IBM Blockchain Hands-On
Blockchain Explained

Lab One – Vehicle Lifecycle Lab

1. Introduction to the Vehicle Lifecycle Lab

This lab allows you to experience a blockchain solution from the point of view of a set of end-users, and in doing so learn about key blockchain concepts. It is not meant to be a technical introduction to blockchain, but will instead focus on the properties of the business network and value that blockchain brings.
The use-case we will work through is one that demonstrates the **lifecycle of a new car**, from the manufacturing and purchasing through to delivery and insurance. It is a good blockchain use-case because there is a defined business network and an identifiable need for trust between the participants of the network.
In this lab, you will be playing the role of the four personas who use the vehicle lifecycle system:

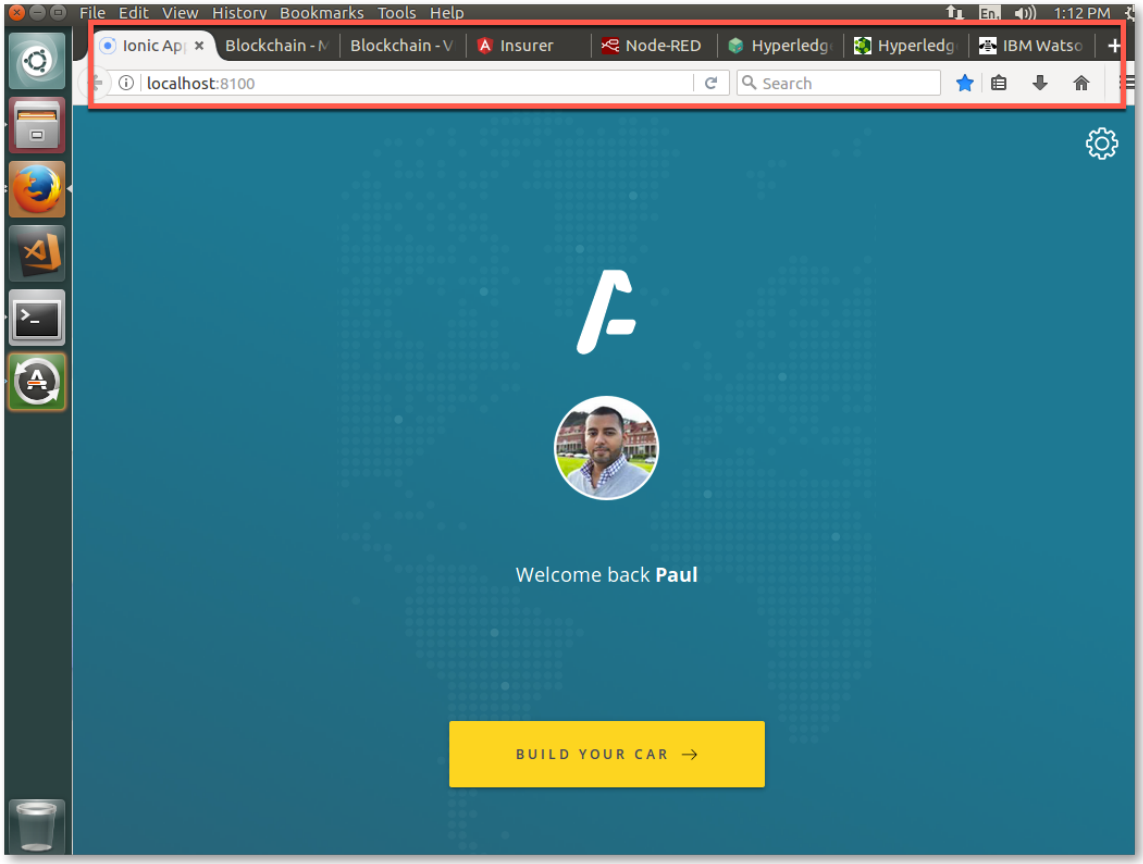
- *Paul* - the buyer/owner of a car
- *Mike* - an employee for the car manufacturer ("Arium")
- *Debbie* - an administrator for the regulator called the Vehicle & Drivers Authority (VDA)
- *Tommen* - an Insurer from an insurance company called Prince

These personas together work on ordering, building, transferring ownership of a vehicle while keeping all the other parties in the network updated and building the trust between them to allow them to work together efficiently.
In this lab, each user's application will be represented by a separate tab in our web browser; of course, in a real blockchain network they each user will be running on different systems in different locations, although all connecting in to the same (but distributed) blockchain network.

Start here. Instructions are always shown on numbered lines.

1. If it is not already running, start the virtual machine for the lab. The instructor will tell you how to do this if you are unsure.

2. Wait for the image to boot, and for the blockchain application and associated services to start. This happens automatically but might take several minutes. The image is ready to use when the web browser is visible and eight tabs fully loaded, as per the screenshot below.



While the virtual machine is starting, let's recap a few blockchain concepts and introduce the scenario.

The most generally accepted definition of a blockchain is of a ***shared, replicated ledger***.

Ledgers have been around for hundreds of years and are records of what a business has done. They're important systems of record because they describe a business's inputs and outputs and thereby give an indication of its worth. Essentially, they are a log of transactions – a transaction being a change in state of an asset.

The problem with ledgers is that each one is owned by a single business, which means that when one business transacts with another, ledgers can get out of sync. What happens when the transaction I've recorded on my ledger doesn't tally with the transaction you've recorded on your ledger? Disputes inevitably occur which need to be resolved through a reconciliation process. This can be slow and expensive.

By having a *shared* ledger it means that all participants of the business network see the same ledger. By *replicating* it across the business network, it means that the ledger is not held in any one single place, which would otherwise make it vulnerable to outages and malicious attack.

Consider the business network that surrounds the purchase and ownership of a car. Today, each participant (for example, manufacturer, insurer or regulator) has their own ledger and the processes that allow them to interact with each other varies from company to company, and can be time consuming to complete. Connectivity between participants is typically done point-to-point using a variety of processes – some manual or slow (e.g. sending a letter), and some automated (e.g. file, REST API, B2B messaging). This plethora of processes is expensive to maintain and can be vulnerable to attack.

In this scenario we will replace these disparate ledgers with a single blockchain, and the individual business processes with a single shared one. By doing this we will make the overall process much quicker and less prone to error.

We will experience the solution through the eyes of four key members of the business network: a private purchaser, the manufacturer, the regulator and the insurer.

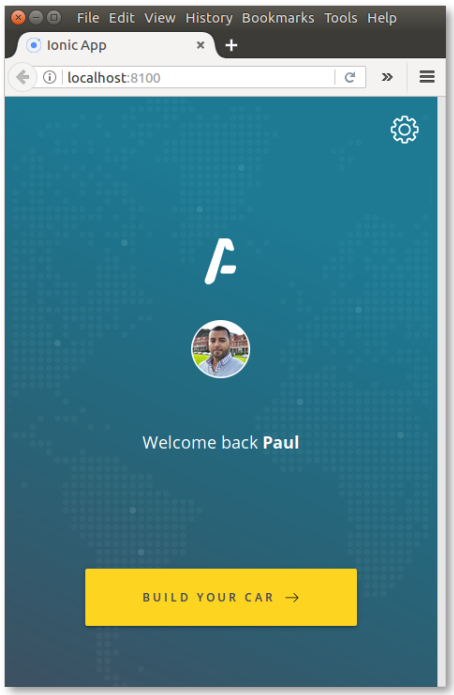
We will start by looking at the ordering process, as experienced by the buyer.

2. Running the lab

2.1. Ordering the car

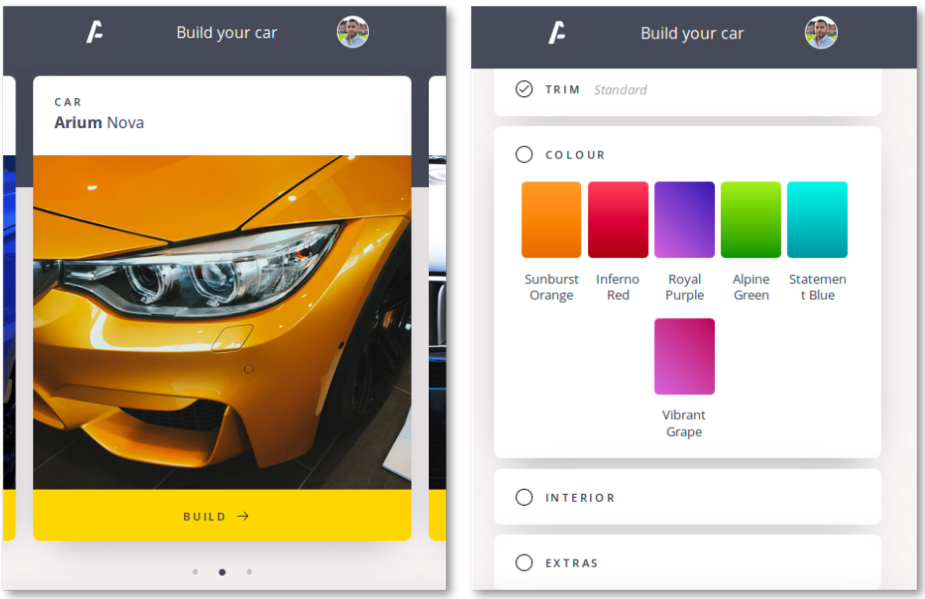
3. If it is not already selected, Switch to the “Ionic App” tab on the web browser (running at localhost:8100).

   Note that Paul’s web page is intended to be viewed as a mobile app; you might want to ungroup this tab from the others by dragging the tab’s title bar off from the others, and resizing the window to make it easier to view and navigate as a mobile app.

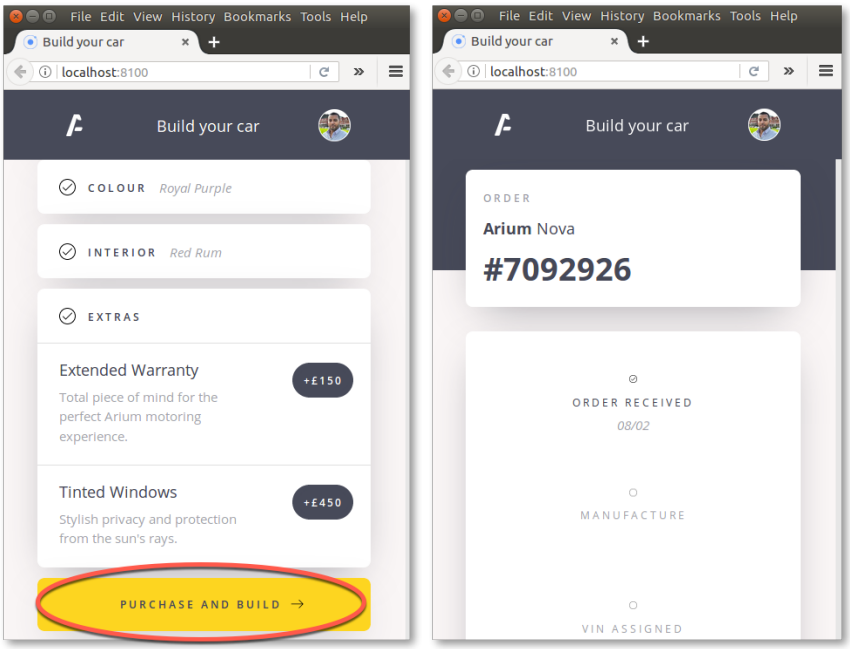


4. In Paul’s app, click **‘Build Your Car’**.

5. Swipe left and right to decide which car to build, and then decide the options on Paul's car.



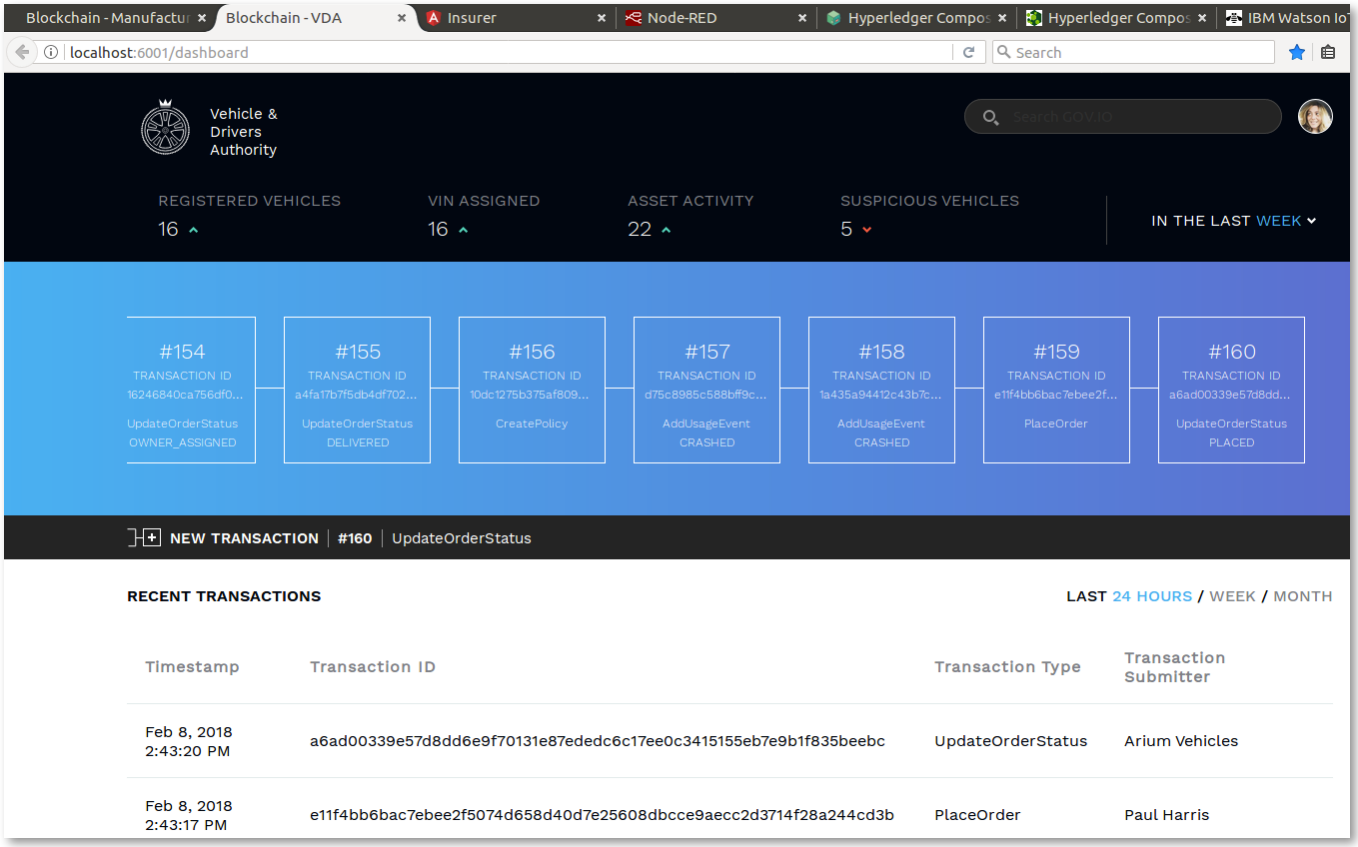
6. Once you have decided on each of Paul's options, click 'Purchase and Build'.



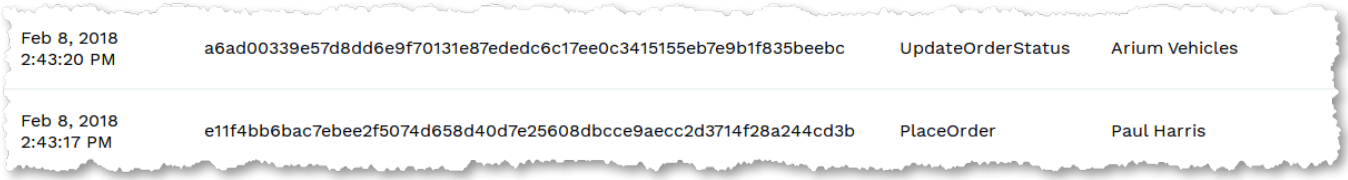
7. Once you place the order, switch to the "Blockchain – VDA" tab (localhost:6001/dashboard).

   As you will recall, the VDA is the regulator who requires notification of all transactions that occur within the business network. Debbie, who works for the regulator, has a dashboard running on her PC that shows all transactions as they occur.

   You will immediately see the VDA dashboard update itself with the latest transaction.



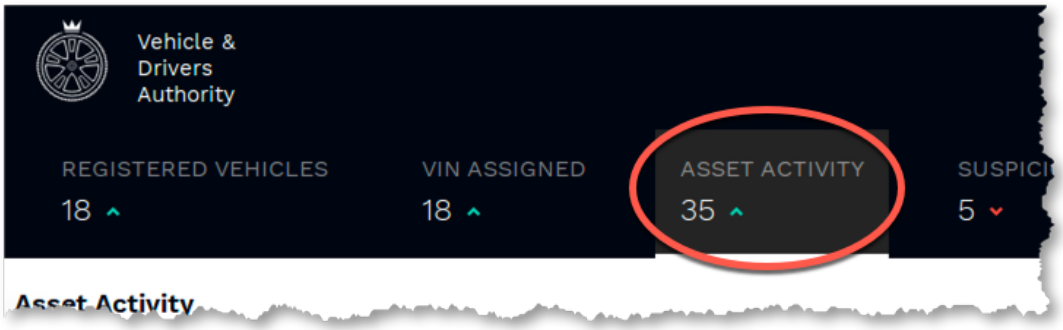
If you look at the "Recent Transactions" log at the bottom of the page, you will see two new transactions listed: a "PlaceOrder" transaction submitted by Paul Harris, and an "UpdateOrderStatus" acknowledgement from Arium Vehicles, our manufacturer.



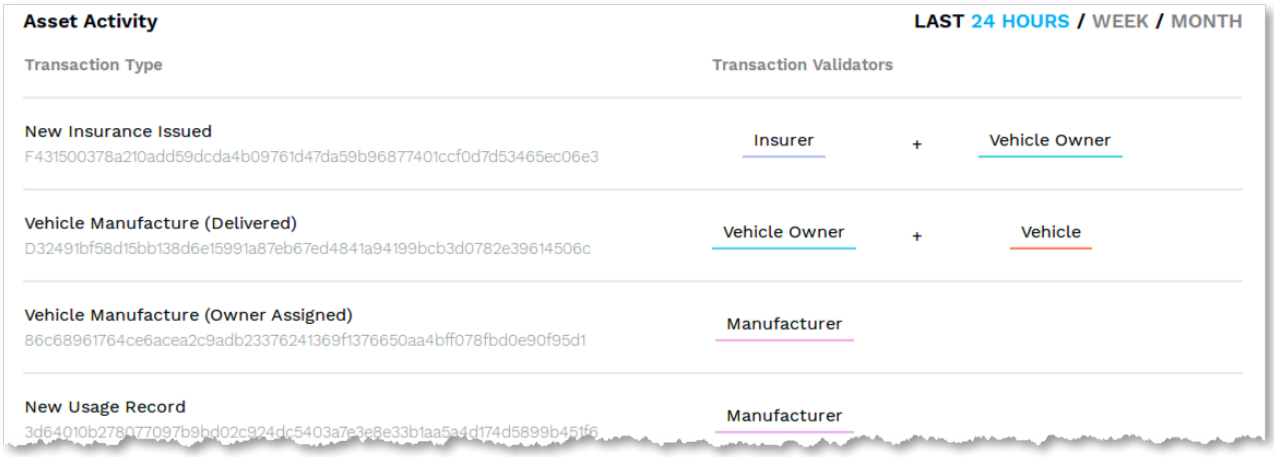
Look in the blue section above this log and you will see those transactions represented graphically as a chain, with the most recent transactions on the right. This is a representation of our blockchain, and the regulator can see everything that is stored on it.

As you will recall, the blockchain is our transaction log which is shared (with appropriate privacy and permissioning) between the participants of our business network. Each block in this chain could potentially actually contain multiple transactions, but here you'll just see each unique transaction inside its own block.

8. Click on 'Asset activity' within the VDA dashboard.



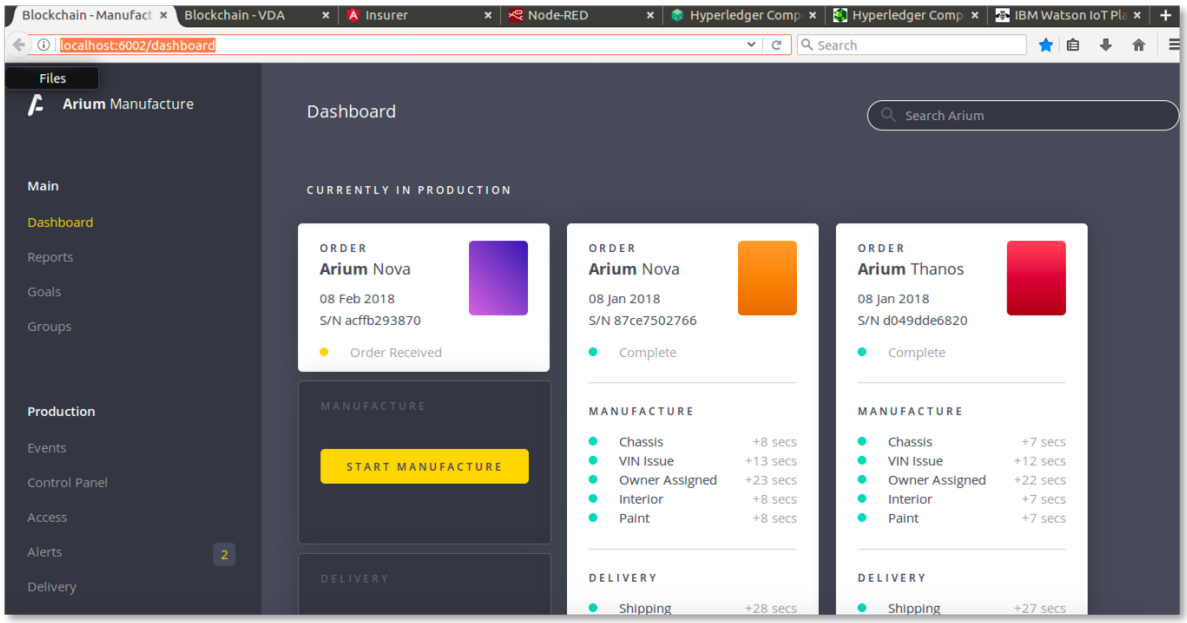
This is an alternative view of the ledger that shows all the transactions that have occurred, and the participants involved.



2.2. Manufacturing the car

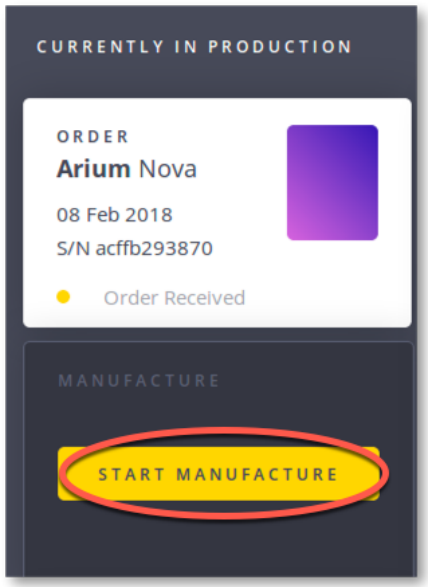
9. Switch to the "Blockchain – Manufacturer" tab (localhost:6002/dashboard).

This is the dashboard that Mike, who works for Arium, uses. He does not have full visibility into the entire blockchain that the regulator requires, but can see the parts of it that pertain to Arium: specifically, he has visibility into all the orders that are coming in so that he can control the manufacturing process.



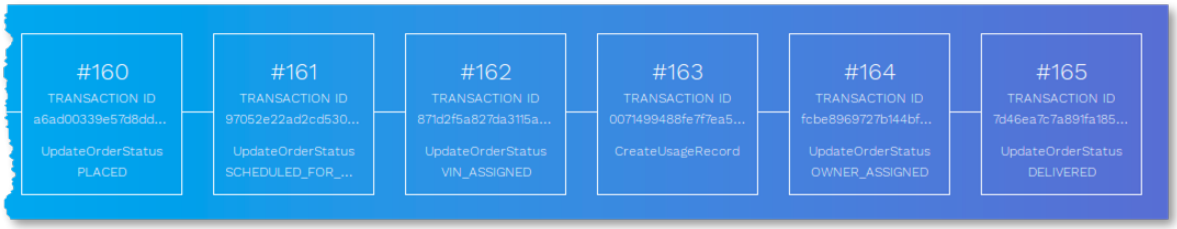
The “Currently in Production” section of this page shows those orders that have been received and the cars that have recently been built. The left-most order in this section will be the car that Paul recently ordered.

10. Click Start Manufacture underneath Paul’s order to start the business process to build a car.

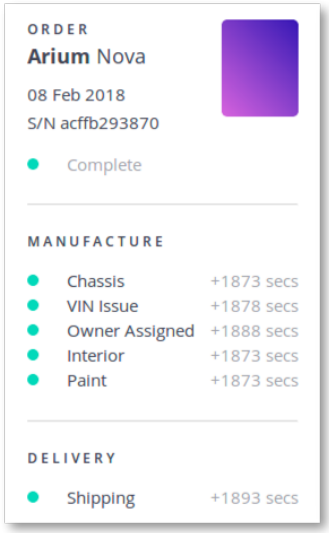


The production process has (of course) been simulated and will take place over the next several seconds; the vehicle will be ‘built’ and blockchain transactions submitted that record status at key milestones of the production process. In a real network, different automated plant systems will trigger these events, which are signed off by the manufacturer, and the issuance of the Vehicle Identification Number might be signed off by the regulator.

11. As the car is being built, switch back to the VDA dashboard to see these key milestones being represented on the unfiltered blockchain.



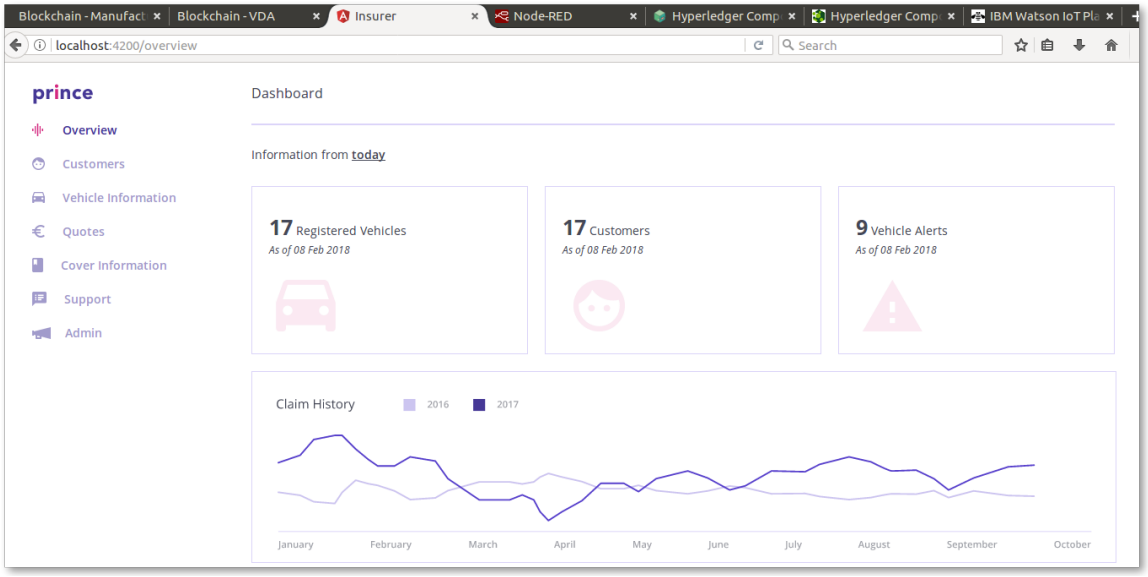
12. Also note how the Manufacturer's view changes as the vehicle is being built, with icons changing to green as those parts of the process are completed.



2.3. Insuring the car

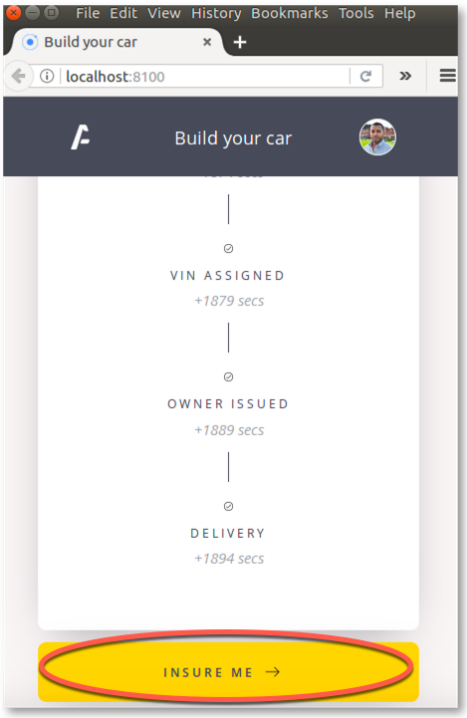
As Paul takes ownership of his new car, we will give him the option to insure it. His insurance company offers a discount if he chooses to provide the insurance company with frequent details of the car's location and other things.
The manufacturer fits the car with a collection of IoT devices, including GPS, air and engine temperature sensors, acceleration information and light information, which can give the insurer information on how the car is being driven, and potentially alert relevant parties if the car is involved in a crash.

13. Switch to the Insurer dashboard (localhost:4200/overview). Ensure that the 'Overview' tab is selected.

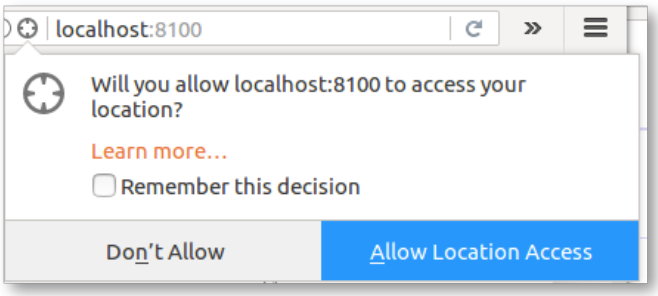


Tommen works for Prince Insurance and this is his dashboard. He requires another subset of information from the blockchain and this view is represented here. He can see information on the cars for which his company is an insurer, and can also approve new polices. (In reality, this latter part can be automated.)

14. Switch back to our car buyer's "Ionic App". After the car has been delivered scroll down to the bottom, and click the **"Insure Me"** button.

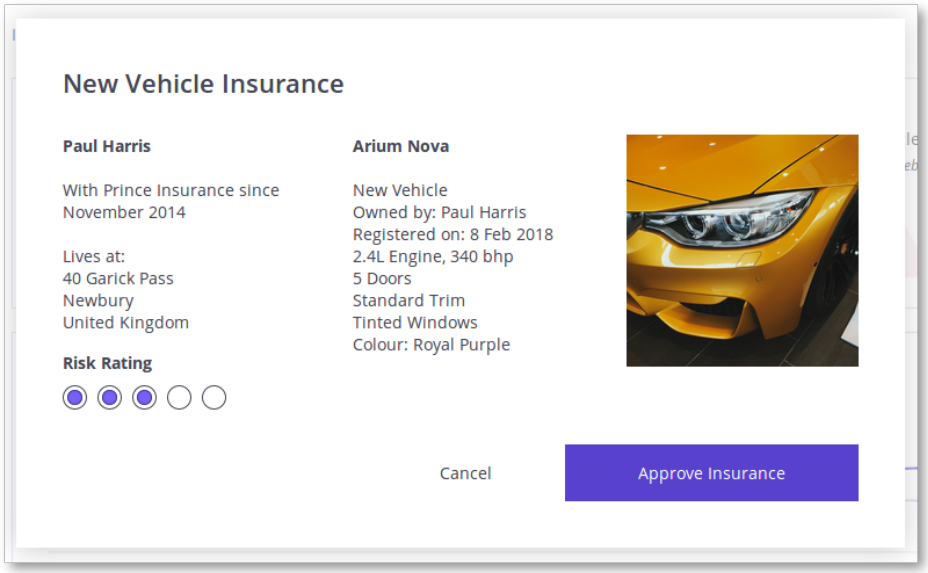


15. Click "**Allow Location Access"** if a popup appears; Paul is willing to share the IoT device's location with the insurance company.



16. Switch back to the "Insurer" view (localhost:4200/overview).

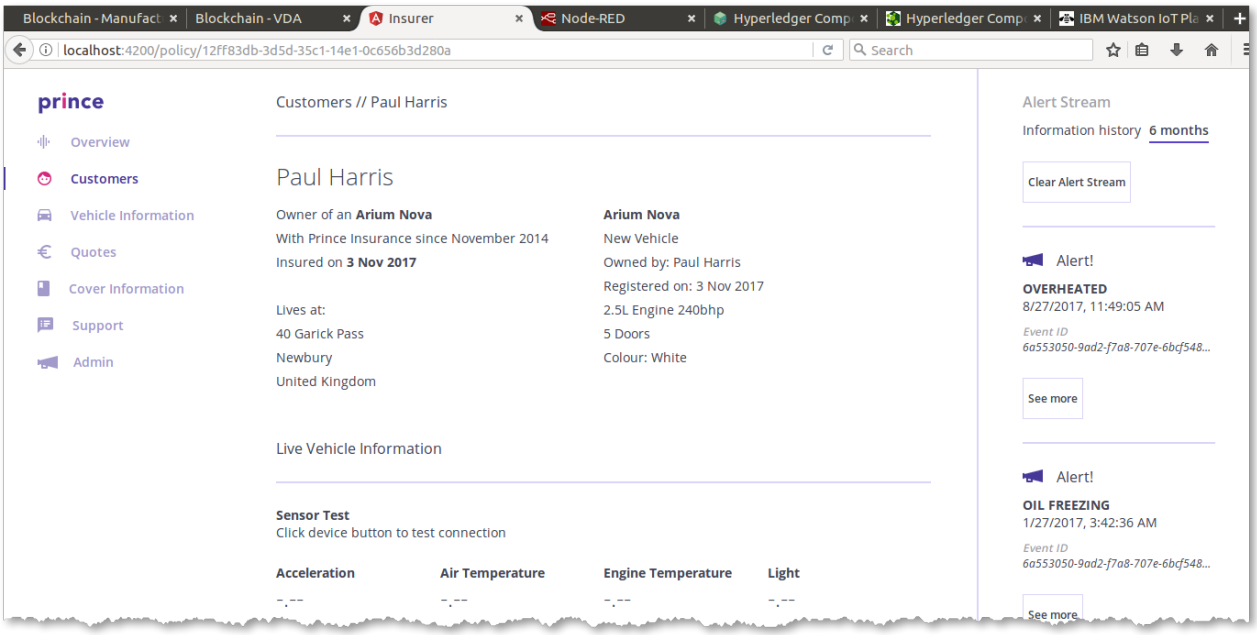
17. Click the "Approve Insurance" button.



18. Wait for the approval to be logged on the blockchain.

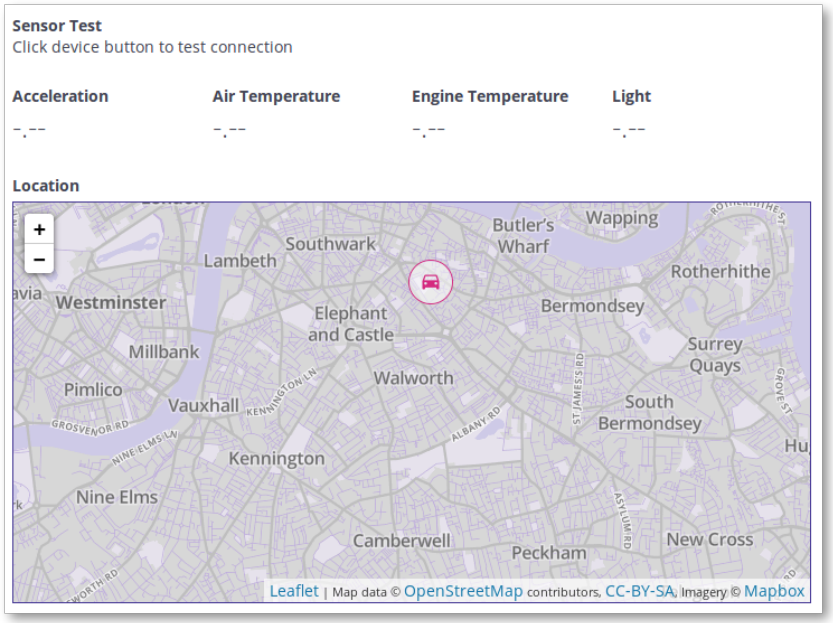
    Paul is now insured by the insurance company.

19. Review the 'Customers' tab to see details of Paul's policy.



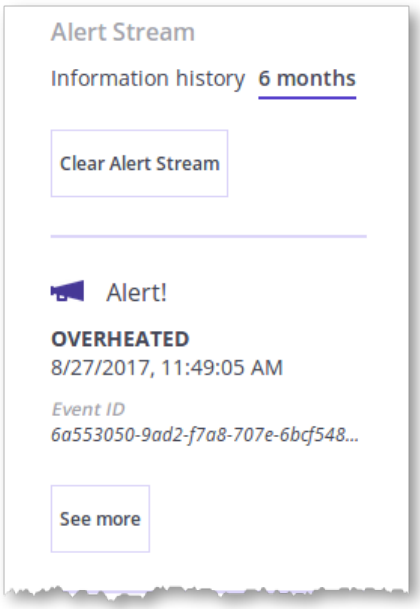
At the top of the page you can see basic details of Paul's policy including his address and car information.

Underneath this is the set of raw readings from the IoT devices attached to Paul's car. This is useful information for debugging; in reality the blockchain is not used to share complete data streams from the IoT sensors as the amount of data is too great and is not relevant to be shared in its entirety.



However, what would be relevant is the analysis of key events in the IoT stream. For example, if the acceleration is shown to be greater than (for example) 2G, this might indicate a crash event that the insurer might care about.

This is shown as a set of alerts on the right hand side of the insurer's customer view:



Without a real sensor tag connected into the application, the information displayed here is either blank or mocked up. In the next section we will inject data into the application using internet of things integration.

It is possible to connect a real sensor tag so that its information is displayed in the Insurer view; we have tested using a Texas Instruments SimpleLink Bluetooth SensorTag. To use this, you need to download the TI SimpleLink Starter app to a nearby mobile device, use it to discover the sensor via Bluetooth, note down the unique address of the tag and enable the “Push to cloud” option to submit the sensor readings so that they can be read by the IBM Watson IoT platform. Then you need to update the “IBM IoTP Test Device” node in the Node-RED flow to monitor the readings from the unique address of the tag from the cloud. Remember to redeploy the Node.RED flow.

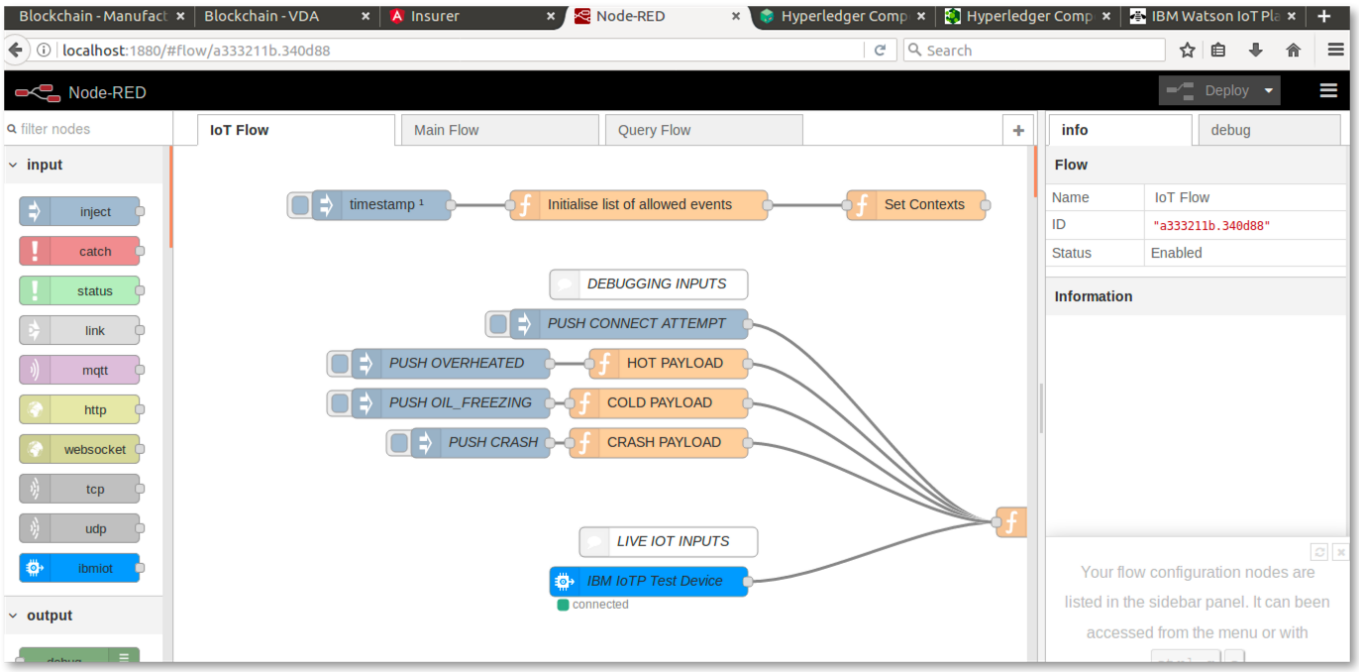
3. Cool Stuff

In this section, we're going to look at how the scenario can be enhanced by bringing blockchain together with internet of things and analytics.

3.1. Internet of Things Integration

We will start by looking at how sensor data from the car makes its way into the blockchain. To do this we will use an integration tool called Node-RED. This includes a graphical interface to describe how data flows from input sources (e.g. a sensor) to output sources (e.g. the blockchain).
Node-RED has connectors for sending data to, and receiving data from, a blockchain running Hyperledger Composer. It also has connectors for receiving data from the IBM Watson IoT platform (for sensor tag integration). We can also generate fake sensor data for testing, in the absence of a physical sensor device.

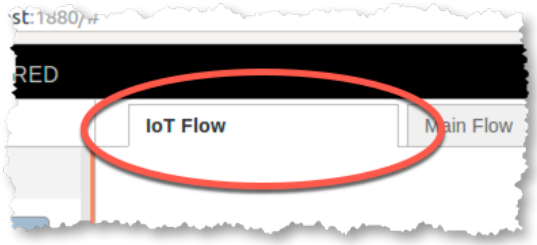
20. Switch to the Node-RED tab (localhost:1880).



The main window shows the flow of how data from devices is mapped to blockchain events. The tabs along the top show the different flows that are deployed. Down the left hand side you can see the available connectors for wiring into the flow. The right hand-side contains the properties of the selected connector and debug information.

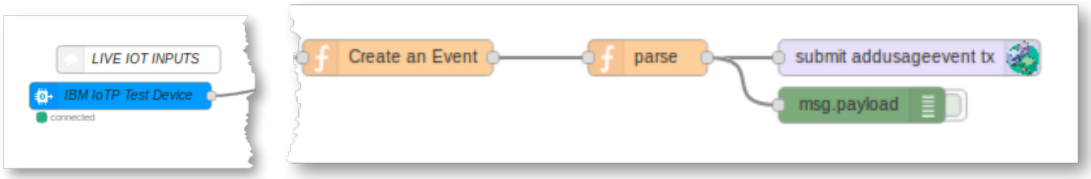
(Note that if you make any changes to the flow, you need to press the "Deploy" button to let them take effect.)

21. Ensure that the "IoT Flow" tab in Node-RED is selected.



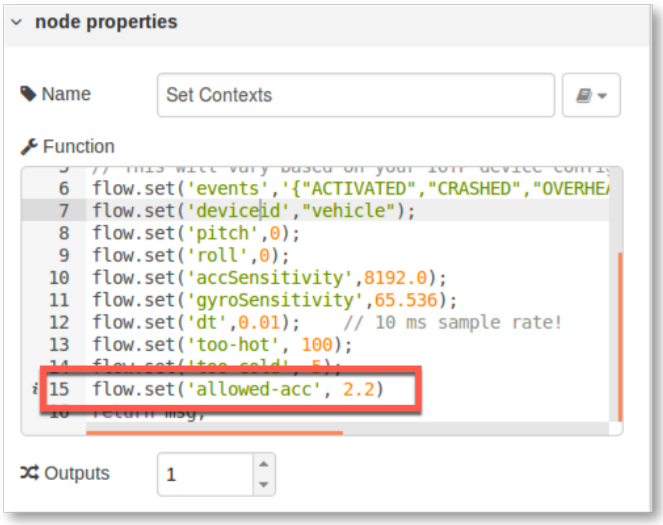
There are some interesting things to note in this flow.

22. Look at the “IBM IoTP Test Device” to “Submit AddUsageEvent TX” flow. This takes readings from a real sensor device and publishes any interesting events to the blockchain.

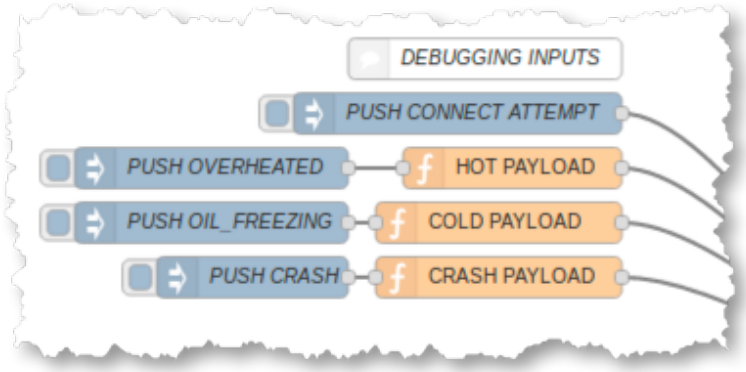


23. Double click the “Set Contexts” connector.

    This shows the thresholds for sending interesting events to the blockchain. For example, if the acceleration is greater than 2.2G, this causes a crash event to be sent to the blockchain.

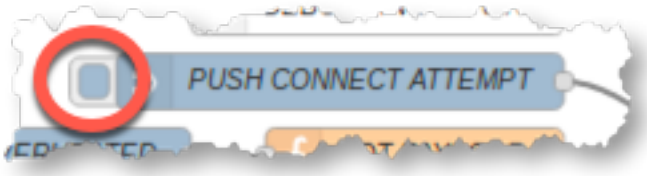


24. Look at the set of connectors next to the “DEBUGGING INPUTS” section: PUSH CONNECT ATTEMPT, PUSH OVERHEATED, PUSH OIL FREEZING and PUSH CRASH.



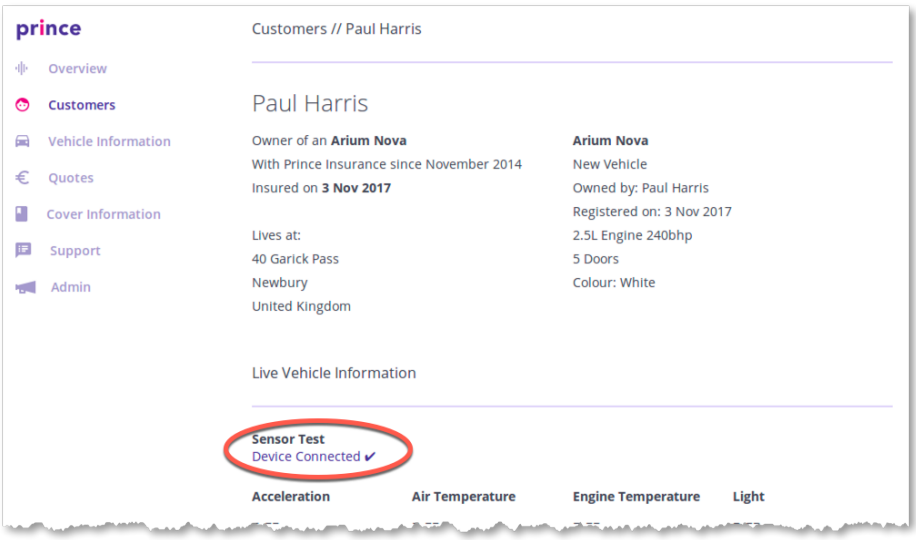
These connectors allow us to simulate an interesting event occurring, in the absence of a real device.

25. Click the rounded square button next to the PUSH CONNECT ATTEMPT connector.



You should see a message saying that data was successfully injected into the flow.

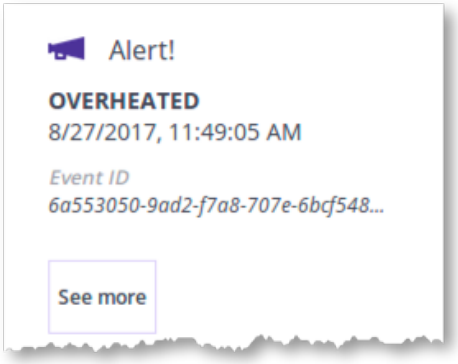
26. Switch to the Insurer tab (localhost:4200), and notice under “Sensor Test” that the vehicle sensor is now connected.



27. Switch back to the Node-RED tab (localhost:1800), and click the button next to the PUSH OVERHEATED node to send an event to the blockchain which denotes Paul's engine overheating.

You should again see a "Successfully injected" message.

28. In the Insurer view you should see an alert that reveals this event to the insurer.



29. Try invoking the other events too (OIL FREEZING, CRASH) to see their effect.

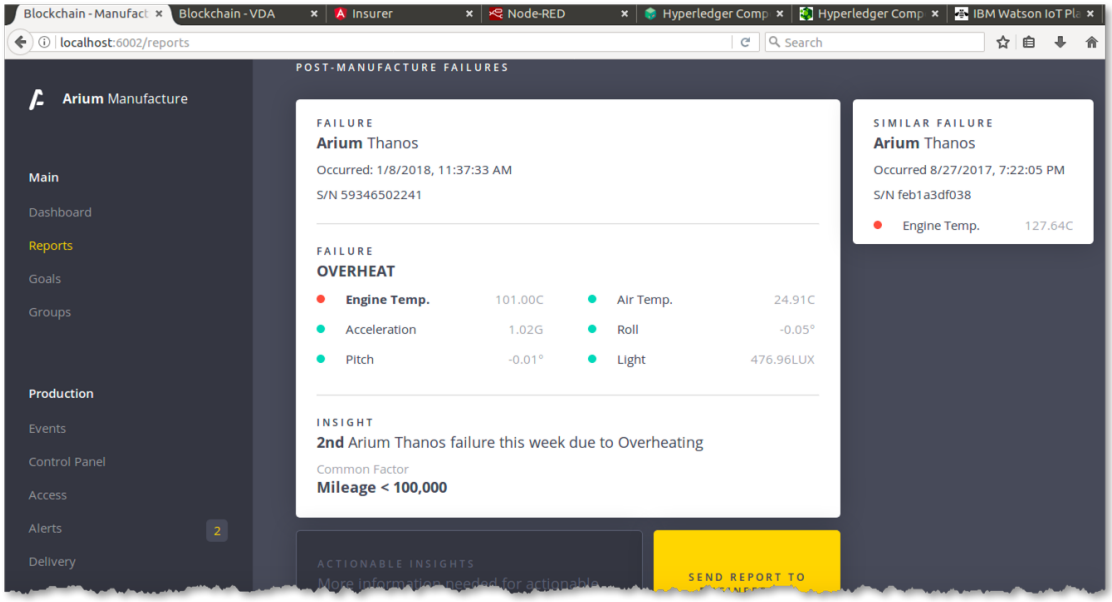
More details on the IBM Watson IoT Platform can be found on a pre-loaded tab in the web browser (https://i5l9uv.internetofthings.ibmcloud.com/dashboard/#/ibmssolanding).

3.2. Analytics

It is possible to use the information stored on the blockchain to provide insight on aggregate usage patterns to interested authorized parties. This gives the power of data analytics on top of the benefits of a blockchain, as a verifiably clean source of information to analyze.

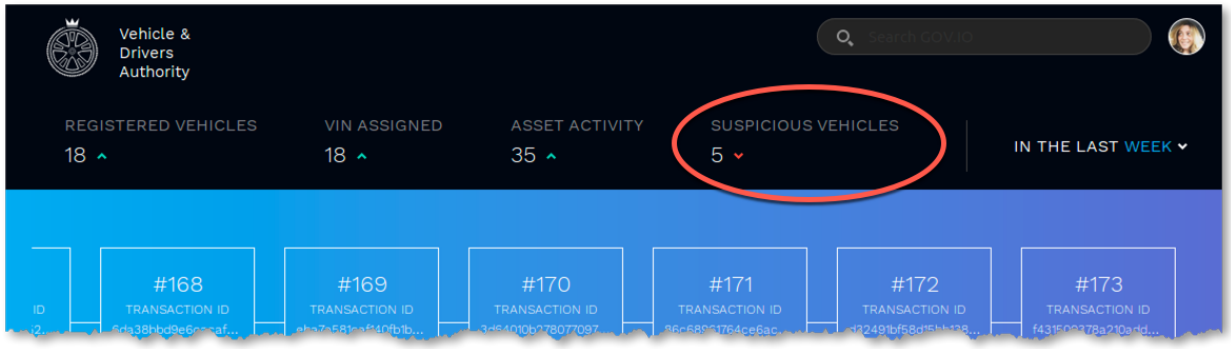
30. Switch back to the Manufacturer view tab (localhost:6002) and click the "Reports" link.

The engine overheated events show in this view. These events were captured in the blockchain and the manufacturer role has permission see this type of event. The manufacturer wishes to detect trends in engine overheated failures in order to determine if a factory defect is causing this condition.



The regulator in this scenario can also run analytics on the transactions on the blockchain to look for suspicious behavior that the smart contract was not designed to prevent from a single invocation.

31. In the Vehicle & Driver Authority dashboard (localhost:6001/dashboard) click the "Suspicious Vehicles" tab near the top.



Here we can see that by performing analytics on the blockchain dataset, we have found a number of vehicles with associated suspicious transactions that may warrant further investigation.

32. Click on 'Mileage anomaly' in the list of suspicious vehicles.

SUSPICIOUS VEHICLES LAST 24 HOU

Vehicle	VIN	Notification
Ridge Cannon - White	312457645	Suspicious ownership sequence
Ridge Rancher - White	326548754	Uninsured vehicle
Morde Pluto - Green	564215468	Insurance write-off but still active
Morde Putt - Black	6437956437	Mileage anomaly
Ridge Cannon - Silver	65235647	Untaxed vehicle

This shows a list of the transactions that are associated with this anomaly. In this instance, the mileage of the vehicle may not match with insurance records - or has even has decreased from previous records.

Morde Putt - Black 6437956437 Mileage anomaly

Timestamp	Transaction	Car Owner
Nov 3, 2017 6:27:51 PM	b6c71558-0873-335c-635e-adcf32115b17	Anastasia

Contact Current Owner ×

4. Under the Hood

In the final section of the lab, we will briefly consider how the scenario was put together. If you wish to find out more about the tools used to create this application, consider completing a follow-on lab; ask the instructor for details.

4.1. Modelling the Scenario

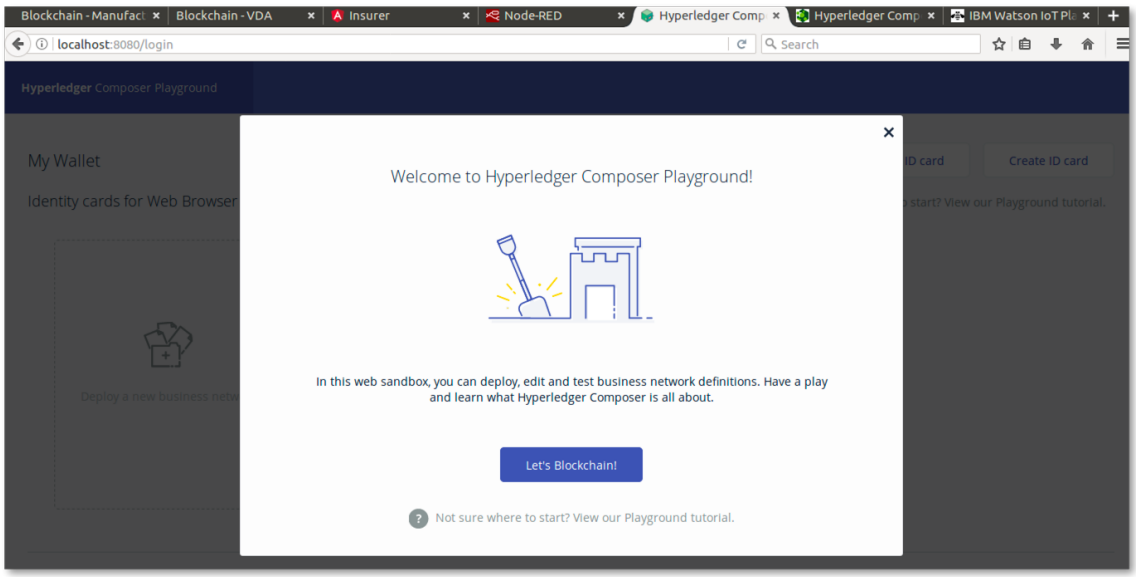
All blockchain use-cases can be described in terms of a set of *assets* (the digital representation of some tangible or intangible object that holds value), *participants* (who wish share information with other participants in a trustworthy way) and *transactions* (which cause the assets to change state).

In our example, the primary asset is the car (obviously), the participants are the owner, manufacturer, regulator and insurer, and as we've seen, there are several transaction types as the car moves through its lifecycle.

These assets, participants and transactions can be modelled in a Linux Foundation tool called Hyperledger Composer, and leveraged through the IBM Blockchain Platform.

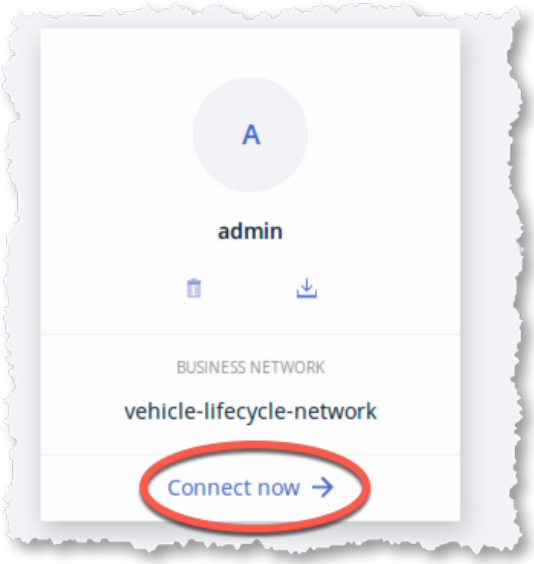
It is useful to develop a model of these concepts as it provides a neat abstraction layer between the business problem being solved and the technical complexities of the underlying blockchain – in much the same way as a compiler shields the programmer from the details of the underlying machine code.

33. Switch to the Hyperledger Composer Playground tab in the web browser (localhost:8001/login).

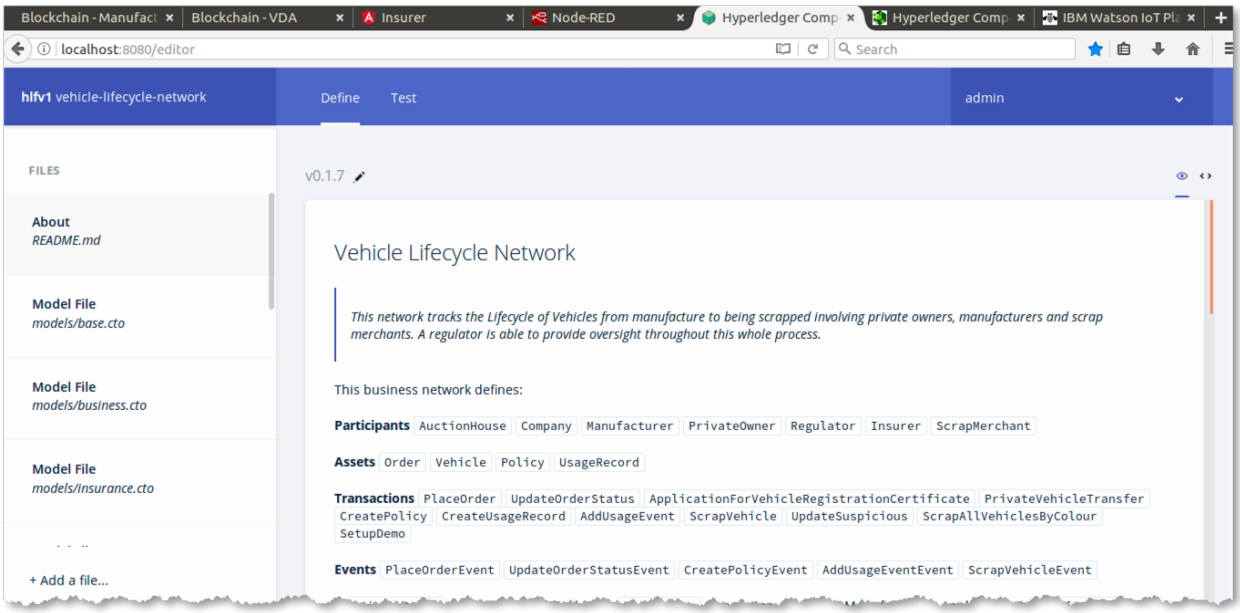


34. Dismiss the welcome dialog by clicking "Let's Blockchain!".

35. Scroll to the bottom of the "My Wallet" screen to see details of our deployed blockchain network (vehicle-lifecycle-network. Click 'Connect now'.



Once the Playground has connected to the blockchain, you will see details of the vehicle lifecycle network.



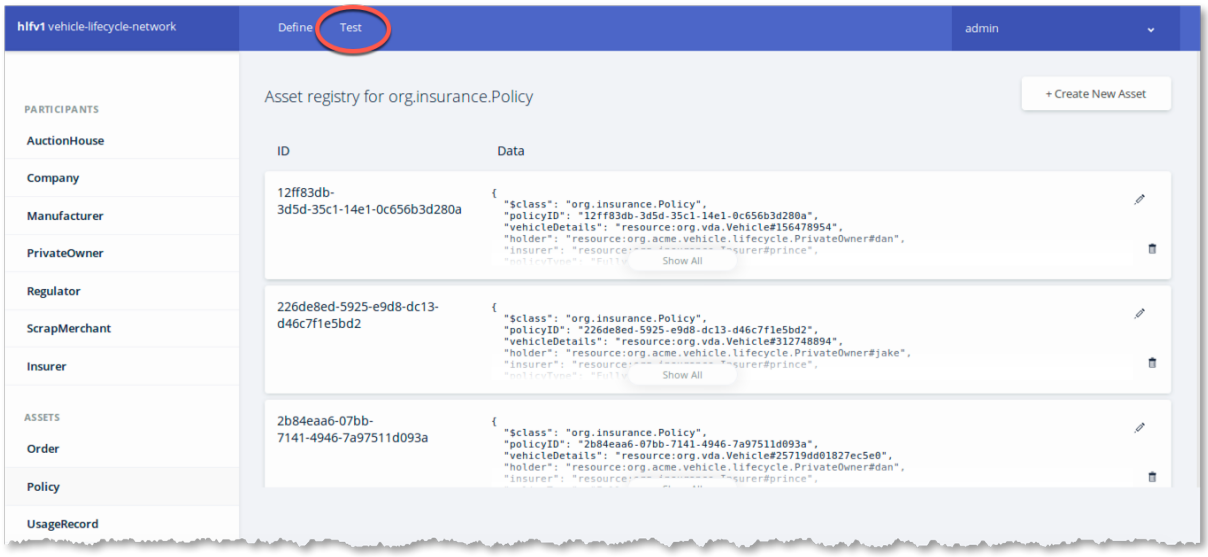
Along the top of the screen are two tabs: “Define” which shows the files used to model the network, and “Test” that allows authorized users (an administrator “admin” - by default) to invoke transactions.

36. With the Define tab selected, click the filenames down the left hand side of the screen to view the contents of the files that comprise the model, transaction logic, access control lists and documentation.

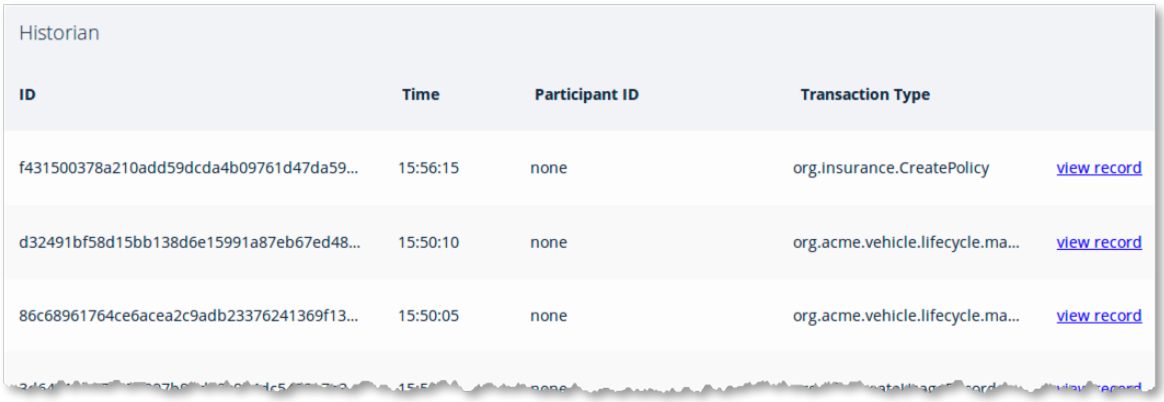
```
asset Vehicle identified by vin {
  o String vin
  o VehicleDetails vehicleDetails
  o VehicleStatus vehicleStatus
  --> Person owner optional
  o String numberPlate optional
  o String suspiciousMessage optional
  o VehicleTransferLogEntry[] logEntries optional
}
```

We will go into details of what these files do in a follow-on lab.

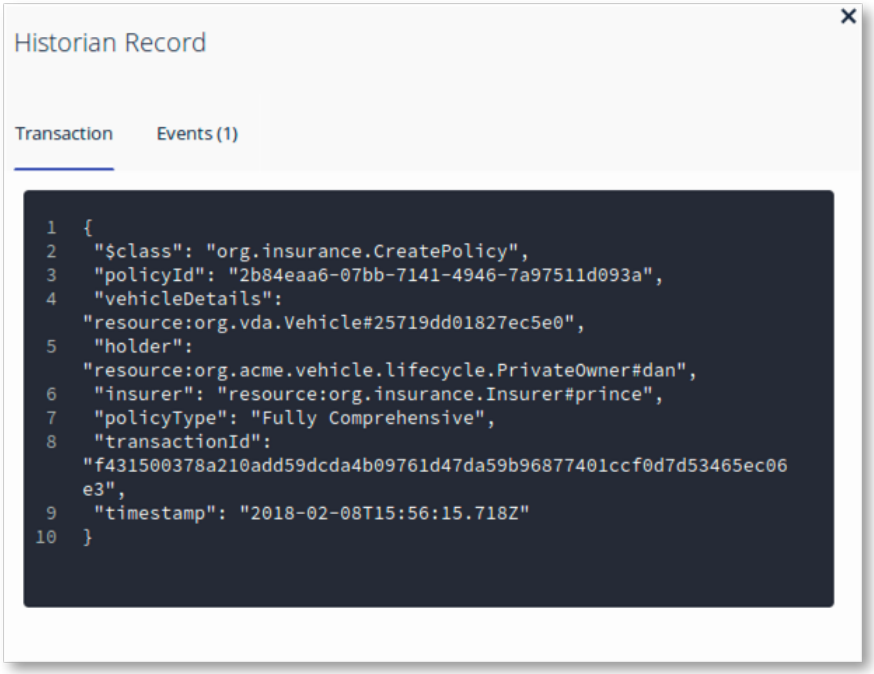
37. With the Test tab selected, click the registries down the left hand side of the screen to view the instances of the assets, participants and transactions that have been created, and their current state.



38. Click 'All Transactions' to view the Transaction Historian. This shows you every transaction that has been recorded on the blockchain that the current user ('admin') has authority to see.



39. Click on any transaction to view details of it.



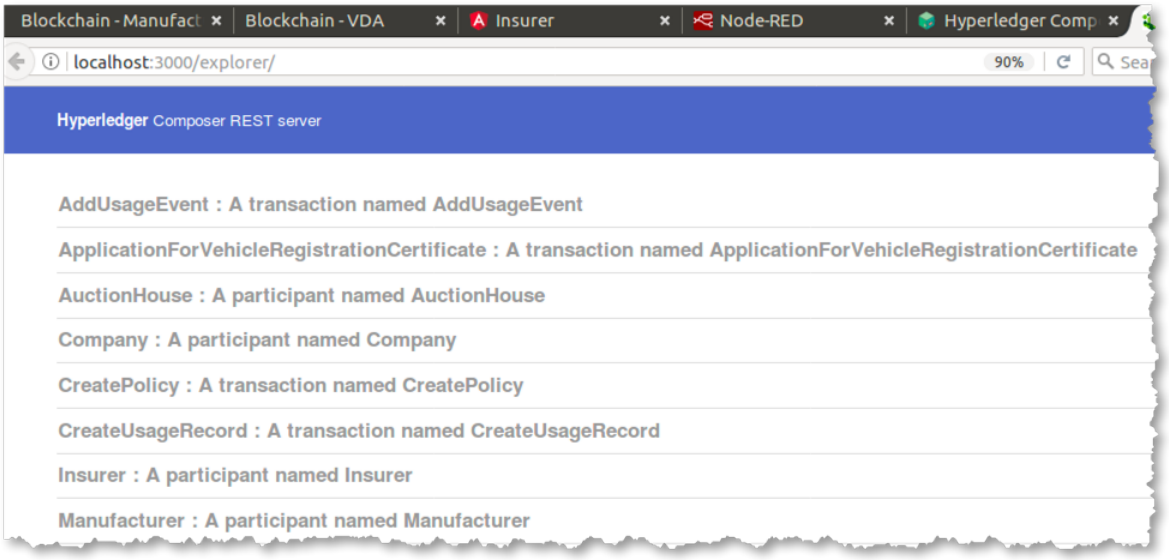
4.2. How the Applications work

While the Playground can be used to test our blockchain scenario, our end-users use mobile apps and dashboards to interact with the running blockchain.

From the files that model this network and implement the transactions, Hyperledger Composer can help this in two ways. Firstly, the models can be used to create skeleton applications that make it easier to develop the end-user applications. Secondly, the models can also be used to generate RESTful APIs that allow client applications and integration middleware to interact with the blockchain.

We will now look at the set of RESTful APIs that have been generated for this scenario.

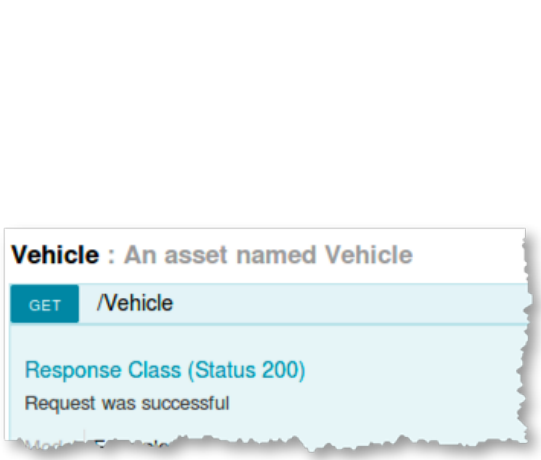
40. Select the Hyperledger Composer REST server tab (localhost:3000/explorer).



This view shows the REST interface that has been generated from the deployed vehicle lifecycle model. End-user applications and integration middleware can invoke these applications by sending HTTP requests that invoke these APIs.

This is how the Node-RED flows interact with the blockchain. Our end-user applications (Paul's mobile app, the VDA view, Insurer dashboard etc.) can also interact in this way, although it is possible for Javascript client applications to instead import (*require*) a Javascript module that interacts the blockchain in a similar way.

41. Review the different APIs available; feel free to try invoking them from the web front end to see what effect it has on the blockchain, on end-user applications and on Playground views.



5. Next Steps

In this lab you have experienced a live blockchain solution through the eyes of four participants of a vehicle network: a buyer/owner, manufacturer, regulator and insurer. A blockchain can be used to great effect in this business network because there is a clear need to share information and value in participants being able to trust the information they see.

Where you go from here is up to you.

If you have a technical background, consider finding out more about the Hyperledger Fabric and Composer technologies and the blockchain development experience. For no charge you can sign up to the IBM Blockchain Platform to play more with the blockchain technology and implementing your first use-case.

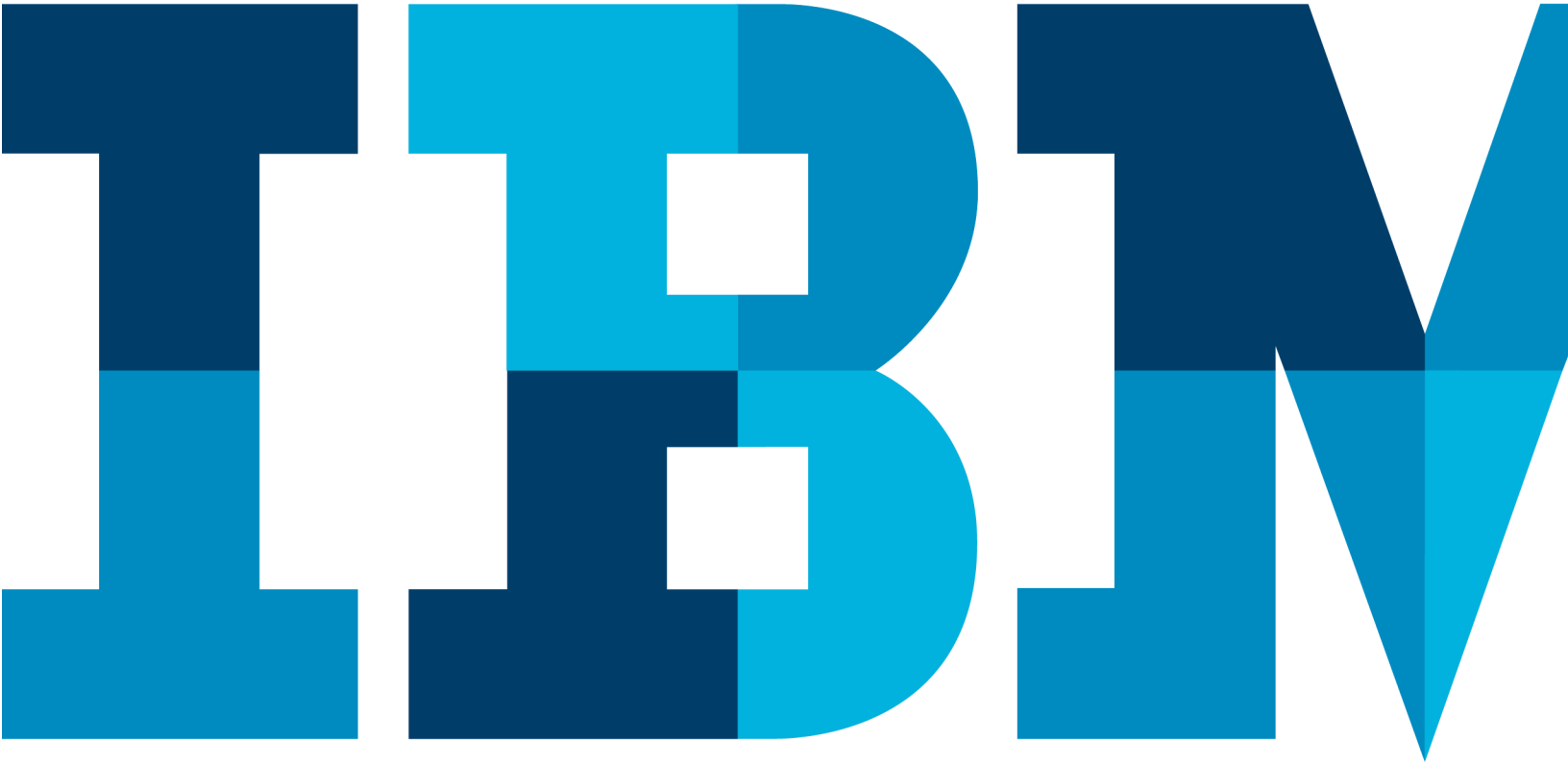
If you are interested in the potential benefits of blockchain in your business, IBM has a bunch of services that can help. Start by going to www.ibm.com/blockchain.

42. Cleanup the hyperledger fabric environment for subsequent labs. Perform the following at the command prompt in the VLD directory:
    a. Open a terminal window
    b. cd VLD
    c. ***./stopAll.sh***

Congratulations on completing the lab!

IBM Blockchain Hands-On
Blockchain Composed

Lab Two – Hyperledger Composer Playground Lab



IBM

Introduction to the hyperledger composer playground lab

Skill requirements:

- There are no skill prerequisites to completing the first section called ‘Car Auction Sample’. It is desirable but not essential to have some background knowledge of JavaScript for the later section called ‘Explore the Editor Views’.

Technical pre-requisites:

- Internet Connection
- Web browser

This section of the lab takes place entirely in the web browser using Hyperledger *Composer Playground*.

Playground simulates the entire blockchain network within the browser by providing a sandpit environment to define, test and explore business networks defined using Composer. It is possible to connect to a live blockchain Hyperledger Fabric instance, or install the Composer Playground on a local machine for more developer friendly tools.

Hyperledger Composer Playground is one method to use Hyperledger Composer, other methods are also available at https://hyperledger.github.io/composer/installing/installing-index.html.

Using Hyperledger Composer Playground

Hyperledger Composer (https://hyperledger.github.io/composer) is an open-source set of tools designed to make building blockchain applications easier.

It allows users to model the business networks, assets and transactions that are required for blockchain applications, and to implement those transactions using simple JavaScript functions. The blockchain applications run on instances of Linux Foundation Hyperledger Fabric (www.hyperledger.org).

The purpose of this lab is to introduce you to the concepts of a blockchain by showing you how a blockchain transfers assets between participants in a business network. We will use the implementation of a simple blind car auction as the scenario for the demo.

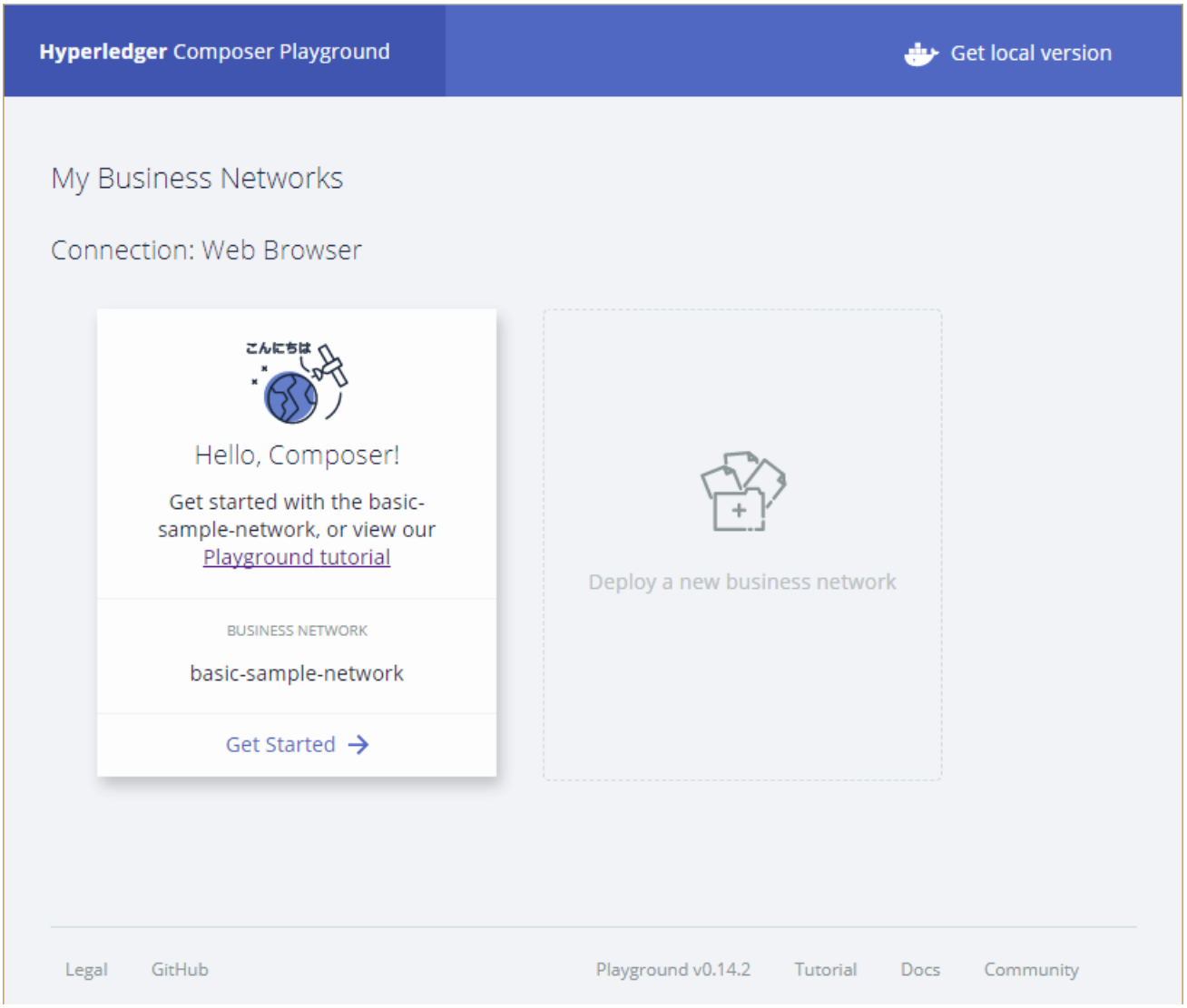
The car auction business network has a set of known participants (buyers and sellers), assets (cars and car listings) and transactions (placing bids and closing auctions). We will model these using Hyperledger Composer Playground and test the business logic that makes the auction work.

Crucially, a blockchain could be used to bring together the buyers and sellers of these assets without needing any trusted third party. However, an auctioneer could be used to provide visibility and governance of the network if required.

Car Auction Sample

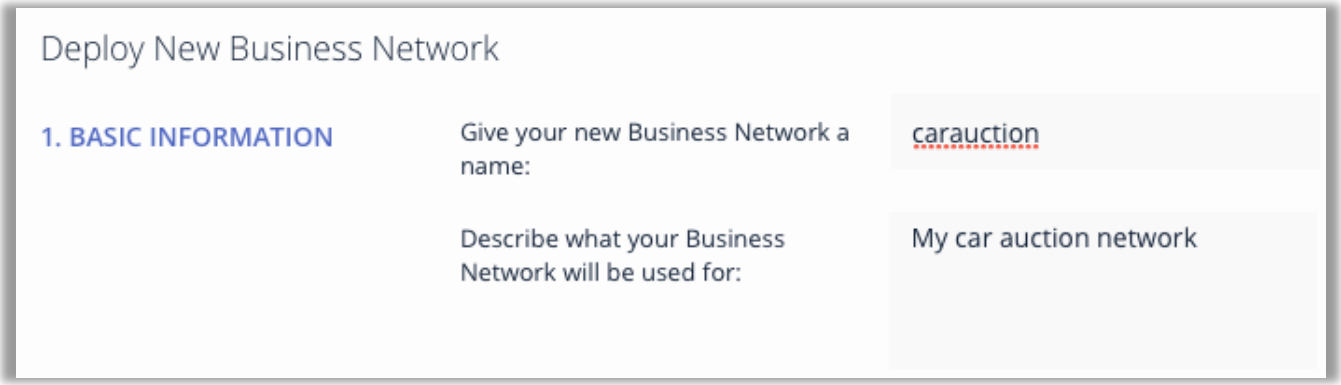
1.1.1. Open the Playground

1. Open a web browser and go to http://composer-playground.mybluemix.net. Dismiss the welcome screen to show the playground wallet screen which is used to connect and deploy new business networks:

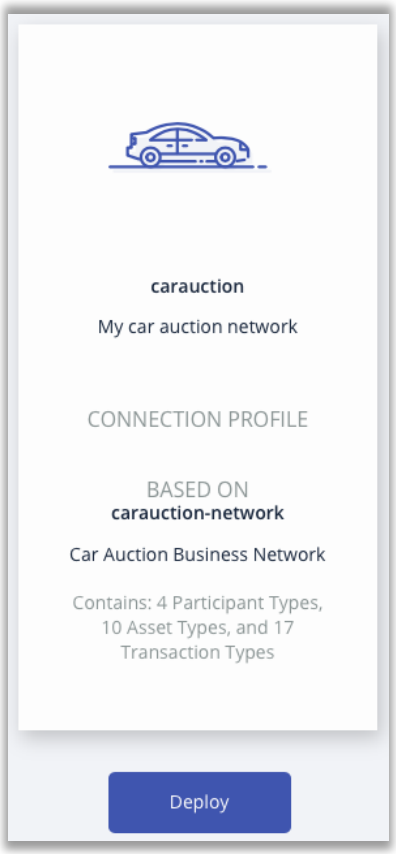


2. Click the “Deploy a business network” box. Then scroll down and select the carauction-network:

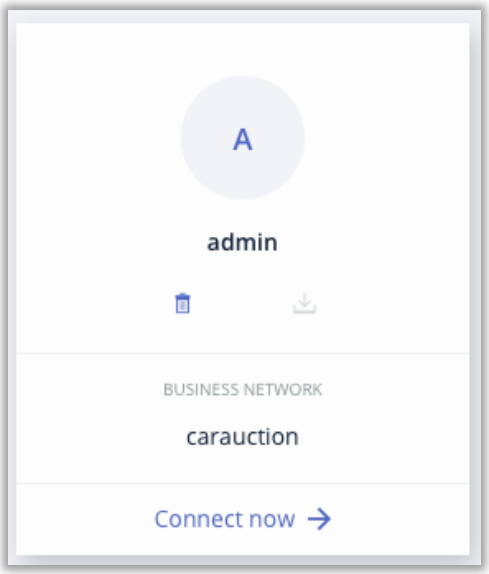
3. Next give the business network a name and description:



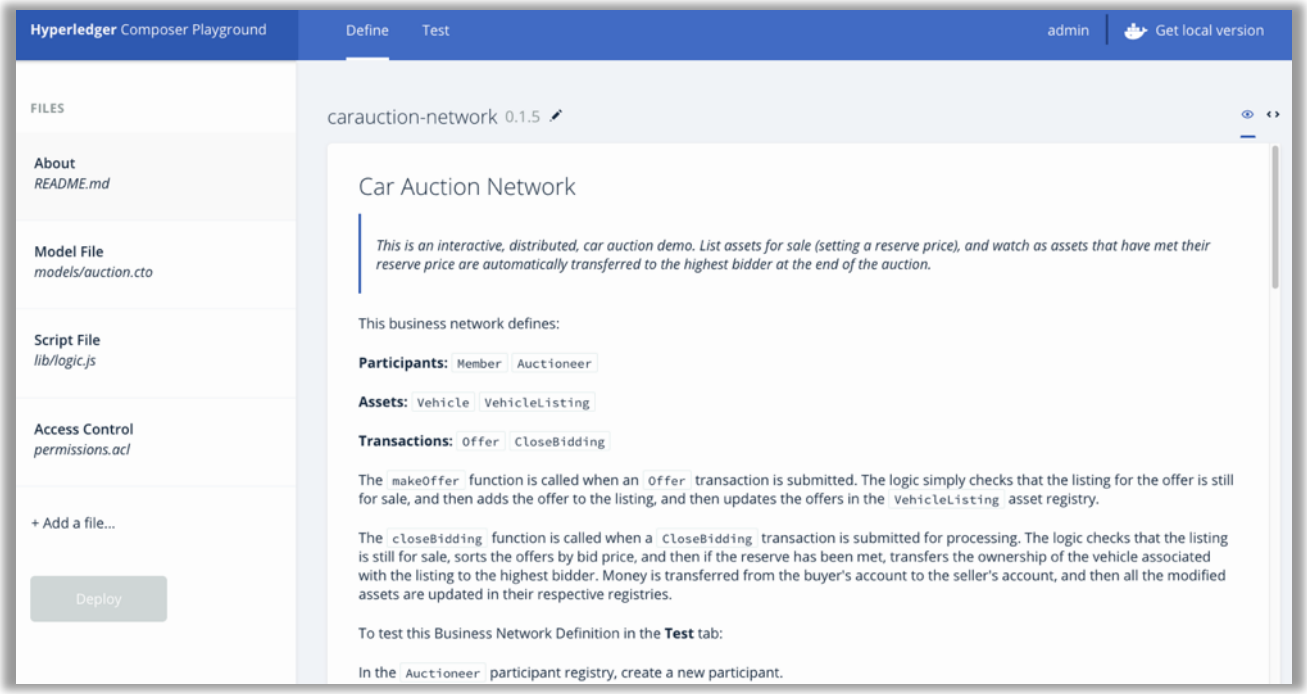
4. Click the Deploy button to deploy the new car auction business network:



5. Click “Connect now” in the new identity card for the carauction network:



6. Take a few minutes to read through the description of the car auction sample, to help understand the participants, assets and transactions associated with this particular network.



1.1.2. Add Three Participants

In the next section we will now work with the deployed car auction blockchain network.

We will first instantiate three *Member* participants of the car auction business network:

- Alice Smith (alice@email.com), who will make a bid on a car,
- Bob Jones (bob@email.com), who will also make a bid on a car, and
- Charlie Brown (charlie@email.com), who currently owns a car.

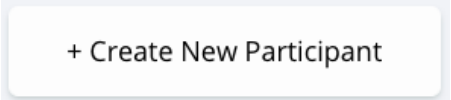
We will not instantiate an Auctioneer in this demo; this could be used in order to provide oversight of the network, although is not necessary.

7. Click the **Test** tab and then click on the *Member* participant registry:

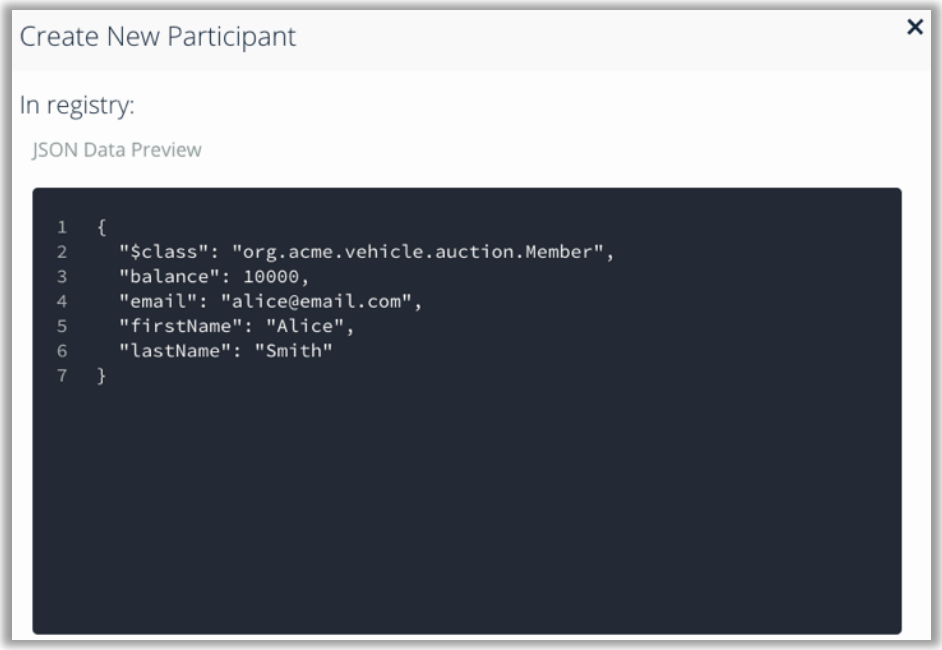
The registry is empty as no members have currently been defined.

8. Click on **Member** to view there are no current members in the environement

9. Click **Create New Participant** to add a new Member.

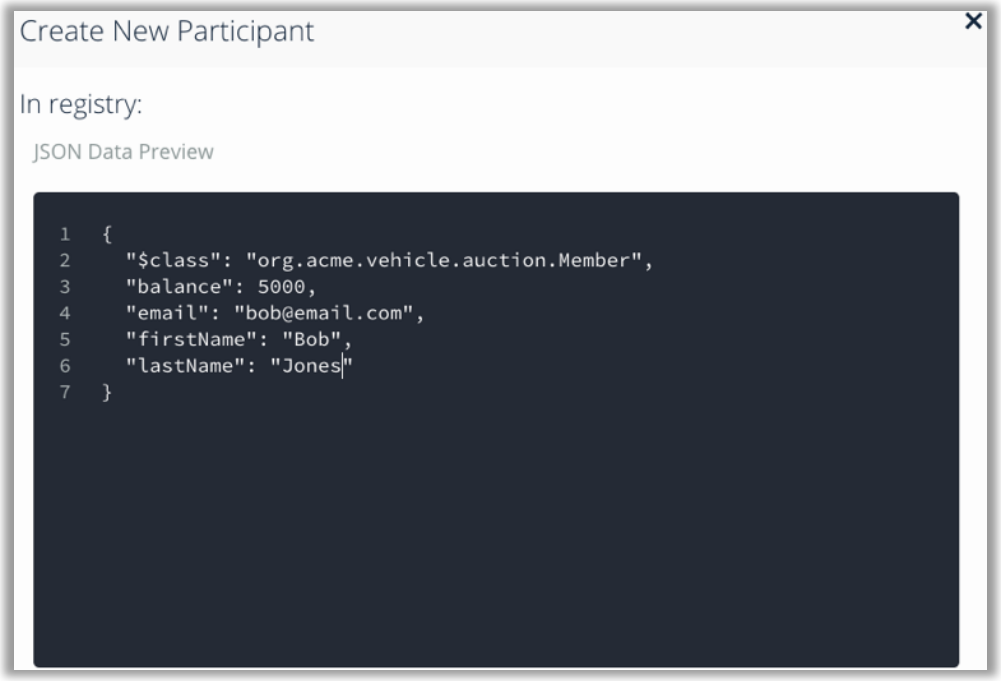


10. Type the correct values into the JSON data structure to add Alice to the business network. Let's give her a starting balance of 10000.

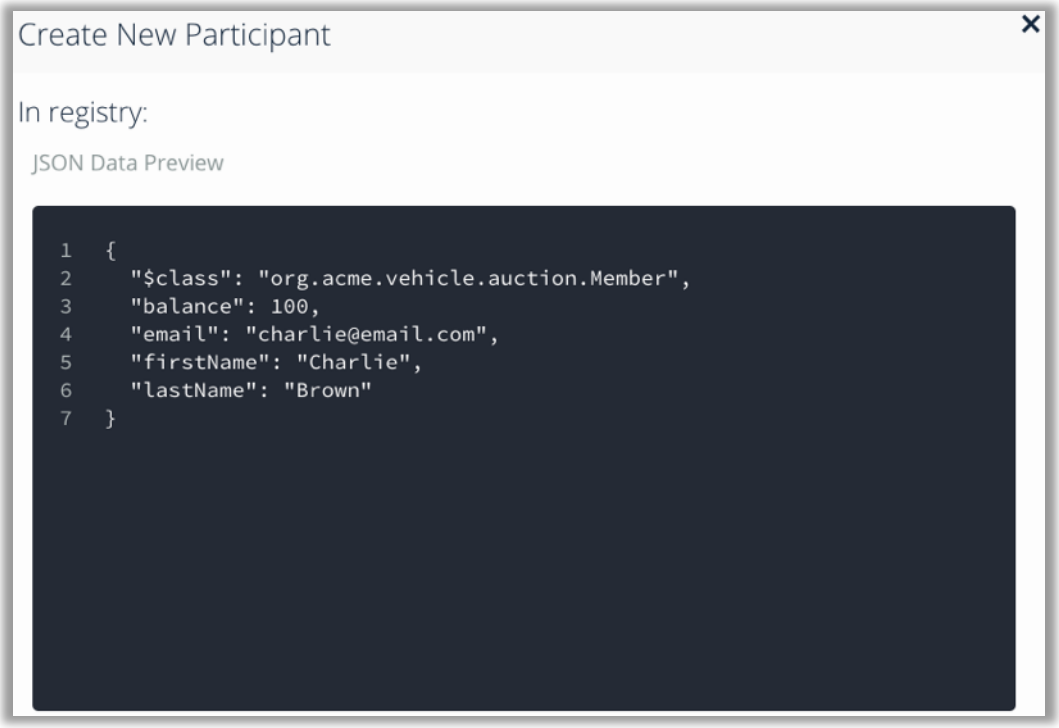


11. Click **Create New** to add Alice to the registry.

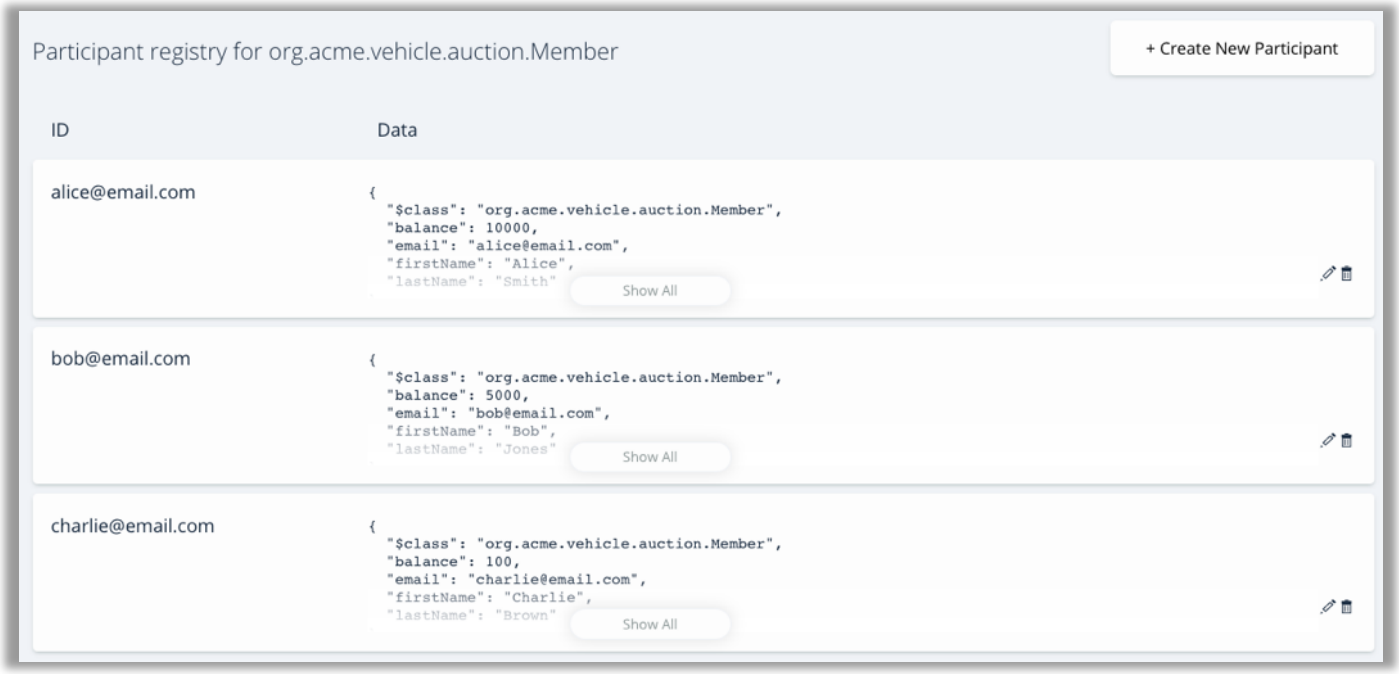
12. Do the same for Bob. Let's give him a starting balance of 5000.



13. Finally do the same for Charlie. He hasn't got so much money (he's selling his car, after all) so let's give him a starting balance of 100.



14. Verify that all participants in the business network have been correctly defined. Use the appropriate Edit button ( ) to make any changes.



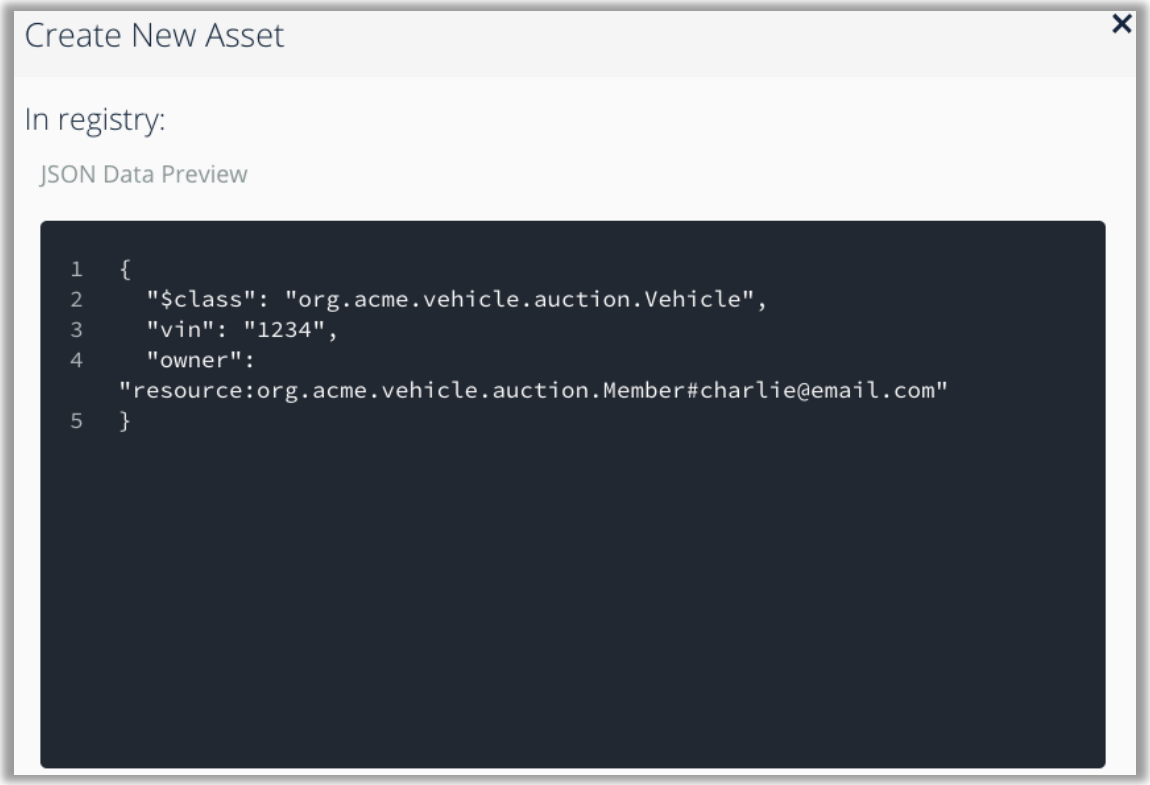
1.1.3. Add an Asset

We will now add Charlie's car to the Vehicle Asset registry.

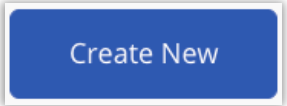
15. Click the ***Vehicle*** asset registry.

16. This registry contains no assets currently. Click the **Create New Asset** button to add a new asset.

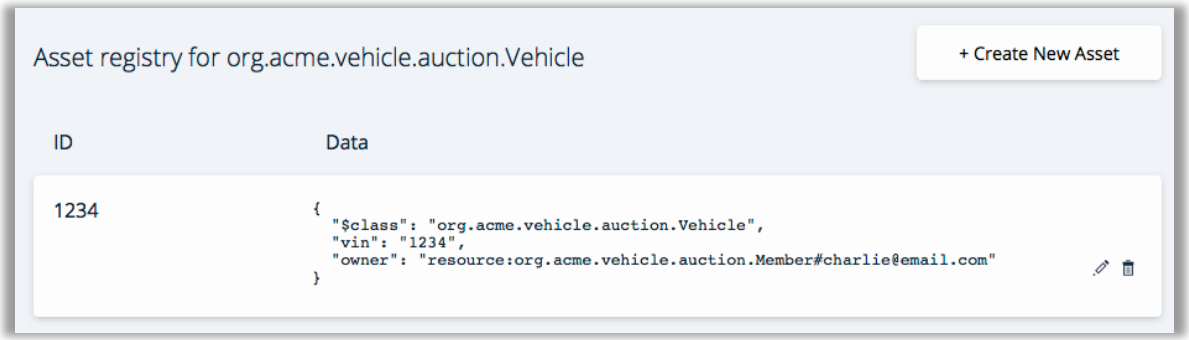
17. Instantiate the car by adding a vehicle identification number (VIN) of 1234 and assign it to Charlie by filling in the JSON object as follows. (We use his email address to identify him; this was specified as the key field in the User definition using the 'identified by' statement.)



18. Click **Create New** to add the new vehicle to the registry.



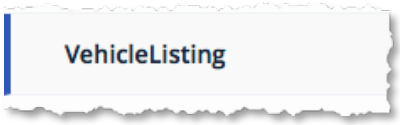
19. View your newly added asset in the registry.



1.1.4. Add a Vehicle Listing

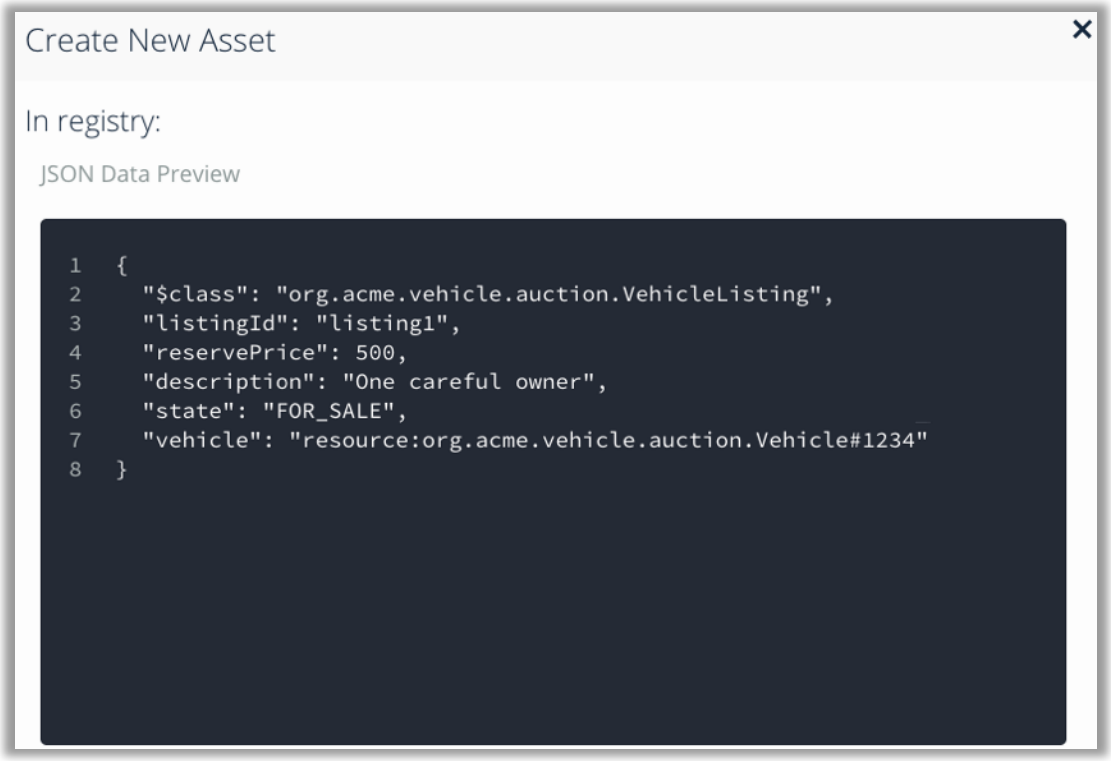
In this section we will put the car up for sale by creating a *VehicleListing* instance.

20. Click the ***VehicleListing*** asset registry. Once more, the VehicleListing registry should be empty.

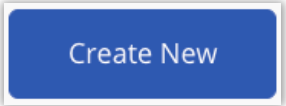


21. Click the **Create New Asset** button to add the asset.

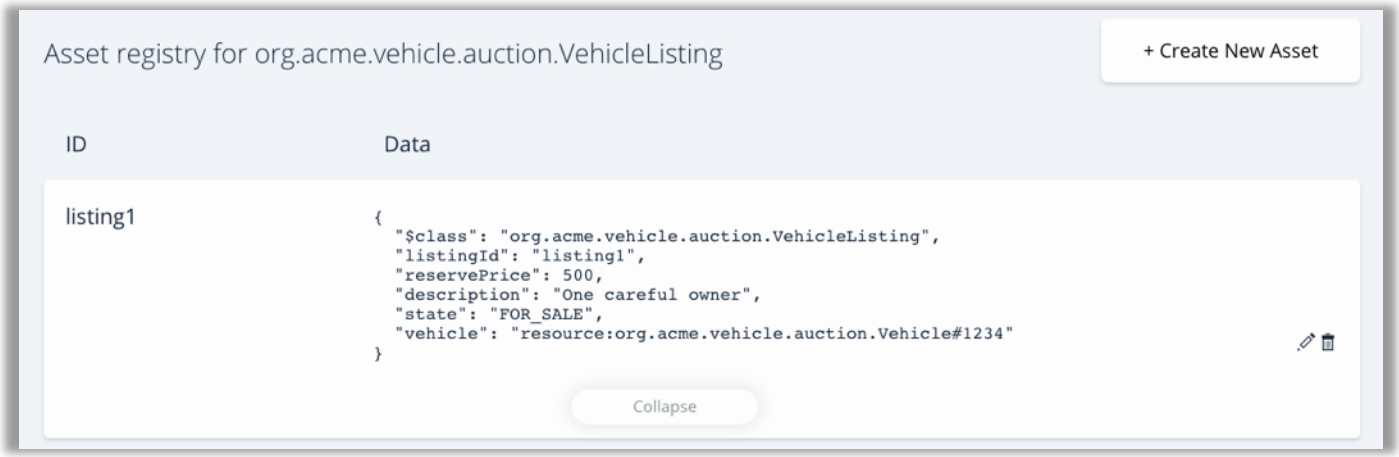
22. Update the fields and remove the random offers to show the below. Syntactic validation of the object occurs at this point, so correct any errors if necessary.



23. Click **Create New** to add the new vehicle listing to the registry.



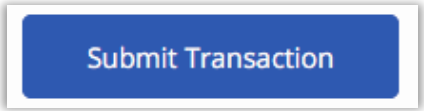
24. View the listing in the registry.



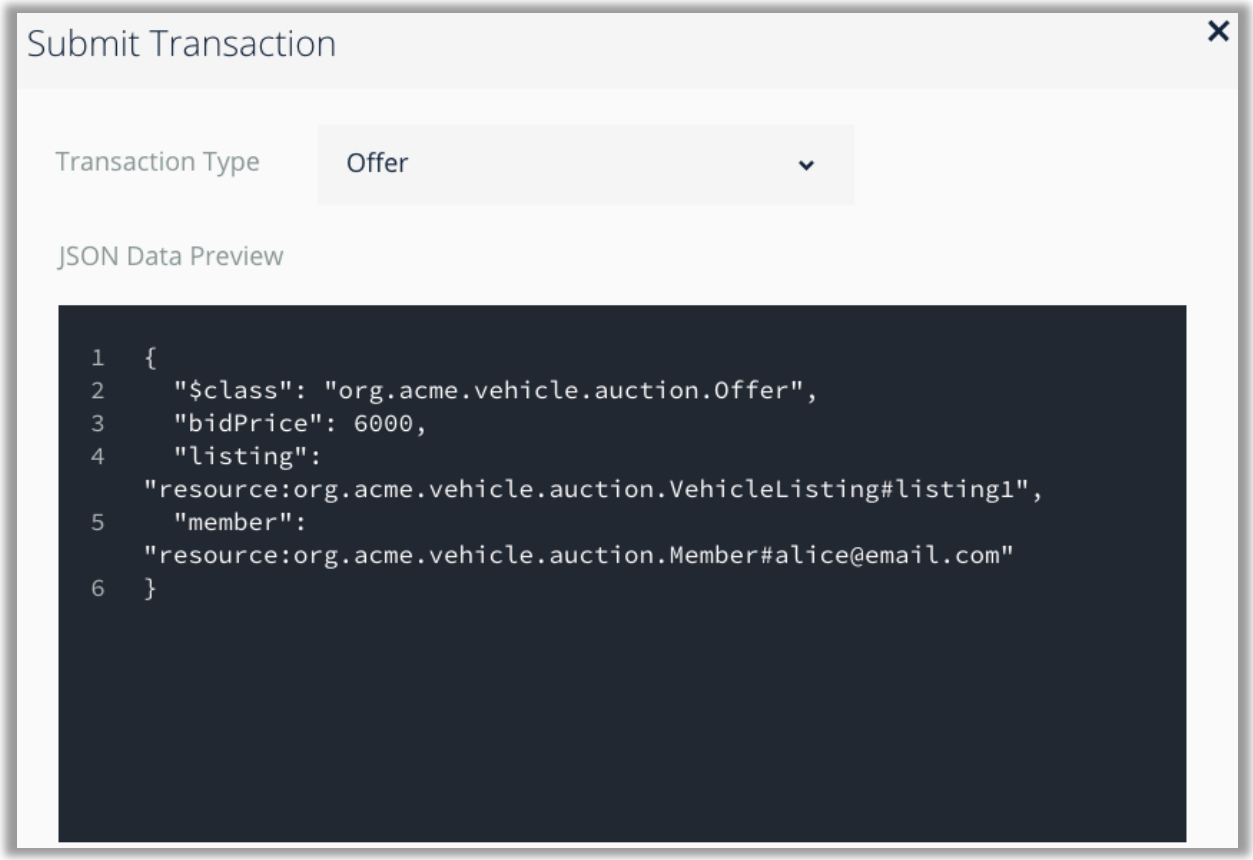
1.1.5. Submit offers on the vehicle

We will now let Alice and Bob bid on the vehicle.

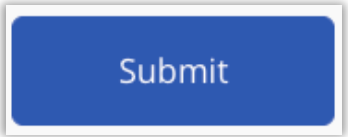
25. Click on the Submit Transaction button



26. Let Alice put in a bid of 6000.

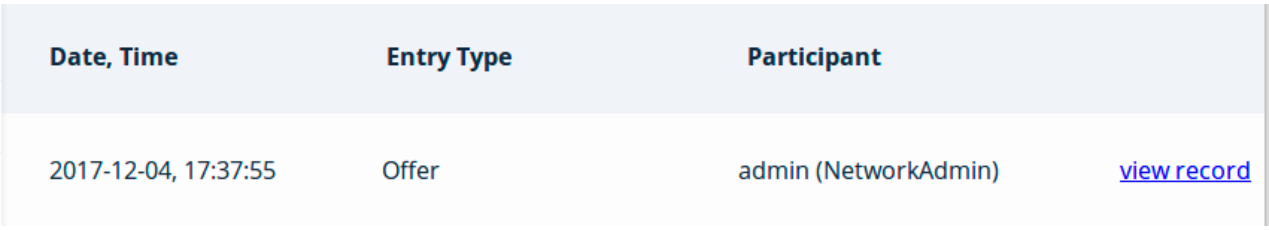


27. Click **Submit** to submit the offer transaction.



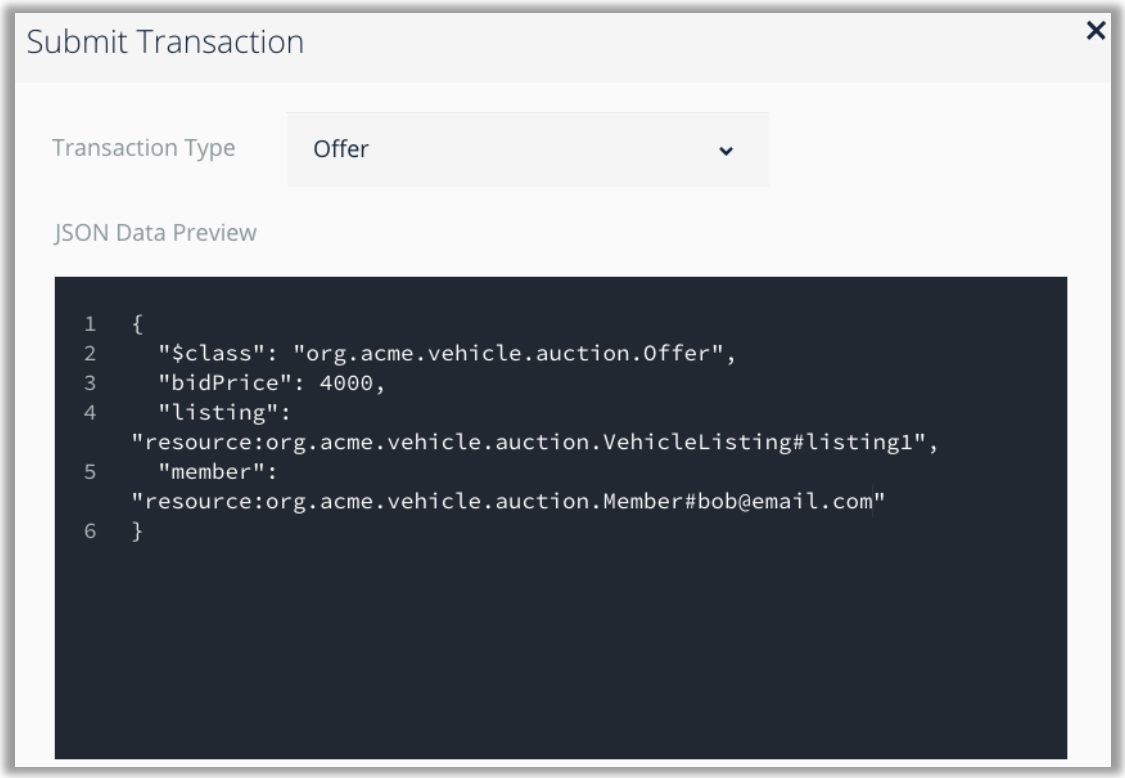
28. See the transaction successful appear in the Historian registry. Swich to view all transactions by clicking ‘All Transactions’:

29. You will also notice additional transactions for creating participants and assets. Click “view record” for more information.



Date, Time	Entry Type	Participant	
2017-12-04, 17:37:55	Offer	admin (NetworkAdmin)	view record

30. Let Bob put in a bid of 4000.



31. Verify the transactions in the registry.

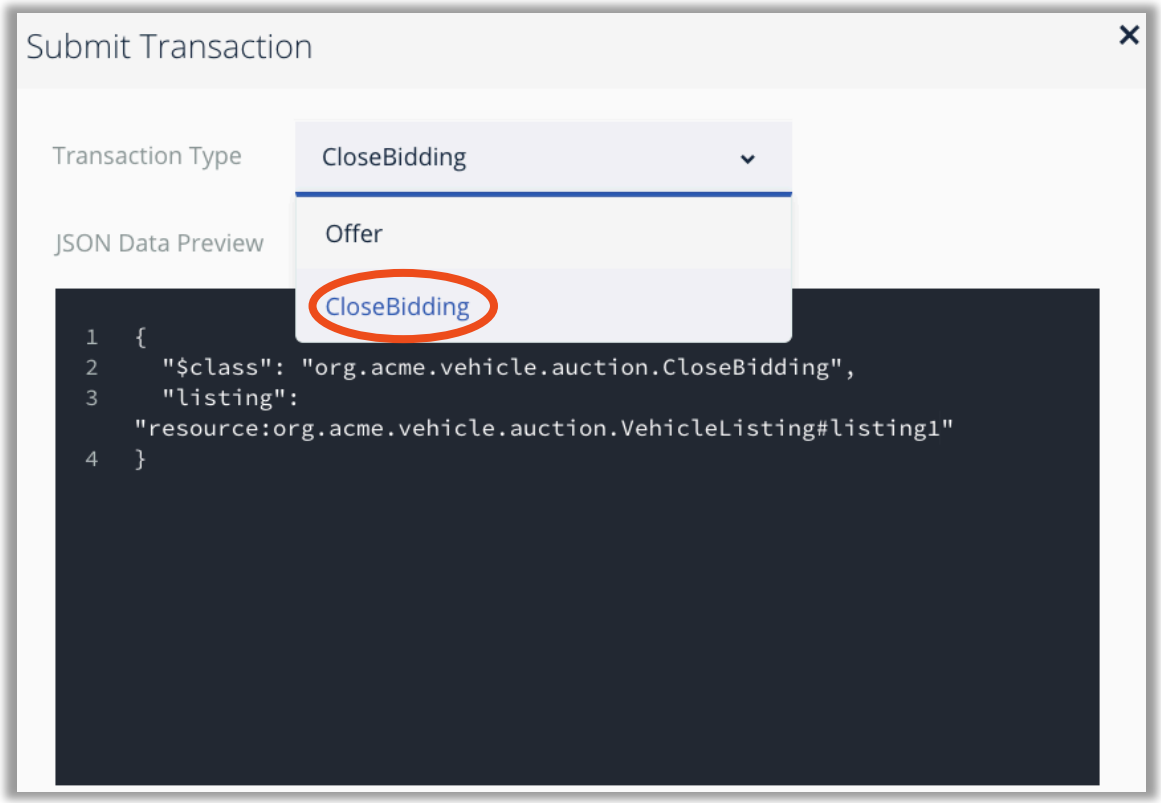
Date, Time	Entry Type	Participant	
2017-12-04, 17:43:19	Offer	admin (NetworkAdmin)	view record
2017-12-04, 17:37:55	Offer	admin (NetworkAdmin)	view record

Note that the transactions cannot be edited or individually deleted once submitted; this is one of the defining characteristics of a blockchain.

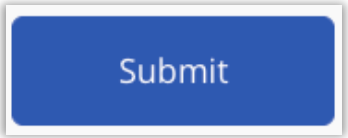
1.1.6. Closing the bidding

To close the bidding on the listing we need to submit a *CloseBidding* transaction.

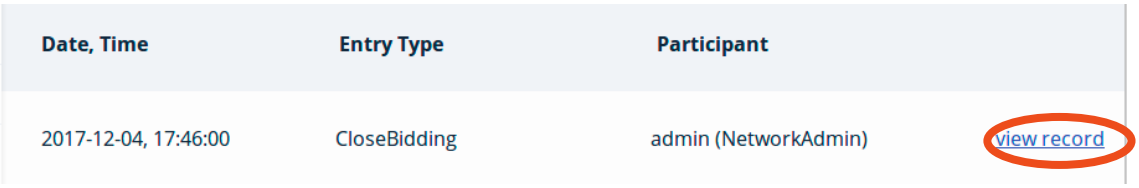
32. Submit a new transaction, this time selecting **CloseBidding** from the drop-down 'Transaction Type' field.



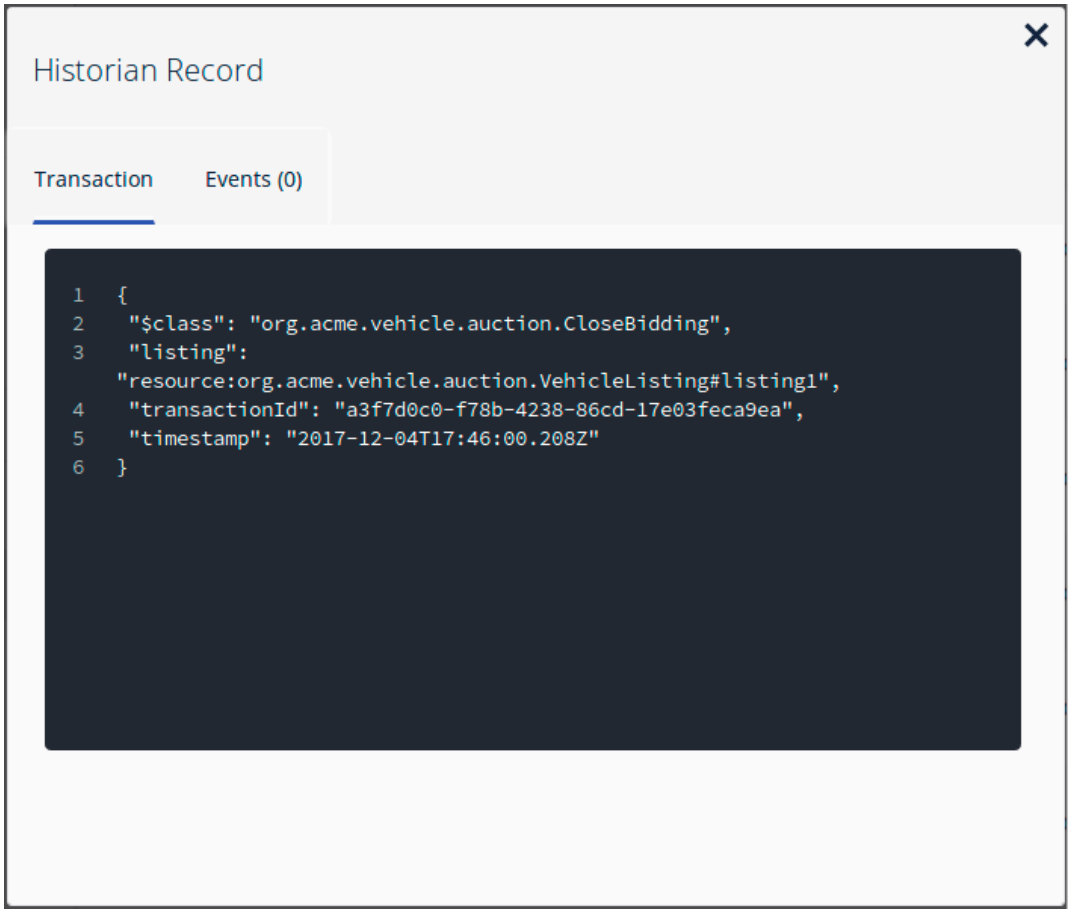
33. Click **Submit** to submit the CloseBidding transaction.



34. Verify that the transaction has been added to the blockchain transaction registry. Click on 'view record' to see the content of the transaction.



Date, Time	Entry Type	Participant	
2017-12-04, 17:46:00	CloseBidding	admin (NetworkAdmin)	view record

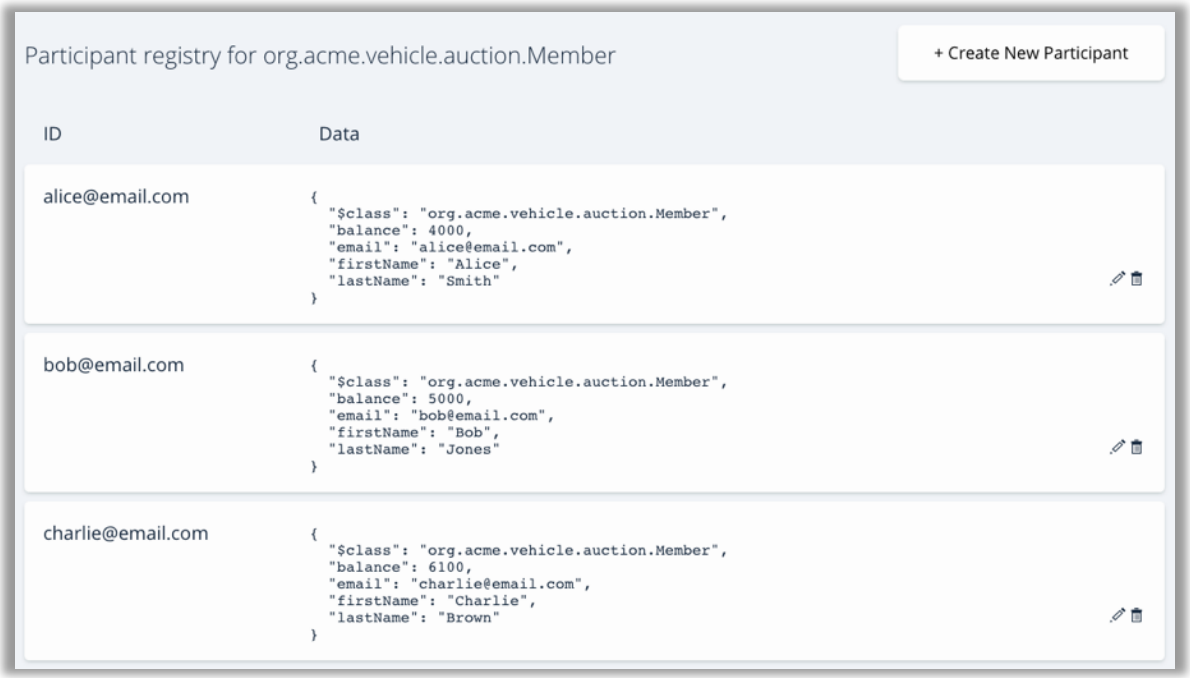


Based on the bids we submitted, Alice should now be the owner as she put in the highest bid. We should also be able to verify that the owner of the car has changed and appropriate balances increased or decreased accordingly.

35. Go to the ***Vehicle*** asset registry to see the vehicle owner has been updated to Alice.

36. You will see the following vehicle owned by Alice in the vehicle registry.

37. Go to the ***Member*** asset registry to see that Charlie's balance has increased by the winning bid amount, and that Alice's balance has decreased by the same.

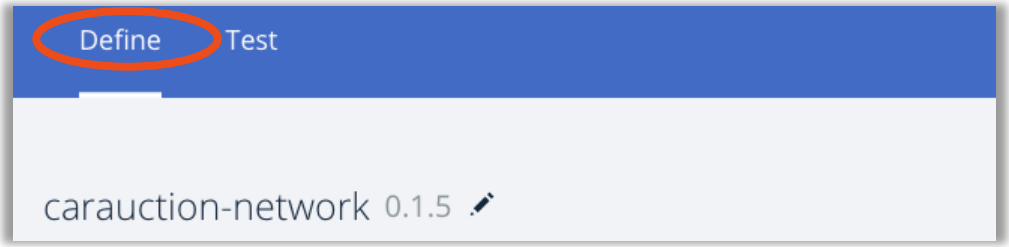


Congratulations! You have completed the first part of this lab.

Explore the Editor Views

1.1.7. Model File

38. Click on the define tab to go back to the main playground window.



39. Click the Model File (models/auction.cto) to open it.

This .cto file models the assets, participants and transactions for this blockchain application.

40. Look at the Vehicle asset:

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

This uses the Hyperledger Composer Modeling Language which will be looked at more later. An *asset* is anything of worth that will be transferred around the blockchain. Here we can see the asset class is called '*Vehicle*' and will have an associated *vin* and a reference (indicated by "-->") to a '*Member*' participant that we will call '*owner*'.

41. Type and add some characters in an appropriate point to show the live validation of the model.

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

Error found!

Error: Syntax error in file undefined. Expected "extends", "identified by", "{", comment, end of line or whitespace but "i" found. Line 17 column 22

42. Scroll down and look at the abstract '*User*' participant.

The participant will be the people or companies within the business network. Each *User* participant will be defined as having a *email*, *firstName* and *lastName*. As the class is **abstract** instances of it cannot be created; instances are instead implemented by the *Member* and *Auctioneer* classes.

```
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 {
}
```

Here the user can become a *Member* requiring a *balance*, or an *Auctioneer* that does not.

43. Look at the *Offer* and *CloseBidding* transaction definitions:

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

transaction CloseBidding {
  --> VehicleListing listing
}
```

The *transaction* definitions give a description of the transactions that can be performed on the blockchain. They are implemented in a Transaction Processor file using the Javascript language.

1.1.8. Transaction Processors

44. Click on the lib/logic.js file:

45. Scroll to **the bottom of the file** to review the logic used to make an offer on a car being auctioned:

```
/**
 * 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);
        });
}
```

This implements the *makeOffer* function, which is executed when the *Offer* transaction is invoked on the blockchain. (It is the **@param** comment above the function that links the full transaction name as defined by the model to the Javascript method that implements it.)

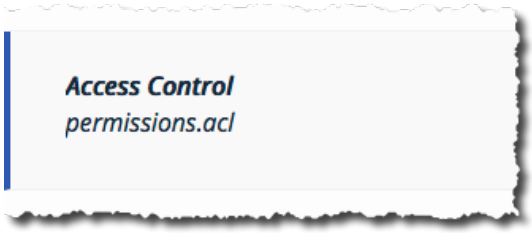
Other Interesting areas of the function implementation include:

a) The logic that the vehicle must be for sale to submit an offer on it
b) The retrieval and update of the asset registry a few lines later
c) Saving the updated asset back to the registry

1.1.9. Access Control List

The final file that defines the blockchain application is the Access Control List, which describes the rules which govern which participants in the business network can work with which parts of the blockchain.

46. Click the permissions.acl file:



47. Look at the ACL rules defined:

```
/**
 * Access Control List for the auction network.
 */
rule Auctioneer {
    description: "Allow the auctioneer full access"
    participant: "org.acme.vehicle.auction.Auctioneer"
    operation: ALL
    resource: "org.acme.vehicle.auction.*"
    action: 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"
```

The rule allows or denies users to access aspects of the blockchain.

Updating the Model (Advanced and Optional)

48. Try updating the model (*auction.cto*) for the *Vehicle* asset definition to include manufacturer make and model fields. Add in new *String* fields and click 'Deploy' to make the changes live.

    Note that when you update the model, the syntax of any existing assets in the registry must be compatible with the new model. Use either the **optional** or **default="…"** qualifiers next to the new fields. If you make incompatible changes, you must first reset the demo.

    Once you've deployed the changes, try adding new *Vehicle* assets to the registry to test the changes.

    For more information on the Hyperledger Composer modelling language please refer to: https://hyperledger.github.io/composer/reference/cto_language.html

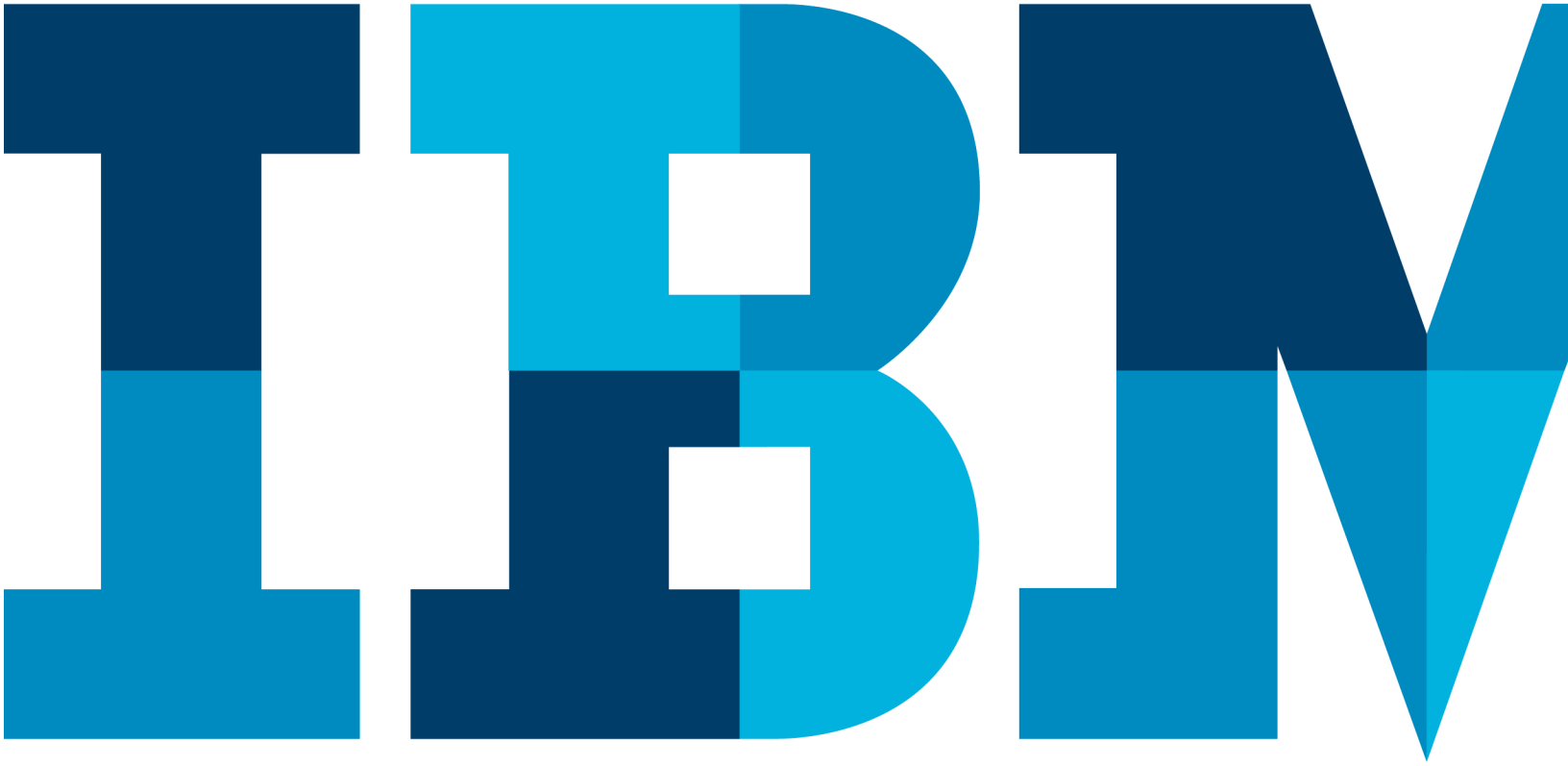
Export the Business Network Archive

49. Exporting to a Business Network Archive will save the Read Me, Model File(s), Script File(s) and Access Control rules that can be easily imported to a local developer enviroment, handed to a network operator to deploy to a live network or saved asa backup. More details on local installation at https://hyperledger.github.io/composer/installing/installing-index.html.

Congratulations! You have completed this lab.

IBM Blockchain Hands-On
Blockchain Composed

Lab Three – Hyperledger Composer Developer Lab



IBM

Introduction to hyperledger composer development lab

The purpose of this lab is to introduce you to the Hyperledger Composer development environment. It is intended to be run on any machine that can meet the Hyperledger Composer specification.

```
Operating Systems: Ubuntu Linux 14.04 / 16.04 LTS (both 64-bit), or Mac OS 10.12
Docker Engine: Version 17.03 or higher
Docker-Compose: Version 1.8 or higher
Node: 6.x (note version 7 is not supported)
npm: 3.10.x
git: 2.9.x
A code editor of your choice, we recommend VSCode.
```

Where to start with hyperledger composer development?

Section 1 will lead you through the installation instructions for Hyperledger Composer and Hyperledger Fabric.
Section 2 will lead you through the creation, deployment and testing of a sample business network application. It will also show you how to generate a REST interface.

If you are running on a machine that has not been configured for Hyperledger Composer (for example, your personal laptop), then install the pre-requisites above and then start with Section 1.

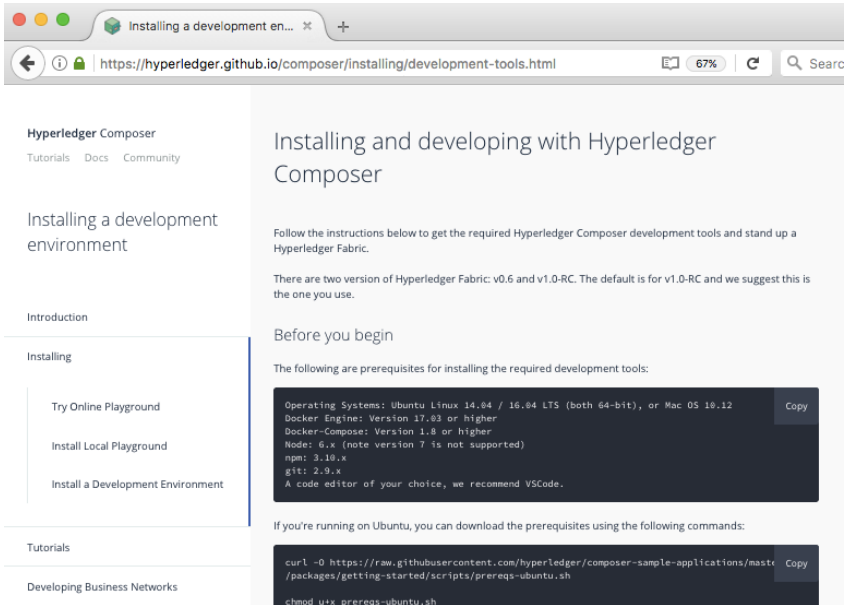
If you are running on a machine that is provided for you as part of a classroom environment, your instructor will tell you where to begin this lab.

Section 1: Installing hyperledger composer development tools

The master copy of the instructions for this section are online. It is recommended that you use the online version where possible, as this may contain updates to the instructions.

1. **Optionally just read thru the material on the Web page below for your own reference. These steps have already been performed for you on the VMWare image.** Bring up a web browser and navigate to the following page:

   https://hyperledger.github.io/composer/latest/installing/development-tools



2. Read thru the material in the link above ***but do not perform any of the steps***.

3. Once you have been able to successfully start the fabric and create a composer profile, you will have completed this section. Run the following shell commands and scripts to ensure the containers are started and the environment is ready for you:
   - **__a. cd ~/fabric-dev-servers**
   - **__b. export FABRIC_VERSION=hlfv11**
   - **__c. ./teardownFabric.sh**
   - **__d. ./startFabric.sh**
   - **__e. ./createPeerAdminCard.sh**
   - **__f. cd ~**

Section 2: hyperledger composer developer and queries Tutorial

The master copy of the instructions for this section is online. The online site allows you to more easily copy and paste snippets of text, which is necessary for some of the steps.

4. Bring up a web browser and navigate to the following page:

   https://hyperledger.github.io/composer/latest/tutorials/developer-tutorial.html

   Follow the instructions contained within this page starting at the **Create a business network structure** section. Ensure your terminal window is open to the **/home/blockchain** directory as subdirectories will be created in that directory. There is no need to install Hyperledger Composer or the Visual Studio Code Editor since the VMWare image has this installed for you. For Step 2: Defining a Business Network, launch the Visual Studio Code editor on your desktop using the  icon to edit source artifacts. Within Visual Studio Code editor, select Open Folder from the File menu, and browse to the tutorial-network directory (/home/blockchain/tutorial-network) created in Step 1 of the tutorial.

5. Bring up a web browser and navigate to the following page:

   https://hyperledger.github.io/composer/latest/tutorials/queries

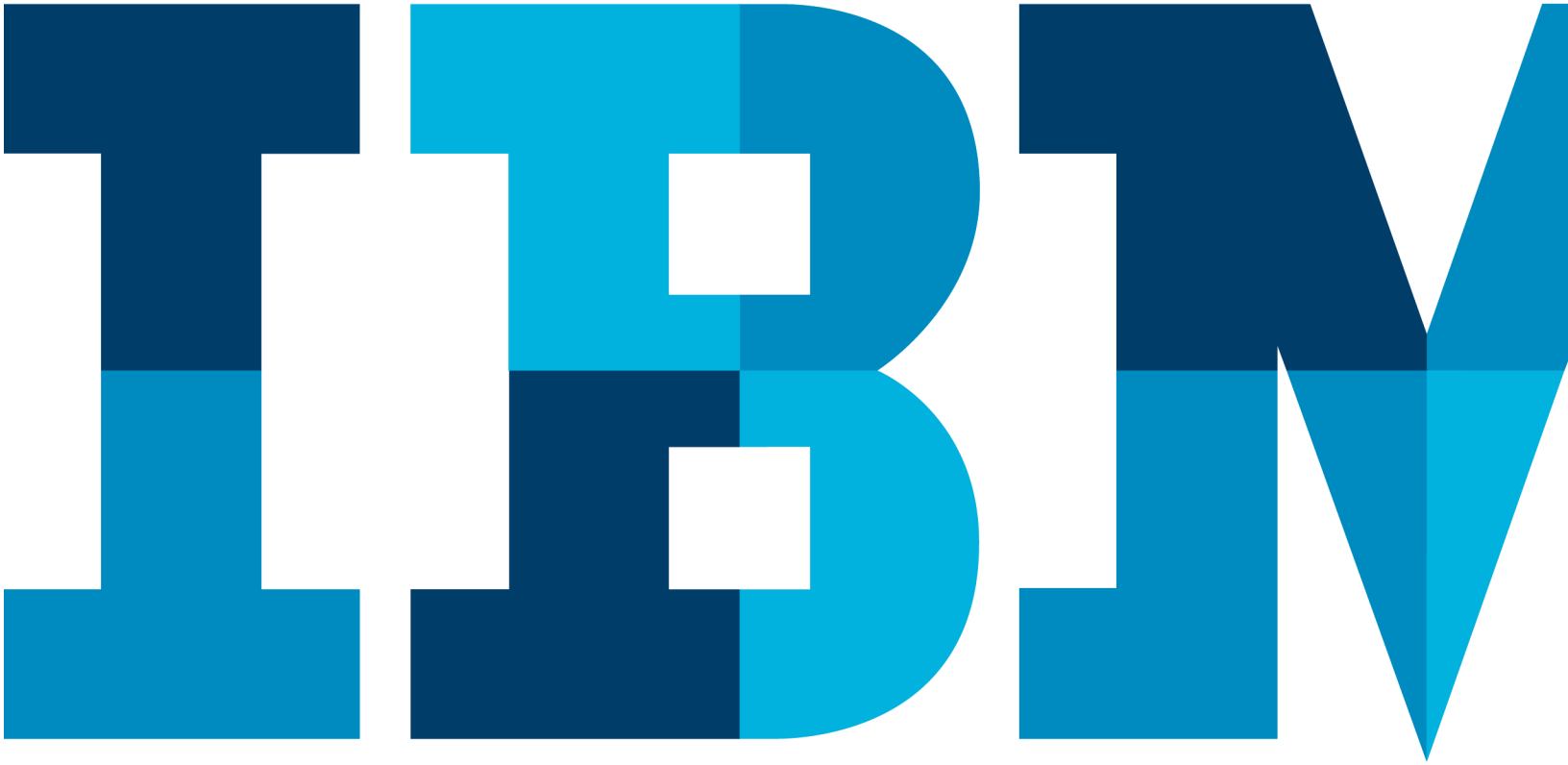
   Follow the instructions contained within this page. Once you have been able to successfully validate you only have one commodity, you will have completed this section. Be sure to complete step 6 below to clean up the environment before moving on to the next lab.

6. Cleanup the Hyperledger Fabric environment for the next lab. Perform the following steps:
   - **__a. Enter control-C at the terminal window where the composer-rest-server is running to stop the server.**
   - **__b. cd ~/fabric-dev-servers**
   - **__c. export FABRIC_VERSION=hlfv11**
   - **__d. ./stopFabric.sh**
   - **__e. ./teardownFabric.sh**

IBM Blockchain Hands-On

Blockchain Explored

Lab 4 – Hyperledger Fabric Lab



IBM

Introduction to the hyperledger fabric lab

The purpose of this lab is to enable you to write your first blockchain application by introducing you to the Hyperledger Fabric SDK.

Prerequisites

The lab can be run on any supported level of Mac OSX, Linux and Windows machines. A browser and internet connectivity is required to complete the lab.

Please note that as several hundred MBs in the form of docker images will be downloaded, suitable internet bandwidth and disk space is required.

The following prerequisite software are also required:

- Git commandLine
- cURL (or Windows equivalent)
- Docker
- Docker Compose
- Node.js

It is important to ensure the correct versions of Docker, Docker Compose and Node.js are installed. Incorrect versions will lead to random errors. Please follow directions on this page for installing the correct versions: http://hyperledger-fabric.readthedocs.io/en/latest/prereqs.html

Writing your first Hyperledger Fabric application

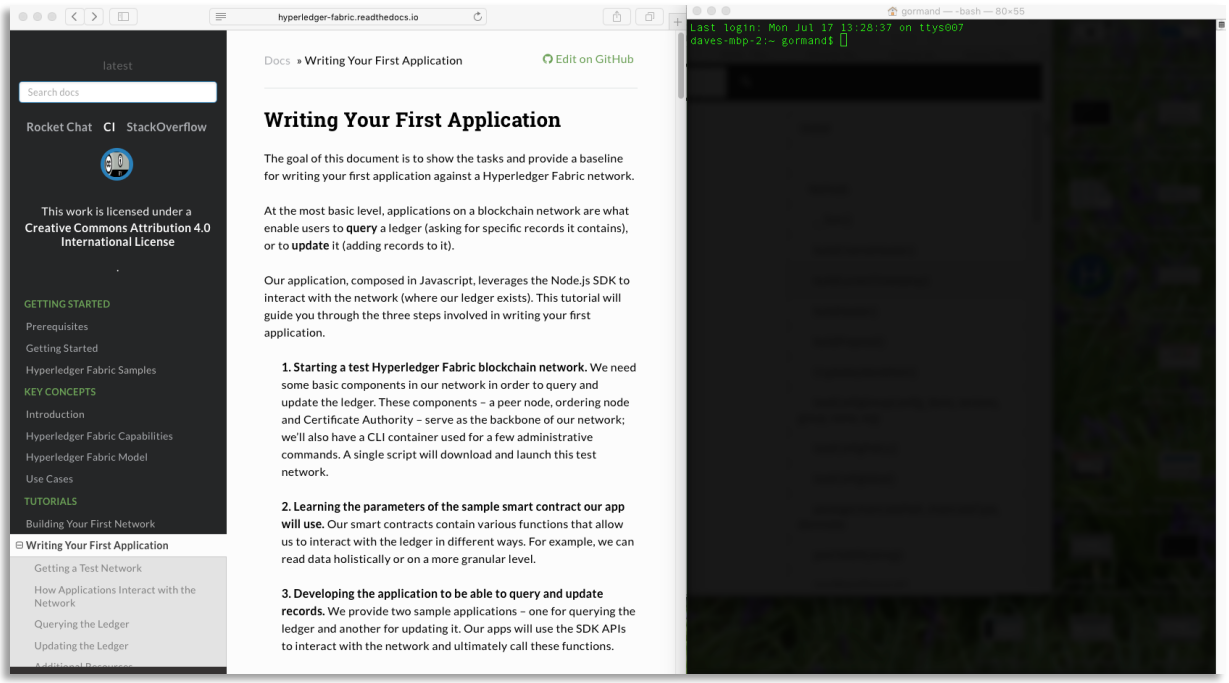
The master copy of the instructions for this lab are online. It is recommended that you use the online version where possible, as this may contain updates to the instructions. The online site also allows you to more easily copy and paste snippets of text, which is necessary for some of the steps.

1. Bring up a web browser and navigate to the following page:

   http://hyperledger-fabric.readthedocs.io/en/latest/write_first_app.html

   You will also need to open a terminal window.

   It is recommended that you open the browser side-by-side with the terminal screen, as you will be working from the browser page and following the instructions in the terminal window.



2. Execute the following to clone the fabcar examples:
   - __a. **cd ~**
   - __b. **curl -sSL https://goo.gl/6wtTN5 | bash -s 1.1.0**
   - __c. **cd ~/fabric-samples/fabcar**
   - __d. **edit the package.json file with the vi editor or VS Code**
   - __e. **modify** dependencies section so it uses explicit package versions as follows:

     "dependencies": {
     "fabric-ca-client": "1.1.0",
     "fabric-client": "1.1.0",
     "grpc": "1.10.1"

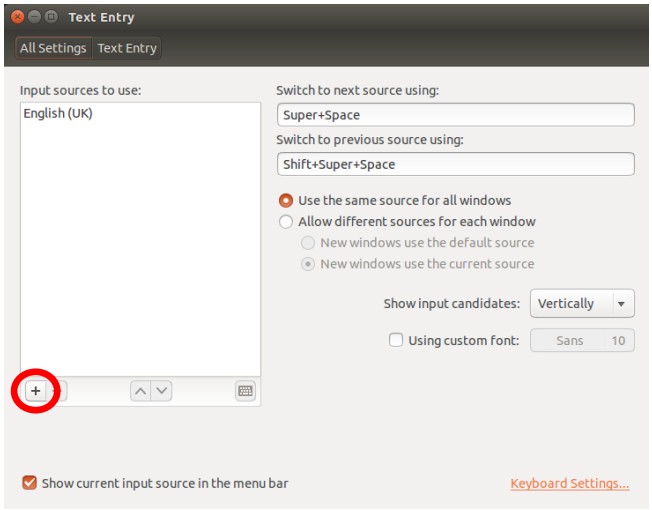
3. Follow all the instructions contained within the tutorial starting at the **Install the clients & launch the network**. Do not visit the prerequisites page as the prereqs have been installed for you. Ensure your terminal window is initially open to the **/home/blockchain** directory. Once you have successfully run the query.js and invoke.js applications to transfer ownership of a car, you will have completed the lab

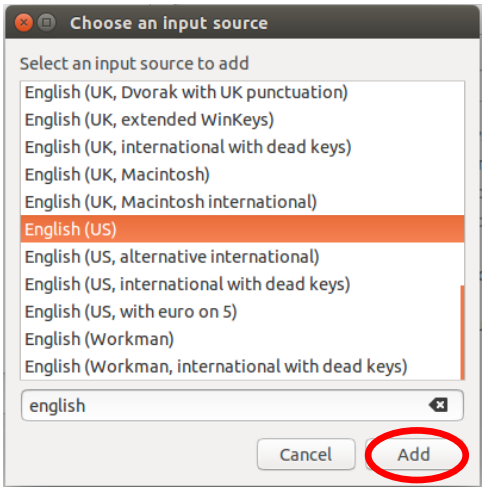
Appendix A. Keyboard Language Change

To change the keyboard language to enable you to use foreign laptops follow these steps:

Click on the  icon in the top right & select **Text Entry Settings...**

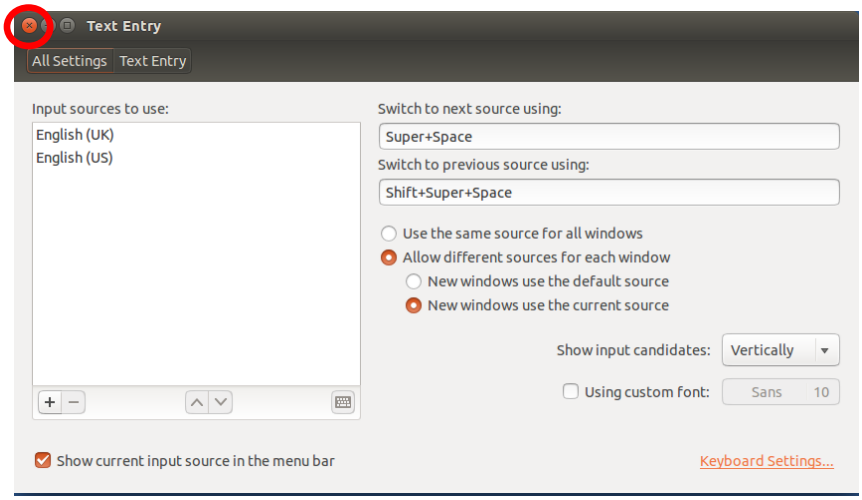
Select the  symbol



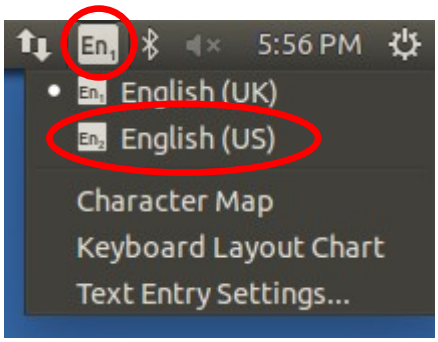


Type your **Language** (E.G. English) and then **country** (E.G. US)
Select the appropriate keyboard and click '**Add**'

Close the Settings box



Select the '**EN**' in the top right of the screen and select your new keyboard



Your keyboard is now ready to use

Appendix B. Notices

This information was developed for products and services offered in the U.S.A.
IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.
IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.
For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:
IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan
The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.
Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.
This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.
Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.
IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.
Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.
Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM

Appendix C. Trademarks and copyrights

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM	AIX	CICS	ClearCase	ClearQuest	Cloudscape
Cube Views	DB2	developerWorks	DRDA	IMS	IMS/ESA
Informix	Lotus	Lotus Workflow	MQSeries	OmniFind	
Rational	Redbooks	Red Brick	RequisitePro	System i	
System z	*Tivoli*	*WebSphere*	*Workplace*	*System p*	

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.
IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency which is now part of the Office of Government Commerce.
Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.
Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.
Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.
ITIL is a registered trademark, and a registered community trademark of The Minister for the Cabinet Office, and is registered in the U.S. Patent and Trademark Office.
UNIX is a registered trademark of The Open Group in the United States and other countries.
Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.
Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.