

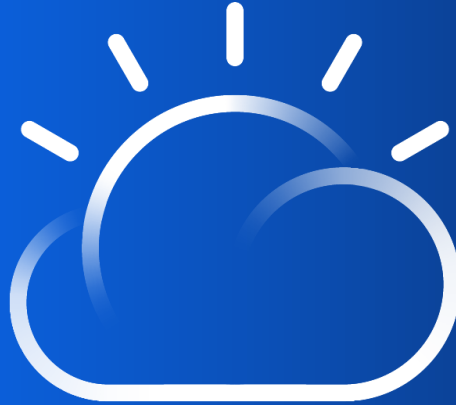
# Develop your First Smart Contract with Hyperledger Composer

— **Carlos Rischio**

Blockchain Technical Leader

**Tito Garrido Ogando**

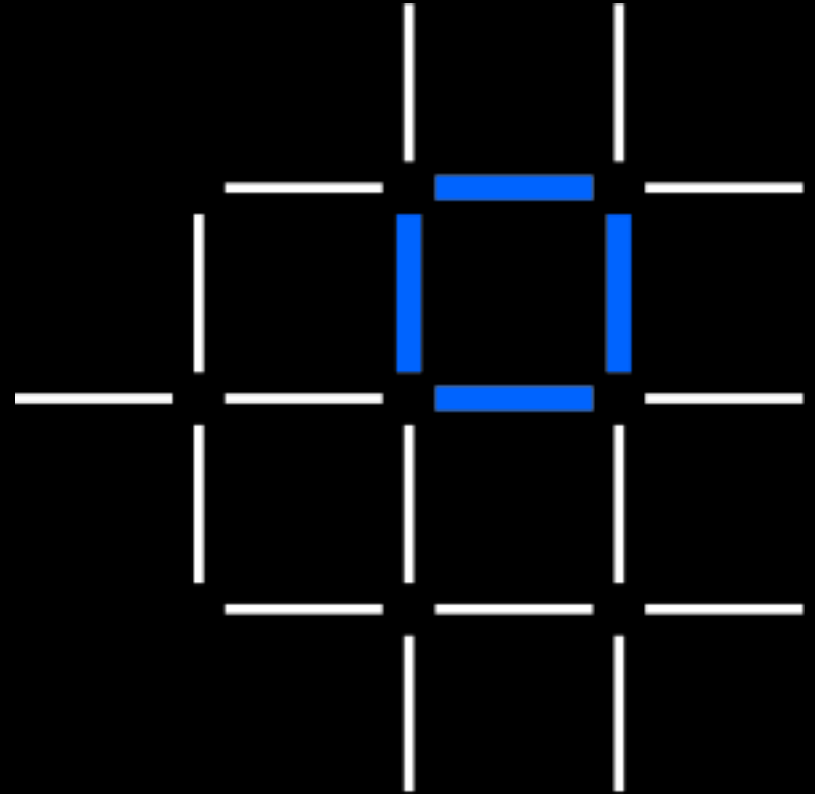
Blockchain Technical Consultant



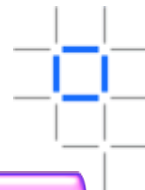
**IBM Cloud**

# Blockchain Composed

A Technical Introduction to Hyperledger Composer

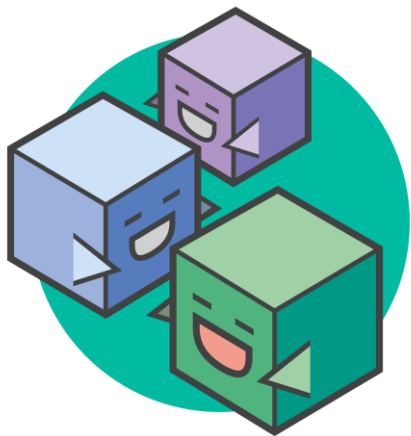


# Hyperledger Composer: Accelerating time to value

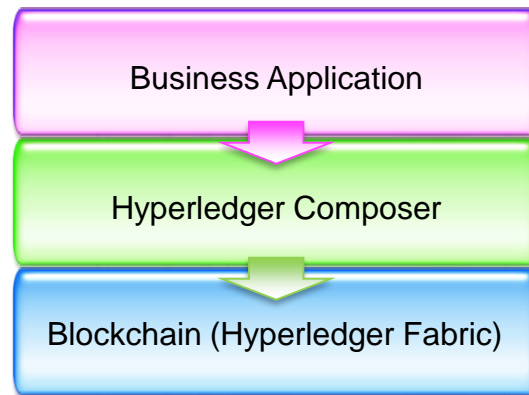


<https://blockchaindevelop.mybluemix.net>

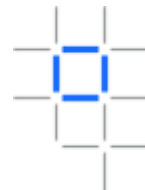
- A suite of high level application **abstractions** for business networks
- Emphasis on business-centric vocabulary for quick solution creation
- Reduce risk, and increase understanding and flexibility



- 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: <https://blockchaindevelop.mybluemix.net>



# Extensive, Familiar, Open Development Toolset

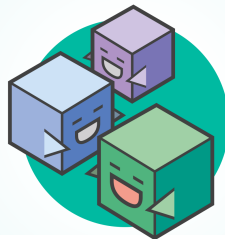


```
asset Animal identi
  o String animal
  o AnimalType sp
  o MovementStatu
  o ProductionTyp
```

Data modelling



JavaScript  
business logic



Web playground

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



Client libraries



Editor support

```
$ composer
```

CLI utilities



Code generation

Powered by



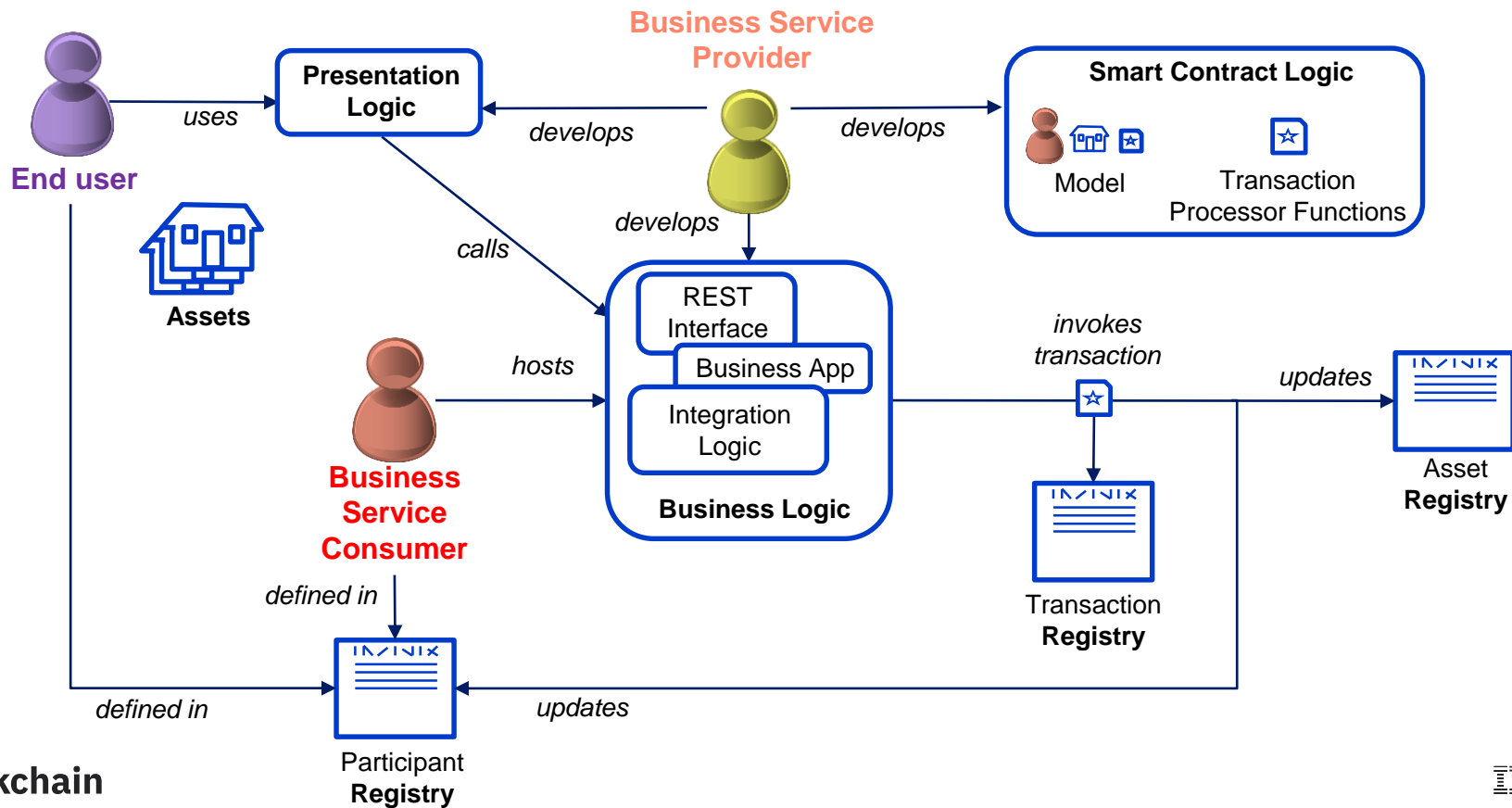
LoopBack  
Node.js Framework



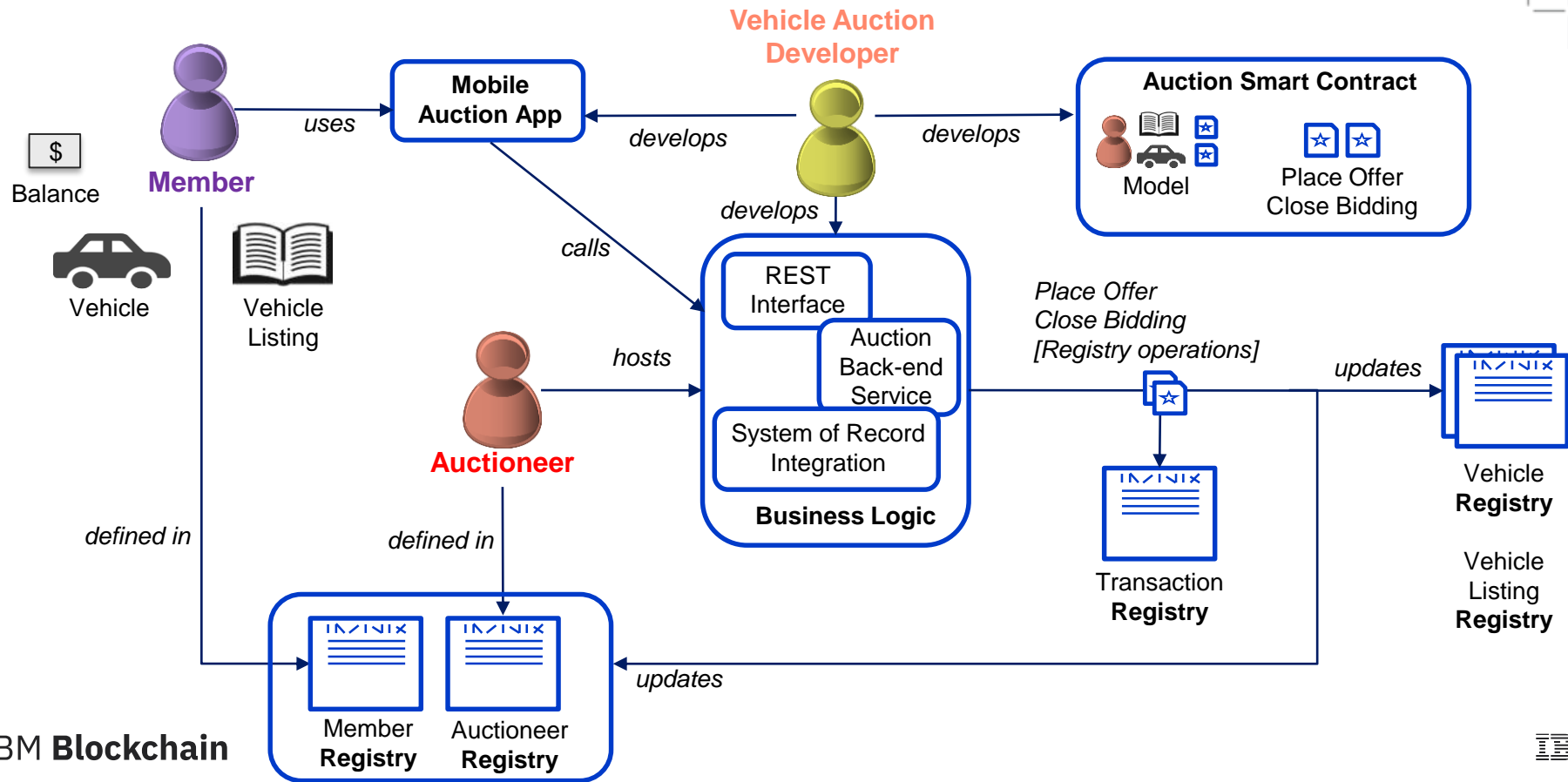
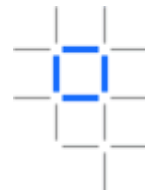
Swagger

Existing systems and  
data

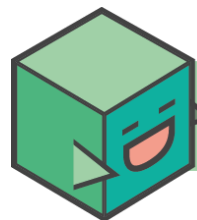
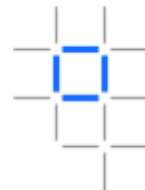
# Key Concepts for the Business Service Provider



# Key Concepts for a Vehicle Auction Developer

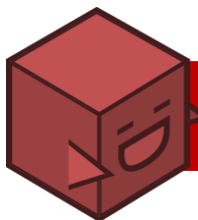


# The Business Service Provider develops three components



## 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



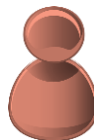
Vehicle



Vehicle Listing



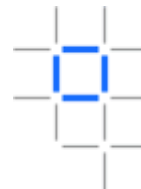
Member



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 {  
}
```

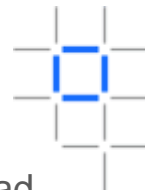
```
transaction Offer {  
  o Double bidPrice  
  --> VehicleListing listing  
  --> Member member  
}  
  
transaction CloseBidding {  
  --> VehicleListing listing  
}
```

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);
```

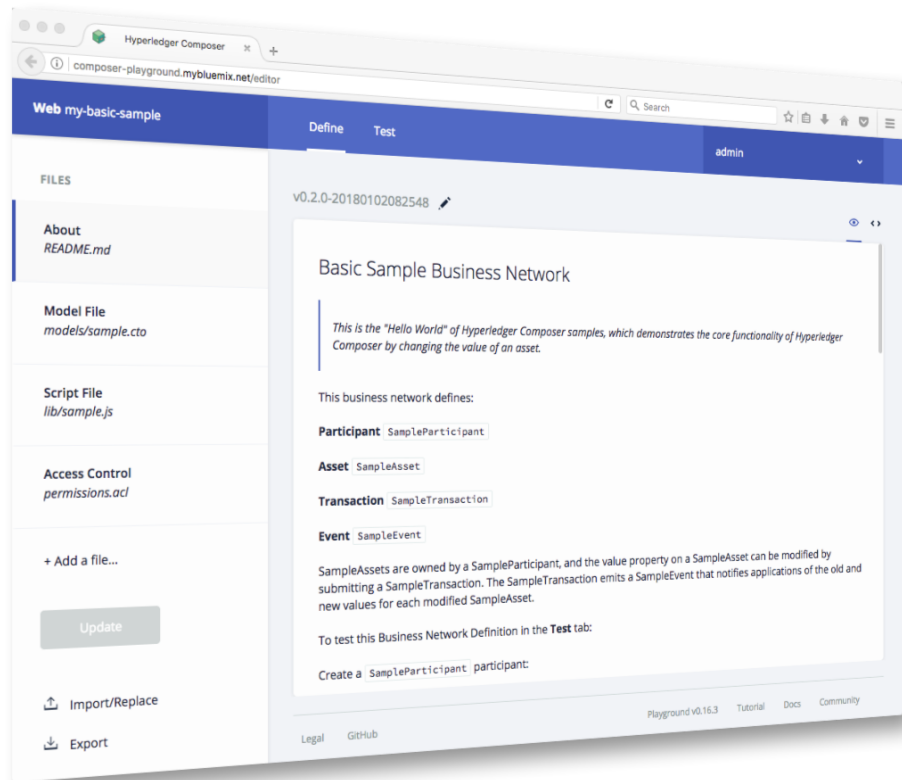
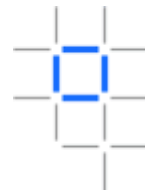
```
businessNetworkConnection.on('SampleEvent', (event) => {  
  console.log(event);  
})
```

- 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)  
}
```

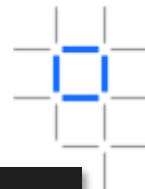
```
return query('selectCommoditiesWithHighQuantity', {})
```

# 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 <https://blockchaindevelop.mybluemix.net>

# General purpose development: Visual Studio Code



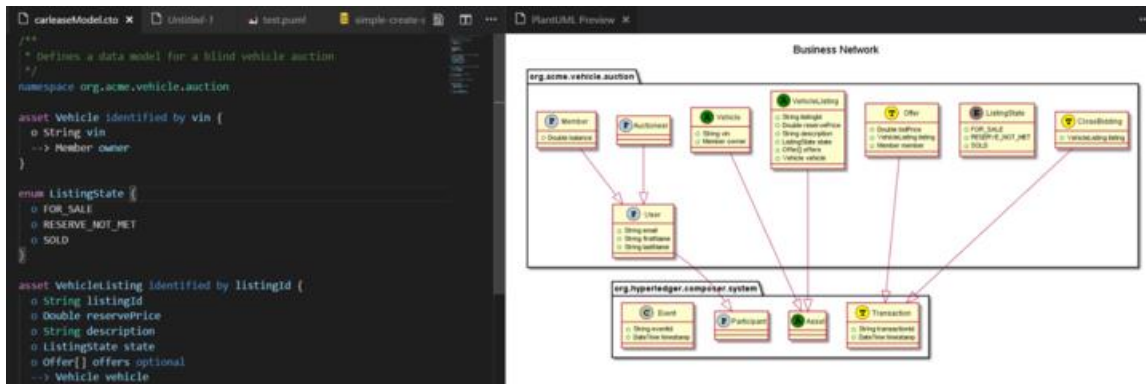
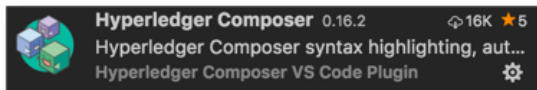
- Composer extension available for this popular tool
- Features to aid rapid Composer development
  - Edit all Composer file types with full syntax highlighting
  - Validation support for models, queries and ACLs
  - Inline error reporting
  - Snippets (press Ctrl+Space for code suggestions)
  - Generate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

```
namespace org.acme.vehicle.lifecycle

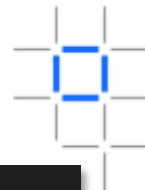
import composer.base.Person
import composer.business.Business

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



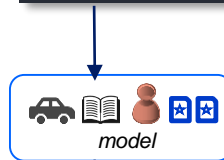
- JavaScript **business applications** *require()* the NPM “composer-client” module
  - This provides the API to access assets, participants and transactions
  - RESTful API (via Loopback) can also be generated... see later
- Command-line tool available to generate **end-user** command-line or Angular2 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
```



VehicleListing

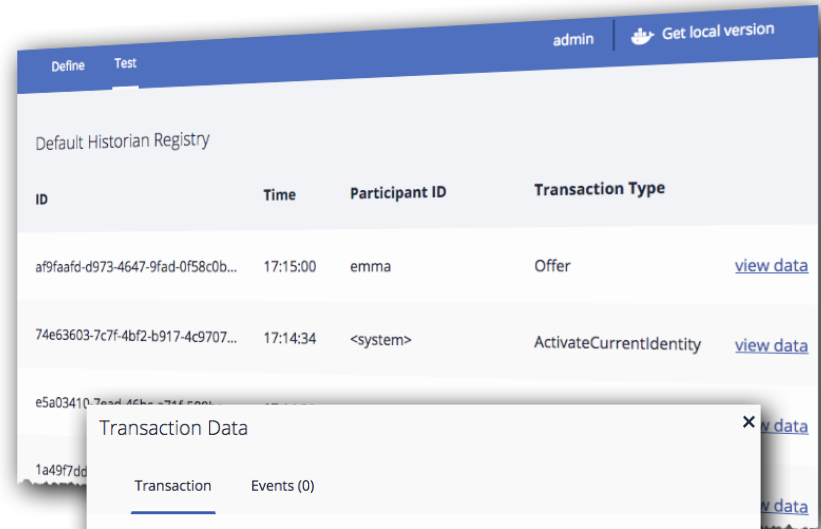
listingId	reservePrice	description	state	offers	vehicle	Actions
LISTING-001	1000	A car listing	FOR_SALE		VIN:67890	<button>Update Asset</button> <button>Delete Asset</button>
LISTING-002	4500	A brand new Ford	FOR_SALE		VIN:12345	<button>Update Asset</button> <button>Delete Asset</button>

File Explorer (MY-APP):

- composer-logs
- e2e
- node\_modules
- src
- angular-cli.json
- editorconfig
- .gitignore
- karma.conf.js
- package.json
- protractor.conf.js
- README.md
- tsconfig.json
- tslint.json

# 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>



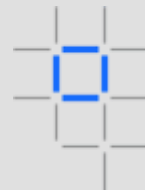
The screenshot shows the 'Playground Historian' interface. At the top, there are tabs for 'Define' and 'Test', and a user profile 'admin' with a 'Get local version' button. Below the tabs, the title 'Default Historian Registry' is displayed. A table lists transactions with columns: ID, Time, Participant ID, and Transaction Type. The first row shows an 'Offer' transaction by 'emma' at 17:15:00. The second row shows an 'ActivateCurrentIdentity' transaction by '<system>' at 17:14:34. Each row has a 'view data' link. A 'Transaction Data' modal is open, showing the details of the first 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
e5a03410-7aad-46ba-74f...			
1a49f7dd...			

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 }
```

# Get started with Hyperledger Composer

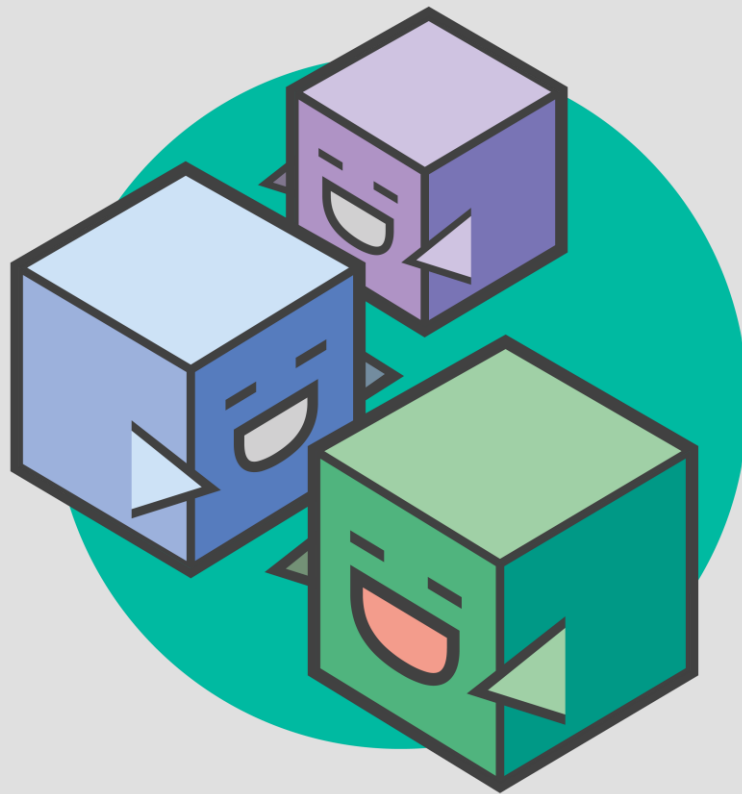


- Define, Test and Deploy Business Networks
- Create domain APIs and sample applications
- Integrate existing systems and data

<https://blockchaindevelop.mybluemix.net>

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

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



# Thank you

## Carlos Rischio

Blockchain Technical Leader

carlosr@br.ibm.com

 rischioto

## Tito Garrido Ogando

Blockchain Technical Consultant

titog@br.ibm.com

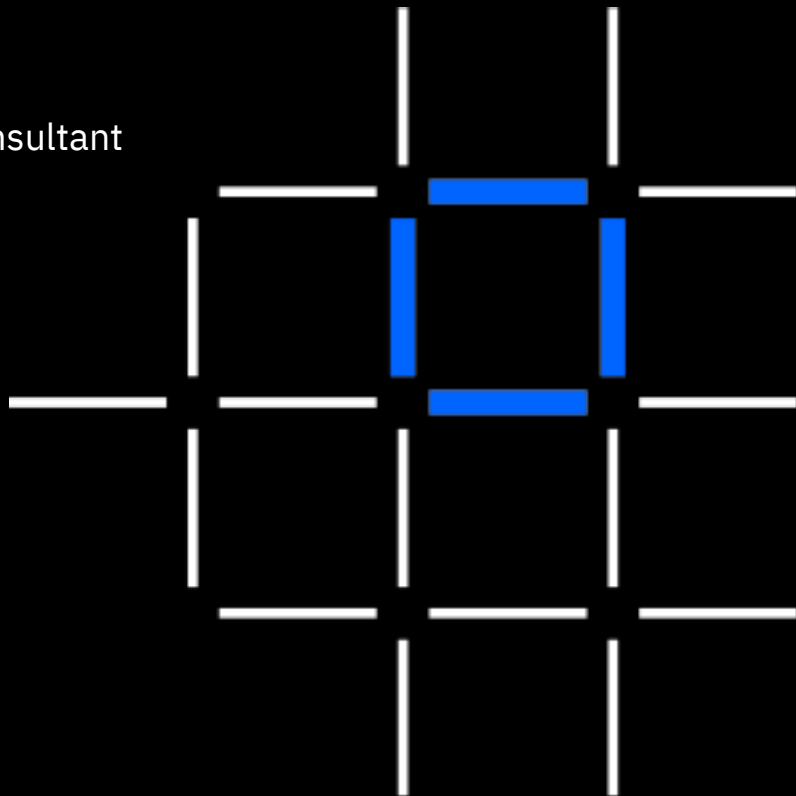
 titogarrido

*Questions? Tweet us or  
go to [ibm.com/blockchain](https://ibm.com/blockchain)*

 @IBMBlockchain

 IBM Blockchain

 IBM Blockchain







© Copyright IBM Corporation 2017. 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.