Hyperledger Composer

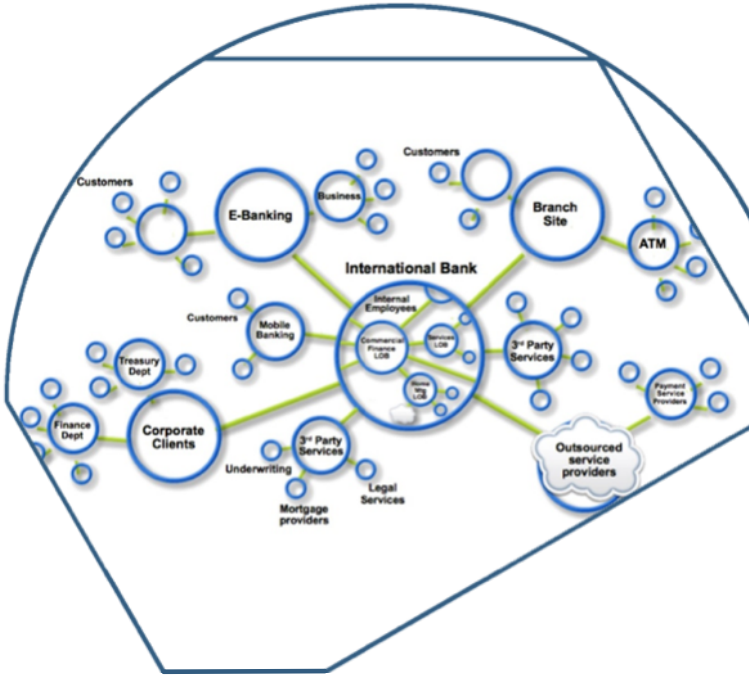
Sanjay Saxena

May 2017

Hyperledger Fabric

- Hyperledger Fabric Recap:
  - Distributed Systems
  - Cryptography, PKI, Secure Hash Functions
  - Pluggable Consensus Algorithms
  - Smart Contracts / Chaincode
  - Key-Value Store
  - Transaction Ledger
  - **Oh My!**

- Hyperledger Fabric provides a shared, replicated ledger
  - Consensus, immutability, finality, provenance



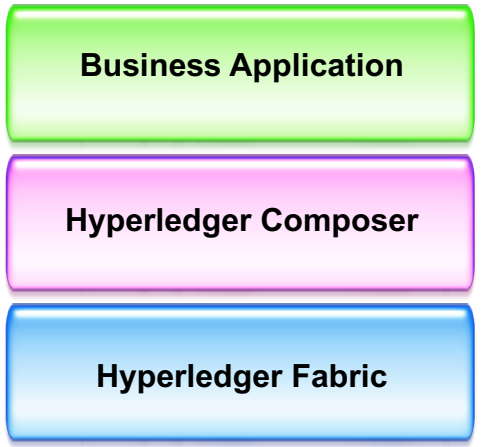
Challenges with Blockchain Development

- Steep Learning Curve
- App developers are exposed to the blockchain complexity
- Building PoCs or prototyping is extremely painful
- Lack of integration with best-of-breed tools and processes

What is Hyperledger Composer?

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

- **Hyperledger Composer**
  - Exposes higher level abstractions over the blockchain
  - Emphasis on business-centric vocabulary/terminology
  - Suitable for Rapid Application Development – Run/Test/Debug
  - Business logic is implemented using JavaScript
- **Features**
  - **Business Network Archive** serves as the deployable unit to Fabric runtime. **Business Network** is made up of
    - **Business Model**, **Business Logic**, and **ACLs**
  - **Applications** use REST APIs to interact with **Business Network**
  - **Integrate** existing systems using Loopback/Swagger/REST
  - All the artifacts are in text format
- **Open Tools, APIs**
  - Leverages Hyperledger Fabric blockchain technology
  - Fully open and part of Linux Foundation Hyperledger
  - Integrates well with Github, npm, Travis CI, Jenkins
  - Docker images and npm packages

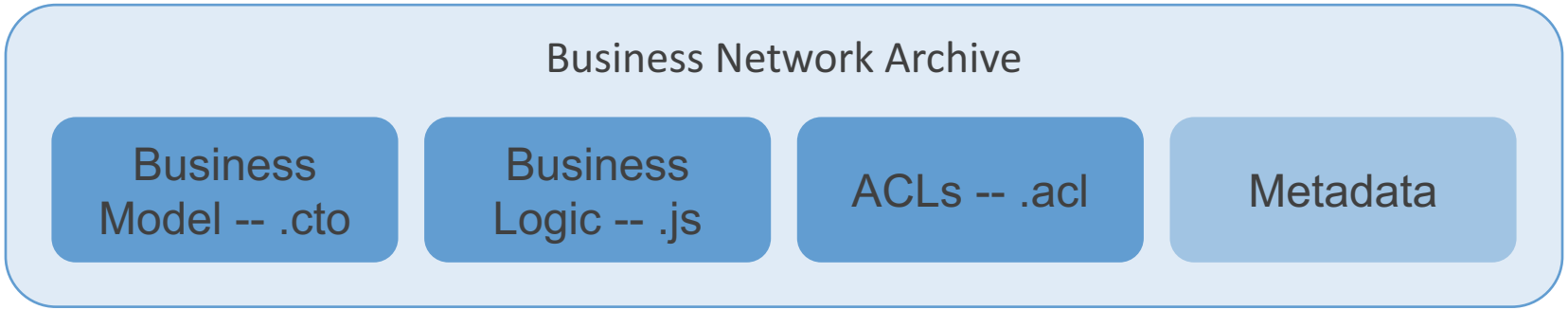


Conceptual Components and Structure

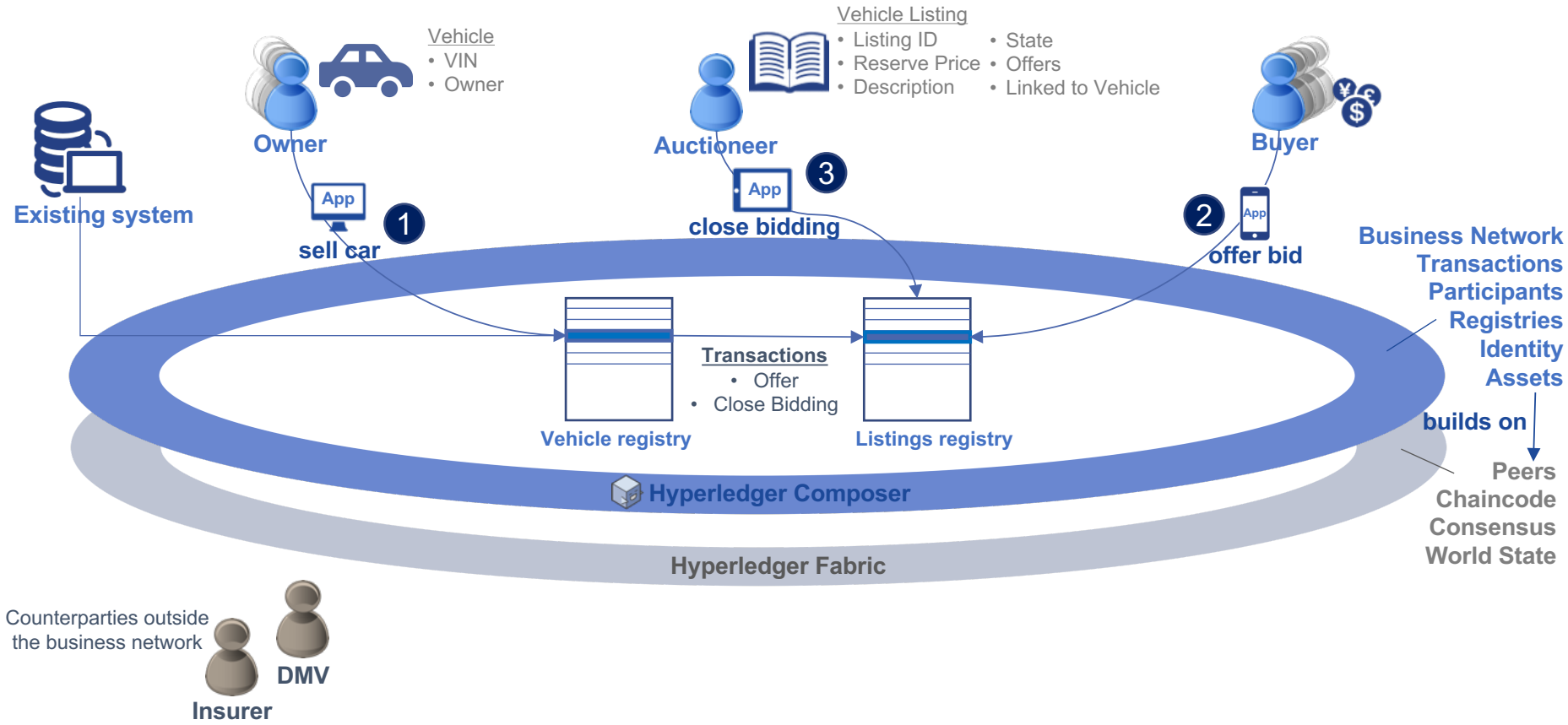
Business Network is defined by **Business Model, Business Logic, ACLs,** and **Metadata** and packaged in a **Business Network Archive**

Developer models the business network, implements the business logic in JavaScript files, and packages them into a business network archive

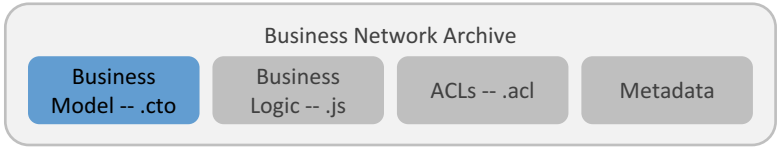
Administrator provisions the target environment and deploys the archive



An Example Business Network – Car Auction Market



The Business Model



- A domain specific language that defines:
  - **Assets**
  - **Participants**
  - **Transactions**

- Matches how we talk about business networks in the real world

carauction-network 0.0.8

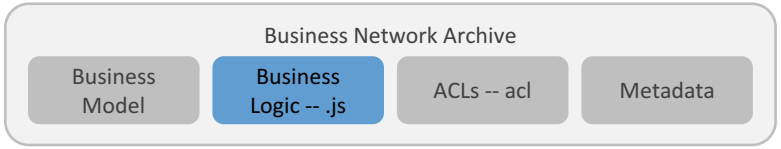
```
/**
 * Defines a data model for a blind vehicle auction
 */
namespace org.acme.vehicle.auction

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

enum ListingState {
  o FOR_SALE
  o RESERVE_NOT_MET
  o SOLD
}

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

Business Logic

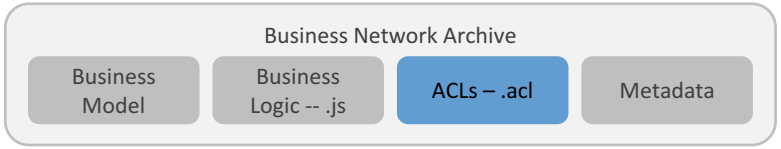


- Provide transaction implementation logic
- Specified in Javascript
- Designed for any reasonable Javascript developer to pick up easily

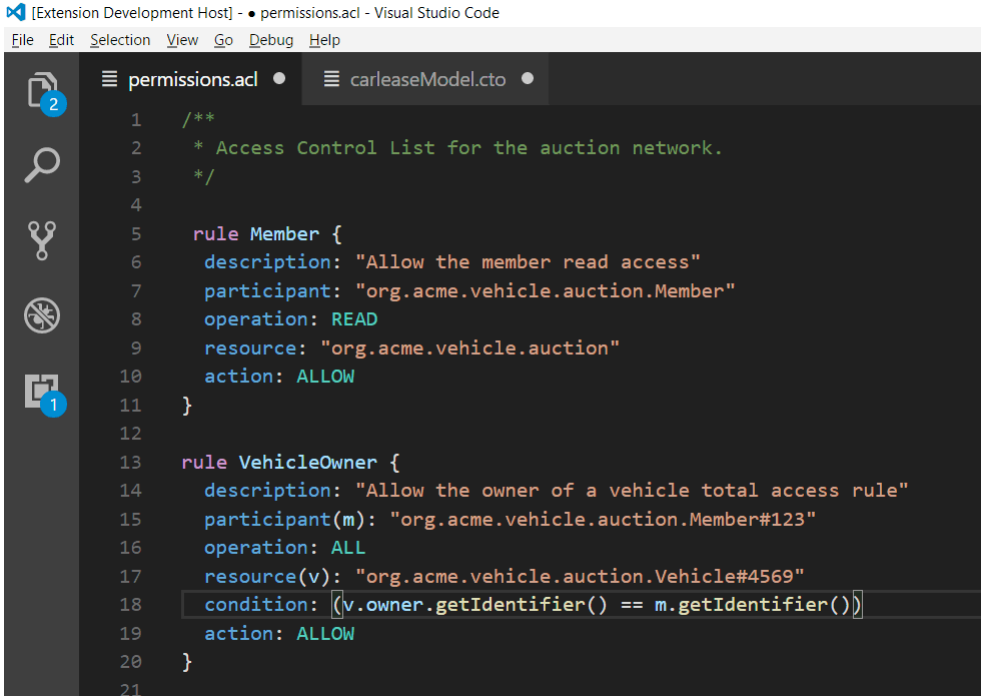
carauction-network 0.0.8

```
84
85  /**
86   * Make an Offer for a VehicleListing
87   * @param {org.acme.vehicle.auction.Offer} offer - the offer
88   * @transaction
89   */
90  function makeOffer(offer) {
91      var listing = offer.listing;
92      if (listing.state !== 'FOR_SALE') {
93          throw new Error('Listing is not FOR SALE');
94      }
95      if (listing.offers == null) {
96          listing.offers = [];
97      }
98      listing.offers.push(offer);
99      return getAssetRegistry('org.acme.vehicle.auction.VehicleListing')
100         .then(function(vehicleListingRegistry) {
101             // save the vehicle listing
102             return vehicleListingRegistry.update(listing);
103         });
104 }
105
106
```

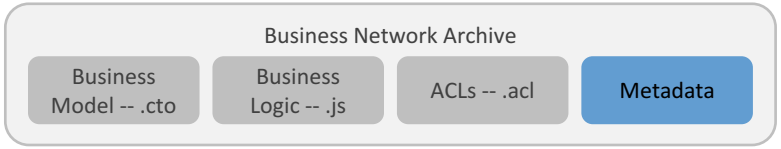
ACL



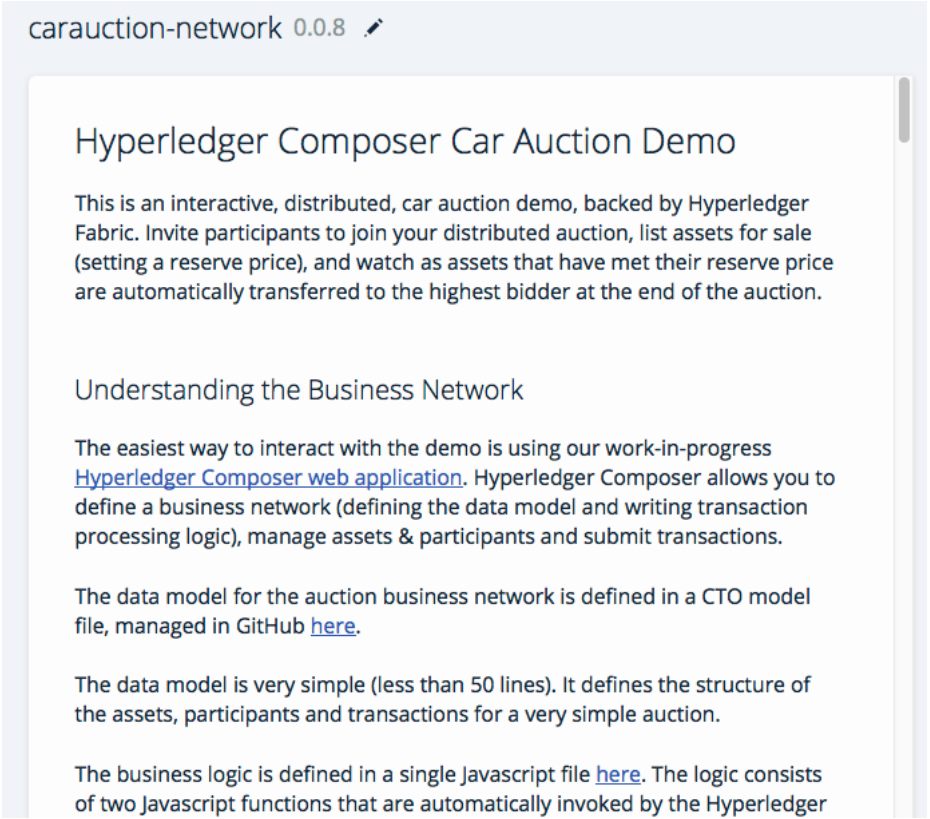
- Separates out access control from business logic making it simpler
  - Can build access control into the business logic if needed
- ACL engine evaluates rules for all access to assets
  - Top down checking



Metadata



- Name & version of the Business Network
- Markdown(.md) documentation for the solution
- Works with tools like Github to turn into a HTML page
- Easily read in raw .md or HTML format


carauction-network 0.0.8

Hyperledger Composer Car Auction Demo

This is an interactive, distributed, car auction demo, backed by Hyperledger Fabric. Invite participants to join your distributed auction, list assets for sale (setting a reserve price), and watch as assets that have met their reserve price are automatically transferred to the highest bidder at the end of the auction.

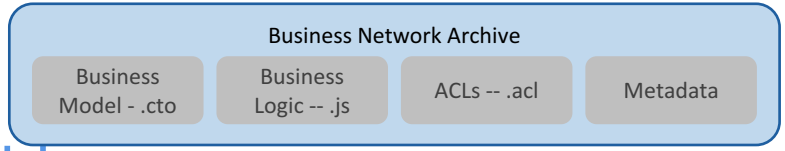
Understanding the Business Network

The easiest way to interact with the demo is using our work-in-progress Hyperledger Composer web application. Hyperledger Composer allows you to define a business network (defining the data model and writing transaction processing logic), manage assets & participants and submit transactions.

The data model for the auction business network is defined in a CTO model file, managed in GitHub here.

The data model is very simple (less than 50 lines). It defines the structure of the assets, participants and transactions for a very simple auction.

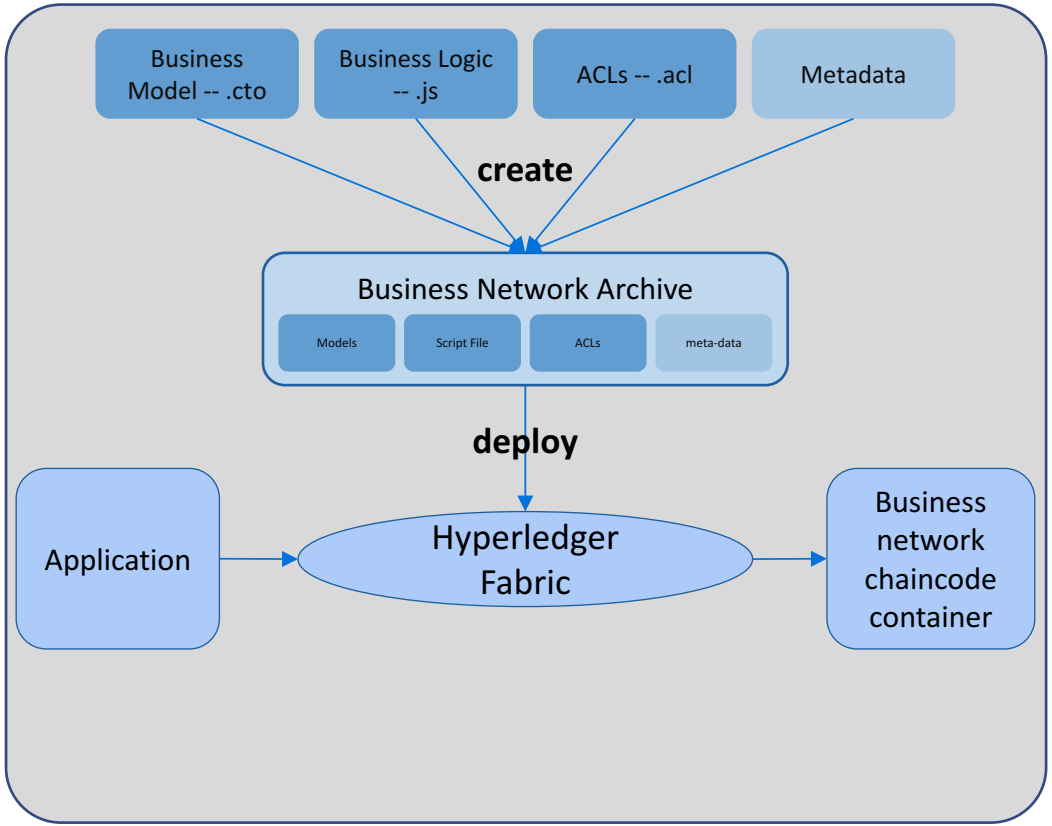
The business logic is defined in a single Javascript file here. The logic consists of two Javascript functions that are automatically invoked by the Hyperledger



What

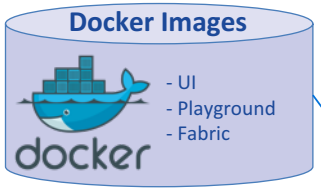
Business Network Archive

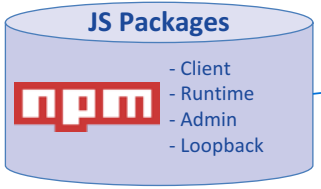
- Business Network Archive is used to package up the artifacts for deployment to a runtime
- Can be deployed using CLI tools
- Can be packed/unpacked by a developer to see its contents, check it, work on it or send it.
- Once deployed, the archive is executed within a chaincode container

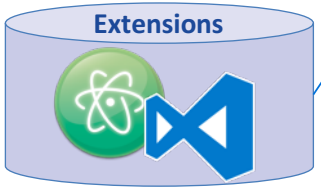


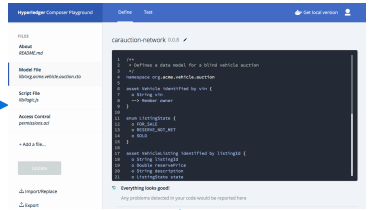
Demo: Using Composer Playground Import and and Test a Car Auction Business Network

Getting Started With Hyperledger Composer





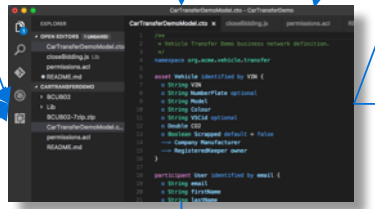




Composer Playground

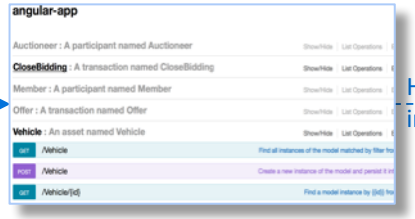
Once tested, download to use locally

Feedback loop

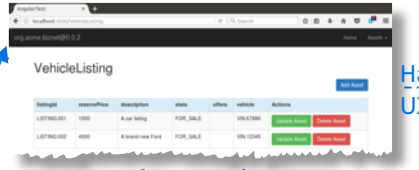


Local Composer Dev

Powered by



Swagger REST API

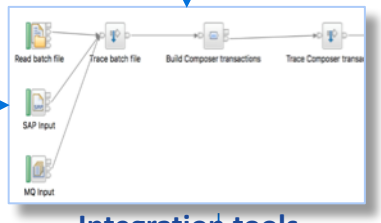


Sample Angular 2 UI

Handed to UX/UI

Final Application

Handed to integration



Integration tools

Local or External fabric

Existing Systems

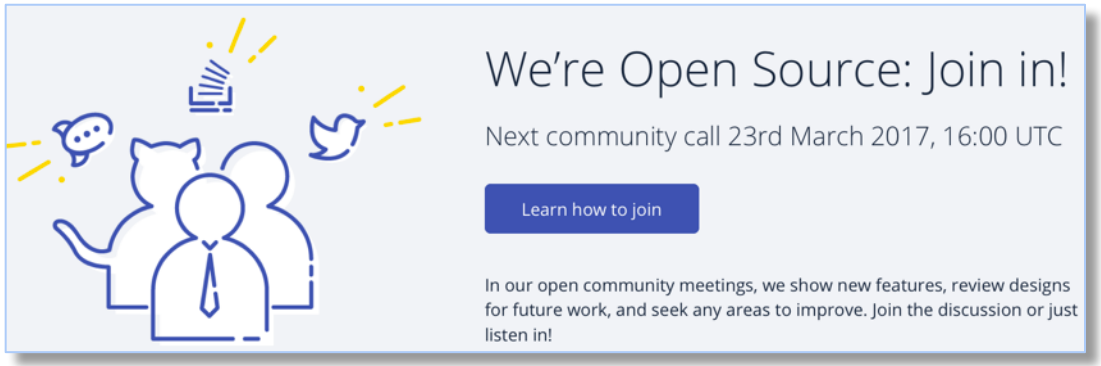
Hyperledger Composer Outlook

- Still early in product lifecycle

- Complete the programming model
  - **Events** for notification and loose coupling of transaction processors
  - **Links** for business network linkage
  - Extensive **query** to support reporting and analytics
  - **Encryption** of data and transactions

- Operational Enhancements
  - Exploit Hyperledger Fabric V1: consensus, channels, CouchDB

- Build Activity Community
  - Open calls every 2 weeks
  - Rocket Chat
  - Hyperledger incubation



Summary

Hyperledger Composer

– Define, Test and Deploy Business Networks

– Create domain APIs and sample applications

– Integrate existing systems and data

Try it now!

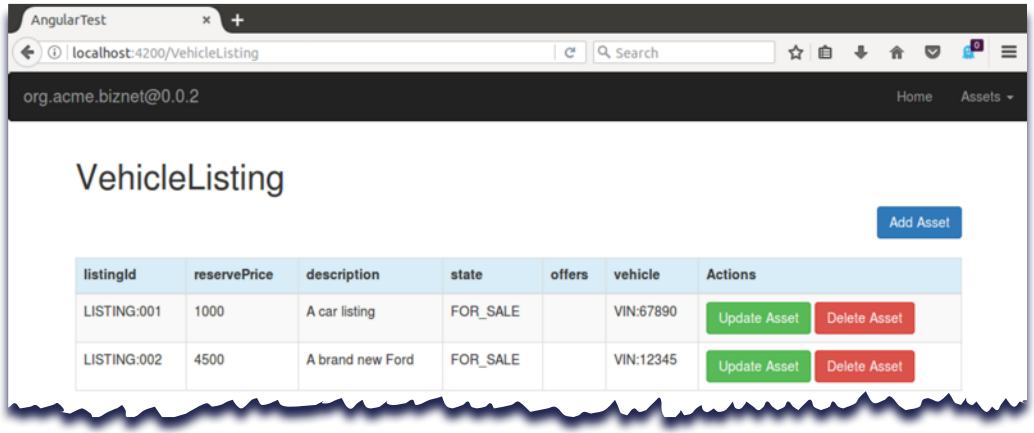
https://hyperledger.github.io/composer/

Thank you!

Generating APIs and sample applications

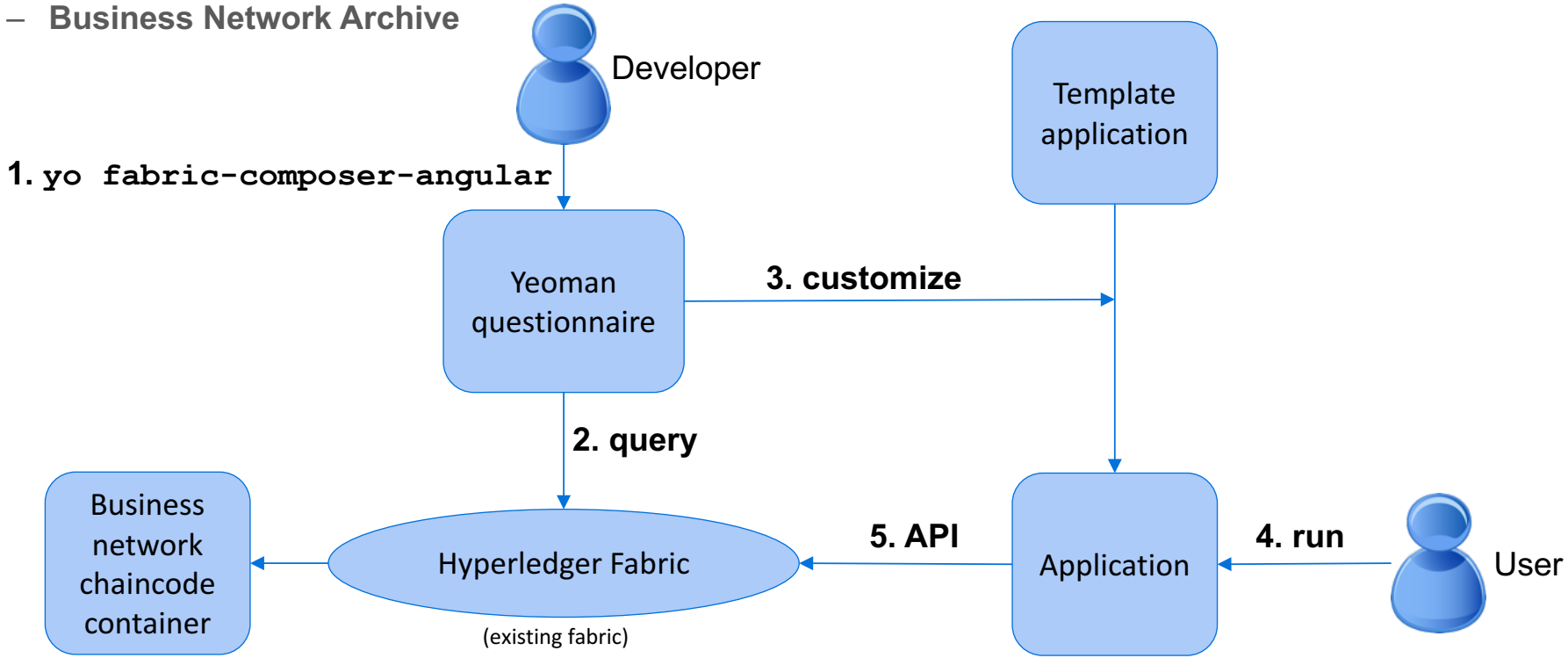
- Programmable business networks
  - Direct consequence of a deployed **model**
- Query network to generate domain APIs
- Generate sample application for deployed business network
- Yeoman questionnaire
  yo fabric-composer[:angular]
- Also generate test cases using **mocha** and **chai** node.js test packages
  - **composer generator tests**
- Programmable business network provides many more opportunities for interaction

```
 ? Your NPM library name: concerto-sample-app
? Short description: Test Concerto project
? Author name: Sophie Black
? Author email: sophie@ampretia.com
? NPM Module name of the Business Network to connect to: digitalproperty-network
? Is the name in NPM registry the same as the Business Network Identifier?: Yes
? What is the Connection Profile to use? defaultProfile
? Enrollment id: WebAppAdmin
? Enrollment Secret: DJY27pEnl16d
configuring: concerto-sample-app
Getting the npm module describing the undefined
   create config/default.json
   create Dockerfile
   create gulpfile.js
```



Generating APIs and sample applications

– **Business Network Archive**



Command Line Interfaces

- Suite of commands to interact with an operational business network
- Target use is scripting and interactive operations
- Packaged as **composer-cli** in npm
- Generate full list with **composer –help.** Individual (sub) commands support **-help**
- CLI uses public APIs
  - Users can create their own CLIs

```
$> composer –help

Commands:
  archive <subcommand>      Composer archive command
  generator <subcommand>    Composer generator command
  identity <subcommand>     Composer identity command
  network <subcommand>      Composer network command
  participant <subcommand>  Composer participant command
  transaction <subcommand>  Composer transaction command

Options:
  --help         Show help  [boolean]
  -v, --version  Show version number  [boolean]


Examples:
  composer identity issue

For more information: http://fabric-composer.org/reference
```

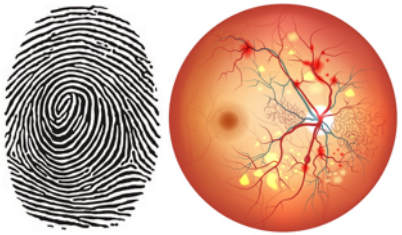
```
$> composer transaction submit –help

composer transaction submit [options]

Options:
  --help                        Show help  [boolean]
  -v, --version                 Show version number  [boolean]
  --connectionProfileName, -p   The connection profile name  [string]
  --businessNetworkName, -n     The business network name  [string] [required]
  --enrollId, -i                The enrollment ID of the user  [string] [required]
  --enrollSecret, -s            The enrollment secret of the user  [string]
  --data, -d                    Transactions JSON object [string] [required]
```

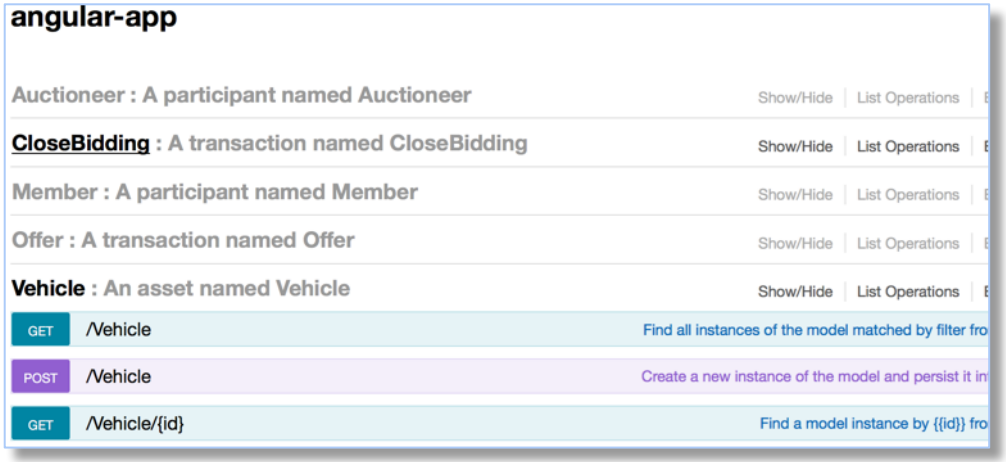
Identity

- Participants use a form of identity to prove that they are who they say they are, and that they are allowed to interact with a business network.
- Participants may have multiple forms of identity for different purposes:

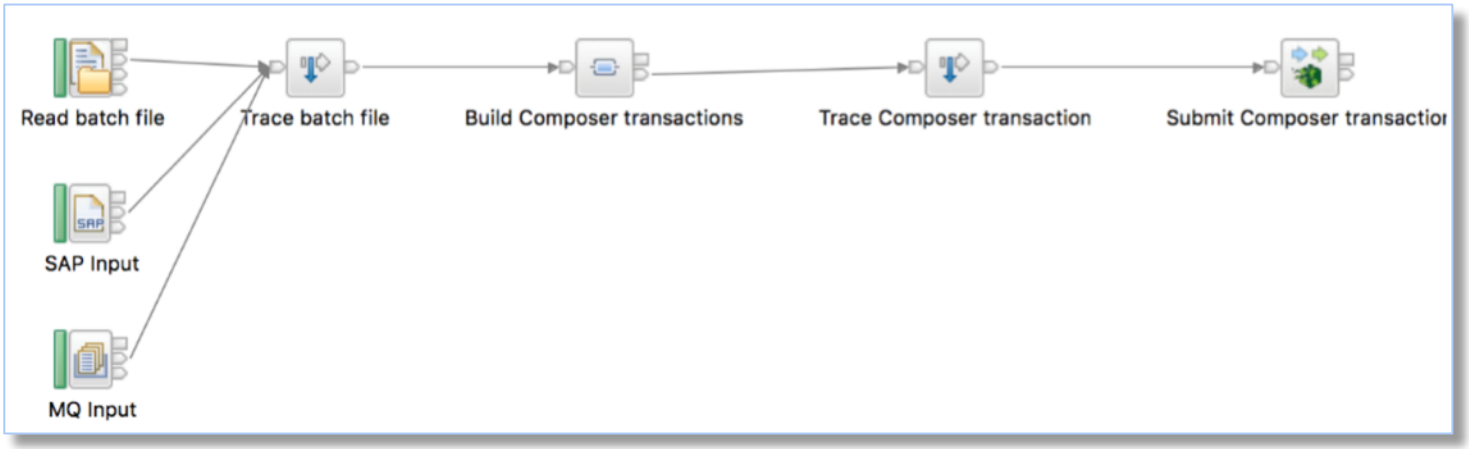


Loopback and REST Support

- Exploit Loopback framework to create REST APIs. https://loopback.io/
- Domain specific APIs very attractive to mobile and web developers. Resources and operations are business-meaningful
- Extensive test facilities for REST methods using loopback
- Provides back-end integration with any loopback compatible product
  - e.g. IBM Integration Bus, API Connect, StrongLoop
  - Outbound notification (under development)



Exploiting the Loopback Connector: Example



- IBM Integration Bus Example
  - Takes input from file, SAP or MQ
- Data mapping from CSV, BAPI/IDOC or binary form to JSON model definition of vehicle
- Currently inbound only
  - Events support will eventually allow outbound integration

Key Concept: Assets

– Represents the resources being exchanged in the business network

1. Define using **asset** keyword in model file

2. Assets have structure – domain relevant **class** name, e.g. vehicle, house, bond

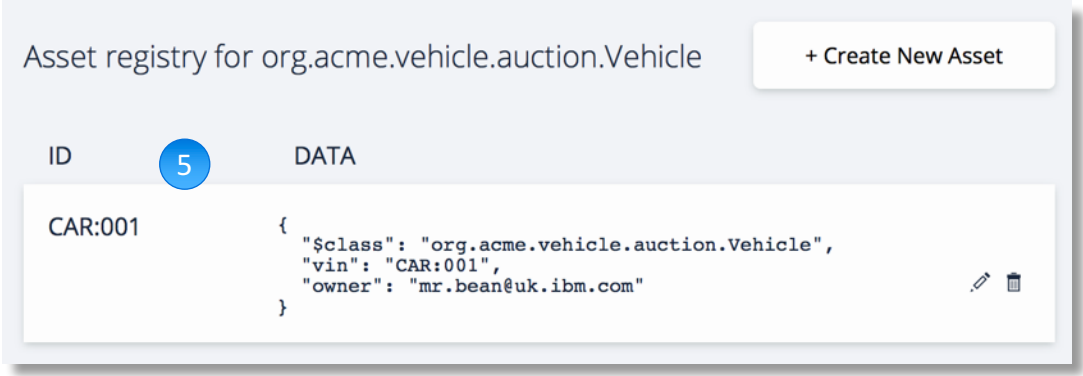
3. Set of **properties,** denoted by ‘o’ (letter).

4. **Relationships** to other resources, denoted by ‘→’. **Optional** elements are allowed. Field validators can be provided

5. Stored in an asset registry. Registries are first class abstraction.

```
1
 6  asset Vehicle identified by vin {
 7    o String vin
 8    --> Member owner   2
 9  }
10
```

```
18  asset VehicleListing identified by listingId {
19    o String listingId
20  3 o Double reservePrice
21    o String description
22    o ListingState state
23    o Offer[] offers optional
24    --> Vehicle vehicle   4
25  }
```



Key Concept: Participants

```
27  abstract participant User identified by email {
28    o String email
29    o String firstName
30    o String lastName
31  }
32
33  participant Member extends User {
34    o Double balance
35  }
```

Participant registry for org.acme.vehicle.auction.Member

ID	DATA
mr.bean@uk.ibm.com	{ "$class": "org.acme.vehicle.auction.Member", "balance": 2500, "email": "mr.bean@uk.ibm.com", "firstName": "Henry", "lastName": "Bean" }
mrs.bean@uk.ibm.com	{ "$class": "org.acme.vehicle.auction.Member", "balance": 5000, "email": "mrs.bean@uk.ibm.com", "firstName": "Henrietta", "lastName": "Bean" }

– Represent the counterparties in the business network

1. Define using **participant** keyword in model file
2. Participants have a **class** name, relevant to the domain, e.g. buyer, seller
3. Set of **properties,** denoted by ‘o’. **Relationships** to other resources, denoted by ‘→’. **Optional** elements are allowed. Field validators can be provided
4. Like assets, can be sub-classed for refinement
5. Stored in a participant registry

Key Concept: Transactions

- Represents the steps that govern resource lifecycle, typically assets

1. Define using **transaction** keyword in model file

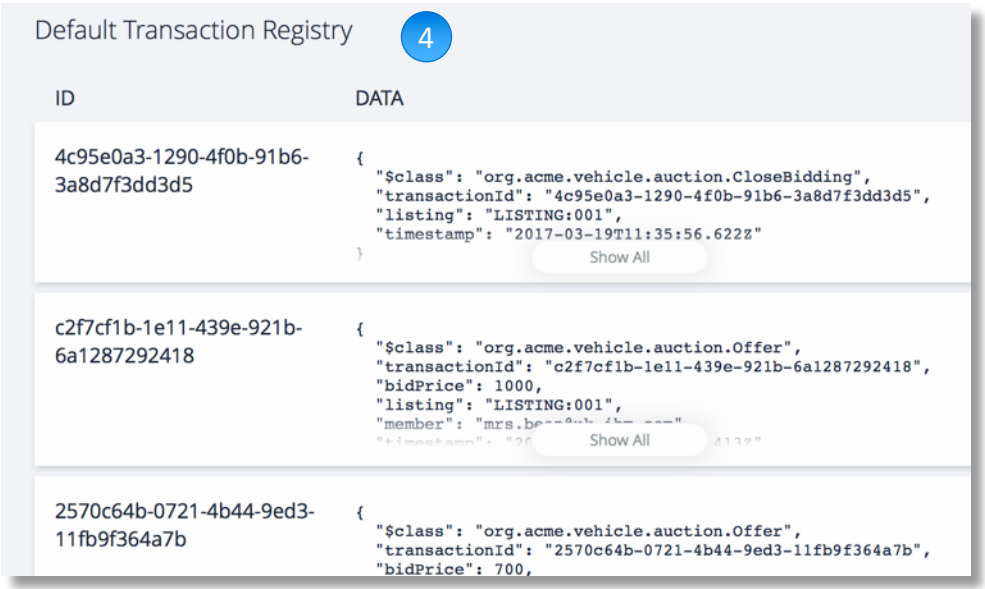
2. Assets have a **class** name, relevant to the domain, e.g. sellVehicle, buyHouse

3. Set of **properties,** denoted by ‘o’. **Relationships** to other resources, denoted by ‘→’. Field validators can be provided

4. Stored in a transaction registry

- Implementation provided separately

```
40  transaction Offer identified by transactionId {
41    o String transactionId
42    o Double bidPrice
43    --> VehicleListing listing
44    --> Member member
45  }
```



Key Concept: Transaction Processors

- Provide transaction implementation logic
- Provided in separate **<tp>.js** files
- **@param** & **@transaction** annotators
- Perform state changes on domain specific resources using model defined syntax

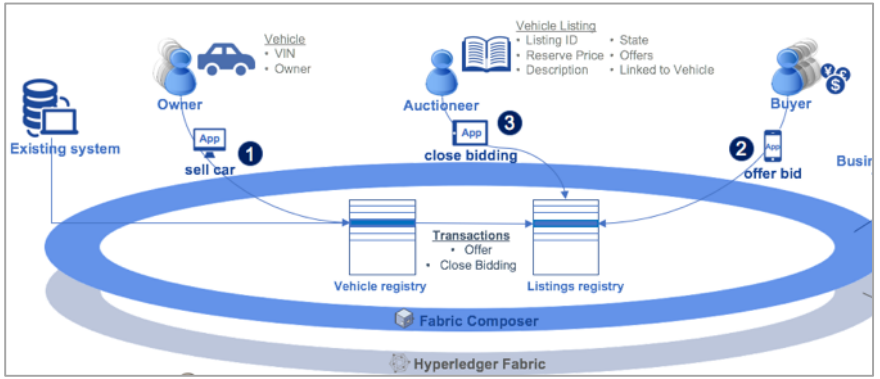
```
84
85  /**
86   * Make an Offer for a VehicleListing
87   * @param {org.acme.vehicle.auction.Offer} offer - the offer
88   * @transaction
89   */
90  function makeOffer(offer) {
91      var listing = offer.listing;
92      if (listing.state !== 'FOR_SALE') {
93          throw new Error('Listing is not FOR SALE');
94      }
95      if (listing.offers == null) {
96          listing.offers = [];
97      }
98      listing.offers.push(offer);
99      return getAssetRegistry('org.acme.vehicle.auction.VehicleListing')
100         .then(function(vehicleListingRegistry) {
101             // save the vehicle listing
102             return vehicleListingRegistry.update(listing);
103         });
104 }
105
```

Key Concept: Access Control Lists

- Separate ACL from application logic
- Defined in a **permissions.acl** file in business network definition
- Flexible model allowing both type and instance access e.g. ‘create cars’ and ‘scrap **my** car’
- Standard ACL model
  - Includes optional condition for more sophisticated ACL checking
- Executed in order until first rule hit
  - DENY has precedence over ALLOW

```
12  rule Member {
13      description: "Allow the member read access"
14      participant: "org.acme.vehicle.auction.Member"
15      operation: READ
16      resource: "org.acme.vehicle.auction"
17      action: ALLOW
18  }
19
20  rule VehicleOwner {
21      description: "Allow the owner of a vehicle total access"
22      participant(m): "org.acme.vehicle.auction.Member"
23      operation: ALL
24      resource(v): "org.acme.vehicle.auction.Vehicle"
25      condition: (v.owner.getIdentifier() == m.getIdentifier())
26      action: ALLOW
27  }
```

Business Network



- A version number <**version.release.mod**>
- Description: **README.md**
  - Freeform description in markdown format
- Model files: **<model>.cto**
  - Defines interfaces for **participants**, **assets, transactions** in a **namespace**
  - Expressive syntax includes references, arrays, enumerations, optionality, defaults…
- Transaction processors: **lib/<functions>.js**
  - Implement business logic for transaction definitions
  - Uses standard Javascript for ease of development and portability
- Access Control List: **permissions.acl**
  - Rules for resource access to business network by participants
- All files formats are plain text, developer and tool friendly