Summary of Public and Permissioned Blockchains and Hyperledger Fabric

Public vs. Permissioned Blockchains Public (permissionless) blockchains utilize Proof of Work (PoW) as their consensus mechanism. In contrast, permissioned blockchains rely on a group of entities with a common goal but without full mutual trust, such as businesses exchanging funds, goods, or information. These blockchains often use Byzantine Fault Tolerance (BFT) consensus for validation.

Smart Contracts Many smart contracts follow the state machine replication model and implement "active replication" using the order-execute architecture. Transactions are first ordered through atomic broadcast and then executed sequentially across all peers to maintain determinism. However, additional transactions may disrupt this order.

Permissioned blockchains have several limitations, including hard-coded consensus mechanisms, programming constraints that limit widespread adoption, sequential transaction execution that reduces performance, challenges in ensuring deterministic transactions, and confidentiality issues due to all peers executing smart contracts.

Introduction to Hyperledger Fabric Hyperledger Fabric is a permissioned blockchain framework that addresses the above limitations. It supports distributed applications written in standard programming languages and has applications across multiple domains. Fabric employs a hybrid replication design, combining passive and active replication in a Byzantine model while utilizing the order-execute-validate paradigm.

Fabric consists of modular components, including:

- **Ordering Service:** Broadcasts state updates to peers and establishes transaction order.
- **Membership Service Provider (MSP):** Manages peer identities using cryptographic credentials to maintain permissioned access.
- **Peer-to-Peer Gossip Protocol:** Disseminates ordered blocks to all peers.
- **Smart Contracts (Chaincode):** Run in isolated container environments, ensuring security and standard language support.
- **Ledger:** Each peer maintains an append-only blockchain and a key-value store snapshot of the latest state.

Challenges in Traditional Blockchains Traditional blockchains use the order-execute architecture, leading to three major issues fixed by hyper ledger fabric :

1. **Sequential Execution:** Reduces performance and is susceptible to DoS attacks. Ethereum mitigates this with transaction fees.
2. **Non-Deterministic Code:** Ensuring deterministic execution is challenging due to the use of generic programming languages, which may cause forks.
3. **Confidentiality Issues:** All smart contracts run on all peers, leading to privacy concerns.

Hyperledger Fabric Architecture Fabric's distributed applications consist of:

- **Chaincode (Smart Contracts):** The core component written by developers.

- **Endorsement Policy:** Defines validation rules and can be modified only by administrators.

Fabric operates through three main types of nodes:

1. **Clients:** Submit transactions.
2. **Peers:** Execute transactions and validate them.
3. **Ordering Service Nodes (OSN):** Establish the total order of transactions.

Transaction Lifecycle in Hyperledger Fabric

1. **Execution Phase:** Clients sign and send transaction proposals to endorsers, which simulate execution in isolated Docker environments. The endorsers generate signed responses, which the client collects until the endorsement policy is met.
2. **Ordering Phase:** The ordering service establishes a total order of transactions, ensuring agreement, hash chain integrity, and validity. Unlike traditional blockchains, consensus in Fabric is separate from execution and validation.
3. **Validation Phase:** Peers receive ordered blocks, evaluate endorsement policies in parallel, check for read/write conflicts, and update the ledger accordingly.

Fabric Components

- **Membership Service:** Maintains node identities and authentication credentials.
- **Ordering Service:** Manages multiple channels, provides atomic broadcast for transaction ordering, and handles channel reconfiguration. It uses an initial genesis block to define service properties and employs Apache Kafka for transaction batching.
- **Peer Gossip Protocol:** Addresses network bandwidth constraints by disseminating data efficiently using push and pull mechanisms. A designated leader peer initiates the gossip protocol to reduce network load.
- **Ledger:** Maintains a blockchain ledger and executes chaincode in an isolated environment, supporting multiple programming languages.
- **System Chaincode:** Enables network configuration updates through a configuration transaction mechanism, ensuring blockchain adaptability.

.