Notes of Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains

Fabric is an open-source system for deploying and operating permissioned blockchains and one of the Hyperledger projects hosted by the Linux foundation. It's securely tracks its execution history in an append-only replicated ledger data structure and has no native cryptocurrency built in.

It is modular and supports modular consensus protocols. This allows the system, to be tailored to particular use cases. Fabric is first in two things:

1. A modular / extensive blockchain system for running distributed applications
2. For supporting distributed applications written in standard general-purpose programming languages (e.g., Go, Java) without being dependent on a domain-specific language or native cryptocurrency.

The permissioned model in Fabric is realized using a portable notion of membership which may be integrated with industry-standard identity management.
A permissioned blockchain lets a group of entities interact securely while they do not fully trust each other. By relying on the identities of the participant nodes, they can use a BFT consensus.

Hyperledger Fabric (Fabric) tries to solve the limitations of state-machine replication blockchain such as Ethereum. These limitations (or design choices) are:

- many distributed applications run concurrently
- applications may be deployed dynamically and by anyone
- the application code is untrusted and potentially malicious
- consensus is hard-coded within the platform
- the trust model of transaction validation is determined by the consensus protocol; it cannot be adapted to the smart contract's requirements
- Smart contracts must be written in a domain-specific language.
- The sequential execution of all transactions by all peers limits performance; complex measures are needed to prevent DDoS attacks.
- Transactions must be deterministic which can be difficult to ensure programmatically
- Every smart contract runs on all peers. It could break confidentiality if a smart contract contains secrets.
- The traditional order-execute architecture: the network first orders the transactions, then executes them in the same order on all peers sequentially.

Fabric introduces a new blockchain architecture aiming at resiliency, flexibility, scalability, and confidentiality. It is the first distributed operating system for permissioned blockchains.
Fabric's execute-order-validate paradigm for distributed execution of untrusted code separates the transaction flow into three steps:

1. Executing the transaction and checking its correctness – then endorsing it.
2. Ordering through a consensus protocol, irrespective of transaction semantics.
3. Transaction validation per application specific trust assumptions which prevents race conditions due to concurrency.

Fabric typically executes transactions before reaching final agreement on their order. It combines both passive and active approached to replication:

Passive: every transaction is endorsed (or executed) by a subset of peers -> it allows for parallel execution and addresses potential non-determinism.

Active: the transaction's effect on the ledger state are only written after reaching consensus on a total order among them, in the deterministic validation step executed by each peer individually.
This allows Fabric to respect application-specific trust assumptions according to the transaction endorsement.

A modular component takes care of the ordering of the state.
It is stateless and decoupled from the peers that execute transactions.
Because of the modular consensus, it can be tailored to the trust assumption of a particular deployment.
The separation of the two roles adds flexibility and allows one to rely on industry standard toolkits for crash fault-tolerance or BFT ordering.

The combination of the hybrid replication design with both an active and passive approach, along with the execute-order-validate paradigm represents the main innovation in Fabric's architecture.
The architecture can scale and support flexible trust assumptions for permissioned blockchains.

The implementation of the architecture is done through these modular building blocks:

- An ordering service: atomically broadcasts state updates to peers and establishes consensus on the transaction order.
- A membership service provider: responsible for associating peers with cryptographic identities; maintains the permissioned nature of Fabric.
- An optional peer-to-peer gossip service: disseminates the blocks output by ordering service to all peers.
- Smart contracts: run within a container environment for isolation. Supports standard programming languages.
- Ledger: each peer locally maintains the ledger in the form of the append-only blockchain and as a snapshot of the most recent state in a key-value store.

The traditional order-execute architecture has some limitations:

- sequential execution: limits the throughput. Difficult to deterministically inferring all dependencies across smart contracts.
- Non-deterministic code: if it's not deterministic, forks are created (in a SMR blockchain).
  This breaks the premise that all peers hold the same state. Addressed by domain-specific programming languages. Smart contracts can be made by anyone: potentially untrusted or inexperienced programmers.
- Confidentiality of execution: some use-cases require that access to smart contract logic, transaction data, or ledger state is restricted. This is not possible if all peers can run all smart contracts. It should be enough to ensure the state is shared with all peers, and not the storing and execution of smart contracts.

Because external conditions such as load, network parameters, faults and attacks may vary over time, a BFT consensus should be reconfigurable and ideally be able to adapt to the changing environment.

The execute-order-validate architecture of Fabric addresses these limitations.

So how does it work?
A distributed application in Fabric consists of two parts:

1. A smart contract (chaincode) – implements application logic and runs during the execution phase. May be written by an untrusted developer. System chaincodes are special smart contracts that exists for managing the blockchain system and maintaining parameters.
2. An endorsement policy – it is evaluated in the validation phase. A typical endorsement policy lets the chaincode specify the endorsers for a transaction in the form of a set of peers that are necessary for endorsement. These endorsement policies cannot be chosen or modified by untrusted developers. It acts as a static library for transaction validation in Fabric. Only designated administrator may have a permission to modify endorsement policies.

How are transactions sent and validated in Fabric? A high level view:

1. A client sends a transaction to the selected peers by the endorsement policy. Specific peers then execute each transaction and their outputs are recorded (endorsed).
2. After execution, the transactions enters the ordering phase which can use a tailored consensus protocol to produce a totally ordered sequence of endorsed transactions grouped in blocks. The blocks are broadcasted to all peers – optionally with the use of gossip.
   Fabric orders transaction outputs combined with state dependencies as compute during the execution phase.
3. All peers validate the blocks in the same order and validation is deterministic -> this is a novel hybrid replication paradigm in the Byzantine model, which combines passive replication (the pre-consensus computation of state updates) and active replication (the post-consensus validation execution results and state changes).

The Fabric network is permissioned, and therefore all the nodes are known.
It is the modular membership service provides (MSP) provides and manages these identities.
There are three roles that a node can choose on of: client, peer, or Ordering Service Nodes (OSN).
The OSN establishes the total order of all transactions in Fabric.
The Fabric network also supports multiple blockchains connected to the same ordering service; they are called channels and can be used to partition the state of the network.