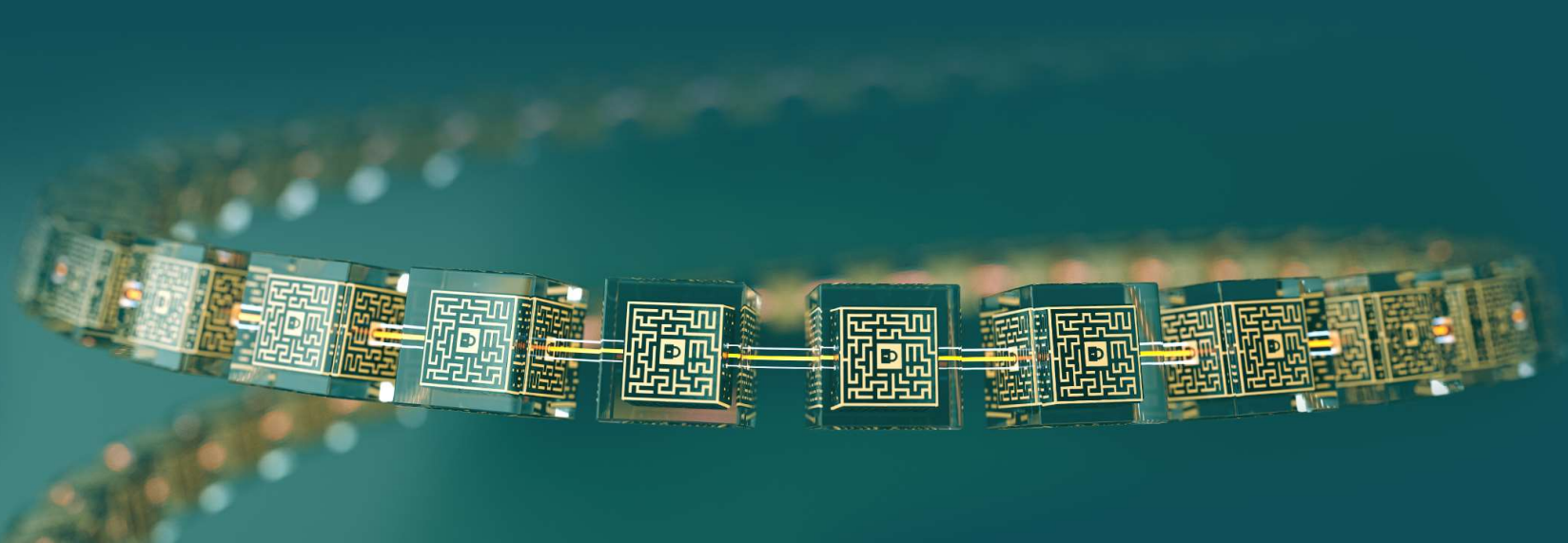


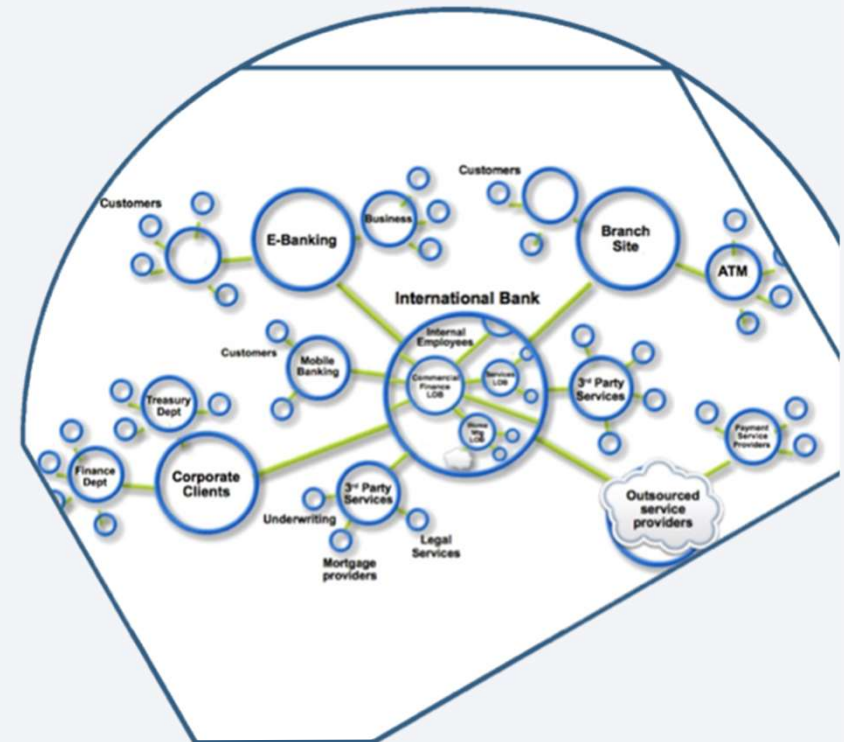
Blockchain Composed



Blockchain Recap



- Blockchain builds on basic business concepts
 - **Business Networks** connect businesses
 - **Participants with Identity**
 - **Assets** flow over business networks
 - **Transactions** describe asset exchange
 - **Contracts** underpin transactions
 - The **ledger** is a log of transactions
- Blockchain is a shared, replicated ledger
 - Consensus, immutability, finality, provenance



What is Hyperledger Composer?



<https://hyperledger.github.io/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

Business Application

Hyperledger Composer

Hyperledger Fabric

Benefits of Hyperledger Composer



Increases understanding

Bridges simply from business concepts to blockchain



Saves time

Develop blockchain applications more quickly and cheaply



Reduces risk

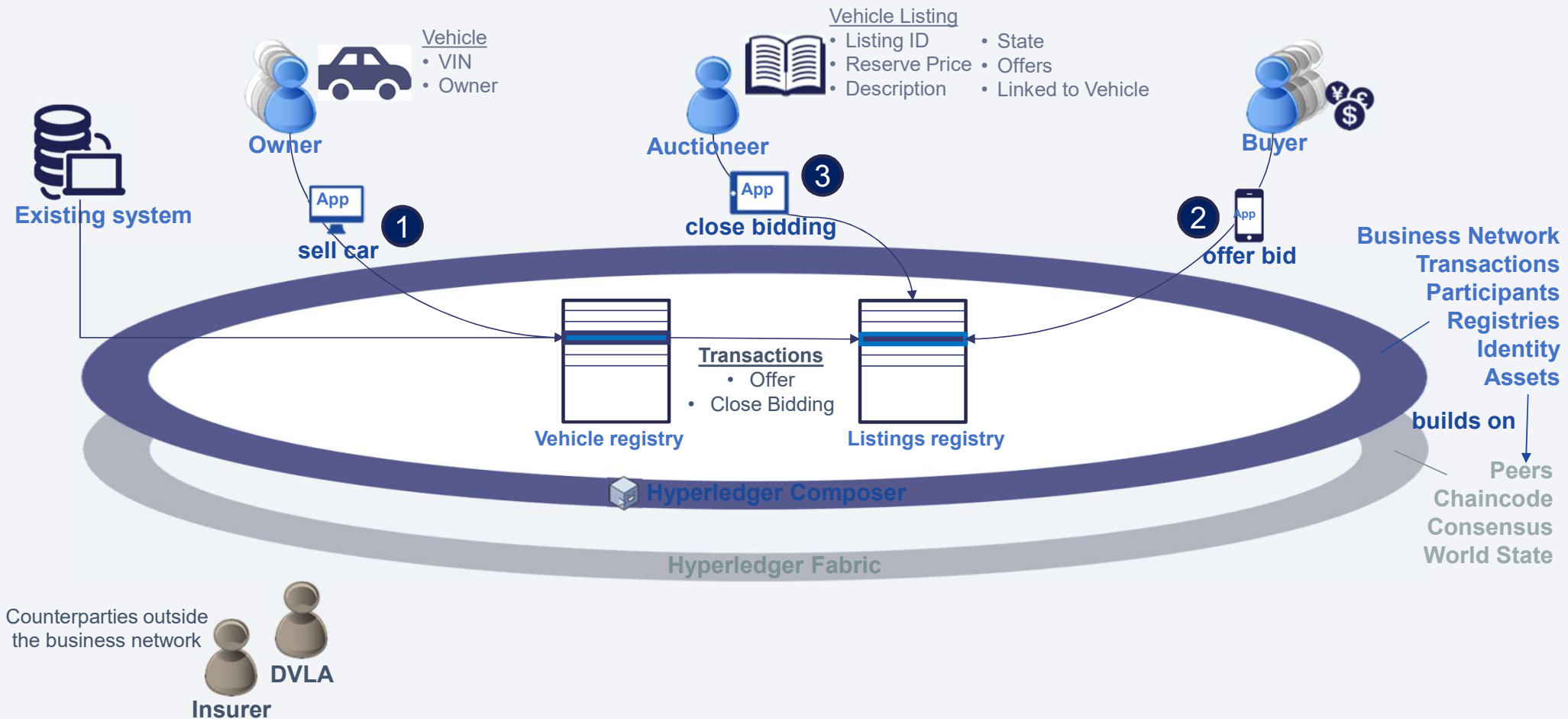
Well tested, efficient design conforms to best practice



Increases flexibility

Higher level abstraction makes it easier to iterate

An Example Business Network – Car Auction Market



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** provisions the target environment and manages deploy



The Model

Business Network Archive

Models

Script File

ACLs

Metadata

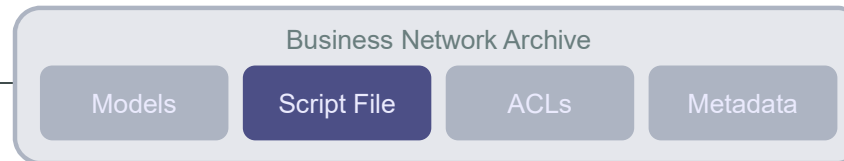


- A domain specific language that defines:
 - **Assets**
 - **Participants**
 - **Transactions**
- Matches how we talk about business networks in the real world

carauktion-network 0.0.8

```
1  /**
2   * Defines a data model for a blind vehicle auction
3   */
4   namespace org.acme.vehicle.auction
5
6   asset Vehicle identified by vin {
7     o String vin
8     --> Member owner
9   }
10
11  enum ListingState {
12    o FOR_SALE
13    o RESERVE_NOT_MET
14    o SOLD
15  }
16
17  asset VehicleListing identified by listingId {
18    o String listingId
19    o Double reservePrice
20    o String description
21    o ListingState state
22    o Offer[] offers optional
23    --> Vehicle vehicle
24  }
25
```

The Script File

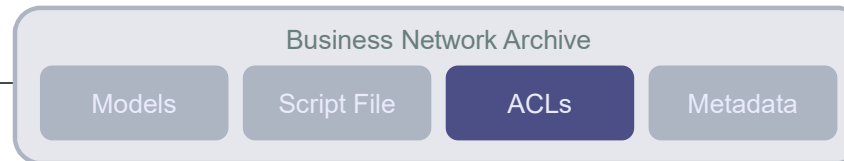


- Provide transaction implementation logic
- Specified in Javascript
- Designed for any reasonable Javascript developer to pick up easily
- Can publish external events from within a transaction so that client applications can respond

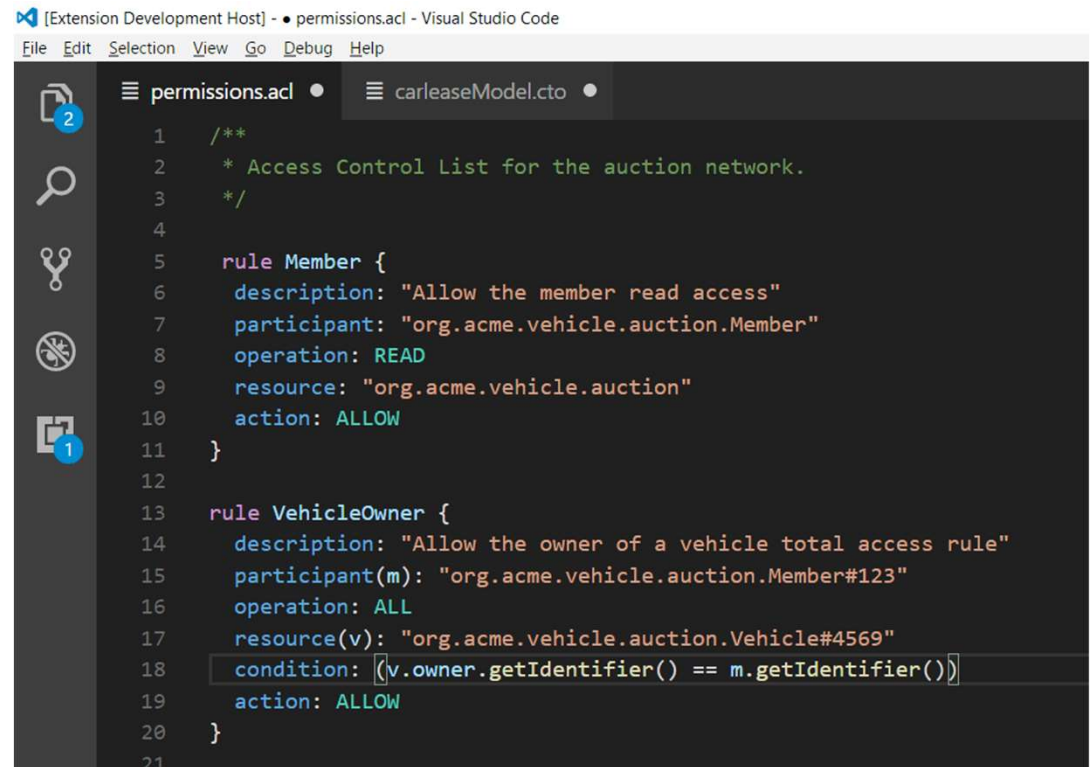
carauction-network 0.0.8

```
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
```


The ACL



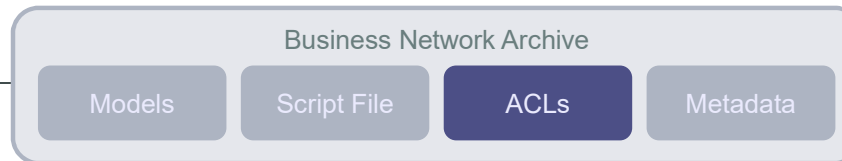
- Separates out access control from business logic making it simpler
 - Can build access control into the business logic if needed
- ACL engine evaluates rules for all access to assets
 - Top down checking
 - If no rule then denies access



```
[Extension Development Host] - • permissions.acl - Visual Studio Code
File Edit Selection View Go Debug Help

2 permissions.acl • carleaseModel.cto •
1  /**
2   * Access Control List for the auction network.
3   */
4
5  rule Member {
6    description: "Allow the member read access"
7    participant: "org.acme.vehicle.auction.Member"
8    operation: READ
9    resource: "org.acme.vehicle.auction"
10   action: ALLOW
11 }
12
13 rule VehicleOwner {
14   description: "Allow the owner of a vehicle total access rule"
15   participant(m): "org.acme.vehicle.auction.Member#123"
16   operation: ALL
17   resource(v): "org.acme.vehicle.auction.Vehicle#4569"
18   condition: (v.owner.getIdentifier() == m.getIdentifier())
19   action: ALLOW
20 }
21
```

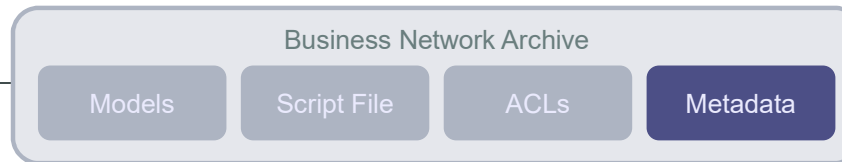
The ACL



- 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
 - If no rule then denies access

```
[Extension Development Host] - • permissions.acl - Visual Studio Code
File Edit Selection View Go Debug Help

2 permissions.acl • carleaseModel.cto
1  /**
2   * Access Control List for the auction network.
3   */
4
5  rule Member {
6    description: "Allow the member read access"
7    participant: "org.acme.vehicle.auction.Member"
8    operation: READ
9    resource: "org.acme.vehicle.auction"
10   action: ALLOW
11 }
12
13 rule VehicleOwner {
14   description: "Allow the owner of a vehicle total access rule"
15   participant(m): "org.acme.vehicle.auction.Member#123"
16   operation: ALL
17   resource(v): "org.acme.vehicle.auction.Vehicle#4569"
18   condition: {v.owner.getIdentifier() == m.getIdentifier()}
19   action: ALLOW
20 }
21
```



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

carauktion-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

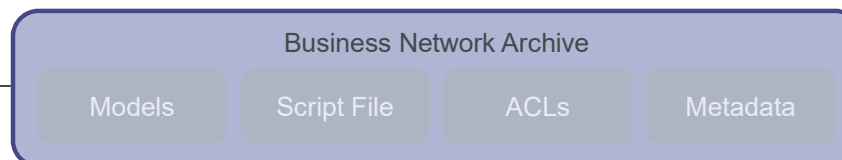
The easiest way to interact with the demo is using our work-in-progress [Hyperledger Composer web application](#). Hyperledger Composer allows you to define a business network (defining the data model and writing transaction processing logic), manage assets & participants and submit transactions.

The data model for the auction business network is defined in a CTO model file, managed in GitHub [here](#).

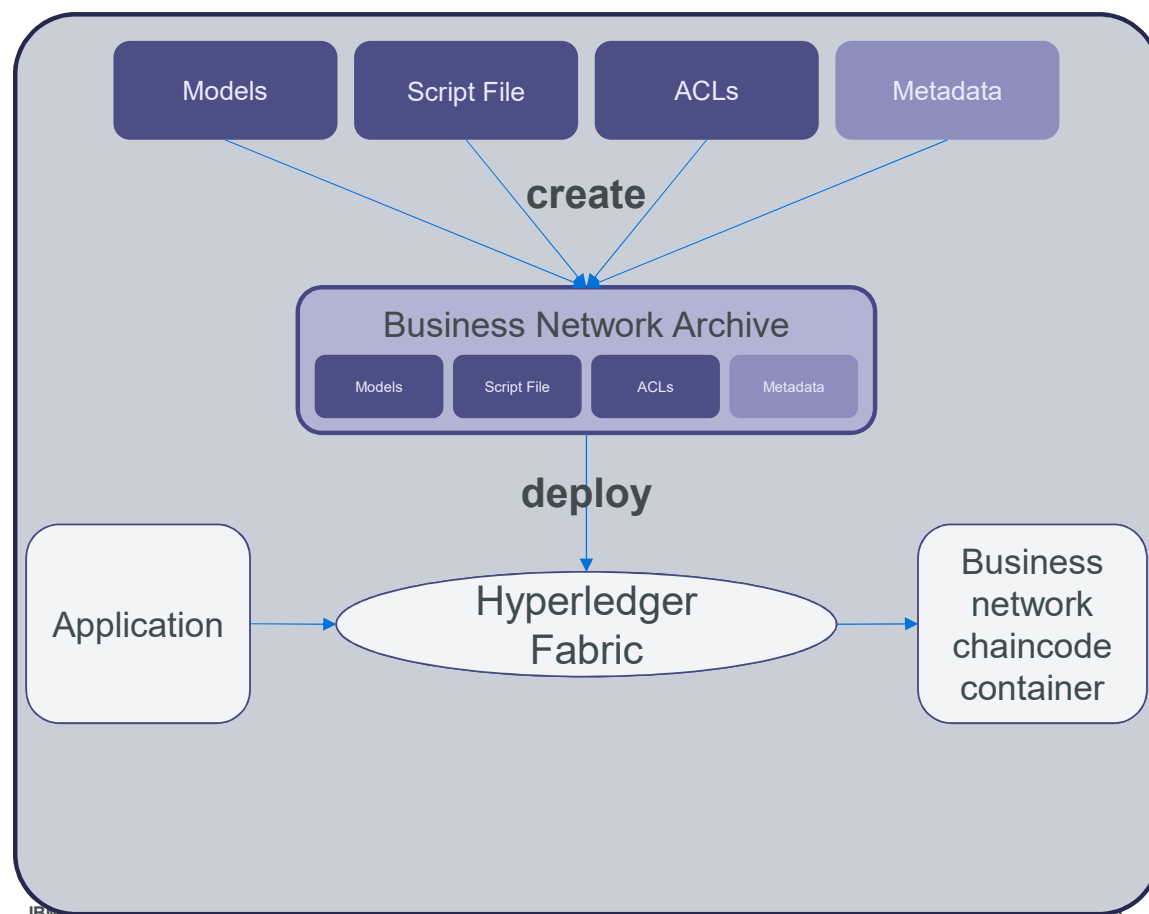
The data model is very simple (less than 50 lines). It defines the structure of the assets, participants and transactions for a very simple auction.

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

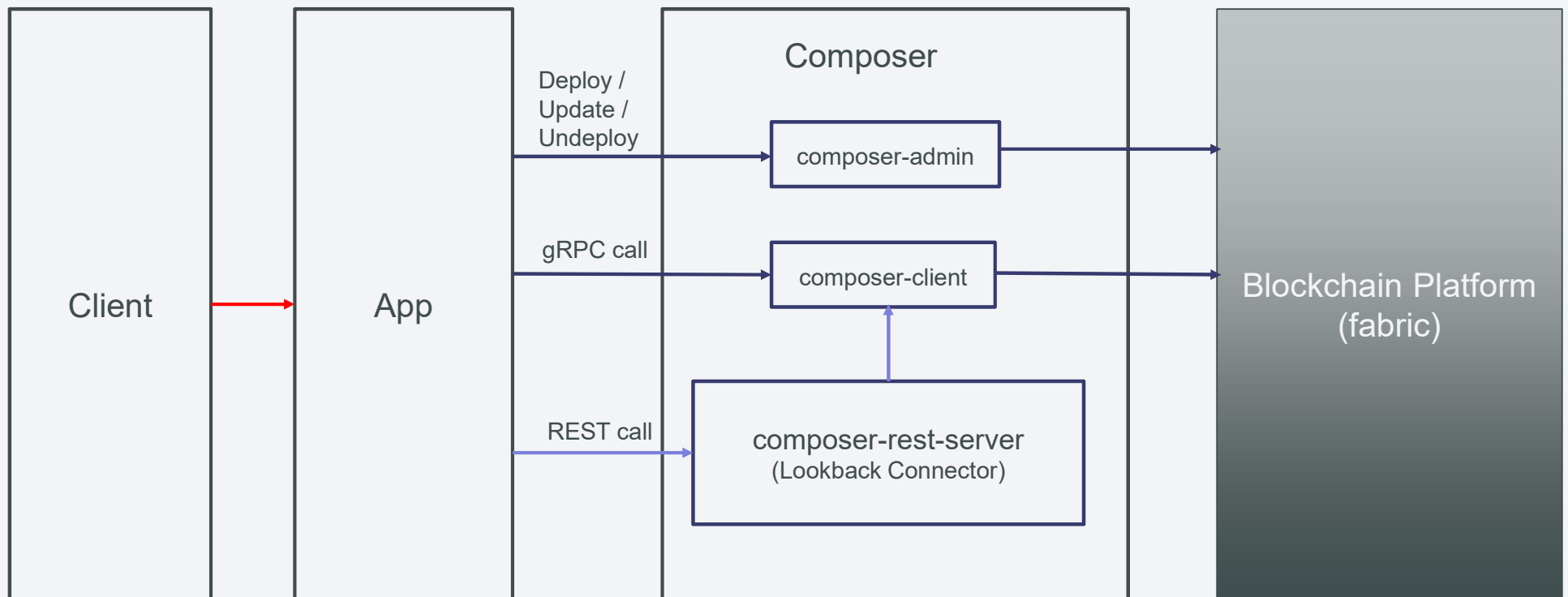
The Archive



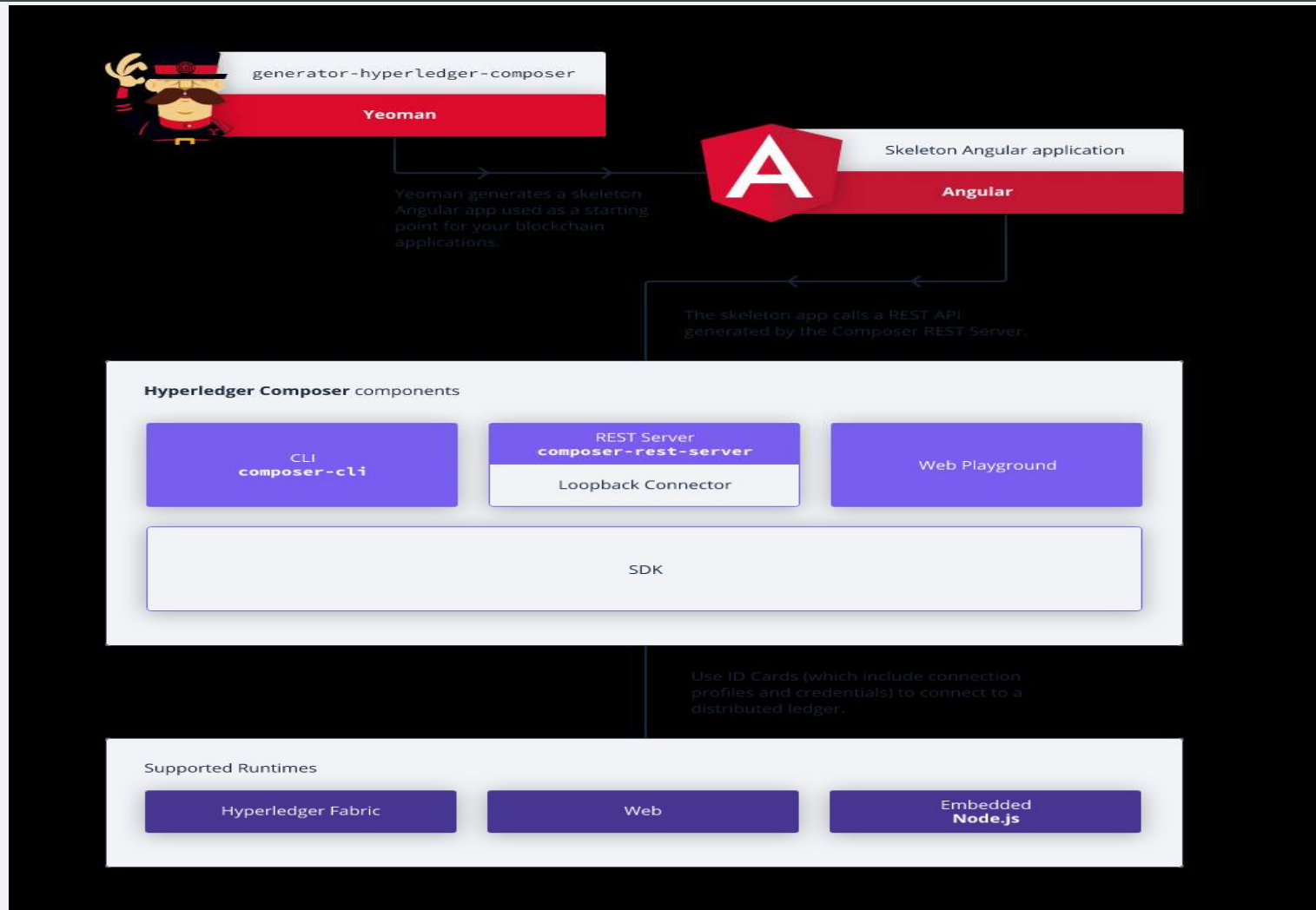
- The Business Network Archive packages up the project for deployment to a runtime
- Can be deployed using CLI tools
- Can be packed/unpacked by a developer to see its contents, check it, work on it or send it.
- At deployment the archive is executed within a chaincode container



High Level Architecture with Hyperledger Composer



Composer Components



Extensive, Familiar, Open Development Toolset

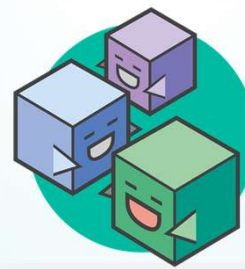


```
asset Animal identi
  o String animal
  o AnimalType sp
  o MovementStatu
  o ProductionTyp
```

Data modelling



JavaScript
business logic



Web playground

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



Client libraries



Editor support

```
$ composer
```

CLI utilities



Code generation

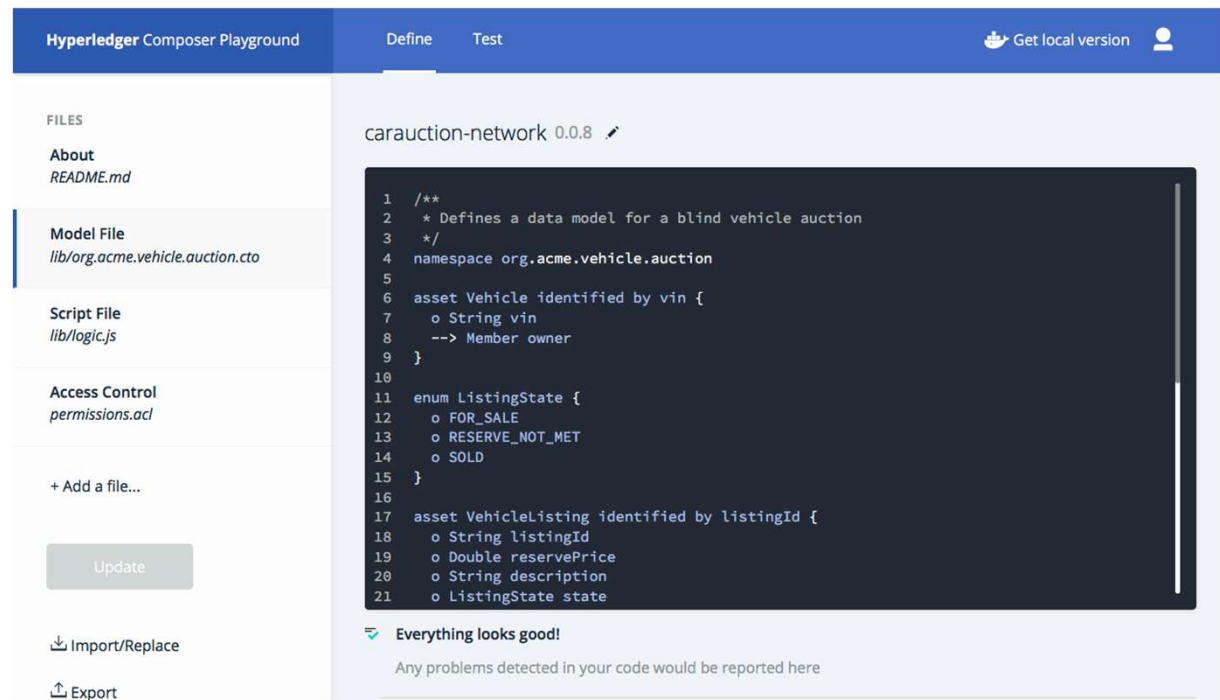


Existing systems and
data

Modelling Business Networks: Composer Playground



- A interactive web tool for the development (and test) of business networks without installing anything
- Developers & technical analysts
- Create business network definition
- 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



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

Testing Business Networks: Composer Playground



- **Test** tab on playground
- Dynamically reflects defined model
 - Creates default registries
- Create, read, update, and delete resources interactively
- Submit transactions interactively
- Fabric and browser-only modes
- Multiple environments, e.g. test, prod

The screenshot displays the Hyperledger Composer Playground interface. The top navigation bar includes the title 'Hyperledger Composer Playground', tabs for 'Define' and 'Test' (with 'Test' being the active tab), a 'Get local version' button, and a user profile icon. The left sidebar contains a menu with sections: 'PARTICIPANTS' (listing 'Auctioneer' and 'Member'), 'ASSETS' (listing 'Vehicle' and 'VehicleListing', with 'Vehicle' selected), and 'TRANSACTIONS' (listing 'All Transactions'). A 'Submit Transaction' button is located at the bottom of the sidebar. The main content area, titled 'Asset registry for org.acme.vehicle.auction.Vehicle', features a '+ Create New Asset' button and a table with two columns: 'ID' and 'Data'. The table lists three assets with their respective IDs and JSON data, each with edit and delete icons.

ID	Data
vin:5055	<pre>{ "\$class": "org.acme.vehicle.auction.Vehicle", "vin": "vin:5055", "owner": "resource:org.acme.vehicle.auction.Member#email:8293" }</pre>
vin:5338	<pre>{ "\$class": "org.acme.vehicle.auction.Vehicle", "vin": "vin:5338", "owner": "resource:org.acme.vehicle.auction.Member#email:6788" }</pre>
vin:7627	<pre>{ "\$class": "org.acme.vehicle.auction.Vehicle", "vin": "vin:7627", "owner": "resource:org.acme.vehicle.auction.Member#email:8745" }</pre>

Model Development, Versioning and Distribution



- GitHub for development
 - Use npm package format for metadata
 - **package.json** version
 - model files and business networks
- npm for distribution
 - Distribute models and networks
 - Networks depend on models
 - Use npm **dependencies**
- Applications **npm install** BNDs
 - Ensures consistency when application connects to network
- See examples in **sample-networks** and **sample-models** repositories
 - These are loaded into playground

model


```
34 lines (33 sloc) | 659 Bytes
1  {
2    "name": "digitalproperty-model",
3    "version": "0.0.11",
4    "description": "Digital Property Network",
5    "scripts": {
```

business network
definition

```
40    "dependencies": {
41      "digitalproperty-model": "latest"
42    },
```

0.7.1 is the latest of 416 releases

github.com/hyperledger/composer

Apache-2.0 

- JavaScript and REST APIs available
- **composer-client** & **composer-admin** npm modules for app devs and admins respectively
- Programming model JSDoc
- Perform complex queries on the registry state (when backed by CouchDB on Hyperledger Fabric V1)
- Domain specific REST APIs also available (see later)

```
let factory = this.businessNetworkDefinition.getFactory();
owner = factory.newInstance('net.biz.digitalPropertyNetwork', 'Person', 'PID:1234567890');
owner.firstName = 'Fred';
owner.lastName = 'Bloggs';
```

```
let landTitle2 = factory.newInstance('net.biz.digitalPropertyNetwork', 'LandTitle', 'LID:6789');
landTitle2.owner = owner;
landTitle2.information = 'A small flat in the city';
```

```
this.titlesRegistry.addAll([landTitle1, landTitle2]);
```

```
this.bizNetworkConnection.getParticipantRegistry('net.biz.digitalPropertyNetwork.Person')
  .then((personRegistry) => {
    return personRegistry.add(owner);
  })
```

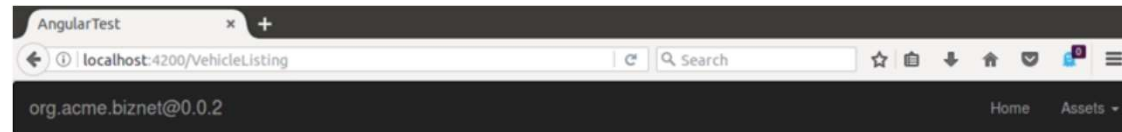
```
query Q1{
  description: "Select all drivers older than 65."
  statement:
    SELECT org.acme.Driver
    WHERE (age>65)
}
```

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
```



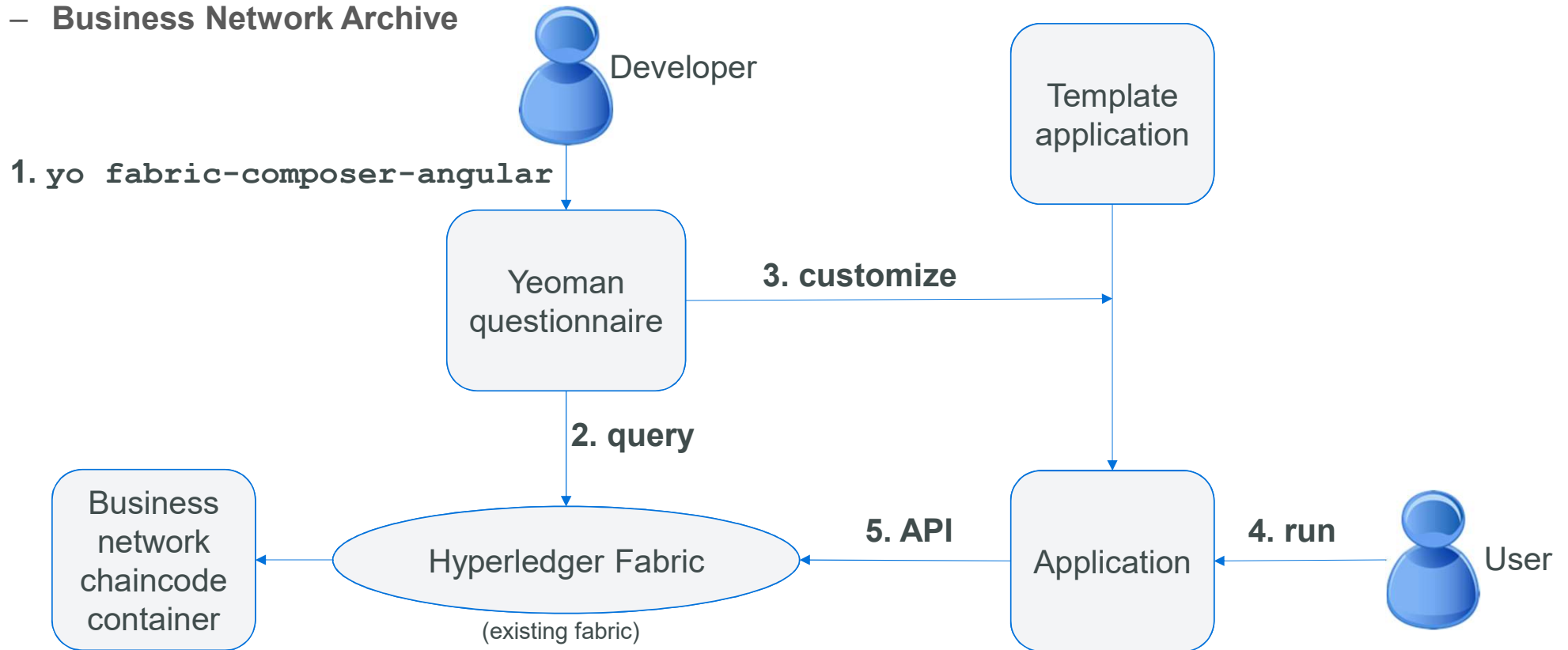
VehicleListing

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>

Generating APIs and sample applications



– Business Network Archive



- 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]
```

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 and Inbound (where supported by middleware)

angular-app

Auctioneer : A participant named Auctioneer

Show/Hide | List Operations | E

CloseBidding : A transaction named CloseBidding

Show/Hide | List Operations | E

Member : A participant named Member

Show/Hide | List Operations | E

Offer : A transaction named Offer

Show/Hide | List Operations | E

Vehicle : An asset named Vehicle

Show/Hide | List Operations | E

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

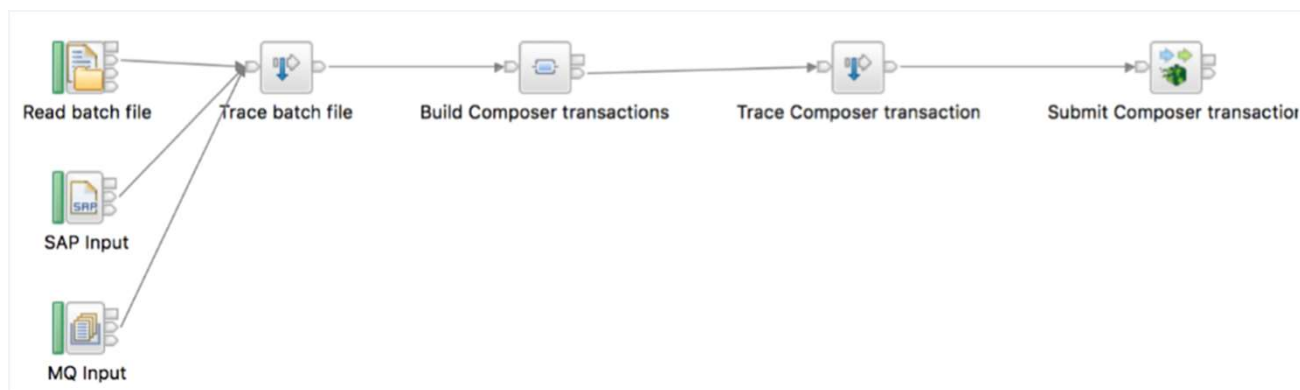
Request URL

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

Hyperledger Composer Outlook

- Still early in product lifecycle
- Lots of improvements planned
 - See <https://github.com/hyperledger/composer/issues>
- An active development community
 - Open community calls every two weeks
 - Rocket Chat
 - Stack Overflow
- Get involved!

Hyperledger Rocket.Chat

You will need a [Linux Foundation ID](#), or alternatively you can log in with Facebook, GitHub, Google, or OpenStack.

Let's chat

Stack Overflow

Ask questions in Stack Overflow with the tag `#hyperledger-composer`.

Ask now

Contribute to the Project

GitHub

Check out the code, feel free to get involved.

GitHub

Community Call

Join our weekly open community calls.

Learn how





Hyperledger Composer

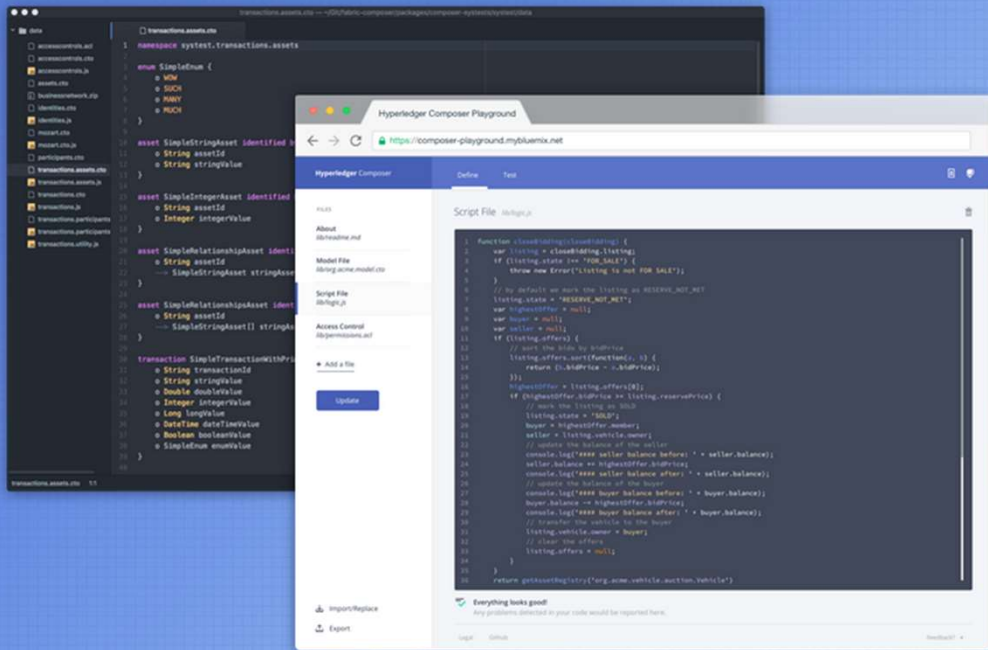
Tutorials Docs Community

Build Blockchain applications and business networks your way

[Install Hyperledger Composer](#)[Try it online](#)

or learn more about a typical [Composer architecture...](#)

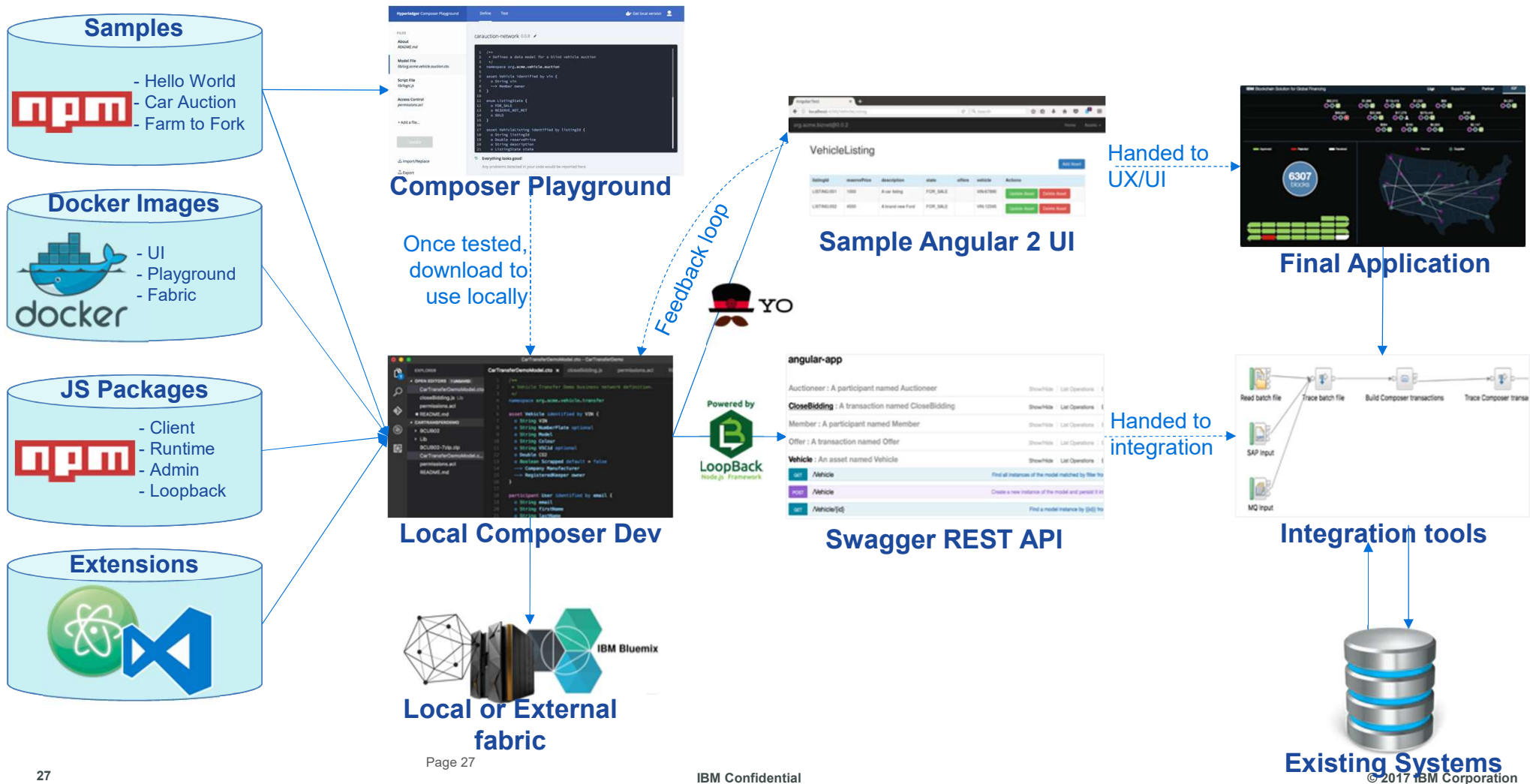




```
1 namespace systest.transactions.assets
2
3 enum SimpleEnum {
4   a NONE
5   b SUCH
6   c ALLOT
7   d NULL
8 }
9
10 asset SimpleStringAsset identified by
11   a String assetId
12   a String stringValue
13 }
14
15 asset SimpleIntegerAsset identified by
16   a String assetId
17   a Integer integerValue
18 }
19
20 asset SimpleRelationshipAsset identified by
21   a String assetId
22   a SimpleStringAsset stringAsset
23 }
24
25 asset SimpleRelationshipAsset identified by
26   a String assetId
27   a SimpleStringAsset[] stringArray
28 }
29
30 transaction SimpleTransactionWithParams {
31   a String transactionId
32   a String stringValue
33   a Double doubleValue
34   a Integer integerValue
35   a Long longValue
36   a DateTime dateTimeValue
37   a Boolean booleanValue
38   a SimpleEnum enumValue
39 }
```

```
1 function classedSimpleTransaction() {
2   var listing = classedListing(listing);
3   if (listing.state !== "FOR_SALE") {
4     throw new Error("Listing is not FOR_SALE");
5   }
6   // By default we mark the listing as RESERVE_NOT_MET
7   listing.state = "RESERVE_NOT_MET";
8   var highestOffer = null;
9   var buyer = null;
10   if (listing.offers) {
11     // Sort the bids by bidPrice
12     listing.offers.sort(function(a, b) {
13       return (b.bidPrice - a.bidPrice);
14     });
15     highestOffer = listing.offers[0];
16     if (highestOffer.bidPrice > listing.reservePrice) {
17       // Mark the listing as SOLD
18       listing.state = "SOLD";
19       buyer = highestOffer.number;
20       seller = listing.vehicle.owner;
21       // Update the balance of the seller
22       console.log("New seller balance before: " + seller.balance);
23       seller.balance = highestOffer.bidPrice;
24       console.log("New seller balance after: " + seller.balance);
25       // Update the balance of the buyer
26       console.log("New buyer balance before: " + buyer.balance);
27       buyer.balance = highestOffer.bidPrice;
28       console.log("New buyer balance after: " + buyer.balance);
29       // Transfer the vehicle to the buyer
30       listing.vehicle.owner = buyer;
31       // Close the offers
32       listing.offers = null;
33     }
34   }
35   return getAssetRegistry("org.hyperledger.composer.system:Vehicle");
36 }
```

Getting Started With Hyperledger Composer



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.

```
5 1
6  asset Vehicle identified by vin {
7     o String vin
8     --> Member owner 2
9  }
```

```
17
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 }
26
```

Asset registry for org.acme.vehicle.auction.Vehicle

+ Create New Asset

ID	DATA
CAR:001	<pre>{ "\$class": "org.acme.vehicle.auction.Vehicle", "vin": "CAR:001", "owner": "mr.bean@uk.ibm.com" }</pre>

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

```
39
40 transaction Offer identified by transactionId {
41   o String transactionId
42   o Double bidPrice
43   --> VehicleListing listing
44   --> Member member
45 }
46
```

Default Transaction Registry

ID	DATA
4c95e0a3-1290-4f0b-91b6-3a8d7f3dd3d5	<pre>{ "\$class": "org.acme.vehicle.auction.CloseBidding", "transactionId": "4c95e0a3-1290-4f0b-91b6-3a8d7f3dd3d5", "listing": "LISTING:001", "timestamp": "2017-03-19T11:35:56.622Z" }</pre> Show All
c2f7cf1b-1e11-439e-921b-6a1287292418	<pre>{ "\$class": "org.acme.vehicle.auction.Offer", "transactionId": "c2f7cf1b-1e11-439e-921b-6a1287292418", "bidPrice": 1000, "listing": "LISTING:001", "member": "mrs.brown" }</pre> Show All
2570c64b-0721-4b44-9ed3-11fb9f364a7b	<pre>{ "\$class": "org.acme.vehicle.auction.Offer", "transactionId": "2570c64b-0721-4b44-9ed3-11fb9f364a7b", "bidPrice": 700, }</pre>

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

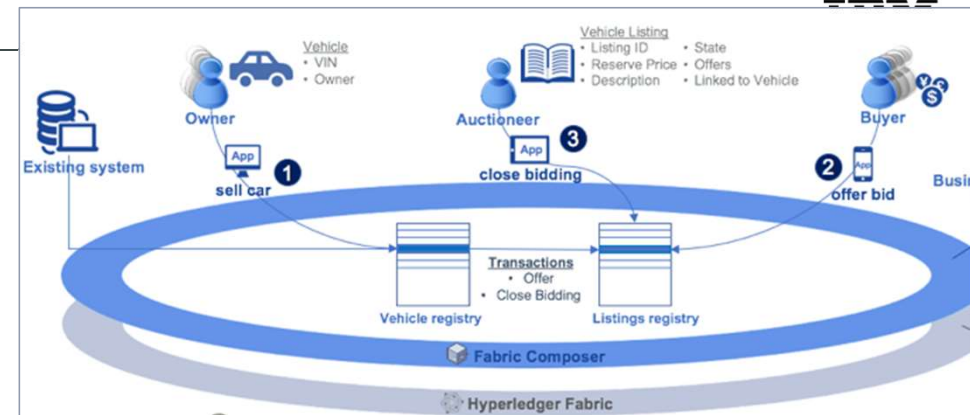


- Separate ACL from application logic
- Defined in a **permissions.acl** file in business network definition
- Flexible model allowing both type and instance access e.g. 'create cars' and 'scrap **my** car'
- Standard ACL model
 - Includes optional condition for more sophisticated ACL checking
- Executed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

```
11
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 }
28
```


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



- Set up a POC environment on IBM cloud: <https://ibm-blockchain.github.io/setup/>
- SSL/TLS enablement: <https://hyperledger.github.io/composer/integrating/getting-started-rest-api.html>
- Authentication: <https://hyperledger.github.io/composer/integrating/enabling-rest-authentication.html>
- Multiuser: <https://hyperledger.github.io/composer/integrating/enabling-multiuser.html>
- External calls: <https://hyperledger.github.io/composer/integrating/call-out.html>
- Multi org set up: <https://hyperledger.github.io/composer/tutorials/deploy-to-fabric-multi-org>
- Hyperledger explorer: <https://github.com/hyperledger/blockchain-explorer>

Thank You!

