**Hyperledger Fabric Release v1.2**

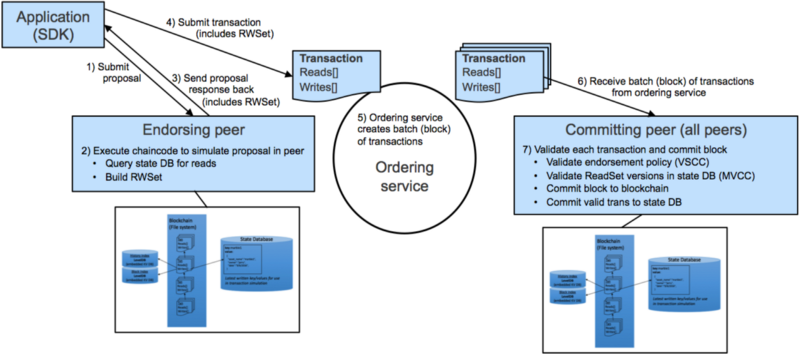
**Components:**

* **State** (latest) of the blockchain is a versioned KV store, which is maintained only by the peers.
* **Ledger** provides a verifiable history of state changes including valid and invalid transactions. Ledger is maintained at both Peers (PeerLedger) and Orderer (OrdererLedger). PeerLedgers can differentiate between valid and invalid transactions. OrdererLedgers are maintained for fault tolerance.
* **Nodes** are the communication entities of the blockchain. A “node” is only a **logical function** in the sense that multiple nodes of different types can run on the same physical server. What counts is how nodes are grouped in “trust domains” and associated to logical entities that control them.
* [**Peer**](https://jira.hyperledger.org/secure/attachment/10362/FabricOverview_draft1.pdf): All network peers, including endorsement peers, simulate transactions and commit validated transactions to their shared ledgers.
* [**Endorsement peer**](https://jira.hyperledger.org/secure/attachment/10362/FabricOverview_draft1.pdf): In addition to performing the role of peer, *endorsement peers also simulate all proposed transactions to check for code determinism and code stability*.

[The endorsing peers verify](https://medium.com/coinmonks/hyperledger-fabric-in-practice-part-1-main-components-and-running-them-locally-aa4b805465fa)

* 1. that the transaction proposal is well formed,
  2. it has not been submitted already in the past,
  3. the signature is valid and
  4. that the submitter is properly authorized to perform the proposed operation on that channel (e.g. writing into the channel).

The endorsing peers take the transaction proposal inputs as arguments to the invoked chaincode’s function. The chaincode is executed against the current ledger state to produce transaction results including a response value, read and write sets. No updates are made to the ledger at this point. The set of these values, along with the endorsing peer’s signature and a YES/NO endorsement statement is passed back as a “proposal response” to the client application.



* [**Ordering nodes**](https://hyperledger-fabric.readthedocs.io/en/release-1.2/arch-deep-dive.html) run the consensus protocol for the blockchain network and validate the order of simulated transactions. Ordering nodes run consensus only, and do not process transactions or maintain a ledger. By contrast, peers process transactions only, and do not run consensus.

Ordering service provides a shared communication channel to clients and peers, offering a broadcast service for messages containing transactions. Clients connect to the channel and may broadcast messages on the channel which are then delivered to all peers.

Ordering service may support multiple channels similar to the topics of a publish/subscribe (pub/sub) messaging system. Clients can connect to a given channel and can then send messages and obtain the messages that arrive.

* [Channels in Hyperledger Fabric](https://www.ibm.com/developerworks/cloud/library/cl-blockchain-private-confidential-transactions-hyperledger-fabric-zero-knowledge-proof/index.html):

A channel is like a virtual blockchain network that sits on top of a physical blockchain network with its own access rules. Channels employ their own transaction ordering mechanism and thus provide scalability, ultimately allowing for effective ordering and partition of huge amounts of data.

From a privacy perspective, channels are useful in cases where a subgroup of the blockchain network’s participants have a lot of transactions in common (enough to justify the creation of a new broadcast order channel).

Of course, the rate of transactions should be high enough and the number of business partners (and resulting channels) should be low enough for this use case to also preserve the scalability advantages of the network’s channel architecture.

The [election of a](https://hyperledger-fabric.readthedocs.io/en/release-1.2/channels.html)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.

* [**Identities**](https://hyperledger-fabric.readthedocs.io/en/release-1.2/identity/identity.html):

Analogy: In a payment network, a card can be used for payment at a merchant’s store. However, only certain card types will be accepted (Visa, Amex etc), so even if your card is valid, it may not be admissible in all stores.

Mapping this to the Fabric network, consider CA (part of PKI provider) as Card Issuer, card as Digital Identity (Organization, Certs etc) and Service limiter as MSP.

* **Fabric CA**

It’s because 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. It is also possible — and fully appropriate — to use a public/commercial root or intermediate CA to provide identification. If you’re interested, you can read a lot more about Fabric CA [in the CA documentation section](http://hyperledger-fabric-ca.readthedocs.io/).

* [Membership Service Providers](https://hyperledger-fabric.readthedocs.io/en/release-1.2/membership/membership.html) **(MSPs) turn verifiable identities into the members of a blockchain network. MSPs identify which Root CAs and Intermediate CAs are trusted to define the members of a trust domain, e.g., an organization**, either by listing the identities of their members, or by identifying which CAs are authorized to issue valid identities for their members, or — as will usually be the case — through a combination of both.

**Transaction Privacy:**

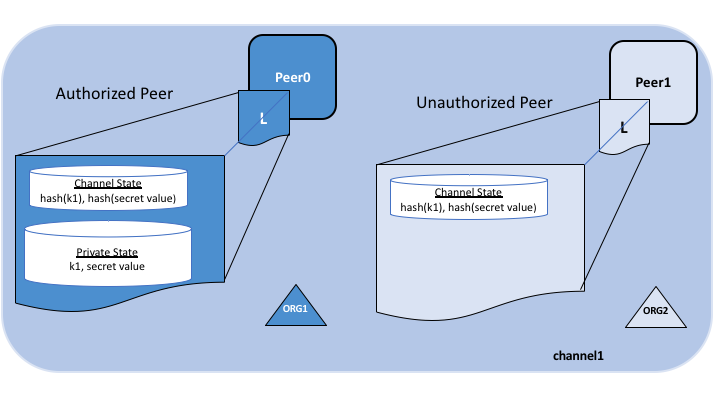
* [**Private data handling on Hyperledger Fabric**](https://hyperledger-fabric.readthedocs.io/en/release-1.2/private-data/private-data.html#what-is-private-data) **(Private transactions)**:

In cases where a group of organizations on a channel need to keep data private from other organizations on that channel, they have the option to create a new channel comprising just the organizations who need access to the data. ***However, creating separate channels in each of these cases creates additional administrative overhead (maintaining chaincode versions, policies, MSPs, etc)***, and doesn’t allow for use cases in which you want all channel participants to see a transaction while keeping a portion of the data private.

That’s why, starting in v1.2, Fabric offers the ability to create **private data collections**, which allow a defined subset of organizations on a channel the *ability to endorse, commit, or query private data without having to create a separate channel.*

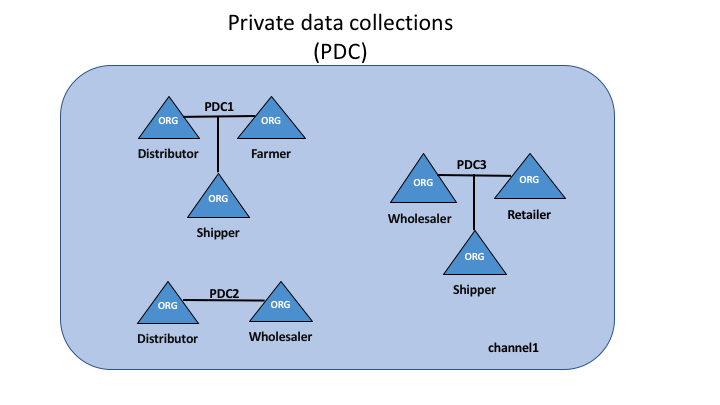
PDCs have two components:

1. The actual private data
2. Hash of that data



Only Authorized peers are allowed to access the private data content; the rest of the peers only have access to hash of the private data (for transaction audit purposes).

Because these databases are kept separate from the database that holds the channel ledger, private data is sometimes referred to as “***SideDB***”.



Transactions dealing with private data are called [Private Transactions](https://www.ibm.com/developerworks/cloud/library/cl-blockchain-private-confidential-transactions-hyperledger-fabric-zero-knowledge-proof/index.html). They do not conceal the parties who are allowed access to the private data. This information is available on the ledger for private data dissemination to take place properly.

Finally, private transactions would need to be accompanied with anonymous client authentication mechanisms to avoid leaking the connection between the identity of the transaction creator and the ledger stored (hashed) data.

### When to use a collection within a channel vs. a separate channel

* 1. Use **channels** when entire transactions (and ledgers) must be kept confidential within a set of organizations that are members of the channel.
  2. Use **collections** when transactions (and ledgers) must be shared among a set of organizations, but when only a subset of those organizations should have access to some (or all) of the data within a transaction. Additionally, since private data is disseminated peer-to-peer rather than via blocks, use private data collections when transaction data must be kept confidential from ordering service nodes.
* **[Access Control of Resources under Composer](https://hyperledger.github.io/composer/v0.16/reference/acl_language)**:

Access control for a business network is defined by an ordered set of ACL rules. ACL rules are defined in a file called *permissions.acl* in the root of the business network. If this file is missing from the business network then all access is permitted.

For example, the rule below states that any instance of the org.example.SampleParticipant type can perform ALL operations on all instances of org.example.SampleAsset IF the participant is the owner of the asset AND the participant submitted a transaction of the org.example.SampleTransaction type to perform the operation:

rule SampleConditionalRuleWithTransaction {

description: "Description of the ACL rule"

participant(m): "org.example.SampleParticipant"

operation: READ, CREATE, UPDATE

resource(v): "org.example.SampleAsset"

transaction(tx): "org.example.SampleTransaction"

condition: (v.owner.getIdentifier() == m.getIdentifier())

action: ALLOW

}

**Thus only specific Participants can change states of specific resources (assets, namespaces etc) using particular transactions**.

* dsf