**Hyperledger course content list also**

## Introduction

A blockchain is an immutable transaction ledger, maintained within a distributed network of peer nodes.

These nodes each maintain a copy of the ledger by applying transactions that have been validated by a consensus protocol, grouped into blocks that include a hash that bind each block to the preceding block.

For enterprise use, we need to consider the following requirements:

1. Participants must be identified/identifiable
2. Networks need to be *permissioned*
3. High transaction throughput performance
4. Low latency of transaction confirmation
5. Privacy and confidentiality of transactions and data pertaining to business transactions

## Hyperledger Fabric

* Fabric has a highly **modular** and **configurable** architecture.
* No domain specific language
* Leverage consensus protocols that **do not require a native cryptocurrency**
* **Pluggable consensus protocols**

when deployed within a single enterprise, or operated by a trusted authority, fully byzantine fault tolerant consensus might be considered unnecessary and an excessive drag on performance and throughput.

In situations such as that, a [crash fault-tolerant](https://en.wikipedia.org/wiki/Fault_tolerance) (CFT) consensus protocol might be more than adequate whereas,

In a multi-party, decentralized use case, a more traditional [byzantine fault tolerant](https://en.wikipedia.org/wiki/Byzantine_fault_tolerance) (BFT) consensus protocol might be required.

## Modularity

* pluggable consensus,
* pluggable identity management protocols such as LDAP or OpenID Connect,
* key management protocols or cryptographic libraries
* A pluggable *ordering service* establishes consensus on the order of transactions and then broadcasts blocks to peers.
* A pluggable *membership service provider* is responsible for associating entities in the network with cryptographic identities.
* The ledger can be configured to support a variety of DBMSs.
* A pluggable *endorsement and validation policy* enforcement that can be independently configured per application.

## Smart Contracts

Most existing smart-contract capable blockchain platforms follow an **order-execute** architecture

* Many smart contracts run concurrently in the network,
* they may be deployed dynamically (in many cases by anyone), and
* application code should be treated as untrusted, potentially even malicious.

Since all transactions are executed sequentially by all nodes, performance and scale is limited. The fact that the smart contract code executes on every node in the system demands that complex measures be taken to protect the overall system from potentially malicious contracts in order to ensure resiliency of the overall system.

## A New Approach

Fabric introduces a new architecture for transactions that we call **execute-order-validate**.

* *execute* a transaction and check its correctness, thereby endorsing it,
* *order* transactions via a (pluggable) consensus protocol, and
* *validate* transactions against an application-specific endorsement policy before committing them to the ledger

In Fabric, an application-specific endorsement policy specifies which peer nodes, or how many of them, need to vouch for the correct execution of a given smart contract.

Thus, each transaction need only be executed (endorsed) by the subset of the peer nodes necessary to satisfy the transaction’s endorsement policy.

*This allows for parallel execution increasing overall performance and scale of the system.*

## Privacy and Confidentiality

In a public, permissionless blockchain network that leverages PoW for its consensus model, transactions are executed on every node.

This means that neither can there be confidentiality of the contracts themselves, nor of the transaction data that they process.

*This lack of confidentiality can be problematic for many business/enterprise use cases.*

For example, in a network of supply-chain partners, some consumers might be given preferred rates as a means of either solidifying a relationship, or promoting additional sales.

Encrypting data is one approach to providing confidentiality; however, in a permissionless network leveraging PoW for its consensus, the encrypted data is sitting on every node. Given enough time and computational resource, the encryption could be broken.

Zero knowledge proofs (ZKP) are another area of research being explored to address this problem, the trade-off here being that, presently, computing a ZKP requires considerable time and computational resources. Hence, the trade-off in this case is performance for confidentiality.

Hyperledger Fabric, being a permissioned platform, enables confidentiality through its channel architecture.

Only those nodes that participate in a channel have access to the smart contract (chaincode) and data transacted, preserving the privacy and confidentiality of both.

To improve upon its privacy and confidentiality capabilities, Fabric has added support for

* [private data](https://hyperledger-fabric.readthedocs.io/en/release-1.4/private-data/private-data.html) and
* is working on zero knowledge proofs (ZKP) available in the future.

## Pluggable Consensus

This modular architecture allows the platform to rely on well-established toolkits for CFT (crash fault-tolerant) or BFT (byzantine fault-tolerant) ordering.

Kafka is CFT or BFT?

In the currently available releases, Fabric offers a CFT ordering service implemented with [Kafka](https://kafka.apache.org/) and [Zookeeper](https://zookeeper.apache.org/).

In subsequent releases, Fabric will deliver a [Raft consensus ordering service](https://raft.github.io/) implemented with etcd/Raft and a fully decentralized BFT ordering service.

A Fabric network can have multiple ordering services supporting different applications or application requirements.

## Performance and Scalability

Performance of a blockchain platform can be affected by many variables such as

* transaction size,
* block size,
* network size,
* as well as limits of the hardware, etc.

The Hyperledger community is currently developing [a draft set of measures](https://docs.google.com/document/d/1DQ6PqoeIH0pCNJSEYiw7JVbExDvWh_ZRVhWkuioG4k0/edit#heading=h.t3gztry2ja8i) within the Performance and Scale working group, along with a corresponding implementation of a benchmarking framework called [Hyperledger Caliper](https://wiki.hyperledger.org/projects/caliper).

<https://hyperledger-fabric.readthedocs.io/en/release-1.4/whatsnew.html>

## Serviceability and operations improvements

The new RESTful operations service provides operators with three services to monitor and manage peer and orderer node operations:

* The logging /logspec endpoint allows operators to dynamically get and set logging levels for the peer and orderer nodes.
* The /healthz endpoint allows operators and container orchestration services to check peer and orderer node liveness and health.
* The /metrics endpoint allows operators to pull operational metrics from peer and orderer nodes.

## Improved programming model for developing applications

* New npm packages provide a layer of abstraction to improve developer productivity and ease of use.

## Private data enhancements

* **Reconciliation**, which allows peers for organizations that are added to private data collections to retrieve the private data for prior transactions to which they now are entitled.
* **Client access control** to automatically enforce access control within chaincode based on the client organization collection membership without having to write specific chaincode logic.

<https://hyperledger-fabric.readthedocs.io/en/release-1.4/key_concepts.html>