prove identity. Because a private key can never be shared publicly, a mechanism is required to enable that proof which is where the MSP comes in. For example, a peer uses its private key to digitally sign, or endorse, a transaction. The MSP on the ordering service contains the peer's public key which is then used to verify that the signature attached to the transaction is valid. The private key is used to produce a signature on a transaction that only the corresponding public key, that is part of an MSP, can match. Thus, **the MSP is the mechanism that allows that identity to be trusted and recognized by the rest of the network without ever revealing the member's private key.**

Despite its name, the Membership Service Provider does not actually provide anything. Rather, the implementation of the MSP requirement is a set of folders that are added to the configuration of the network and is used to define an organization both inwardly (organizations decide who its admins are) and outwardly (by allowing other organizations to validate that entities have the authority to do what they are attempting to do). **Whereas Certificate Authorities generate the certificates that represent identities, the MSP contains a list of permissioned identities.**

The MSP identifies which Root CAs and Intermediate CAs are accepted to define the members of a trust domain by listing the identities of their members, or by identifying which CAs are authorized to issue valid identities for their members.

But the power of an MSP goes beyond simply listing who is a network participant or member of a channel. It is the MSP that turns an identity into a **role** by identifying specific privileges an actor has on a node or channel. **Note that when a user is registered with a Fabric CA, a role of admin, peer, client, orderer, or member must be associated with the user.** For example, identities registered with the "peer" role should, naturally, be given to a peer. Similarly, identities registered with the "admin" role should be given to organization admins.

MSPs occur in two domains in a blockchain network:

- Locally on an actor's node (**local MSP**)
- In channel configuration (**channel MSP**)

The key difference between local and channel MSPs is not how they function – both turn identities into roles – but their **scope**. Each MSP lists roles and permissions at a particular level of administration.