Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Technical Deep Dive into Hyperledger Fabric v1.0

Blockchain CoE Pune

28 Apr 2018

People matter, results count.

Hyperledger Project

What is Hyperledger Project?

- Linux Foundation project announced on 17 December 2015
- Hyperledger project aims to transform the way business transactions are conducted around the world
- An open source, collaborative development strategy supporting multiple players in multiple industries
- Several companies have submitted candidates for the base code;

8	300+	142
CODE BASES	CONTRIBUTORS	MEMBERS

Community + Code

Linux
Hyperledger Project

Projects Under Hyperledger Umbrella

Hyperledger Frameworks

Hyperledger Sawtooth

Hyperledger Sawtooth is a modular platform for building, deploying, and running distributed ledgers. Hyperledger Sawtooth includes a novel consensus algorithm, Proof of Elapsed Time (PoET), which targets large distributed validator populations with minimal resource consumption.

Hyperledger Iroha

Hyperledger Iroha is a business blockchain framework designed to be simple and easy to incorporate into infrastructural projects requiring distributed ledger technology.

Hyperledger Fabric

Intended as a foundation for developing applications or solutions with a modular architecture, Hyperledger Fabric allows components, such as consensus and membership services, to be plug-and-play.

Hyperledger Burrow

Hyperledger Burrow is a permissionable smart contract machine. The first of its kind when released in December, 2014, Burrow provides a modular blockchain client with a permissioned smart contract interpreter built in part to the specification of the Ethereum Virtual Machine (EVM).

Hyperledger Indy

Hyperledger Indy provides tools, libraries, and reusable components for providing digital identities rooted on blockchains or other distributed ledgers so that they are interoperable across administrative domains, applications, and any other silo.

Hyperledger Tools

Hyperledger Cello

Hyperledger Cello aims to bring the on-demand "as-a-service" deployment model to the blockchain ecosystem to reduce the effort required for creating, managing and terminating blockchains.

Hyperledger Composer

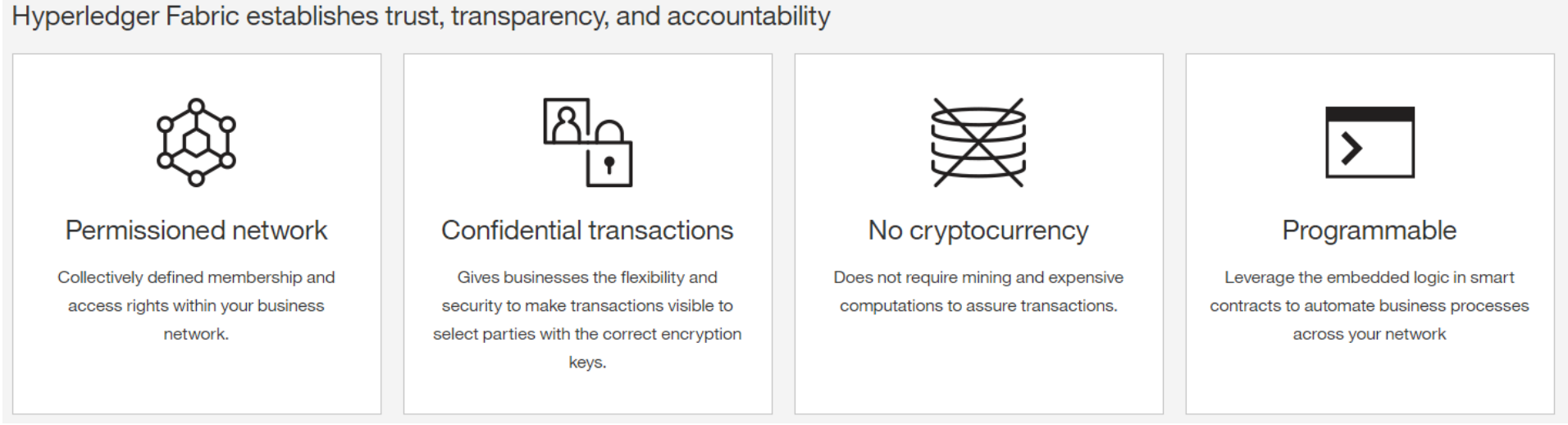
Hyperledger Composer is a collaboration tool for building blockchain business networks, accelerating the development of smart contracts and their deployment across a distributed ledger.

Hyperledger Explorer

Hyperledger Explorer can view, invoke, deploy or query blocks, transactions and associated data, network information, chain codes and transaction families, as well as any other relevant information stored in the ledger.

What is Hyperledger Fabric (HLF)?

- IBM's implementation of Blockchain technology
- It include nodes, APIs, membership services, and more
- Provides a permissioned network, with known identities
- No cryptocurrencies
- Provides a foundation for developing Blockchain solutions with a modular architecture, pluggable implementations, and container technology

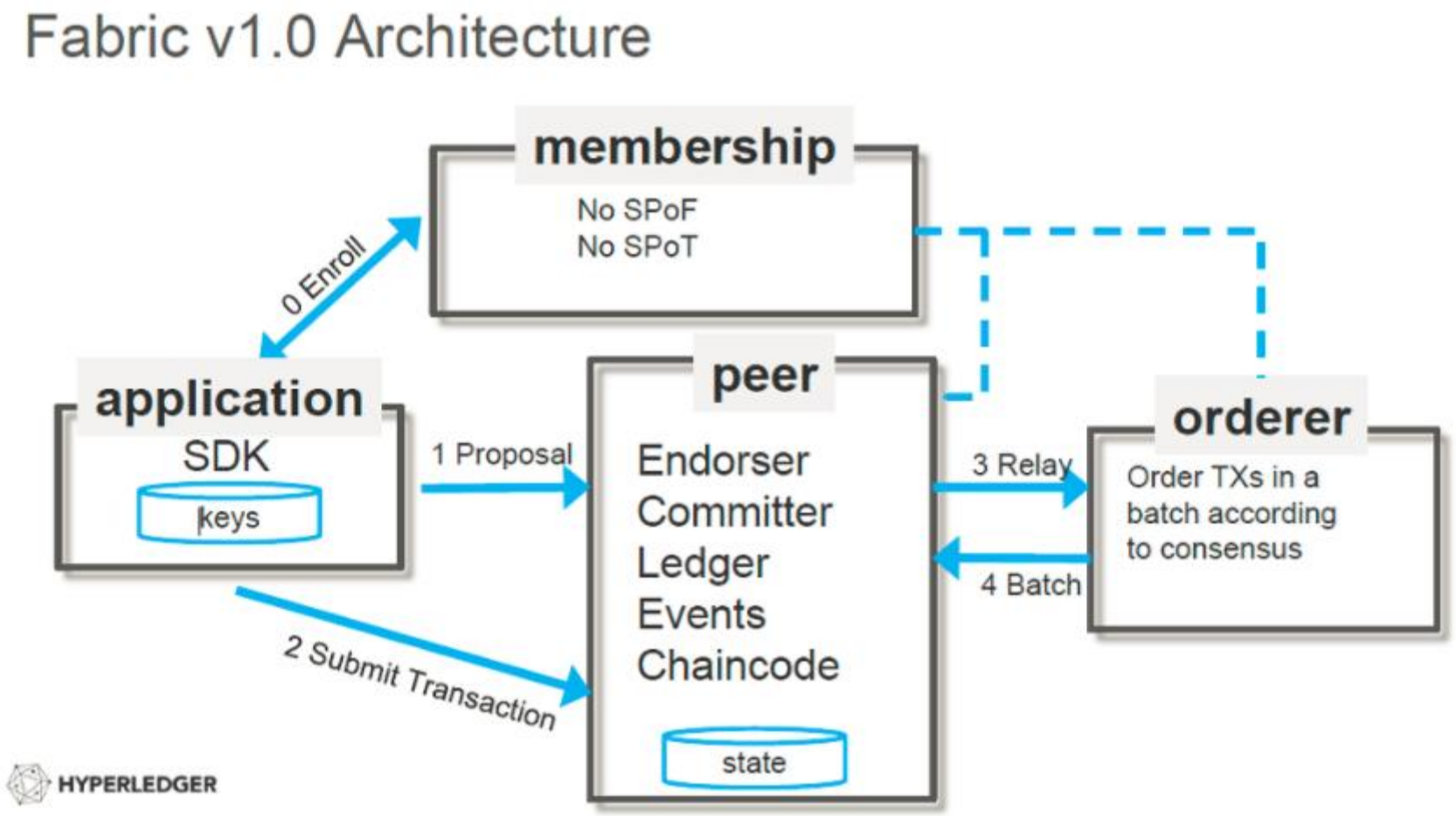


Hyperledger Fabric Vs Ehtereum Vs Ripple Vs Bitcoin

	Hyperledger Fabric	Ethereum	Ripple	Bitcoin
Description of Platform	General purpose Blockchain	General purpose Blockchain	Payments Blockchain	Payments Blockchain
Governance	Linux Foundation	Ethereum Developers	Ripple Labs	Bitcoin Developers
Currency	None	Ether	XRP	BTC
Mining Reward	N/A	Yes	No	Yes
State	Key-value database	Account data	None	Transaction data
Consensus Network	Pluggable : Solo/Kafka/SBFT	Mining	Ripple Protocol	Mining
Network	Private	Public or Private	Public	Public
Privacy	Open to Private	Open	Open	Open
Smart Contracts	Multiple programming languages like Java, GO.	'Solidity' programming language	None	Possible, but not obvious

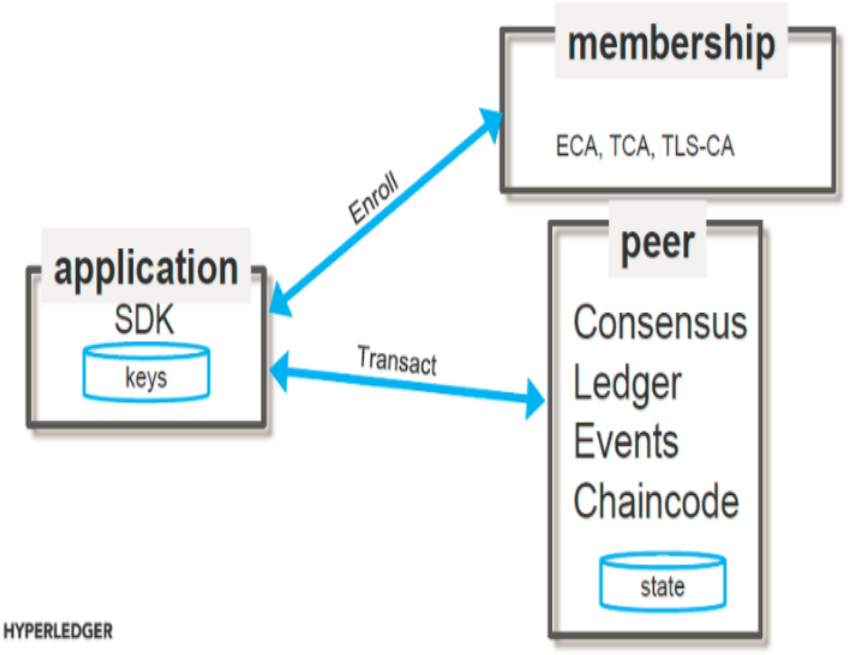
Architecture

Hyperledger Fabric v1.0 Architecture

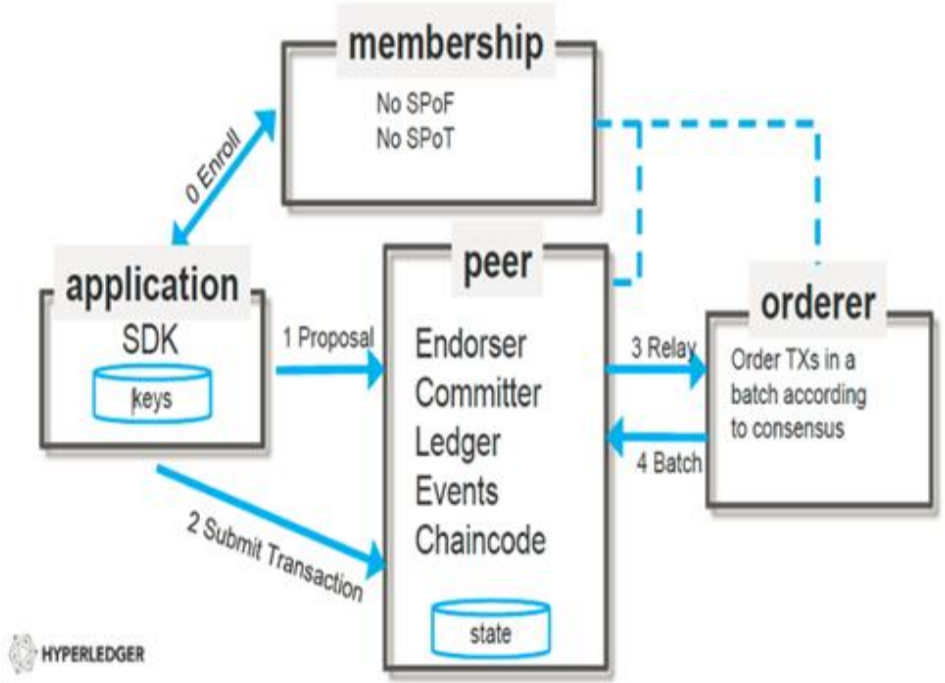


Hyperledger Fabric v0.6 Vs v1.0 Architecture

Fabric v0.6 Architecture



Fabric v1.0 Architecture

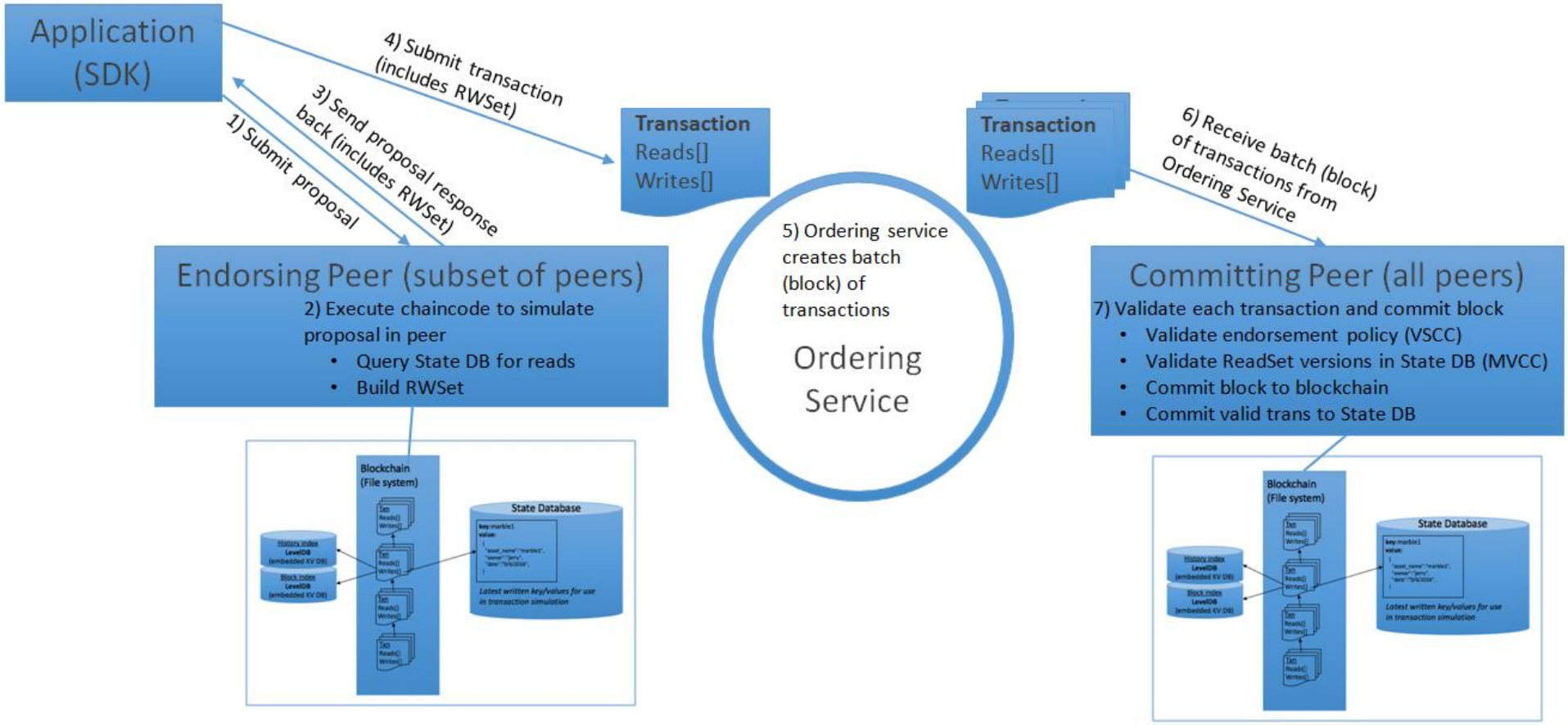


Transaction

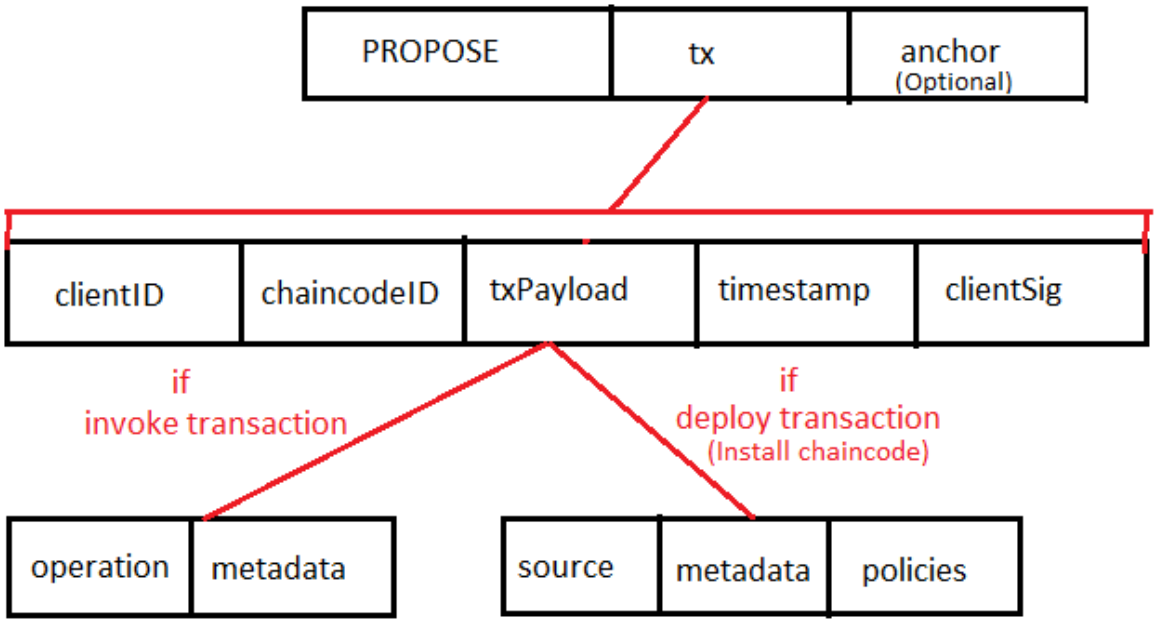
Transaction Lifecycle v1.0

1. Client application creates transaction proposal (chaincode function and arguments) and sends to endorsing peer(s).
2. Endorsing peer executes chaincode, generates ReadWriteSet based on keys that were read and written.
3. Endorsing peer(s) send back proposal response (including response payload and ReadWriteSet) to client application
4. Client application may or may not submit as a transaction to ordering service. Transaction includes ReadWriteSet from proposal response
5. If client application submitted as transaction, ordering service packages the transaction into a block of ordered transactions.
6. Blocks are delivered to all peers (including the original endorsing peers).
7. Peers validate and commit block trans:
   - runs validation logic (VSCC to check endorsement policy, and MVCC to check that ReadSet versions haven't changed in State DB since simulation time)
   - indicates in block which trans are valid and invalid
   - commits block to blockchain on file system, and commits valid transactions within block to state database 'atomically'
   - fires events so that application client listening via SDK knows which transactions were valid/invalid

Transaction Lifecycle v1.0 with State DB



Client's Proposal Message Format

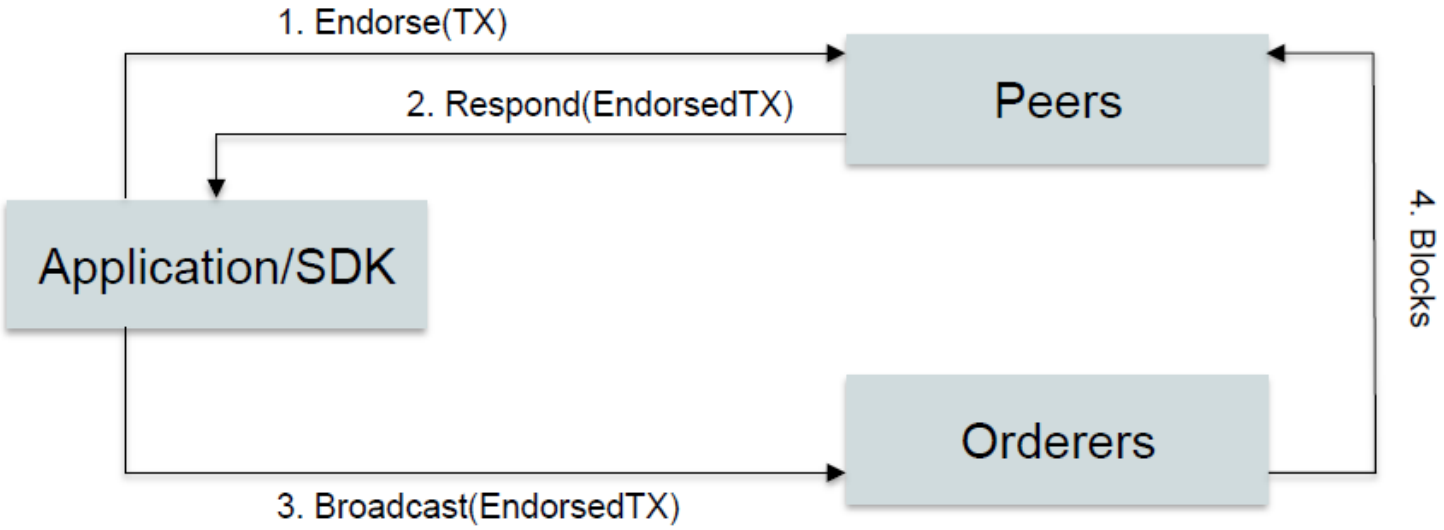


Note: If the client specifies the anchor argument, an endorser endorses a transaction only upon *read* version numbers of corresponding keys in its local KVS match anchor.

- clientID is an ID of the submitting client,
- chaincodeID refers to the chaincode to which the transaction pertains,
- txPayload is the payload containing the submitted transaction itself,
- timestamp is a monotonically increasing (for every new transaction) integer maintained by the client,
- clientSig is signature of a client on other fields of tx.
- anchor contains *read version dependencies*, or more specifically, key's read version in the world state (readset).

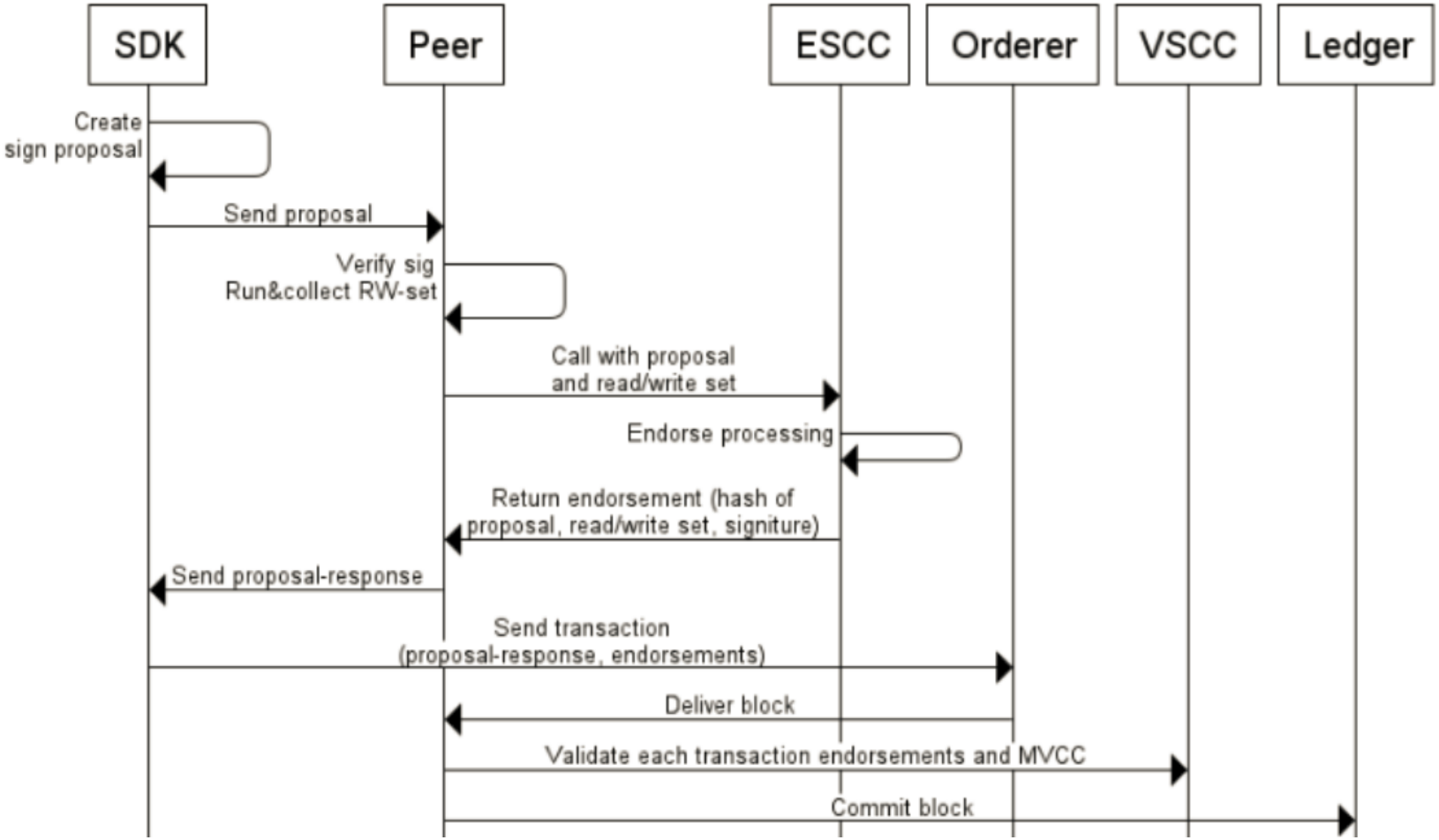
- operation denotes the chaincode operation (function) and arguments,
- metadata denotes attributes related to the invocation.
- source denotes the source code of the chaincode,
- metadata denotes attributes related to the chaincode and application,
- policies contains policies related to the chaincode such as the endorsement policy.

Transaction Data Flow



1. App requests 1 or more peers to endorse a transaction, which executes the chaincode but doesn't commit the changes
2. Each endorsing peer signs (endorses) the result and returns to the app
3. App collects endorsements and sends to ordering service
4. Orderer delivers blocks to the peers (on the chain), who validate and commit blocks to the ledger

Transaction Flow Inside a Peer



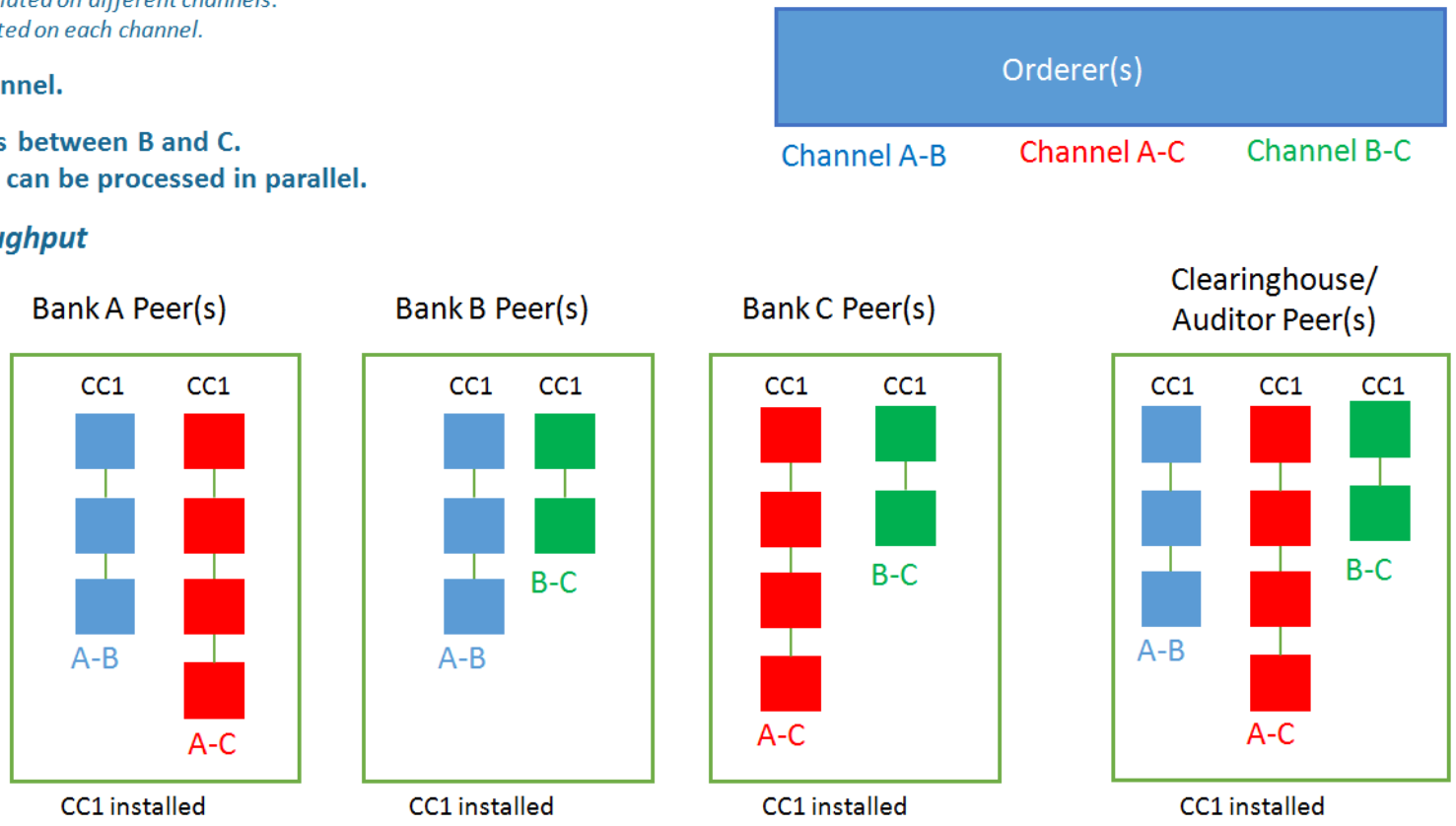
Example Scenario with Multi-Channel

Chaincode1 installed on all 4 peers.
Chaincode1 instantiated on all 3 channels*
**Different chaincodes could be instantiated on different channels.*
**Multiple chaincodes can be instantiated on each channel.*

One distributed ledger per channel.

Bank A cannot see transactions between B and C.
Blocks from different channels can be processed in parallel.

➔ ***Privacy + increased throughput***



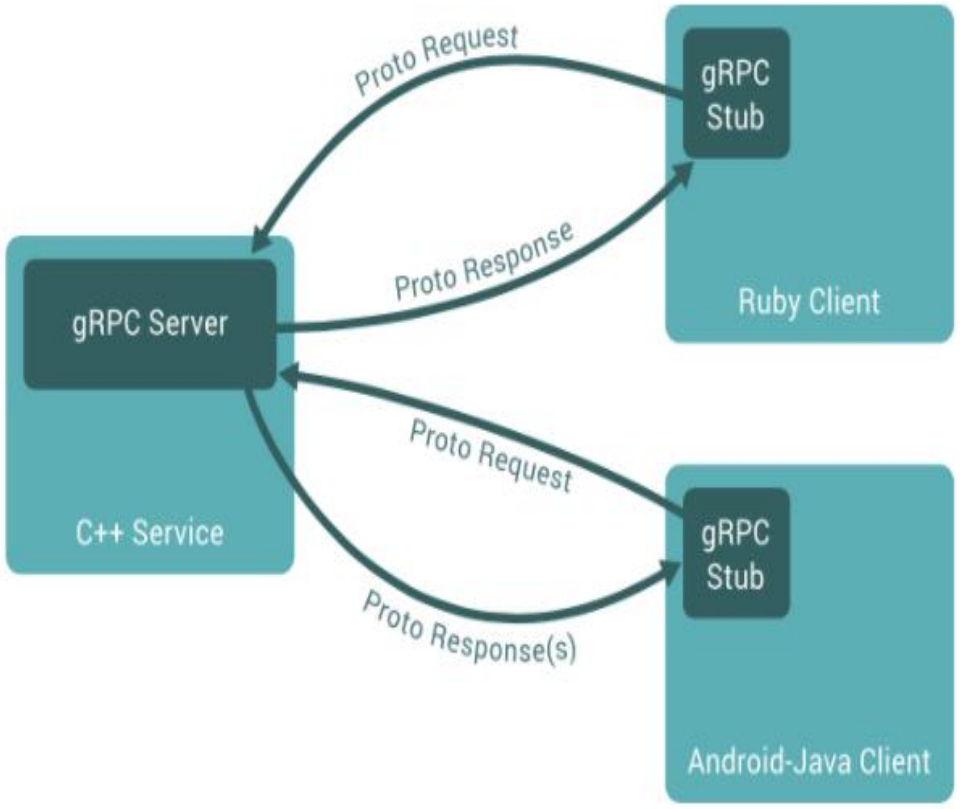
Protocols

Gossip: Data Distribution Protocol

Hyperledger Fabric optimizes blockchain network performance, security and scalability by dividing workload across transaction execution (endorsing and committing) peers and transaction ordering nodes. This decoupling of network operations requires a secure, reliable and scalable data dissemination protocol to ensure data integrity and consistency. To meet these requirements, the fabric implements a ***gossip data dissemination protocol****.*

gossip *protocol guaranteeing data consistency and integrity across the channel. Each peer on a channel periodically send its data to randomly-selected peers on the same channel. Each peer on a channel is continuously receiving current and consistent data from multiple peers. Any data that is transmitted from a Byzantine node, which is neither current nor consistent, is identified and filtered out of the network (channel).*

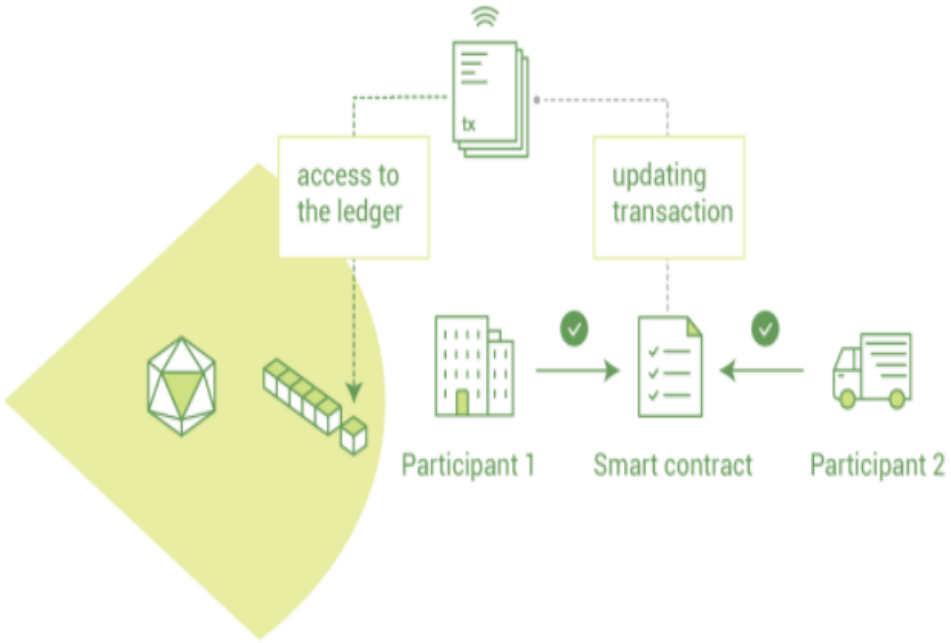
gRPC: Communication Protocol



gRPC is a high performance, open-source remote procedure call (RPC) framework similar to CORBA that can run anywhere. It enables client and server applications to communicate transparently, and makes it easier to build connected systems. The gRPC framework is developed and open-sourced by Google. gRPC uses standards based HTTP/2 as transport allowing it to easily traverse proxies and firewalls.

Smart Contract

Overview



Smart contracts implement business logic specifically oriented towards a blockchain's distributed, transparent ledger. Smart contracts service two classes of message:

- *transactions, which change ledger state, and*
- *queries, which read ledger state*

Chaincode

*The widely-used term, **smart contract**, is referred to as "**chaincode**" in Hyperledger Fabric.*

Chaincode is a program contains business logic. Chaincode initializes and manages the ledger state through transactions submitted by applications.

*The supported chaincode language is **Go** and **Java**. They are also looking into developing a templating language (such as **Apache Velocity**).*

Two types of chaincode:

- *User-defined chaincode is encapsulated in a Docker container;*
- *System chaincode runs in the same process as the peer;*

Chaincode API

I. **main()** - main function starts up the chaincode in the container during instantiate
II. **Init()** - Init is called during chaincode instantiation to initialize any data. Note that chaincode upgrade also calls this function to reset. This is similar to constructor in Java.
III. **Invoke()** - Invoke is called per transaction on the chaincode. Each transaction is either a 'get' or a 'set' on the asset created by Init function. The 'set' method may create a new asset by specifying a new key-value pair. Note, this is the only function from where you can call other user defined functions for better code modularity. E.g query(), delete(), get(), set() etc.

Chaincode Interface (*shim.ChaincodeStubInterface*)

GetStringArgs() []string - GetStringArgs returns the arguments intended for the chaincode Init and Invoke as a string array. Only use GetStringArgs if the client passes arguments intended to be used as strings.

GetFunctionAndParameters() (string, []string) - GetFunctionAndParameters returns the first argument as the function name to be called and the rest of the arguments as parameters in a string array.

GetTxID() string - GetTxID returns the tx_id of the transaction proposal.

*GetState(key string) ([]byte, error) - GetState returns the value of the specified `key` from the ledger. If the key does not exist in the state database, (nil, nil) is returned. Note that GetState doesn't read data from the **writeset**, which has not been committed to the ledger.*

*PutState(key string, value []byte) error - PutState puts the specified `key` and `value` into the transaction's **writeset** as a data-write proposal. Note that PutState doesn't effect the ledger until the transaction is validated and successfully committed.*

*DelState(key string) error - DelState records the specified `key` to be deleted in the **writeset** of the transaction proposal. The `key` and its value will be deleted from the ledger when the transaction is validated and successfully committed.*

User Vs System Chaincode

System chaincode has the same programming model except that it runs within the peer process rather than in an isolated container like normal chaincode. Therefore, system chaincode is built into the peer executable.

List of System Chaincode

1. *LSCC Lifecycle system chaincode handles lifecycle requests described above.*
2. *CSCC Configuration system chaincode handles channel configuration on the peer side.*
3. *QSCC Query system chaincode provides ledger query APIs such as getting blocks and transactions.*
4. *ESCC Endorsement system chaincode handles endorsement by signing the transaction proposal response.*
5. *VSCC Validation system chaincode handles the transaction validation, including checking endorsement policy and multiversioning concurrency control.*

Install Chaincode

*The **install** transaction packages a **chaincode's** source code into a prescribed format called a **ChaincodeDeploymentSpec** (or CDS) and installs it on a peer node that will run that chaincode. **Note**: You must install the chaincode on **each** endorsing peer (one peer in each organisation) node of a channel that will run your chaincode.*

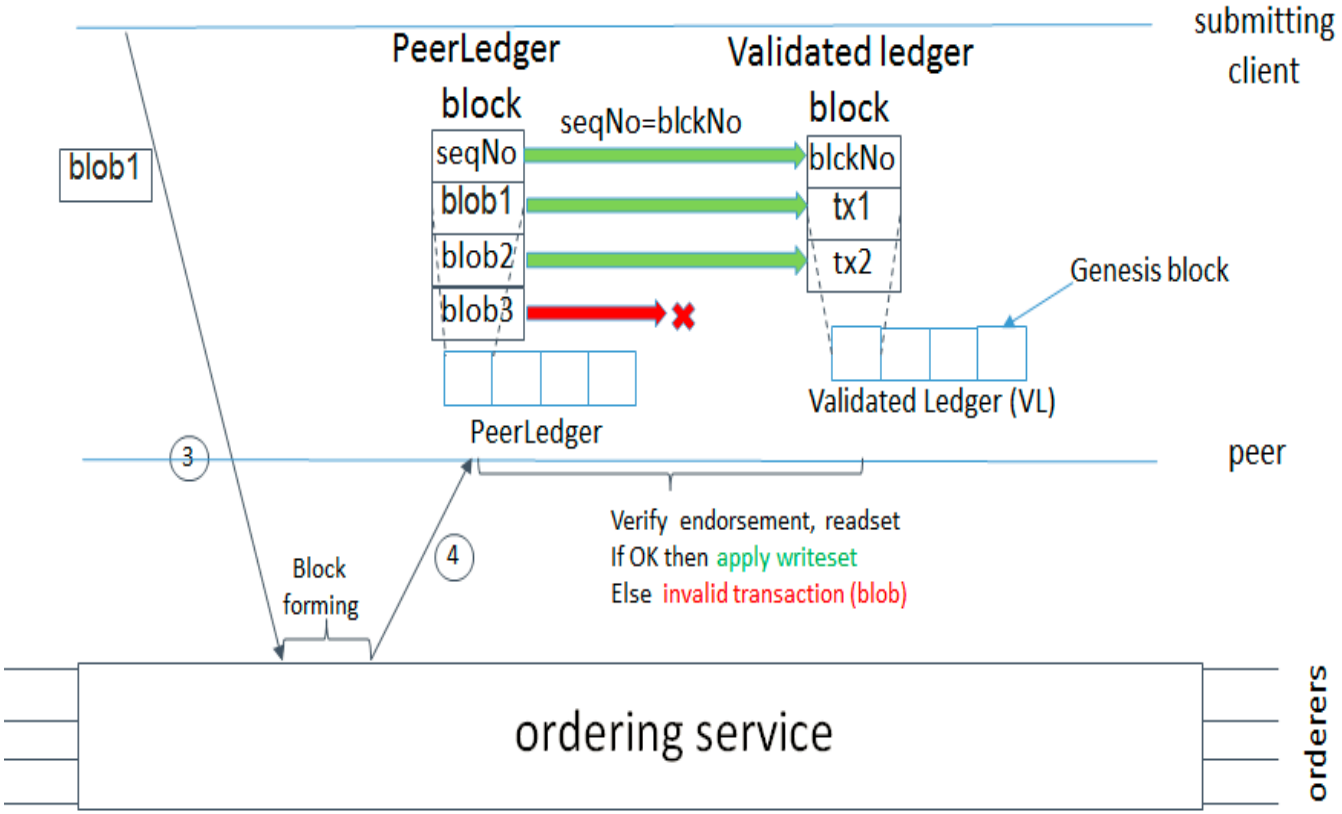
Peer

Overview

In Peer-to-Peer architectures where there are no special machines that provide a service or manage the network resources. Instead all responsibilities are uniformly divided among all machines, known as peers. Peers can serve both as clients and as servers.

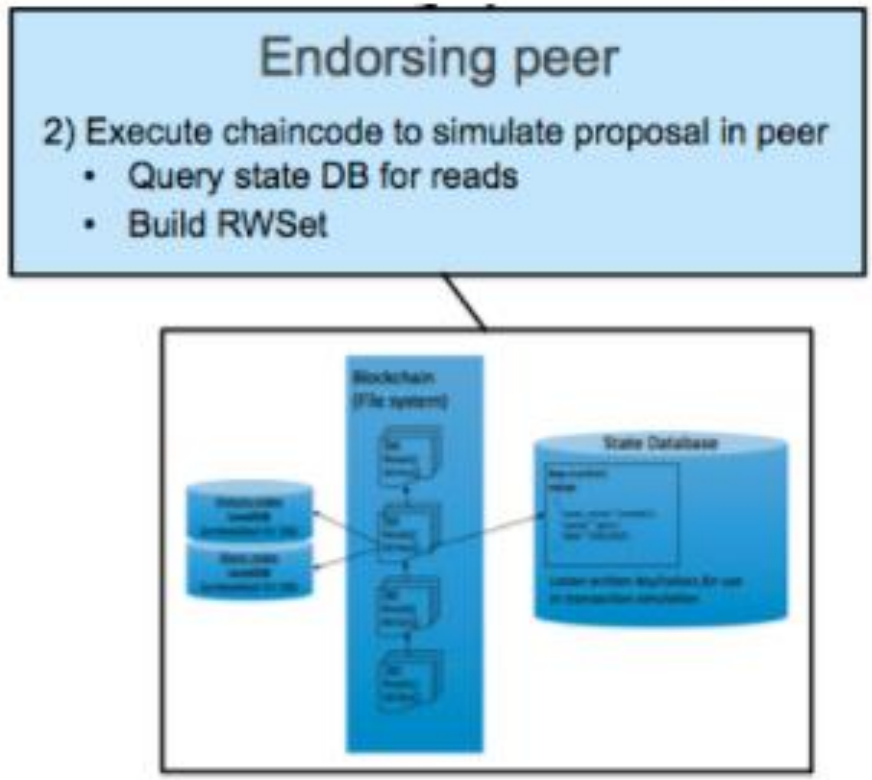
Fabric define Peer as a network entity that maintains a ledger and runs chaincode containers in order to perform read/write operations to the ledger. Peers are owned and maintained by members.

PeerLedger (Ledger Checkpointing)



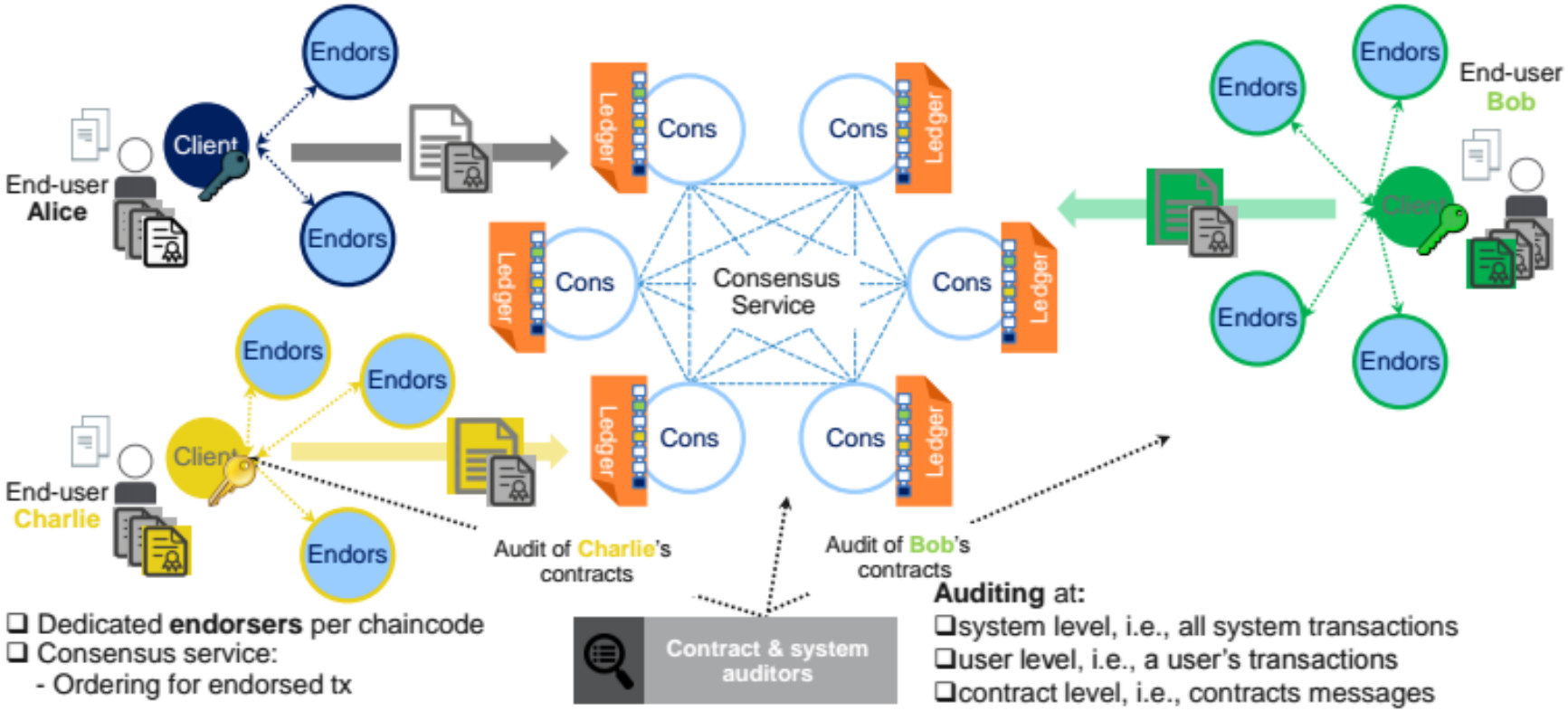
Peer maintain PeerLedger (with valid and invalid transactions), State and additionally Validated Ledger (valid transactions only). Invalid transactions are filtered out and only valid transactions block get added into VLedger through checkpointing process.

Endorsing Peer

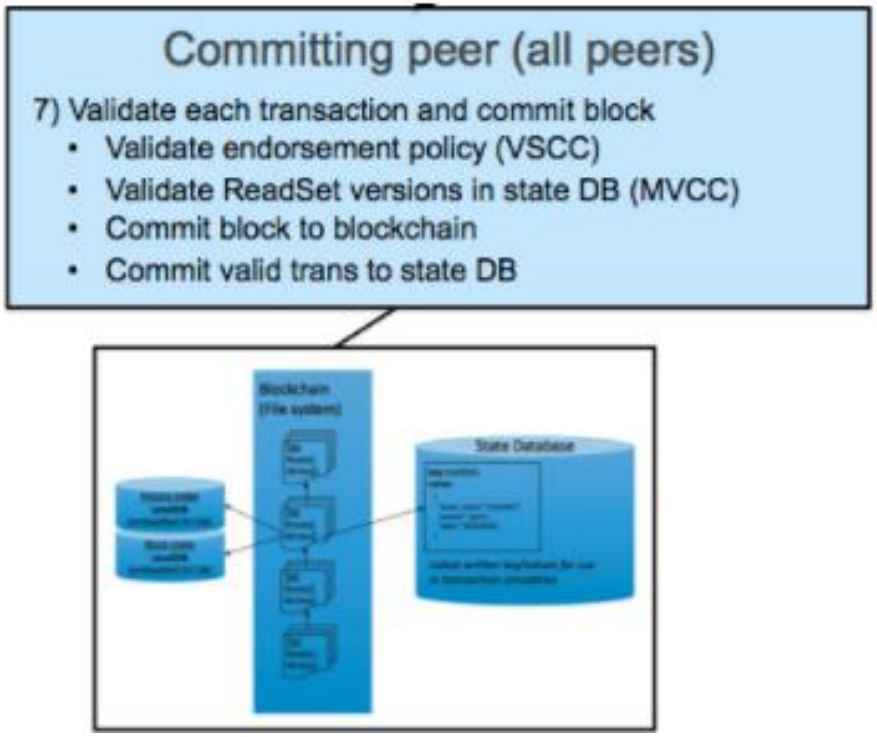


Refers to the process where specific peer nodes execute a transaction and return a YES/NO response to the client application that generated the transaction proposal. Chaincode applications have corresponding endorsement policies, in which the endorsing peers are specified.

Separating Transaction Endorsement from Consensus



Committing Peer



Maintainers of the ledger, a.k.a the committer nodes. Note that all peers are always committers.

Anchor Peer

Each Member(Organisation) on a channel has an anchor peer (or multiple anchor peers to prevent single point of failure), allowing for peers belonging to different Members(Organisations) to discover all existing peers that are available on a channel.

Leading Peer

The election of a 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. The consensus service orders transactions and delivers them, in a block, to each leading peer, which then distributes the block to its member peers, and across the channel, using the gossip protocol.

Ledger

Overview

Ledger provides a verifiable history of all successful state changes (valid transactions) and unsuccessful attempts to change state (invalid transactions), occurring during the operation of the system.

Ledger is constructed by the ordering service as a totally ordered hashchain of blocks of (valid or invalid) transactions. The hashchain imposes the total order of blocks in a ledger and each block contains an array of totally ordered transactions. This imposes total order across all transactions.

There are two types of ledger:

- *PeerLedger*
- *OrdererLedger*

***Note**: PeerLedger differs from the OrdererLedger in that peers locally maintain a bitmask that tells apart valid transactions from invalid ones*

Ledger (Chain + State)

The ledger is the sequenced, tamper-resistant record of all state transitions in the fabric. State transitions are a result of chaincode invocations ('transactions') submitted by participating parties. Each transaction results in a set of asset key-value pairs that are committed to the ledger as creates, updates, or deletes.

The ledger is comprised of a blockchain ('chain') to store the immutable, sequenced record in blocks, as well as a state database to maintain current fabric state. There is one ledger per channel. Each peer maintains a copy of the ledger for each channel of which they are a member.

Chain

The chain is a transaction log, structured as hash-linked blocks, where each block contains a sequence of N transactions. The block header includes a hash of the block's transactions, as well as a hash of the prior block's header. In this way, all transactions on the ledger are sequenced and cryptographically linked together. In other words, it is not possible to tamper with the ledger data, without breaking the hash links. The hash of the latest block represents every transaction that has come before, making it possible to ensure that all peers are in a consistent and trusted state.

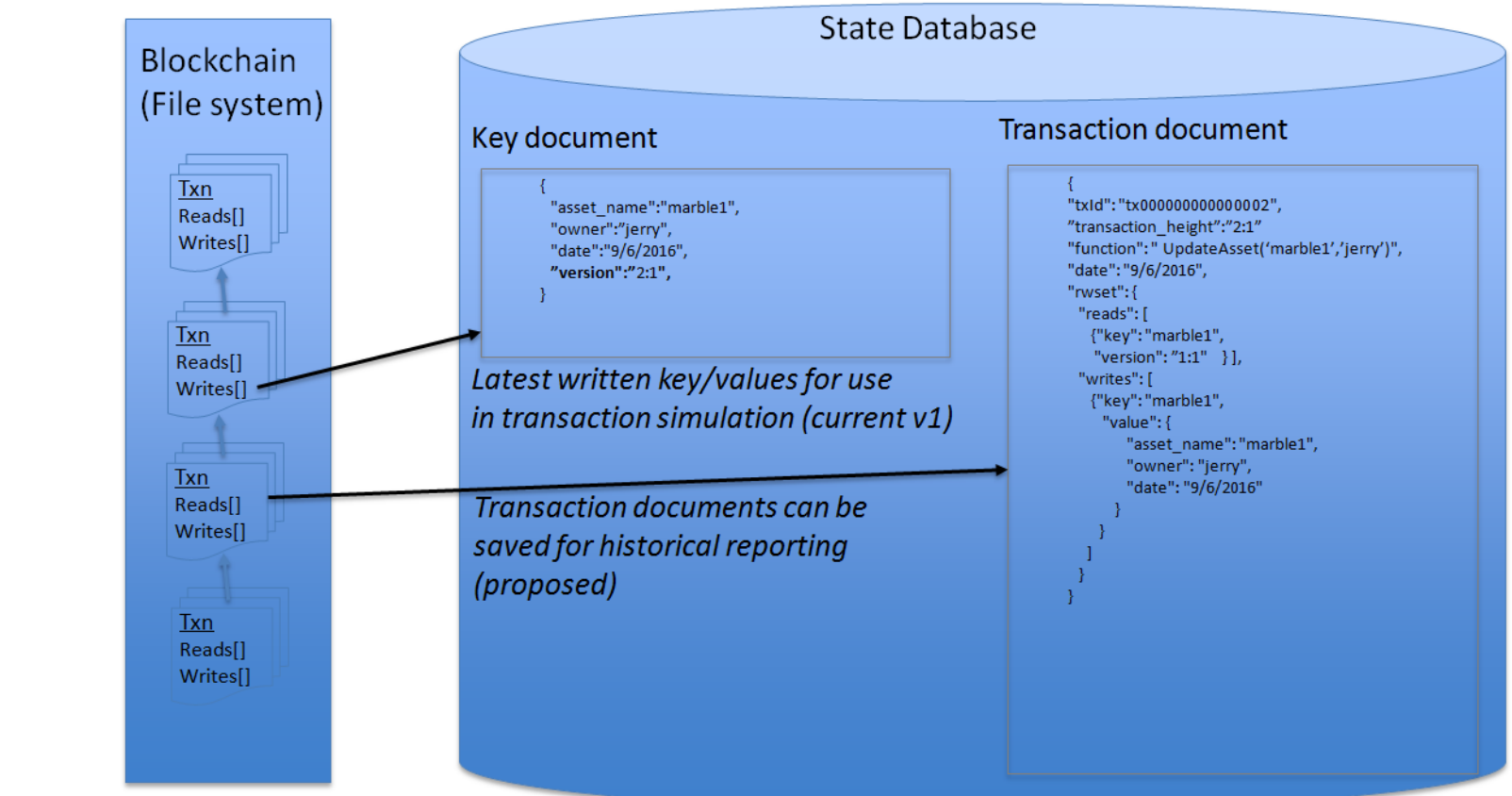
The chain is stored on the peer file system (either local or attached storage), efficiently supporting the append-only nature of the blockchain workload.

State

The ledger's current state data represents the latest values for all keys ever included in the chain transaction log. Since current state represents all latest key values known to the channel, it is sometimes referred to as World State.

Chaincode invocations execute transactions against the current state data. To make these chaincode interactions extremely efficient, the latest values of all keys are stored in a state database. The state database is simply an indexed view into the chain's transaction log, it can therefore be regenerated from the chain at any time. The state database will automatically get recovered (or generated if needed) upon peer startup, before transactions are accepted.

Ledger v1.0

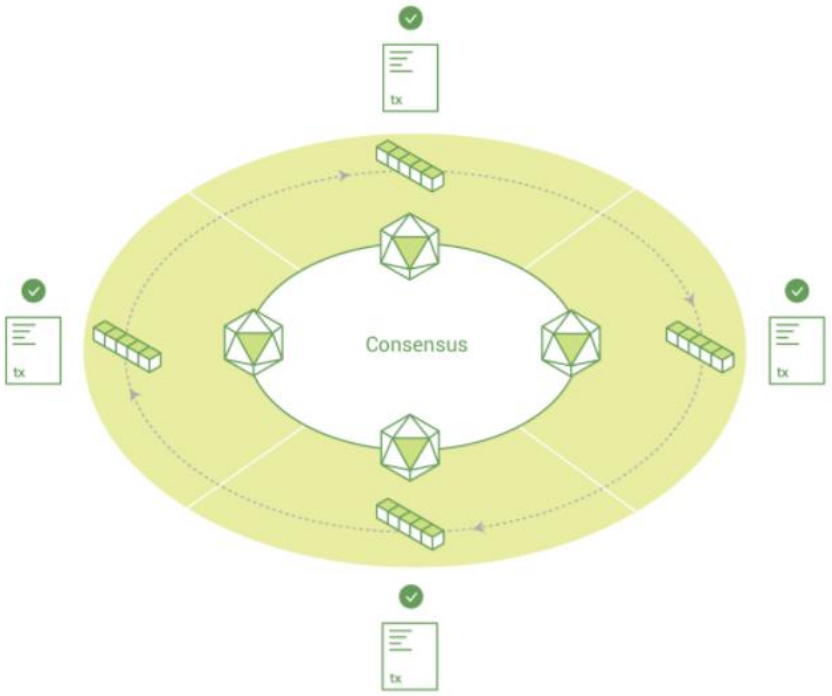


Immutable source of truth

'Index' of the blockchain for runtime queries

Consensus

Overview



A broader term overarching the entire transactional flow, which serves to generate an agreement on the order and to confirm the correctness of the set of transactions constituting a block.

Modular consensus in Fabric V1.0

There are three pluggable consensus mechanism available in Fabric v1.0. These are:

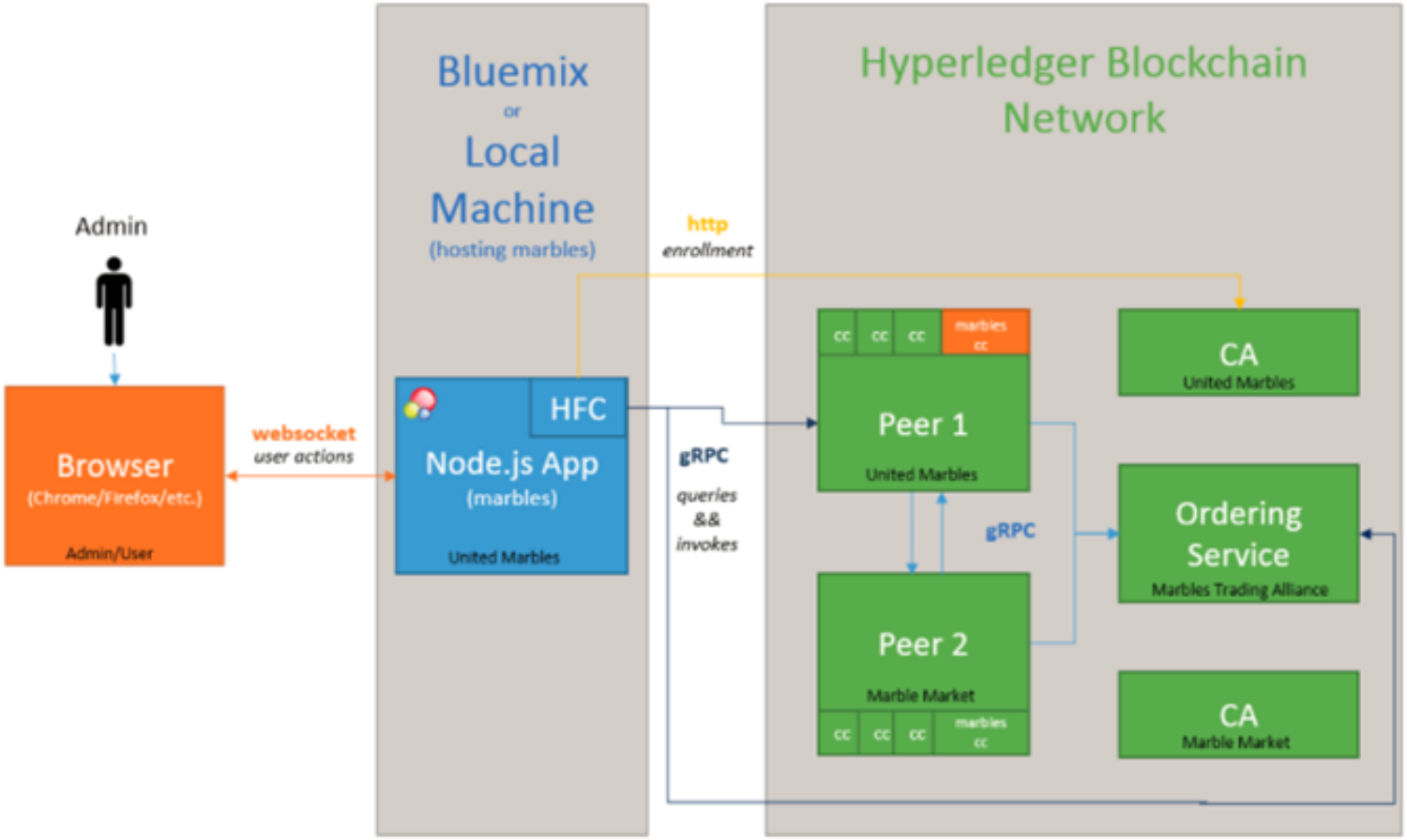
- ***Solo orderer***
  - *One host only, acting as specification during development (ideal functionality)*
- ***Apache Kafka**, a distributed pub/sub streaming platform*
  - *Tolerates crashes among member nodes, has Apache Zookeeper*
  - *Focus on high throughput*
  - *Each channel maps to a separate single-partition topic in Kafka.*
- ***SBFT** - A simple implementation of Practical Byzantine Fault Tolerance (PBFT)*
  - *Tolerates f < n/3 Byzantine faulty nodes among n*
  - *Focus on resilience*

State DB

State Database options - Queryability

- *In a **key/value database** such as **LevelDB**, the content is a blob and only queryable by key*
  - *Does not meet chaincode, auditing, reporting requirements for many use cases*
- *In a **document database** such as **CouchDB**, the content is JSON and fully queryable*
  - *Meets a large percentage of chaincode, auditing, and simple reporting requirements*
  - *For deeper reporting and analytics, replicate data to an analytics engine such as Spark (future)*
  - *Id/document data model compatible with existing chaincode key/value programming model, therefore no application changes are required when modeling chaincode data as JSON*
- ***SQL data stores** would require more complicated relational transformation layer, as well as schema management.*

Application Communication Flow



Fabric CA

Overview

*The Hyperledger Fabric CA is a **Certificate Authority** (CA) in Hyperledger Fabric.*

It provides features such as:

- *Registration of identities, or connects to LDAP as the user registry*
- *Issuance of Enrollment Certificates (ECerts)*
- *Issuance of Transaction Certificates (TCerts)*
- *Certificate renewal and revocation*

Fabric PKI Components

- *Certificate Authority (CA)*
- *Registration Authority (RA)*
- *A certificate database*
- *A certificate storage*

List of Authorities in Fabric PKI

- *Root Certificate Authority - Root CA*
- *Registration Authority (RA) – User management*
- *Enrollment Certificate Authority (ECA) - Intermediate CA*
- *Transaction Certificate Authority (TCA) - Intermediate CA*
- *TLS Certificate Authority (TLS-CA) - Intermediate CA*

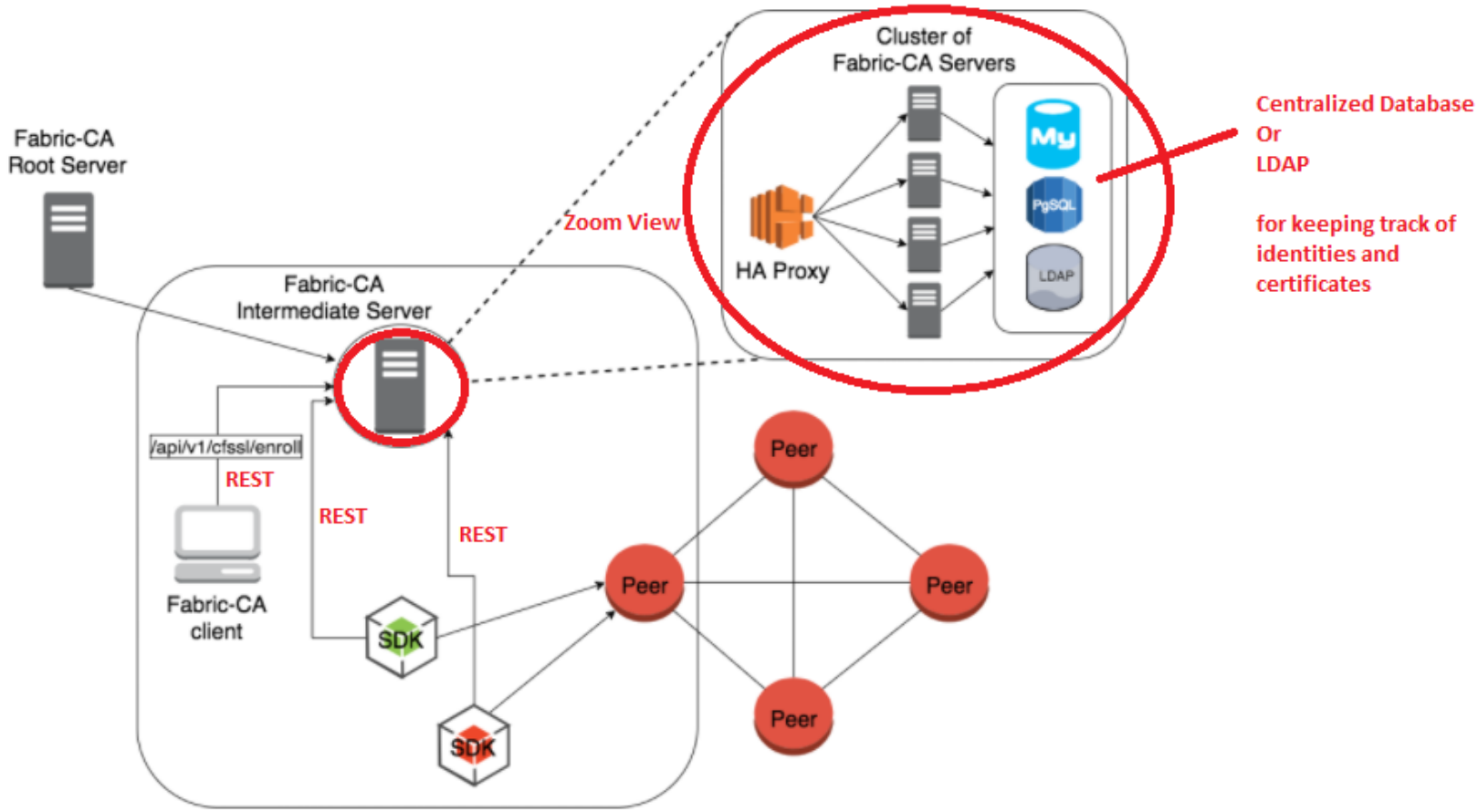
Certificate Types in Fabric PKI

- *Enrollment Certificates (**ECerts**) - They are issued for all roles, i.e. users, peers, orderers.*
- *Transaction Certificates (**TCerts**) - They are issued only to users.*
- *TLS-Certificates (**TLS-Certs**) - TLS-Certs are certificates used for system-to-system/component-to-system/component-to-component communications.*

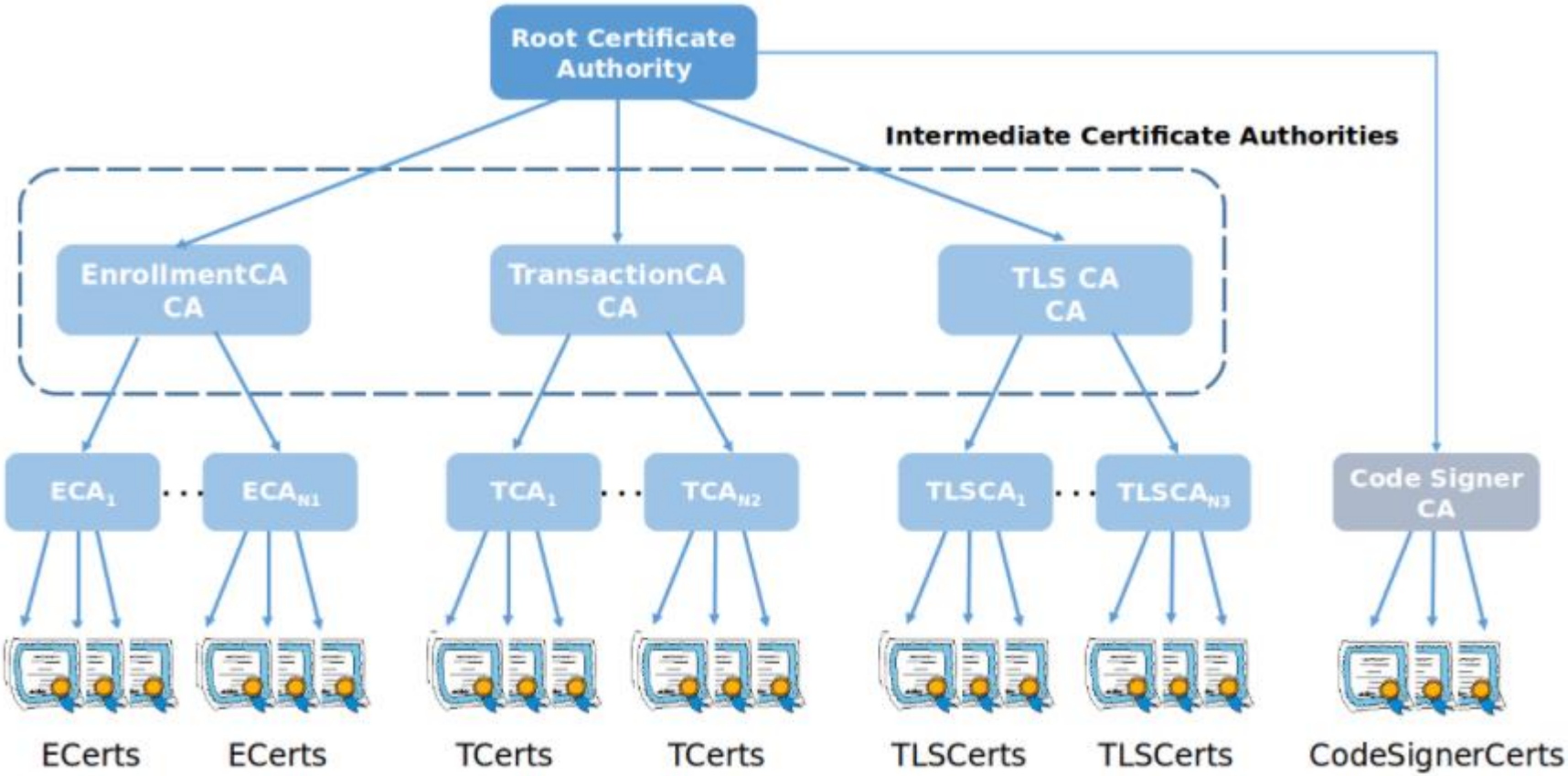
***Note**: The ECA, TCA, and TLS CA certificates are self-signed, where the TLS CA is provisioned as a trust anchor (Root CA).*

Fabric-CA Architecture

The diagram below illustrates how the Hyperledger Fabric CA server fits into the overall Hyperledger Fabric architecture.



“Chain of Trust” Hierarchy in Fabric



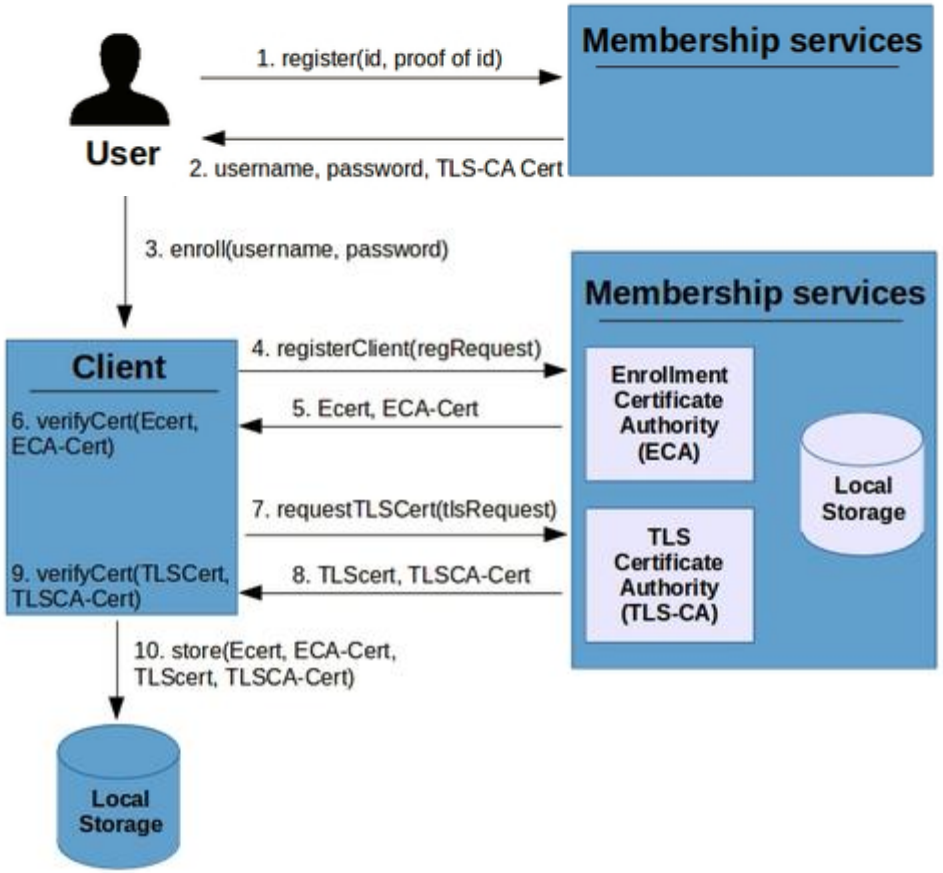
Fabric Certificate

*As of v1.0, **identities** are based on the Public Key Infrastructure (PKI) standards. Every orderer node, every peer node and every user/transactor must have a key pair with the public key wrapped in a x.509 certificate signed by a Certificate Authority (CA).*

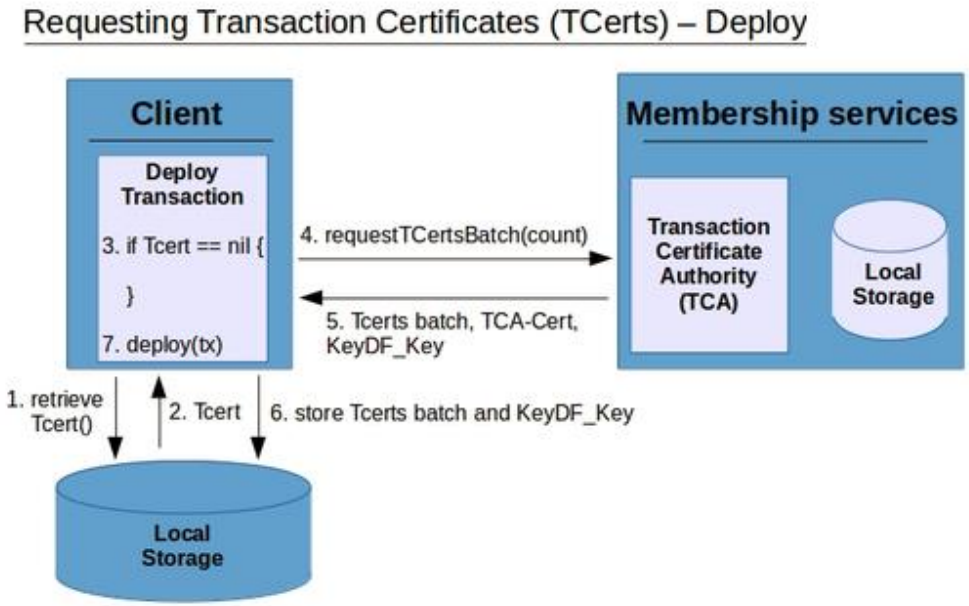
We will need a whole bunch of certificates because there are many identities involved:

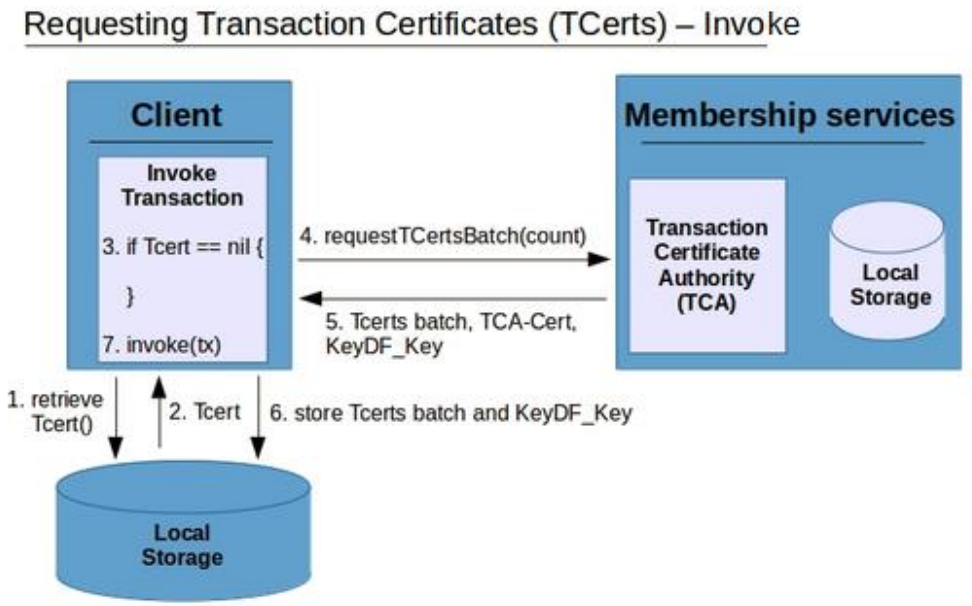
- *peers need identities to sign endorsements*
- *orderers need identities to sign proposed blocks for the committers to validate and append to the ledger*
- *applications need identities to sign transaction requests*
- *the Fabric CA themselves also need identities, so their signatures in the certificates can be validated*

Enrollment Flow (User, Client, Peer, Orderer)



Transaction Flow (Deploy Tx & Invoke Tx)





Security and Privacy

Overview

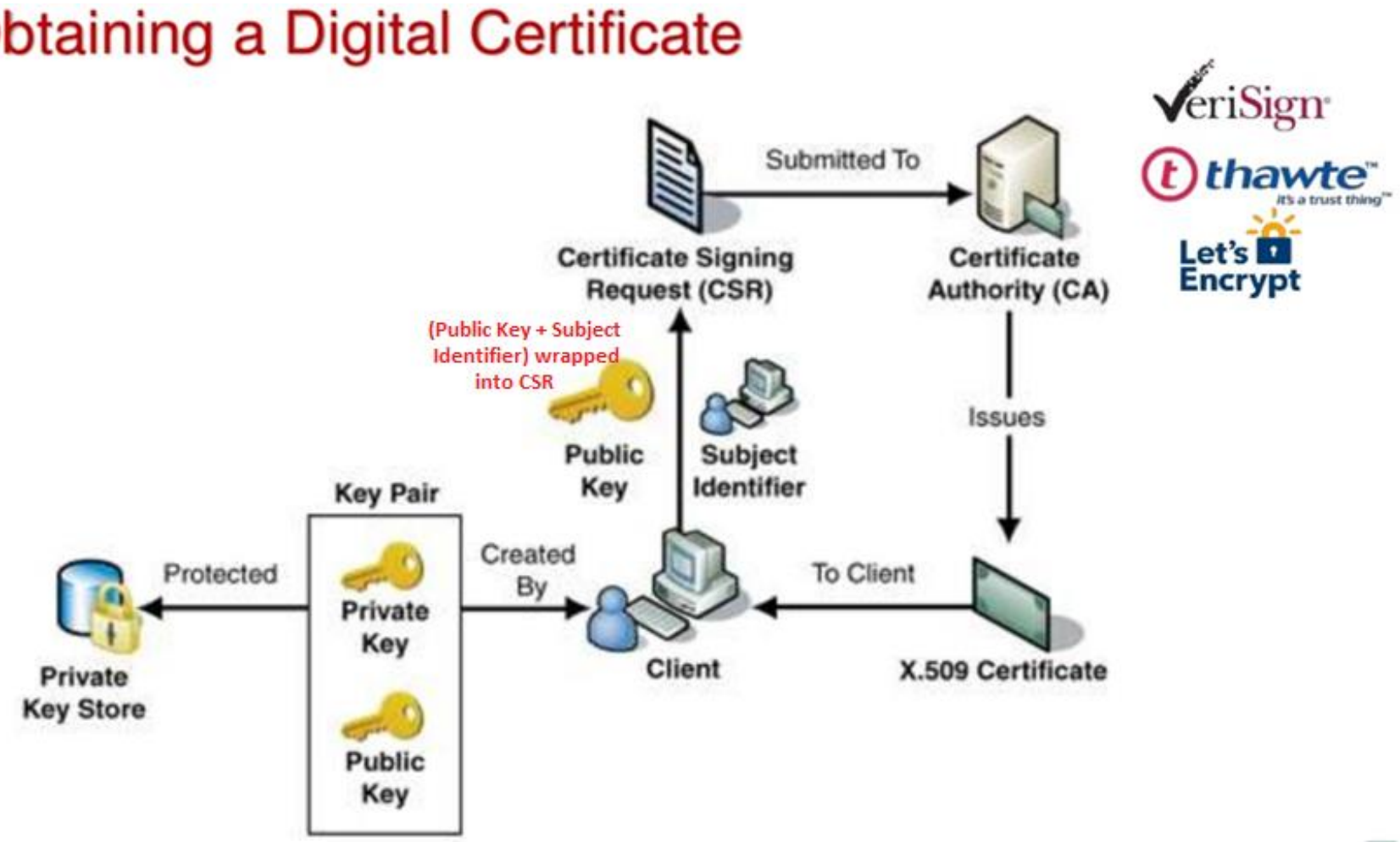
Hyperledger Fabric underpins a transactional network where all participants have known identities. ***Public Key Infrastructure*** *(PKI) is used to generate cryptographic certificates which are tied to organizations, network components, and end users or client applications. As a result, data access control can be manipulated and governed on the broader network and on channel levels. This "****permissioned****" notion of Hyperledger Fabric, coupled with the existence and capabilities of channels, helps address scenarios where privacy and confidentiality are paramount concerns.*

PKI (Public Key Infrastructure)

Identities are established with x.509 certificates. Certificate to be considered valid on the Fabric, it must be signed by a trusted Certificate Authority (CA). Fabric supports any standard CAs. In addition, Fabric also provides a CA server. We will need a whole bunch of certificates because there are many identities involved:

- *peers need identities to sign endorsements.*
- *orderers need identities to sign proposed blocks for the committers to validate and append to the ledger.*
- *clients need identities to sign transaction requests.*

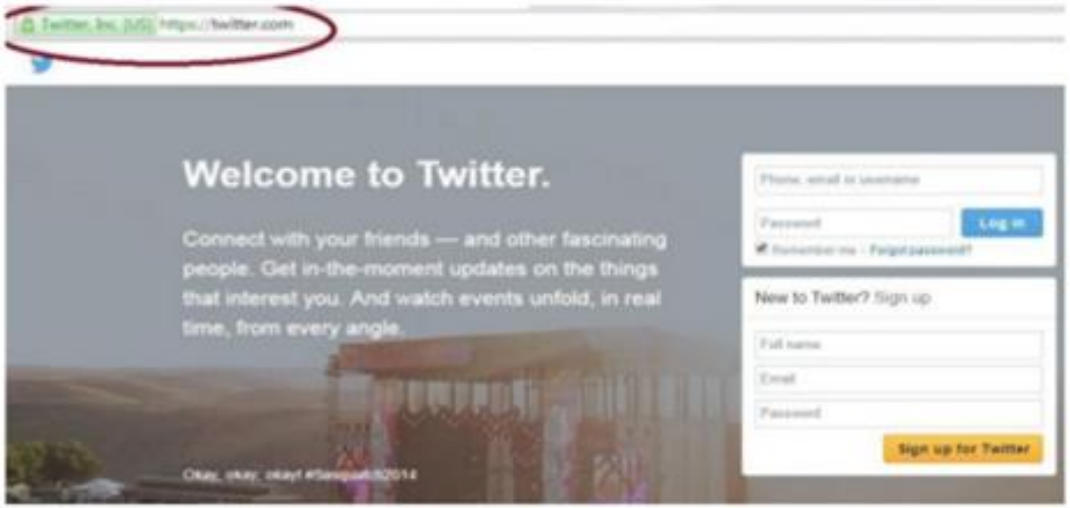
PKI (Certificate is signed by CA)



PKI (Self-signed Certificate)

- *Certificate is signed by own private key hence self-signed.*
- *Best for development.*
- *Very easy to generate locally with Fabric's cryptogen tool or OpenSSL.*

Self-Signed Certificate Vs Signed by Trusted CA



Signed by Certificate Authority

Self-Signed Certified

CSR - Certificate Signing Request

A Certificate Signing Request (CSR) is a request to sign a digital certificate. When you generate a certificate, the certificate signing request is sent to a trusted Certificate Authority (CA), such as VeriSign. The CA reads the CSR and returns a signed certificate to you. After you receive the signed certificate from the CA, you can use the signed certificate. Common attributes of a CSR:

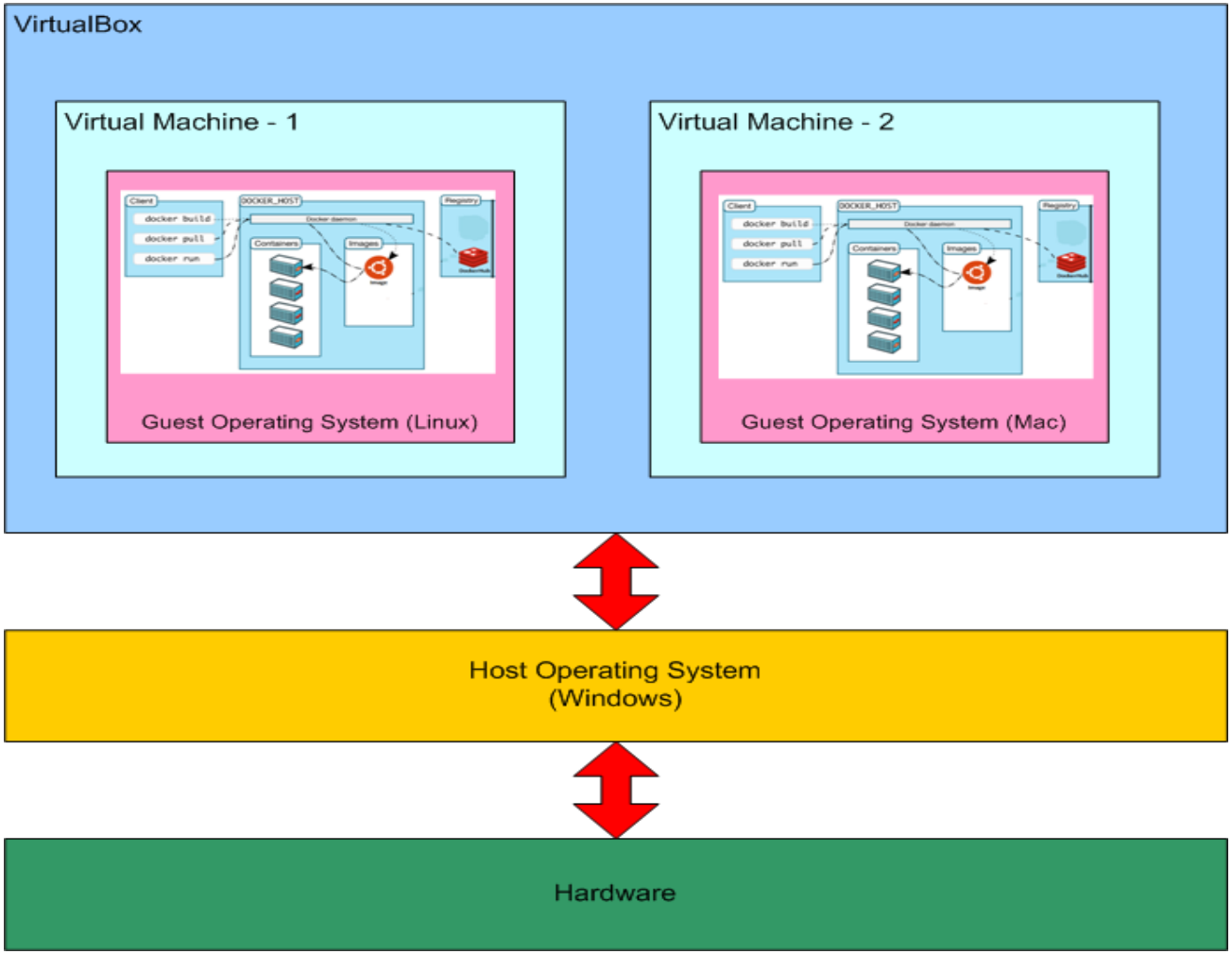
- *Common Name (CN) The Fully Qualified Domain Name (FQDN) of your server. This must match exactly what you type in your web browse.*
- *Organization The legal name of your organization.*
- *City/State/Country*
- *Email An email address used to contact your organization.*
- *Public Key The public key that will go into the certificate. The public key is created automatically.*

Generate a CSR using OpenSSL

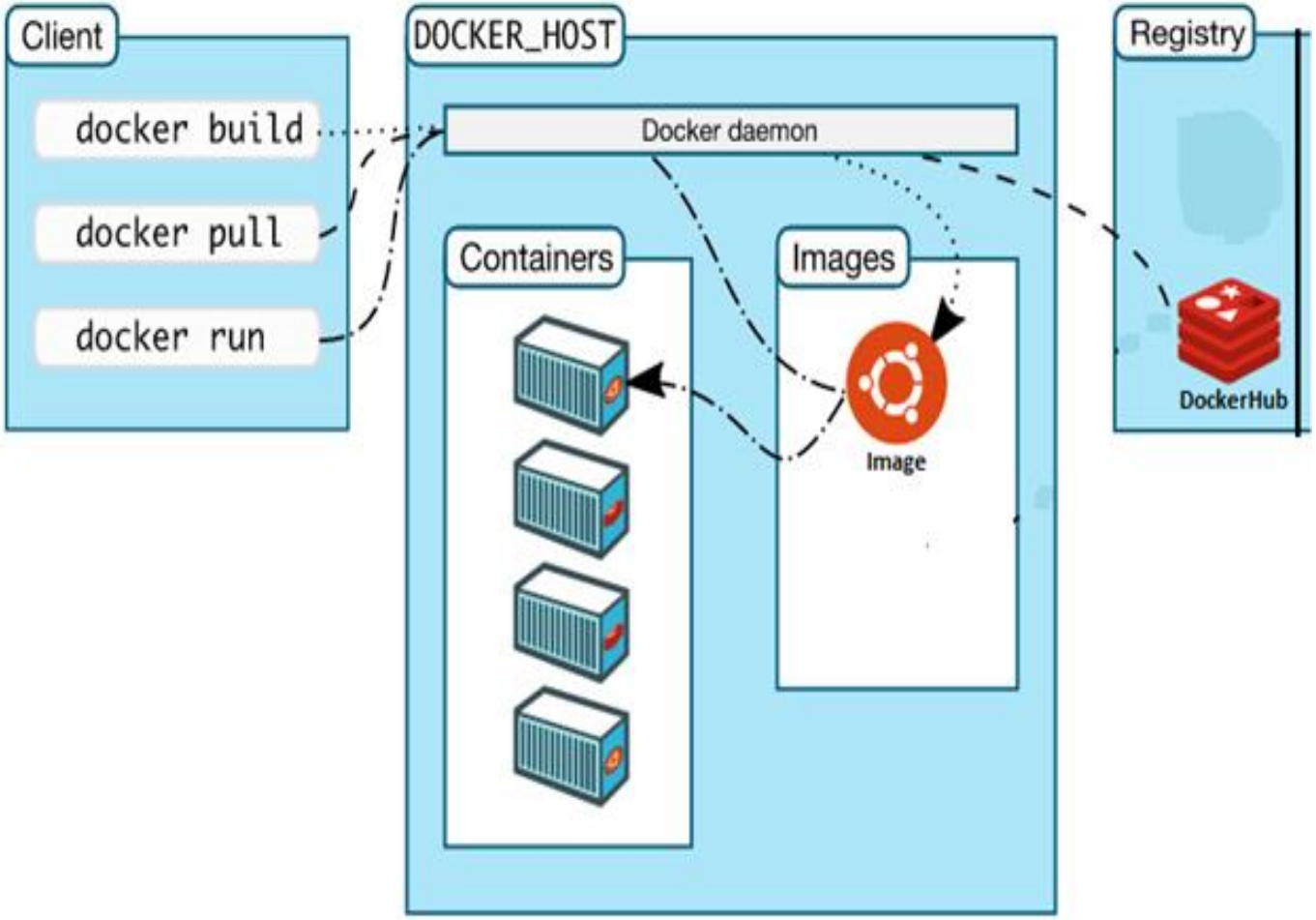
- *Run openssl command. The command prompt changes to OpenSSL>*
- *To generate a private key, at the OpenSSL> genrsa -out mykey.key 1024*
- *To generate a CSR with the private key, at the OpenSSL> req -new -key mykey.key -text -out mycsr.csr*
- *Add information about your organization to create a Distinguished Name (DN) for the CSR.*
- *OpenSSL> req -in mycsr.csr -noout -text*
- *OpenSSL> exit*

Docker

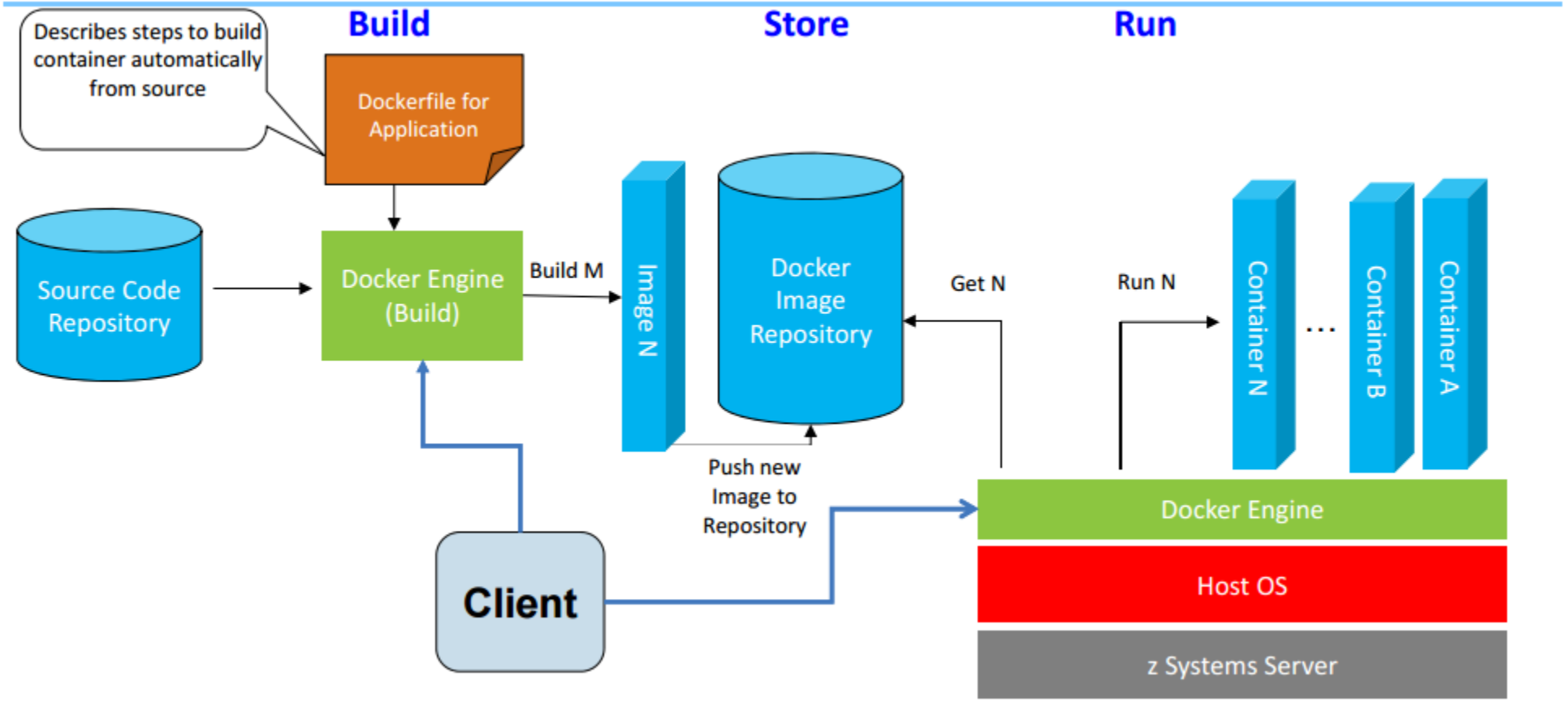
Docker Architecture



Docker Components



Docker Basic Functions



Software Development Kit (SDK)

Overview

The Hyperledger Fabric client SDK provides a structured environment of libraries for developers to write and test chaincode applications. The SDK is fully configurable and extensible through a standard interface. Components, including cryptographic algorithms for signatures, logging frameworks and state stores, are easily swapped in and out of the SDK. The SDK provides APIs for transaction processing, membership services, node traversal and event handling.

Supported SDK (Node.js, Java, Python, Go)

- *Node.js* - *https://github.com/hyperledger/fabric-sdk-node*
- *Java* - *https://github.com/hyperledger/fabric-sdk-java*
- *Python* - *https://github.com/hyperledger/fabric-sdk-py*
- *Go* - *https://github.com/hyperledger/fabric-sdk-go*

Some Features of the SDK

- ❑ create a new channel
- ❑ send channel information to a peer to join
- ❑ install chaincode on a peer
- ❑ instantiate chaincode in a channel
- ❑ submitting a transaction
- ❑ query a chaincode for the latest application state
- ❑ various other query capabilities
- ❑ monitoring events
- ❑ supports both TLS (grpcs://) or non-TLS (grpc://) connections to peers and orderers
- ❑ register a new user
- ❑ enroll a user to obtain the enrollment certificate signed by the Fabric CA
- ❑ revoke an existing user by enrollment ID

Basic Understanding of Node.js

Node.js is an open source command line tool built for the server side JavaScript code. The JavaScript is executed by the V8 JavaScript Engine (developed by Google built for Google Chrome browser, Written in C++, V8 compiles JavaScript source code to native machine code instead of interpreting it in real time). Node.js uses asynchronous programming! (Non-blocking call through attach event based callback functions) with single-threaded to serve client requests. So, No multithreading / locking bugs. Another reason is JavaScript. You can use node to share code between the browser and your backend. And the last reason is raw speed by V8 engine.

Hyperledger Composer

Overview

Hyperledger Composer enables architects and developers to quickly create "full-stack" blockchain solutions. I.e. business logic that runs on the blockchain, REST APIs that expose the blockchain logic to web or mobile applications, as well as integrating the blockchain with existing enterprise systems of record.

Composer integrates well with a modern development best practices and tools: Atom and VisualStudio Code editors, GitHub for source code management, unit test using Mocha, npm for managing versioned binary artifacts, Travis CI and Jenkins for continuous integration, deployment using Docker images or as npm packages; so that developers and operations engineers can easily move applications from a development laptop to a managed cloud environment.

** *The key concept for Composer is the **Business Network Definition (BND)**. It defines the data model, business logic and access control rules (ACL) for your blockchain solution. It executes on Hyperledger Fabric.*

Benefits of Fabric Composer

Increases understanding

Bridges simply from business concepts to blockchain

Saves time

Develop blockchain applications more quickly and cheaply

Reduces risk

Well tested, efficient design conforms to best practice

Increases flexibility

Higher level abstraction makes it easier to iterate

Open Toolset for Developers

Data modelling

JavaScript business logic

Web playground

Client libraries

Atom

Visual Studio Code

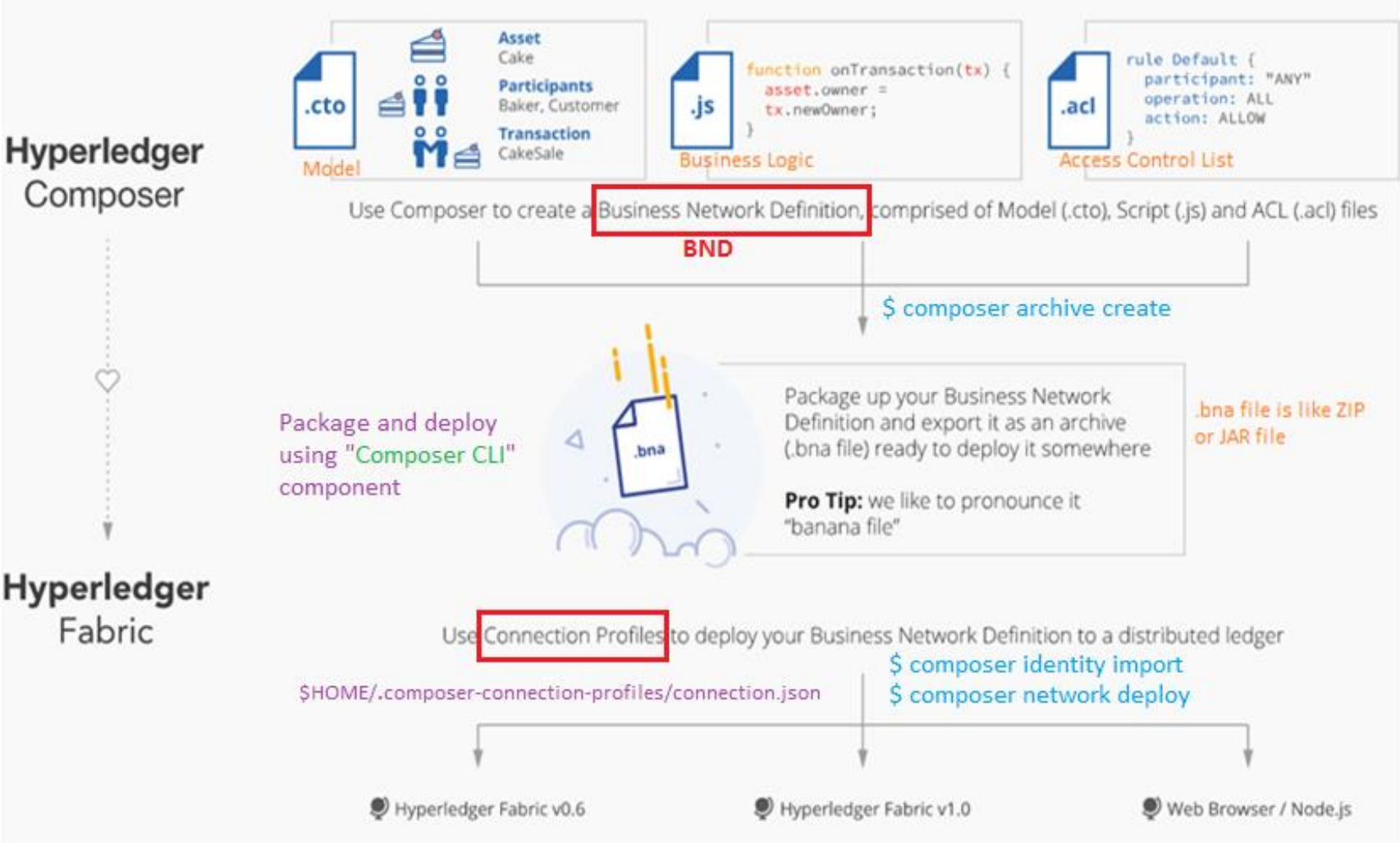
Editor support

CLI utilities

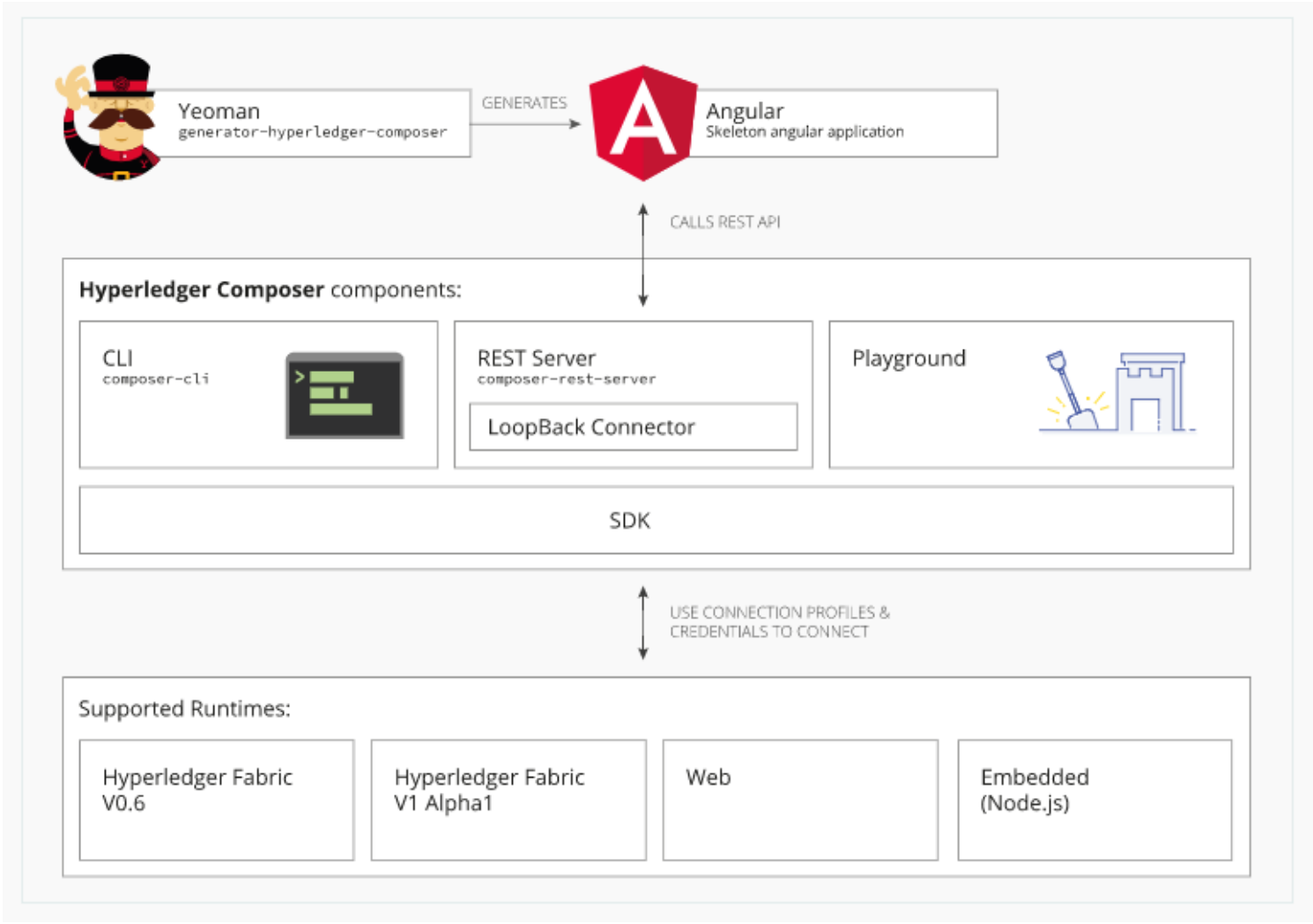
Code generation

Integration

Composer Architecture

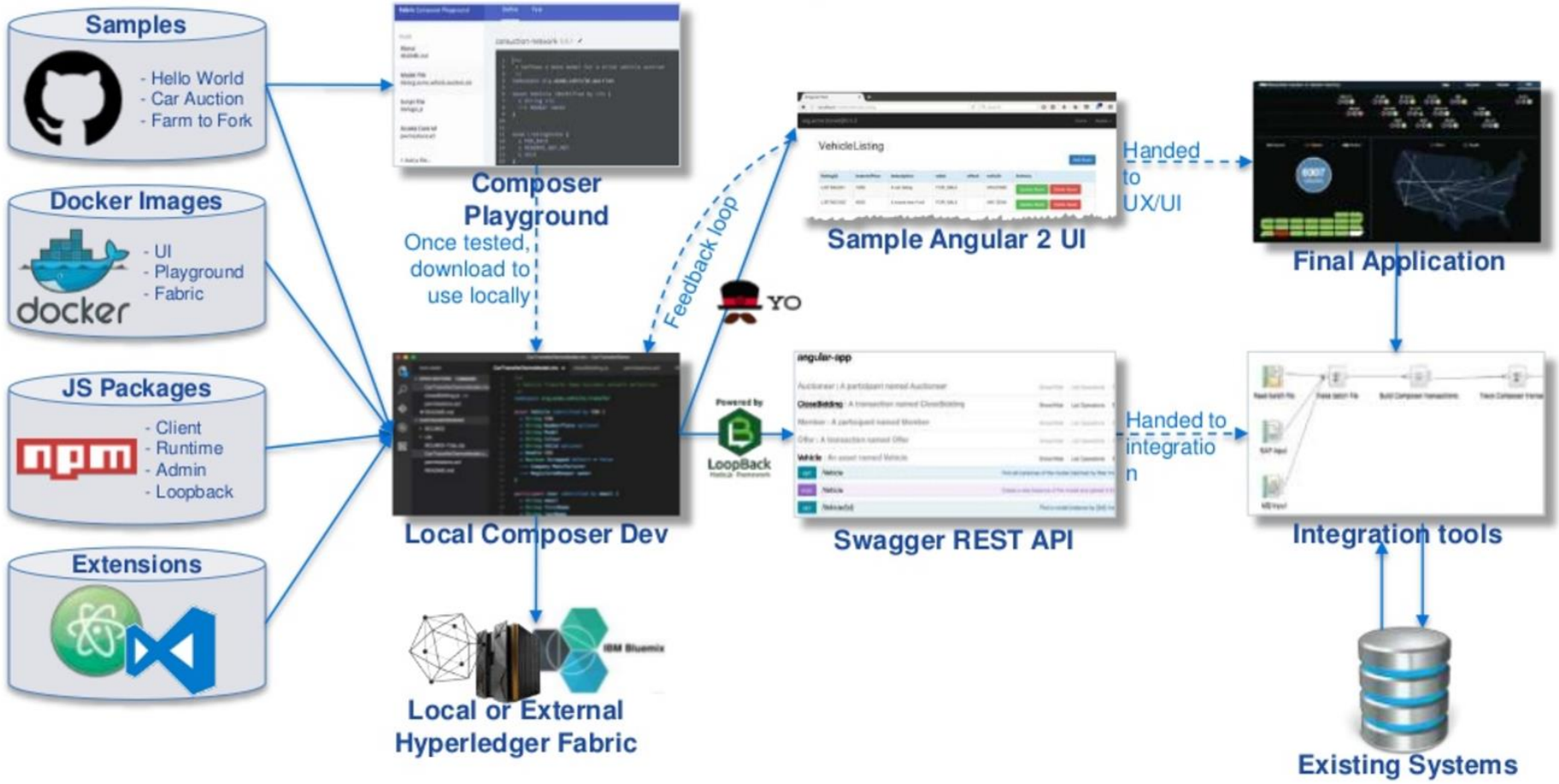


Typical Composer Solution Architecture



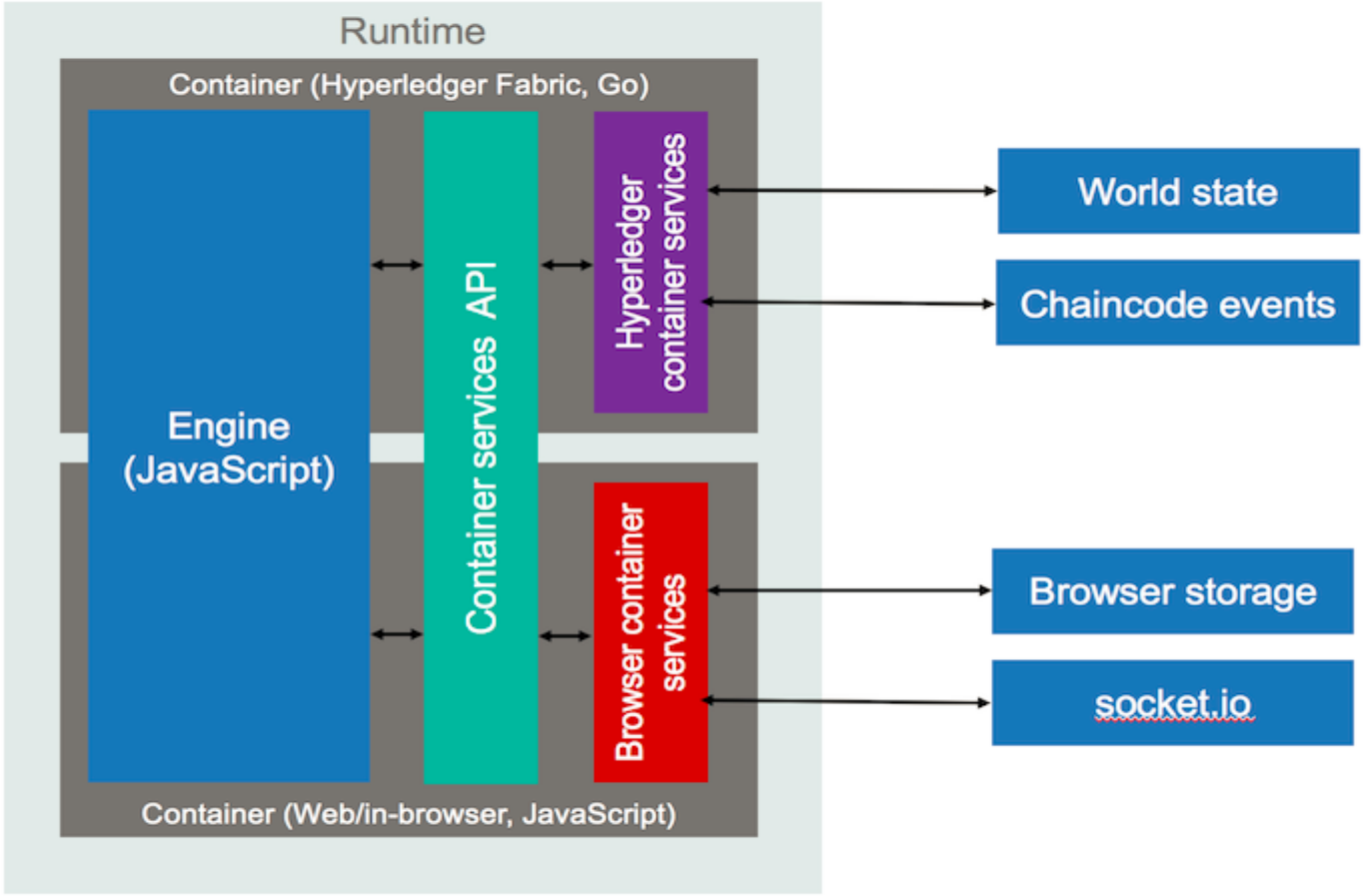
Getting Started with Composer

https://www.slideshare.net/dselman/hyperledger-composer-update-2017-0405

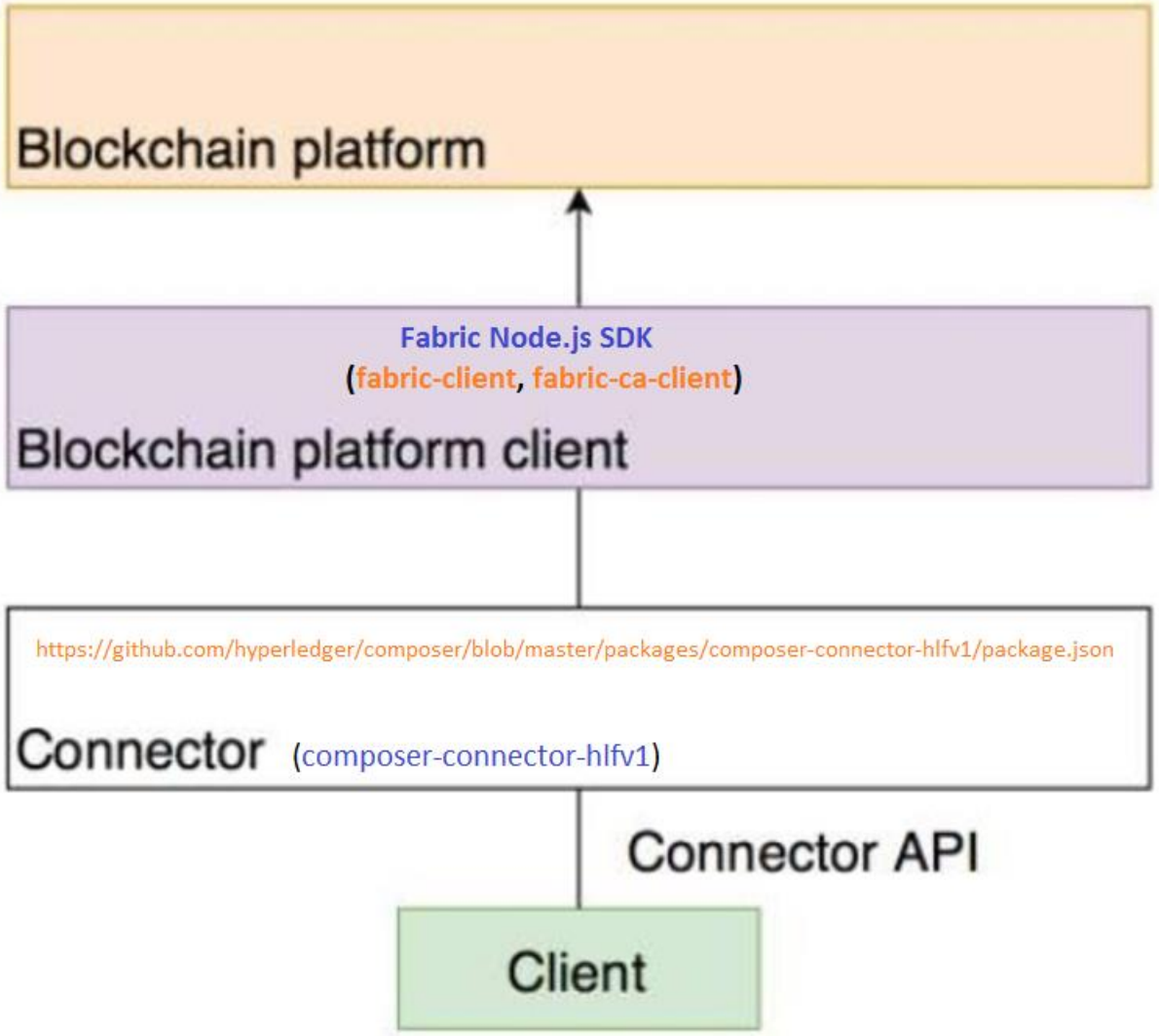


Composer Runtime Components

https://github.com/hyperledger/composer/tree/master/packages/composer-runtime

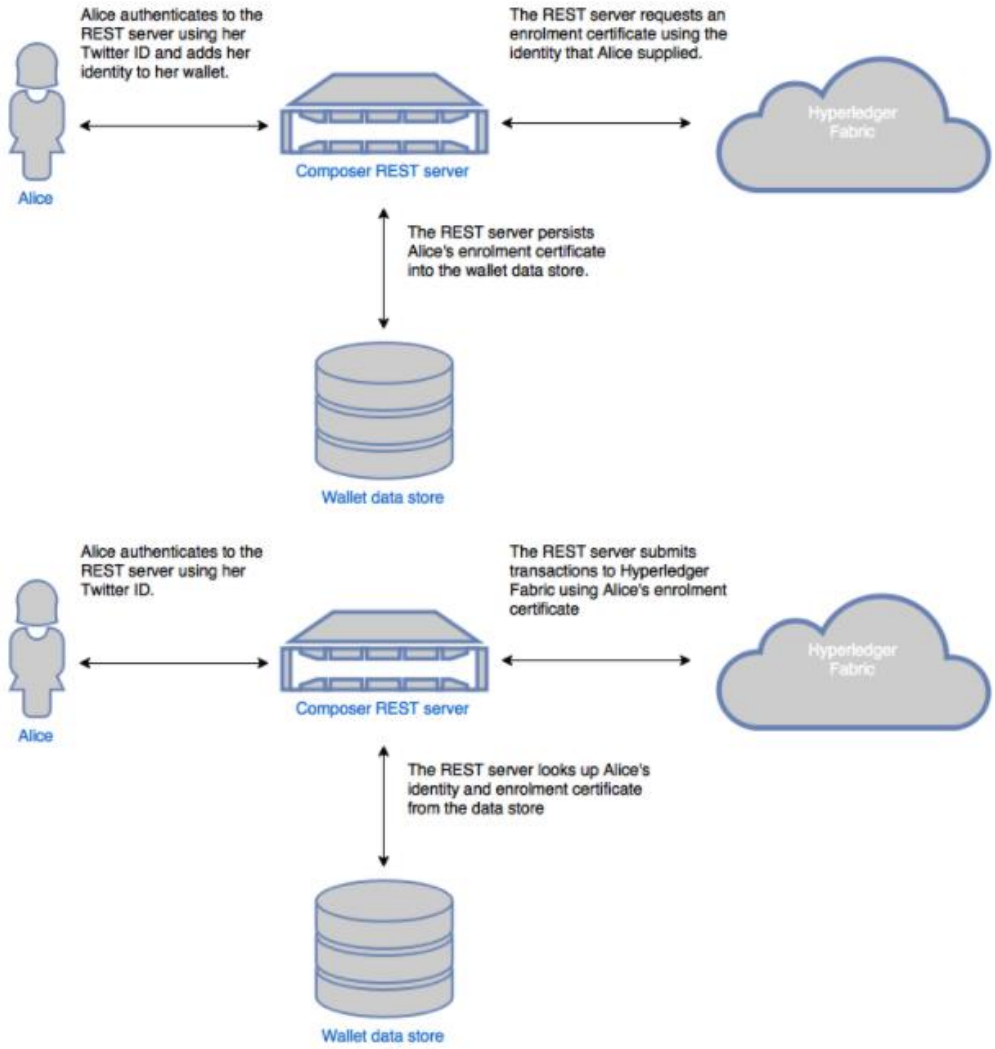


Composer Connector and Blockchain Platform

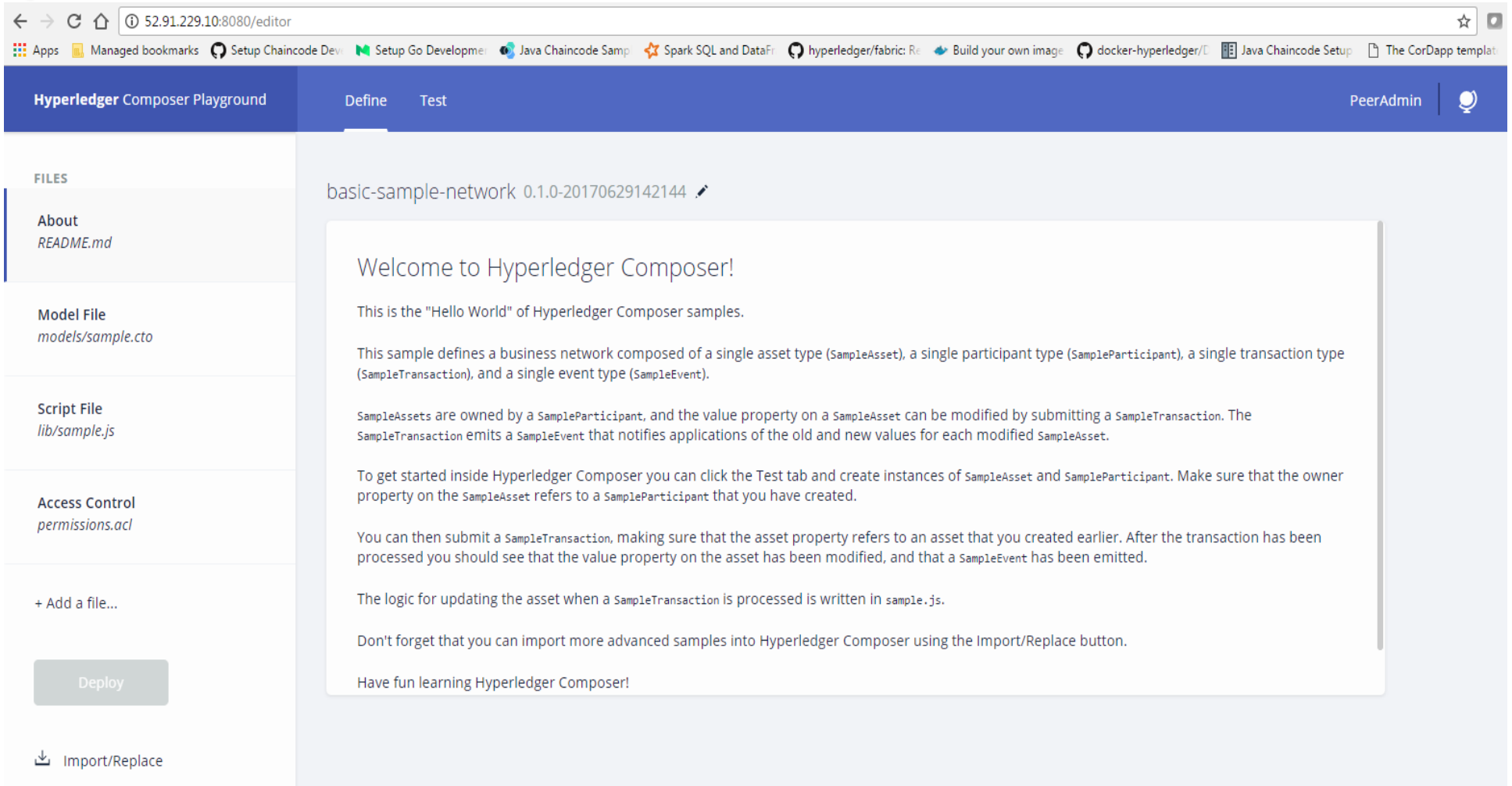


Note that there is a restriction currently for connection profiles to hlfv1. Although you can add multiple orderers to a connection profile, only the first one will receive the request. This is a limitation of the Hyperledger Fabric Node SDK.

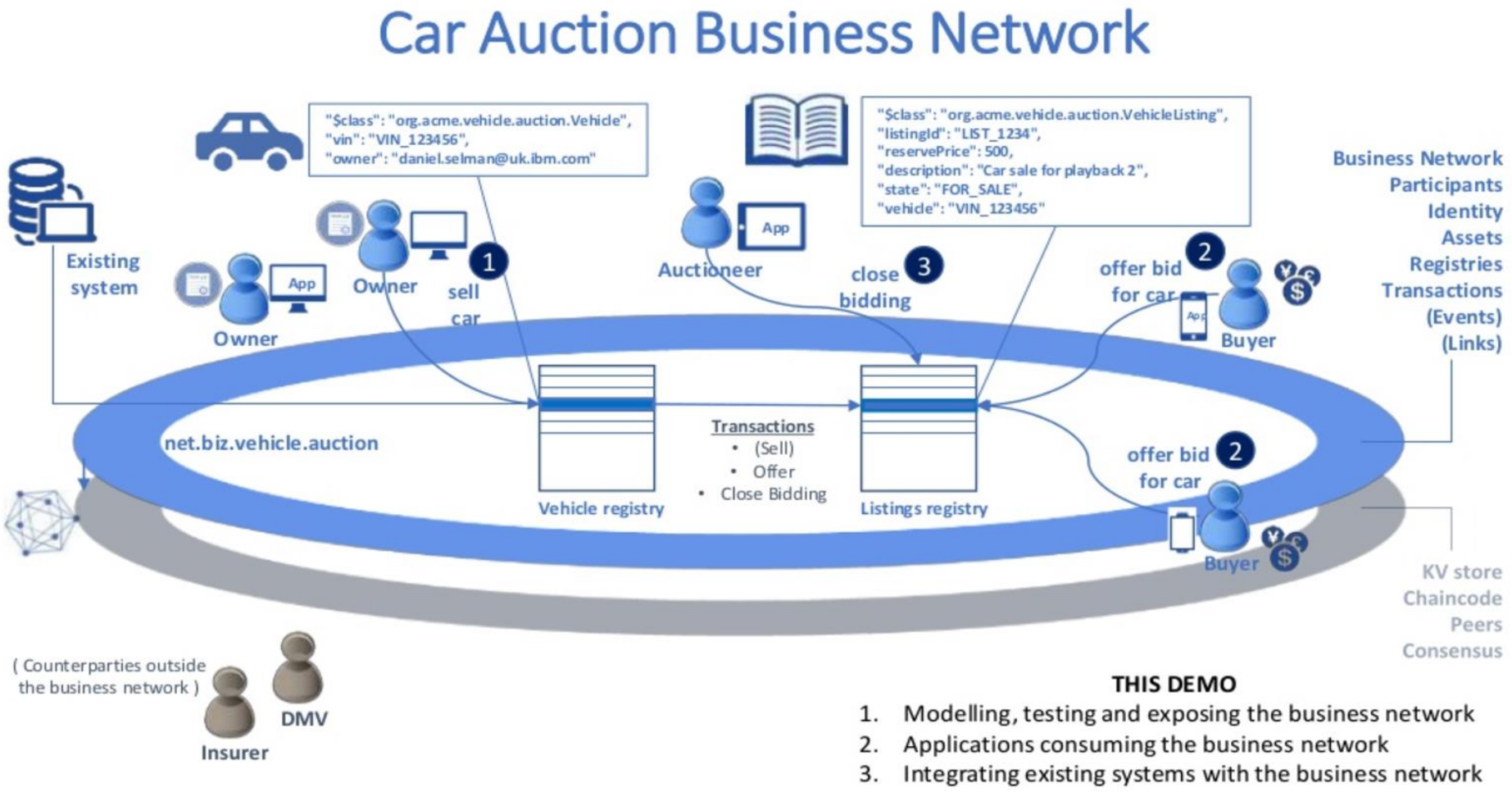
Composer REST Server – Client Authentication



Hyperledger Composer Playground Web UI

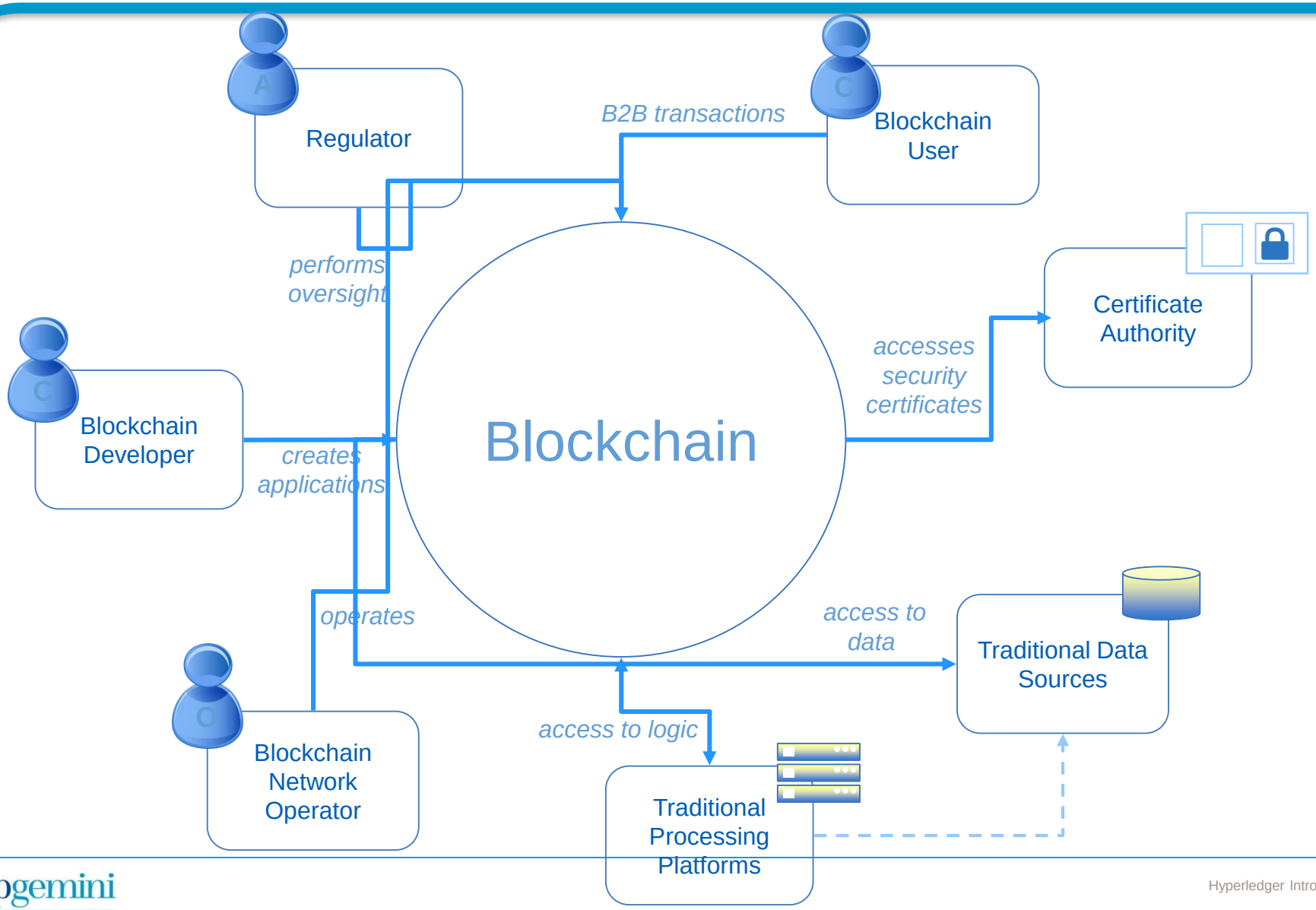


Sample Business Network

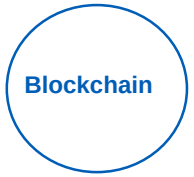


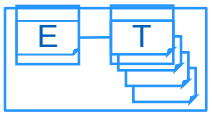
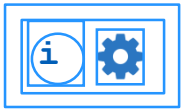
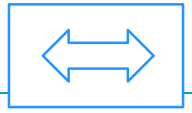
Participants in a HLF Blockchain Network

User Roles in HLF:
- Client
- Validator
- Auditor
- Peer

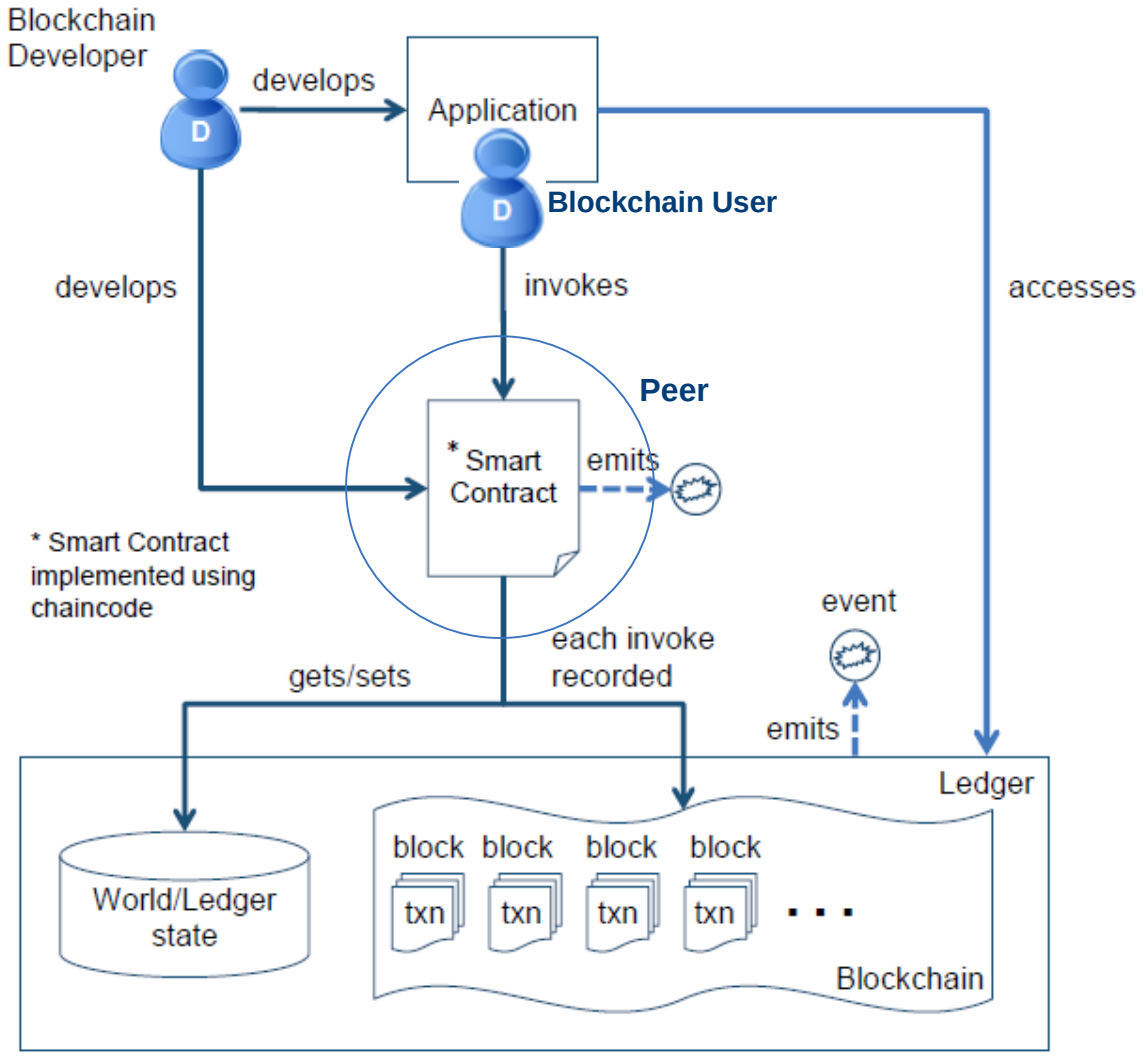


HLF Blockchain Components



Ledger		contains the current world state of the ledger and a Blockchain of transaction invocations
Smart Contract		encapsulates business network transactions in code. transaction invocations result in gets and sets of ledger state
Consensus Network		a collection of network data and processing peers forming a Blockchain network. Responsible for maintaining a consistently replicated ledger
Membership		manages identity and transaction certificates, as well as other aspects of permissioned access
Events		creates notifications of significant operations on the Blockchain (e.g. a new block), as well as notifications related to smart contracts. Does not include event distribution.
Systems Management		provides the ability to create, change and monitor Blockchain components
Wallet		securely manages a user's security credentials
Systems Integration		responsible for integrating Blockchain bi-directionally with external systems. Not part of Blockchain, but used with it.

HLF Blockchain Application Flow



References

- Fabric Protocol Specification - https://github.com/hyperledger/fabric/blob/master/docs/protocol-spec.md
- Fabric Courses - https://developer.ibm.com/courses/all-courses/blockchain-for-developers
- Fabric Documentation - http://hyperledger-fabric.readthedocs.io/en/latest

Thank you!