# Frequently Asked Questions

## Endorsement

**Endorsement architecture**:

|  |  |
| --- | --- |
| **Question:** | How many peers in the network need to endorse a transaction? |
| **Answer:** | The number of peers required to endorse a transaction is driven by the endorsement policy that is specified in the chaincode definition. |
| **Question:** | Does an application client need to connect to all peers? |
| **Answer:** | Clients only need to connect to as many peers as are required by the endorsement policy for the chaincode. |

## Security & Access Control

|  |  |
| --- | --- |
| **Question:** | How do I ensure data privacy? |
| **Answer:** | There are various aspects to data privacy. First, you can segregate your network into channels, where each channel represents a subset of participants that are authorized to see the data for the chaincodes that are deployed to that channel.  Second, you can use [private-data](https://hyperledger-fabric.readthedocs.io/en/latest/private-data/private-data.html) to keep ledger data private from other organizations on the channel. A private data collection allows a defined subset of organizations on a channel the ability to endorse, commit, or query private data without having to create a separate channel. Other participants on the channel receive only a hash of the data. For more information refer to the [Using Private Data in Fabric](https://hyperledger-fabric.readthedocs.io/en/latest/private_data_tutorial.html) tutorial. Note that the key concepts topic also explains [when to use private data instead of a channel](https://hyperledger-fabric.readthedocs.io/en/latest/private-data/private-data.html#when-to-use-a-collection-within-a-channel-vs-a-separate-channel).  Third, as an alternative to Fabric hashing the data using private data, the client application can hash or encrypt the data before calling chaincode. If you hash the data then you will need to provide a means to share the source data. If you encrypt the data then you will need to provide a means to share the decryption keys.  Fourth, you can restrict data access to certain roles in your organization, by building access control into the chaincode logic.  Fifth, ledger data at rest can be encrypted via file system encryption on the peer, and data in-transit is encrypted via TLS. |
| **Question:** | Do the orderers see the transaction data? |
| **Answer:** | No, the orderers only order transactions, they do not open the transactions. If you do not want the data to go through the orderers at all, then utilize the private data feature of Fabric. Alternatively, you can hash or encrypt the data in the client application before calling chaincode. If you encrypt the data then you will need to provide a means to share the decryption keys. |

## Application-side Programming Model

|  |  |
| --- | --- |
| **Question:** | How do application clients know the outcome of a transaction? |
| **Answer:** | The transaction simulation results are returned to the client by the endorser in the proposal response. If there are multiple endorsers, the client can check that the responses are all the same, and submit the results and endorsements for ordering and commitment. Ultimately the committing peers will validate or invalidate the transaction, and the client becomes aware of the outcome via an event, that the SDK makes available to the application client. |
| **Question:** | How do I query the ledger data? |
| **Answer:** | Within chaincode you can query based on keys. Keys can be queried by range, and composite keys can be modeled to enable equivalence queries against multiple parameters. For example a composite key of (owner,asset\_id) can be used to query all assets owned by a certain entity. These key-based queries can be used for read-only queries against the ledger, as well as in transactions that update the ledger.  If you model asset data as JSON in chaincode and use CouchDB as the state database, you can also perform complex rich queries against the chaincode data values, using the CouchDB JSON query language within chaincode. The application client can perform read-only queries, but these responses are not typically submitted as part of transactions to the ordering service. |
| **Question:** | How do I query the historical data to understand data provenance? |
| **Answer:** | The chaincode API GetHistoryForKey() will return history of values for a key. |
| **Question:** | How to guarantee the query result is correct, especially when the peer being queried may be recovering and catching up on block processing? |
| **Answer:** | The client can query multiple peers, compare their block heights, compare their query results, and favor the peers at the higher block heights. |

## Chaincode (Smart Contracts and Digital Assets)

|  |  |
| --- | --- |
| **Question:** | Does Hyperledger Fabric support smart contract logic? |
| **Answer:** | Yes. We call this feature [Chaincode](https://hyperledger-fabric.readthedocs.io/en/latest/glossary.html#chaincode). It is our interpretation of the smart contract method/algorithm, with additional features.  A chaincode is programmatic code deployed on the network, where it is executed and validated by chain validators together during the consensus process. Developers can use chaincodes to develop business contracts, asset definitions, and collectively-managed decentralized applications. |
| **Question:** | How do I create a business contract? |
| **Answer:** | There are generally two ways to develop business contracts: the first way is to code individual contracts into standalone instances of chaincode; the second way, and probably the more efficient way, is to use chaincode to create decentralized applications that manage the life cycle of one or multiple types of business contracts, and let end users instantiate instances of contracts within these applications. |
| **Question:** | How do I create assets? |
| **Answer:** | Users can use chaincode (for business rules) and membership service (for digital tokens) to design assets, as well as the logic that manages them.  There are two popular approaches to defining assets in most blockchain solutions: the stateless UTXO model, where account balances are encoded into past transaction records; and the account model, where account balances are kept in state storage space on the ledger.  Each approach carries its own benefits and drawbacks. This blockchain technology does not advocate either one over the other. Instead, one of our first requirements was to ensure that both approaches can be easily implemented. |
| **Question:** | Which languages are supported for writing chaincode? |
| **Answer:** | Chaincode can be written in any programming language and executed in containers. Currently, Golang, node.js and java chaincode are supported. |
| **Question:** | Does the Hyperledger Fabric have native currency? |
| **Answer:** | No. However, if you really need a native currency for your chain network, you can develop your own native currency with chaincode. One common attribute of native currency is that some amount will get transacted (the chaincode defining that currency will get called) every time a transaction is processed on its chain. |

## Differences in Most Recent Releases

|  |  |
| --- | --- |
| **Question:** | Where can I find what are the highlighted differences between releases? |
| **Answer:** | The differences between any subsequent releases are provided together with the [Releases](https://hyperledger-fabric.readthedocs.io/en/latest/releases.html). |
| **Question:** | Where to get help for the technical questions not answered above? |
| **Answer:** | Please use [StackOverflow](https://stackoverflow.com/questions/tagged/hyperledger). |

## Ordering Service

|  |  |
| --- | --- |
| **Question:** | **I have an ordering service up and running and want to switch consensus algorithms. How do I do that?** |
| **Answer:** | This is explicitly not supported. |

|  |  |
| --- | --- |
| **Question:** | **What is the orderer system channel?** |
| **Answer:** | The orderer system channel (sometimes called ordering system channel) is the channel the orderer is initially bootstrapped with. It is used to orchestrate channel creation. The orderer system channel defines consortia and the initial configuration for new channels. At channel creation time, the organization definition in the consortium, the /Channel group’s values and policies, as well as the /Channel/Orderer group’s values and policies, are all combined to form the new initial channel definition. |

|  |  |
| --- | --- |
| **Question:** | **If I update my application channel, should I update my orderer system channel?** |
| **Answer:** | Once an application channel is created, it is managed independently of any other channel (including the orderer system channel). Depending on the modification, the change may or may not be desirable to port to other channels. In general, MSP changes should be synchronized across all channels, while policy changes are more likely to be specific to a particular channel. |

|  |  |
| --- | --- |
| **Question:** | **Can I have an organization act both in an ordering and application role?** |
| **Answer:** | Although this is possible, it is a highly discouraged configuration. By default the /Channel/Orderer/BlockValidation policy allows any valid certificate of the ordering organizations to sign blocks. If an organization is acting both in an ordering and application role, then this policy should be updated to restrict block signers to the subset of certificates authorized for ordering. |

|  |  |
| --- | --- |
| **Question:** | **I want to write a consensus implementation for Fabric. Where do I begin?** |
| **Answer:** | A consensus plugin needs to implement the Consenter and Chain interfaces defined in the [consensus package](https://github.com/hyperledger/fabric/blob/release-2.0/orderer/consensus/consensus.go). There is a plugin built against [raft](https://github.com/hyperledger/fabric/tree/release-2.0/orderer/consensus/etcdraft) . You can study it to learn more for your own implementation. The ordering service code can be found under the [orderer package](https://github.com/hyperledger/fabric/tree/release-2.0/orderer). |

|  |  |
| --- | --- |
| **Question:** | **I want to change my ordering service configurations, e.g. batch timeout, after I start the network, what should I do?** |
| **Answer:** | This falls under reconfiguring the network. Please consult the topic on [configtxlator](https://hyperledger-fabric.readthedocs.io/en/latest/commands/configtxlator.html). |

### BFT

|  |  |
| --- | --- |
| **Question:** | **When is a BFT version of the ordering service going to be available?** |
| **Answer:** | No date has been set. We are working towards a release during the 1.x cycle, i.e. it will come with a minor version upgrade in Fabric. Track [FAB-33](https://jira.hyperledger.org/browse/FAB-33) for updates. |