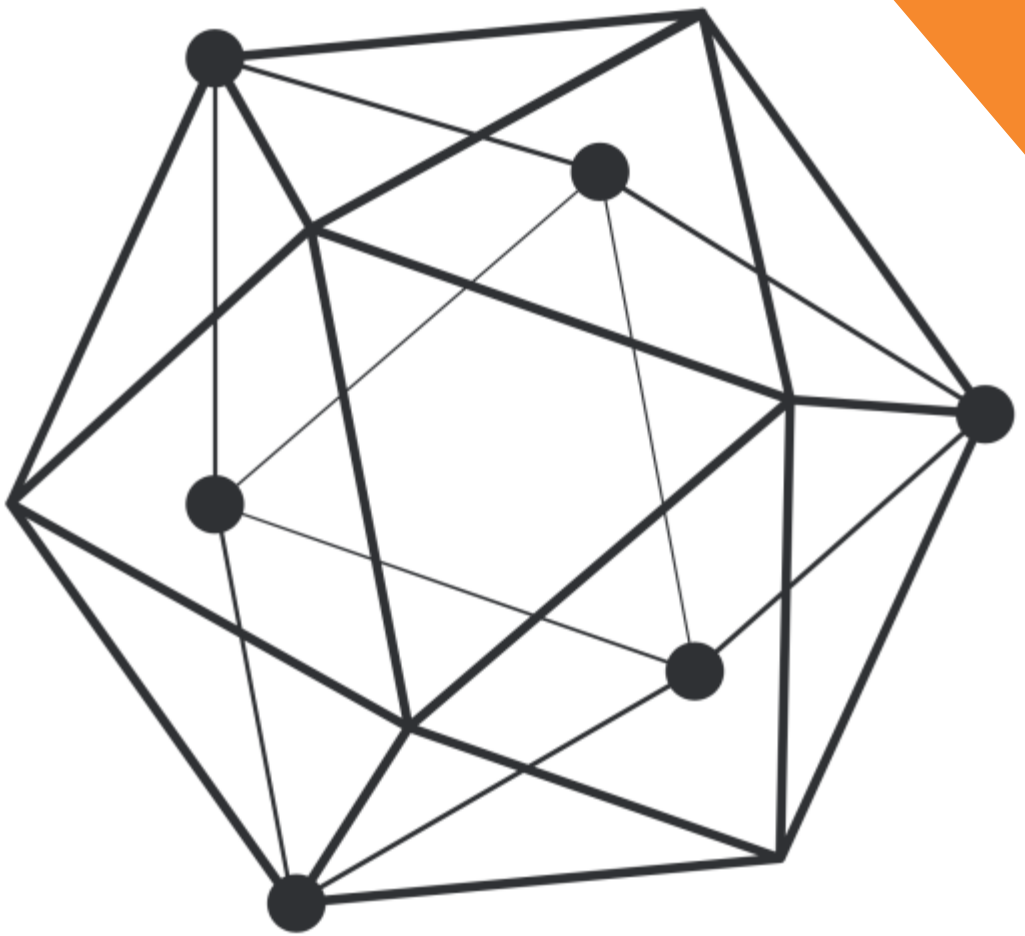
Course Outline

1. Cryptocurrency and Block chain
2. Delving into BlockChain
3. Bitcoin and Block chain
4. Bitcoin Mining
5. Ethereum
6. Setting up private Blockchain Environment using Ethereum Platform
7. Hyperledger
8. Setting up Development Program using Hyperledger composer
9. Create or Deploy our private Blockchain on Multi chain
10. Prospect of Blockchain

IntelliPaat



Hyperledger

Agenda

At the end of this session you will be able to:

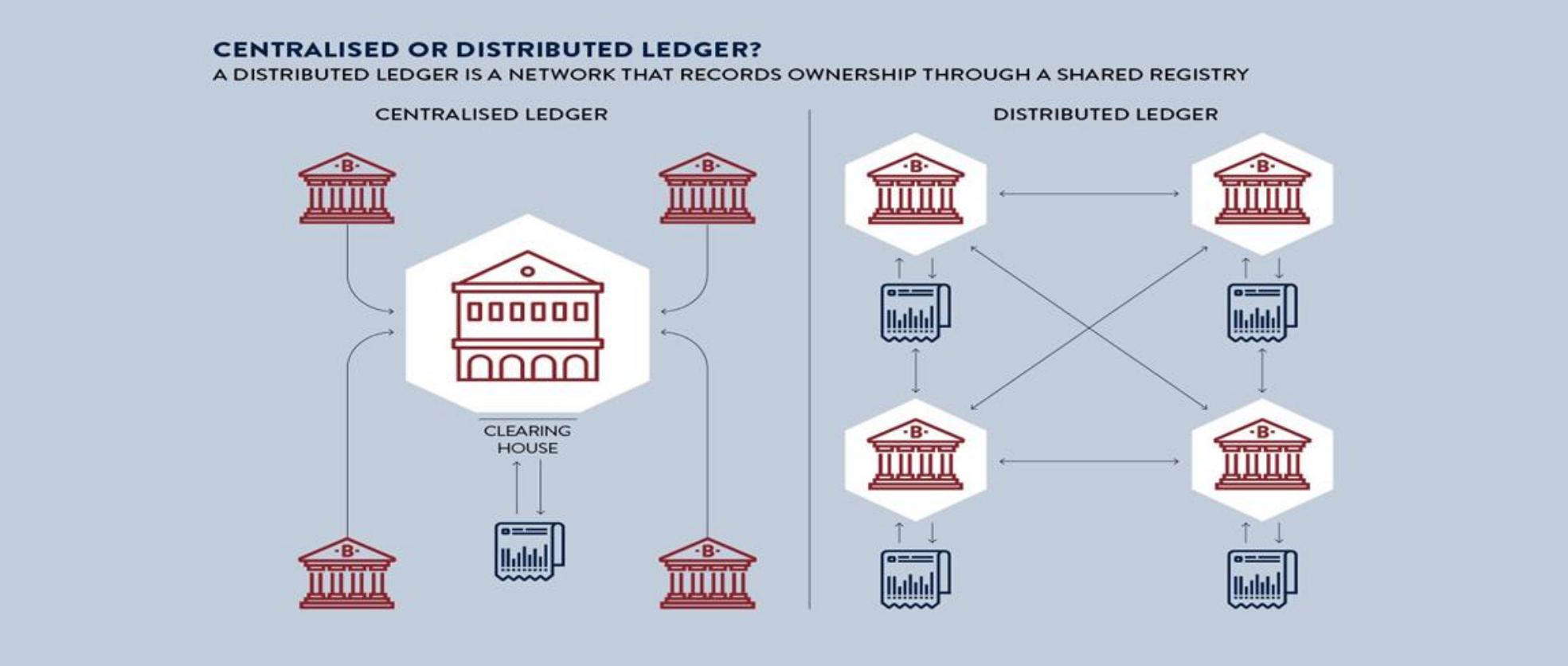
- Define Hyperledger Blockchain
- Understand Hyperledger Consensus Algorithm
- Explain Hyperledger Iroha
- Identify Hyperledger Components
- Describe Setting up Channels, Policies, and Chaincodes
- List Hyperledger Explorer Components
- Define Hyperledger Composer

Hyperledger Blockchain

Hyperledger Blockchain

Let us know more about Hyperledger Blockchain.

Business Network- Traditional v/s Blockchain



Business Network - A traditional view

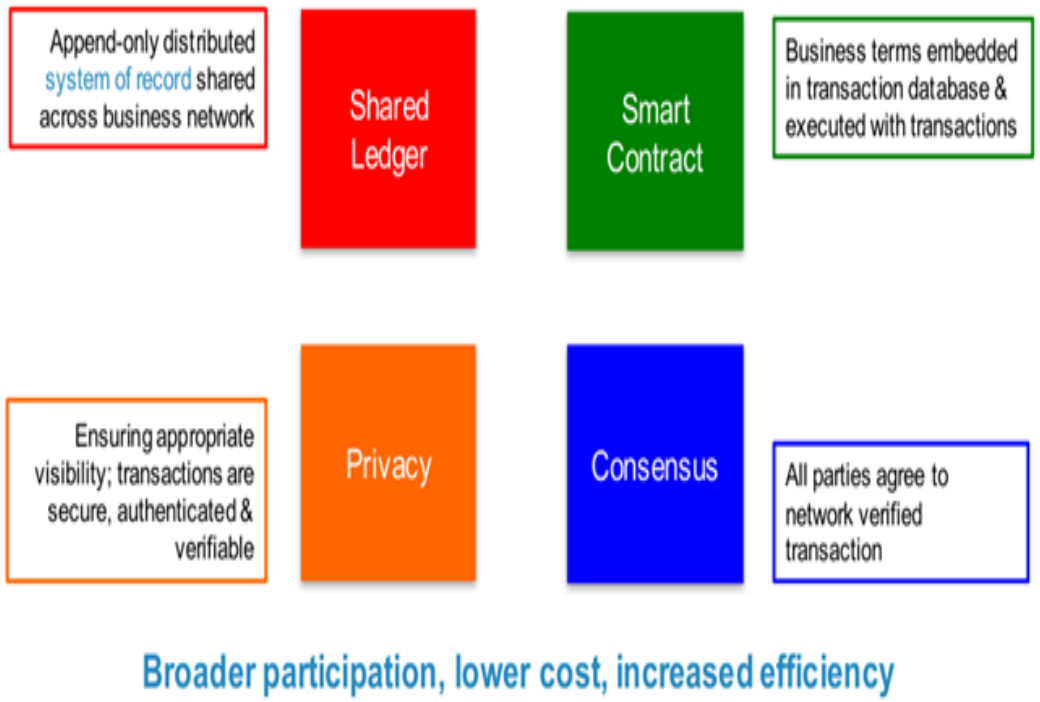
Business Network - The blockchain view

Blockchain for business?

It's the mix of a common, unalterable record that streamline business procedures and open new open doors for advancement

Consensus in blockchain is organized between individuals which wipes out expensive dangers and wasteful aspects as resources & change hands all through a business of blockchain

This enterprise-ready blockchain platform makes it easy to activate and manage a secure business network across multiple organizations



What is Hyperledger?

Hyperledger (or Hyperledger project) is a cross-industry collaborative effort to create blockchain-based open standard for distributed ledgers for globally conducted business transactions. The project has been started by Linux Foundation and backed by technological, financial, banking and supply chain companies worldwide.

The project aims to create an **open-standard, public, decentralised public ledger based on blockchain technology** to advance worldwide business transaction processing in terms of cost-effectiveness, speed and traceability.

Founding members of the initiative represent a diverse group of stakeholders:

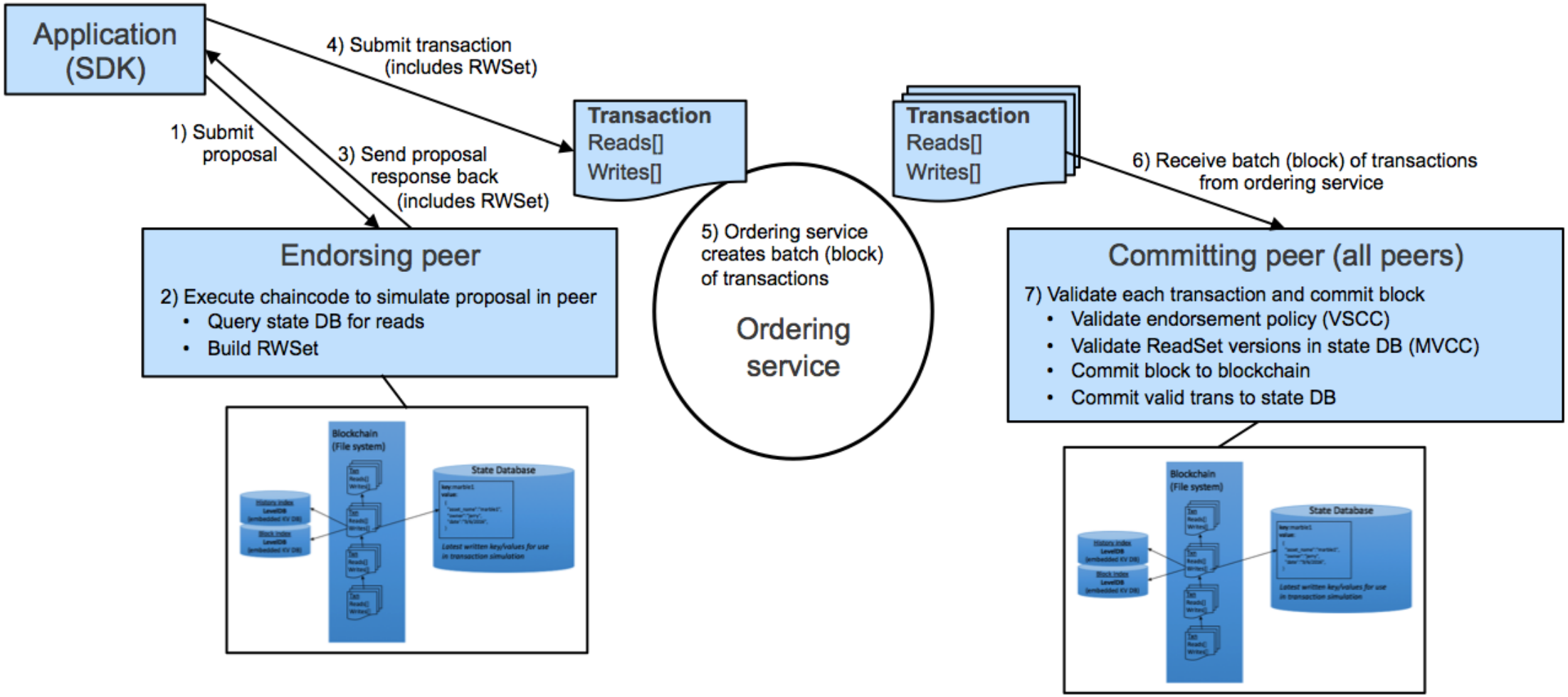
- ABN AMRO
- Accenture
- ANZ Bank
- Blockchain Ltd
- BNY Mellon
- Calastone
- Cisco
- CME Group
- ConsenSys
- Credits
- The Depository Trust & Clearing Corporation (DTCC)
- CLS
- Deutsche Börse Group
- Digital Asset Holdings
- Fujitsu Limited
- Guardtime
- Hitachi
- Wells Fargo
- IBM
- Intel
- IntellectEU
- J.P. Morgan
- Red Hat
- SWIFT
- NEC
- NTT DATA
- R3
- State Street
- Symbiont
- Vmware

Problems with existing Blockchains

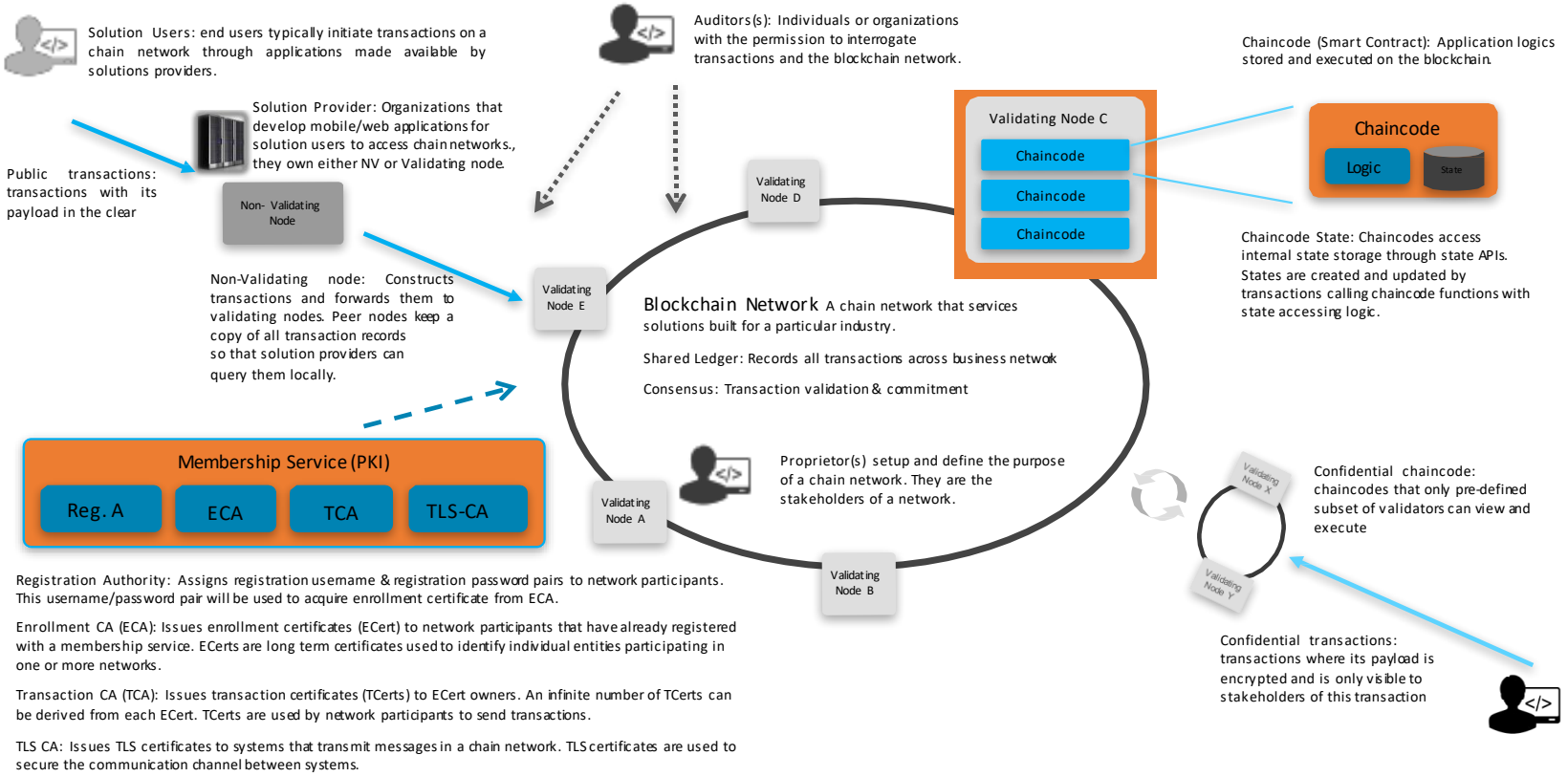
Characteristic	Ethereum	Hyperledger Fabric	R3 Corda
Description of platform	– Generic blockchain platform	– Modular blockchain platform	– Specialized distrib-uted ledger platform for financial industry
Governance	– Ethereum developers	– Linux Foundation	– R3
Mode of operation	– Permissionless, public or private[4]	– Permissioned, private	– Permissioned, private
Consensus	– Mining based on proof-of-work (PoW) – Ledger level	– Broad understand-ing of consensus that allows multiple approaches – Transaction level	– Specific understand-ing of consensus (i.e., notary nodes) – Transaction level
Smart contracts	– Smart contract code (e.g., Solidity)	– Smart contract code (e.g., Go, Java)	– Smart contract code (e.g., Kotlin, Java) – Smart legal contract (legal prose)
Currency	– Ether – Tokens via smart contract	– None – Currency and tokens via chaincode	– None

Goals of hyperledger

Hyperledger - How it works

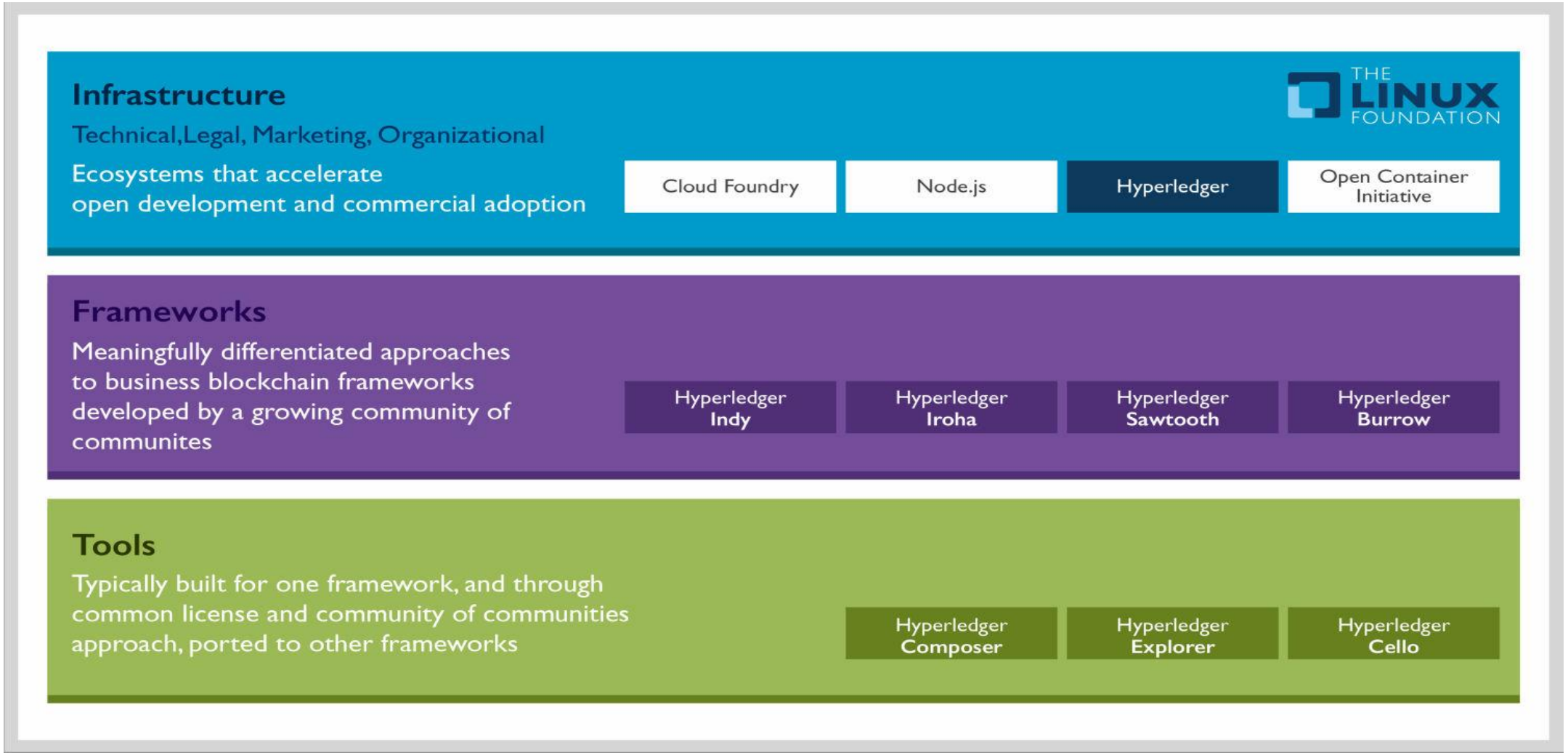


Hyperledger - How it works



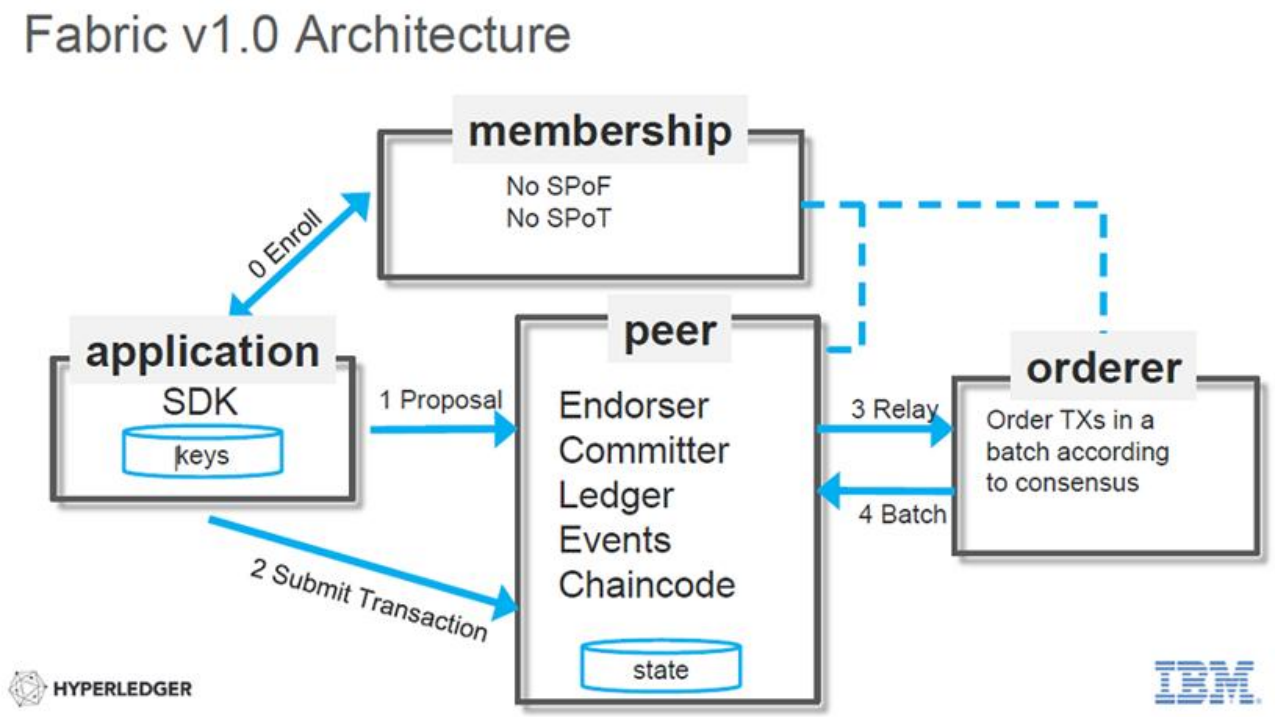
The Blockchain Network

Hyperledger Frameworks and tools



Hyperledger Architecture

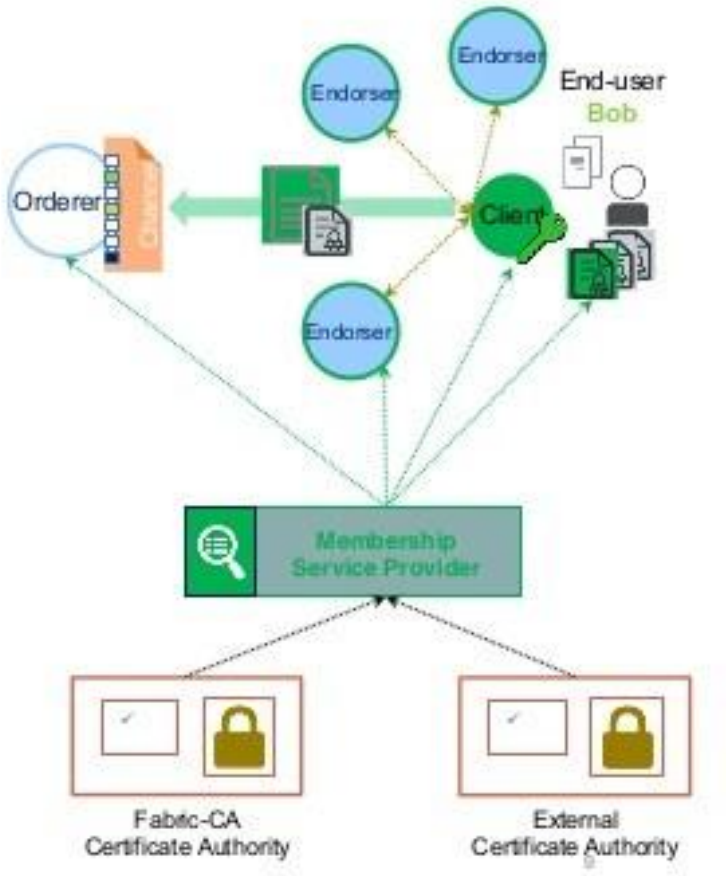
- Hyperledger reference architecture is aligned in three categories: Membership, Blockchain and Chaincode
- These categories are a logical structure, not a physical depiction of partitioning of components into separate processes, address spaces or (virtual) machines

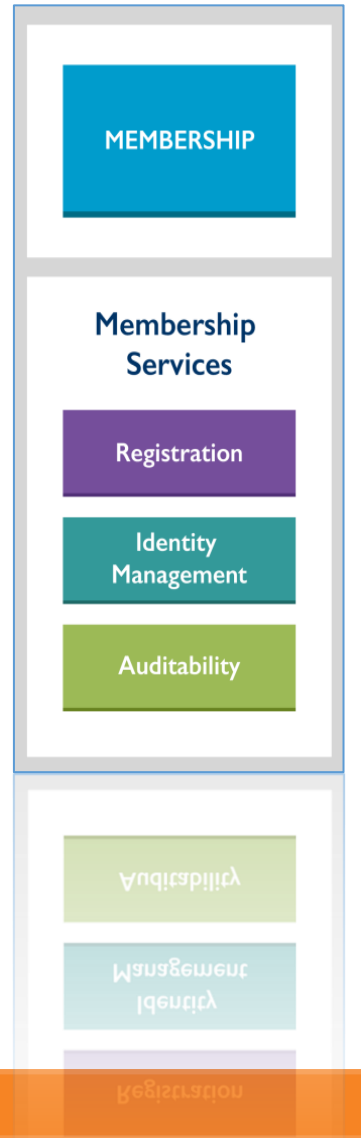


Membership

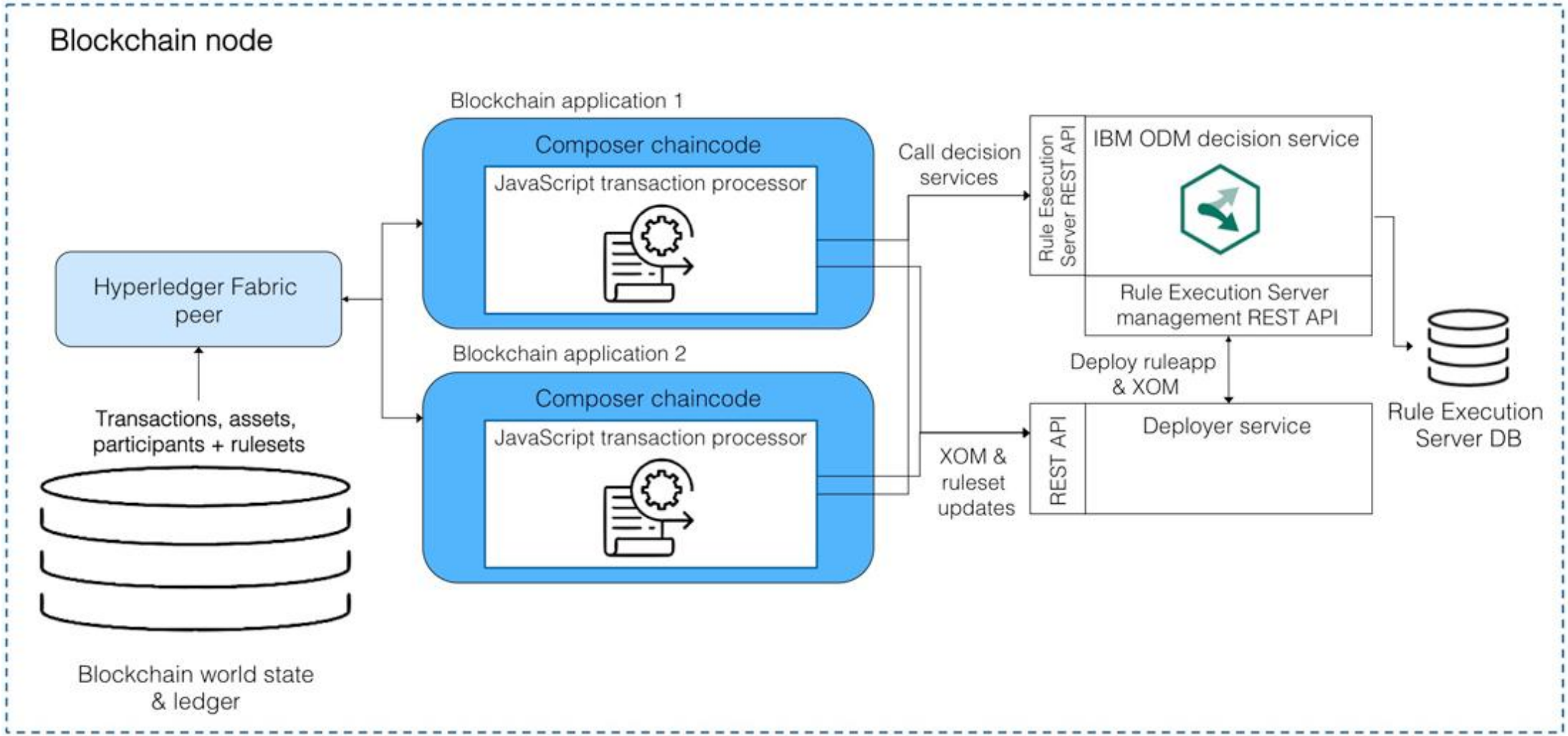
Membership Service Provider

- An abstraction of identity provider
  - <MSP.id, MSP.sign, MSP.verify, MSP.validateid, MSP.admin>
  - govern application, endorser and orderer identities
- Used as building blocks for access control frameworks
  - at the system level (read/write access on system controls, and channel creation)
  - at the channel level (read/write access),
  - at the chaincode level (invocation access)
- Represent a consortium or a member





Blockchain



Chaincode

Chaincode is a piece of code that lets you interact with a network's shared ledger

Whenever you invoke a transaction on the network, you are effectively calling a function in a piece of chaincode that reads and writes values to the ledger

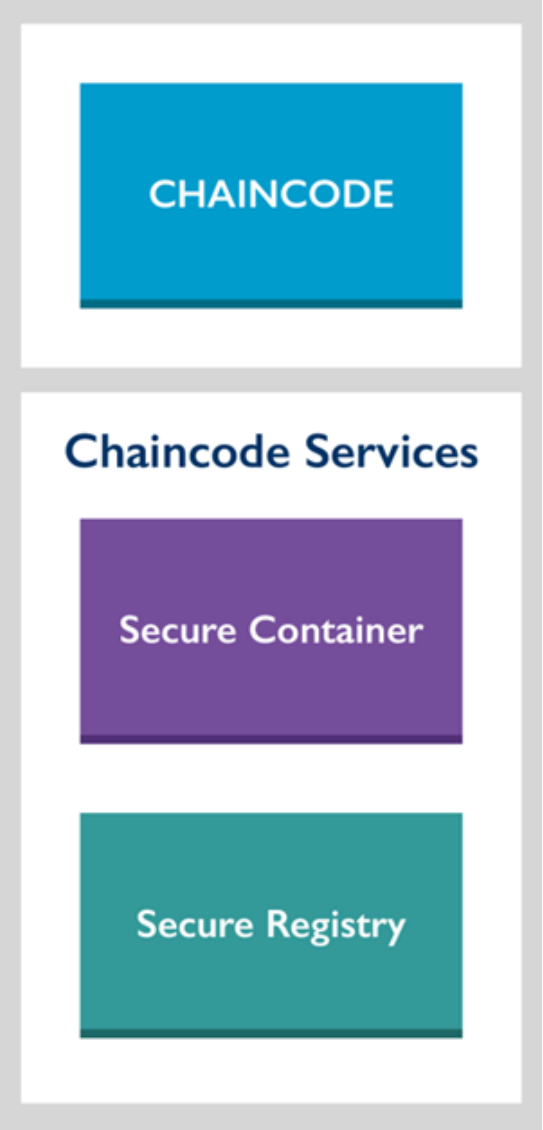
Chaincode services uses Docker to host the chaincode

Docker provides a secured, lightweight method to sandbox chaincode execution

The environment is a secured container, along with a set of signed base images containing secure OS and chaincode language, runtime and SDK images for Golang

Additional programming languages can be enabled, if required

Secure Registry Servicesenables Secured Docker Registry of base Hyperledger images and custom images containing chaincodes



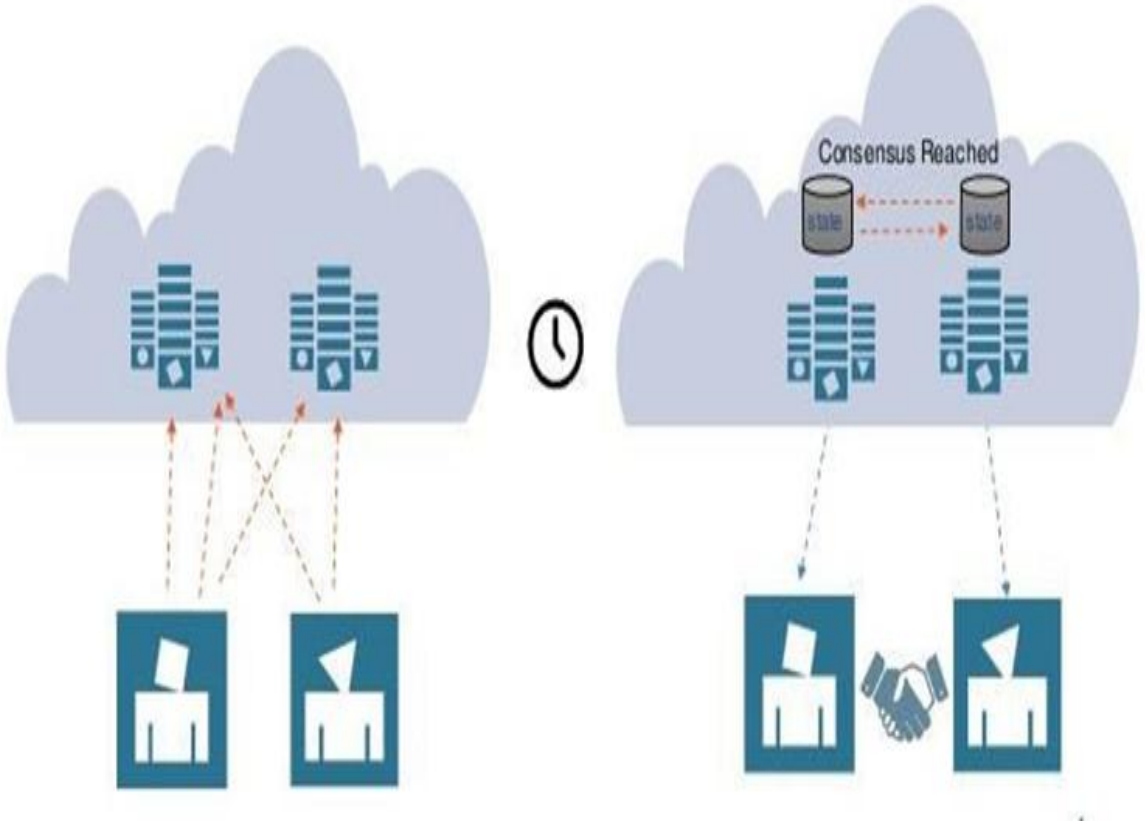
Hyperledger Consensus Algorithm

Hyperledger Consensus Algorithm

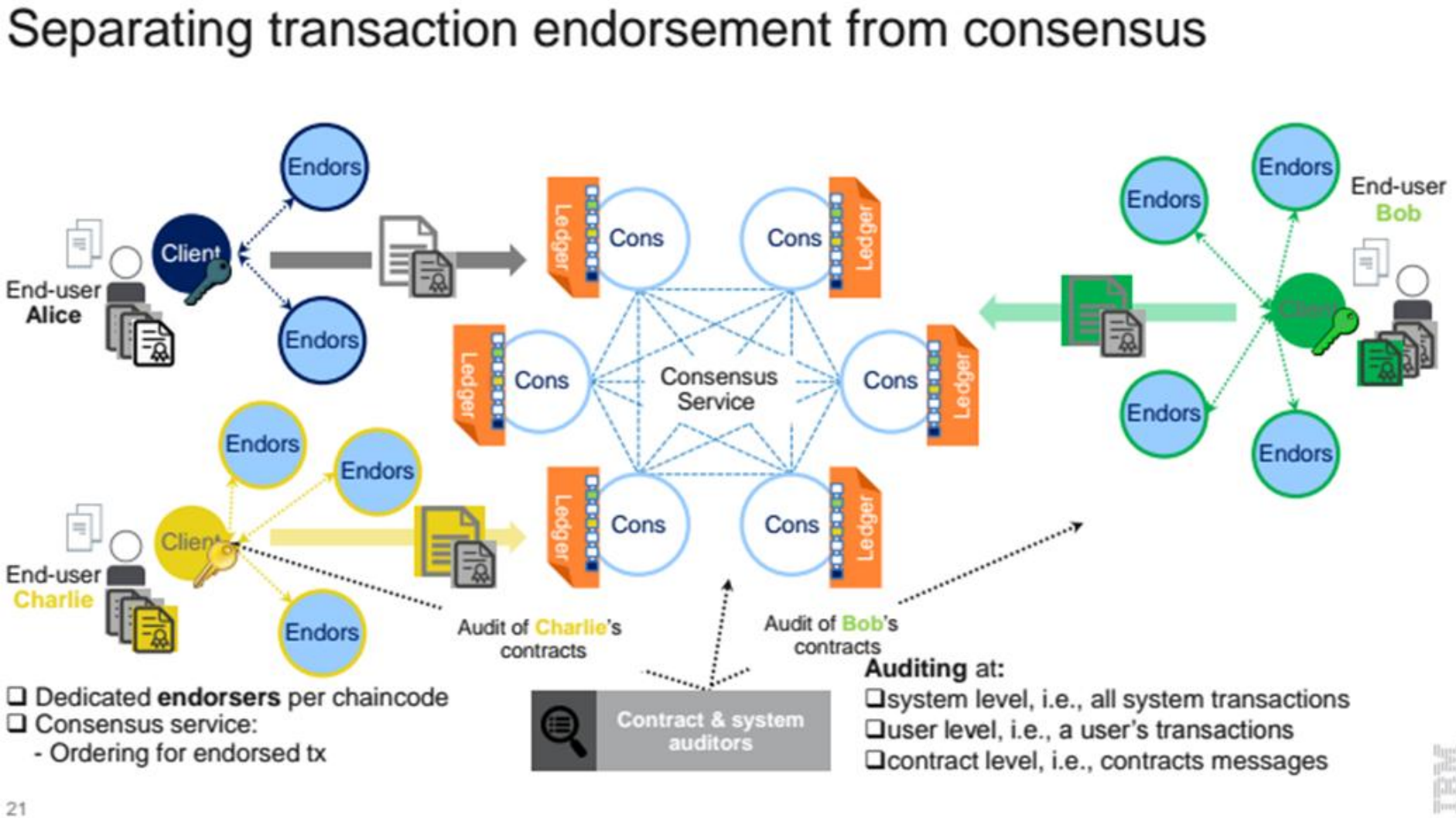
Let us learn about different consensus algorithms for Hyperledger.

Hyperledger Consensus

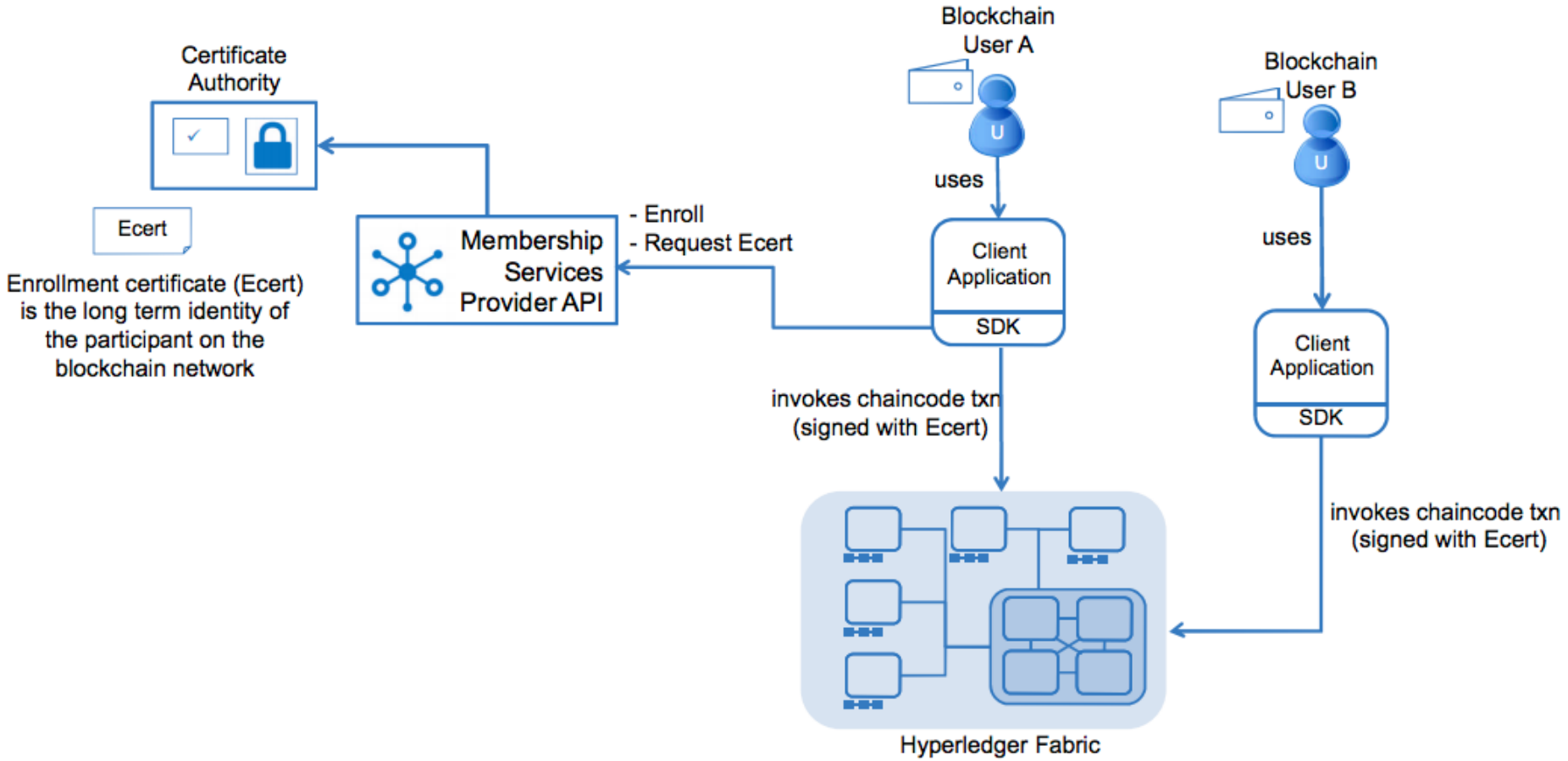
- Because different industries and regions may run their own networks, different networks might need to deploy different consensus algorithms to fit their usage scenarios
- Consensus algorithms under the Hyperledger protocol must be pluggable, allowing users to select the consensus algorithm of their choice during deployment.
- The Hyperledger protocol will provide an implementation of Byzantine Fault Tolerance (BFT) in its initial release, using the PBFT protocol



Generalized Hyperledger Consensus process flow



Application Programming Interface



Application Programming Interface

The API has the following categories:

Identity: Enrollment to get certificates or revoke a certificate

Address: Target and source of a transaction

Storage: External store for files or documents

Event: Sub/pub events on blockchain

Transaction: Unit of execution on the ledger

Chaincode: Program running on the ledger

Blockchain: Content of the ledger

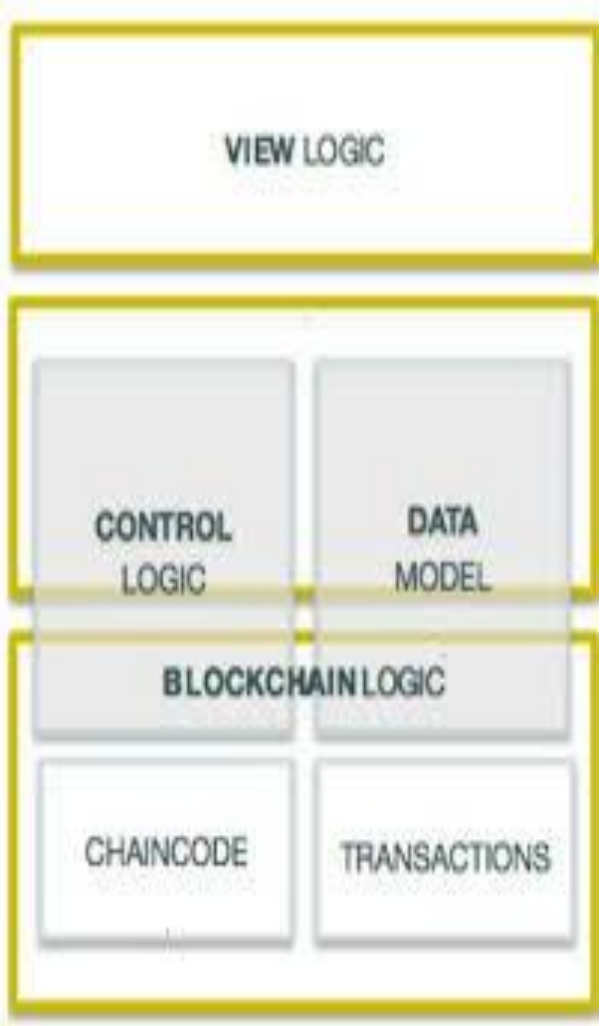
Network: Information about the blockchain network

Application Model

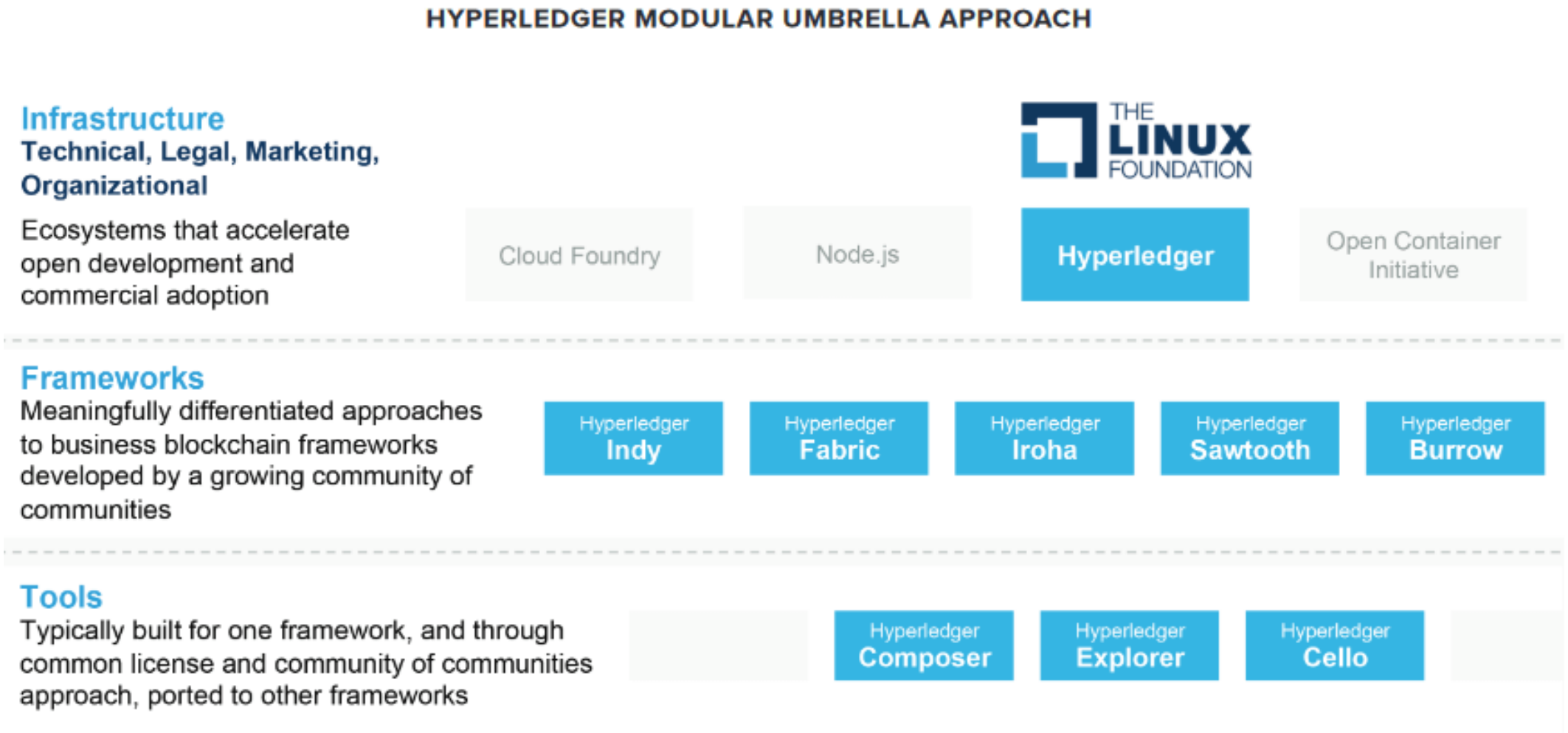
An Hyperledger application follows a MVC-B architecture (Model View Control Blockchain)

- VIEW LOGIC - Mobile or Web UI interacting with control logic
- CONTROL LOGIC-Coordinates between UI, Data Model and Hyperledger
- APIs to drive transitions and chaincode
- DATA MODEL- Application Data Model – manages offchain data, including documents and large files
- Blockchain Logic- Are extensions of the Controller Logic and Data Model into the Blockchain realm. Controller logic is enhanced by chaincode, and the data model is enhanced with transactions on the blockchain.

For example, a PaaS application using Node.js might have a Web frontend user interface or a native mobile app with a backend model provided by a data management service. The control logic may interact with one or more chaincodes to process transactions on the blockchain



Hyperledger Projects and Tools



Hyperledger Iroha

Hyperledger Iroha

Hyperledger Iroha features a simple construction.
Let’s have a look at the Hyperledger Iroha .

Hyperledger Iroha

The name Iroha is "the basics" , "ABCs" or "the most basic element of all"

- Hyperledger Iroha features a simple construction; modern, domain-driven C++ design, emphasis on mobile application development and a new, chain-based Byzantine Fault Tolerant consensus algorithm, called Sumeragi.

- Iroha is a distributed ledger project that was designed to be simple and easy to incorporate into infra- structural projects requiring distributed ledger technology

The goals of Iroha are to:

- Provide an environment for C++ developers to contribute to Hyperledger
- Provide infrastructure for mobile and web application support
- Provide a framework to experiment with new APIs and consensus algorithms that could potentially be incorporated into Fabric in the future.

Why hyperledger Iroha?

> Currently, the Hyperledger Project lacks an infrastructure project written in C++, thus limiting the potential developers who can contribute

Iroha most important features:

- Creation and management of custom complex assets, such as currencies or indivisible rights, serial numbers, patents, etc.
- Management of user accounts
- Taxonomy of accounts based on domains — or sub-ledgers in the system
- The system of rights and verification of user permissions for the execution of transactions and queries in the system
- Validation of business rules for transactions and queries in the systemNew, chain-based Byzantine fault tolerant consensus algorithm, called Sumeragi

Sawtooth Lake

Sawtooth makes it easy to develop and deploy an application by providing a clear separation between the application level and the core system level. Sawtooth provides smart contract abstraction that allows application developers to write contract logic in a language of their choice.

It is written in Python and designed for use cases in many fields from IoT to Financials

Sawtooth is built to solve the challenges of permissioned (private) networks. Clusters of Sawtooth nodes can be easily deployed with separate permissioning.

Pluggable consensus algorithms

Most blockchains have three elements: A shared record of the current state of the blockchain, A way of inputting new data A way of agreeing on that data

Sawtooth Lake merges the first two into a signal process they call a transaction family

Why Sawtooth Lake different?

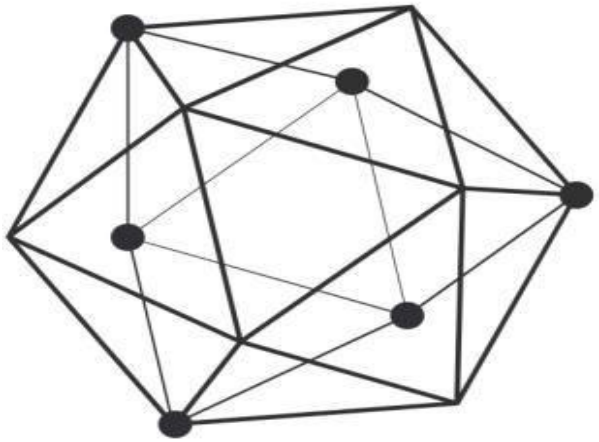
Keeps distributed ledgers distributed.

Sawtooth users are able to define their own custom “transaction family” with data models and transaction languages tailored to their use case

Hyperledger Sawtooth supports creating and broadcasting events. This allows applications to:

- Subscribe to events that occur related to the blockchain, such as a new block being committed or switching to a new fork. Subscribe to application specific events defined by a transaction family. Relay information about the execution of a transaction back to clients without storing that data in state.

Tools in hyperledger



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, transactuons and asociated data, network information, chain codes and transaction families as well as any other relevant information stored in the ledger

Hyperledger Cello

> Hyperledger Cello is a blockchain provision and operation system, which helps manage blockchain networks

Using Cello, everyone can easily:

- Hyperledger Cello is a blockchain module toolkit and one of the Hyperledger projects hosted by The Linux Foundation.
- 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.
- Maintain a pool of running blockchain networks on top of baremetals, Virtual Clouds (e.g., virtual machines, vsphere Clouds), Container clusters (e.g., Docker, Swarm, Kubernetes).
- It helps to Check the system status, adjust the chain numbers, scale resources... through dashboards

Main Features of Cello

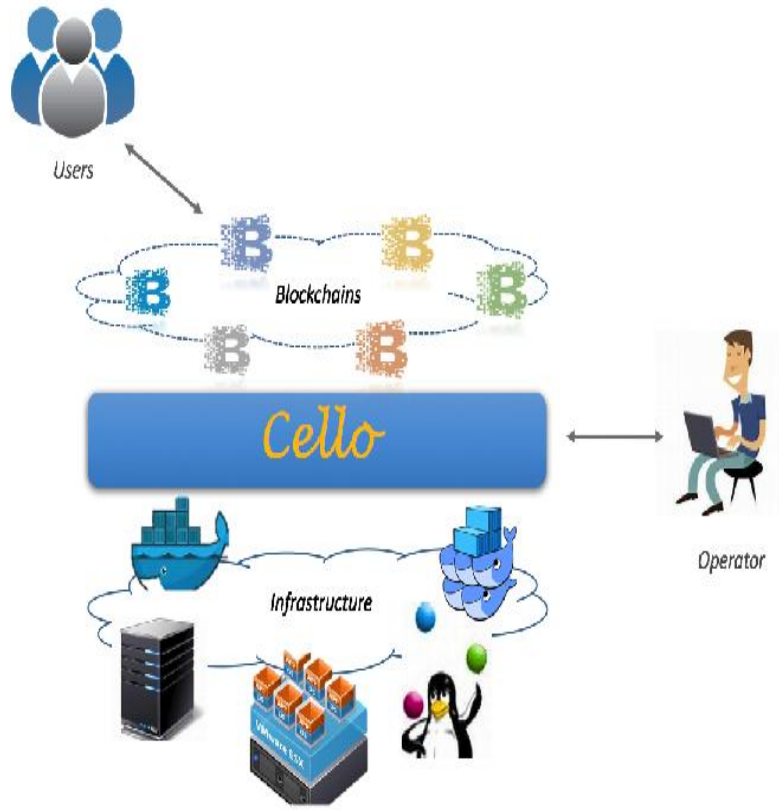
Manage the lifecycle of blockchains, e.g., create/start/stop/delete/keep health automatically

Support customized (e.g., size, consensus) blockchains request, currently we mainly support Hyperledger fabric

Support native Docker host, swarm or Kubernetes as the worker nodes. More supports on the way

Support heterogeneous architecture, e.g., X86, POWER, and Z, from bare-metal servers to virtual machines

Extend with monitor, log, health and analytics features by employing additional components



Hyperledger Explorer

Hyperledger Blockchain Explorer is a blockchain module designed to create a user-friendly Web application

It was initially contributed by IBM .

Designed to create a user-friendly Web application, Hyperledger Blockchain Explorer can view, invoke, deploy or query blocks, transactions and associated data, network information (name, status, list of nodes), chain codes and transaction families, as well as any other relevant information stored in the ledger.

Hyperledger fabric

- Hyperledger's first incubation project, Fabric, is a permissioned blockchain platform
- Hyperledger fabric was implemented in Go programming language
- It is made for enabling consortium blockchains with different degrees of permissions
- Hyperledger Fabric is a platform for distributed ledger solutions underpinned by a modular architecture delivering high degrees of confidentiality, resiliency, flexibility and scalability

Key Features

- Channels for sharing confidential information
- Ordering Service delivers transactions consistently to peers in the network
- Endorsement policies for transactions
- CouchDB world state supports wide range of queries
- Bring-your-own Membership Service Provider (MSP)

Hyperledger Fabric Capabilities

Identity management

Privacy and confidentiality

Efficient processing

Chaincode functionality

Modular design

Fabric issues transactions with derived certificates that are unlinkable to the owning participant, thereby offering anonymity on the network.

The content of each transaction is encrypted to ensure only the intended participants can see the content

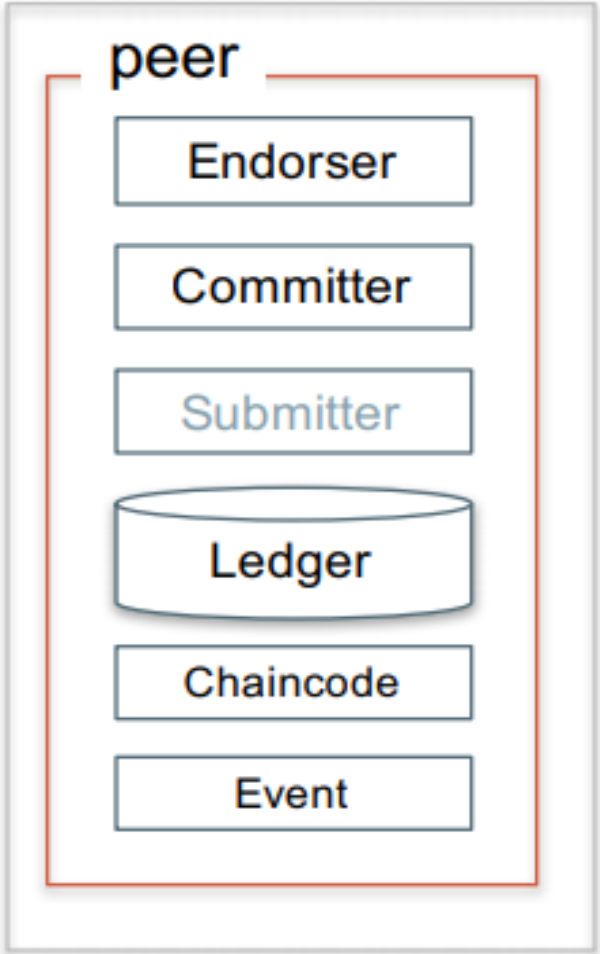
Hyperledger Components

Hyperledger Components

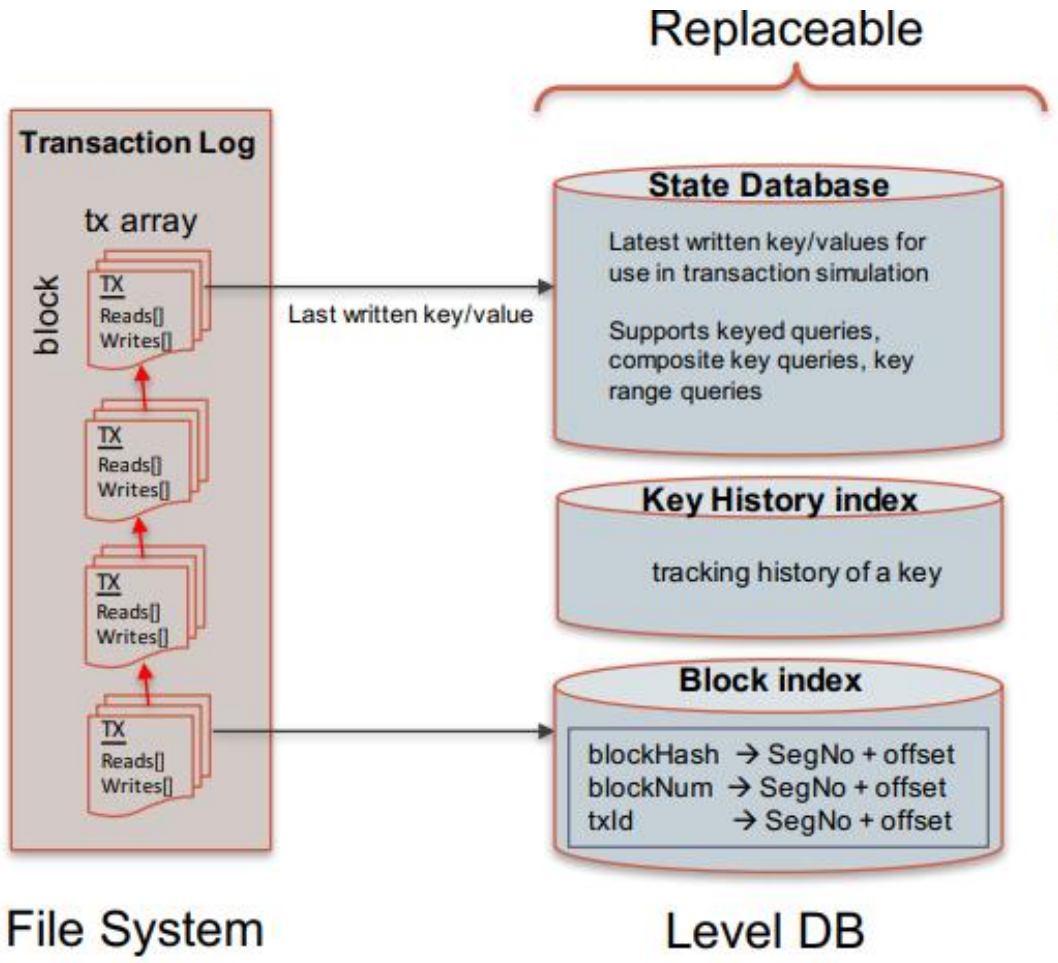
Let’s have a look at the Hyperledger Components.

Peer

A Peer is a node on the network maintaining state of the
ledger and managing chaincodes
• Any number of Peers may participate in a network
• A Peer can be an endorser, committer and/or submitter
(submitter has not been implemented). An endorser is always
a committer
– An endorser executes and endorses transactions
– A committer verifies endorsements and validates transaction results
• A Peer manages event hub and deliver events to the subscribers
• Peers form a peer-to-peer gossip network



Ledger



CouchDB (external option) supports keyed queries, composite key queries, key range queries, plus full data rich queries (beta in 1.0)

Channel

A data partitioning mechanism to control transaction visibility only to stakeholders

Consensus takes place within a channel by members of the channel
– Other members on the network are not allowed to access the channel and will not see transactions on the channel

A chaincode may be deployed on multiple channels, each instance is isolated within its channel
– A chaincode may query another chaincode in other channel (ACL applied)

Membership Service Provider

- An abstraction of identity provider

– <MSP.id, MSP.sign, MSP.verify, MSP.validateid, MSP.admin>

– govern application, endorser and orderer identities

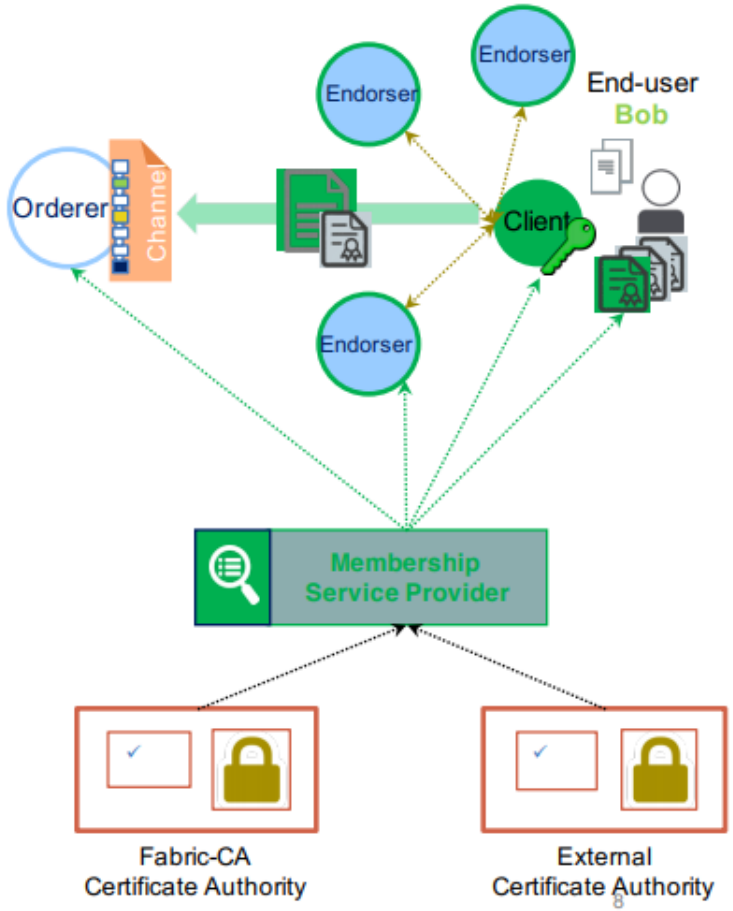
- Used as building blocks for access control frameworks

– at the system level (read/write access on system controls, and channel creation)

– at the channel level (read/write access),

– at the chaincode level (invocation access)

- Represent a consortium or a member



Crypto Service Provider

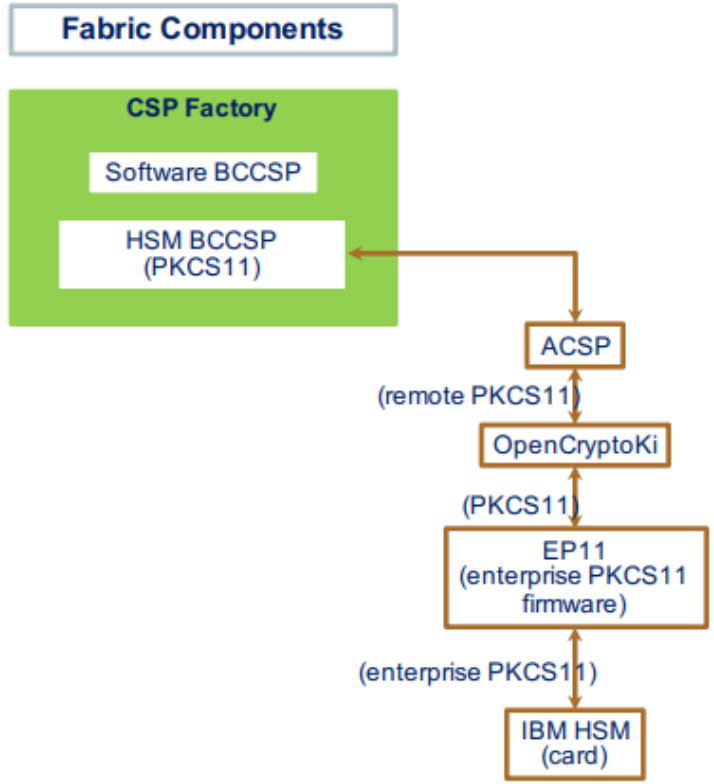
- CSP abstracts crypto standards (software and hardware) to enable plugging in different implementation

– Alternate implementations of crypto interface can be used within the Fabric code, without modifying the core

- Support for Multiple CSPs

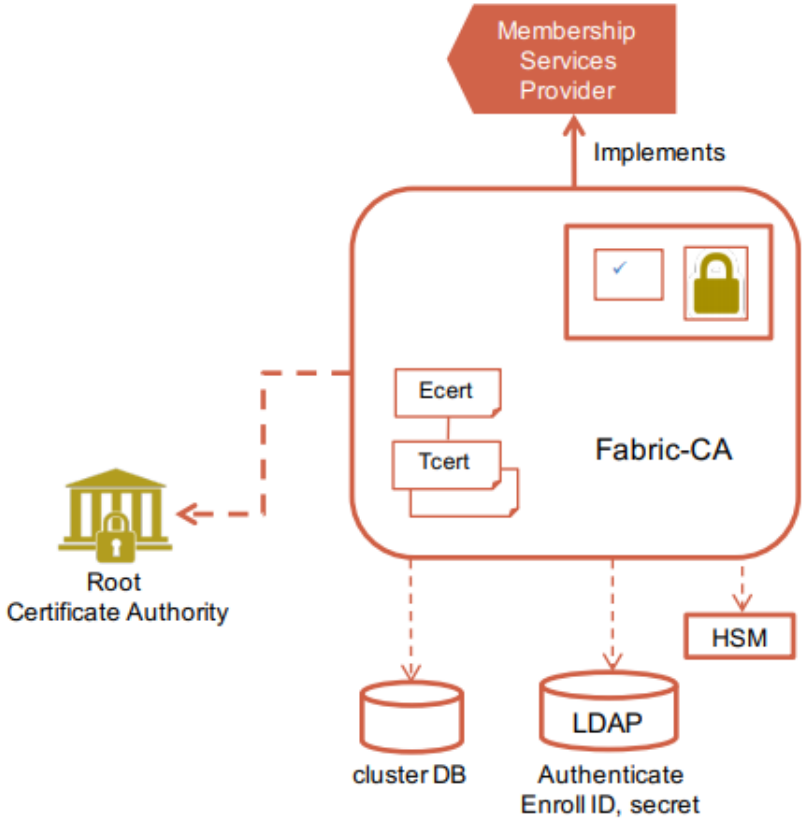
– Easy addition of more types of CSPs, e.g., of different HSM types

– Enable the use of different CSP on different system components transparently



Fabric-CA

- Default implementation of the Membership Services Provider Interface.
- Issues Ecerts (long-term identity) and Tcerts (disposable certificate)
- Supports clustering for HA characteristics
- Supports LDAP for user authentication
- Supports HSM



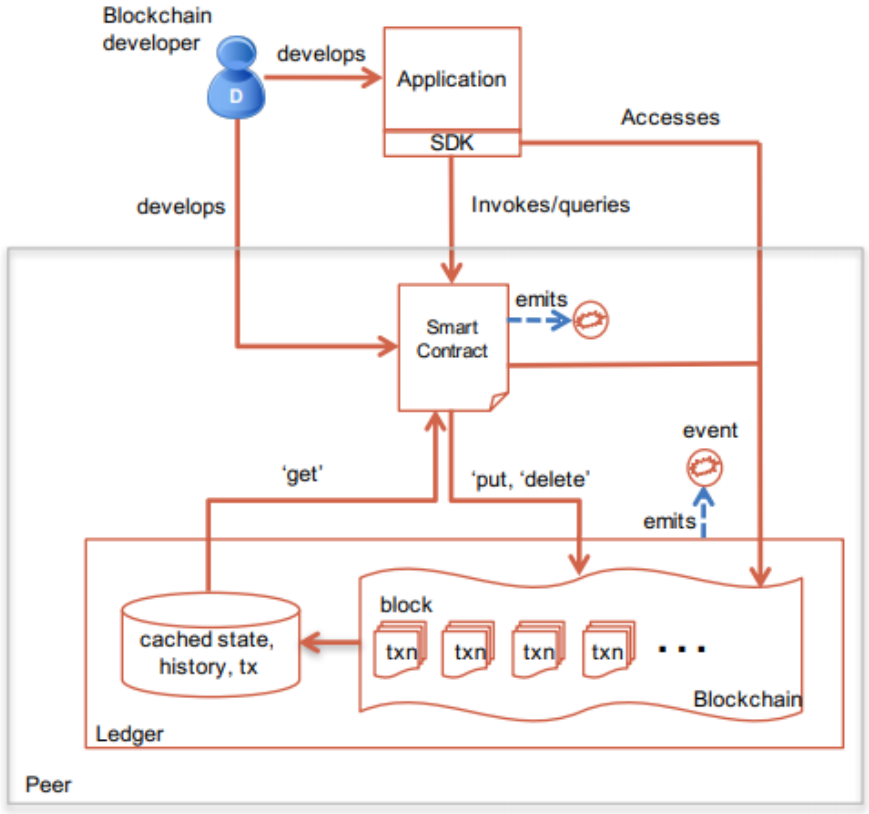
Overview of Application Flow

- Developers create applicationand smart contracts (chaincodes)

– Chaincodes are deployed on the network and control the state of the ledger

– Application handles user interface and submits transactions to the network which call chaincodes

- Network emits events on block of transactions allowing applications to integrate with other systems



Bootstrapping a Network

- Decide on members (MSPs) controlling the ordering service

– Set up MSP configuration for each member (root certs, signing certs, key, admins)

– Set up policies governing the network (who has privilege to modify config and create channels)

– Start up orders with the configuration

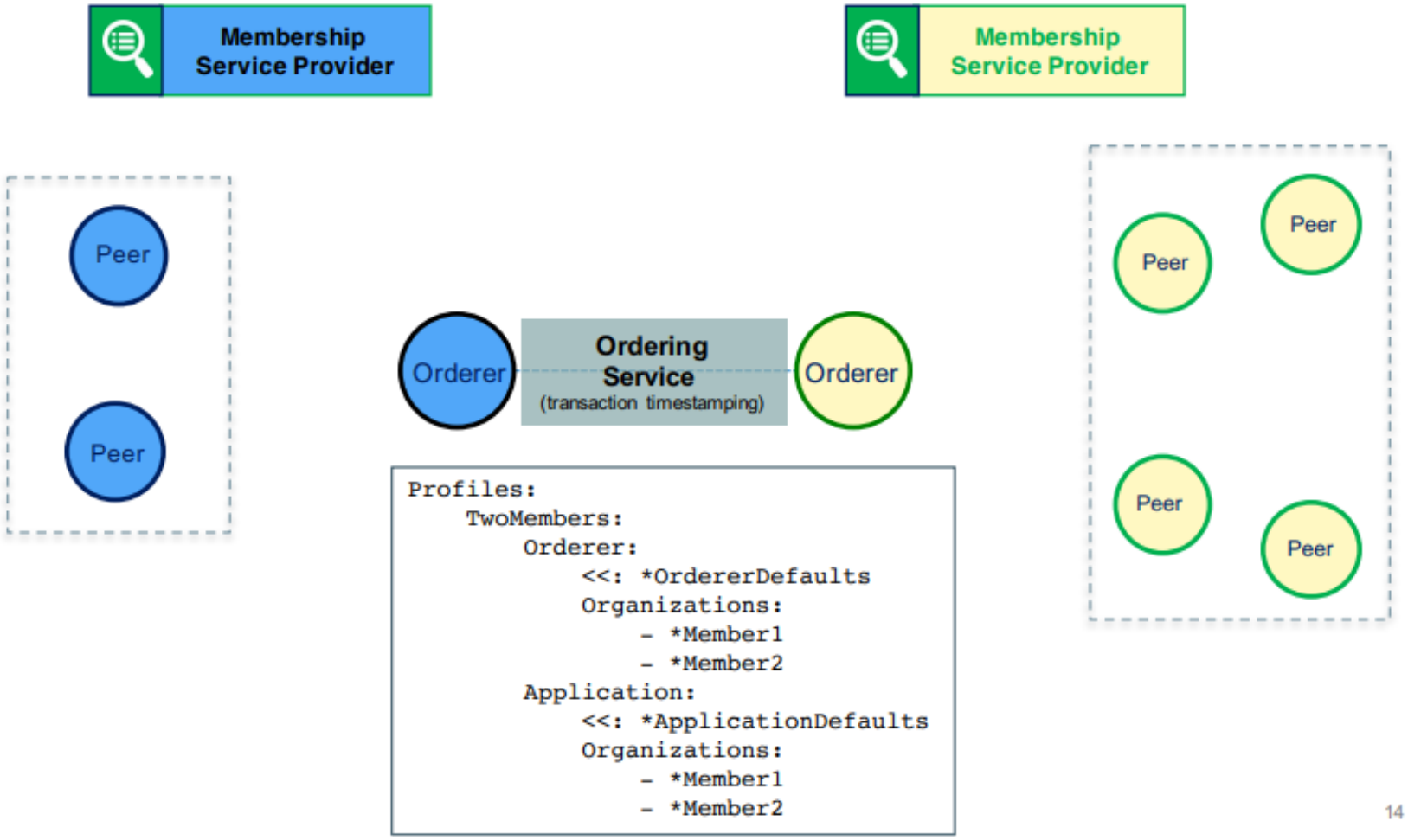
- Each member decides on the number of peers to participate

– For each peer, issue peer identity (local MSP configuration) and start it up

- At this point, we have a network of peers and orderers

– Peers are not yet connected to orderers nor to each other

Two-Member Network



Setting up Channels, Policies, and Chaincodes

Setting up Channels, Policies, and Chaincodes

> Let’s have a look at the business network, and how many more channels are required?

Setting up Channels, Policies, and Chaincodes

- Depending on the business network, 1 or more channels may be required

- To create a channel, send a configuration transaction to the ordering service specifying members of the channel, ACL policies, anchor peers
  - The configuration becomes part of the genesis block of the channel
  - Then notify members to join the channel (a peer may join multiple channels)

- **Deploy chaincodes on the channel with appropriate endorsement policy**
- **Now the network is ready for transacting**

Consensus Redefined

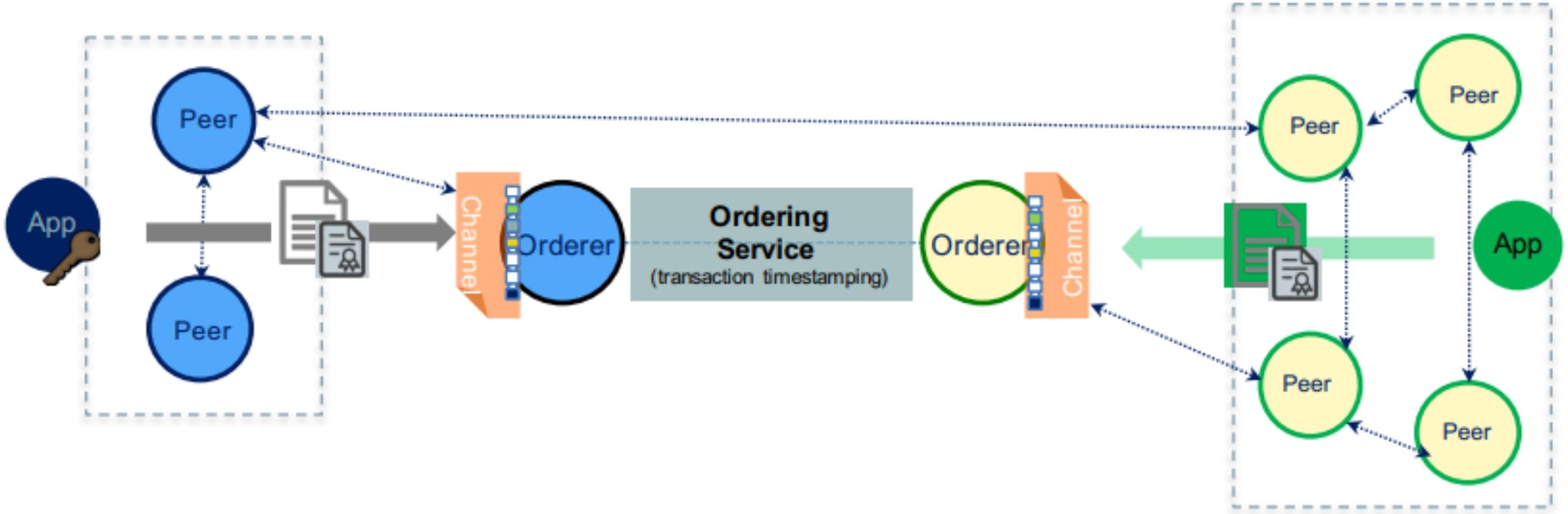
Consensus = Transaction Endorsement + Ordering + Validation

- Endorsement: Each stakeholder decides whether to accept or reject a transaction
- Ordering: Sort all transactions within a period into a block to be committed in that order
- Validation: Verify transaction endorsement satisfied the policy andtransaction transformation is valid according to multiversion concurrency control (MVCC)

Transaction Endorsement

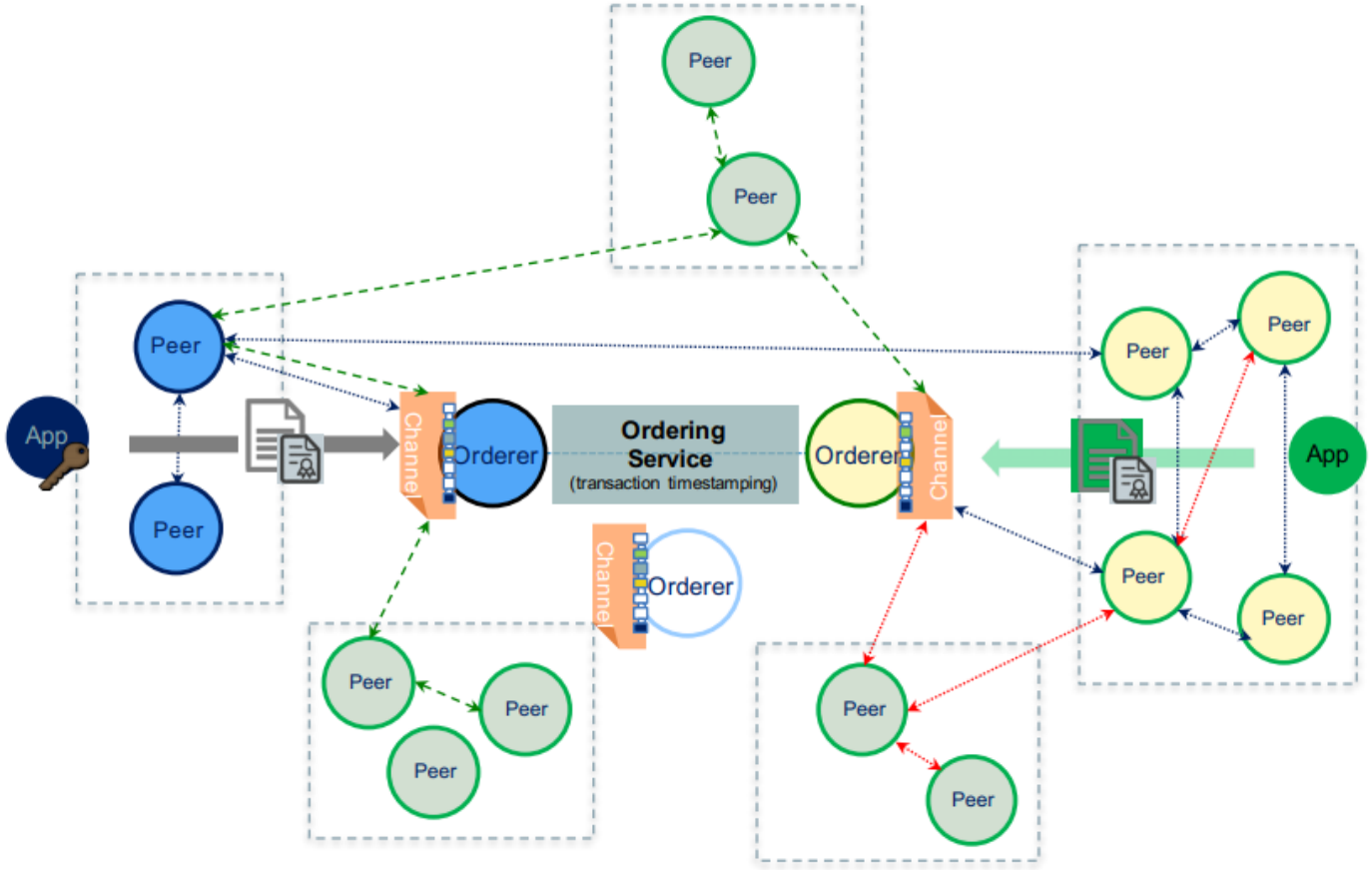
- An endorsement is a signed response of the result of a transaction execution
- An endorsement policy encapsulates the requirement for a transaction to be accepted by the stakeholders, either explicit or implicit
  - A signature from both member1 and member2
  - Either a signature from both member1 and member2 or a signature from member3
  - A signature from John Doe
- The endorsement policy is specified during a chaincode instantiation on a channel; each channel-chaincode may have different endorsement policy

Two-Member Network with A Channel

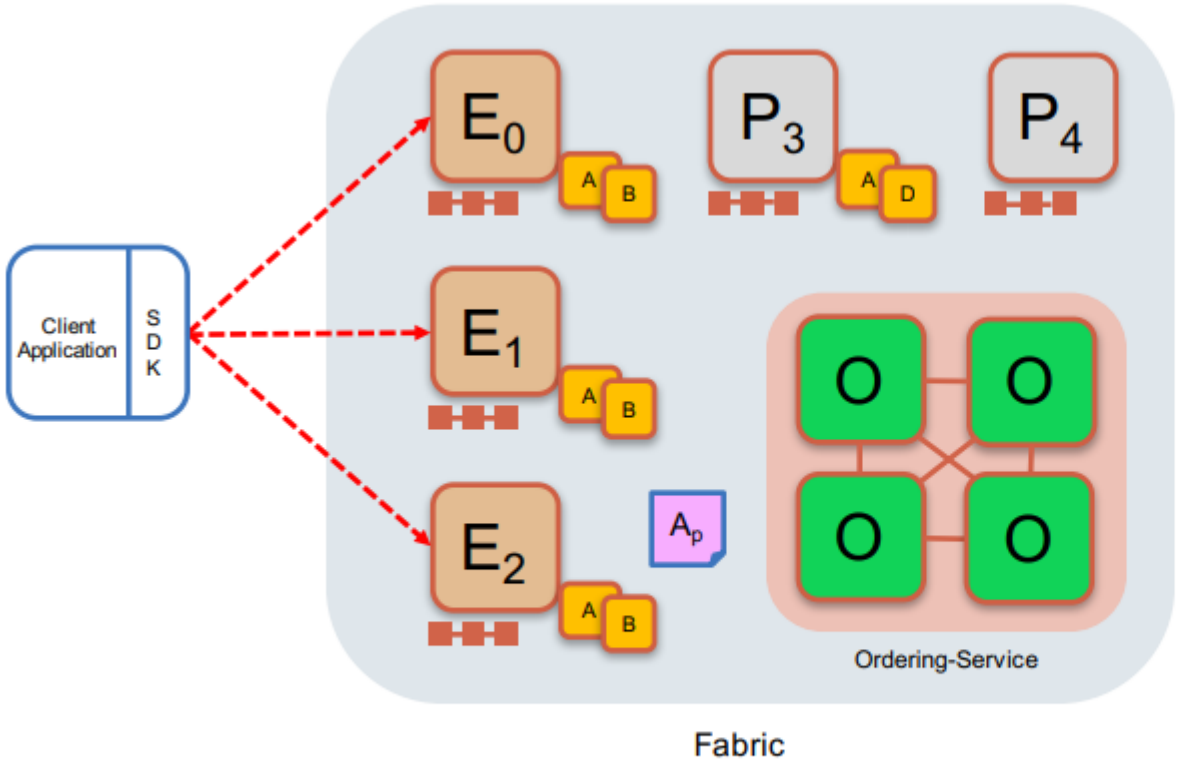


What if we want to add more members ?

N-Member Network with Multichannel



Sample transaction: Step 1/7 – Propose transaction

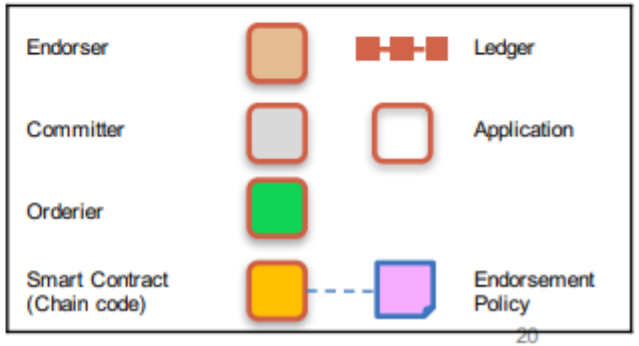


Application proposes transaction

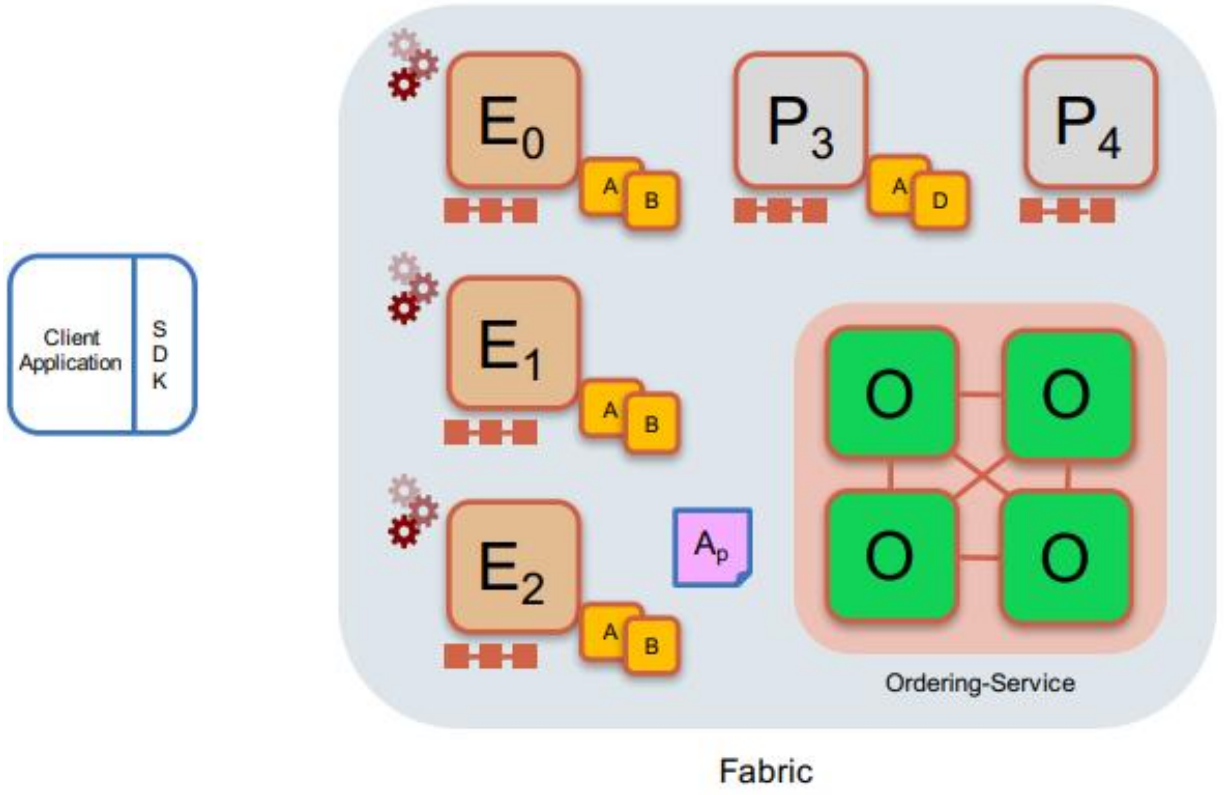
Endorsement policy:
- "E_0, E_1 and E_2 must sign"
- (P_3, P_4 are not part of the policy)

Client application submits a transaction proposal for **chaincode A.** It must target the required peers {E_0, E_1, E_2}

Key:



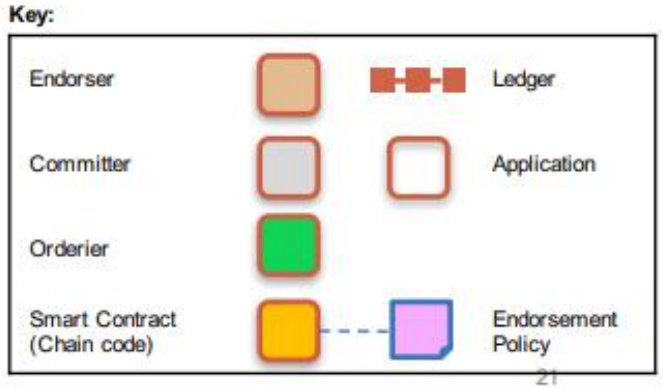
Sample transaction: Step 2/7 – Execute proposal



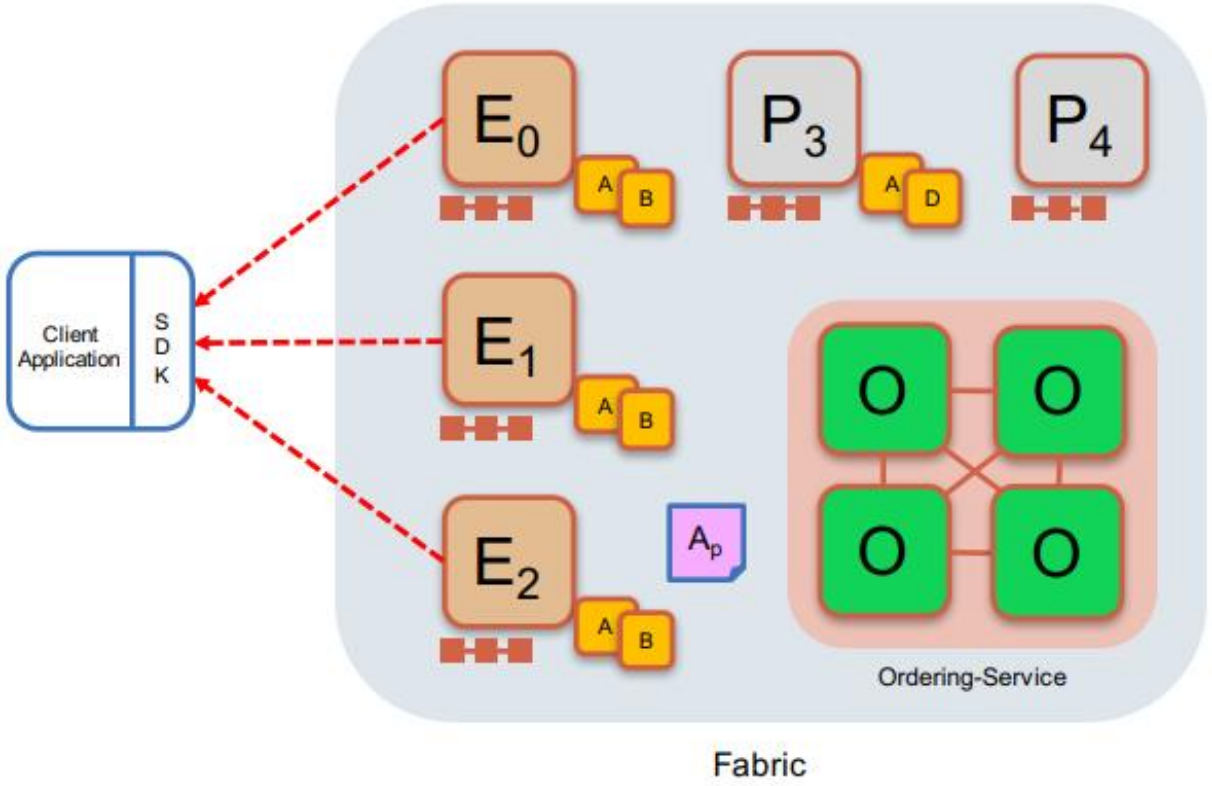
Endorsers Execute Proposals

E_0, E_1 & E_2 will each execute the *proposed* transaction. None of these executions will update the ledger

Each execution will capture the set of **R**ead and **W**ritten data, called **RW sets,** which will now flow in the fabric.

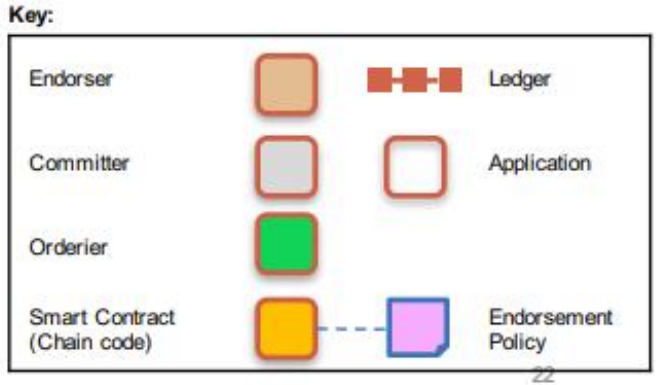


Sample transaction: Step 3/7 – Proposal Response

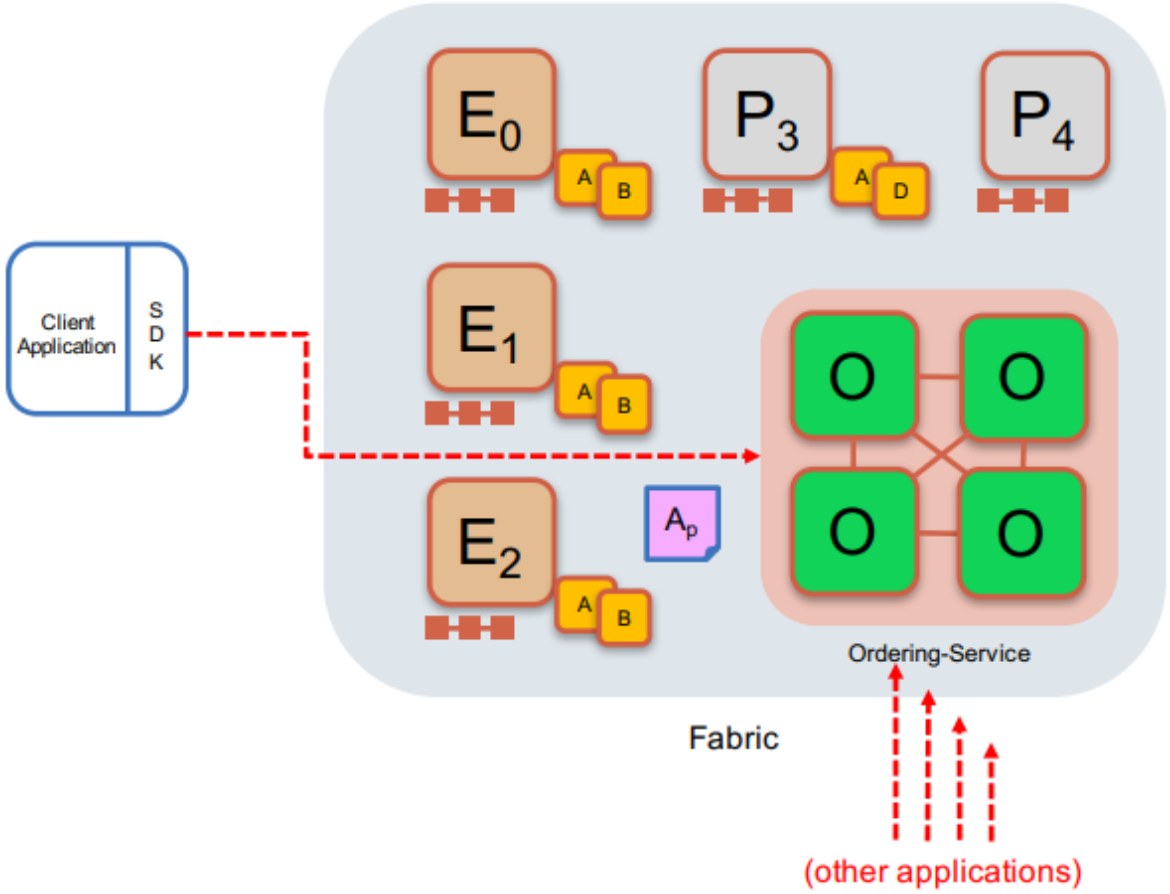


Application receives responses

The RW sets are signed by each endorser and returned to the application



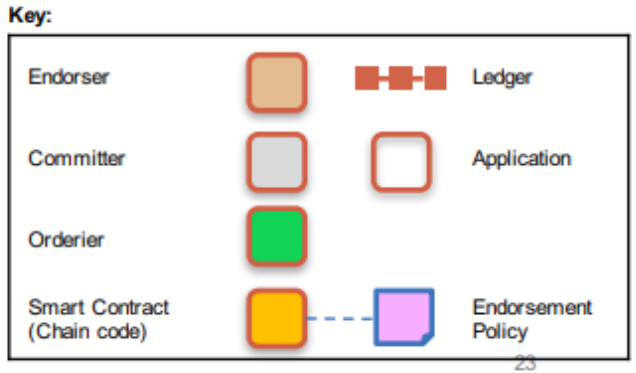
Sample transaction: Step 4/7 – Order Transaction



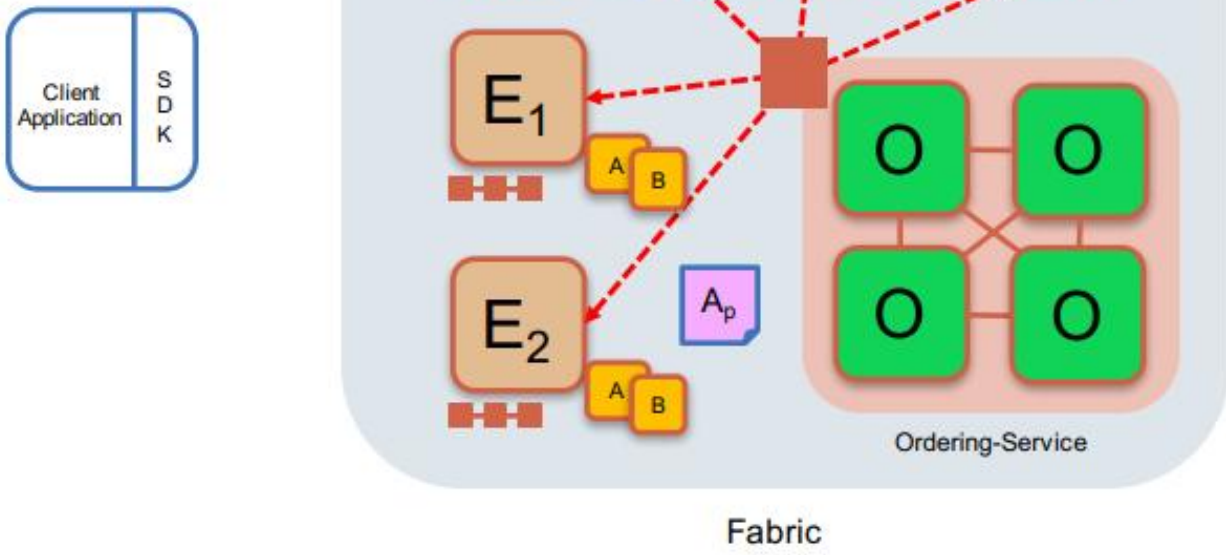
Application submits responses for ordering

Application submits responses as a **transaction** to be ordered.

Ordering happens across the fabric in parallel with transactions submitted by other applications



Sample transaction: Step 5/7 – Deliver Transaction

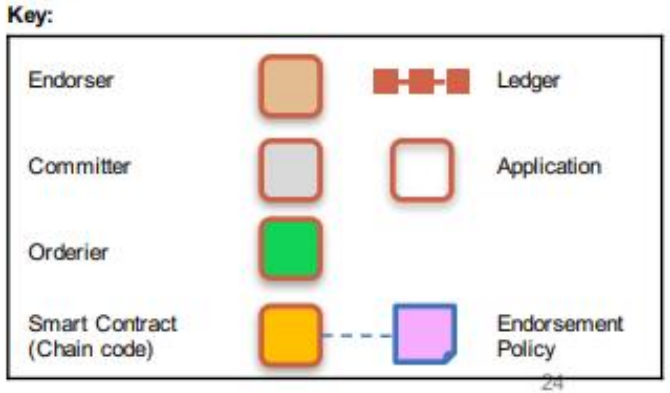


Orderer delivers to all committing peers

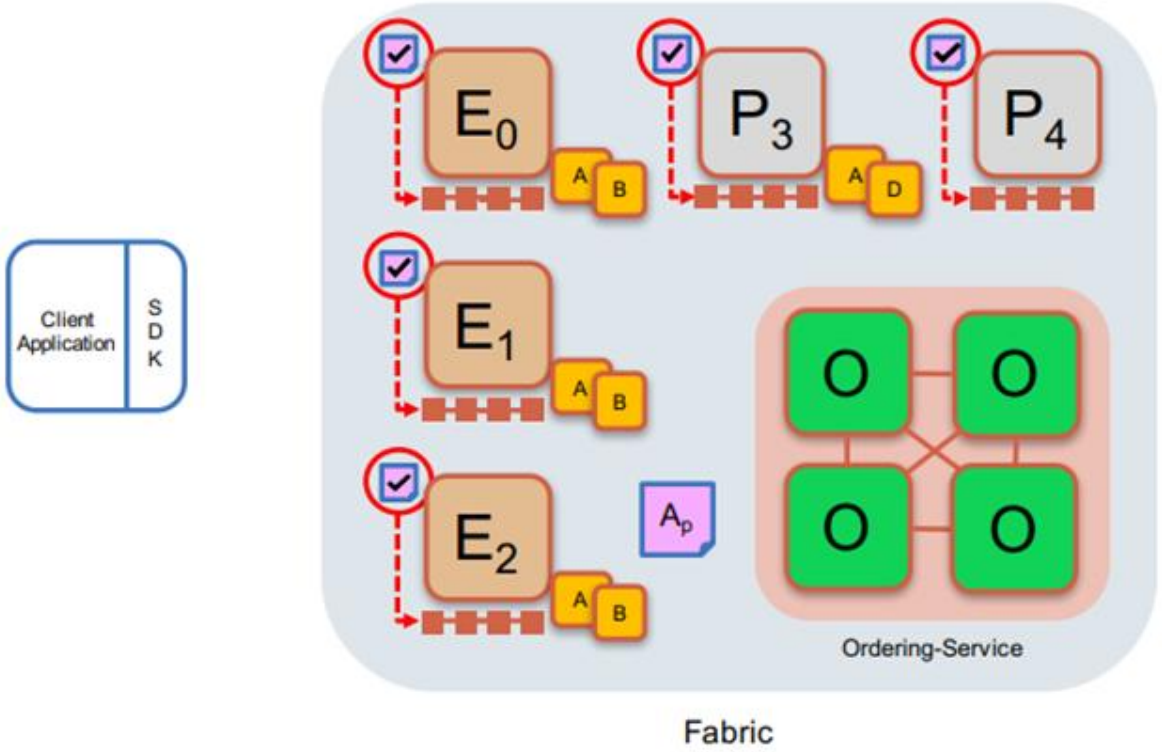
Ordering service collects transactions into blocks for distribution to committing peers. Peers can deliver to other peers using gossip (not shown)

Different ordering algorithms available:

- SOLO (single node, development)
- Kafka (blocks map to topics)
- SBFT (tolerates faulty peers, future)



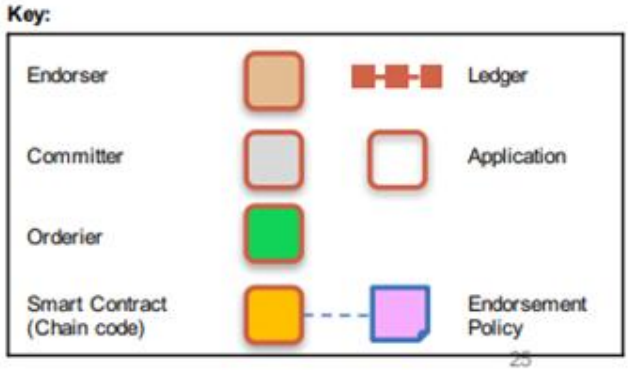
Sample transaction: Step 6/7 – Validate Transaction



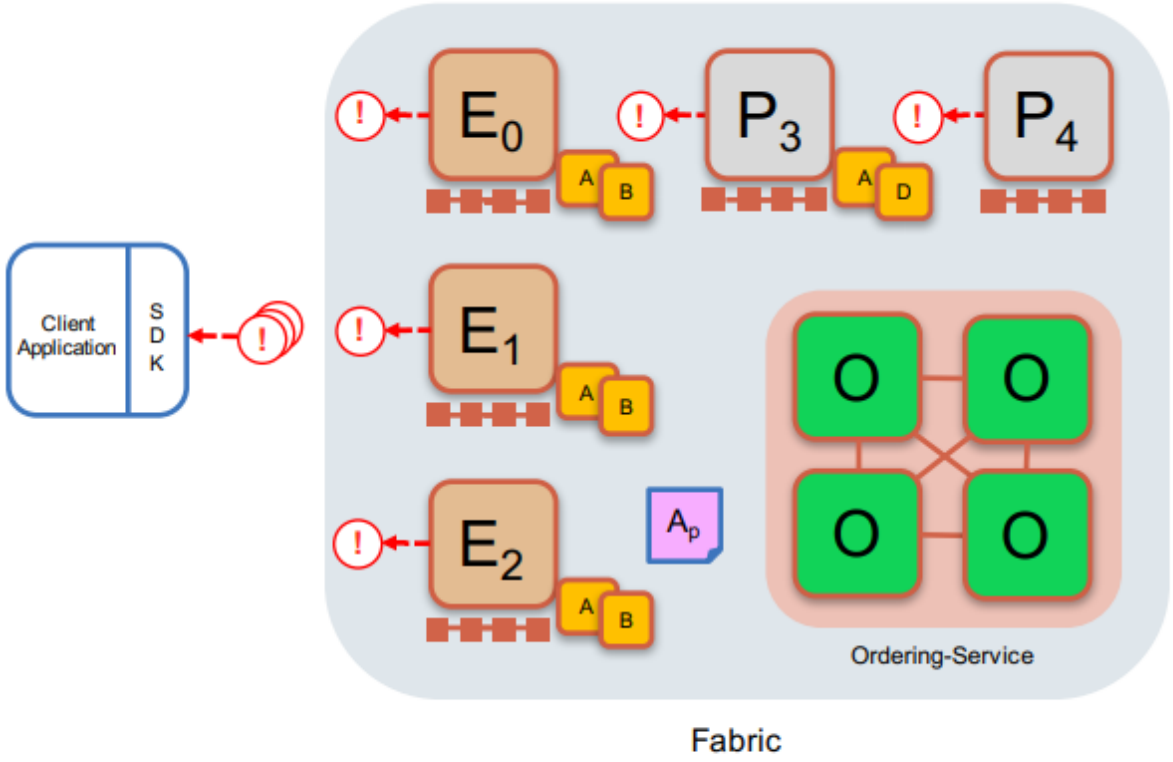
Committing peers validate transactions

Every committing peer validates against the endorsement policy. Also check RW sets are still valid for the current state

Transactions are written to the ledger and update caching DBs with validated transactions



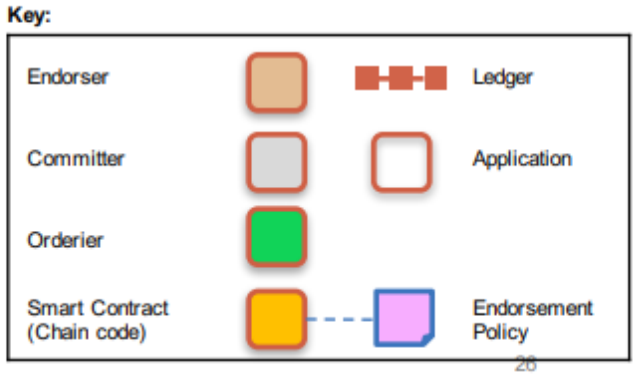
Sample transaction: Step 7/7 – Notify Transaction



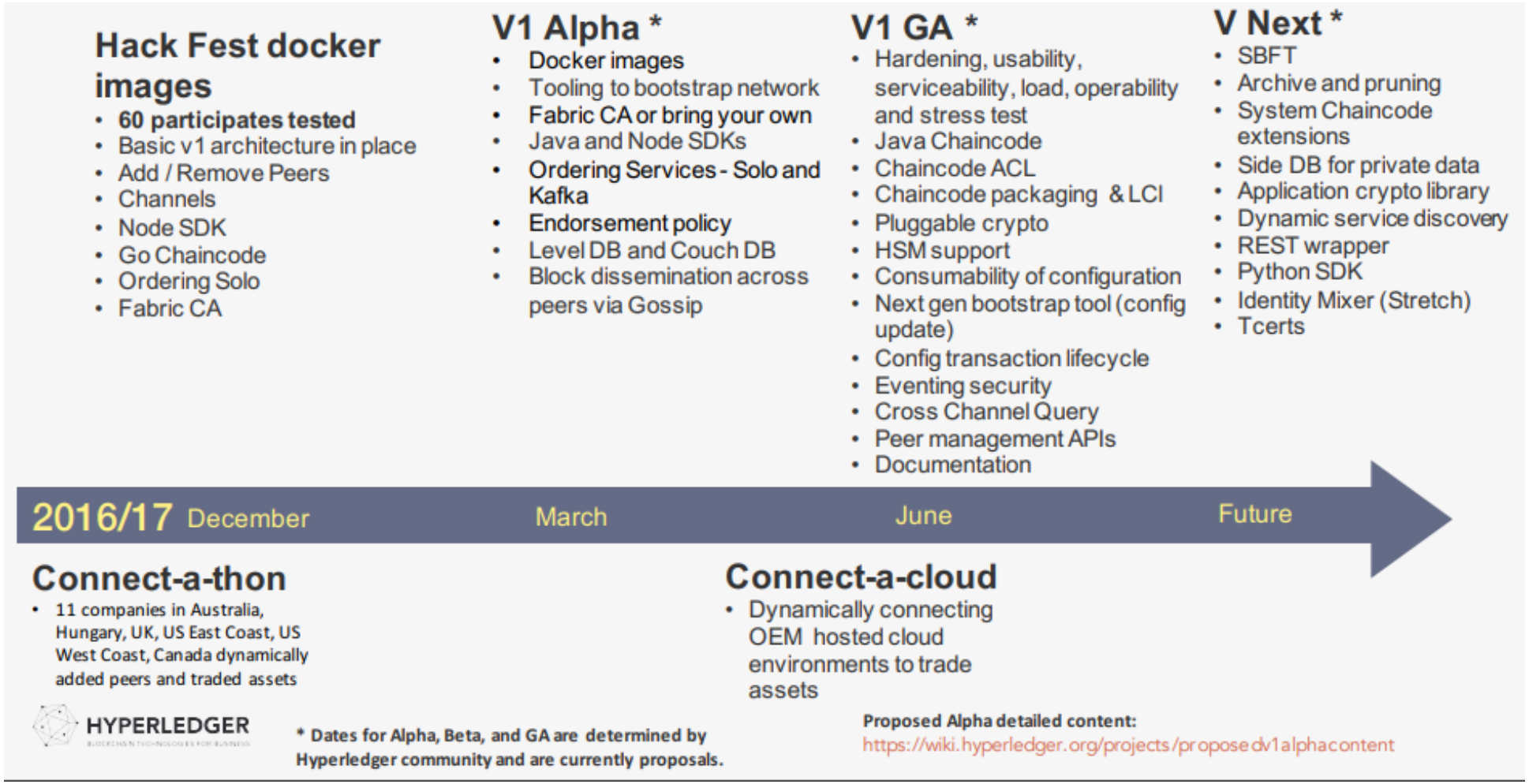
Committing peers notify applications

Applications can register to be notified when transactions succeed or fail, and when blocks are added to the ledger

Applications will be notified by each peer to which they are connected



Hyperledger Fabric Roadmap

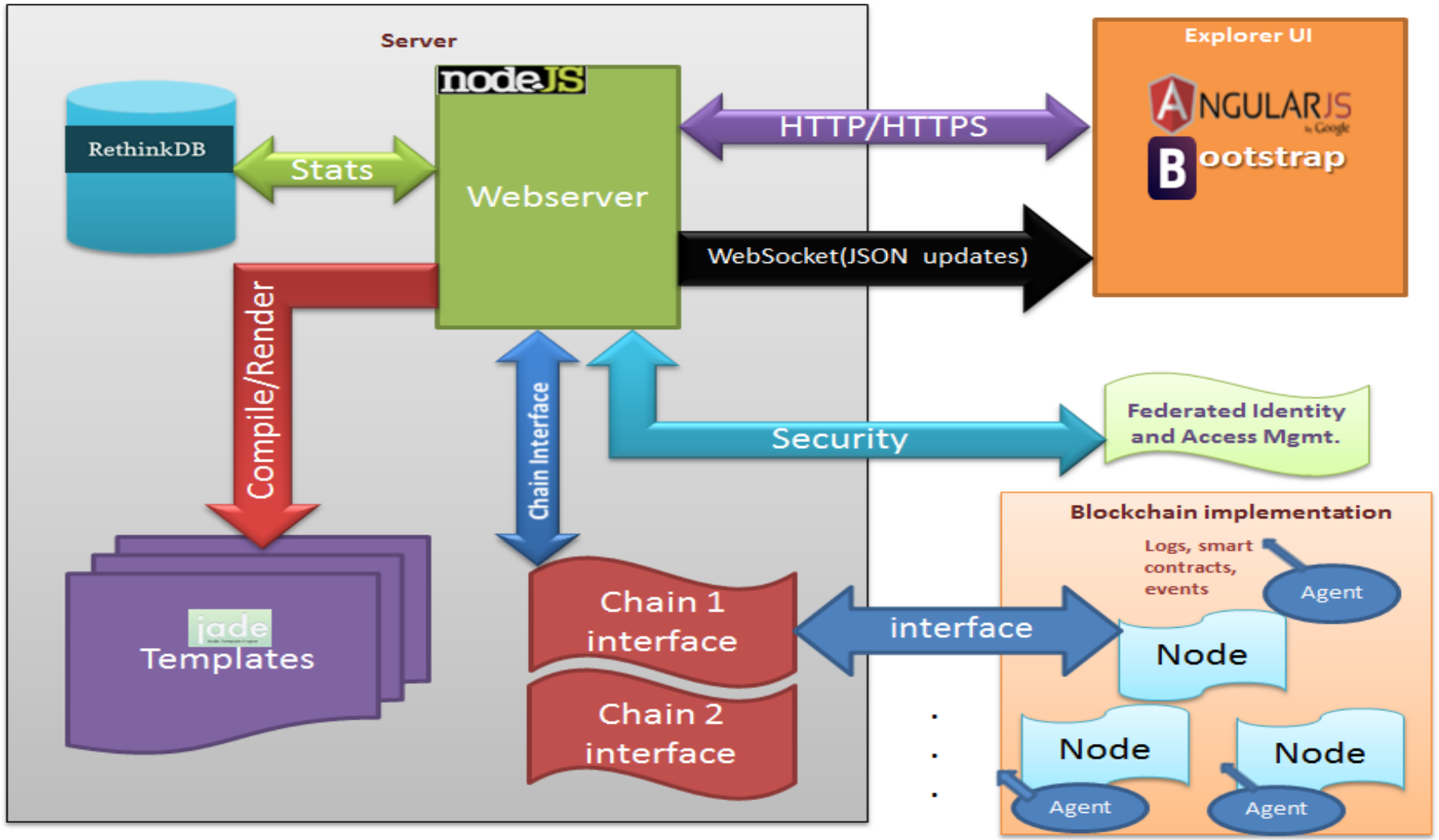


Hyperledger Explorer Components

Hyperledger Explorer Components

Let us learn about Hyperledger Explorer Components.

Hyperledger Explorer Components



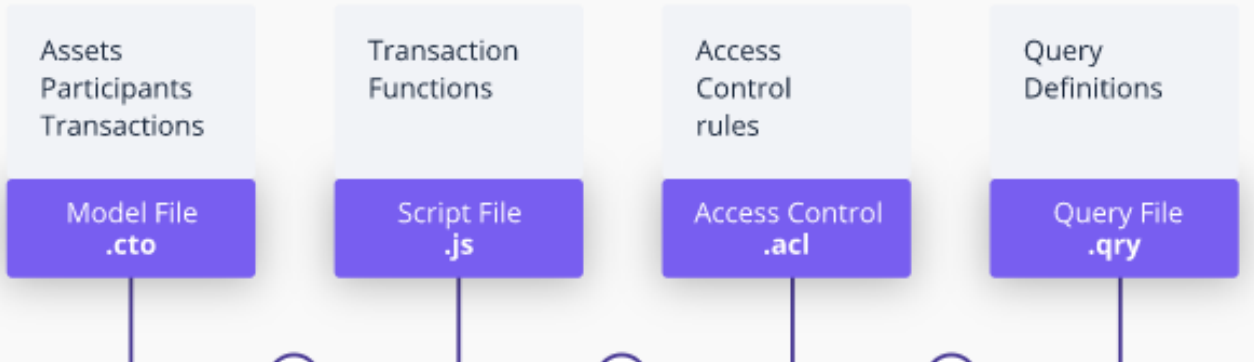
Hyperledger Composer

- Hyperledger Composer is an extensive, open development toolset and framework to make developing blockchain applications easier
- Hyperledger Composer is a set of collaboration tools for building blockchain business networks that make it simple and fast for business owners and developers to create smart contracts and blockchain applications to solve business problems.
- Hyperledger Composer allows you us model our business network and integrate existing systems and data with your blockchain applications
- Applications use business centric APIs to invoke transactions that create, delete, and update assets and transfer them between participants
- Assets, participants and transactions are recorded in the worlds state in registries

Key Development concepts

- Model files describe the assets, participants, transactions, and events that exist in a business network
  - Expressive modelling language includes relationships, arrays, and validation rules
  - Data serialized as JSON, and is fully validated by the Hyperledger Composer runtime
- Access control lists define rules for sharing and privacy
  - Rules are automatically enforced by the Hyperledger Composer runtime
- Transaction processor functions implement additional business requirements
  - Standard JavaScript code executed on the Blockchain by the Hyperledger Composer runtime
- A business network definition is the set of the above for a given business network

Business Model archive



Use Composer to create a Business Network Definition, comprised of Model (.cto), Script (.js), ACL (.acl) and Query (.qry) files.

Business Network Archive
.bna

Package up your Business Network Definition and export it as an archive (.bna file) ready to deploy it somewhere.

Hyperledger Fabric
Cloud / Local

Web Browser / Node.js
Online

Use ID Cards (which include connection profiles and credentials) to deploy your Business Network Definition to a distributed ledger.

Model files

```
namespace org.acme.sample

asset SampleAsset identified by assetId {
  o String assetId
  --> SampleParticipant owner
  o String value
}

participant SampleParticipant identified by participantId {
  o String participantId
  o String firstName
  o String lastName
}

transaction SampleTransaction identified by transactionId {
  o String transactionId
  --> SampleAsset asset
  o String newValue
}

event SampleEvent identified by eventId {
  o String eventId
  --> SampleAsset asset
  o String oldValue
  o String newValue
}
```

Namespaces group related types

Specific keywords for defining assets, participants, transactions, and events in a business network

Non-abstract types must have an identifying field

Types have a set of properties, denoted with a leading “o”

Types can have relationships to other types, denoted with a leading ”-->”

Access Control lists

```
rule EverybodyCanReadEverything {
    description: "Allow all participants read access to all resources"
    participant: "org.acme.sample.SampleParticipant"
    operation: READ
    resource: "org.acme.sample"
    action: ALLOW
}

rule EverybodyCanSubmitTransactions {
    description: "Allow all participants to submit transactions"
    participant: "org.acme.sample.SampleParticipant"
    operation: CREATE
    resource: "org.acme.sample.SampleTransaction"
    action: ALLOW
}

rule OwnerHasFullAccessToTheirAssets {
    description: "Allow all participants full access to their assets"
    participant(p): "org.acme.sample.SampleParticipant"
    operation: ALL
    resource(r): "org.acme.sample.SampleAsset"
    condition: (r.owner.getIdentifier() === p.getIdentifier())
    action: ALLOW
}
```

Rules are executed in order from top to bottom. If no rule permits access, then access is denied

Each rule permits or denies permission to a participant to perform an operation on a resource. Resources are assets, participants, etc

Rules can contain complex conditions. Conditions are written in JavaScript, and can access the participant and resource

Transaction processor functions

```
/**
 * Sample transaction processor function.
 * @param {org.acme.sample.SampleTransaction} tx The sample transaction instance.
 * @transaction
 */
function sampleTransaction(tx) {

    // Save the old value of the asset.
    var oldValue = tx.asset.value;

    // Update the asset with the new value.
    tx.asset.value = tx.newValue;

    // Get the asset registry for the asset.
    return getAssetRegistry('org.acme.sample.SampleAsset')
        .then(function (assetRegistry) {

            // Update the asset in the asset registry.
            return assetRegistry.update(tx.asset);

        })
        .then(function () {

            // Emit an event for the modified asset.
            var event = getFactory().newEvent('org.acme.sample', 'SampleEvent');
            event.asset = tx.asset;
            event.oldValue = oldValue;
            event.newValue = tx.newValue;
            emit(event);

        });

}
```

Transaction processor functions are written in standard JavaScript (ES5) and they can use the runtime API to interact with the Blockchain

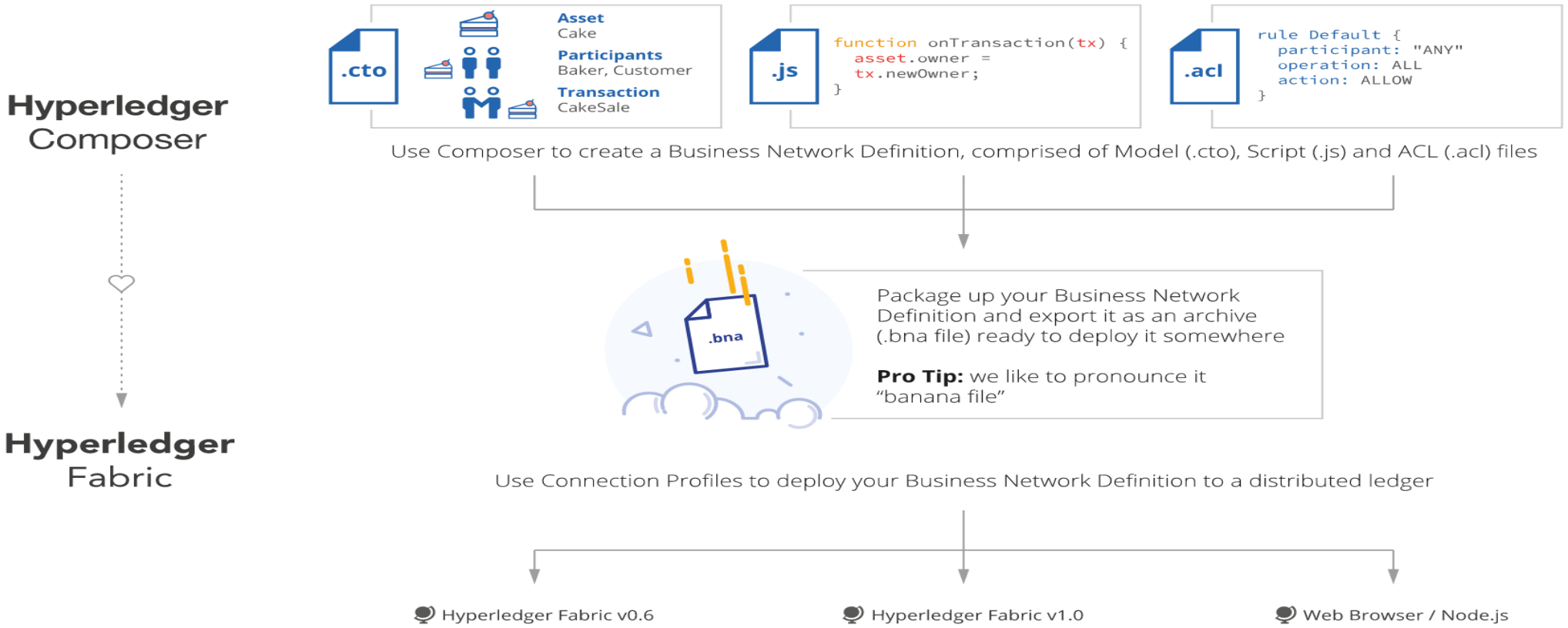
Annotations in the comments define the transaction that the function accepts, and that the function is a transaction processor function

Transaction processor functions are atomic; all of the updates are applied, or no updates are applied

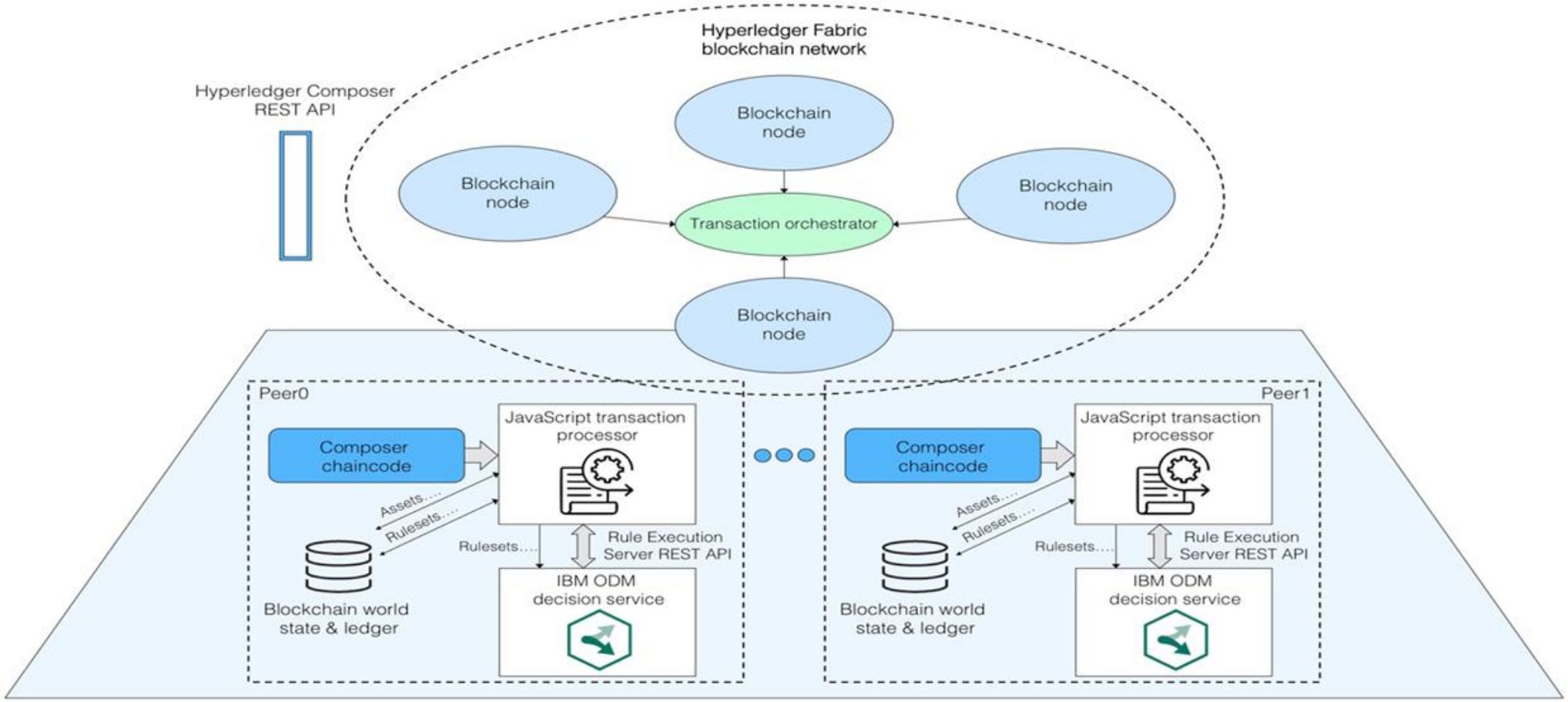
Interact with the registry APIs to add, update, and remove assets stored on the Blockchain in the world state

Use the event APIs to publish events to external applications for notifications and triggering downstream processing

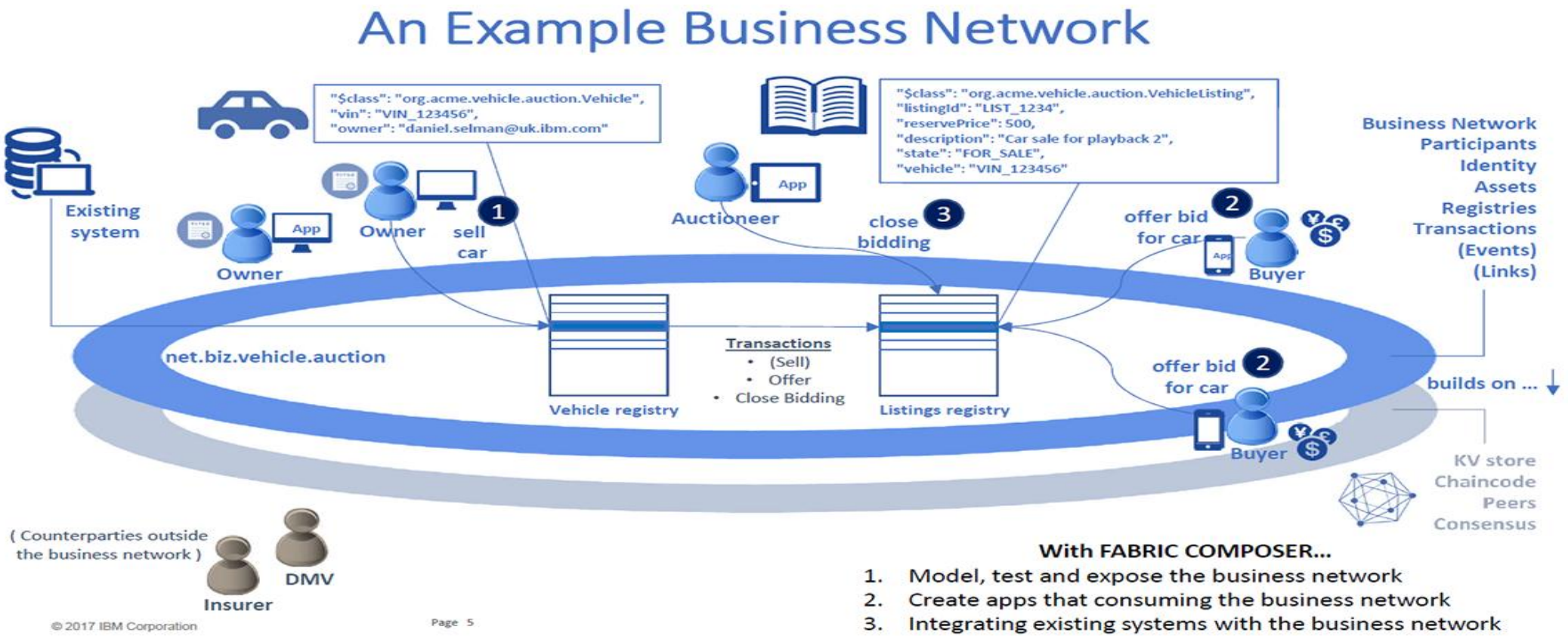
Business Network Definition



Business Network Execution



An example Business Network with Composer



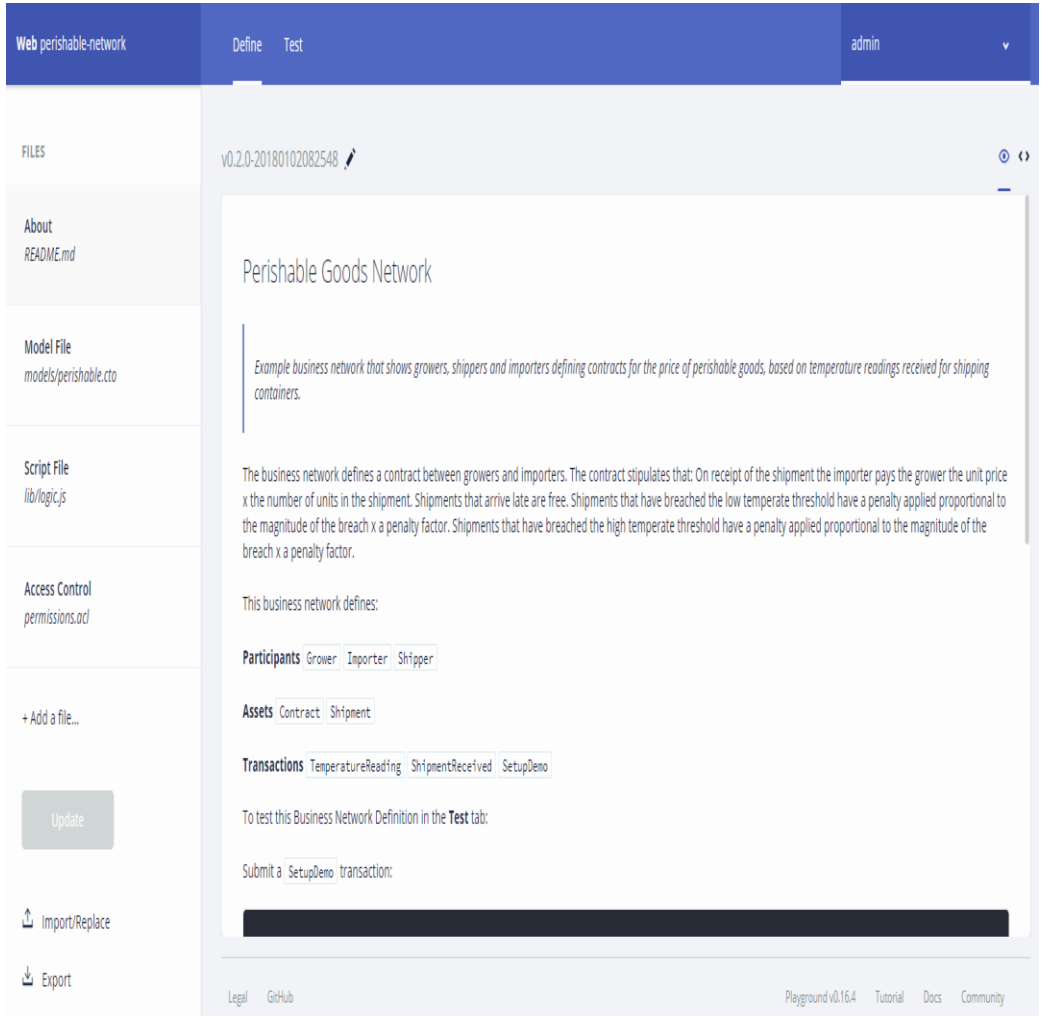
Composer Playground

- A interactive web tool for the development (and test) of business networks without installing anything

- It is a interactive web tool for the development (and test) of business networks without installing anything

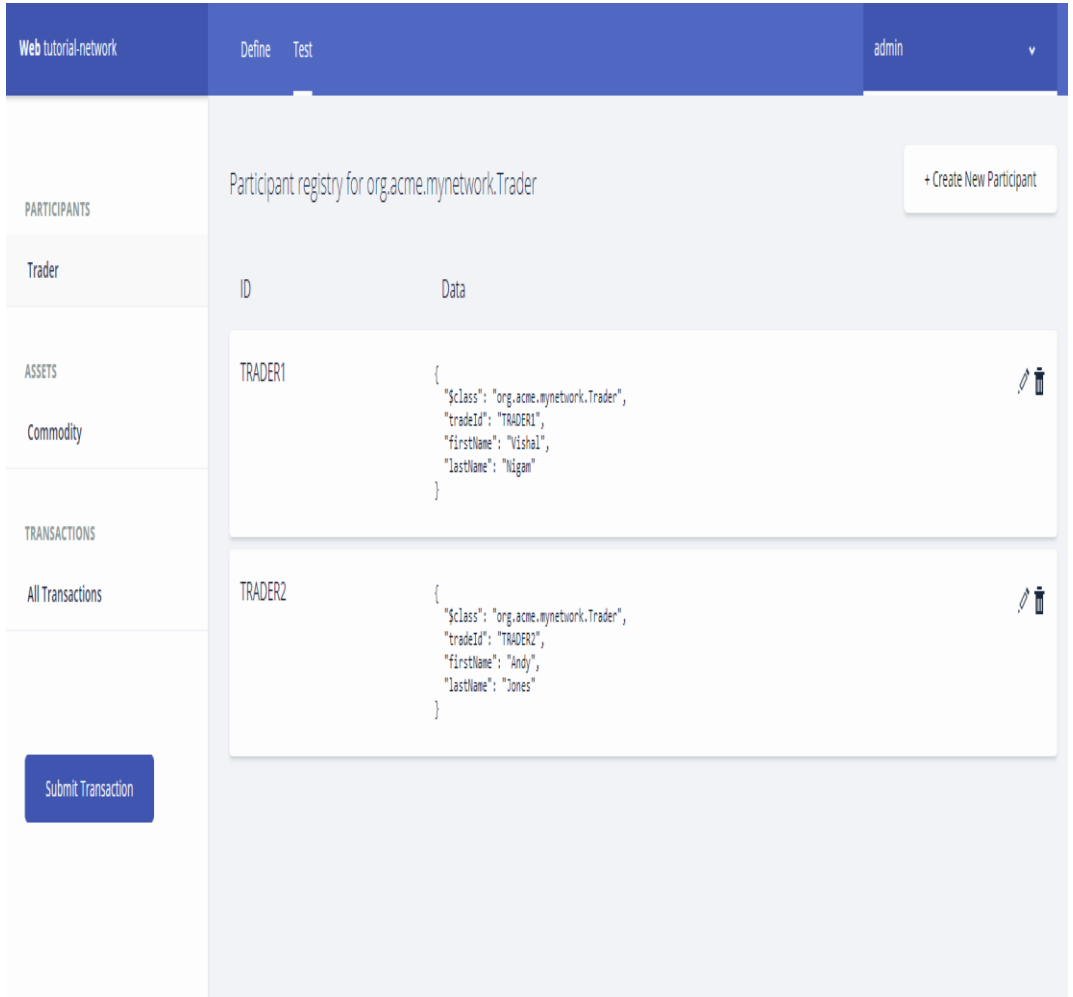
- DSL for specific assets, participants & transactions in your network
  - Live content assist, syntax checking

- Non-web options also possible
  - Hosted & local playgrounds
  - Atom & VSCode plug-ins



Test Business Networks: Composer Playground

- Test tab on playground reflects dynamically defined model
  - Creates default registries
- Create, read, update, and delete resources Submit transactions
- Multiple environments supported, e.g. test, prod



Hyperledger Composer

Hyperledger Composer

Let’s see what is Hyperledger Composer and what purpose they serve.

Hyperledger Composer

> Hyperledger Composer is an extensive, open development toolset and framework tc make developing blockchain applications easier. Our primary goal is to accelerate time to value, and make it easier to integrate your blockchain applications with the existing business systems. You can use Composer to rapidly develop use cases and deploy a blockchain solution in weeks rather than months. Composer allows you to model your business network and integrate existing systems and data with your blockchain applications.

- Hyperledger Composer supports the existing Hyperledger Fabric blockchain infrastructure and runtime, which supports pluggable blockchain consensus protocols to ensure that transactions are validated according to policy by the designated business network participants.

How Hyperledger Composer work in practice?

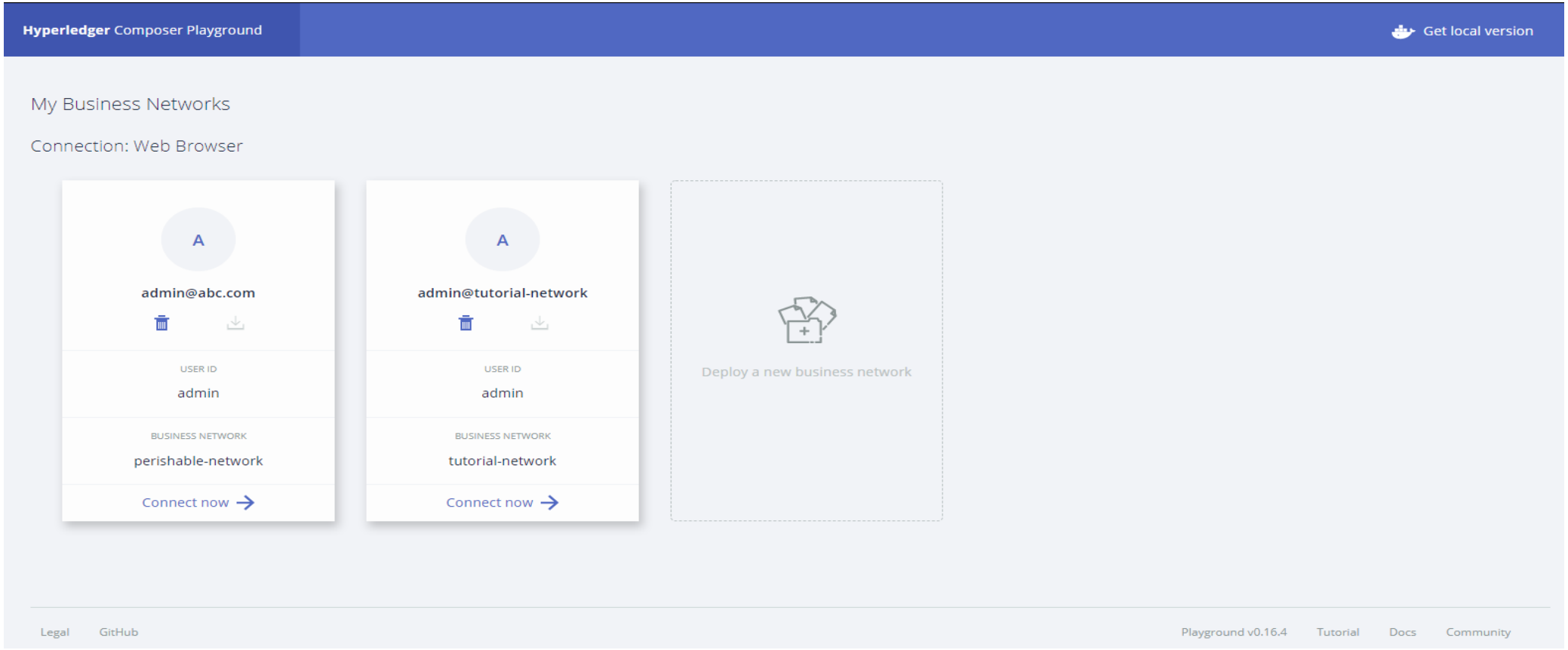
For an example of a business network in action; a realtor can quickly model their business network as such:

- Assets: houses and listings
- Participants: buyers and homeowners
- Transactions: buying or selling houses, and creating and closing listings

- Participants can have their access to transactions restricted based on their role as either a buyer, seller, or realtor
- The realtor can then create an application to present buyers and sellers with a simple user interface for viewing open listings and making offers
- This business network could also be integrated with existing inventory system, adding new houses as assets and removing sold properties
- Relevant other parties can be registered as participants, for example a land registry might interact with a buyer to transfer ownership of the land

Step First: Open the Hyperledger Composer Playground

Open Composer Playground. You should see the My Wallet screen



Step Two: Creating a new business network

Next, we want to create a new business network from scratch
A business network has a couple of defining properties; a name, and an optional description. (You can also choose to base a new business network on an existing template, or import your own template)

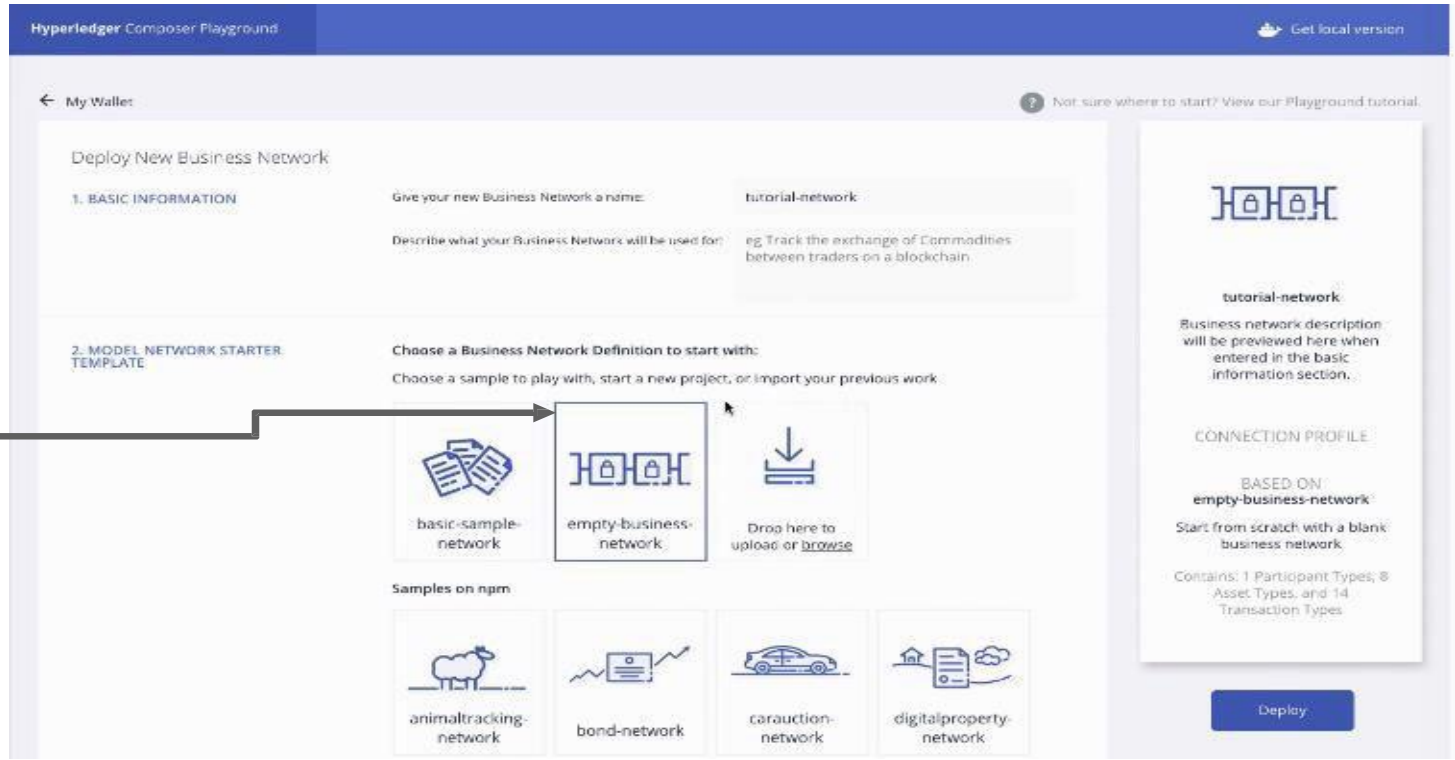
Follow these steps:

1. Click Deploy a new business network under the Web Profile heading to get started.
2. The new business network needs a name, let's call it new-network
3. Optionally, you can put in a network description.
4. Next we must select a business network to base ours on, because we want to build the network from scratch, click empty-business-network

Choosing a Business Network Definition

To start with choose a Business Network definition from the given options

Chose Empty Network

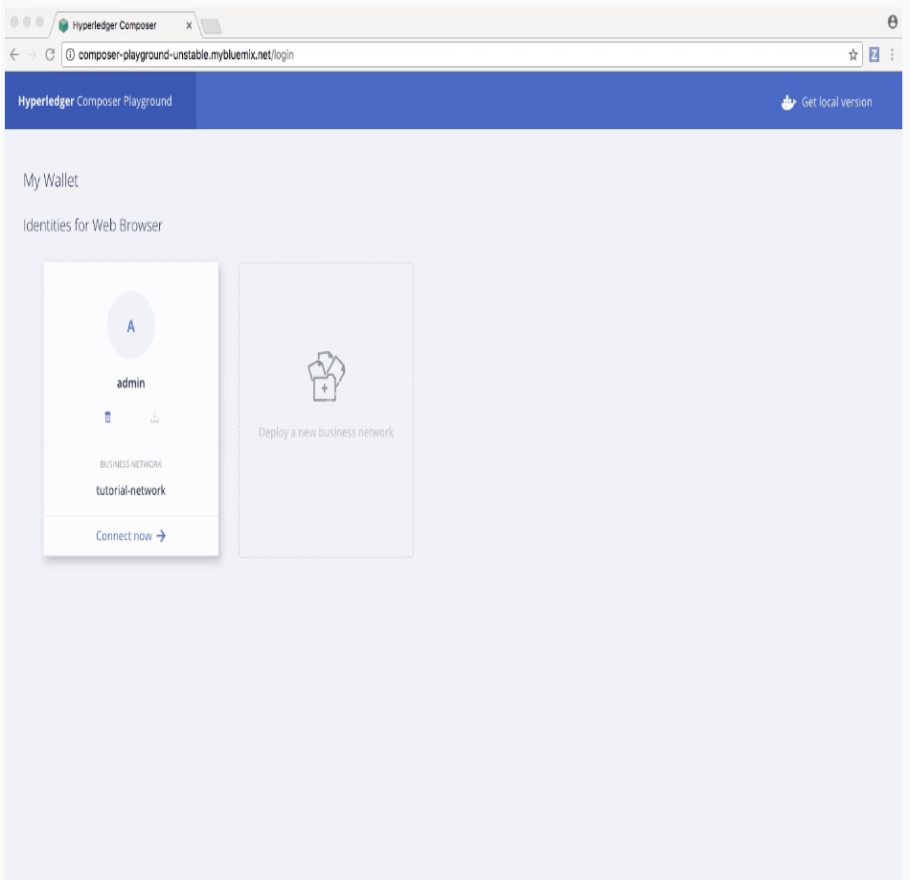


Step Three: Connecting to the business network

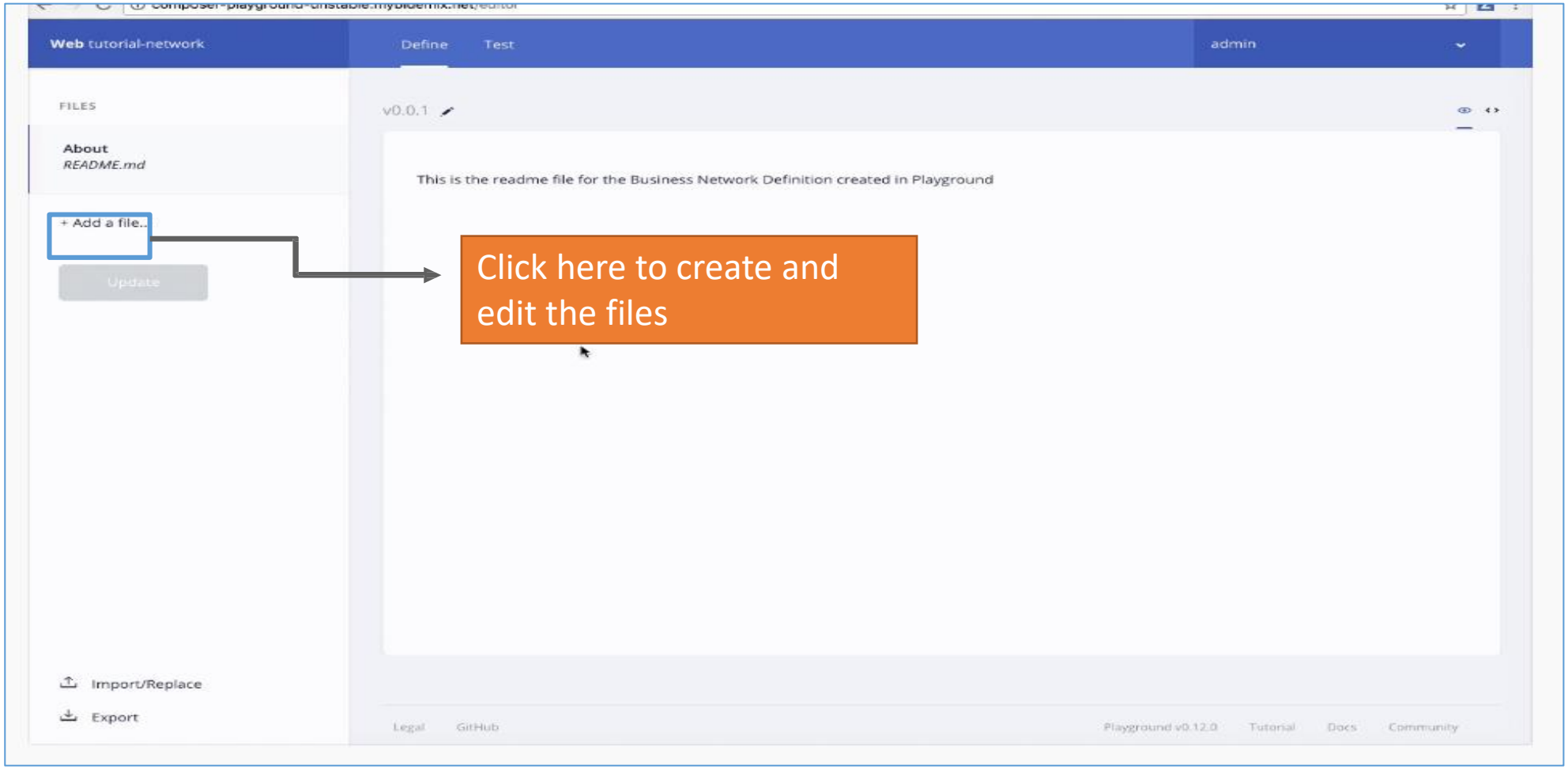
Now that we've created and deployed the business network, you should see a new ID card called admin for our business network new-network in your wallet

- When connecting to an external blockchain, ID cards represent everything necessary to connect to a business network. They include connection details, authentication material, and metadata

- To connect to our business network click Connect now under our ID card



Adding files to make up a Business Network Definition



Step Four: Adding a model file

As you can see, we're in the Define tab right now, this tab is where you create and edit the files that make up a business network definition

As we selected an empty business network template, we need to define our business network files

The first step is to add a model file. Model files define the assets, participants, transactions, and events in our business network

- Click the Add a file button.
- Click the Model file and click Add.
- Delete the lines of codes in the model file and replace it with this:

This domain model defines a single asset type Commodity and single participant type Trader and a single transaction type Trade that is used to modify the owner of a commodity.

Step Five: Adding a transaction processor script file

- Now that the domain model has been defined, we can define the transaction logic for the business network
    - These functions are automatically executed when a transaction is submitted for processing
- Click the Add a file button.
- Click the Script file and click Add.
- Delete the lines of code in the script file and replace it with the following code:

```
/**
* Track the trade of a commodity
from one trader to another
@param {org.acme.mynetwork.Trade}
trade
- the trade to be processed
@transaction */
function tradeCommodity(trade) {
trade.commodity.owner =
trade.newOwner; return
getAssetRegistry('org.acme.mynetwork.
Commodity') .then(function
(assetRegistry)
{ return
assetRegistry.update(trade.commodity
);
}); }
```

Step Six: Adding an access control file

Access control files define the access control rules for business networks. (While you can have multiple model or script files, you can only have one access control file in any business network)

- Click the Add a file button
- Click the Access Control file and click Add
- Delete the lines of code in the access control file and replace it with the following code:

```
/** * Access control rules for
mynetwork */ rule Default {
description: "Allow all
participants access to all
resources" participant: "ANY"
operation: ALL resource:
"org.acme.mynetwork.*" action:
ALLOW } rule SystemACL {
description: "System ACL to
permit all access" participant:
"org.hyperledger.composer.system.
Participant" operation: ALL
resource:
"org.hyperledger.composer.system.
**" action: ALLOW }
```

This access control rule allows all participants to access all business network resources, and allows all users to have system access control privileges

Step Seven: Deploying the updated business network

Now that we've created our model, script, and access control files, we need to deploy and test our business network

Click Update to deploy our new model, script, and access control files

```
/**
 * My commodity trading network
 */
namespace org.acme.mynetwork
asset Commodity identified by tradingSymbol {
    o String tradingSymbol
    o String description
    o String mainExchange
    o Double quantity
    --> Trader owner
}
participant Trader identified by tradeId {
    o String tradeId
    o String firstName
    o String lastName
}
transaction Trade {
    --> Commodity commodity
    --> Trader newOwner
}
```

Copy

Step Eight: Testing the business network definition

Next, we need to test our business network by creating some participants (in this case Traders), creating an asset (a Commodity), and then using our Trade transaction to change the ownership of the Commodity

Click the **Test** tab to get started

Step Nine: Creating participants

The first thing we should add to our business network is two participants. (Ensure that you have the Trader tab selected on the left, and click **Create New Participant** in the upper right)

- What you can see is the data structure of a Trader participant. We want some easily recognizable data, so delete the code that's there and paste the following:

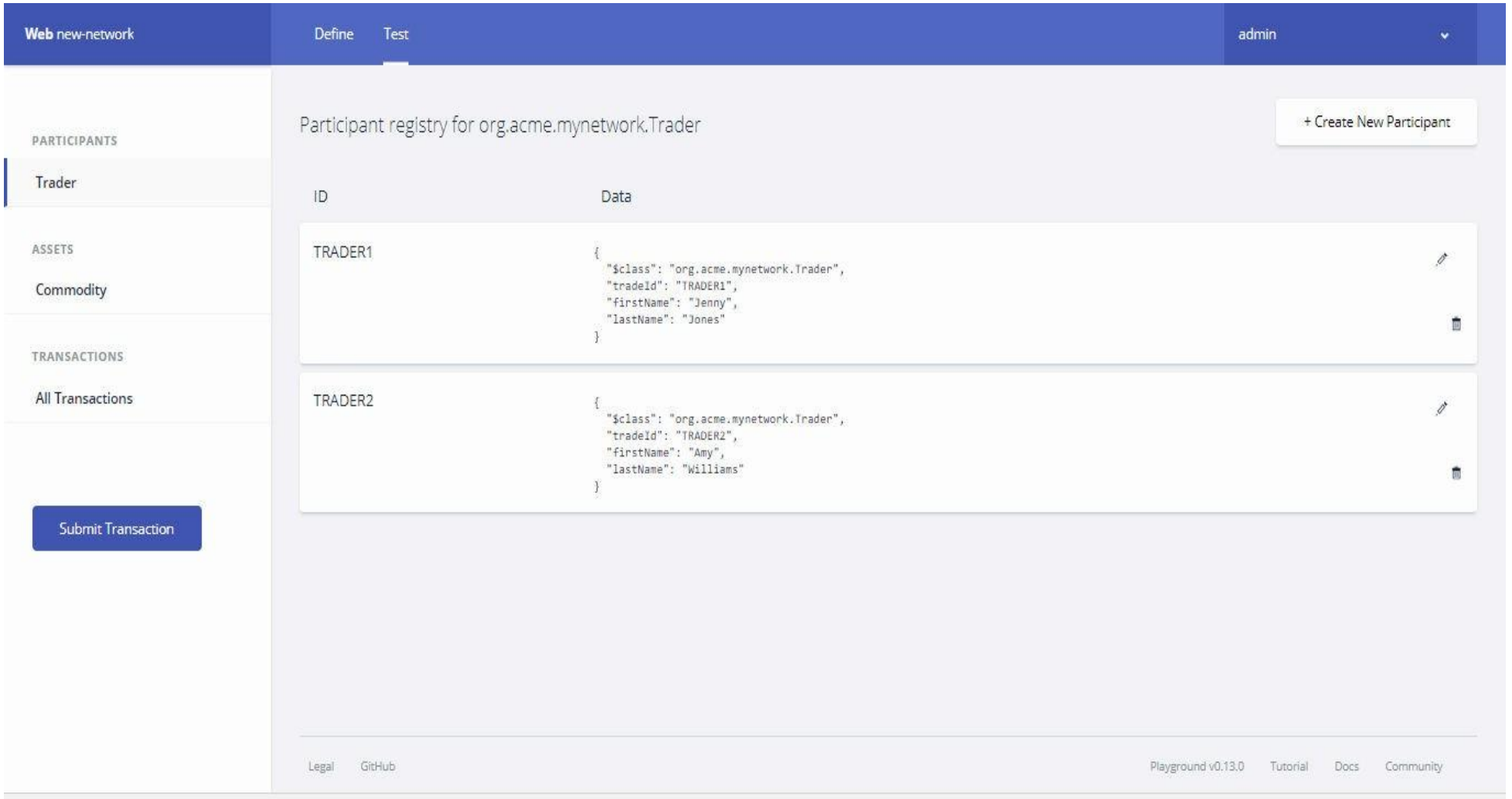
```
{ "$class": "org.acme.mynetwork.Trader", "tradeId": "TRADER1", "firstName":
"Jenny", "lastName": "Jones" }
```

- Click Create New to create the participant
- You should be able to see the new Trader participant you've created. We need another Trader to test our Trade transaction though, so create another Trader, but this time, use the following data:

```
{ "$class": "org.acme.mynetwork.Trader", "tradeId": "TRADER2", "firstName":
"Amy", "lastName": "Williams" }
```

Note: Make sure that both participants exist in the Trader view before moving on!

Two Participants created: TRADER1, TRADER2



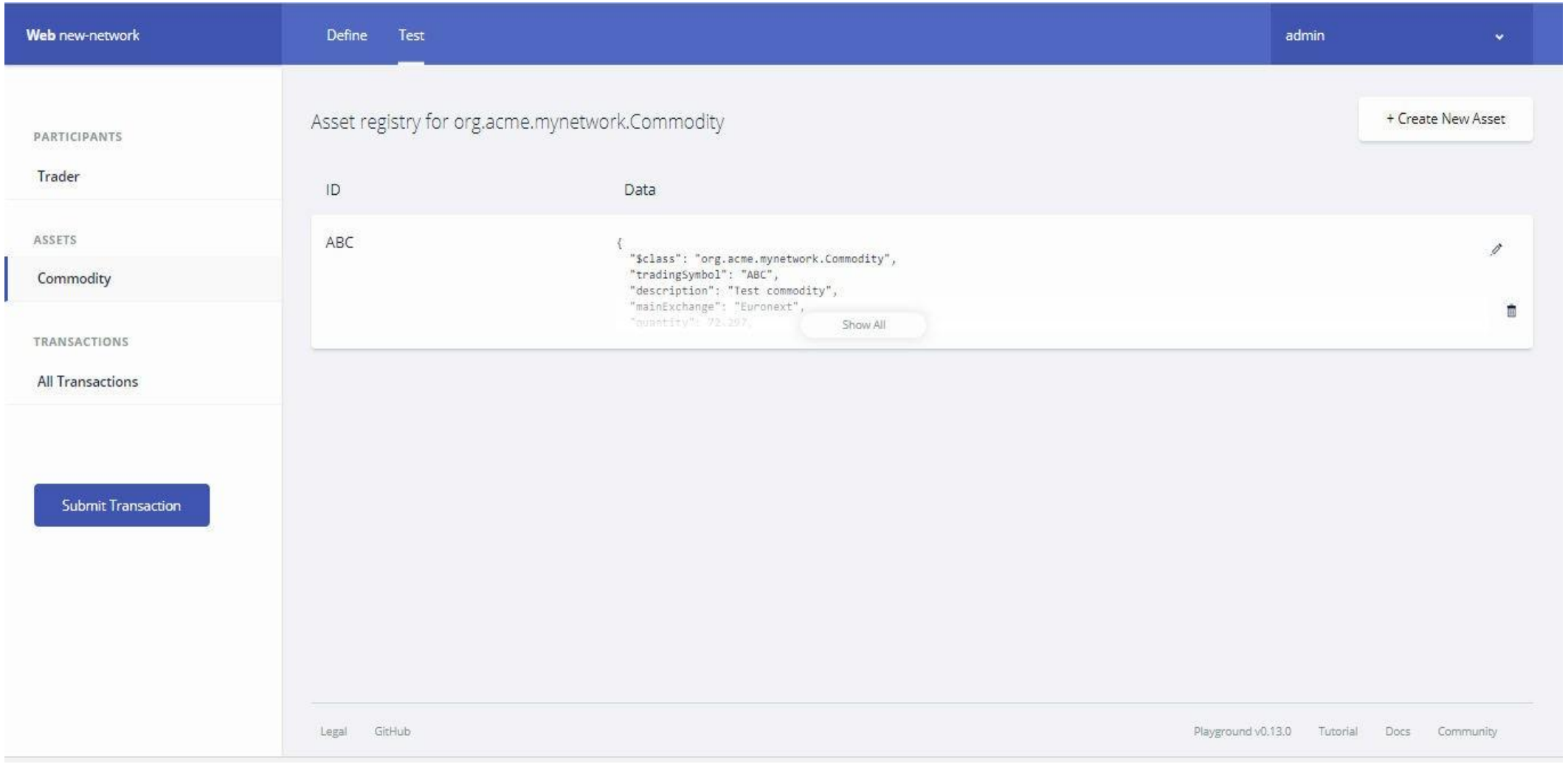
Step Ten: Creating an asset

Now that we have two Trader participants, we need something for them to trade. Creating an asset is very similar to creating a participant. The Commodity we're creating will have an owner property indicating that it belongs to the Trader with the tradeId of TRADER1

- Click the Commodity tab under Assets and click Create New Asset.
- Delete the asset data and replace it with the following:

```
{
   "$class": "org.acme.mynetwork.Commodity", "tradingSymbol": "ABC",
   "description": "Test commodity", "mainExchange": "Euronext",
   "quantity": 72.297,
   "owner": "resource:org.acme.mynetwork.Trader#TRADER1 "
}
```

Asset created



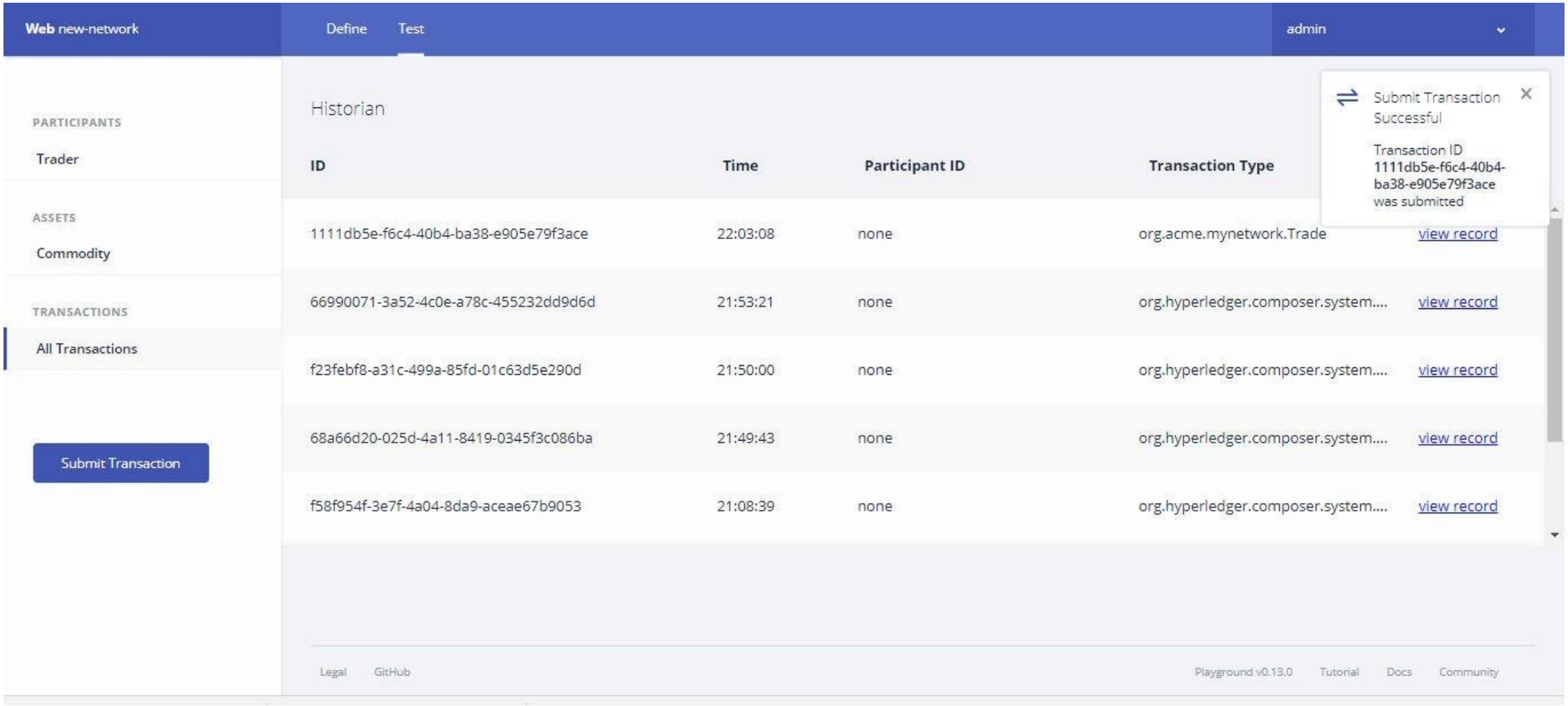
Step Eleven: Transferring the commodity

Now that we have two Traders and a Commodity to trade between them, we can test our Trade transaction

- To test the Trade transaction:
- Click the Submit Transaction button on the left
- Ensure that the transaction type is Trade
- Replace the transaction data with the following, or just change the details:
- Click Submit

```
{
"$class":  "org.acme.mynetwork.Trade", "commodity":
"resource:org.acme.mynetwork.Commodity#ABC", "newOwner":
"resource:org.acme.mynetwork.Trader#TRADER2"
}
```

Submit transaction successful



Change in Ownership

To check that our asset has changed ownership from TRADER1 to TRADER2, click the Commodity tab, and expand the data section for the asset. You should see that the owner is listed as:
resource:org.acme.mynetwork.Trader#TRADER2

Check all Transactions

- To view the full transaction history of our business network, click **All Transactions** on the left

- You can see that certain actions we performed using the UI, like creating the Trader participants and
- the Commodity asset, are recorded as transactions, even though they're not defined as transactions in our business network model. These transactions are known as 'System Transactions' and are common to all business networks, and defined in the Hyperledger Composer Runtime

Thank You

Email us – support@intellipaat.com

Visit us - https://intellipaat.com