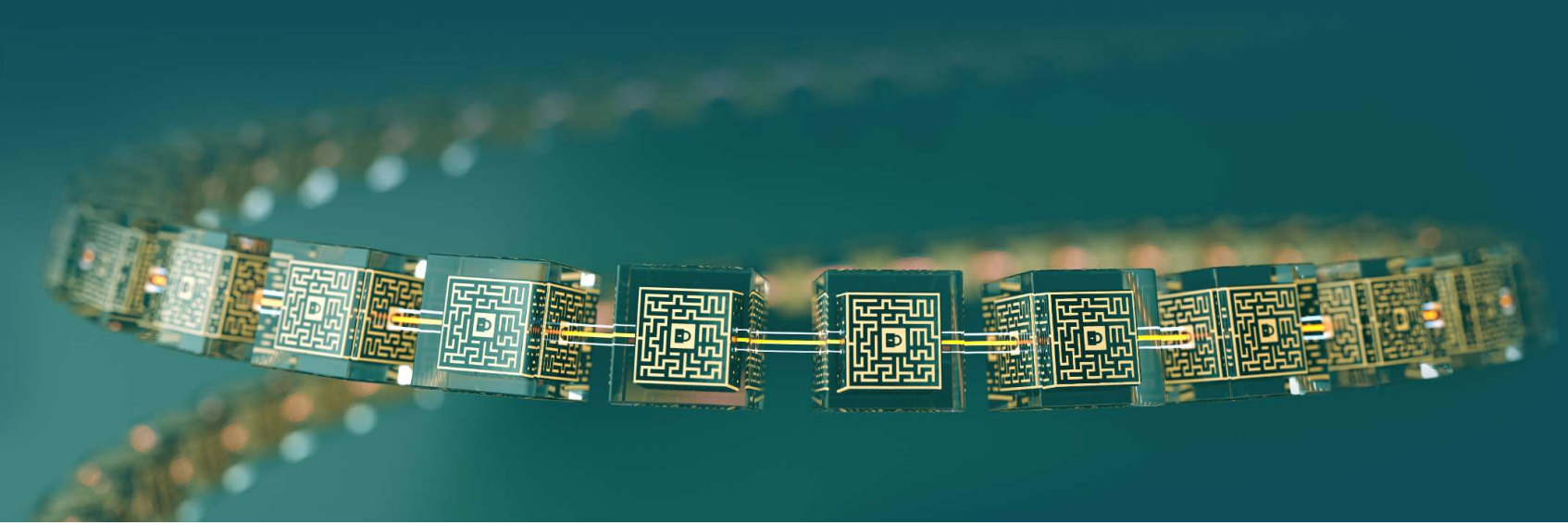
IBM Blockchain

August 2017

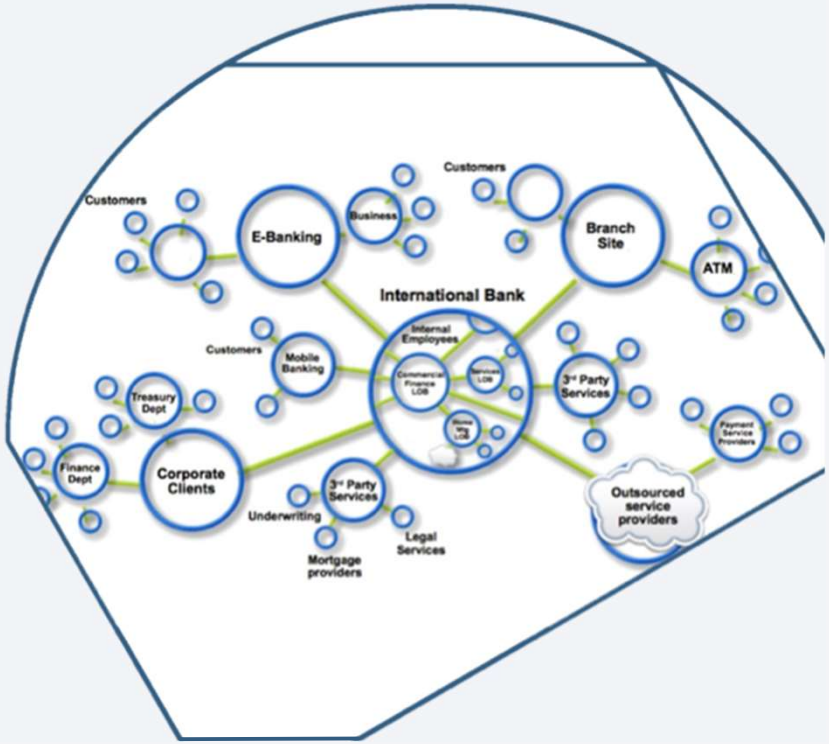
Blockchain Composed



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?

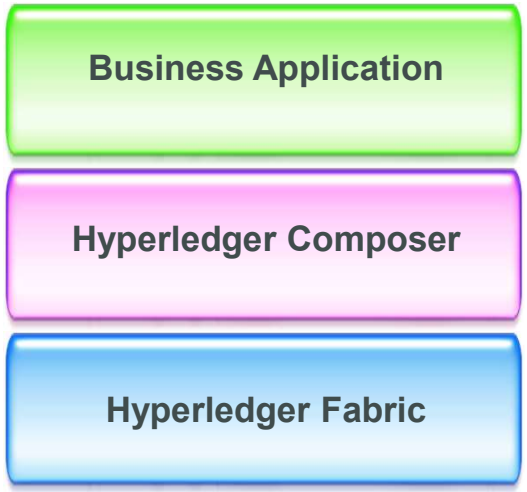
https://hyperledger.github.io/composer/

- 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



Benefits of Hyperledger Composer

Increases understanding

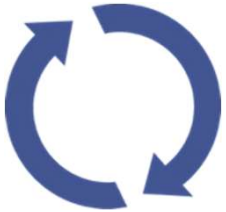
Bridges simply from business concepts to blockchain

Saves time

Develop blockchain applications more quickly and cheaply

Reduces risk

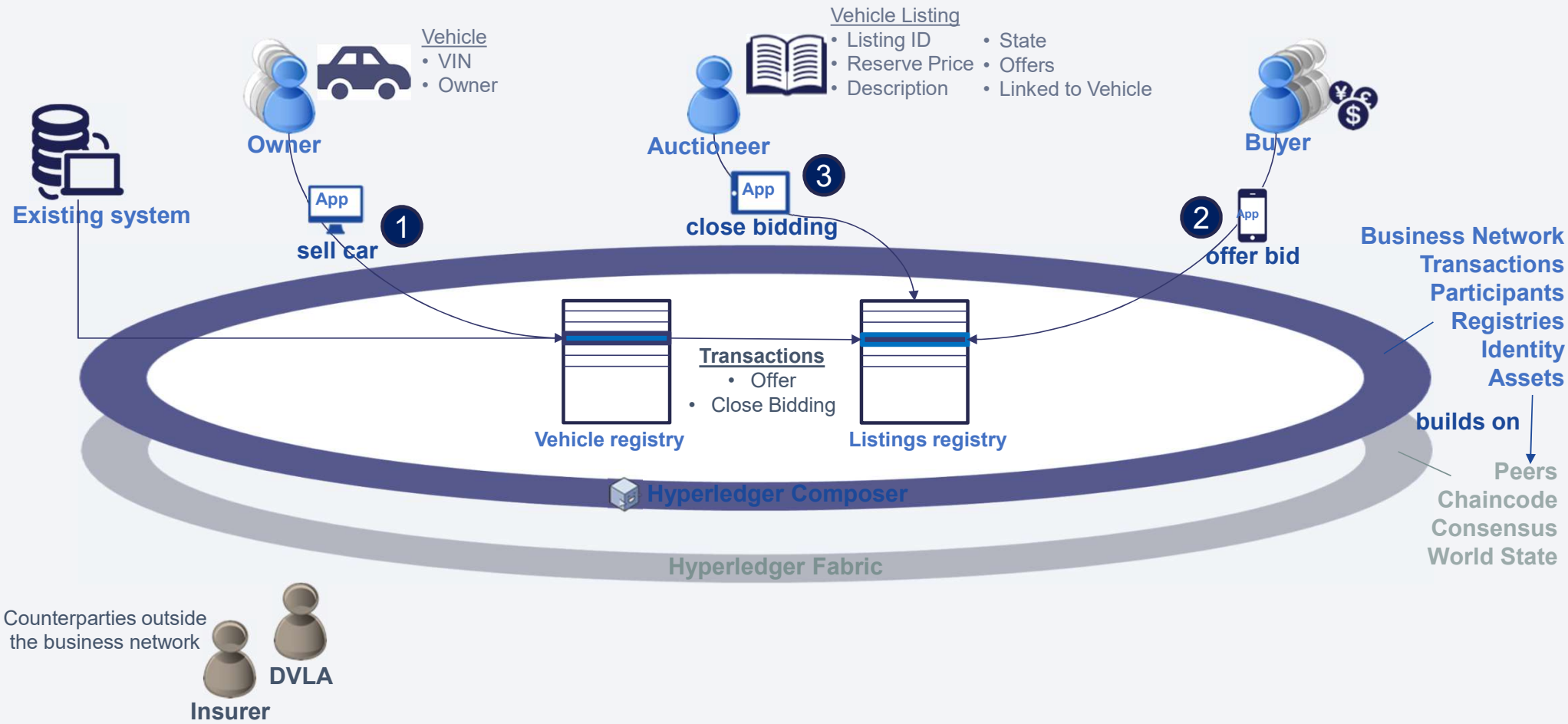
Well tested, efficient design conforms to best practice



Increases flexibility

Higher level abstraction makes it easier to iterate

An Example Business Network – Car Auction Market

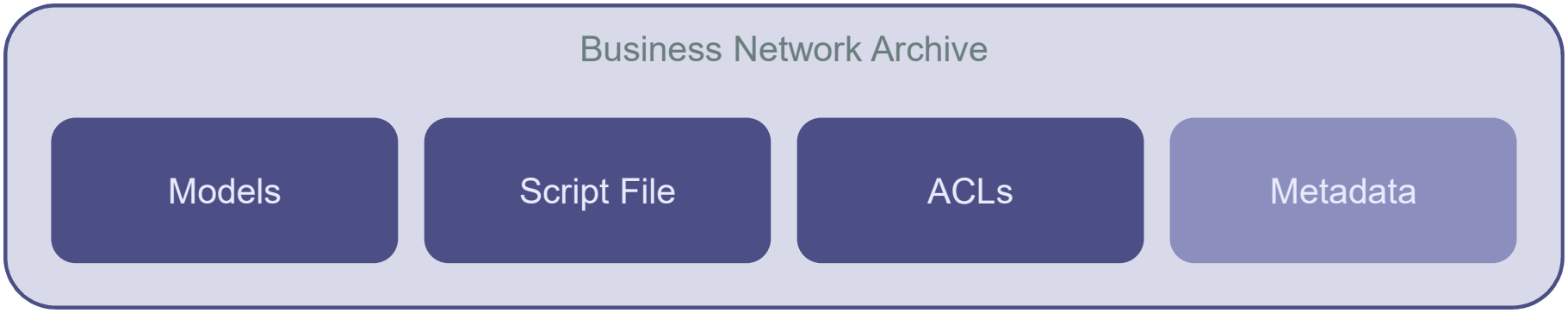


Conceptual Components and Structure

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

- 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
}
```

The Script File

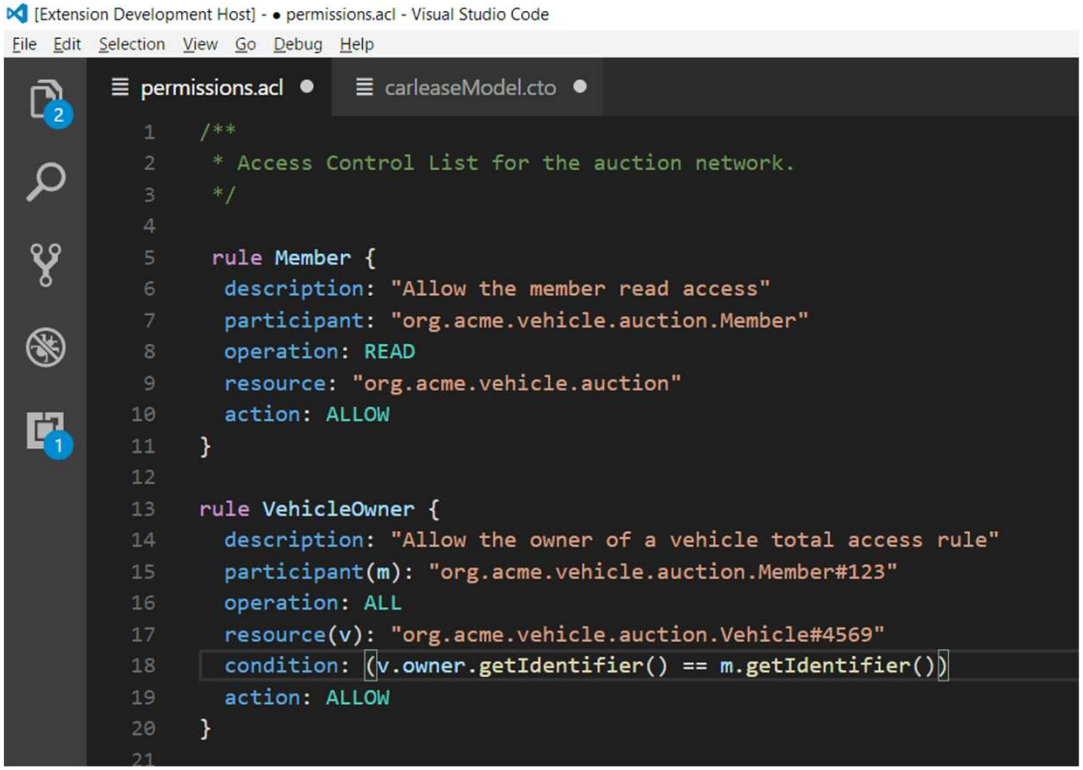
- Provide transaction implementation logic
- Specified in Javascript
- Designed for any reasonable Javascript developer to pick up easily
- Can publish external events from within a transaction so that client applications can respond

caraucion-network 0.0.8

```
/**
 * 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') {
        throw new Error('Listing is not FOR SALE');
    }
    if (listing.offers == null) {
        listing.offers = [];
    }
    listing.offers.push(offer);
    return getAssetRegistry('org.acme.vehicle.auction.VehicleListing')
        .then(function(vehicleListingRegistry) {
            // save the vehicle listing
            return vehicleListingRegistry.update(listing);
        });
}
```

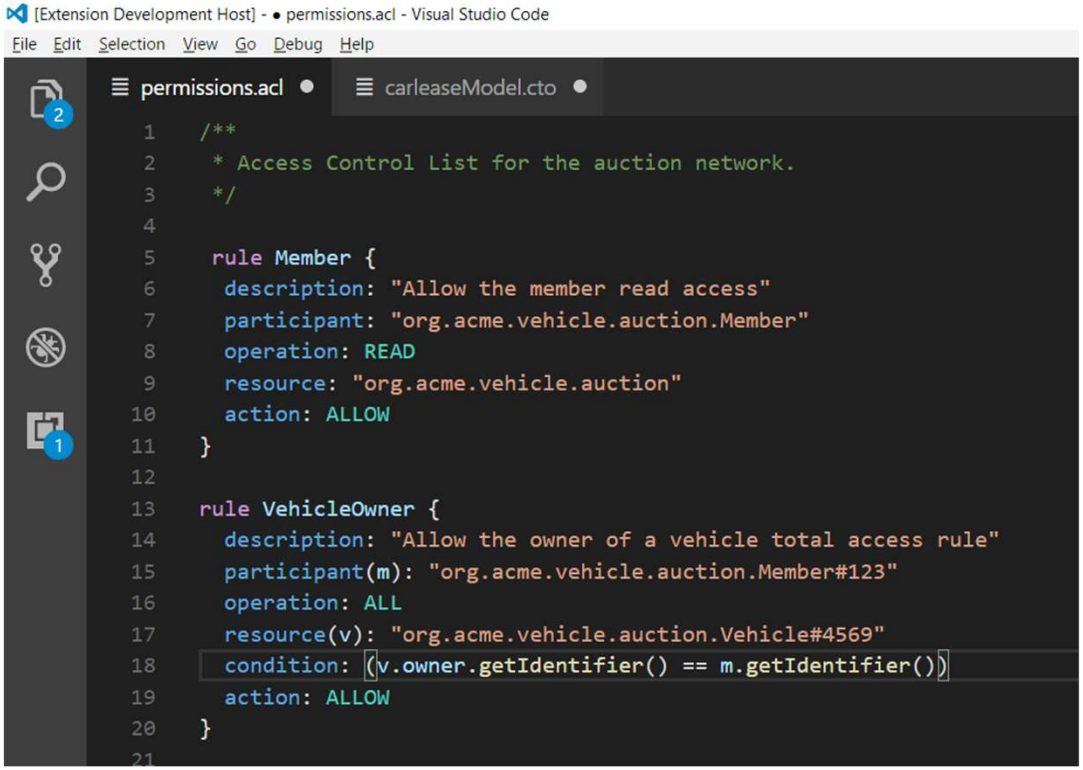
The ACL

- 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



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



Metadata

- 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

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.

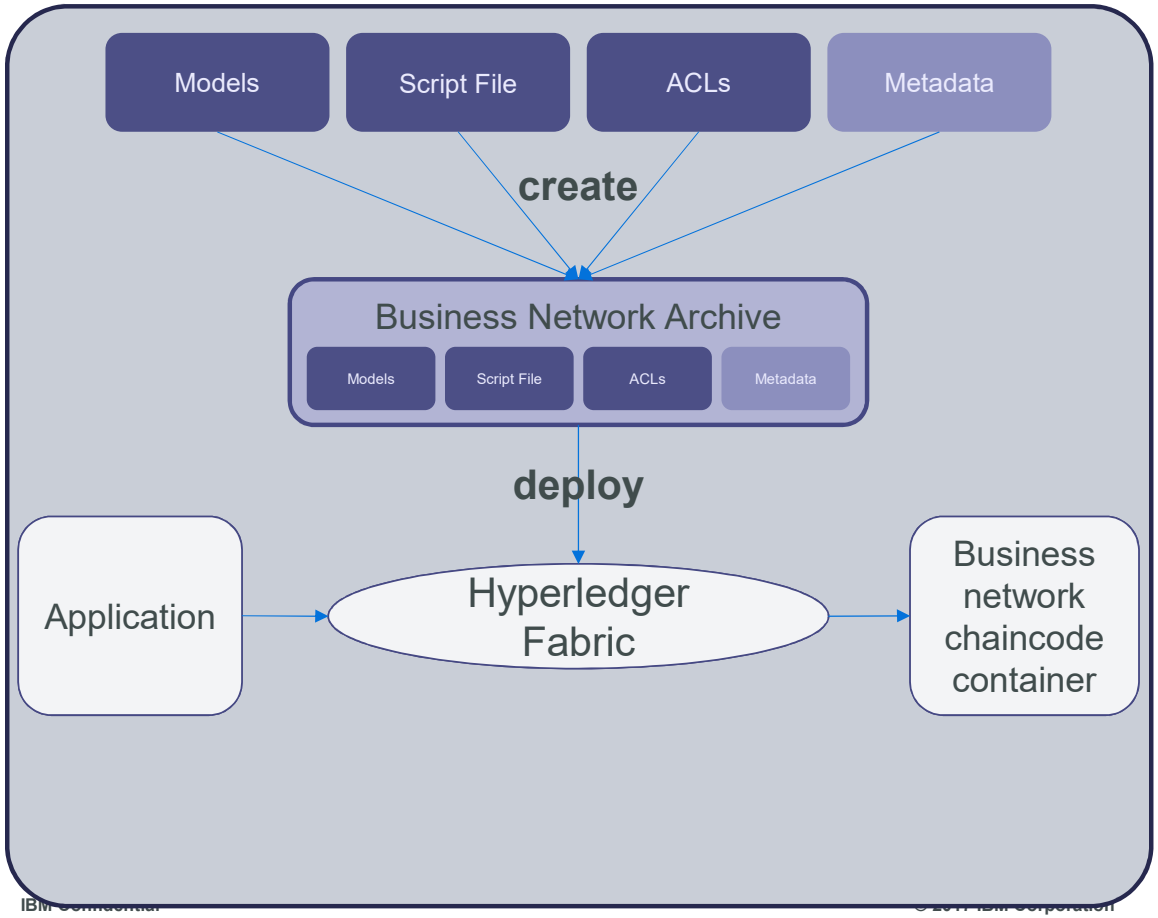
The data model for the auction business network is defined in a CTO model file, managed in GitHub here.

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.

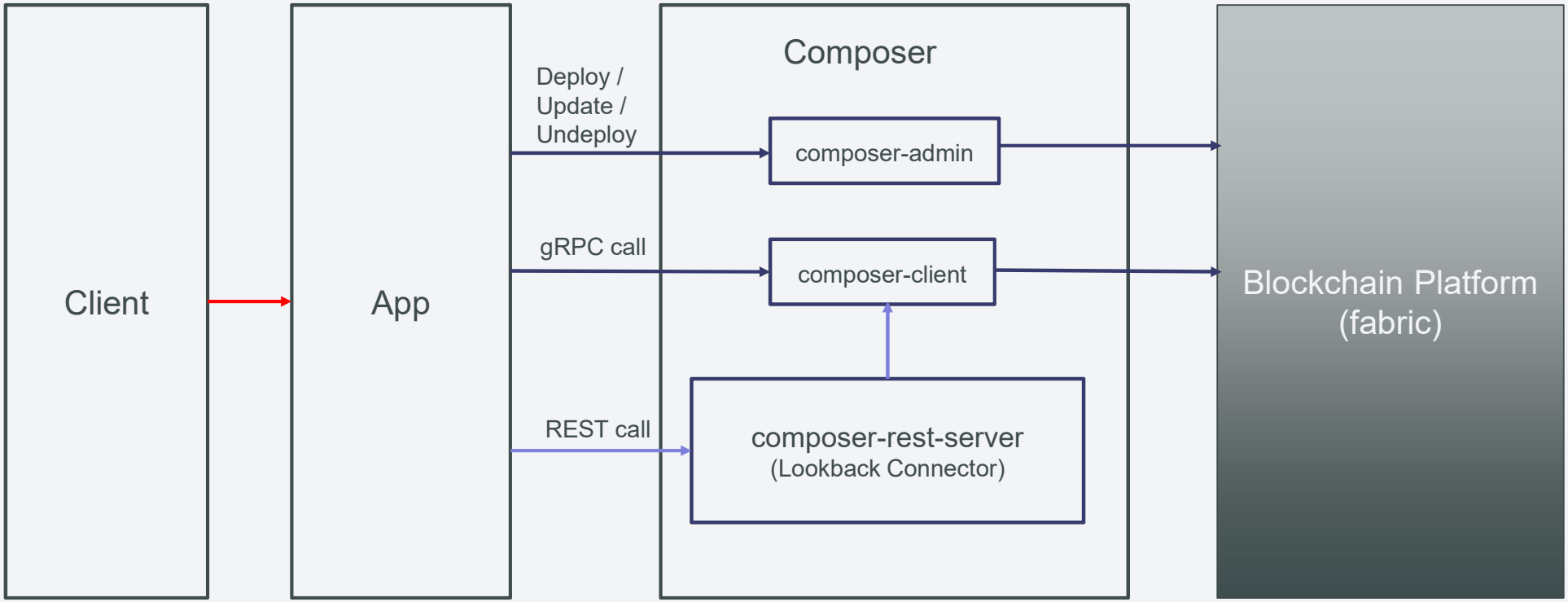
The business logic is defined in a single Javascript file here. The logic consists of two Javascript functions that are automatically invoked by the Hyperledger

The Archive

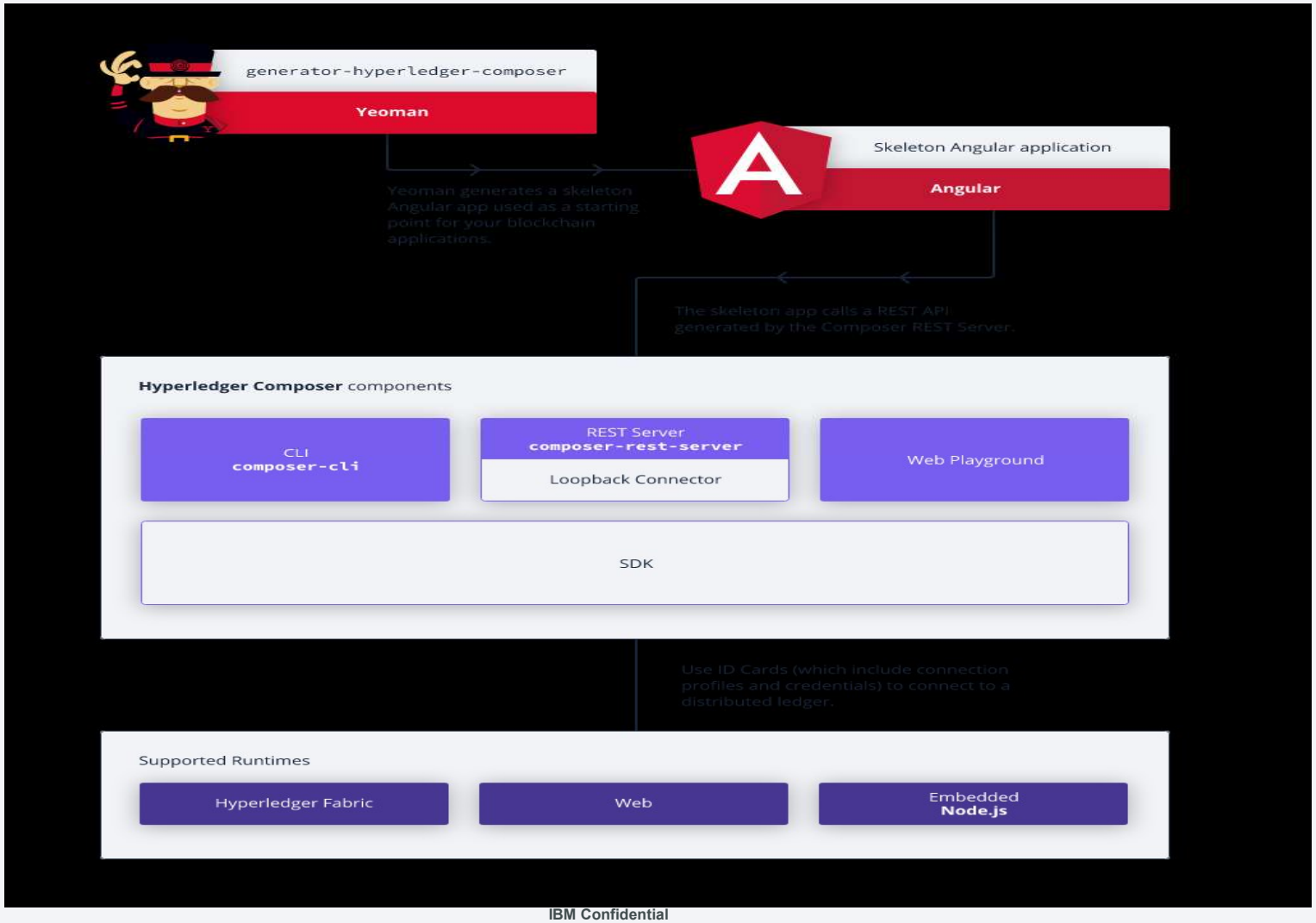
- The Business Network Archive packages up the project to for deployment to a runtime
- Can be deployed using CLI tools
- Can be packed/unpacked by a developer to see its contents, check it, work on it or send it.
- At deployment the archive is executed within a chaincode container



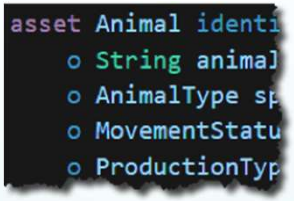
High Level Architecture with Hyperledger Composer



Composer Components

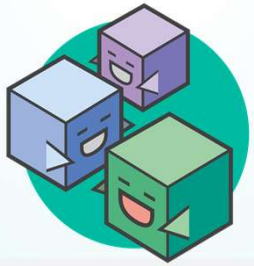


Extensive, Familiar, Open Development Toolset



Data modelling

JavaScript business logic



Web playground

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

Client libraries

Editor support

```
$ composer
```

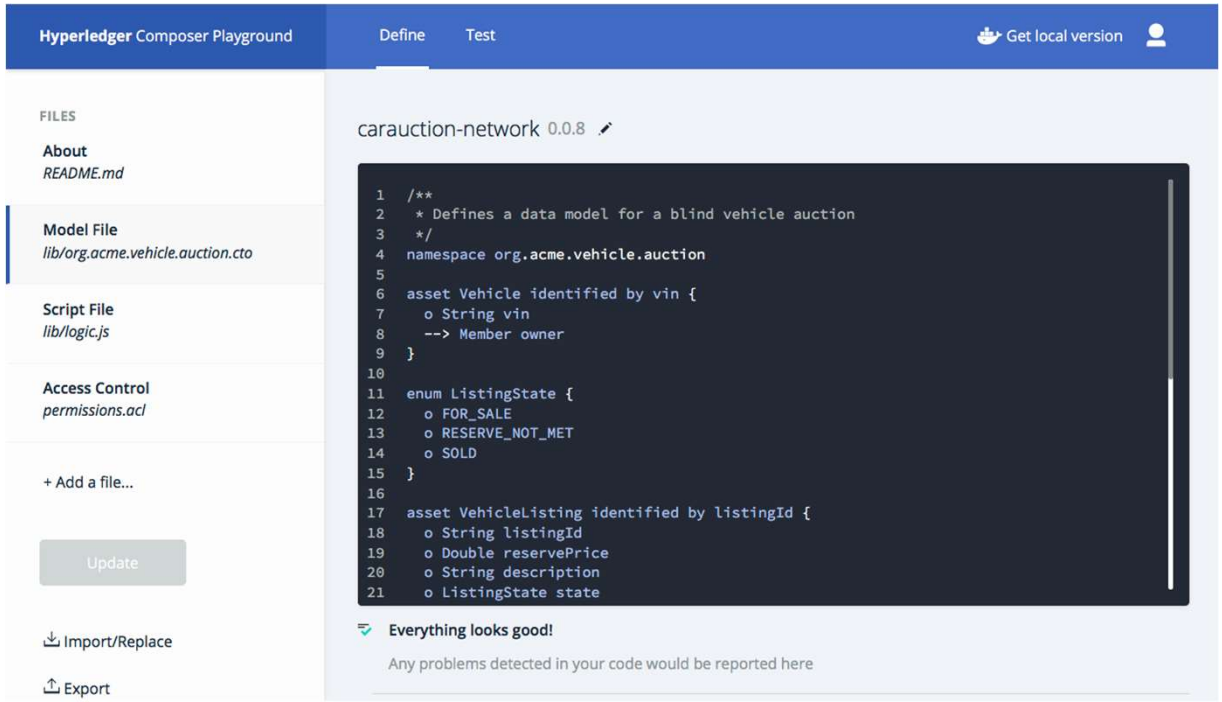
CLI utilities

Code generation

Existing systems and data

Modelling Business Networks: Composer Playground

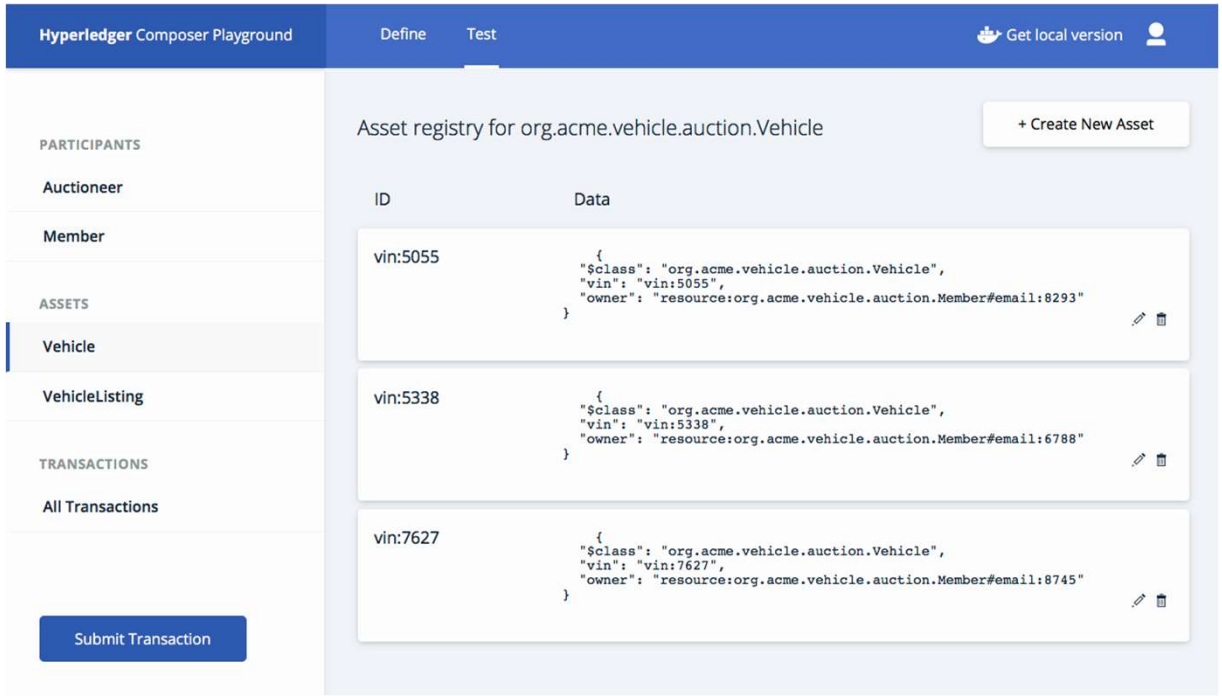
- A interactive web tool for the development (and test) of business networks without installing anything
- Developers & technical analysts
- Create business network definition
- DSL for specific assets, participants & transactions in your network
  - Live content assist, syntax checking
- Non-web options 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



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

Applications and Javascript APIs

- 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]);
```

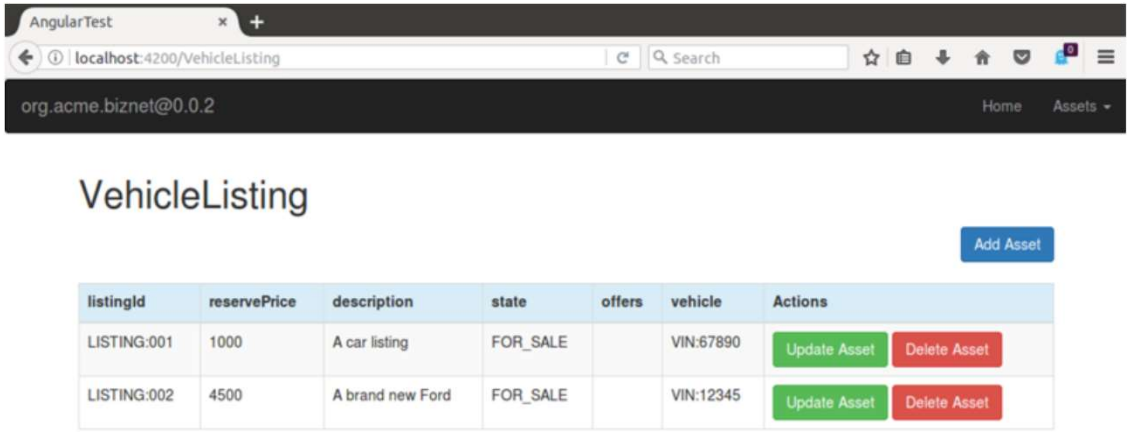
```
this.bizNetworkConnection.getParticipantRegistry('net.biz.digitalPropertyNetwork.Person')
  .then((personRegistry) => {
      return personRegistry.add(owner);
  })
```

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

Generating APIs and sample applications

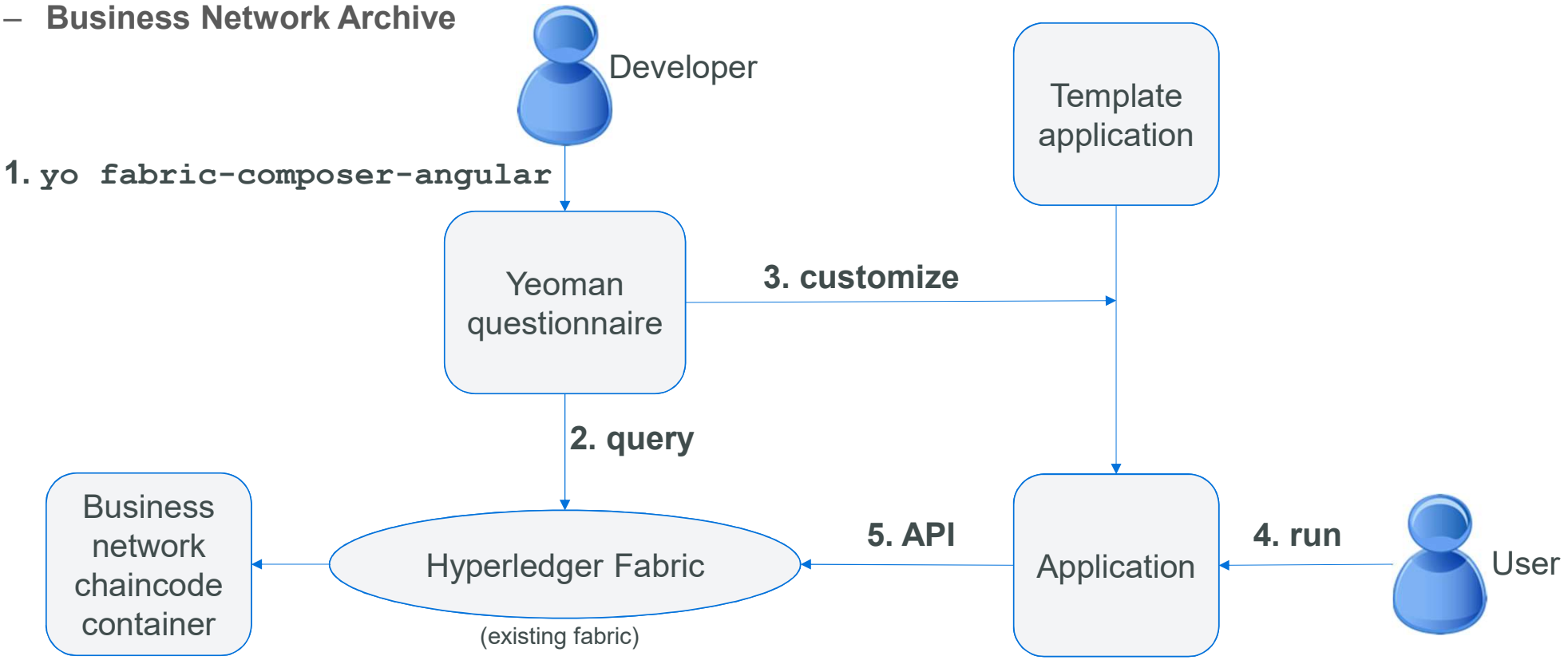
- Programmable business networks
  - Direct consequence of a deployed **model**
- Query network to generate domain APIs
- Generate sample application for deployed business network
- Yeoman questionnaire
  yo fabric-composer[:angular]
- Also generate test cases using **mocha** and **chai** node.js test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



Generating APIs and sample applications

– **Business Network Archive**



Command Line Interfaces

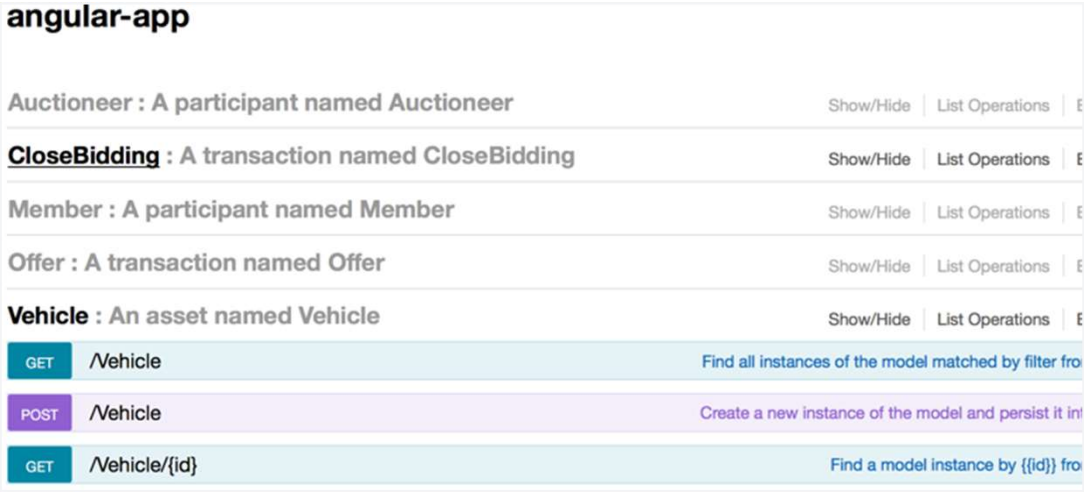
- Suite of commands to interact with an operational business network
- Target use is scripting and interactive operations
- Packaged as **composer-cli** in npm
- Generate full list with **composer –help.** Individual (sub) commands support **-help**
- CLI uses public APIs
  - Users can create their own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

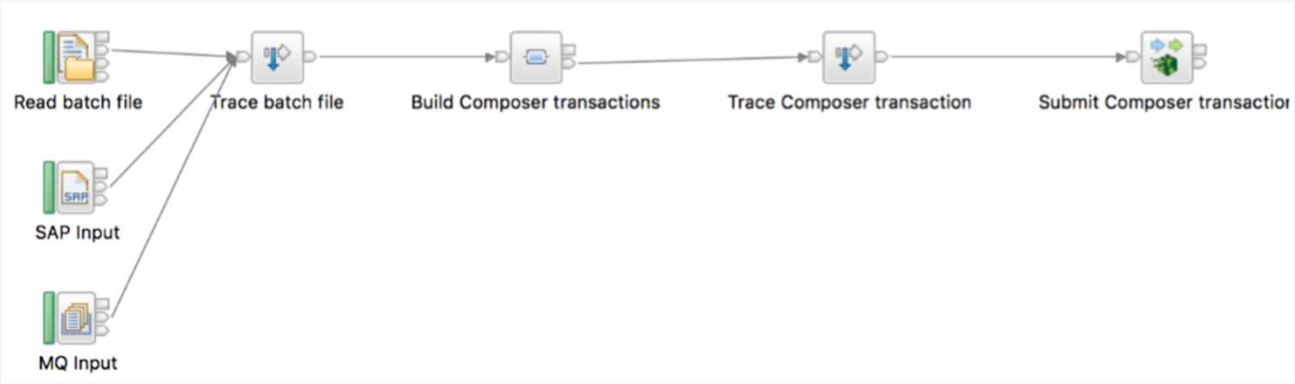
```
$> composer transaction submit –help

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]
```

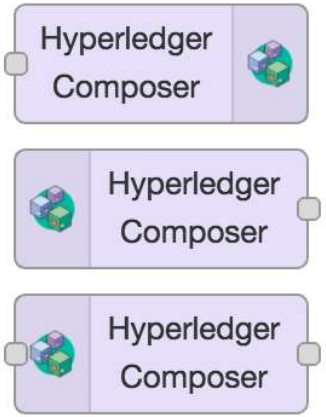
Loopback and REST Support

- 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)



Exploiting Loopback: Examples



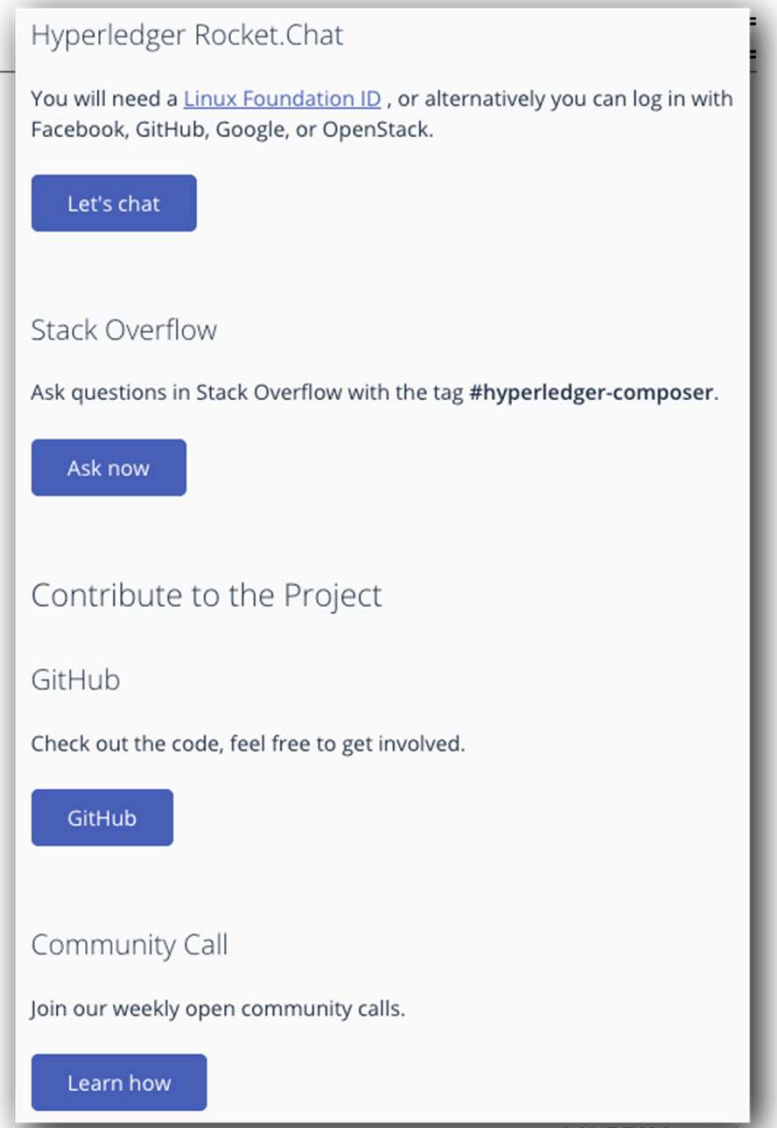


- **IBM Integration Bus**
  - IIB V10 contains Loopback connector
  - Example above takes input from file, SAP or MQ
  - Data mapping from CSV, BAPI/IDOC or binary form to JSON model definition

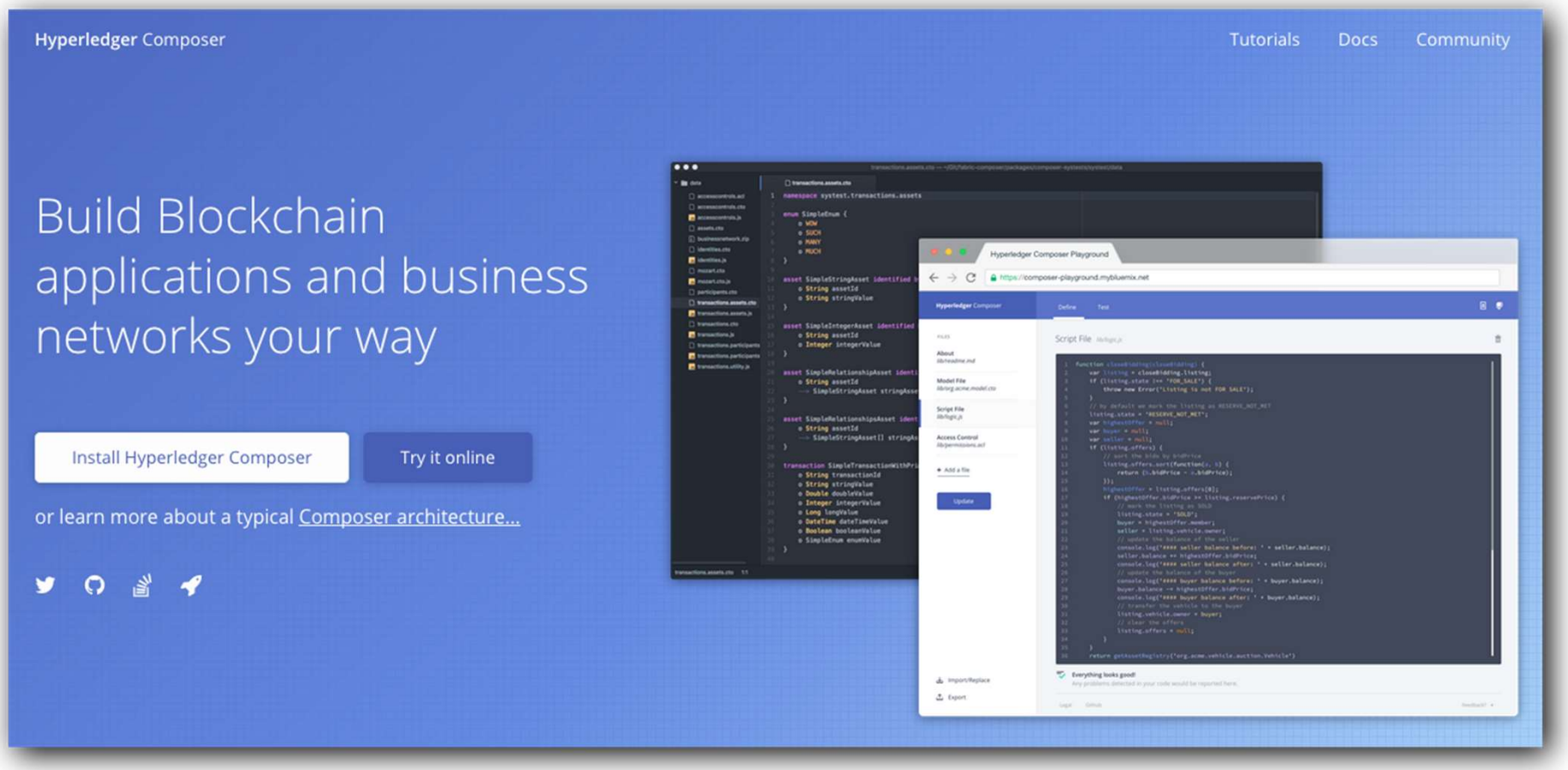
- **Node.RED**
  - Pre-built nodes available for Composer
  - Connect to hardware devices, APIs and online services
  - Install direct from Node.RED UI
    - Manage Palette -> Install -> node-red-contrib-composer

Hyperledger Composer Outlook

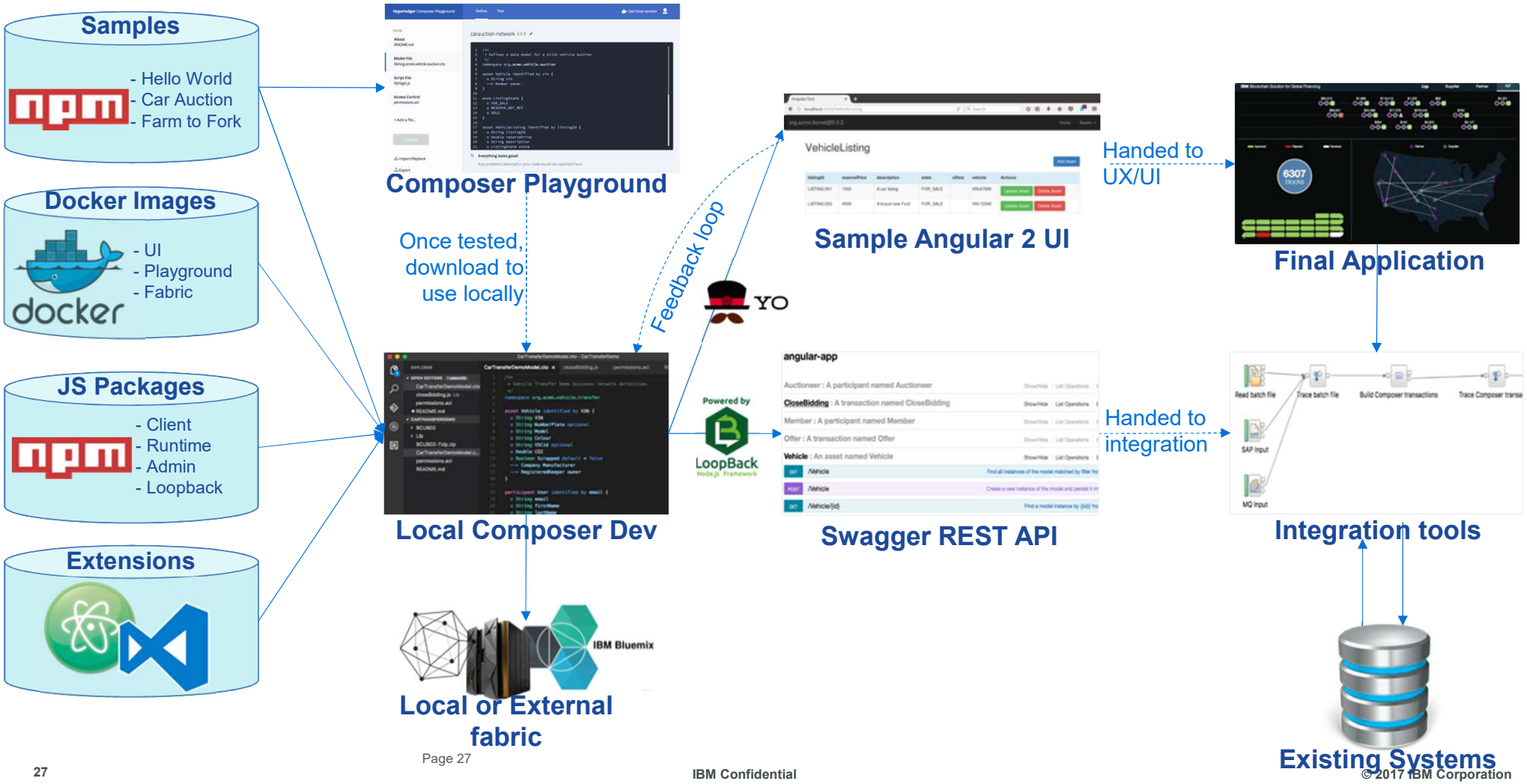
- Still early in product lifecycle
- Lots of improvements planned
  - See https://github.com/hyperledger/composer/issues
- An active development community
  - Open community calls every two weeks
  - Rocket Chat
  - Stack Overflow
- Get involved!



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



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.

1

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

2

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

3

4

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

+ Create New Asset

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

5

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

IBM

– 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

1 2

```
transaction Offer identified by transactionId {
  o String transactionId
  o Double bidPrice
  --> VehicleListing listing
  --> Member member
}
```

3

Default Transaction Registry 4

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

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

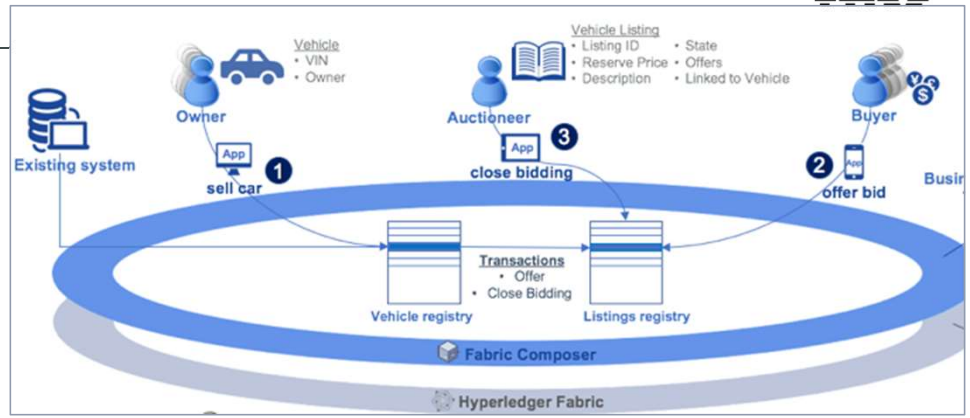
```
/**
 * Make an Offer for a VehicleListing
 * @param {org.acme.vehicle.auction.Offer} offer - the offer
 * @transaction
 */
function makeOffer(offer) {
    var listing = offer.listing;
    if (listing.state !== 'FOR_SALE') {
        throw new Error('Listing is not FOR SALE');
    }
    if (listing.offers == null) {
        listing.offers = [];
    }
    listing.offers.push(offer);
    return getAssetRegistry('org.acme.vehicle.auction.VehicleListing')
        .then(function(vehicleListingRegistry) {
            // save the vehicle listing
            return vehicleListingRegistry.update(listing);
        });
}
```

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

```
rule Member {
    description: "Allow the member read access"
    participant: "org.acme.vehicle.auction.Member"
    operation: READ
    resource: "org.acme.vehicle.auction"
    action: ALLOW
}

rule VehicleOwner {
    description: "Allow the owner of a vehicle total access"
    participant(m): "org.acme.vehicle.auction.Member"
    operation: ALL
    resource(v): "org.acme.vehicle.auction.Vehicle"
    condition: (v.owner.getIdentifier() == m.getIdentifier())
    action: ALLOW
}
```

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

Production considerations

- 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

August 2017

Thank You!