Hyperledger Composer basics, Part 2: Refine and deploy your blockchain network

Install development tools, unit test, and deploy your network to the IBM Cloud

J Steven Perry

November 30, 2017

This tutorial builds on the sample blockchain network introduced in Part 1 and takes you deeper into the Hyperledger Composer suite of developer tools. You'll model an IoT GPS sensor in a shipping container by adding GPS readings to the Shipment asset, and modify the smart contract (chaincode) to send an alert when the Shipment reaches its destination port.

This tutorial builds on **Part 1**, where you learned how to model and test a simple business network in a local version of the Hyperledger Composer Playground. Now let's go deeper into Hyperledger Composer and ultimately import your network model to the Online Playground on the IBM Cloud.

(And when you complete this Part 2 tutorial, then **Part 3** wraps everything up by showing you how to install Hyperledger Fabric on your computer, deploy your business network to your local instance, and interact with the sample network blockchain application.)

You'll first need to install a few developer tools; the video in this tutorial will guide you through the installation. Then you'll make changes to the sample Perishable Goods network that you worked with in Part 1. Specifically, you'll model an IoT GPS sensor in the shipping container by adding GPS readings to the Shipment asset, and modify the smart contract (chaincode) to send an alert when the Shipment reaches its destination port.

You'll also learn how to unit test your blockchain networks by using a Behavior-Driven Development (BDD) tool called Cucumber. You'll use a Cucumber feature file to test the smart contract's logic.

Finally, you'll import the model into the Online Playground hosted on the IBM Cloud, where you can interact with the model and submit transactions through the Playground UI, just like you did in Part 1. Because the Online Playground is on the IBM Cloud, no installation is needed to use it.

Prerequisites

Beyond these few prerequisites, the next section will walk you through the developer tools you need to install.

- Be sure you've worked through Part 1 of this series
- Docker 17.03 or higher
- Web browser

1. Set up your environment

At the time of this writing, Composer is supported only on Ubuntu Linux and MacOS, but Windows support is in the works.

For most of the tools you need to install, the installation instructions for Ubuntu Linux and MacOS are virtually the same, and I'll call out the few differences as we go. (To watch how to set up your computer, see the video.)

You gotta get git

Git is one of the most popular source code repository management tools today. To follow along with the tutorial, you'll need to install git. On Ubuntu, installing git is a snap: `sudo apt-get install git` and you're done. If you run MacOS, git will be installed with the command-line tools for MacOS when you install Node.js in the next section.

1a. Install Node.js

The easiest way to install Node.js is to use nvm, which stands for Node Version Manager. As the name implies, nvm is used to manage the version of Node that is installed on your computer. To install nvm, follow the instructions that match your platform.

Install nvm on Ubuntu Linux

The Hyperledger Composer team provides a script to install nvm. To install nvm, execute the following:

```
curl -O https://hyperledger.github.io/composer/prereqs-ubuntu.sh
chmod u+x prereqs-ubuntu.sh
./prereqs-ubuntu.sh
```

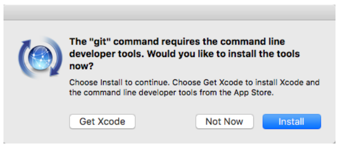
You'll be prompted for your password (make sure you are a sudoer on your system). For more information, visit Installing and developing with Hyperledger Composer.

When the script completes, you're ready to install Node.js. Skip to the "Use nvm to install Node.js" section now.

Install nvm on MacOS

MacOS provides its own version of many popular command-line tools like git, make, and svn. To see if you have the xcode command-line tools installed, open a terminal window, type `git`, and press Enter.

Figure 1. If you see this message, you need to install the command-line tools


The "git" command requires the command line developer tools. Would you like to install the tools now?

Choose Install to continue. Choose Get Xcode to install Xcode and the command line developer tools from the App Store.

If you see a message that looks like Figure 1, click the **Install** button to install the command-line tools. The installation takes less than 5 minutes.

Run this command from the terminal:

```
                curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.33.6/install.sh | bash
```

Note: Check the README on the nvm GitHub repo to make sure you are installing the latest version (0.33.6 at the time of this writing).

The script makes changes to your Bash shell settings, so you need to exit the terminal window, and open a new one to load the new settings.

To verify that nvm is installed correctly, execute the `nvm --version` command from the terminal window. You'll see output like this:

```
                $ nvm --version
                0.33.6
```

Now you're ready to install Node.js using nvm.

Use nvm to install Node.js

To install Node.js, go to the command line (Ubuntu) or open a terminal window (MacOS) to run the `nvm install --lts` command to install the LTS version of Node.js:

```
nvm install --lts
```

You'll see output like this:

```
$ nvm install --lts
Installing latest LTS version.
Downloading and installing node v8.9.0...
Downloading https://nodejs.org/dist/v8.9.0/node-v8.9.0-darwin-x64.tar.gz...
######################################################################## 100.0%
Computing checksum with shasum -a 256
Checksums matched!
.
.
(LOTS MORE OUTPUT FROM THE COMPILER)
.
.
Now using node v8.9.0 (npm v5.5.1)
```

The installation can take several minutes while the code is compiled for your platform (this is the case for MacOS, at least), so be patient.

Finally, verify that Node.js is installed:

```
$ node -v
v8.9.0
```

In this case, I have installed Node.js 8.9.0, which is the latest LTS version at the time of this writing.

1b. Install Composer command-line interface (CLI)

You use the command-line interface (CLI) to create, deploy, and update business networks, and perform other functions related to your blockchain networks.

To install the Composer CLI, go to the command line (Ubuntu) or open terminal window (MacOS), and enter this command:

```
npm install -g composer-cli
```

npm stands for Node Package Manager and was installed when nvm installed Node.js. Normally when you install a Node.js package through npm, it is available only within the directory tree where you installed it. Specifying the `-g` option tells npm to install the package globally, which makes it available to any Node.js project on your computer.

Verify that composer-cli was installed correctly. Run the `composer -v` from the command line (Ubuntu) or terminal window (MacOS), and the version number as output:

```
$ composer -v
v0.15.0
```

1c. Install VSCode

VSCode is an open source editor from Microsoft. The source code is freely available for download from Microsoft's VSCode GitHub repo.

You don't have to install VSCode to do Hyperledger Composer development, but we recommend it, and VSCode is actually pretty nice. There is a Hyperledger Composer extension for VSCode, which you can easily install and enable. It provides smooth integration with Git, along with syntax highlighting for Composer business network model files.

Install VSCode on Ubuntu Linux

To install on Ubuntu (or other Debian-based Linux), choose one of these options:

- `.deb`— Debian package
- `.rpm`—RPM Package Manager (originally called Red Hat Package Manager)
- `.tar.gz`— Tarball

Pick the method you like best, and follow the detailed installation instructions on the VSCode website.

Install VSCode on MacOS

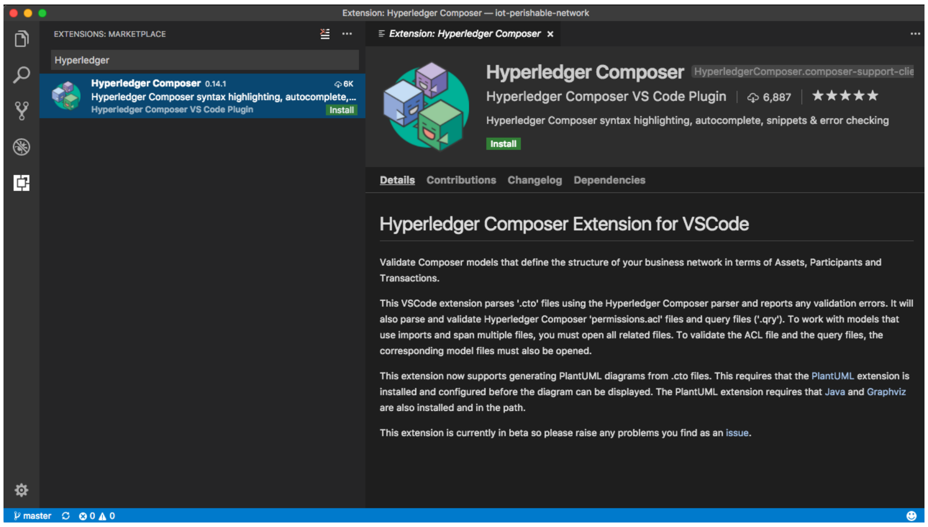
To install on MacOS, click the **Download for Mac** button, and a zip file containing the VSCode app will be downloaded to your Mac. See the VSCode website for detailed instructions on how to install and run VSCode.

1d. Install the Hyperledger Composer extension

To take full advantage of syntax highlighting when editing Hyperledger Composer files, be sure to install the Composer extension for VSCode. Start VSCode, and click the Extensions icon on the left side of the UI (or press **Cmd + X** on your Mac) to open the Extensions editor.

Type `Hyperledger` in the search field, and you will see the Hyperledger Composer extension in the list just below the Search field, as shown in Figure 2. Click the **Install** button, and when that finishes, restart VSCode to activate the extension.

Figure 2. VSCode Extension Marketplace


Extension: Hyperledger Composer — iot-perishable-network
EXTENSIONS: MARKETPLACE
Hyperledger
Hyperledger Composer 0.14.1
6K
Hyperledger Composer syntax highlighting, autocomplete,...
Hyperledger Composer VS Code Plugin
Install
Extension: Hyperledger Composer
Hyperledger Composer
HyperledgerComposer.composer-support-clie
Hyperledger Composer VS Code Plugin
6,887
★★★★★
Hyperledger Composer syntax highlighting, autocomplete, snippets & error checking
Install
Details
Contributions
Changelog
Dependencies
master
0 0

Hyperledger Composer Extension for VSCode

Validate Composer models that define the structure of your business network in terms of Assets, Participants and Transactions.

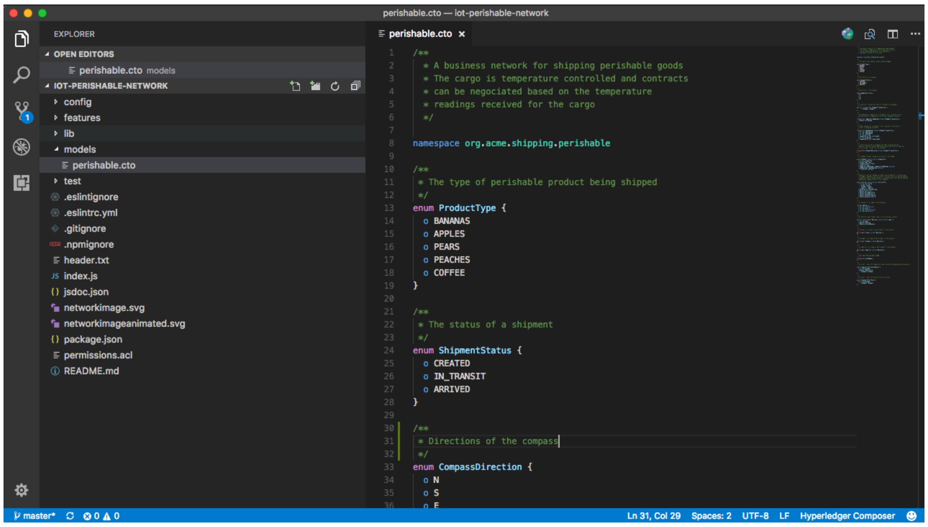
This VSCode extension parses '.cto' files using the Hyperledger Composer parser and reports any validation errors. It will also parse and validate Hyperledger Composer 'permissions.acl' files and query files ('.qry'). To work with models that use imports and span multiple files, you must open all related files. To validate the ACL file and the query files, the corresponding model files must also be opened.

This extension now supports generating PlantUML diagrams from .cto files. This requires that the PlantUML extension is installed and configured before the diagram can be displayed. The PlantUML extension requires that Java and Graphviz are also installed and in the path.

This extension is currently in beta so please raise any problems you find as an issue.

Now whenever you use VSCode to edit a Hyperledger Composer project file, its syntax will highlight automatically. Figure 3 shows the `perishable.cto` model file opened in the VSCode editor window. Notice the syntax highlighting of the `namespace` and `enum` keywords.

Figure 3. Perishable Goods network in VSCode



```
/**
 * A business network for shipping perishable goods
 * The cargo is temperature controlled and contracts
 * can be negociated based on the temperature
 * readings received for the cargo
 */

namespace org.acme.shipping.perishable

/**
* The type of perishable product being shipped
*/
enum ProductType {
  o BANANAS
  o APPLES
  o PEARS
  o PEACHES
  o COFFEE
}

/**
* The status of a shipment
*/
enum ShipmentStatus {
  o CREATED
  o IN_TRANSIT
  o ARRIVED
}

/**
* Directions of the compass
*/
enum CompassDirection {
  o N
  o S
  o E
```

Video: Set up your environment

This video shows you how to set up your environment.

To view this video, **Set up your environment** , please access the online version of the article. If this article is in the developerWorks archives, the video is no longer accessible.

2. Run the build

Now that you have tools, it's time to put them to work. You will clone the `perishable-network` GitHub repository I have provided, then build and unit test the code using the Node.js tools you just installed.

Choose a location on your computer where you will work with Hyperledger Composer and the network models. For example, I use `~/HyperledgerComposer` as my Composer root directory, and I set an environment variable in the Bash shell I'm using:

```
$ export COMPOSER_ROOT=~/HyperledgerComposer
```

To keep this tutorial location-agnostic, I'll refer to this directory as `$COMPOSER_ROOT`. When you see `$COMPOSER_ROOT` in the examples that follow, I am referring to the location you have chosen. I suggest you set `$COMPOSER_ROOT` to that location as I did above.

Go to a command line (Ubuntu) or open a terminal window (MacOS), navigate to your `$COMPOSER_ROOT` directory, and enter this command: `git clone https://github.com/makotogo/developerWorks.git` as shown below.

```
$ cd $COMPOSER_ROOT
$ pwd
/Users/sperry/HyperledgerComposer
$ git clone https://github.com/makotogo/developerWorks.git
Cloning into 'developerWorks'...
remote: Counting objects: 364, done.
remote: Compressing objects: 100% (45/45), done.
remote: Total 364 (delta 33), reused 78 (delta 25), pack-reused 277
Receiving objects: 100% (364/364), 146.69 KiB | 754.00 KiB/s, done.
Resolving deltas: 100% (162/162), done.
```

Now you have the code. It's time to build and test it.

You use the Node.js package manager (npm) to run a build, and then execute the unit tests I have provided. Execute these commands:

```
cd $COMPOSER_ROOT/developerWorks/perishable-network
npm install && npm test
```

Here's what is going on: First you navigate to the directory where the perishable-network code lives. Then you run `npm install`, which sets up the local Node.js environment (that is, local to the `perishable-network`). Then you run `npm test`, which executes the unit tests that are part of the project (see `package.json`).

You'll see lots of output, but the last of it looks like this, indicating the tests were successful:

```
    .
    .
  Perishable Shipping Network
    #shipment
Adding temperature 4.5 to shipment SHIP_001
Received at: Wed Nov 01 2017 10:58:12 GMT-0500 (CDT)
Contract arrivalDateTime: Thu Nov 02 2017 10:58:12 GMT-0500 (CDT)
Lowest temp reading: 4.5
Highest temp reading: 4.5
Payout: 2500
Grower: farmer@email.com new balance: 2500
Importer: supermarket@email.com new balance: -2500
      # should receive base price for a shipment within temperature range (90ms)
Adding temperature 1 to shipment SHIP_001
Received at: Wed Nov 01 2017 10:58:12 GMT-0500 (CDT)
Contract arrivalDateTime: Thu Nov 02 2017 10:58:12 GMT-0500 (CDT)
Lowest temp reading: 1
Highest temp reading: 4.5
Min temp penalty: 0.2
Payout: 1500
Grower: farmer@email.com new balance: 4000
Importer: supermarket@email.com new balance: -4000
      # should apply penalty for min temperature violation (81ms)
Adding temperature 11 to shipment SHIP_001
Received at: Wed Nov 01 2017 10:58:12 GMT-0500 (CDT)
Contract arrivalDateTime: Thu Nov 02 2017 10:58:12 GMT-0500 (CDT)
Lowest temp reading: 1
Highest temp reading: 11
Min temp penalty: 0.2
Max temp penalty: 0.30000000000000004
Payout: 999.9999999999998
Grower: farmer@email.com new balance: 5000
Importer: supermarket@email.com new balance: -5000
      # should apply penalty for max temperature violation (74ms)
```

```
                3 passing (1s)
```

3. Refine the Perishable Goods business network

The Perishable Goods network models a business network that includes: a Grower, a Shipper, and an Importer. The specifics are spelled out in the README.md file. The agreement among the various participants in this network is modeled using the CTO modeling language, and enforced by the chaincode (smart contract) written in JavaScript.

In this theoretical exercise in this section, an IoT GPS sensor has been added to the cargo containers to provide the container ship's location. You have been asked to add readings taken by this sensor to the network model, and send an event when the ship reaches its destination.

To add the GPS sensor to the network model, you'll need to make changes to the business model, and write additional unit tests. Instead of writing the unit tests in Mocha, I'll show you how to write them using a tool called Cucumber that has a more human-readable syntax and is just as powerful.

3a. Modify the network definition

When you have made the changes in this section by pasting in the code I've provided, your solution should look just like the one in the `developerWorks/iot-perishable-network` directory. Feel free to use this model as a reference.

To add the GPS sensor, you need to make a few changes to your model. Start VSCode, and open the root directory of the Perishable Goods network (`$COMPOSER_ROOT)/developerWorks/perishable-network`). Open the model file, called `perishable.cto`, located in the `models` directory.

Add a new `enum` to represent the major locations on the compass just below the `enum ShipmentStatus`:

```
                /**
                 * Directions of the compass
                 */
                enum CompassDirection {
                  o N
                  o S
                  o E
                  o W
                }
```

The directions are `N` for North, `S` for South, and so on. It is important to constrain the data entered for a set of GPS coordinates, and this `enum` is used to constrain the values that can be entered into the model to ensure they are valid.

Each time a GPS reading is taken, it is recorded to the blockchain as a transaction, which means you need to add a transaction to the model for it. Just below the `TemperatureReading` transaction, add a new `GpsReading` transaction:

Listing 1. Transaction to handle recording GPS readings in the blockchain

```
/**
 * A GPS reading for a shipment. E.g. received from a device
 * within a shipping container
 */
transaction GpsReading extends ShipmentTransaction {
  o String readingTime
  o String readingDate
  o String latitude
  o CompassDirection latitudeDir
  o String longitude
  o CompassDirection longitudeDir
}
```

A GPS reading requires several parameters that include when the reading was taken, along with the latitude and longitude. This information is provided as parameters to the transaction.

Next, in order for the transaction to store the GPS reading in the blockchain, the information needs to be a part of a blockchain asset. Because a GPS reading taken from the shipping container is conceptually part of a shipment, it's a natural fit to add the readings to the `Shipment` asset (just like the `TemperatureReading`s). Add the highlighted line (line 7) below to the `Shipment` asset:

```
asset Shipment identified by shipmentId {
  o String shipmentId
  o ProductType type
  o ShipmentStatus status
  o Long unitCount
  o TemperatureReading[] temperatureReadings optional
  o GpsReading[] gpsReadings optional
  --> Contract contract
}
```

Finally, add two events to the model: one when a temperature threshold is violated, and another when the container ship has reached its destination port:

Listing 2. New events - when temperature in the container exceeds the contractual tolerance, and when the container ship arrives in port

```
/**
 * An event - when the temperature goes outside the agreed-upon boundaries
 */
event TemperatureThresholdEvent {
  o String message
  o Double temperature
  --> Shipment shipment
}

/**
 * An event - when the ship arrives at the port
 */
event ShipmentInPortEvent {
  o String message
  --> Shipment shipment
}
```

3b. Add chaincode

You modeled the GPS sensor and the transaction to add GPS readings to the model. Now you need to write the JavaScript chaincode that handles updating the blockchain. Open `lib/logic.js`. You'll need to make several changes to this file.

First, add code to the `temperatureReading` function, which handles the `TemperatureReading` transaction. Replace the entire body of the method with the one below (the lines that are added are highlighted just for reference):

```
                function temperatureReading(temperatureReading) {

                    var shipment = temperatureReading.shipment;
                    var NS = "org.acme.shipping.perishable";
                    var contract = shipment.contract;
                    var factory = getFactory();

                    console.log('Adding temperature ' + temperatureReading.centigrade + ' to shipment ' +
shipment.$identifier);

                    if (shipment.temperatureReadings) {
                        shipment.temperatureReadings.push(temperatureReading);
                    } else {
                        shipment.temperatureReadings = [temperatureReading];
                    }

                    if (temperatureReading.centigrade < contract.minTemperature ||
                        temperatureReading.centigrade > contract.maxTemperature) {
                        var temperatureEvent = factory.newEvent(NS, 'TemperatureThresholdEvent');
                        temperatureEvent.shipment = shipment;
                        temperatureEvent.temperature = temperatureReading.centigrade;
                        temperatureEvent.message = 'Temperature threshold violated! Emitting TemperatureEvent
for shipment: ' + shipment.$identifier;
                        emit(temperatureEvent);
                    }

                    return getAssetRegistry(NS + '.Shipment')
                        .then(function (shipmentRegistry) {
                            // add the temp reading to the shipment
                            return shipmentRegistry.update(shipment);
                        });
                }
```

This code checks the current temperature reading against the contractual stipulations, and if either the min or max temperature limits have been exceeded, a `TemperatureThresholdEvent` event is emitted.

Next, add a new function to handle the `GpsReading` transaction.
Note: it is important that you add the comment block as well. It contains two important annotations (`@param` and `@transaction`) that you will need.

```
                /**
                 * A GPS reading has been received for a shipment
                 * @param {org.acme.shipping.perishable.GpsReading} gpsReading - the GpsReading transaction
                 * @transaction
                 */
                function gpsReading(gpsReading) {

                    var factory = getFactory();
                    var NS = "org.acme.shipping.perishable";
                    var shipment = gpsReading.shipment;
                    var PORT_OF_NEW_YORK = '/LAT:40.6840N/LONG:74.0062W';

                    var latLong = '/LAT:' + gpsReading.latitude + gpsReading.latitudeDir + '/LONG:' +
                        gpsReading.longitude + gpsReading.longitudeDir;

                    if (shipment.gpsReadings) {
                        shipment.gpsReadings.push(gpsReading);
                    } else {
                        shipment.gpsReadings = [gpsReading];
                    }

                    if (latLong == PORT_OF_NEW_YORK) {
                        var shipmentInPortEvent = factory.newEvent(NS, 'ShipmentInPortEvent');
                        shipmentInPortEvent.shipment = shipment;
                        var message = 'Shipment has reached the destination port of ' + PORT_OF_NEW_YORK;
                        shipmentInPortEvent.message = message;
                        emit(shipmentInPortEvent);
                    }

                    return getAssetRegistry(NS + '.Shipment')
                    .then(function (shipmentRegistry) {
                        // add the temp reading to the shipment
                        return shipmentRegistry.update(shipment);
                    });
                }
```

The chaincode for the transaction stores this GPS reading in the array of `GpsReading`s in the `Shipment` asset. It then checks to see if this GPS reading corresponds to the destination port, and if so, emits a `ShipmentInPort` event. Finally, the blockchain is updated with the current state of the `Shipment`.

3c. Add Cucumber feature tests

Let's sum up. You added a transaction to the model, and two new events. Now it's time to unit test the changes to make sure they work. The Hyperledger Composer team recommends using Cucumber to unit test Composer business models.

Create a new file in the `features` folder called `iot-perishable.feature` and open it in VSCode. I'll explain briefly each of the sections you need to add. In the "Unit testing with Cucumber" section, I'll explain Cucumber more thoroughly. But let's get something working first.

First, you tell Cucumber about the feature you want to test, along with any background (setup) that needs to be performed before each unit test. Add the following code to your empty `iot-perishable.feature` file.

Listing 3. Cucumber feature and background

```
                Feature: IoT Perishable Network

                    Background:
                        Given I have deployed the business network definition ..
                        And I have added the following participants
                        """
                        [
                        {"$class":"org.acme.shipping.perishable.Grower", "email":"grower@email.com",
 "address":{"$class":"org.acme.shipping.perishable.Address", "country":"USA"}, "accountBalance":0},
                        {"$class":"org.acme.shipping.perishable.Importer", "email":"supermarket@email.com",
 "address":{"$class":"org.acme.shipping.perishable.Address", "country":"UK"}, "accountBalance":0},
                        {"$class":"org.acme.shipping.perishable.Shipper", "email":"shipper@email.com",
 "address":{"$class":"org.acme.shipping.perishable.Address", "country":"Panama"}, "accountBalance":0}
                        ]
                        """
                        And I have added the following asset of type org.acme.shipping.perishable.Contract
                            | contractId | grower           | shipper               | importer           |
 arrivalDateTime  | unitPrice | minTemperature | maxTemperature | minPenaltyFactor | maxPenaltyFactor |
                            | CON_001    | grower@email.com | supermarket@email.com | supermarket@email.com |
 10/26/2018 00:00 | 0.5       | 2              | 10             | 0.2              | 0.1              |

                        And I have added the following asset of type org.acme.shipping.perishable.Shipment
                            | shipmentId | type    | status     | unitCount | contract |
                            | SHIP_001   | BANANAS | IN_TRANSIT | 5000      | CON_001  |
                        When I submit the following transactions of type
 org.acme.shipping.perishable.TemperatureReading
                            | shipment | centigrade |
                            | SHIP_001 | 4          |
                            | SHIP_001 | 5          |
                            | SHIP_001 | 10         |
```

Now you need to add some unit tests, called Scenarios. Add the Scenario below just after the Background block in the `iot-perishable.feature` file.

Listing 4. Scenario: when temperature range is within the agreed-upon boundaries

```
        Scenario: When the temperature range is within the agreed-upon boundaries
            When I submit the following transaction of type org.acme.shipping.perishable.ShipmentReceived
                | shipment |
                | SHIP_001 |

            Then I should have the following participants
            """
            [
            {"$class":"org.acme.shipping.perishable.Grower", "email":"grower@email.com", "address":
{"$class":"org.acme.shipping.perishable.Address", "country":"USA"}, "accountBalance":2500},
            {"$class":"org.acme.shipping.perishable.Importer", "email":"supermarket@email.com", "address":
{"$class":"org.acme.shipping.perishable.Address", "country":"UK"}, "accountBalance":-2500},
            {"$class":"org.acme.shipping.perishable.Shipper", "email":"shipper@email.com", "address":
{"$class":"org.acme.shipping.perishable.Address", "country":"Panama"}, "accountBalance":0}
            ]
            """
```

This Scenario makes sure that the Grower is paid the full amount when the temperature in the cargo container is within the agreed-upon limits.

Now add a scenario where the min temperature threshold of 2 degrees Celsius is violated by 2 degrees.

Listing 5. Scenario: When the low/min temperature threshold is breached by 2 degrees C

```
                Scenario: When the low/min temperature threshold is breached by 2 degrees C
                    Given I submit the following transaction of type
 org.acme.shipping.perishable.TemperatureReading
                        | shipment | centigrade |
                        | SHIP_001 | 0          |

                    When I submit the following transaction of type
 org.acme.shipping.perishable.ShipmentReceived
                        | shipment |
                        | SHIP_001 |

                    Then I should have the following participants
                    """
                    [
                    {"$class":"org.acme.shipping.perishable.Grower", "email":"grower@email.com", "address":
{"$class":"org.acme.shipping.perishable.Address", "country":"USA"}, "accountBalance":500},
                    {"$class":"org.acme.shipping.perishable.Importer", "email":"supermarket@email.com",
 "address":{"$class":"org.acme.shipping.perishable.Address", "country":"UK"}, "accountBalance":-500},
                    {"$class":"org.acme.shipping.perishable.Shipper", "email":"shipper@email.com", "address":
{"$class":"org.acme.shipping.perishable.Address", "country":"Panama"}, "accountBalance":0}
                    ]
                    """
```

In this Scenario, the min temperature drops to zero degrees Celsius, and the Grower is penalized for each degree Celsius below the threshold.

Now add a Scenario where the max temperature threshold is exceeded by 2 degrees Celsius.

Listing 6. Scenario: When the hi/max temperature threshold is breached by 2 degrees C

```
                Scenario: When the hi/max temperature threshold is breached by 2 degrees C
                    Given I submit the following transaction of type
 org.acme.shipping.perishable.TemperatureReading
                        | shipment | centigrade |
                        | SHIP_001 | 12          |

                    When I submit the following transaction of type
 org.acme.shipping.perishable.ShipmentReceived
                        | shipment |
                        | SHIP_001 |

                    Then I should have the following participants
                    """
                    [
                    {"$class":"org.acme.shipping.perishable.Grower", "email":"grower@email.com", "address":
{"$class":"org.acme.shipping.perishable.Address", "country":"USA"}, "accountBalance":1500},
                    {"$class":"org.acme.shipping.perishable.Importer", "email":"supermarket@email.com",
 "address":{"$class":"org.acme.shipping.perishable.Address", "country":"UK"}, "accountBalance":-1500},
                    {"$class":"org.acme.shipping.perishable.Shipper", "email":"shipper@email.com", "address":
{"$class":"org.acme.shipping.perishable.Address", "country":"Panama"}, "accountBalance":0}
                    ]
                    """
```

In this Scenario, the max temperature goes above 10 degrees Celsius, and the Grower is penalized by each degree Celsius above the threshold.

Finally, add three scenarios (two `TemperatureThresholdEvent`s and one `ShipmentInPortEvent`) to the model as shown in Listing 2.

Listing 7. Scenario: Events

```
                Scenario: Test TemperatureThresholdEvent is emitted when the min temperature threshold is
violated
                    When I submit the following transactions of type
org.acme.shipping.perishable.TemperatureReading
                        | shipment | centigrade |
                        | SHIP_001 | 0          |

                    Then I should have received the following event of type
org.acme.shipping.perishable.TemperatureThresholdEvent
                        | message                                                                  |
temperature | shipment |
                        | Temperature threshold violated! Emitting TemperatureEvent for shipment: SHIP_001 |
0           | SHIP_001 |


                Scenario: Test TemperatureThresholdEvent is emitted when the max temperature threshold is
violated
                    When I submit the following transactions of type
org.acme.shipping.perishable.TemperatureReading
                        | shipment | centigrade |
                        | SHIP_001 | 11         |

                    Then I should have received the following event of type
org.acme.shipping.perishable.TemperatureThresholdEvent
                        | message                                                                  |
temperature | shipment |
                        | Temperature threshold violated! Emitting TemperatureEvent for shipment: SHIP_001 |
11          | SHIP_001 |


                Scenario: Test ShipmentInPortEvent is emitted when GpsReading indicates arrival at
destination port
                    When I submit the following transaction of type org.acme.shipping.perishable.GpsReading
                        | shipment | readingTime | readingDate | latitude | latitudeDir | longitude |
longitudeDir |
                        | SHIP_001 | 120000      | 20171025    | 40.6840  | N           | 74.0062   | W
           |

                    Then I should have received the following event of type
org.acme.shipping.perishable.ShipmentInPortEvent
                        | message                                                                   |
shipment |
                        | Shipment has reached the destination port of /LAT:40.6840N/LONG:74.0062W | SHIP_001
|
```

Finally, you need to modify `package.json` so that it looks like the one below. I've highlighted the lines you need to modify (line 18) and add (lines 41 and 42).

```
                {
                  "engines": {
                    "composer": "^0.15.0"
                  },
                  "name": "perishable-network",
                  "version": "0.1.11",
                  "description": "Shipping Perishable Goods Business Network",
                  "networkImage": "https://github.com/makotogo/developerWorks/perishable-network/
networkimage.svg",
                  "networkImageanimated": "https://github.com/makotogo/developerWorks/perishable-network/
networkimageanimated.svg",
                  "scripts": {
                    "prepublish": "mkdirp ./dist && composer archive create  --sourceType dir --sourceName .
 -a ./dist/perishable-network.bna",
                    "pretest": "npm run lint",
                    "lint": "eslint .",
                    "postlint": "npm run licchk",
                    "licchk": "license-check",
                    "postlicchk": "npm run doc",
                    "doc": "jsdoc --pedantic --recurse -c jsdoc.json",
                    "test": "mocha -t 0 --recursive && cucumber-js",
                    "deploy": "./scripts/deploy.sh"
                  },
                  "repository": {
                    "type": "git",
                    "url": "https://github.com/makotogo/developerWorks.git"
                  },
                  "keywords": [
                    "shipping",
                    "goods",
                    "perishable",
                    "composer",
                    "composer-network",
                    "iot"
                  ],
                  "author": "Hyperledger Composer",
                  "license": "Apache-2.0",
                  "devDependencies": {
                    "browserfs": "^1.2.0",
                    "chai": "^3.5.0",
                    "composer-admin": "^0.14.0-0",
                    "composer-cli": "^0.14.0-0",
                    "composer-client": "^0.14.0-0",
                    "composer-connector-embedded": "^0.14.0-0",
                    "composer-cucumber-steps": "^0.14.0-0",
                    "cucumber": "^2.2.0",
                    "eslint": "^3.6.1",
                    "istanbul": "^0.4.5",
                    "jsdoc": "^3.4.1",
                    "license-check": "^1.1.5",
                    "mkdirp": "^0.5.1",
                    "mocha": "^3.2.0",
                    "moment": "^2.17.1"
                  },
                  "license-check-config": {
                    "src": [
                      "**/*.js",
                      "!./coverage/**/*",
                      "!./node_modules/**/*",
                      "!./out/**/*",
                      "!./scripts/**/*"
                    ],
                    "path": "header.txt",
                    "blocking": true,
                    "logInfo": false,
                    "logError": true
                  }
                }
```

Notice you added two new Node modules (lines 41 and 42 above) to `package.json`. To install them, go to the command line (Ubuntu) or open a terminal window (MacOS) and run the `npm install` command.

3d. Run the unit tests

Now run the unit tests by executing `npm test` from the command line. There will be lots of output, but this time, in addition to the Mocha tests, you will see one block of output for each Cucumber feature Scenario. The Cucumber feature tests start out like this (I'm just showing the first one for space considerations).

```
                Feature: IoT Perishable Network

                  Scenario: When the temperature range is within the agreed-upon boundaries
                  # Given I have deployed the business network definition ..
                  # And I have added the following participants
                      """
                      [
                      {"$class":"org.acme.shipping.perishable.Grower", "email":"grower@email.com", "address":
{"$class":"org.acme.shipping.perishable.Address", "country":"USA"}, "accountBalance":0},
                      {"$class":"org.acme.shipping.perishable.Importer", "email":"supermarket@email.com",
 "address":{"$class":"org.acme.shipping.perishable.Address", "country":"UK"}, "accountBalance":0},
                      {"$class":"org.acme.shipping.perishable.Shipper", "email":"shipper@email.com",
 "address":{"$class":"org.acme.shipping.perishable.Address", "country":"Panama"}, "accountBalance":0}
                      ]
                      """
                  # And I have added the following asset of type org.acme.shipping.perishable.Contract
                      | contractId | grower           | shipper                | importer              |
 arrivalDateTime  | unitPrice | minTemperature | maxTemperature | minPenaltyFactor | maxPenaltyFactor |
                      | CON_001    | grower@email.com | supermarket@email.com | supermarket@email.com |
 10/26/2018 00:00 | 0.5       | 2              | 10             | 0.2              | 0.1              |
                  # And I have added the following asset of type org.acme.shipping.perishable.Shipment
                      | shipmentId | type    | status     | unitCount | contract |
                      | SHIP_001   | BANANAS | IN_TRANSIT | 5000      | CON_001  |
                  # When I submit the following transactions of type
 org.acme.shipping.perishable.TemperatureReading
                      | shipment | centigrade |
                      | SHIP_001 | 4          |
                      | SHIP_001 | 5          |
                      | SHIP_001 | 10         |
                  # When I submit the following transaction of type
 org.acme.shipping.perishable.ShipmentReceived
                      | shipment |
                      | SHIP_001 |
                  # Then I should have the following participants
                      """
                      [
                      {"$class":"org.acme.shipping.perishable.Grower", "email":"grower@email.com", "address":
{"$class":"org.acme.shipping.perishable.Address", "country":"USA"}, "accountBalance":2500},
                      {"$class":"org.acme.shipping.perishable.Importer", "email":"supermarket@email.com",
 "address":{"$class":"org.acme.shipping.perishable.Address", "country":"UK"}, "accountBalance":-2500},
                      {"$class":"org.acme.shipping.perishable.Shipper", "email":"shipper@email.com",
 "address":{"$class":"org.acme.shipping.perishable.Address", "country":"Panama"}, "accountBalance":0}
                      ]
                      """
```

The last few lines of output look like this, indicating the unit tests pass.

```
                    .
                    .
                  # And I have added the following asset of type org.acme.shipping.perishable.Shipment
                      | shipmentId | type    | status     | unitCount | contract |
                      | SHIP_001   | BANANAS | IN_TRANSIT | 5000      | CON_001  |
                  # When I submit the following transactions of type
 org.acme.shipping.perishable.TemperatureReading
                      | shipment | centigrade |
                      | SHIP_001 | 4          |
                      | SHIP_001 | 5          |
                      | SHIP_001 | 10         |
                  # When I submit the following transaction of type org.acme.shipping.perishable.GpsReading
                      | shipment | readingTime | readingDate | latitude | latitudeDir | longitude |
longitudeDir |
                      | SHIP_001 | 120000      | 20171025    | 40.6840  | N           | 74.0062   | W
           |
                  # Then I should have received the following event of type
org.acme.shipping.perishable.ShipmentInPortEvent
                      | message                                                        | shipment |
                      | Shipment has reached the destination port of /LAT:40.6840N/LONG:74.0062W | SHIP_001 |

                6 scenarios (6 passed)
                44 steps (44 passed)
                0m02.788s
                $
```

A closer look at unit testing with Cucumber

Okay, I know I threw a lot at you in the previous section. So let me take a minute and explain it in more detail, starting with an overview of Cucumber.

Cucumber overview

Cucumber is a unit testing tool designed for Behavior Driven Development (BDD), a style of development related to Test-Driven Development (TDD), but more focused on the behavior of the application, rather than just the functional correctness of the code. It's a subtle yet important difference.

You write Cucumber test scenarios in a language called Gherkin. You've already seen it in the previous section. The syntax is fairly simple:

```
                Feature : The application feature I want to test
                  Background:
                    Given some pre-condition that applies to each test

                  Scenario: The test I want to run
                      Given something
                      And some other thing
                      When I do XYZ
                      Then ABC is the expected outcome
```

A *step definition* is a small piece of code that links a **pattern** in the Gherkin plain language text to **actual code** that runs the test. You are responsible for writing the step definitions, or your tests will not run.

When Cucumber runs across one of the following keywords: `When`, `Given`, `Then`, `And`, or `But`, it looks for a step definition to associate the text with that follows the keyword. By the way, Cucumber makes no distinction between the keywords; they are all just steps to Cucumber. In other words, to Cucumber, `When ABC` and `Given ABC` both mean "locate and execute a step with pattern ABC."

For example, when Cucumber sees `Given I have added the following participants`, it looks for a step whose pattern matches `I have added the following`, and if it finds one, it executes it. If not, you must provide it.

The good news is the Hyperledger Composer team has already provided steps for most of the common usages of the Composer client API. This means that you can write Cucumber tests in the Gherkin language just like those you've already seen without writing any code yourself! The code is contained in the JavaScript library `composer-cucumber-steps` that you added to `package.json` in the previous section.

Working with Cucumber

So let's walk through one of the scenarios. I'll describe what Cucumber is doing and show you the actual JavaScript code that runs for the steps from the `composer-cucumber-steps` Node.js module.

Refer to Listing 8. In this scenario, there are three keywords:

- **`Given`** I submit the following transaction of type org.acme.shipping.perishable.TemperatureReading
- **`When`** I submit the following transaction of type org.acme.shipping.perishable.ShipmentReceived
- **`Then`** I should have the following participants

Cucumber makes no distinction between the three keywords (they are just steps), but the patterns following the keywords must match one (and only one) step. Otherwise, the test will not run, because there is no code to run it.

In the first instance, Cucumber looks for a pattern (that is, a regular expression) that matches "I submit the following transaction of type org.acme.shipping.perishable.TemperatureReading" and finds one. The JavaScript function that runs when this step executes looks like this:

Listing 8. composer-cucumber-steps/lib/transactionsteps.js

```
                'use strict';

                module.exports = function () {

                    this.When(/^I submit the following transactions? of type ([.\w]+)\.(\w+)$/, function
 (namespace, name, table) {
                        return this.composer.submitTransactions(namespace, name, table);
                    });

                    this.When(/^I submit the following transactions?$/, function (docString) {
                        return this.composer.submitTransactions(null, null, docString);
                    });

                };
```

Wait. Where's the `this.Given()` function, right? The functions shown above all start with `this.When`. How does Cucumber know to invoke one of the functions above when your Gherkin contains `Given`? Remember, Cucumber doesn't distinguish between any of the step keywords (`Given, When, And`, and so forth). It only matches the pattern. In fact, if you code a Cucumber step definition with the same pattern for both `Given` and `When`, Cucumber will report a duplicate step definition error!

The pattern to match is the regular expression between the `/` characters in parentheses, including the capture groups (`([.\w]+)\.(\w+)`), which are used to parse the arguments that are passed to the function that gets invoked: *namespace* (`org.acme.shipping.perishable`), *name* (`TemperatureReading`), and *table* (I'll talk more about the table argument shortly).

The code that executes for the step that matches the pattern "I should have the following participants" matches this code:

```
                'use strict';

                module.exports = function () {

                    this.Given(/^I have added the following participants? of type ([.\w]+)\.(\w+)$/, function
(namespace, name, table) {
                        return this.composer.addParticipants(namespace, name, table);
                    });

                    this.Given(/^I have added the following participants?$/, function (docString) {
                        return this.composer.addParticipants(null, null, docString);
                    });

                    this.When(/^I add the following participants? of type ([.\w]+)\.(\w+)$/, function
(namespace, name, table) {
                        return this.composer.addParticipants(namespace, name, table);
                    });

                    this.When(/^I add the following participants?$/, function (docString) {
                        return this.composer.addParticipants(null, null, docString);
                    });

                    this.When(/^I update the following participants? of type ([.\w]+)\.(\w+)$/, function
(namespace, name, table) {
                        return this.composer.updateParticipants(namespace, name, table);
                    });

                    this.When(/^I update the following participants?$/, function (docString) {
                        return this.composer.updateParticipants(null, null, docString);
                    });

                    this.When(/^I remove the following participants? of type ([.\w]+)\.(\w+)$/, function
(namespace, name, table) {
                        return this.composer.removeParticipants(namespace, name, table);
                    });

                    this.When(/^I remove the following participants?$/, function (docString) {
                        return this.composer.removeParticipants(null, null, docString);
                    });

                    this.Then(/^I should have the following participants? of type ([.\w]+)\.(\w+)$/, function
(namespace, name, table) {
                        return this.composer.testParticipants(namespace, name, table);
                    });

                    this.Then(/^I should have the following participants?$/, function (docString) {
                        return this.composer.testParticipants(null, null, docString);
                    });

                    this.Then(/^I should not have the following participants? of type ([.\w]+)\.(\w+)$/,
function (namespace, name, table) {
                        return this.composer.testNoParticipants(namespace, name, table);
                    });

                    this.Then(/^I should not have the following participants?$/, function (docString) {
                        return this.composer.testNoParticipants(null, null, docString);
                    });

                };
```

As you can see, the code for steps related to Participants handles much more than just "I should have", such as "I remove", "I add", and lots more.

Complex objects in Gherkin

You may have noticed there were two ways to represent objects from the Perishable Goods network model in the Gherkin examples. First, like this:

```
                | contractId | grower            | shipper               | importer              |
arrivalDateTime  | unitPrice | minTemperature | maxTemperature | minPenaltyFactor | maxPenaltyFactor |
                | CON_001    | grower@email.com | supermarket@email.com | supermarket@email.com | 10/26/2018
00:00 | 0.5       | 2               | 10               | 0.2                | 0.1                |
```

Second, like this:

```
                """
                [
                {"$class":"org.acme.shipping.perishable.Grower", "email":"grower@email.com", "address":
{"$class":"org.acme.shipping.perishable.Address", "country":"USA"}, "accountBalance":1500},
                {"$class":"org.acme.shipping.perishable.Importer", "email":"supermarket@email.com",
 "address":{"$class":"org.acme.shipping.perishable.Address", "country":"UK"}, "accountBalance":-1500}
                ]
                """
```

The first type of object representation is called a *data table*, and it's a pipe-delimited tuple, where the first line contains the property names, followed by one or more lines that contain the property values. The data table does not support nested objects. When you use a data table to represent your objects, you have to tell the step the type of object that follows. The object in this example is a `org.acme.shipping.perishable.Contract` object, and it was specified on the same line as the step (see Listing 2). Use a data table when your objects do not contain other required objects from the model (an optional object can be omitted).

The second type uses type a *docstring*, which is a JSON representation of the object, and supports nested objects (for example, notice the nested `Address` object). When you use a docstring, you do not have to specify the object type in the step because you must do so as part of the docstring (see Listing 5).

Use docstring syntax when an object you are trying to represent contains a nested, required object, or when you want to provide data values for a nested optional object.

4. Deploy your modified network to the IBM Cloud

Now that you have modified the network and unit tested it, it's time to load it up into the Online Playground and interact with it on the IBM Cloud at https://composer-playground.mybluemix.net/.

In Part 1, I showed you how to work with Playground to define a network using the Perishable Goods boilerplate model, instantiate the model, and submit transactions against the in-memory blockchain. In this section, you'll use the Perishable Goods network Business Network Archive

(BNA) file created by the build (that is, `npm install`) to deploy the network using Playground's import feature. But first you need to make a few code changes and then run a build to create the BNA file.

4a. Make some code changes

Make sure the Perishable Goods network model that you refined earlier in this tutorial is open in VSCode. Now open `lib/logic.js` in the editor window. Add a console log message to the temperatureReading() function as shown in line 22:

Listing 9. logic.js —`temperatureReading()`

```
                function temperatureReading(temperatureReading) {

                    var shipment = temperatureReading.shipment;
                    var NS = 'org.acme.shipping.perishable';
                    var contract = shipment.contract;
                    var factory = getFactory();

                    console.log('Adding temperature ' + temperatureReading.centigrade + ' to shipment ' +
shipment.$identifier);

                    if (shipment.temperatureReadings) {
                        shipment.temperatureReadings.push(temperatureReading);
                    } else {
                        shipment.temperatureReadings = [temperatureReading];
                    }

                    if (temperatureReading.centigrade < contract.minTemperature ||
                        temperatureReading.centigrade > contract.maxTemperature) {
                        var temperatureEvent = factory.newEvent(NS, 'TemperatureThresholdEvent');
                        temperatureEvent.shipment = shipment;
                        temperatureEvent.temperature = temperatureReading.centigrade;
                        temperatureEvent.message = 'Temperature threshold violated! Emitting TemperatureEvent
for shipment: ' + shipment.$identifier;
                        console.log(message);
                        emit(temperatureEvent);
                    }
```

Now locate the `gpsReading()` function and add a console log message as shown in line 22:

```
                function gpsReading(gpsReading) {

                       var factory = getFactory();
                       var NS = "org.acme.shipping.perishable";
                       var shipment = gpsReading.shipment;
                       var PORT_OF_NEW_YORK = '/LAT:40.6840N/LONG:74.0062W';

                       var latLong = '/LAT:' + gpsReading.latitude + gpsReading.latitudeDir + '/LONG:' +
                           gpsReading.longitude + gpsReading.longitudeDir;

                       if (shipment.gpsReadings) {
                           shipment.gpsReadings.push(gpsReading);
                       } else {
                           shipment.gpsReadings = [gpsReading];
                       }

                       if (latLong == PORT_OF_NEW_YORK) {
                           var shipmentInPortEvent = factory.newEvent(NS, 'ShipmentInPortEvent');
                           shipmentInPortEvent.shipment = shipment;
                           var message = 'Shipment has reached the destination port of ' + PORT_OF_NEW_YORK;
                           shipmentInPortEvent.message = message;
                           console.log(message);
```

```
                            emit(shipmentInPortEvent);
                        }

                        return getAssetRegistry(NS + '.Shipment')
                        .then(function (shipmentRegistry) {
                            // add the temp reading to the shipment
                            return shipmentRegistry.update(shipment);
                        });
                    }
```

Now when you run the `TemperatureReading` and `GpsReading` transactions and they emit events, you can see them in the JavaScript console.

4b. Create the Business Network Archive

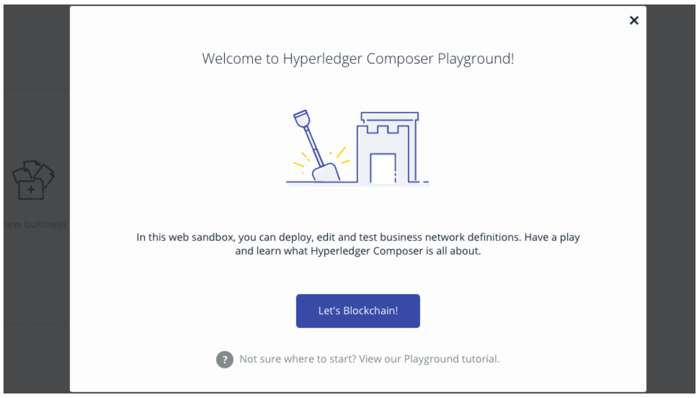
Go to the command line (Ubuntu) or open a terminal window (MacOS), navigate to the `perishable-network` directory, which is located in your `$COMPOSER_ROOT` directory, and run a build to create the BNA file by executing `npm install` from the command line. You'll see output like this:

```
                $ cd $COMPOSER_ROOT
                $ pwd
                /Users/sperry/HyperledgerComposer
                $ cd developerWorks/perishable-network/
                $ npm install

                > perishable-network@0.1.11 prepublish /Users/sperry/HyperledgerComposer/developerWorks/
perishable-network
                > mkdirp ./dist && composer archive create  --sourceType dir --sourceName . -a ./dist/
perishable-network.bna

                Creating Business Network Archive


                Looking for package.json of Business Network Definition
                 Input directory: /Users/sperry/HyperledgerComposer/developerWorks/perishable-network

                Found:
                 Description: Shipping Perishable Goods Business Network
                 Name: perishable-network
                 Identifier: perishable-network@0.1.11

                Written Business Network Definition Archive file to
                 Output file: ./dist/perishable-network.bna

                Command succeeded
```

4c. Import the model into the Online Playground on the IBM Cloud

Now go to the Online Playground on the IBM Cloud: https://composer-playground.mybluemix.net/. You should see the (by now) familiar Welcome screen:

Figure 4. Playground Welcome screen on the IBM Cloud

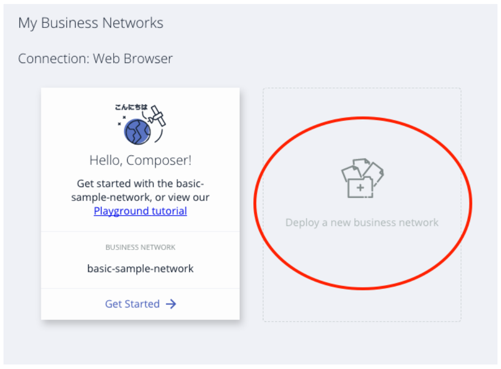

Welcome to Hyperledger Composer Playground!

In this web sandbox, you can deploy, edit and test business network definitions. Have a play and learn what Hyperledger Composer is all about.

If you don't see the Welcome screen, clear your browser's local storage. In Chrome, for example, under **Settings > Advanced > Content Settings > Cookies > All cookies** and **site data > localhost**, click the trashcan icon to remove local storage. If you're using a different browser, follow the instructions specific to that browser, and delete all local storage.

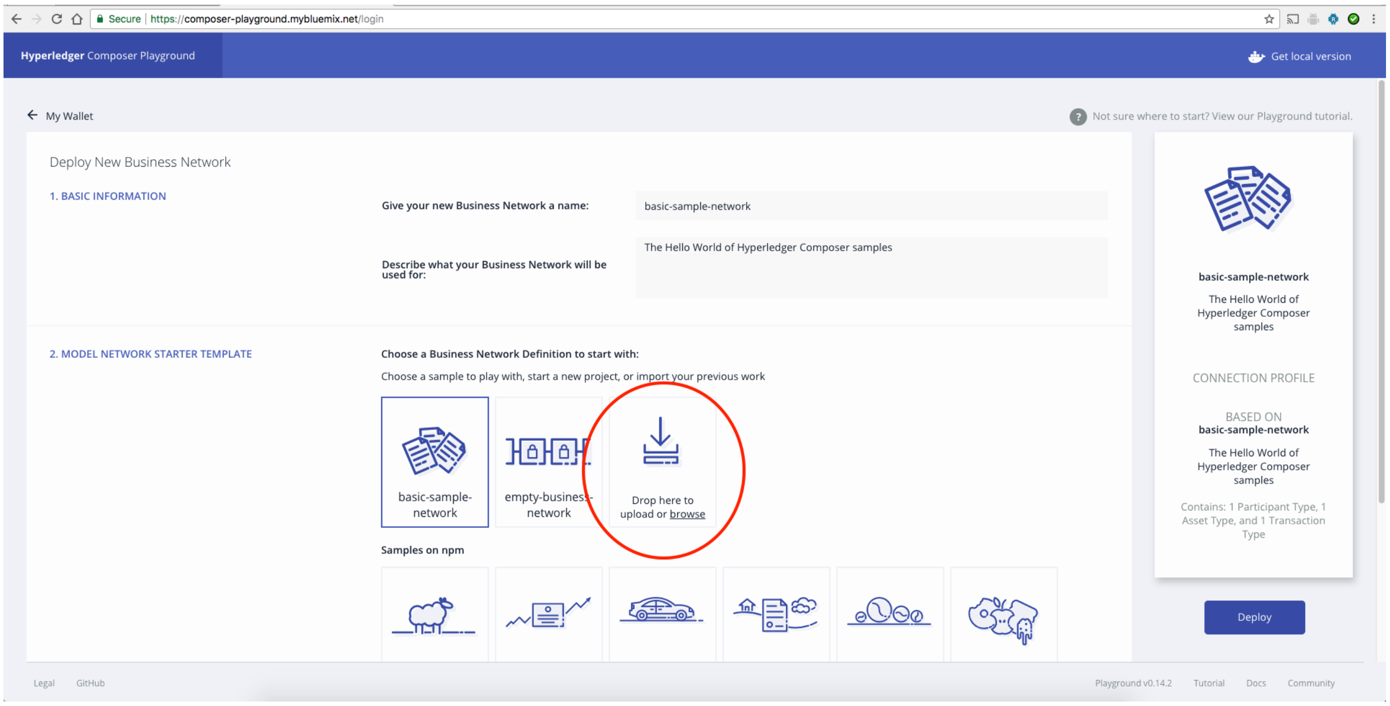
Click the **Let's Blockchain** button to get started. When you see the My Business Networks view, click **Deploy a new business network**.

Figure 5. My Business Networks



On the next screen, click **Drop here to upload or browse**. A File dialog opens. Navigate to the `$COMPOSER_ROOT/developerWorks/perishable-network/dist` directory, locate the file called `perishable-network.bna`, and click **Open**.

Figure 6. Deploy new business network


Secure | https://composer-playground.mybluemix.net/login
Hyperledger Composer Playground
Get local version
My Wallet
Not sure where to start? View our Playground tutorial.
Legal
GitHub
Playground v0.14.2
Tutorial
Docs
Community
Deploy

Deploy New Business Network

1. BASIC INFORMATION

Give your new Business Network a name: basic-sample-network

Describe what your Business Network will be used for: The Hello World of Hyperledger Composer samples

2. MODEL NETWORK STARTER TEMPLATE

Choose a Business Network Definition to start with:

Choose a sample to play with, start a new project, or import your previous work

basic-sample-network | empty-business-network | Drop here to upload or browse

Samples on npm

basic-sample-network

The Hello World of Hyperledger Composer samples

CONNECTION PROFILE

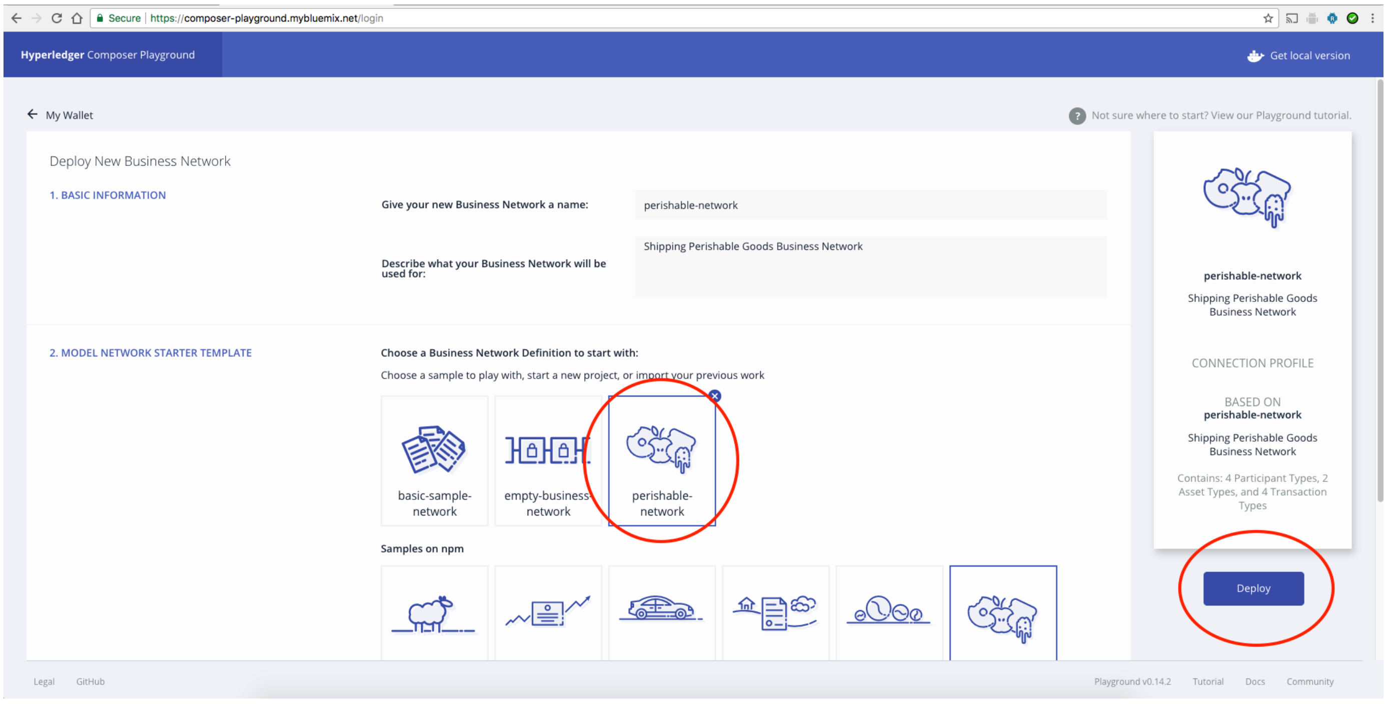
BASED ON basic-sample-network

The Hello World of Hyperledger Composer samples

Contains: 1 Participant Type, 1 Asset Type, and 1 Transaction Type

The title of the tile you just clicked on changes to **perishable-network**. Click the **Deploy** button to deploy the model.

Figure 7. Deploy new business network


Secure | https://composer-playground.mybluemix.net/login
Hyperledger Composer Playground
Get local version
My Wallet
Not sure where to start? View our Playground tutorial.
Legal
GitHub
Playground v0.14.2
Tutorial
Docs
Community
Deploy

Deploy New Business Network

1. BASIC INFORMATION

Give your new Business Network a name: perishable-network

Describe what your Business Network will be used for: Shipping Perishable Goods Business Network

2. MODEL NETWORK STARTER TEMPLATE

Choose a Business Network Definition to start with:

Choose a sample to play with, start a new project, or import your previous work

basic-sample-network | empty-business-network | perishable-network

Samples on npm

perishable-network

Shipping Perishable Goods Business Network

CONNECTION PROFILE

BASED ON perishable-network

Shipping Perishable Goods Business Network

Contains: 4 Participant Types, 2 Asset Types, and 4 Transaction Types

Now the model is deployed to the Online Playground in the IBM Cloud.

4d. Interact with the model

Interacting with the model should be familiar from Part 1 of this series. In fact, the Online Playground is an IBM Cloud version of the local Playground you used in Part 1, so I won't go over how to interact with your model again. Please review Part 1 if you need a refresher.

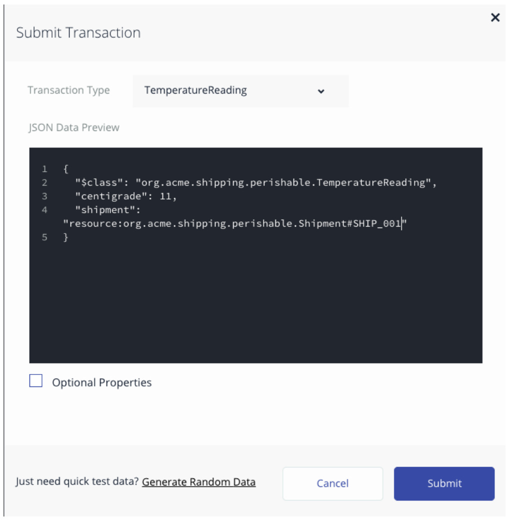
In the section titled "Refine the Perishable Goods business network," you ran unit tests to verify that the code in the new model works. Now you will interact with the model through the Online Playground to test those same transactions and verify them using the JavaScript console. Click **Connect now** on the Admin card to begin.

First, instantiate the model by clicking the **Test** tab just below the address bar in the browser. Click the **Submit Transaction** button, then select the SetupDemo transaction, and click **Submit**.

Now open the JavaScript console. How you do this depends on your browser. If you're using Chrome, select **View > Developer > JavaScript console**.

Next, submit a `TemperatureReading` transaction for `Shipment` SHIP_001 of 11 degrees C, which is above the max temperature threshold and will trigger a `TemperatureThresholdEvent`, as shown in Figure 8.

Figure 8. Submit `TemperatureReading` transaction



```
{
  "$class": "org.acme.shipping.perishable.TemperatureReading",
  "centigrade": 11,
  "shipment":
"resource:org.acme.shipping.perishable.Shipment#SHIP_001"
}
```

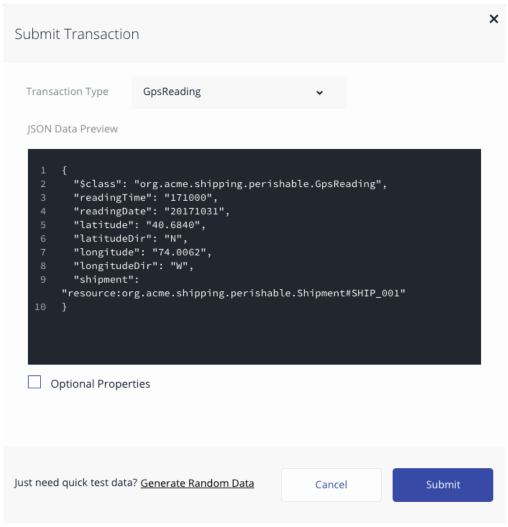
You should see the following message (among many others, so you may need to look carefully) in the JavaScript console: `Temperature threshold violated! Emitting TemperatureEvent for shipment: SHIP_001`. This indicates that the event is emitted properly (though we already knew that from running the Cucumber unit test successfully earlier, didn't we?).

Now submit a `GpsReading` transaction for `Shipment` SHIP_001 with the following parameters, as shown in Figure 9.

- readingTime = 171000
- readingDate = 20171031

- latitude = 40.6840
- latitudeDir = N
- longitude = 74.0062
- longitudeDir = W

Figure 9. Submit `GpsReading` transaction



Transaction Type: GpsReading

```
{
  "$class": "org.acme.shipping.perishable.GpsReading",
  "readingTime": "171000",
  "readingDate": "20171031",
  "latitude": "40.6840",
  "latitudeDir": "N",
  "longitude": "74.0062",
  "longitudeDir": "W",
  "shipment":
"resource:org.acme.shipping.perishable.Shipment#SHIP_001"
}
```

☐ Optional Properties

You should see the following message in the JavaScript console: `Shipment has reached the destination port of /LAT:40.6840N/LONG:74.0062W`.

Congratulations! You have a working model running in the IBM Cloud.

Conclusion to Part 2

In this tutorial, you set up the tools for local Hyperledger Composer development on your computer. You modified the Perishable Goods network and added a new transaction and two new events, and then unit tested those events with Cucumber.

Then you built a Composer Business Network Archive (BNA) file from the modified Perishable Goods network and deployed and tested it in the Online Playground hosted on the IBM Cloud.

Now you're ready for **Part 3**, where you'll install more tools to really harness the power of Hyperledger Composer by generating a REST interface. You'll install and run Hyperledger Fabric on your computer, and generate a GUI that you can use to interact with the Perishable Goods network running on your computer just as it would on a production server. No more Playground in Part 3!

Related topics

- All parts of this series
- Hyperledger Composer documentation
- Perishable Goods network (GitHub)
- More sample business network definitions (GitHub)
- Building a blockchain PoC in ten minutes using Hyperledger Composer
- IBM Blockchain Developer Center
- IBM Blockchain Platform
- Blockchain courses for developers