

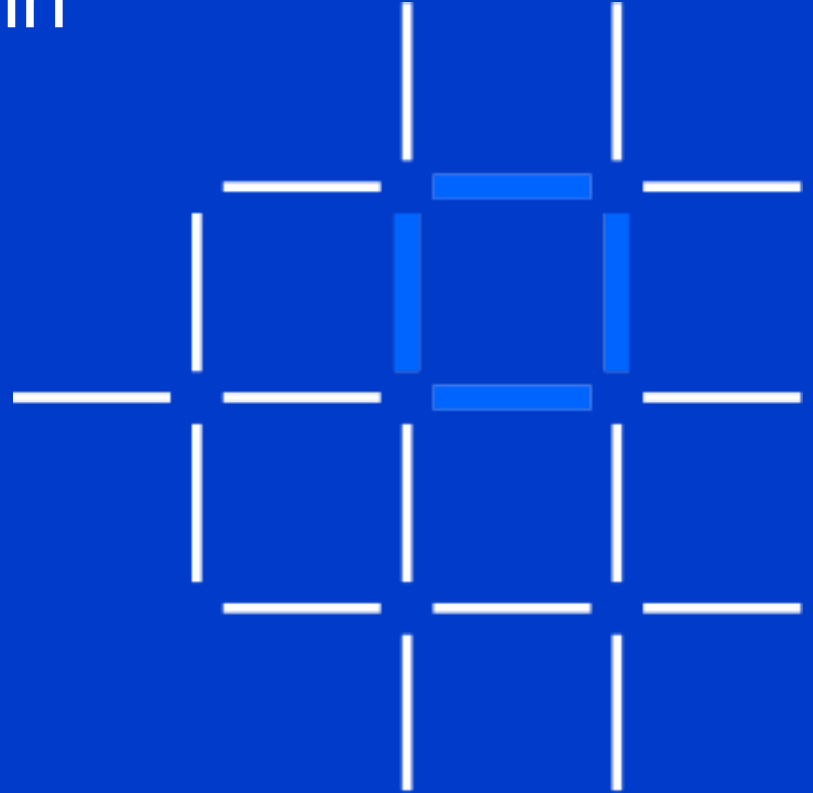
# Create and Deploy a Blockchain App with the IBM Blockchain Platform - Demo

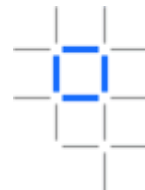
*Lennart Frantzell,  
IBM Developer Advocate,  
San Francisco  
alf@us.ibm.com*

**ibm.biz/lacollision**

V1 29 April 2018

IBM **Blockchain**



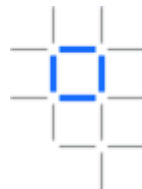


# **ibm.biz/lacollision**

Blockchain Collision Intro.pdf

Blockchain Car Auction.pdf

## **Survey: ibm.biz/bcfeedback**



**Harvard  
Business  
Review**

**INTERNATIONAL BUSINESS**

# The Promise of Blockchain Is a World Without Middlemen

by **Vinay Gupta**

MARCH 06, 2017



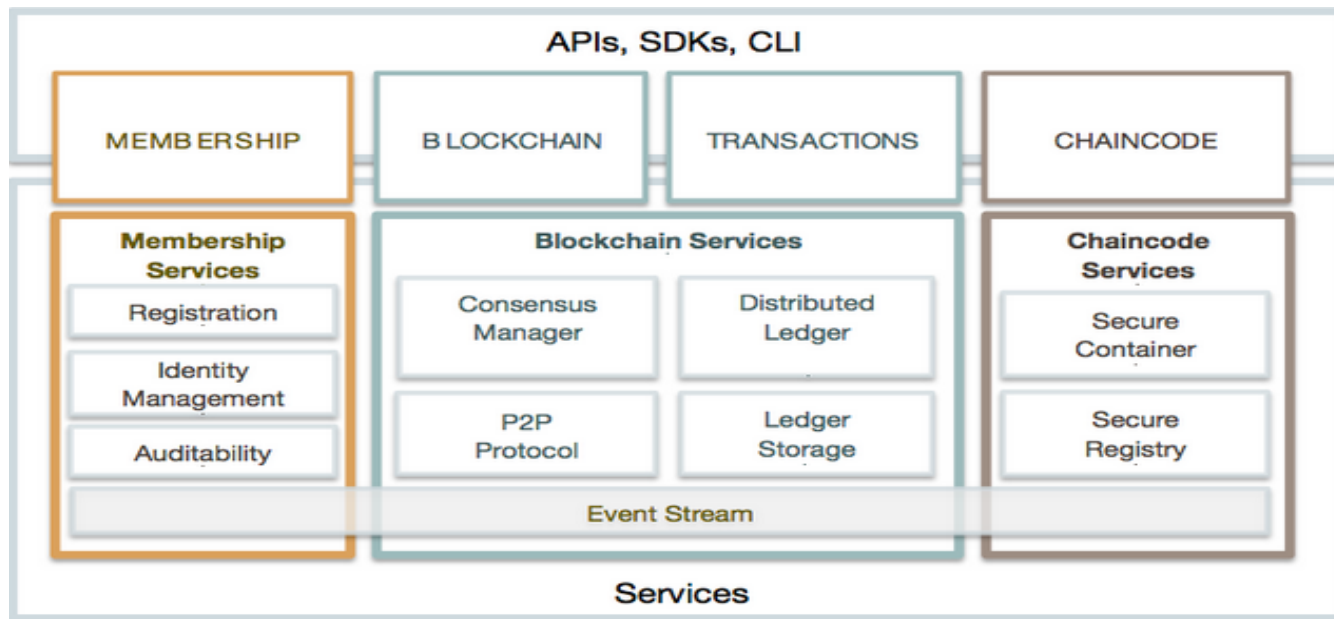
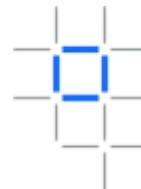
Hyperledger Fabric is an **open source blockchain framework** hosted by The Linux Foundation.

<https://www.hyperledger.org/projects/fabric>

<https://github.com/hyperledger/fabric>

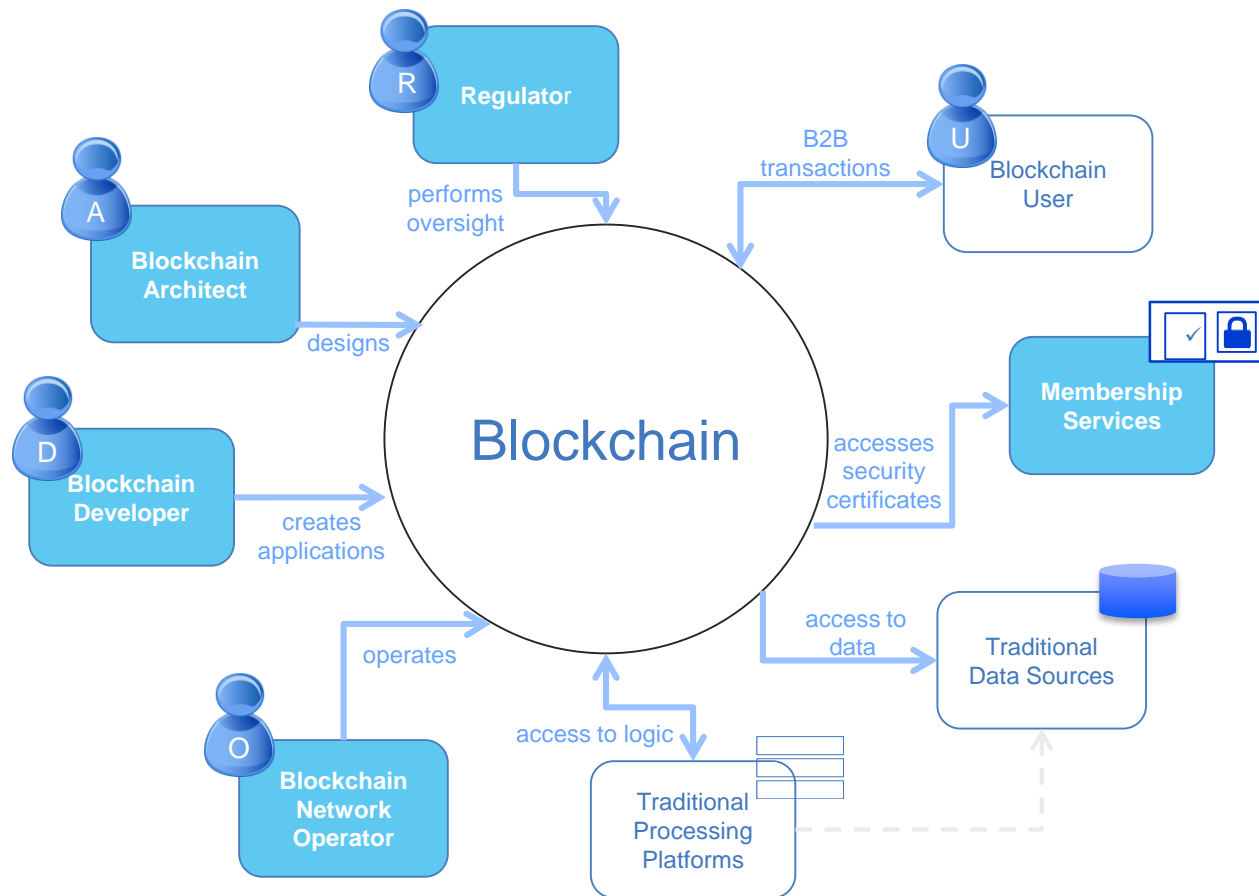
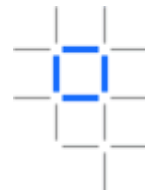
<https://hyperledger-fabric.readthedocs.io/en/release-1.1/>

# Hyperledger Fabric overview

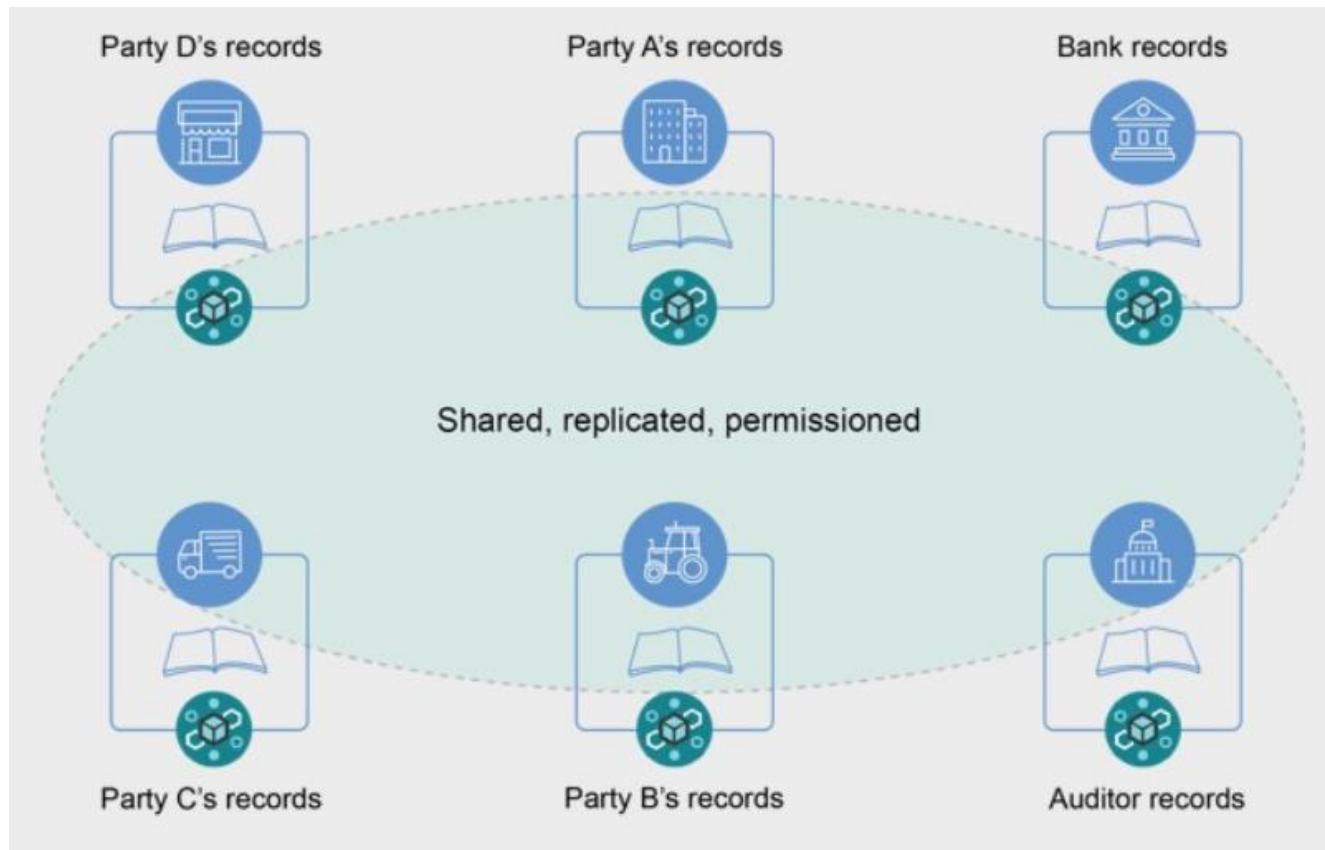
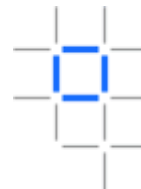


- Hyperledger Fabric allows components, such as consensus and membership services, to be **plug-and-play**.
- Hyperledger Fabric leverages **container technology** (Docker) to host **smart contracts called “chaincode”** that comprise the application logic of the system.
- Chaincode is like Stored Procedures in the traditional database world

# Actors in a Blockchain solution



# The shared Ledger



# A Blockchain is:



- A **decentralized virtual ledger** for recording transactions **without central authority** through a **distributed cryptographic protocol**.
- It is made up of three technologies
  - cryptographic algorithms,
  - a distributed protocol, (gRPC)
  - and replicated data which combined provide a trustworthy service to a group of nodes that do not fully trust each other.
- Source: [https://www.zurich.ibm.com/dccl/papers/cachin\\_dccl.pdf](https://www.zurich.ibm.com/dccl/papers/cachin_dccl.pdf)

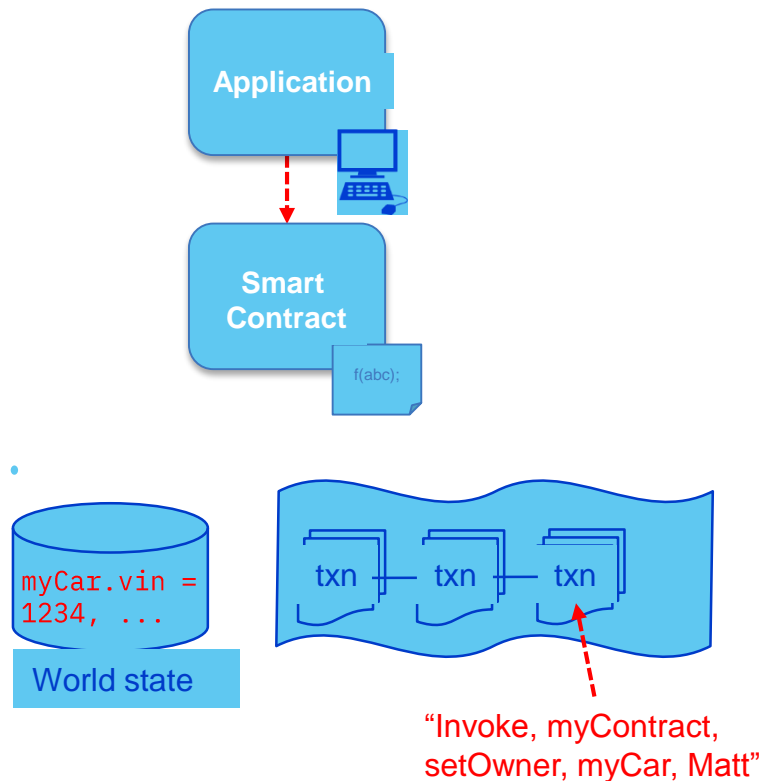
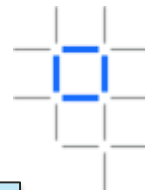




- **Chaincode is a piece of code that is deployed into a network of Hyperledger fabric peer nodes that enables interaction with that network's shared ledger.**
- Chaincode is a program, written in Go, node.js, and eventually in other programming languages such as Java, that implements a prescribed interface.
- **A chaincode typically handles business logic agreed to by members of the network**, so it may be considered as a “smart contract”.

<http://hyperledger-fabric.readthedocs.io/en/release-1.1/chaincode.html>

# Working with the ledger example: a change of ownership transaction



Transaction input - sent from application

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

...

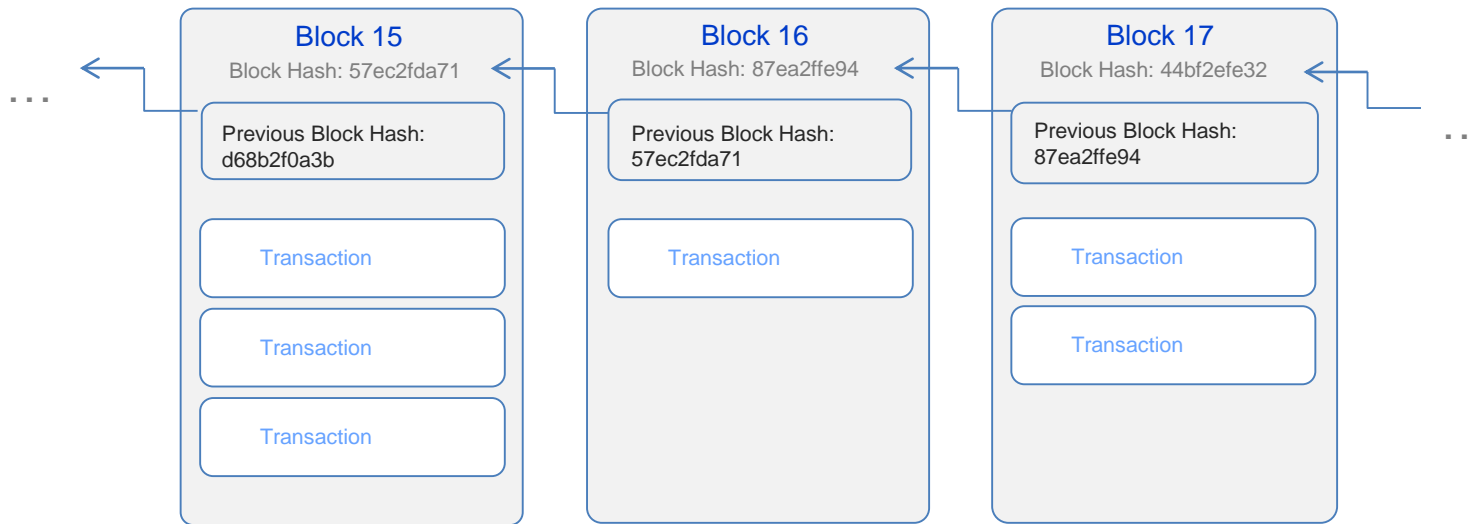
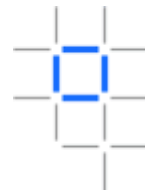
Smart contract implementation

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

World state: new contents

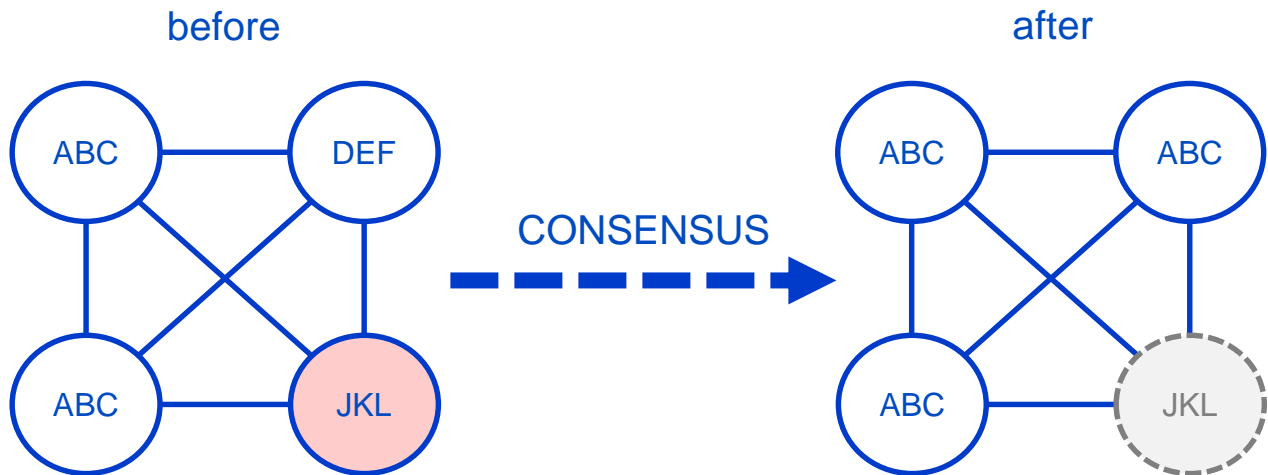
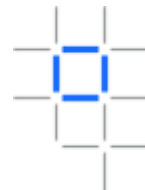
```
myCar.vin = 1234  
myCar.owner = Matt  
myCar.make = Audi  
...
```

# Block detail (simplified)



- A blockchain is made up of a series of blocks with new blocks always added to the end
- Each block contains zero or more transactions and some additional metadata
- Blocks achieve immutability by including the result of a hash function of the previous block
- The first block is known as the “genesis” block

# Consensus: The process of maintaining a consistent ledger



Keep all peers up-to-date  
Fix any peers in error  
Ignoring all malicious nodes



# Consensus algorithms have different strengths and weaknesses



Solo /  
No-ops

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



PBFT-based

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



Kafka /  
Zookeeper

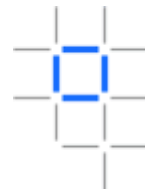
Ordering service distributes blocks to peers

**PROs:** Efficient and fault tolerant

**CONS:** Does not guard against malicious activity

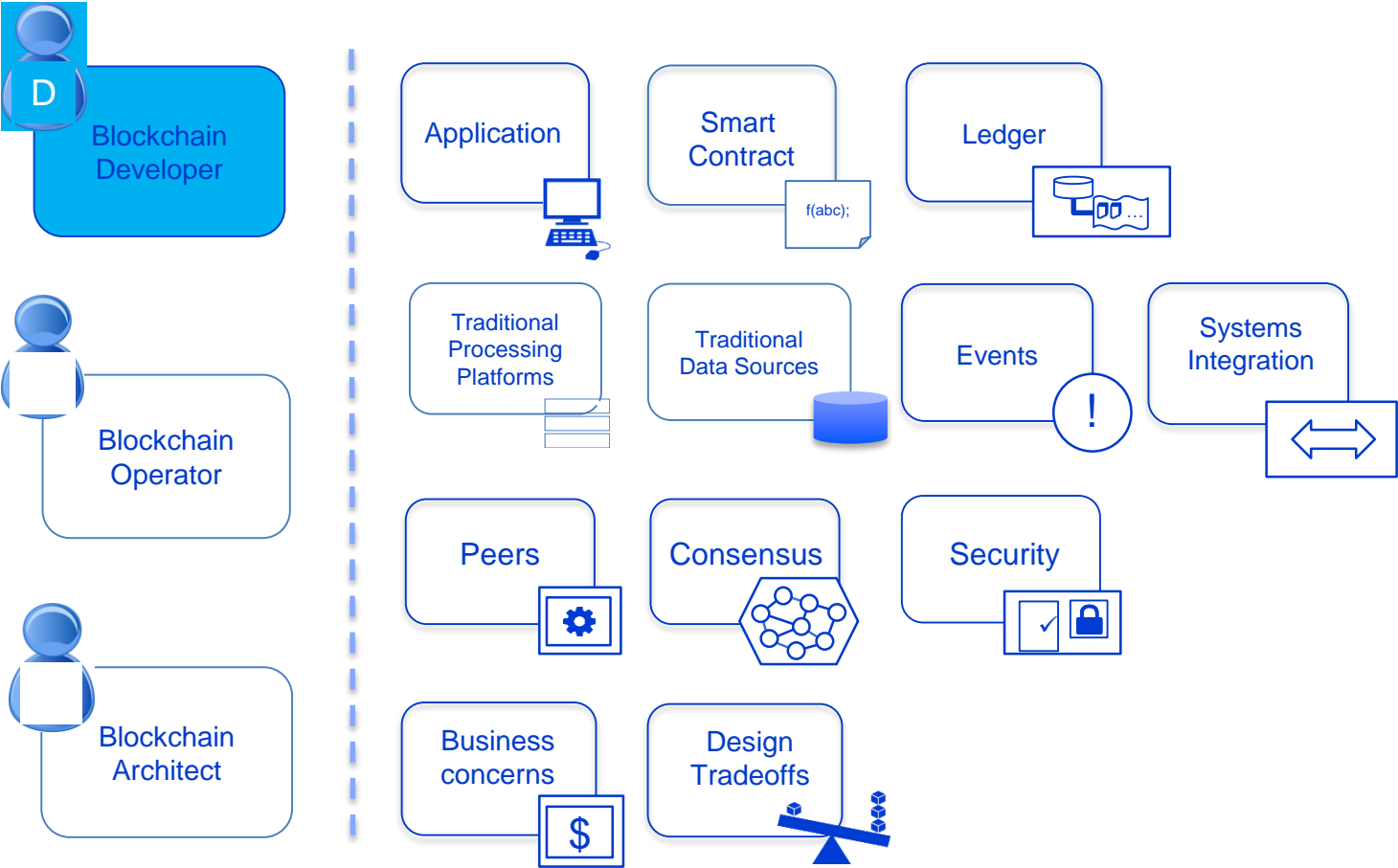
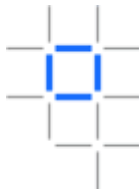
Example usage: Hyperledger Fabric V1

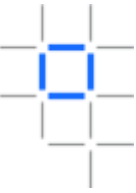
# Security: Encryption and Signing



- Cryptography basics
  - Every member of the network has (at least) one public key and one private key
  - Assume that every member of the network knows all public keys and only their own private keys
  - Encryption is the process applying a transformation function to data such that it can only be decrypted by the other key in the public/private key pair
  - Users can sign data with a private key; others can verify that it was signed by that user
- For example
  - **Alice can sign a transaction with her private key such that anyone can verify it came from her**
  - **Anyone can encrypt a transaction with Bob's public key; only Bob's private key can decrypt it**
- In private, permissioned blockchains
  - Transactions and smart contracts can be signed to verify where they originated
  - Transactions and their payloads can be encrypted such that only authorized participants can decrypt

# Summary of Key Concepts

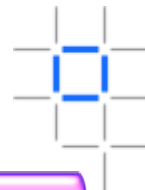




<https://www.hyperledger.org/projects/composer>

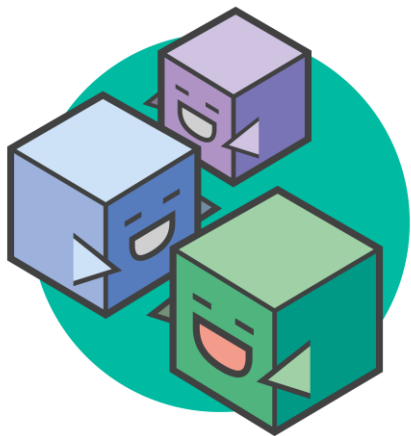


# Hyperledger Composer: Accelerating time to value

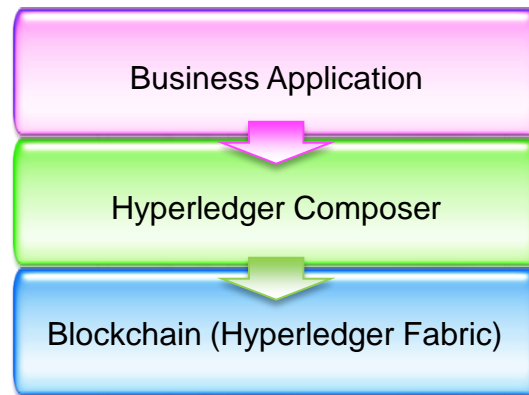


<https://hyperledger.github.io/composer/>

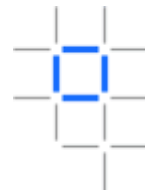
- A suite of high level application **abstractions** for business networks
- Emphasis on business-centric vocabulary for quick solution creation
- Reduce risk, and increase understanding and flexibility



- Features
  - Model your business networks, test and expose via APIs
  - Applications invoke APIs transactions to interact with business network
  - Integrate existing systems of record using loopback/REST
- Fully open and part of Linux Foundation Hyperledger
- Try it in your web browser now: <http://composer-playground.mybluemix.net/>



# Benefits of Hyperledger Composer



## **Increases understanding**

Bridges simply from  
business concepts to  
blockchain



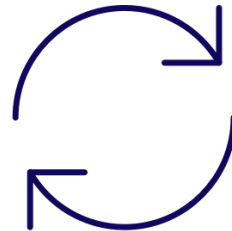
## **Saves time**

Develop blockchain  
applications more  
quickly and cheaply



## **Reduces risk**

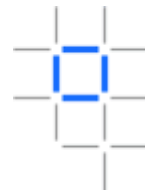
Well tested, efficient  
design conforms to  
best practice



## **Increases flexibility**

Higher level  
abstraction makes it  
easier to iterate

# Extensive, Familiar, Open Development Toolset



```
asset Animal identi
  o String animal
  o AnimalType sp
  o MovementStatu
  o ProductionTyp
```

Data modelling



JavaScript  
business logic



Web playground

```
composer-client
composer-admin
```



Client libraries



Editor support

```
$ composer
```

CLI utilities



Code generation

Powered by



LoopBack  
Node.js Framework



Swagger

Existing systems and  
data



What is Hyperledger  
Composer?



Application Development

*Writing the application*

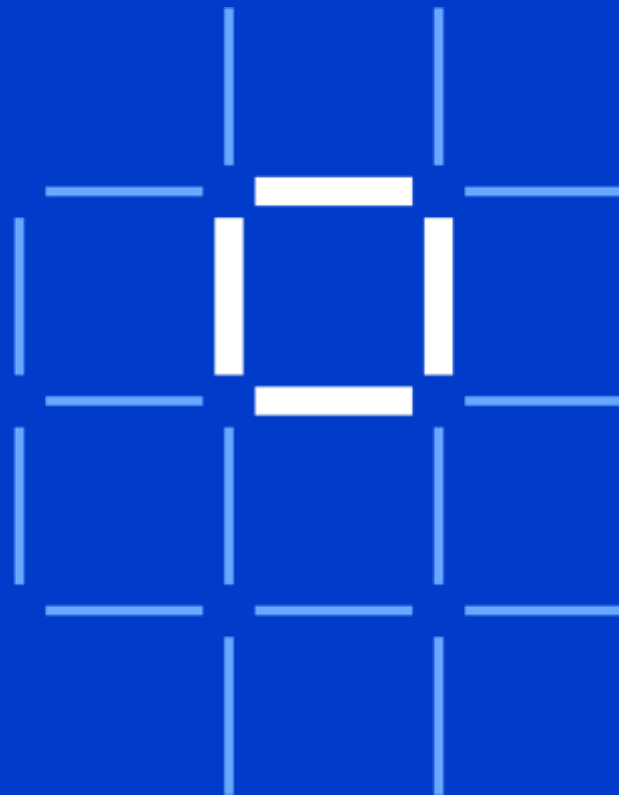
*Modeling the business network*



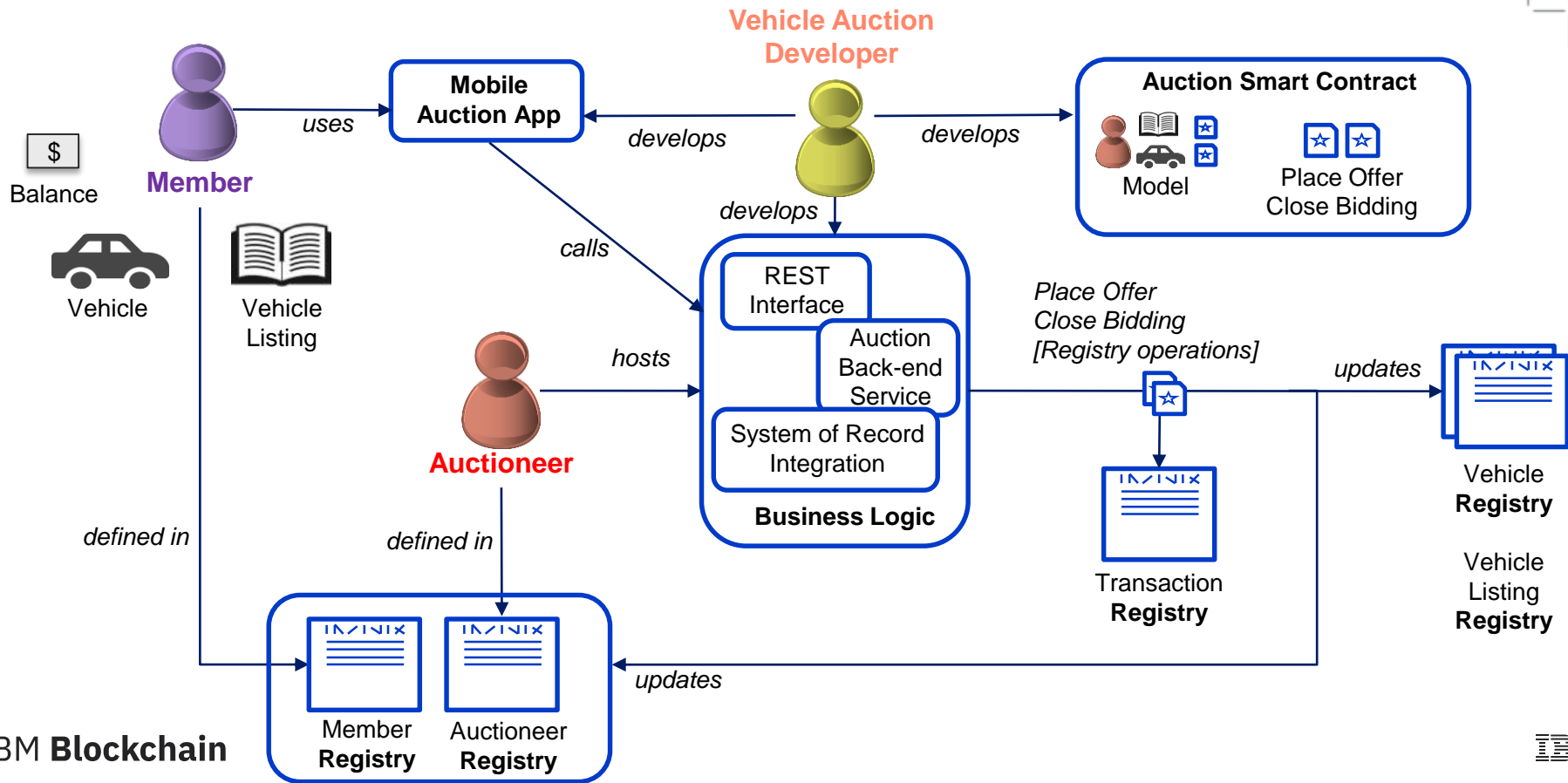
Effective Administration

*Deploying to a blockchain*

*Interacting with systems of record*



# Key Concepts for a Vehicle Auction Developer

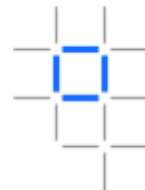





Deploy at IBM  
Blockchain Platform in  
the IBM Cloud



# IBM Blockchain Starter Plan Beta



Blockchain /

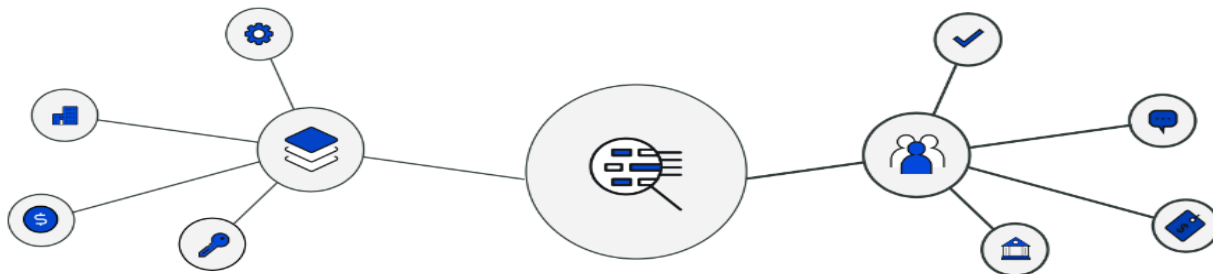
 Blockchain-BostonWed

Location: US South

Org: Developer Advocacy

Space: dev

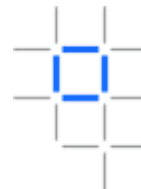
Welcome back, alf



Use the Network Monitor to view and add network nodes, create channels, install chaincode applications, and test confidential transactions.

Enter Monitor

# Overview



## Overview

View and manage network resources for your organization. You can stop or start your resources and view the log file of the resource by selecting View Logs under "Actions". [Learn more](#)

↕ Connection Profile

| <input type="checkbox"/> | TYPE    | NAME       | STATUS    |
|--------------------------|---------|------------|-----------|
| <input type="checkbox"/> | Orderer | orderer    | ● Running |
| <input type="checkbox"/> | CA      | org1-ca    | ● Running |
| <input type="checkbox"/> | Peer    | org1-peer1 | ● Running |







# Members

View and manage the members (organizations) of the network on the Members tab. You can invite other organizations to the network or add more organizations in addition to the two organizations that you own by default. You can also view and manage admin certificates that are associated to your organizations on the Certificates tab. [Learn more](#)


Members      Certificates

 Add Member


| MEMBERS (2/16)              | MSP ID | REQUESTER | STATUS   | ACTION  |
|-----------------------------|--------|-----------|--|---|
| Company A<br>alf@us.ibm.com | org1   | --        |  Joined |  |
| Company B<br>alf@us.ibm.com | org2   | --        |  Joined |  |

# Channels

You must have a channel to propose a transaction. If you're not a member of any channels, you can create one by clicking "Request Channel". [Learn more](#)

 Search Channels

Request Channel

| ID             | TIME CREATED       | BLOCK HEIGHT | PEERS      | ACTIONS   |
|----------------|--------------------|--------------|------------|---|
| defaultchannel | 04/18/18 10:39 PDT | 23           | org1-peer1 |  |
|                |                    |              |            |   |

\* Channels created with the SDK cannot be edited here (yet)

# APIs

Interact with the network by using APIs in the Swagger UI. You can also use the network credentials and integrate the APIs to your own application. [Learn more](#)



## API reference list

Use the Swagger UI to try out the available catalog of REST APIs against the network.

[Swagger UI](#)

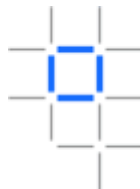


## Network credentials

Use these network credentials to access the resource endpoints that your application needs to. Use "key" as your "Username" and "secret" as your "Password" when you authorize your APIs in the Swagger UI.

```
{
  "url": "https://ibmblockchain-starter.ng.bluemix.net",
  "network_id": "nabbe2cb650394e3794e233bd1b5153c3",
  "key": "org1",
  "secret": 
}
```

[Show secret](#)



## Notifications

You get a notification whenever a creation or update request for a channel that you are included is submitted. Review, vote, and subscribe to channel requests with the buttons under "ACTION". [Learn more](#)

All (2)

Pending (1)

Completed (1)



Search notifications



NAME ▾

DATE UPDATED ▾

MY STATUS ▾



Channel Request  
Join "bostonchannel"

By: Company A  
20 April, 2018 - 11:44:36 AM

● Vote Accepted



Channel Request  
Join "defaultchannel"

By: Company A  
18 April, 2018 - 10:39:28 AM

● Vote Accepted

# Write code

You can develop your blockchain use case with IBM Blockchain Platform development environment, and then deploy the use case back to this network. [Learn more.](#)



## Develop code

The IBM Blockchain Platform provides a free development environment based on industry standard tools and technologies (Hyperledger Composer, Javascript, Docker, Yeoman, and more) that you can use to model and code a ready-to-deploy use case.

You can trial this environment online or install it on your local machine. Learn more and download the tools at our website.

[Visit website](#)




## Deploy code

After you create your blockchain use case, you can deploy it to this network in the cloud platform following the Deployment Guide.

[Deployment guide](#)

# Install code

Chaincode must be installed on a peer. Select a peer and then install a chaincode on it. After you install the chaincode, you can request to instantiate it on a channel by clicking the Instantiate button under "Actions" for that chaincode. Clicking elsewhere on the chaincode will show you what channels the chaincode is instantiated on. [Learn more](#). Do you already have a .bna file to deploy? See our [deployment guide](#).

Choose peer...

 Install Chaincode

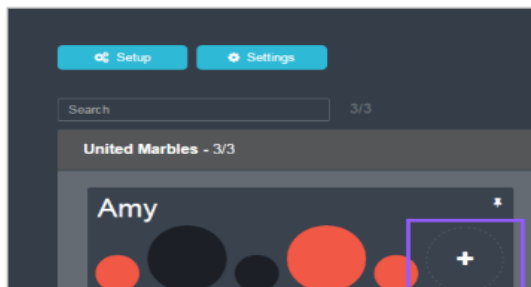
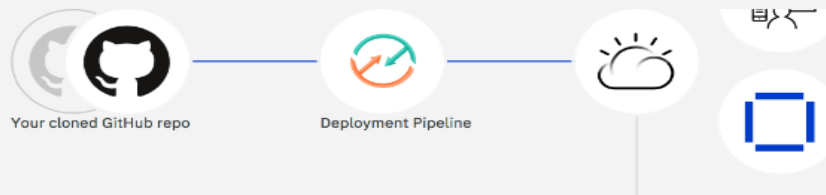
| CODE ID        | VERSION      | ACTIONS   |
|----------------|--------------|---|
| marbles_sample | 3a30a0df1829 |  |

# Try samples

Sample applications provide you a better understanding of blockchain networks. You can also develop your own apps by installing our developer tools. This sample deployment approach leverages IBM Cloud Toolchains and helps to deploy sample applications, including chaincode, front-end apps, and a dev-ops pipeline in a single step. [Learn more](#)

Cloud Toolchain. Each sample deployment comes with source control, deployment pipeline, and the "chaincode" running on your blockchain network.

After you deploy the sample, any changes you push to the source control are automatically deployed to your Blockchain Platform service via the deployment pipeline.



## Marbles

Deployed on 04/20/18

[Toolchain Management](#)

Remove

Launch

## Vehicle Manufacture



- Track a new car from order to delivery
- Composer model and JavaScript
- Angular web app

Deploy via Toolchain

[View on GitHub](#)



## Build Proposal



The invocation needs to be signed and packaged as a proposal. This is done here, in the Marbles application.

## Endorsement



Next, we send the transaction proposal to our peer for endorsement. The peer will simulate the transaction, verify the proposal, and then sign the proposal. It is then sent back to Marbles.

## Ordering



Then Marbles will send our endorsed proposal to the orderer. The orderer will sequence transactions from throughout the network including ours.

## Validate & Commit



The block containing our transaction is sent from the orderer to our peer. Finally it is validated by the peer and then committed to the peer's ledger.

\* Note the speed of this process is slowed down for demonstration purposes

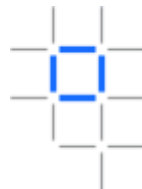
## Invoke Complete

Close



# Support

Get help with your blockchain network. [Learn more](#)

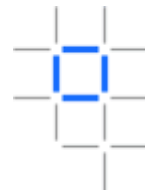


## Support

## Release Notes

- > **GETTING STARTED**
- > **COMMUNITY HELP**
- > **SUPPORT TICKET**
- > **BLOCKCHAIN SAMPLE APPLICATIONS**
- > **HYPERLEDGER FABRIC**
- > **HYPERLEDGER COMPOSER**

# Where do we go from here?



**[ibm.biz/lacollision](https://ibm.biz/lacollision)**

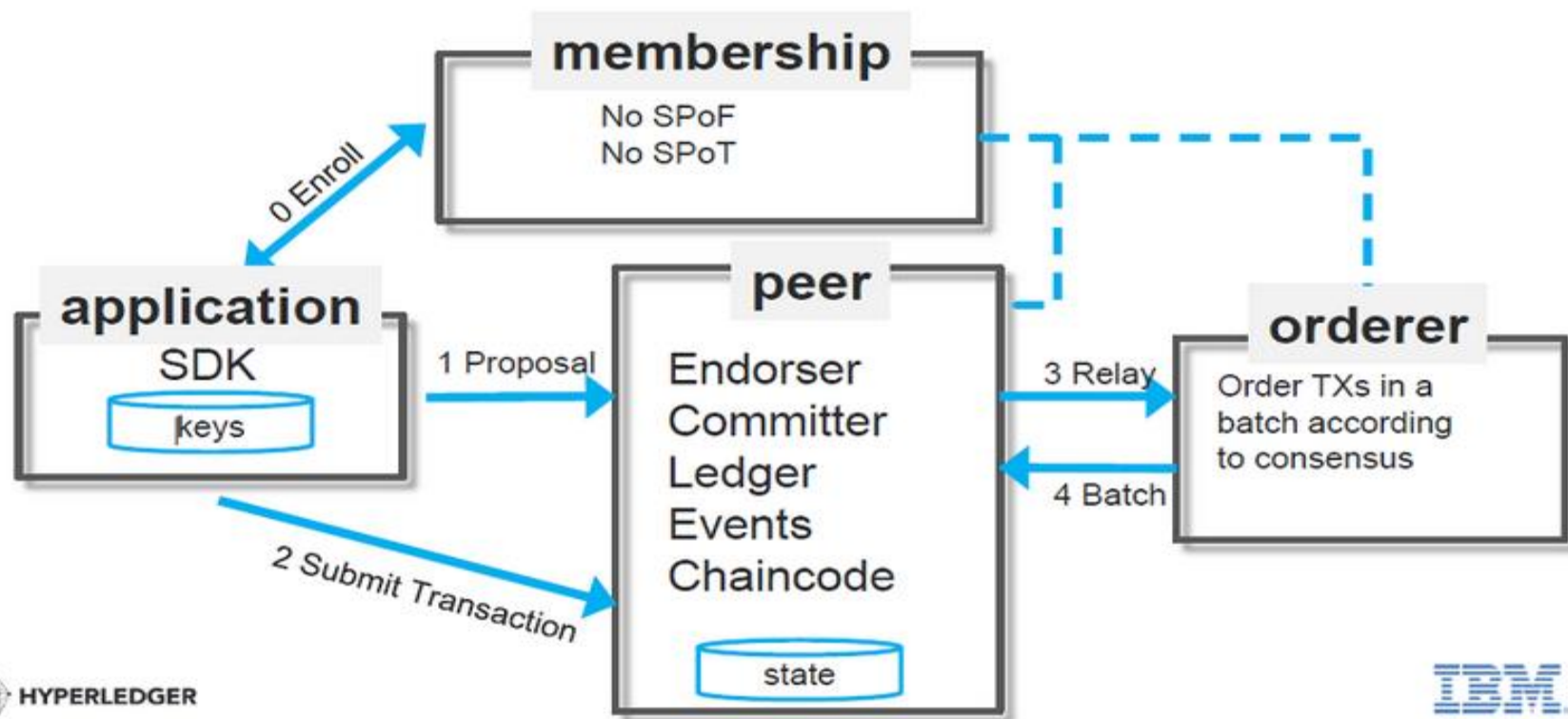
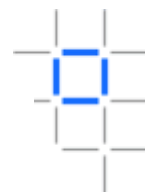
**[ibm.biz/bcfeedback](https://ibm.biz/bcfeedback)**



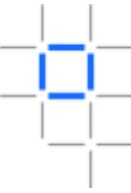
•

# Appendix

# Fabric v1.0 Architecture

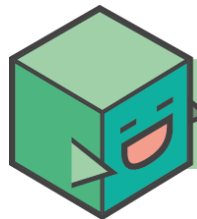
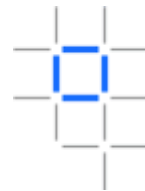


# A Blockchain is:



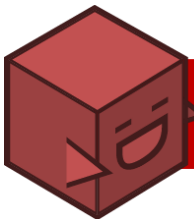
- A distributed ledger is a type of database that is **shared, replicated, and synchronized** among the members of a network.
- The distributed ledger **records the transactions, such as the exchange of assets or data, among the participants in the network.**
- Participants in the network **govern and agree by consensus** on the updates to the records in the ledger.
- **No central, third-party mediator, such as a financial institution or clearinghouse,** is involved.
- Every record in the distributed ledger has a timestamp and **unique cryptographic signature**, thus making the ledger an auditable history of all transactions in the network.

# The Business Service Provider develops three components



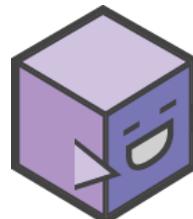
## Smart Contracts

- Implements the logic deployed to the blockchain
  - **Models** describe assets, participants & transactions
  - **Transaction processors** provide the JavaScript implementation of transactions
  - **ACLs** define privacy rules
  - May also define events and registry queries



## Business Logic

- **Services** that interact with the registries
  - Create, delete, update, query and invoke smart contracts
  - Implemented inside business applications, integration logic and REST services
- Hosted by the Business Application Consumer



## Presentation Logic

- Provides the **front-end** for the end-user
  - May be several of these applications
- Interacts with business logic via standard interfaces (e.g. REST)
- Composer can generate the REST interface from model and a sample application

# Assets, Participants and Transactions



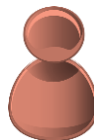
Vehicle



Vehicle Listing



Member



Auctioneer



Place Offer  
Close Bidding

```
asset Vehicle identified by vin {  
  o String vin  
  --> Member owner  
}
```

```
asset VehicleListing identified by listingId {  
  o String listingId  
  o Double reservePrice  
  o String description  
  o ListingState state  
  o Offer[] offers optional  
  --> Vehicle vehicle  
}
```

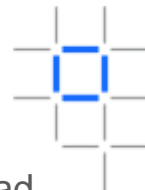
```
abstract participant User identified by email {  
  o String email  
  o String firstName  
  o String lastName  
}  
  
participant Member extends User {  
  o Double balance  
}  
  
participant Auctioneer extends User {  
}
```

```
transaction Offer {  
  o Double bidPrice  
  --> VehicleListing listing  
  --> Member member  
}  
  
transaction CloseBidding {  
  --> VehicleListing listing  
}
```

Transaction  
Processors

```
/**  
 * Close the bidding for a vehicle listing and choose the  
 * highest bid that is  
 * @param {org.acme.vehicle.auction.VehicleListing} listing - the listing  
 * @transaction  
 */  
function closeBidding(listing) {  
  var listing = closeBidding(listing);  
  if (listing.state != 'FOR_SALE') {  
    /**  
     * Make an Offer for a VehicleListing  
     * @param {org.acme.vehicle.auction.Offer} offer - the offer  
     * @transaction  
     */  
    function makeOffer(offer) {  
      var listing = offer.listing;  
      if (listing.state != 'FOR_SALE') {
```

# Access Control



```
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 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  
}
```

```
rule SystemACL {  
  description: "System ACL to permit all access"  
  participant: "org.hyperledger.composer.system.Participant"  
  operation: ALL  
  resource: "org.hyperledger.composer.system.*"  
  action: ALLOW  
}
```

- It is possible to restrict which resources can be read and modified by which participants
  - Rules are defined in an .acl file and deployed with the rest of the model
  - Transaction processors can also look up the current user and implement rules programmatically
- ACL rules can be simple (e.g. everybody can read all resources) or more complex (e.g. only the owner of an asset can do everything to it)
- Application supplies credentials (userid/secret) of the participant when connecting to the Fabric network
  - This also applies to Playground!
  - Remember to grant System ACL all access if necessary



# Events and Queries



- Events allow applications to take action when a transaction occurs
  - Events are **defined** in models
  - Events are **emitted** by transaction processor scripts
  - Events are **caught** by business applications
- Caught events include transaction ID and other relevant information

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

```
// 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);
```

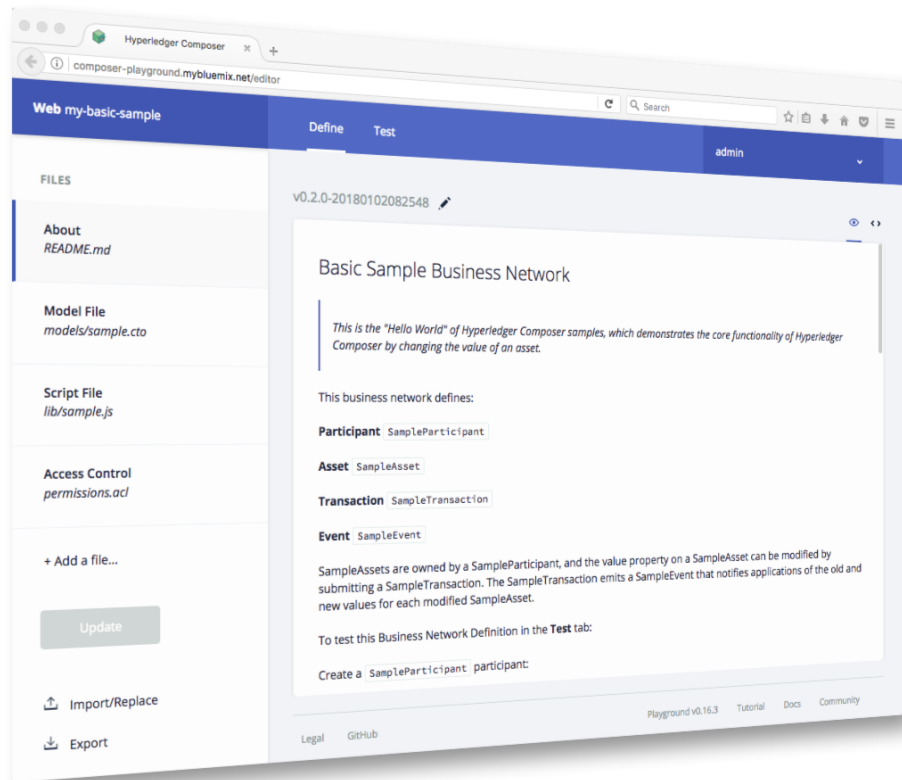
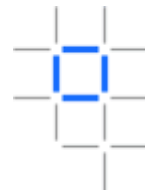
```
businessNetworkConnection.on('SampleEvent', (event) => {  
  console.log(event);  
})
```

- Queries allow applications to perform complex registry searches
  - They can be statically defined in a separate .qry file or generated dynamically by the application
  - They are invoked in the application using *buildQuery()* or *query()*
  - Queries require the blockchain to be backed by CouchDB

```
query selectCommoditiesWithHighQuantity {  
  description: "Select commodities based on quantity"  
  statement:  
    | SELECT org.acme.trading.Commodity  
    | WHERE (quantity > 60)  
}
```

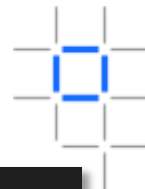
```
return query('selectCommoditiesWithHighQuantity', {})
```

# Smart Contract Development: Composer Playground



- Web tool for defining and testing Hyperledger Composer models and scripts
- Designed for the application developer
  - Define assets, participants and transactions
  - Implement transaction processor scripts
  - Test by populating registries and invoking transactions
- Deploy to instances of Hyperledger Fabric V1, or simulate completely within browser
- Install on your machine or run online at <http://composer-playground.mybluemix.net>

# General purpose development: Visual Studio Code



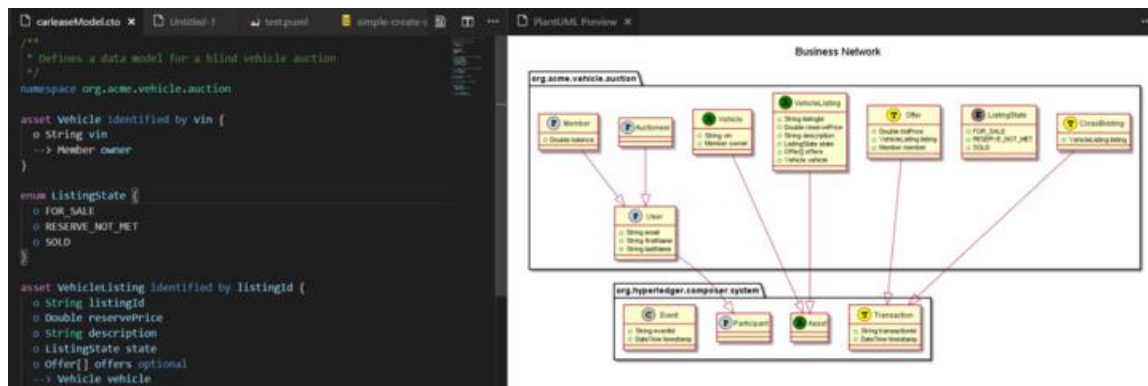
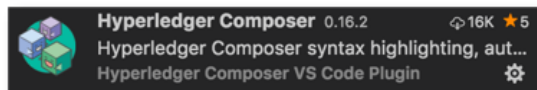
- Composer extension available for this popular tool
- Features to aid rapid Composer development
  - Edit all Composer file types with full syntax highlighting
  - Validation support for models, queries and ACLs
  - Inline error reporting
  - Snippets (press Ctrl+Space for code suggestions)
  - Generate UML diagrams from models
- Install directly from Code Marketplace

```
namespace org.acme.vehicle.lifecycle

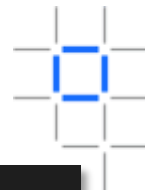
import composer.base.Person
import composer.business.Business

participant PrivateOwner identified by email extends Person {
  o String email
}
```

```
[Composer] IllegalModelException: Could not find super type Person
participant PrivateOwner identified by email extends Person {
  o String email
}
```



# Creating the Business and End-User Applications



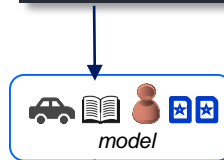
- JavaScript **business applications** *require()* the NPM “composer-client” module
  - This provides the API to access assets, participants and transactions
  - RESTful API (via Loopback) can also be generated... see later
- Command-line tool available to generate **end-user** command-line or Angular2 applications from model
  - Also helps with the generation of unit tests to help ensure quality code

```
const BusinessNetworkConnection = require('composer-client')
    .BusinessNetworkConnection;

this.bizNetworkConnection = new BusinessNetworkConnection();
this.titlesRegistry = this.bizNetworkConnection.getAssetRegistry
    ('net.biz.digitalPropertyNetwork.LandTitle')

let landTitle1 = factory.newResource
    ('net.biz.digitalPropertyNetwork', 'LandTitle', 'LID:1148');
landTitle1.owner = ownerRelation;
landTitle1.information = 'A nice house in the country';
this.titlesRegistry.add(landTitle1);
```

```
yo hyperledger-composer
```



| listingId   | reservePrice | description      | state    | offers | vehicle   | Actions   |
|-------------|--------------|------------------|----------|--------|-----------|---|
| LISTING-001 | 1000         | A car listing    | FOR_SALE |        | VIN:67890 | <button>Update Asset</button> <button>Delete Asset</button> |
| LISTING-002 | 4500         | A brand new Ford | FOR_SALE |        | VIN:12345 | <button>Update Asset</button> <button>Delete Asset</button> |

# Debugging

- Playground Historian allows you to view all transactions
  - See what occurred and when
- Diagnostics framework allows for application level trace
  - Uses the *Winston* Node.js logging framework
  - Application logging using DEBUG env var
  - Composer Logs sent to stdout and `./logs/trace_<processid>.trc`
- Fabric chaincode tracing also possible (see later)
- More information online:  
<https://hyperledger.github.io/composer/problems/diagnostics.html>

The screenshot shows the 'Test' tab of the Playground Historian. It displays a table of transactions with columns: ID, Time, Participant ID, and Transaction Type. A modal window titled 'Transaction Data' is open, showing the details of a transaction.

| ID                                 | Time     | Participant ID | Transaction Type        |
|------------------------------------|----------|----------------|-------------------------|
| af9faafd-d973-4647-9fad-0f58c0b... | 17:15:00 | emma           | Offer                   |
| 74e63603-7c7f-4bf2-b917-4c9707...  | 17:14:34 | <system>       | ActivateCurrentIdentity |

**Transaction Data**

```
1 {
2   "$class": "org.hyperledger.composer.system.HistorianRecord",
3   "transactionId": "af9faafd-d973-4647-9fad-0f58c0ba7d15",
4   "transactionType": "Offer",
5   "transactionInvoked":
6     "resource:org.hyperledger.composer.system.Transaction#af9faafd-
7       d973-4647-9fad-0f58c0ba7d15",
8   "participantInvoking":
9     "resource:org.hyperledger.composer.system.Participant#emma",
10  "identityUsed":
11    "resource:org.hyperledger.composer.system.Identity#8d0fdf5ef7c0062f
12      67853ecf9b36544b2e2c36f0e9b9536166dc0f056a62a032",
13  "eventsEmitted": [],
14  "transactionTimestamp": "2017-08-11T16:15:00.161Z"
15 }
```



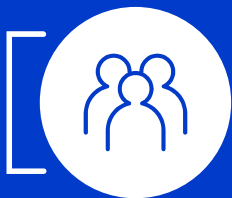
What is Hyperledger  
Composer?



Application Development

*Writing the application*

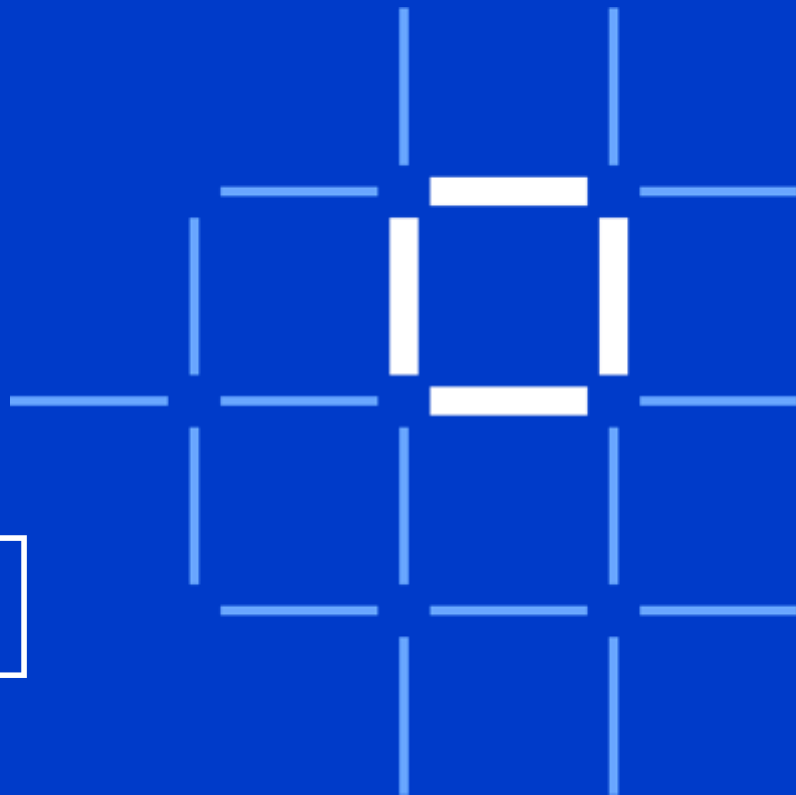
*Modeling the business network*



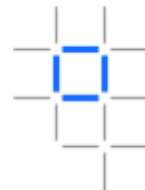
Effective Administration

*Deploying to a blockchain*

*Interacting with systems of record*



# Connection Profiles to Hyperledger Fabric



**Basic Configuration**

Connection Profile Name: hlfabric

Orderer(s):  
Orderer URL: grpc://orderer.example.com:7050

Channel: composerchannel

MSP ID: Org1MSP

Certificate Authority (CA):  
URL: http://ca.org1.example.com:7054  
Name: ca.org1.example.com

Peer(s):  
Peer Request URL: grpc://peer0.org1.example.com:7051  
Peer Event URL: grpc://peer0.org1.example.com:7053

Key Value Store Directory: /home/composer/.composer-credentials

**Advanced**

Use this profile [button] Export Connection Profile [button]

- Use **connection profiles** to describe Fabric connection parameters
  - One connection profile required per channel
- Not necessary for web-based simulation
- Enrollment in Hyperledger Fabric network required (see later)
  - Issue Fabric identity from Composer participants

- Connection profiles currently used by Composer only
  - Plans to implement common connection profiles that can be used by both Fabric and Composer

```
1 connection.json x
2 {
3   "type": "hlfv1",
4   "orderers": [
5     { "url": "grpc://localhost:7050" }
6   ],
7   "ca": { "url": "http://localhost:7054",
8     "name": "ca.org1.example.com"
9   },
10  "peers": [
11    {
12      "requestURL": "grpc://localhost:7051",
13      "eventURL": "grpc://localhost:7053"
14    }
15  ],
16  "keyValStore": "${HOME}/.composer-credentials",
17  "channel": "composerchannel",
18  "mspID": "Org1MSP",
19  "timeout": "300"
20 }
```

# Participant Identity



- The **Network Service Consumer** issues network participants with an **identity** in order to connect to Hyperledger Fabric
  - Issued as a Hyperledger Fabric userid/secret
  - Automatically swapped for a certificate on first use
  - Packaged in a Business Network Card and supplied when the client application connects
- Composer Participant to Fabric Identity mapping is stored on the blockchain in an *identity registry*
- Usually, only **Business Service Consumers** have a Fabric identity
  - **End-users** log in to the business application using a separately managed identity; blockchain transactions invoked by proxy
- Manage identity from Playground, Javascript, REST or command line
  - For example: Test connection, issue identity, bind an identity to a participant, revoke an identity, list identities

Issue New Identity

Issue a new ID to a participant in your business network

ID Name\*

Participant\*

☐ Allow this ID to issue new IDs (H)

Issuing an identity generates a one-time secret. You can choose to send this to somebody or use it yourself when it has been issued.

Cancel Create New

Identity Issued

E

|                    |                    |
|--------------------|--------------------|
| CONNECTION PROFILE | hlfv1              |
| USER ID            | emma_id            |
| BUSINESS NETWORK   | carauction-network |

Use it yourself

Just add the business network card to your wallet to start using the new identity yourself

Give the business network card a name  
e.g. emma\_id@carauction-network

Add to wallet

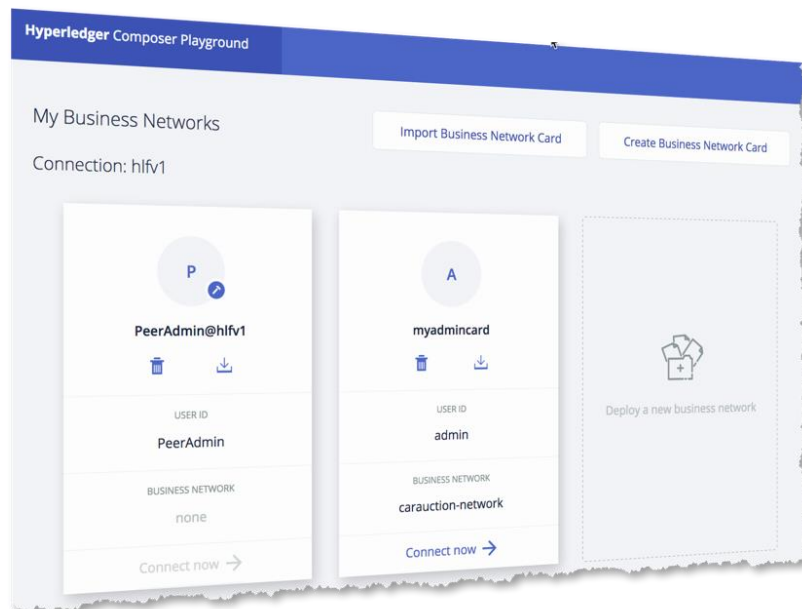
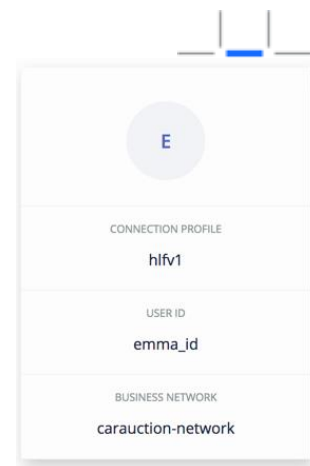
Send it to someone else

For security, new identities can only be enrolled once



# Business Network Cards

- Business Network Cards are a convenient packaging of *identity* and *connection profile*
  - Contains everything you need to connect to blockchain business network
  - Each card refers to a single participant and single business network
  - Analogous to an ATM card



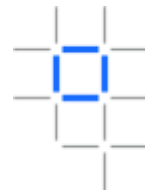
- Manage cards from both Playground and command-line
  - Create, delete, export, import, list
  - Create requires userid/secret or certificate/private key
- Use cards to connect to Fabric from Playground, command-line or from within your application

```
composer network deploy -a my.bna -c my.card
```

```
// Connect and log in to HLF
var businessNetwork = new BusinessNetworkConnection();
return businessNetwork.connect('cardName')
.then(function(businessNetworkDefinition){
    // Connected
});
```

# Systems of Record Integration

- Domain specific APIs very attractive to mobile and web developers. Resources and operations are business-meaningful
- Composer exploits Loopback framework to create REST APIs: <https://loopback.io/>
- Extensive test facilities for REST methods using loopback
- Secured using JS Passport, giving >400 options for authentication
- Composer provides back-end integration with any loopback compatible product
  - e.g. IBM Integration Bus, API Connect, StrongLoop
  - Outbound and Inbound (where supported by middleware)



## angular-app

Auctioneer : A participant named Auctioneer

Show/Hide | List Operations | t

CloseBidding : A transaction named CloseBidding

Show/Hide | List Operations | t

Member : A participant named Member

Show/Hide | List Operations | t

Offer : A transaction named Offer

Show/Hide | List Operations | t

Vehicle : An asset named Vehicle

Show/Hide | List Operations | t

|      |               |   |
|------|---------------|---|
| GET  | /Vehicle      | Find all instances of the model matched by filter fro |
| POST | /Vehicle      | Create a new instance of the model and persist it in  |
| GET  | /Vehicle/{id} | Find a model instance by {{id}} fro                   |

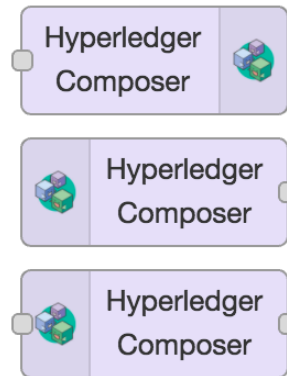
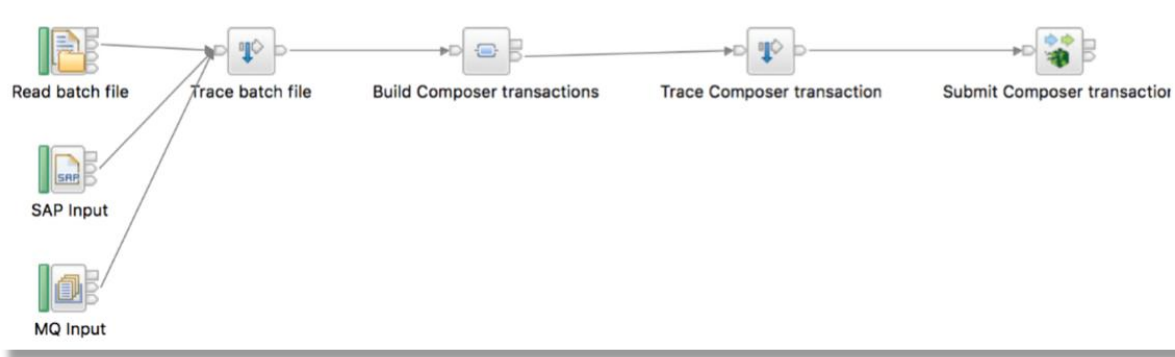
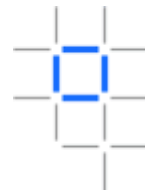
### Request URL

http://0.0.0.0:3000/api/Vehicle

### Response Body

```
{
  {
    "$class": "org.acme.vehicle.auction.Vehicle",
    "vin": "VEH:1234",
    "owner": "odowda@uk.ibm.com"
  }
}
```

# Exploiting Loopback: Examples



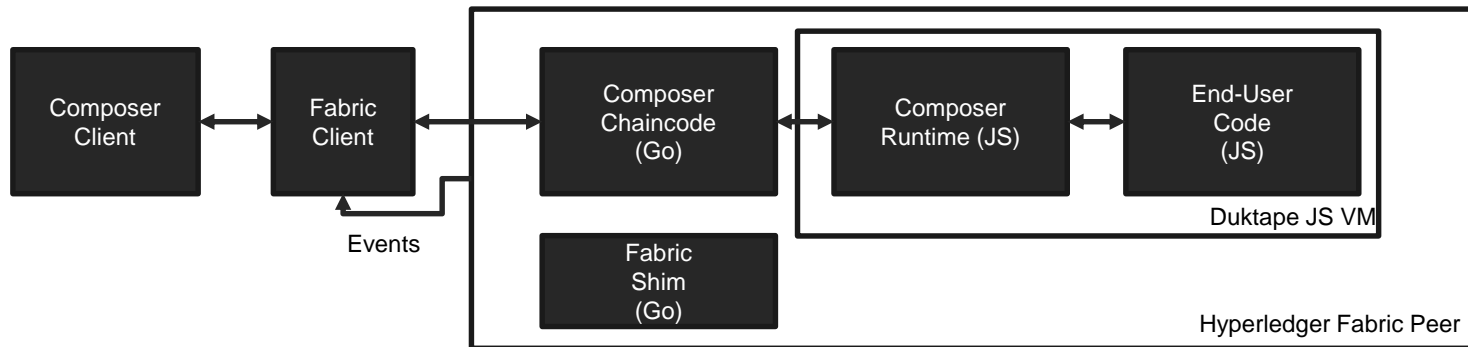
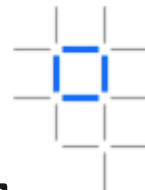
## – IBM Integration Bus

- IIB V10 contains Loopback connector
- Example above takes input from file, SAP or MQ
- Data mapping from CSV, BAPI/IDOC or binary form to JSON model definition

## – Node.RED

- Pre-built nodes available for Composer
- Connect to hardware devices, APIs and online services
- Install direct from Node.RED UI
  - Manage Palette -> Install -> node-red-contrib-composer

# How Composer Maps to Fabric Chaincode



- Each Business Network is deployed to its own chaincode container
  - Container contains a static piece of Go chaincode that starts a Javascript virtual machine running transaction processors
- Browse these containers to view diagnostic information (docker logs)
- Embedded chaincode is not a Composer external interface

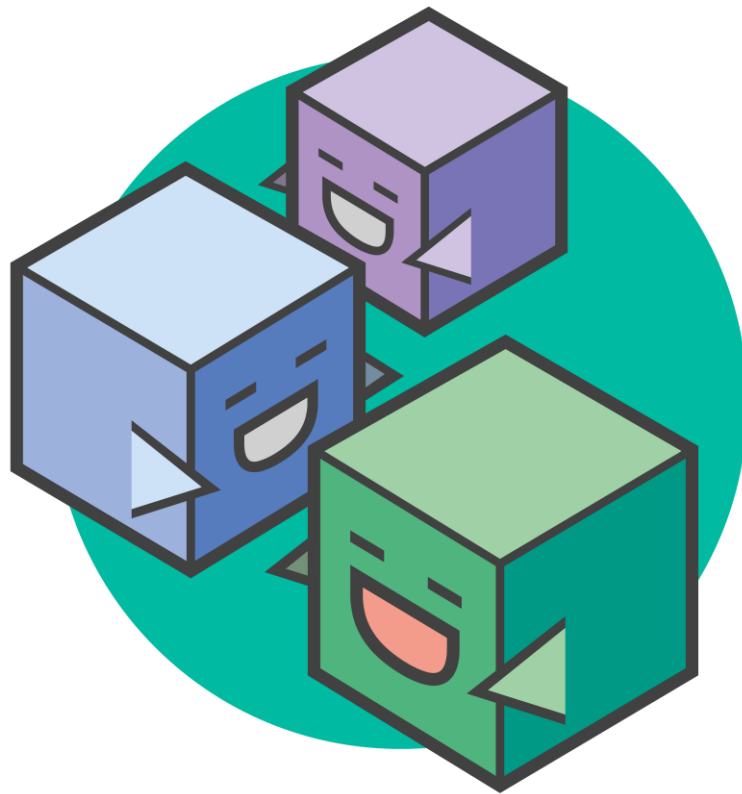
# Get started with Hyperledger Composer

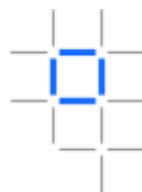


- Define, Test and Deploy Business Networks
- Create domain APIs and sample applications
- Integrate existing systems and data

<https://hyperledger.github.io/composer/>

<http://composer-playground.mybluemix.net/>





Welcome to Hyperledger Composer Playground!

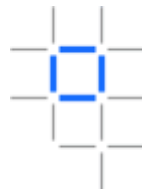


In this web sandbox, you can deploy, edit and test business network definitions. Have a play and learn what Hyperledger Composer Playground is all about.

Let's Blockchain!



Not sure where to start? View our Playground tutorial.



## My Business Networks

Connection: Web Browser



Hello, Composer!

Get started with the basic-sample-network, or view our [Playground tutorial](#)

BUSINESS NETWORK

basic-sample-network

Get Started →



Deploy a new business network

## Choose a Business Network Definition to start with:

Choose a sample to play with, start a new project, or import your previous work



basic-sample-network



empty-business-network



Drop here to upload or [browse](#)

## Samples on npm



animaltracking-network



bond-network



carauction-network



digitalproperty-network



marbles-network



perishable-network



pii-network



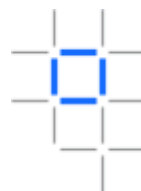
trade-network



vehicle-lifecycle-network



vehicle-manufacture-network







© Copyright IBM Corporation 2018. All rights reserved. The information contained in these materials is provided for informational purposes only, and is provided AS IS without warranty of any kind, express or implied. Any statement of direction represents IBM's current intent, is subject to change or withdrawal, and represents only goals and objectives. IBM, the IBM logo, and other IBM products and services are trademarks of the International Business Machines Corporation, in the United States, other countries or both. Other company, product, or service names may be trademarks or service marks of others.