Part 1: Hyperledger Composer
What is Hyperledger Composer?

- Blockchains provide a low-level interface for business applications
  - Smart contract code run on a distributed processing system
  - Inputs go into an immutable ledger; outputs to a data store
  - Applications are built on top of a low level of abstraction

- Hyperledger Composer
  - A suite of high level application abstractions for business networks
  - Emphasis on business-centric vocabulary for quick solution creation

- Features
  - Model your business network, test and deploy
  - Applications use APIs to interact with a business network
  - Integrate existing systems of record using loopback/REST

- Open Tools, APIs and libraries to support these activities
  - Exploits Hyperledger Fabric blockchain technology
  - Fully open and part of Linux Foundation Hyperledger

https://hyperledger.github.io/composer/
Benefits of Hyperledger 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
An Example Business Network – Car Auction Market

1. Owner
   - Vehicle
     - VIN
     - Owner
   - Sell car

2. Buyer
   - Offer bid

3. Auctioneer
   - Close bidding

Existing system

Vehicle registry

Transactions
- Offer
- Close Bidding

Listings registry

Counterparties outside the business network
- DVLA
- Insurer

Business Network
- Transactions
- Participants
- Registries
- Identity
- Assets

Builds on
- Peers
- Chaincode
- Consensus
- World state

Hyperledger Composer

Hyperledger Fabric

App builds on Vehicle
- VIN
- Owner

Vehicle Listing
- Listing ID
- Reserve Price
- Offers
- Description
- Linked to Vehicle

©2017 IBM Corporation
Conceptual Components and Structure of Composer

Business Network is defined by Models, Script Files, ACLs and Metadata and packaged in a Business Network Archive

Solution Developer models the business network, implements the script files that define transaction behaviour and packages into a business network archive

Solution Administrator provision the target environment and may manage deploy

Business Network Archive

- Models
- Script File
- ACLs
- Metadata
Extensive, Familiar, Open Development Toolset

- Data modelling
- JavaScript
- Business logic
- Web playground
- Client libraries
- Editor support
- CLI utilities
- Code generation
- Existing systems and data

$ composer
- composer-client
- composer-admin

JavaScript

Data modelling

Web playground

Client libraries

Editor support

CLI utilities

Code generation

Existing systems and data
Part 2: Blockchain fabric development
Actors in a Blockchain Solution

- **Blockchain Architect** designs the architecture
- **Blockchain Developer** creates applications
- **Blockchain Network Operator** operates the network
- **Regulator** performs oversight on transactions
- **Blockchain User** accesses B2B transactions
- **Membership Services** ensures secure access to data
- **Traditional Data Sources** provide access to data
- **Traditional Processing Platforms** access to logic

Blockchain plays a central role in connecting these actors.
Actors in a Blockchain Solution

- **Blockchain Architect**: Responsible for the architecture and design of the blockchain solution.
- **Blockchain User**: The business user, operating in a business network. This role interacts with the Blockchain using an application. They are not aware of the Blockchain.
- **Blockchain Regulator**: The overall authority in a business network. Specifically, regulators may require broad access to the ledger’s contents.
- **Blockchain Developer**: The developer of applications and smart contracts that interact with the Blockchain and are used by Blockchain users.
- **Blockchain Operator**: Manages and monitors the Blockchain network. Each business in the network has a Blockchain Network operator.
- **Membership Services**: Manages the different types of certificates required to run a permissioned Blockchain.
- **Traditional Processing Platform**: An existing computer system which may be used by the Blockchain to augment processing. This system may also need to initiate requests into the Blockchain.
- **Traditional Data Sources**: An existing data system which may provide data to influence the behavior of smart contracts.
Components in a Blockchain Solution

- **Ledger**: A ledger is a channel’s chain and current state data which is maintained by each peer on the channel.

- **Smart Contract**: Software running on a ledger, to encode assets and the transaction instructions (business logic) for modifying the assets.

- **Peer Network**: 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.

- **Membership**: Membership Services authenticates, authorizes, and manages identities on a permissioned blockchain network.

- **Events**: Creates notifications of significant operations on the blockchain (e.g. a new block), as well as notifications related to smart contracts.

- **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.
Blockchain developers’ primary interests are…

…and how they interact with the ledger and other systems of record:

They should NOT have to care about operational concerns, such as:

- Peers
- Consensus
- Security
How the Developer Interacts with the Ledger

A ledger often consists of two data structures

- **Blockchain**
  - A linked list of blocks
  - Each block describes a set of transactions (e.g. the inputs to a smart contract invocation)
  - Immutable – blocks cannot be tampered

- **World State**
  - An ordinary database (e.g. key/value store)
  - Stores the combined outputs of all transactions
  - Not usually immutable
Working with the Ledger: Example of a Change of Ownership Transaction (change car1 owner to Matt)

Transaction input - sent from application

```
invoke(myContract, setOwner, myCar, Matt)
```

Smart contract implementation

```
setOwner(Car, newOwner) {
    set Car.owner = newOwner
}
```

World state: new contents

```
myCar.vin = 1234
myCar.owner = Matt
myCar.make = Audi
```
Integrating with Existing Systems

1. System events

2. Blockchain events

3. Call into blockchain network from existing systems

4. Call out to existing systems

Blockchain network

Existing systems
Part 3: Blockchain architecture
The Blockchain Administrator (Operator)

Blockchain administrators’ primary interests are in the deployment and operation of part of the blockchain:

- Peers
- Consensus
- Security

They should NOT have to care about development concerns, such as:

- Application code
- Smart contract code
- Events and integration
Consensus: The Process of Maintaining a Consistent Ledger

Keep all peers up to date.

Fix any peers in error.

Ignore all malicious nodes.
Consensus: Typical Flow of Execution

- Details vary significantly between blockchain implementations, but a typical flow is:

  - The application submits a request to invoke a transaction
  - The transaction is shared around the network
  - A designated peer creates a block containing the transaction
  - The block’s transactions are executed and output stored in a delta
  - The network attempts to agree on the correct result
  - If there is agreement, the correct output is applied to the world state

- The process to agree the consistent state of the ledger is known as consensus
Some Examples of Consensus Algorithms

- Proof of work
- Proof of stake
- Solo
- Kafka/Zookeeper
- Proof of Elapsed Time
- PBFT-based
Consensus Algorithms have Different Strengths and Weaknesses

**Proof of work**

Require validators to solve difficult cryptographic puzzles

**PROs:** Works in untrusted networks

**CONS:** Relies on energy use; slow to confirm transactions

Example usage: Bitcoin, Ethereum

**Proof of stake**

Require validators to hold currency in escrow

**PROs:** Works in untrusted networks

**CONS:** Requires intrinsic (crypto)currencies, "Nothing at stake" problem

Example usage: Nxt

**Wait time in a trusted execution environment randomizes block generation**

**PROs:** Efficient

**CONS:** Currently tailored towards one vendor

Example usage: Sawtooth-Lake
Consensus Algorithms have Different Strengths and Weaknesses

Validators apply received transactions without consensus
PROs: Very quick; suited to development
CONS: No consensus; can lead to divergent chains
Example usage: Hyperledger Fabric V1

Practical Byzantine Fault Tolerance implementations
PROs: Reasonably efficient and tolerant against malicious peers
CONS: Validators are known and totally connected
Example usage: Hyperledger Fabric V0.6

Ordering service distributes blocks to peers
PROs: Efficient and fault tolerant
CONS: Does not guard against malicious activity
Example usage: Hyperledger Fabric V1
Security: Public vs. Private Blockchains

Public blockchains

- For example, Bitcoin
- Transactions are viewable by anyone
- Participant identity is more difficult to control

Private blockchains

- For example, Hyperledger Fabric
- Network members are known but transactions are secret

• Some use cases require anonymity, others require privacy
  – Some may require a mixture of the two, depending on the characteristics of each participant

• Most **business** use cases require private, permissioned blockchains
  – Network members know who they’re dealing with (required for KYC, AML, etc.)
  – Transactions are (usually) confidential between the participants concerned
  – Membership is controlled
Certificate Authorities and Blockchain

- Certificate Authority issues certificates to Client Application SDK, which signs/encrypts transactions.
- Client Application SDK uses the certificates to request certificates from the Certificate Authority.
- Client Application SDK verifies/decrypts transactions using Blockchain.
- User A signs/encrypts transactions with Blockchain.
- User B uses Blockchain to verify/decrypt transactions.
Other Nonfunctional Requirements

• Performance
  – The amount of data being shared
  – Number and location of peers
  – Latency and throughput
  – Batching characteristics

• Security
  – Type of data being shared, and with whom
  – How is identity achieved
  – Confidentiality of transaction queries
  – Who verifies (endorses) transactions

• Resiliency
  – Resource failure
  – Malicious activity
  – Non-determinism

Consider the trade-offs between performance, security, and resiliency!
### Nodes and Roles

<table>
<thead>
<tr>
<th><strong>Committing Peer</strong></th>
<th>Maintains ledger and state. Commits transactions. May hold smart contract (chaincode).</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Endorsing Peer</strong></td>
<td>Specialized committing peer that receives a transaction proposal for endorsement, responds granting or denying endorsement. Must hold smart contract</td>
</tr>
<tr>
<td><strong>Ordering Nodes (service)</strong></td>
<td>Approves the inclusion of transaction blocks into the ledger and communicates with committing and endorsing peer nodes. Does not hold smart contract. Does not hold ledger.</td>
</tr>
</tbody>
</table>
Sample Transaction: Step 1/7 – Propose Transaction

Application proposes transaction

Endorsement policy:
• “E₀, E₁ and E₂ must sign”
• (P₃, P₄ are not part of the policy)

Client application submits a transaction proposal for Smart Contract A. It must target the required peers {E₀, E₁, E₂}.

Key:
- Endorser
- Ledger
- Committing Peer
- Application
- Ordering Node
- Smart Contract (Chaincode)
- Endorsement Policy

Client Application Submits a Transaction Proposal for Smart Contract A. It Must Target the Required Peers {E₀, E₁, E₂}.
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 Read and Written data, called RW sets, which will now flow in the fabric.

Transactions can be signed and encrypted.

Key:

- **Endorser**
- **Ledger**
- **Committing Peer**
- **Application**
- **Ordering Node**
- **Smart Contract (Chaincode)**
- **Endorsement Policy**
Sample Transaction: Step 3/7 – Proposal Response

Application receives responses

RW sets are asynchronously returned to application.

The RW sets are signed by each endorser, and also includes each record version number.

This information will be checked much later in the consensus process.

Key:

- Endorser
- Ledger
- Committing Peer
- Application
- Ordering Node
- Smart Contract (Chaincode)
- Endorsement Policy
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 proposed blocks for distribution to committing peers. Peers can deliver to other peers in a hierarchy (not shown).

Different ordering algorithms available:
- SOLO (Single node, development)
- Kafka (Crash fault tolerance)

Key:
- Endorser
- Committing Peer
- Ordering Node
- Smart Contract (Chaincode)
- Ledger
- Application
- Endorsement Policy
Committing peers validate transactions

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

Validated transactions are applied to the world state and retained on the ledger.

Invalid transactions are also retained on the ledger but do not update world state.

Key:

- **Endorser**
- **Ledger**
- **Comitting Peer**
- **Application**
- **Ordering Node**
- **Endorsement Policy**
- **Smart Contract (Chaincode)**
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.

Key:

- Endorser
- Committing Peer
- Ordering Node
- Smart Contract (Chain code)
- Ledger
- Application
- Endorsement Policy

©2017 IBM Corporation