Hyperledger Composer Playground and the Perishable Goods Network

10/5/2017 Lennart Frantzell alf@us.ibm.com from existing materials

Table of Contents

Introduction

The **Hyperledger Composer** Playground is a web sandbox where you can deploy, edit and test **Business Network Definitions.**

https://hyperledger.github.io/composer/tutorials/playground-guide.html

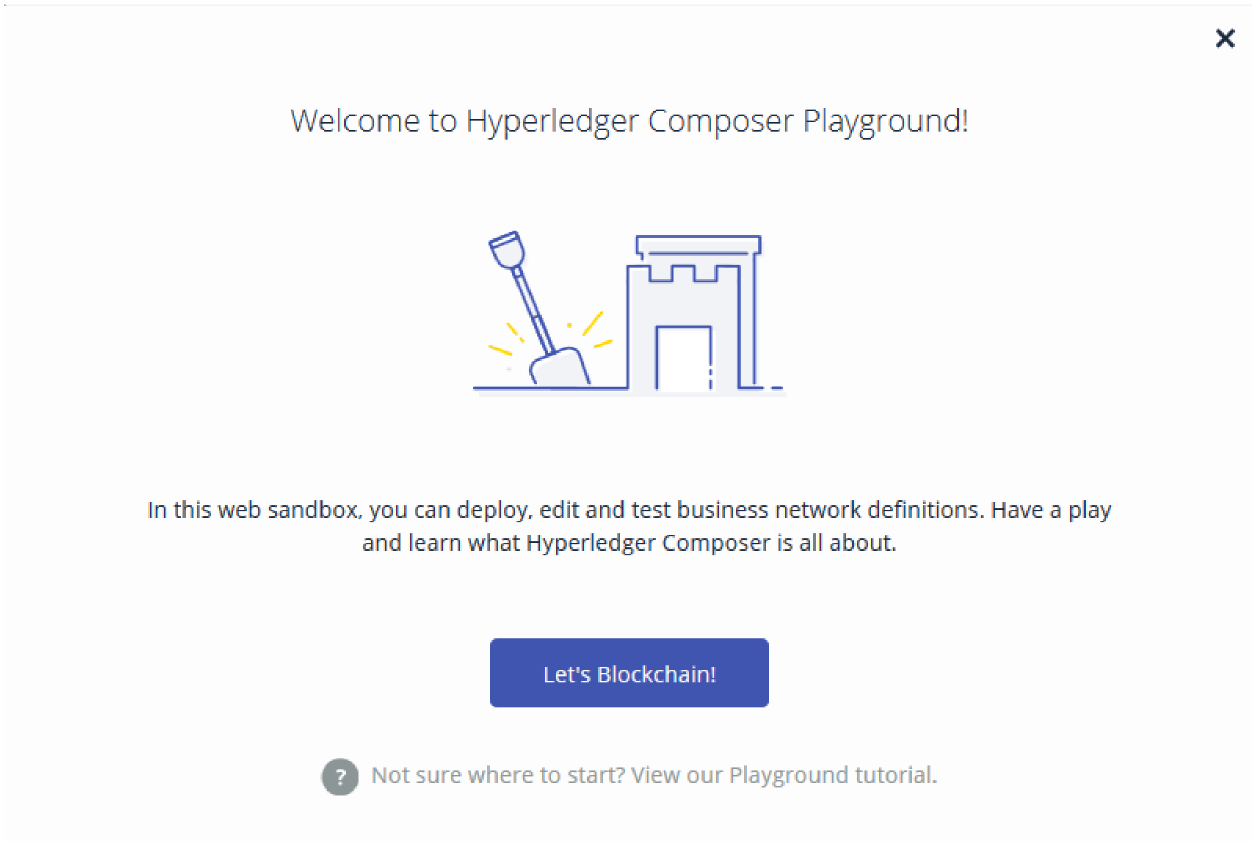
You can also export **your Business Network Definitions** and install them on a Hyperledger Fabric.

Step 1 Setting up the Perishable Goods Network

In this hands-on lab we will create a perishable Blockchain network with Hyperledger Composer.

We will then exercise the network to learn how it works.

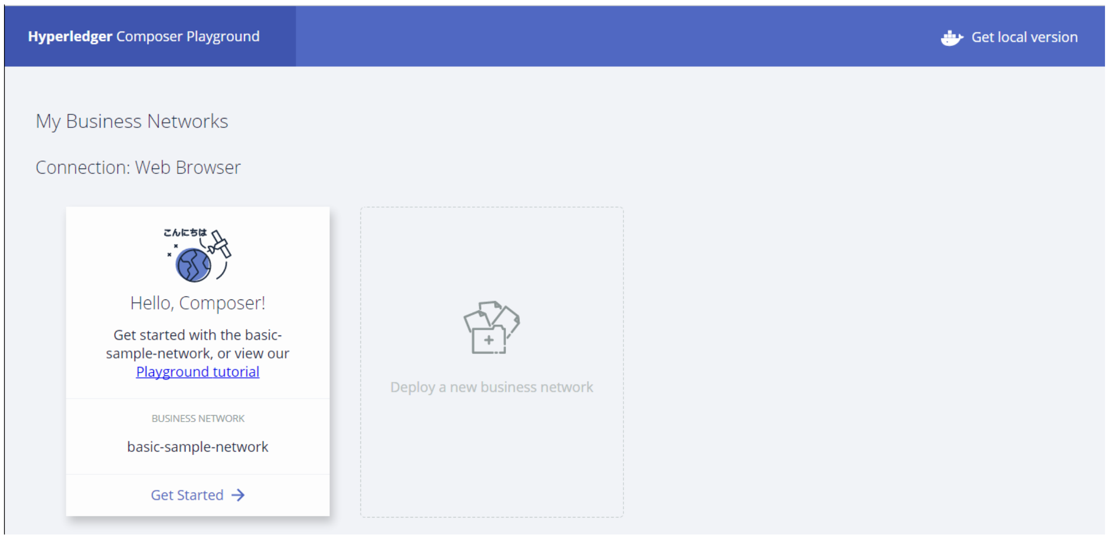
Log into the **Composer Playground**: https://composer-playground.mybluemix.net/



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.

Click on the blue **Lets Blockchain**! Button.


Hyperledger Composer Playground

Get local version

My Business Networks

Connection: Web Browser

Hello, Composer!

Get started with the basic-sample-network, or view our Playground tutorial

BUSINESS NETWORK

basic-sample-network

Get Started

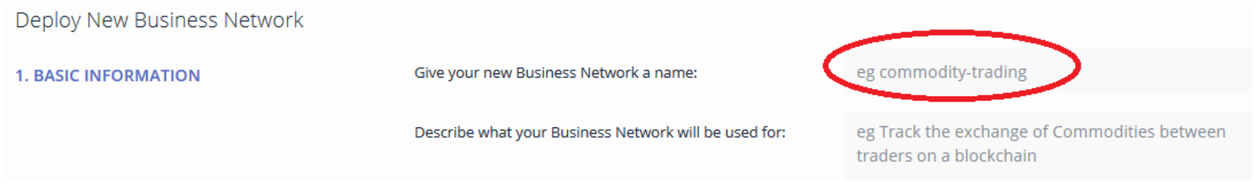
Deploy a new business network

We are now in the Hyperledger Composer home screen.

We click on **Deploy a new business network** in the right light blue tile

This will bring up a new window with two sections.

1) BASIC INFORMATION. And in this section we enter **perishable-network** as shown in red in the screen below.


Deploy New Business Network

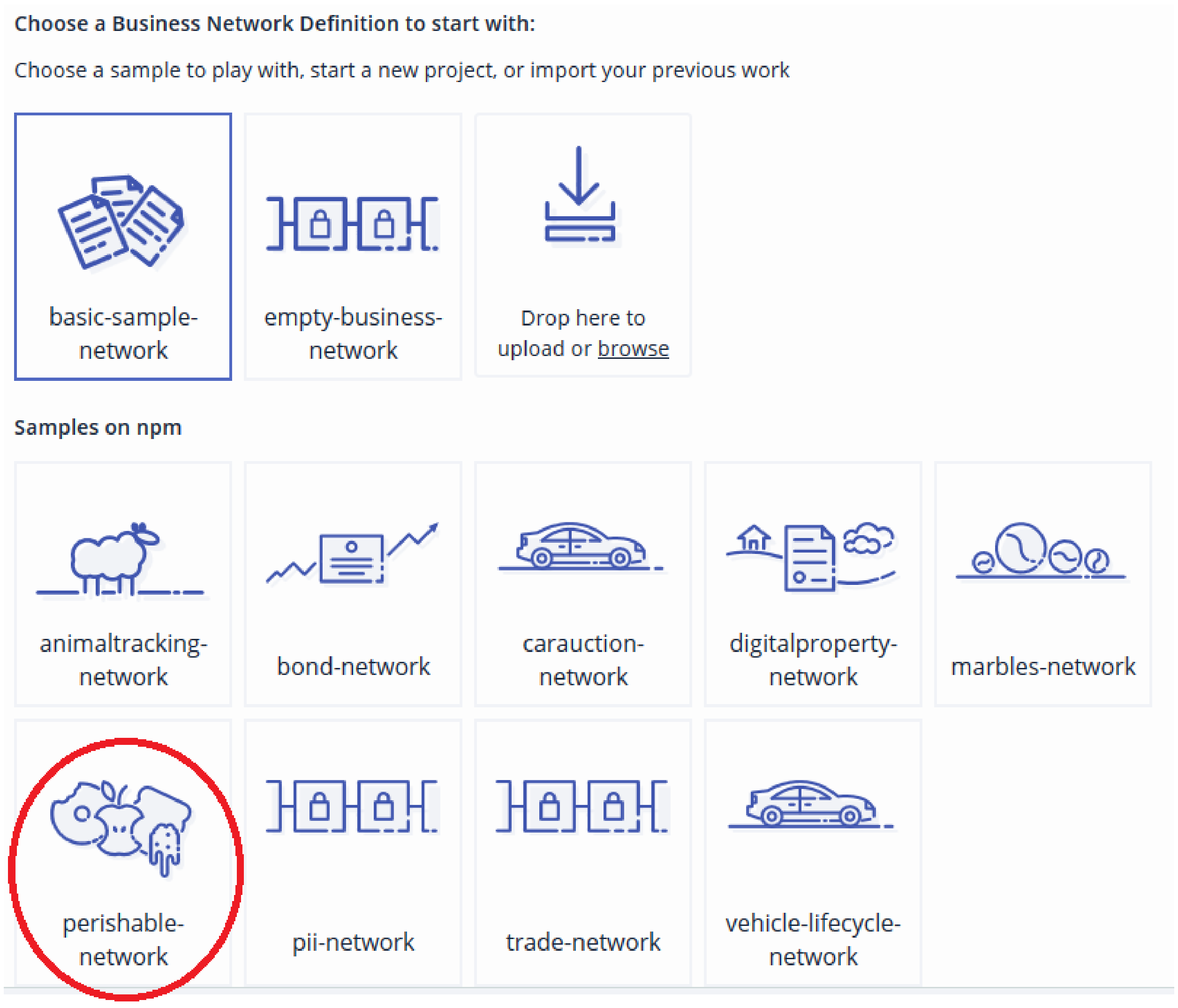
1. BASIC INFORMATION

Give your new Business Network a name: eg commodity-trading

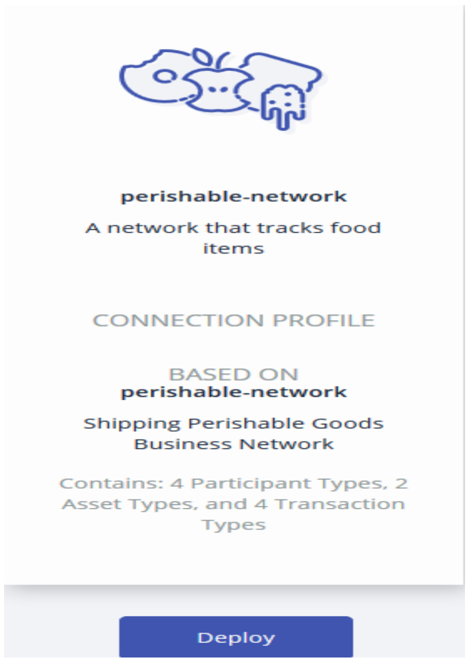
Describe what your Business Network will be used for: eg Track the exchange of Commodities between traders on a blockchain

Under the **BASIC INFORMATION** field is an optional field**: DESCRIBE WHAT YOUR BUSINESS NETWORK WILL BE USED FOR:**

2) We will then scroll down section two of the window**, Model Network Starter Template**, **Choose a Business Network Definition to start with**: until we find the **Perishable Network** icon, circled in red below. We will select it



Background Information: **Food Safety,** of which the **Perishable Network** is an example, is an important Use Case for Blockchain**.** https://www.ibm.com/blogs/blockchain/category/blockchain-in-food-safety/


perishable-network

A network that tracks food items

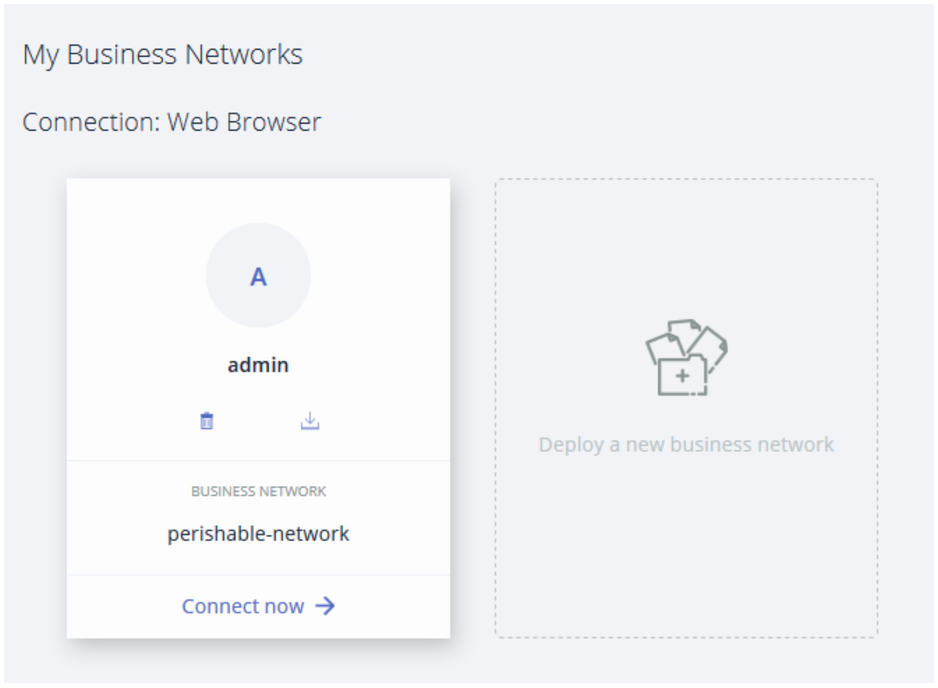
CONNECTION PROFILE

BASED ON
perishable-network

Shipping Perishable Goods Business Network

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

As we select the Perishable Network we see a new tile appear on the right-hand site of the screen, a Connection Profile, with a blue **Deploy** button. We will click on it.

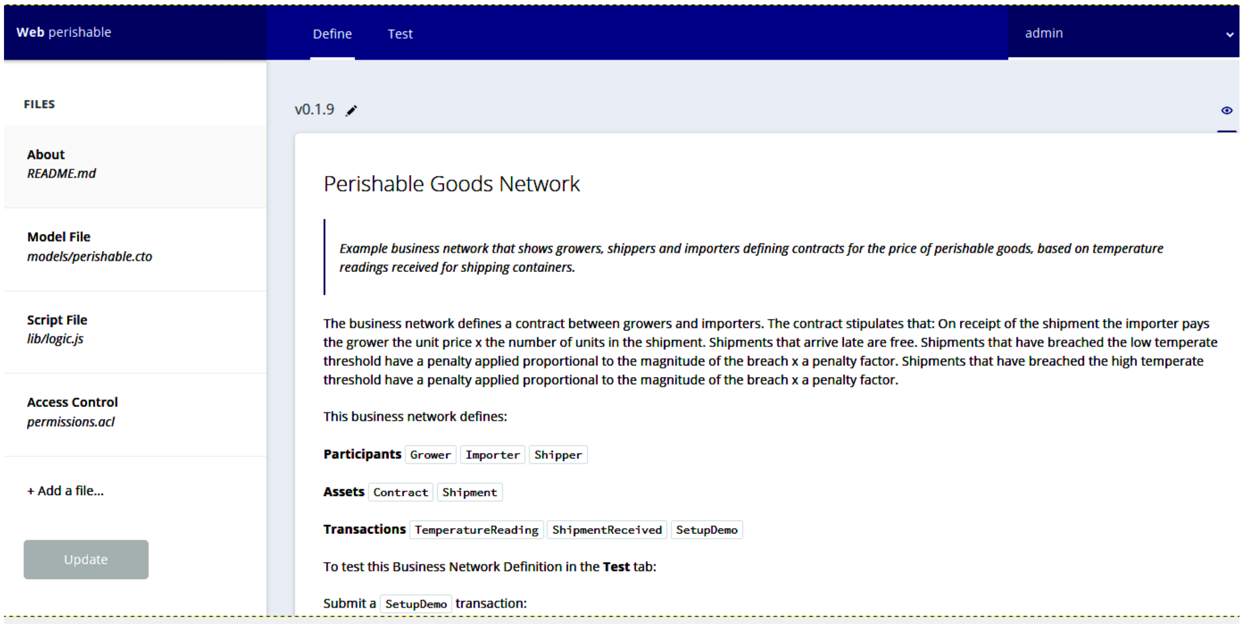

My Business Networks

Connection: Web Browser

This will bring us to a new screen with a white tile on the left that says perishable-network. We will click on the **Connect now ->** link.

This will bring up a window with a blue ribbon at the top with **Define** and **Test**.

In the left pane we see three base files in the **Perishable Goods Network,** the **Model File, Script File** and **Access Control** file. In the main pane we see Perishable Goods Network.



Perishable Goods Network

Example business network that shows growers, shippers and importers defining contracts for the price of perishable goods, based on temperature readings received for shipping containers.

The business network defines a contract between growers and importers. The contract stipulates that: On receipt of the shipment the importer pays the grower the unit price x the number of units in the shipment. Shipments that arrive late are free. Shipments that have breached the low temperate threshold have a penalty applied proportional to the magnitude of the breach x a penalty factor. Shipments that have breached the high temperate threshold have a penalty applied proportional to the magnitude of the breach x a penalty factor.

This business network defines:

Participants `Grower` `Importer` `Shipper`

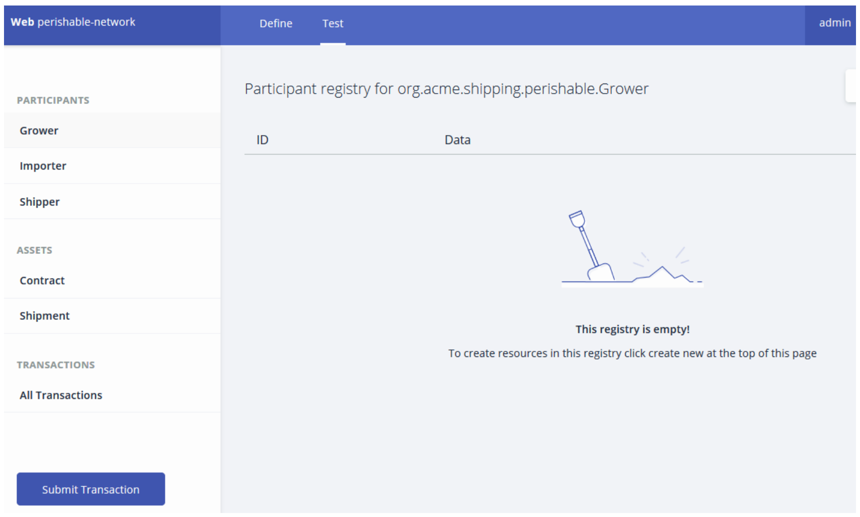
Assets `Contract` `Shipment`

Transactions `TemperatureReading` `ShipmentReceived` `SetupDemo`

To test this Business Network Definition in the **Test** tab:

Submit a `SetupDemo` transaction:

Define tab. Let’s now click on the **Test tab** in the blue ribbon**.**



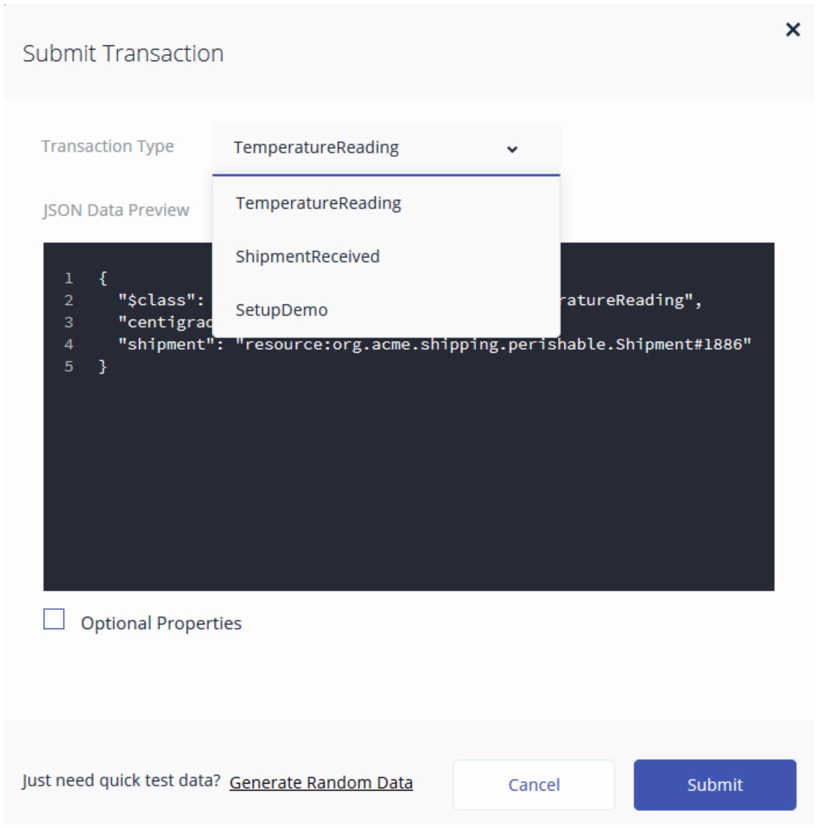
Participant registry for org.acme.shipping.perishable.Grower

This registry is empty!

To create resources in this registry click create new at the top of this page

If we look at the left pane we see the participants of our network, Grower, Importer and Shipper We have two Assets, A Contract and a Shipment. But these have only been defined, they contain no data. To import the data let’s do the following:

At the bottom left-hand corner we see a blue **Submit Transaction**. Let’s click on it.


Submit Transaction

Transaction Type: TemperatureReading

JSON Data Preview

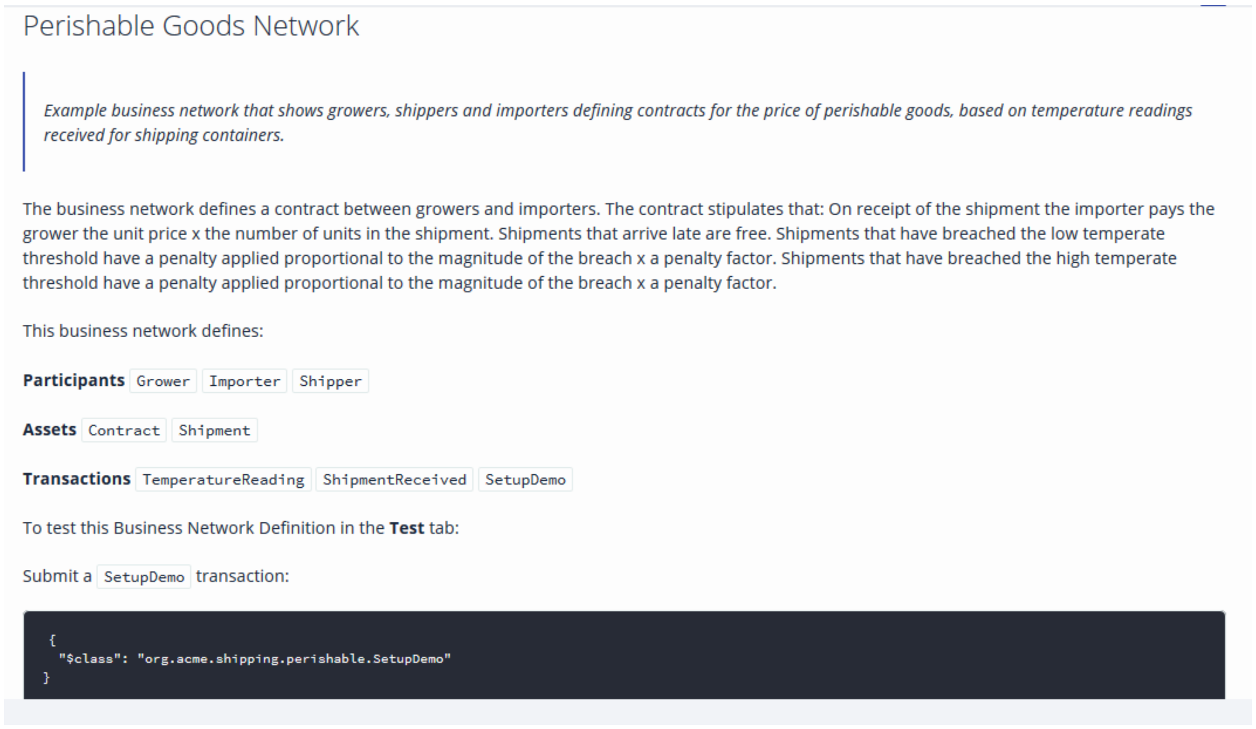
```
{
  "$class": …atureReading",
  "centigrad…
  "shipment": "resource:org.acme.shipping.perishable.Shipment#1886"
}
```

We see a black window with a list of Transaction types at the top. Let's select **SetupDemo** and click on the Blue Submit button at the bottom of the window.

Please Note: If you get an error running the **SetupDemo** there is an easy work-around:

Click on the **Define** Tab in the blue ribbon, then scroll down the screen and copy the **SetupDemo** code in the black section. Then click on the **Test** tab in the blue ribbon. Click on the Blue **Submit Transaction** in the lower left hand corner and then select **SetupDemo** in the popup window, and then paste the code we just copied into the popup window and click **Submit**.

That should work.



Perishable Goods Network

> *Example business network that shows growers, shippers and importers defining contracts for the price of perishable goods, based on temperature readings received for shipping containers.*

The business network defines a contract between growers and importers. The contract stipulates that: On receipt of the shipment the importer pays the grower the unit price x the number of units in the shipment. Shipments that arrive late are free. Shipments that have breached the low temperate threshold have a penalty applied proportional to the magnitude of the breach x a penalty factor. Shipments that have breached the high temperate threshold have a penalty applied proportional to the magnitude of the breach x a penalty factor.

This business network defines:

Participants `Grower` `Importer` `Shipper`

Assets `Contract` `Shipment`

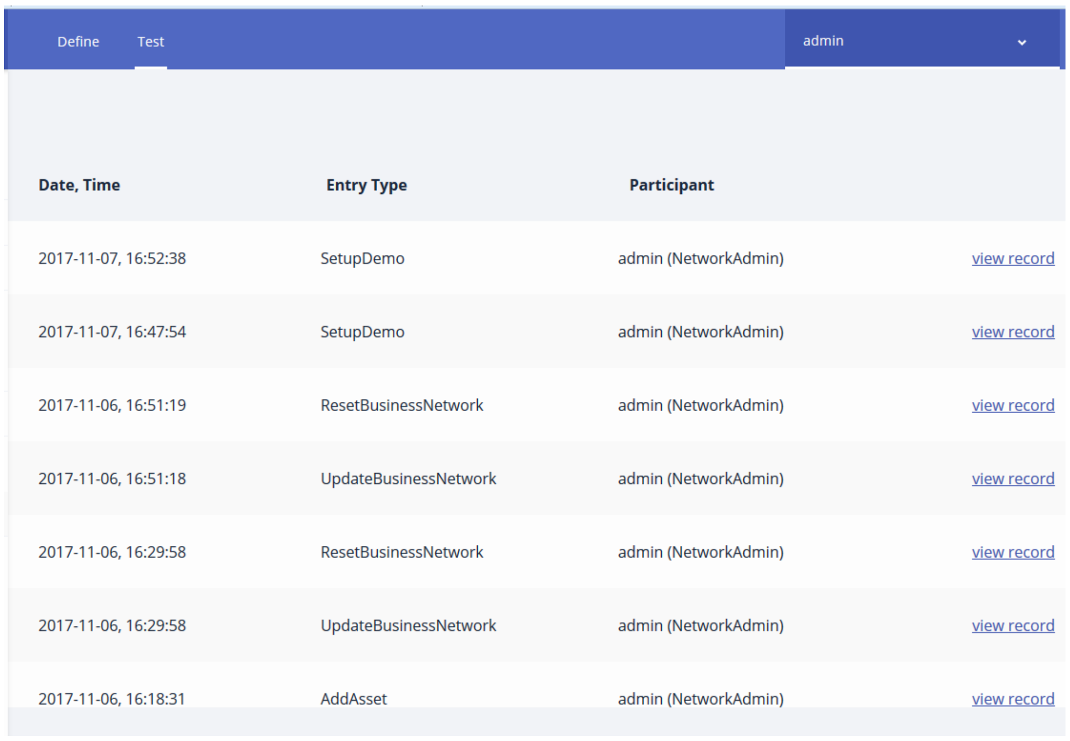
Transactions `TemperatureReading` `ShipmentReceived` `SetupDemo`

To test this Business Network Definition in the **Test** tab:

Submit a `SetupDemo` transaction:

```
{
  "$class": "org.acme.shipping.perishable.SetupDemo"
}
```

After we have run the Setup script we look at the Test Tab and click on **All Transactions** in the left pane, where we can see all the transactions that have taken place to set up the Perishable Goods Network.



Date, Time	Entry Type	Participant	
2017-11-07, 16:52:38	SetupDemo	admin (NetworkAdmin)	view record
2017-11-07, 16:47:54	SetupDemo	admin (NetworkAdmin)	view record
2017-11-06, 16:51:19	ResetBusinessNetwork	admin (NetworkAdmin)	view record
2017-11-06, 16:51:18	UpdateBusinessNetwork	admin (NetworkAdmin)	view record
2017-11-06, 16:29:58	ResetBusinessNetwork	admin (NetworkAdmin)	view record
2017-11-06, 16:29:58	UpdateBusinessNetwork	admin (NetworkAdmin)	view record
2017-11-06, 16:18:31	AddAsset	admin (NetworkAdmin)	view record

We can click on **View Record** to see the contents of each transaction.

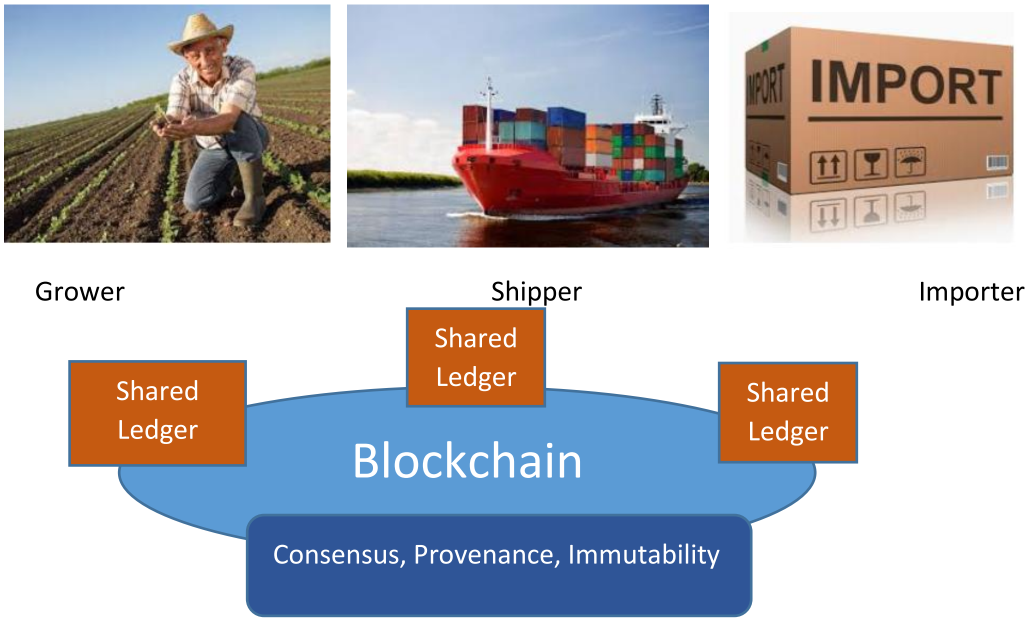
Let’s look at the Perishable Goods Network

The Hyperledger Composer Modeling Language

The Perishable Goods Network is written in the Hyperledger Composer Modeling Language. A full tutorial of the Hyperledger Composer Modeling Language can be found at this link:

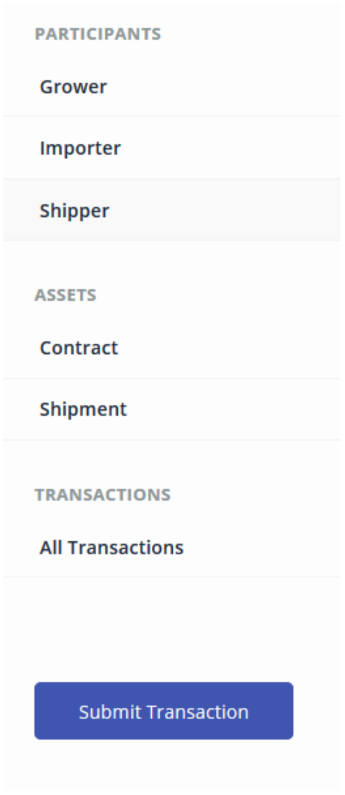
https://hyperledger.github.io/composer/reference/cto_language.html

The Perishable Goods Network architecture:



The Perishable Goods Network consists of three parties that share the same ledger. A Grower, a Shipper and an Importer. The shared ledger keeps track of the Account Balance of the three parties, as we will show below.

Let us now look at the code. To look at the contents of the assets, we click on the left tab we looked at before, and select the various participants. That now contain data.



1. Let’s begin with the three participants that all use the Blockchain network: a **Grower**, an **Importer** and a **Shipper**.

Participant registry for org.acme.shipping.perishable.Grower

farmer@email.com

```
{
  "$class": "org.acme.shipping.perishable.Grower",
  "email": "farmer@email.com",
  "address": {
    "$class": "org.acme.shipping.perishable.Address",
    "country": "USA"
  },
  "accountBalance": 0
}
```

Participant registry for org.acme.shipping.perishable.Importer

```
{
  "$class": "org.acme.shipping.perishable.Importer",
  "email": "supermarket@email.com",
  "address": {
    "$class": "org.acme.shipping.perishable.Address",
    "country": "UK"
  },
  "accountBalance": 0
}
```

Participant registry for org.acme.shipping.perishable.Shipper

```
{
  "$class": "org.acme.shipping.perishable.Shipper",
  "email": "shipper@email.com",
  "address": {
    "$class": "org.acme.shipping.perishable.Address",
    "country": "Panama"
  },
  "accountBalance": 0
}
```

What binds these three participants together is a **Smart Contract** that describes the rules that determines what the parties expect of the transactions involved in shipping fruit from the grower to the importer and which is stored in the Blockchain.

2. There are two **assets:** the **Contract** and the **Shipment**

The **Contract** defines the participants in the network and their obligations. In this case an arrival date, a unit price, a minimum and maximum temperature during the shipment, as well as a minimum and maximum penalty factor in case the shipper misses his agreed-upon arrival date.

These conditions are all encapsulated in the Smart Contract and visible to all participants in the network, which have agreed them in the contract through the **consensus** mechanism.

Asset registry for org.acme.shipping.perishable.Contract

```
{
  "$class": "org.acme.shipping.perishable.Contract",
  "contractId": "CON_001",
  "grower": "resource:org.acme.shipping.perishable.Grower#farmer@email.com",
  "shipper":
"resource:org.acme.shipping.perishable.Shipper#shipper@email.com",
  "importer":
"resource:org.acme.shipping.perishable.Importer#supermarket@email.com",
  "arrivalDateTime": "2017-10-10T16:31:48.653Z",
  "unitPrice": 0.5,
  "minTemperature": 2,
  "maxTemperature": 10,
  "minPenaltyFactor": 0.2,
  "maxPenaltyFactor": 0.1
}
```

The **shipment** defines the type of goods being shipped, the unit count and encapsulates the temperature readings during the shipment. There is also a link to the **Smart Contract**.

Asset registry for org.acme.shipping.perishable.Shipment

```
{
  "$class": "org.acme.shipping.perishable.Shipment",
  "shipmentId": "SHIP_001",
  "type": "BANANAS",
  "status": "ARRIVED",
  "unitCount": 5000,
  "temperatureReadings": [
    {
      "$class": "org.acme.shipping.perishable.TemperatureReading",
      "centigrade": 8,
      "shipment": "resource:org.acme.shipping.perishable.Shipment#SHIP_001",
      "transactionId": "5d7e0439-ff34-46c5-8ad6-6a1a61218713",
      "timestamp": "2017-10-09T16:33:09.032Z"
    }
  ],
  "contract": "resource:org.acme.shipping.perishable.Contract#CON_001"
}
```

Let's look at the support files:

The Model File

```
/**
   * A business network for shipping perishable goods
   * The cargo is temperature controlled and contracts
   * can be negotiated 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
}

/**
  * An abstract transaction that is related to a Shipment
  */
abstract transaction ShipmentTransaction {
        --> Shipment shipment
}

/**
  * An temperature reading for a shipment. E.g. received from a
  * device within a temperature controlled shipping container
  */
transaction TemperatureReading extends ShipmentTransaction {
    o Double centigrade
}

/**
  * A notification that a shipment has been received by the
  * importer and that funds should be transferred from the importer
  * to the grower to pay for the shipment.
  */
transaction ShipmentReceived extends ShipmentTransaction {
}

/**
  * A shipment being tracked as an asset on the ledger
  */
asset Shipment identified by shipmentId {
    o String shipmentId
    o ProductType type
    o ShipmentStatus status
    o Long unitCount
    o TemperatureReading[] temperatureReadings optional
    --> Contract contract
}

/**
  * Defines a contract between a Grower and an Importer to ship using
  * a Shipper, paying a set unit price. The unit price is multiplied by
  * a penality factor proportional to the deviation from the min and max
  * negociated temperatures for the shipment.
  */
asset Contract identified by contractId {
    o String contractId
    --> Grower grower
    --> Shipper shipper
    --> Importer importer
    o DateTime arrivalDateTime
    o Double unitPrice
    o Double minTemperature
    o Double maxTemperature
    o Double minPenaltyFactor
    o Double maxPenaltyFactor
}

/**
  * A concept for a simple street address
  */
concept Address {
    o String city optional
    o String country
    o String street optional
    o String zip optional
}

/**
  * An abstract participant type in this business network
  */
abstract participant Business identified by email {
    o String email
    o Address address
    o Double accountBalance
}

/**
  * A Grower is a type of participant in the network
  */
participant Grower extends Business {
}

/**
  * A Shipper is a type of participant in the network
  */
participant Shipper extends Business {
}

/**
  * An Importer is a type of participant in the network
  */
participant Importer extends Business {
}

/**
  * JUST FOR INITIALIZING A DEMO
  */
transaction SetupDemo {
}
```

The Script file containing the application logic

```
/*
  * Licensed under the Apache License, Version 2.0 (the "License");
  * you may not use this file except in compliance with the License.
  * You may obtain a copy of the License at
  *
  * http://www.apache.org/licenses/LICENSE-2.0
  *
  * Unless required by applicable law or agreed to in writing, software
  * distributed under the License is distributed on an "AS IS" BASIS,
  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  * See the License for the specific language governing permissions and
  * limitations under the License.
  */

/**
  * A shipment has been received by an importer
  * @param {org.acme.shipping.perishable.ShipmentReceived} shipmentReceived - the
ShipmentReceived transaction
  * @transaction
  */
function payOut(shipmentReceived) {

        var contract = shipmentReceived.shipment.contract;
        var shipment = shipmentReceived.shipment;
        var payOut = contract.unitPrice * shipment.unitCount;

        console.log('Received at: ' + shipmentReceived.timestamp);
        console.log('Contract arrivalDateTime: ' + contract.arrivalDateTime);

        // set the status of the shipment
        shipment.status = 'ARRIVED';

        // if the shipment did not arrive on time the payout is zero
        if (shipmentReceived.timestamp > contract.arrivalDateTime) {
                payOut = 0;
                console.log('Late shipment');
        } else {
                // find the lowest temperature reading
                if (shipment.temperatureReadings) {
                        // sort the temperatureReadings by centigrade
                        shipment.temperatureReadings.sort(function (a, b) {
                                return (a.centigrade - b.centigrade);
                        });
                        var lowestReading = shipment.temperatureReadings[0];
                        var highestReading =
shipment.temperatureReadings[shipment.temperatureReadings.length - 1];
                        var penalty = 0;
                        console.log('Lowest temp reading: ' + lowestReading.centigrade);
                        console.log('Highest temp reading: ' + highestReading.centigrade);

                        // does the lowest temperature violate the contract?
                        if (lowestReading.centigrade < contract.minTemperature) {
                                penalty += (contract.minTemperature - lowestReading.centigrade) *
contract.minPenaltyFactor;
                                console.log('Min temp penalty: ' + penalty);
                        }

                        // does the highest temperature violate the contract?
                        if (highestReading.centigrade > contract.maxTemperature) {
                                penalty += (highestReading.centigrade - contract.maxTemperature) *
contract.maxPenaltyFactor;
                                console.log('Max temp penalty: ' + penalty);
                        }

                        // apply any penalities
                        payOut -= (penalty * shipment.unitCount);

                        if (payOut < 0) {
                                payOut = 0;
                        }
                }
        }

        console.log('Payout: ' + payOut);
        contract.grower.accountBalance += payOut;
        contract.importer.accountBalance -= payOut;

        console.log('Grower: ' + contract.grower.$identifier + ' new balance: ' +
contract.grower.accountBalance);
        console.log('Importer: ' + contract.importer.$identifier + ' new balance: ' +
contract.importer.accountBalance);

        return getParticipantRegistry('org.acme.shipping.perishable.Grower')
                .then(function (growerRegistry) {
```

```
                // update the grower's balance
                return growerRegistry.update(contract.grower);
            })
            .then(function () {
                return getParticipantRegistry('org.acme.shipping.perishable.Importer');
            })
            .then(function (importerRegistry) {
                // update the importer's balance
                return importerRegistry.update(contract.importer);
            })
            .then(function () {
                return getAssetRegistry('org.acme.shipping.perishable.Shipment');
            })
            .then(function (shipmentRegistry) {
                // update the state of the shipment
                return shipmentRegistry.update(shipment);
            });
}

/**
  * A temperature reading has been received for a shipment
  * @param {org.acme.shipping.perishable.TemperatureReading} temperatureReading - the
TemperatureReading transaction
  * @transaction
  */
function temperatureReading(temperatureReading) {

    var shipment = temperatureReading.shipment;

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

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

    return getAssetRegistry('org.acme.shipping.perishable.Shipment')
        .then(function (shipmentRegistry) {
            // add the temp reading to the shipment
            return shipmentRegistry.update(shipment);
        });
}

/**
  * Initialize some test assets and participants useful for running a demo.
  * @param {org.acme.shipping.perishable.SetupDemo} setupDemo - the SetupDemo transaction
  * @transaction
  */
function setupDemo(setupDemo) {

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

        // create the grower
        var grower = factory.newResource(NS, 'Grower', 'farmer@email.com');
        var growerAddress = factory.newConcept(NS, 'Address');
        growerAddress.country = 'USA';
        grower.address = growerAddress;
        grower.accountBalance = 0;

        // create the importer
        var importer = factory.newResource(NS, 'Importer', 'supermarket@email.com');
        var importerAddress = factory.newConcept(NS, 'Address');
        importerAddress.country = 'UK';
        importer.address = importerAddress;
        importer.accountBalance = 0;

        // create the shipper
        var shipper = factory.newResource(NS, 'Shipper', 'shipper@email.com');
        var shipperAddress = factory.newConcept(NS, 'Address');
        shipperAddress.country = 'Panama';
        shipper.address = shipperAddress;
        shipper.accountBalance = 0;

        // create the contract
        var contract = factory.newResource(NS, 'Contract', 'CON_001');
        contract.grower = factory.newRelationship(NS, 'Grower', 'farmer@email.com');
        contract.importer = factory.newRelationship(NS, 'Importer', 'supermarket@email.com');
        contract.shipper = factory.newRelationship(NS, 'Shipper', 'shipper@email.com');
        var tomorrow = setupDemo.timestamp;
        tomorrow.setDate(tomorrow.getDate() + 1);
        contract.arrivalDateTime = tomorrow; // the shipment has to arrive tomorrow
        contract.unitPrice = 0.5; // pay 50 cents per unit
        contract.minTemperature = 2; // min temperature for the cargo
        contract.maxTemperature = 10; // max temperature for the cargo
        contract.minPenaltyFactor = 0.2; // we reduce the price by 20 cents for every degree below the
min temp
        contract.maxPenaltyFactor = 0.1; // we reduce the price by 10 cents for every degree above the
max temp

        // create the shipment
        var shipment = factory.newResource(NS, 'Shipment', 'SHIP_001');
        shipment.type = 'BANANAS';
        shipment.status = 'IN_TRANSIT';
        shipment.unitCount = 5000;
        shipment.contract = factory.newRelationship(NS, 'Contract', 'CON_001');
        return getParticipantRegistry(NS + '.Grower')
                .then(function (growerRegistry) {
                        // add the growers
                        return growerRegistry.addAll([grower]);
                })
                .then(function() {
                        return getParticipantRegistry(NS + '.Importer');
                })
                .then(function(importerRegistry) {
                        // add the importers
                        return importerRegistry.addAll([importer]);
                })
                .then(function() {
                        return getParticipantRegistry(NS + '.Shipper');
                })
                .then(function(shipperRegistry) {
                        // add the shippers
                        return shipperRegistry.addAll([shipper]);
                })
                .then(function() {
                        return getAssetRegistry(NS + '.Contract');
                })
                .then(function(contractRegistry) {
                        // add the contracts
                        return contractRegistry.addAll([contract]);
                })
                .then(function() {
                        return getAssetRegistry(NS + '.Shipment');
                })
                .then(function(shipmentRegistry) {
                        // add the shipments
                        return shipmentRegistry.addAll([shipment]);
                });
}
```

The access control file

```
/**
  * Sample access control list.
  */
rule Default {
        description: "Allow all participants access to all resources"
        participant: "ANY"
        operation: ALL
        resource: "org.acme.shipping.perishable.*"
        action: ALLOW
}

rule SystemACL {
    description:    "System ACL to permit all access"
    participant: "org.hyperledger.composer.system.Participant"
    operation: ALL
    resource: "org.hyperledger.composer.system.**"
    action: ALLOW
}
```

We now have our business network running.

To test this Business Network Definition in the **Test** tab:

Submit a `SetupDemo` transaction:

```
{
    "$class": "org.acme.shipping.perishable.SetupDemo"
}
```

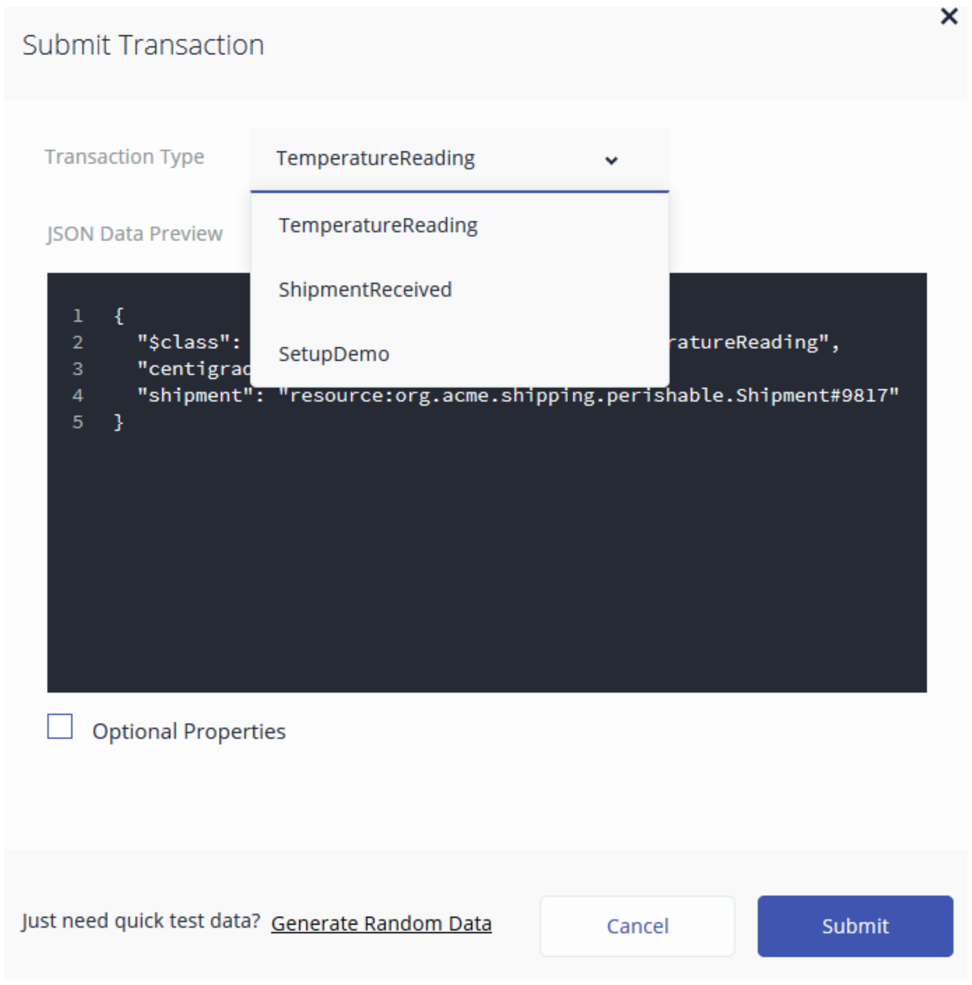
This transaction populates the Participant Registries with a `Grower`, an `Importer` and a `Shipper`. The Asset Registries will have a `Contract` asset and a `Shipment` asset.

Important, out of the box the Grower, Importer and Shipper all have an account balance of $0.

Step 2. Running the Perishable Goods Network

To run the Perishable Goods Network we click on the **Test** tab in the blue ribbon at the top of the window.

We have two commands we can use: **Temperature Reading** and **Shipment Received**.



```
{
  "$class":                                    atureReading",
  "centigrac
  "shipment": "resource:org.acme.shipping.perishable.Shipment#9817"
}
```

Submit a TemperatureReading transaction

TemperatureReading is a stand-in for temperature sensors on board the plane or ship that transports the fruit. And allows us to modify the temperature so that it violates the provisions in the **Smart Contract.**

```
{
    "$class": "org.acme.shipping.perishable.TemperatureReading",
    "centigrade": 5,
    "shipment": "resource:org.acme.shipping.perishable.Shipment#SHIP_001"
}
```

If the temperature reading falls outside the min/max range of the contract, the price received by the grower will be reduced. You may submit several readings if you wish. Each reading will be aggregated within `SHIP_001` Shipment Asset Registry.

Submit a ShipmentReceived transaction

For the SHIP_001 shipment. It triggers the payout to the grower, based on the parameters of the `CON_001` contract.

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

The Grower's and Importer's account balance will change

If the date-time of the `ShipmentReceived` transaction is after the `arrivalDateTime` on `CON_001` then the grower will not receive any payment for the shipment.

Congratulations!

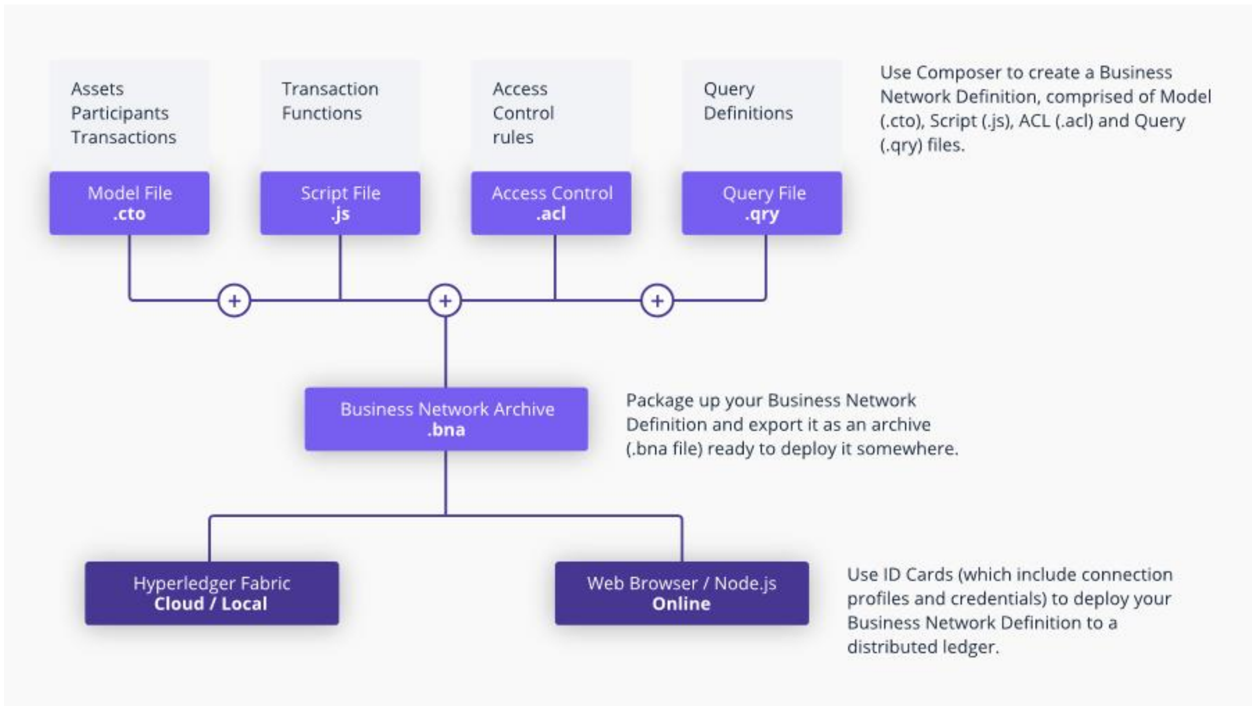
Step 3 Installing the Composer locally

https://hyperledger.github.io/composer/reference/composer.network.deploy.html

Step 4 Where do we go from here?

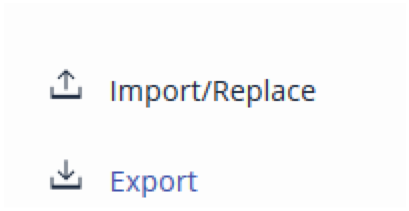
Create your own Business Network Definitions from scratch.

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


Assets
Participants
Transactions
Transaction
Functions
Access
Control
rules
Query
Definitions
Use Composer to create a Business Network Definition, comprised of Model (.cto), Script (.js), ACL (.acl) and Query (.qry) files.
Model File
.cto
Script File
.js
Access Control
.acl
Query File
.qry
Business Network Archive
.bna
Package up your Business Network Definition and export it as an archive (.bna file) ready to deploy it somewhere.
Hyperledger Fabric
Cloud / Local
Web Browser / Node.js
Online
Use ID Cards (which include connection profiles and credentials) to deploy your Business Network Definition to a distributed ledger.

Export BNA file.

https://hyperledger.github.io/composer/business-network/bnd-deploy.html



Appendix

https://hyperledger.github.io/composer/tutorials/tutorials.html
https://hyperledger.github.io/composer/introduction/introduction.html