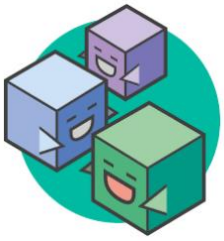
Hyperledger Composer Playground and the Perishable Goods Network



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

Table of Contents

Hyperledger Composer Playground and 1
the Perishable Goods Network 1
Introduction 3
Step 1 Setting up the Perishable Goods Network 3
Let's look at the Perishable Goods Network 6
The Hyperledger Composer Modeling Language 6
The Perishable Goods Network architecture: 6
Participant registry for org.acme.shipping.perishable.Grower 7
Participant registry for org.acme.shipping.perishable.Importer 8
Participant registry for org.acme.shipping.perishable.Shipper 8
Asset registry for org.acme.shipping.perishable.Contract 8
Asset registry for org.acme.shipping.perishable.Shipment 9
Let's look at the support files: 9
The Model File 9
The Script file containing the application logic 12
The access control file 17
Step 2. Running the Perishable Goods Network 20
Submit a TemperatureReading transaction 20
Submit a ShipmentReceived transaction 20
Step 3 Installing the Composer locally 21
Step 4 Where do we go from here? 21
Export BNA file. 21

Introduction

The **Hyperledger Composer** Playground is a web sandbox where you can deploy, edit and test business network definitions.

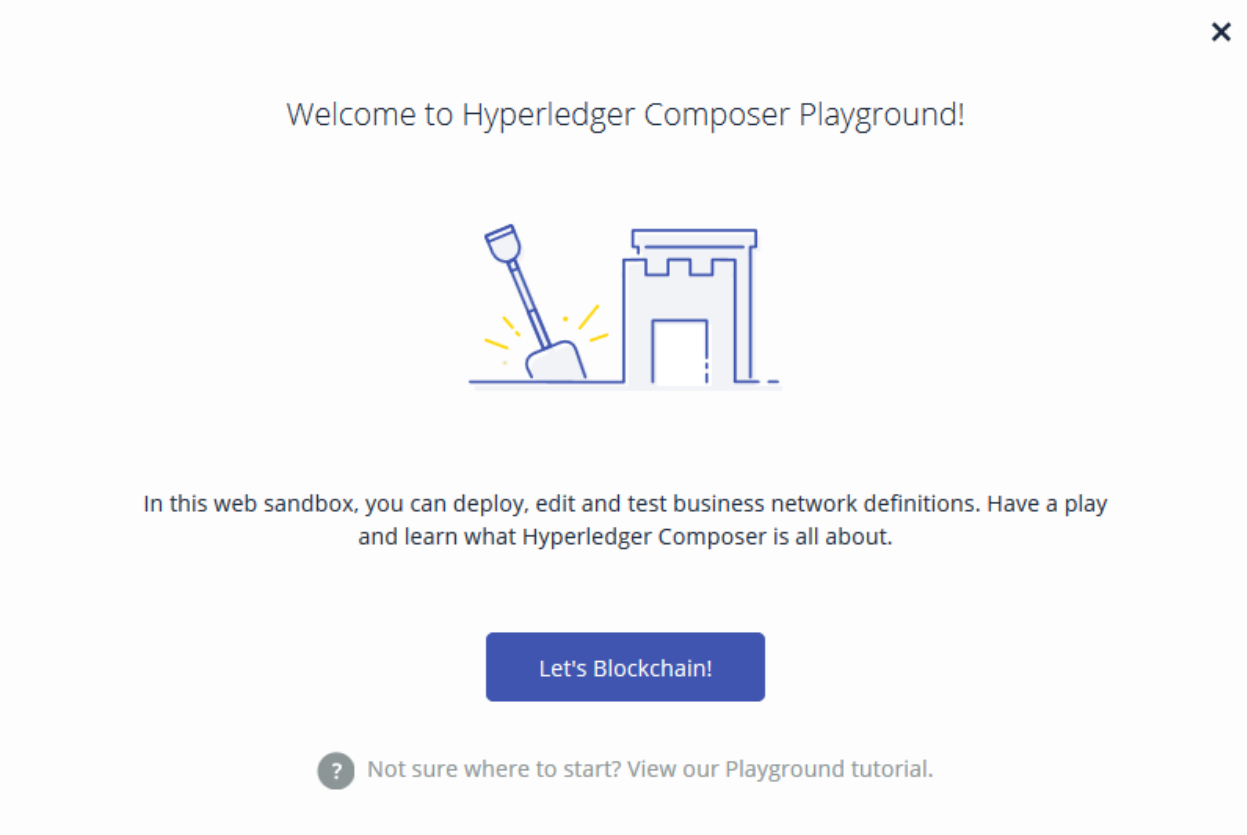
https://hyperledger.github.io/composer/tutorials/playground-guide.html

Step 1 Setting up the Perishable Goods Network

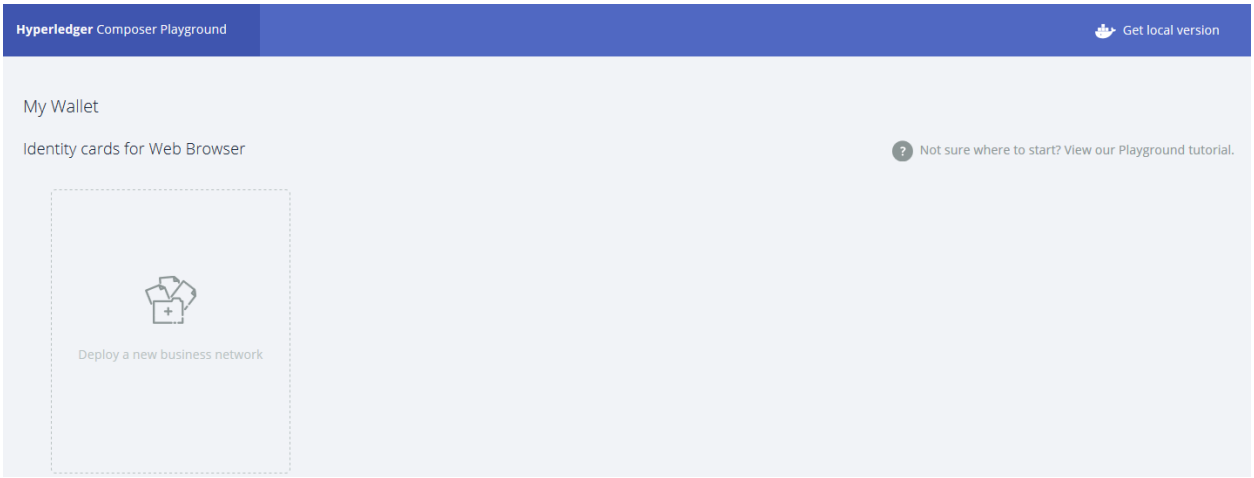
In this hands-on lab we will create a perishable Blockchain network with Hyperledger Composer.

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

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



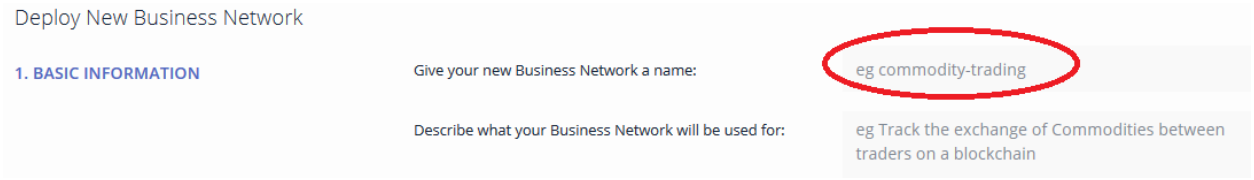
Click on the blue **Lets Blockchain**! Button.



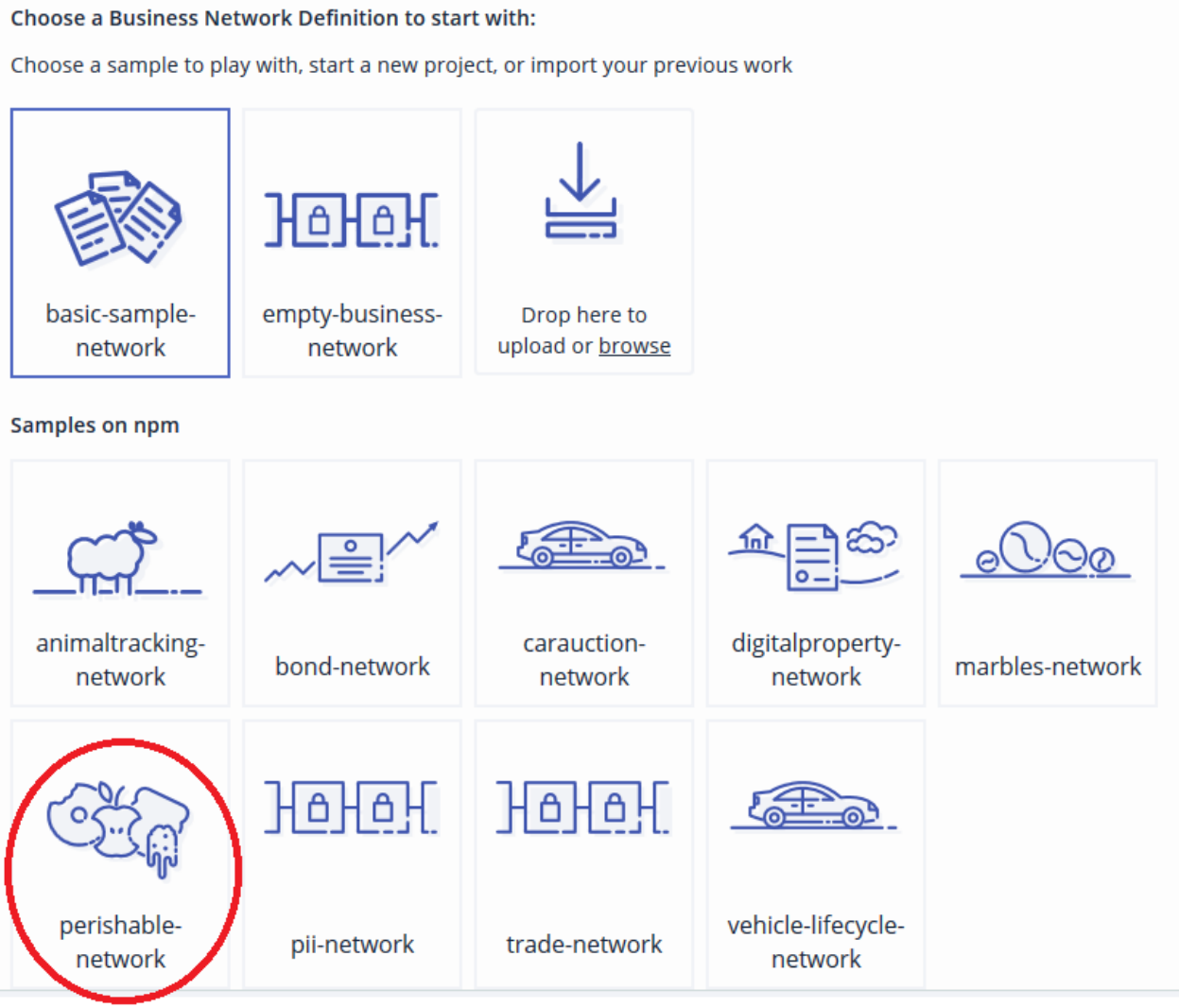
We are now in the Hyperledger Composer home screen.

We will click on the tile on the left, **Deploy a new Business Network**.

This will bring up a wizard.



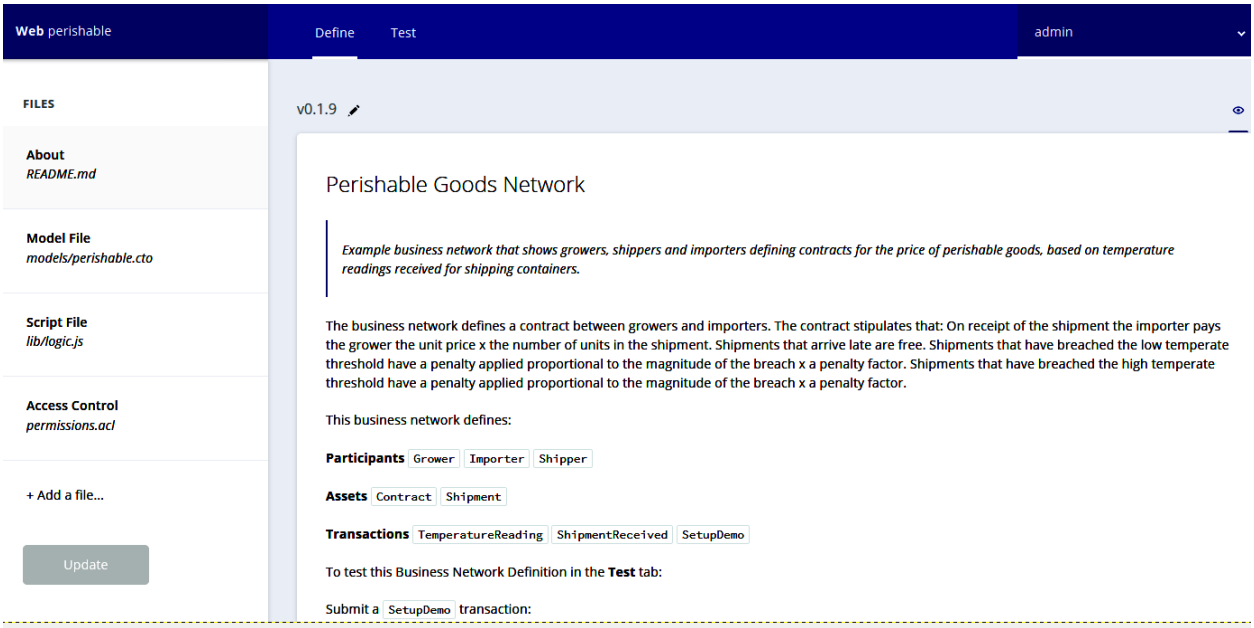
Here we will name our app, see picture above.



We will then select a Business Network Definition, and we choose, **perishable- network,**

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/



Let's look at the Perishable Goods Network

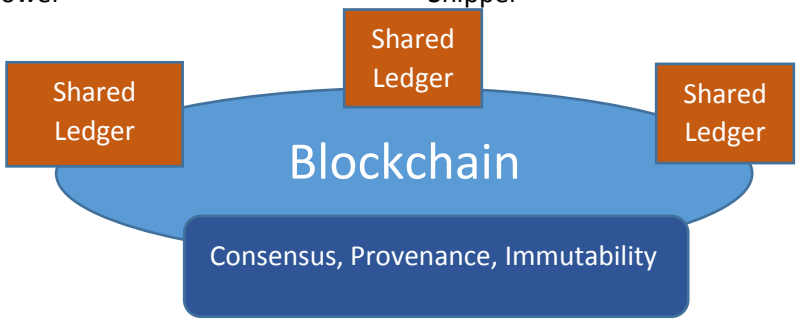
The Hyperledger Composer Modeling Language

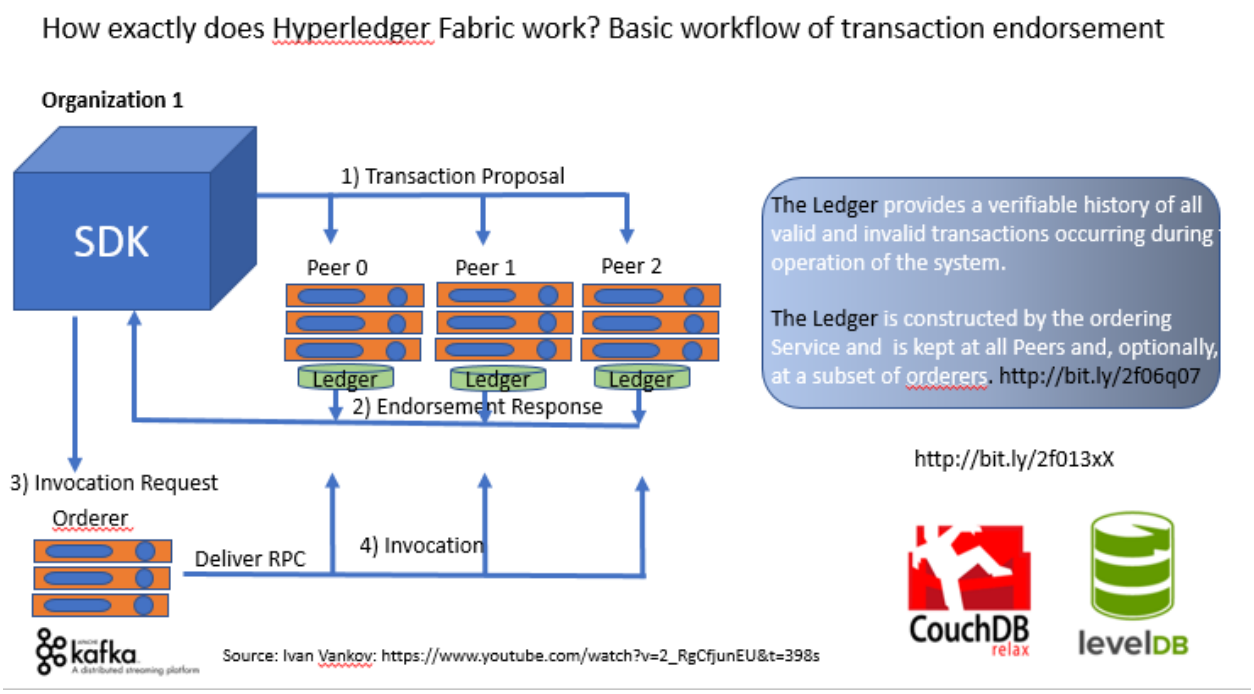
The Perishable Goods Network is written in the Hyperledger Composer Modeling Language

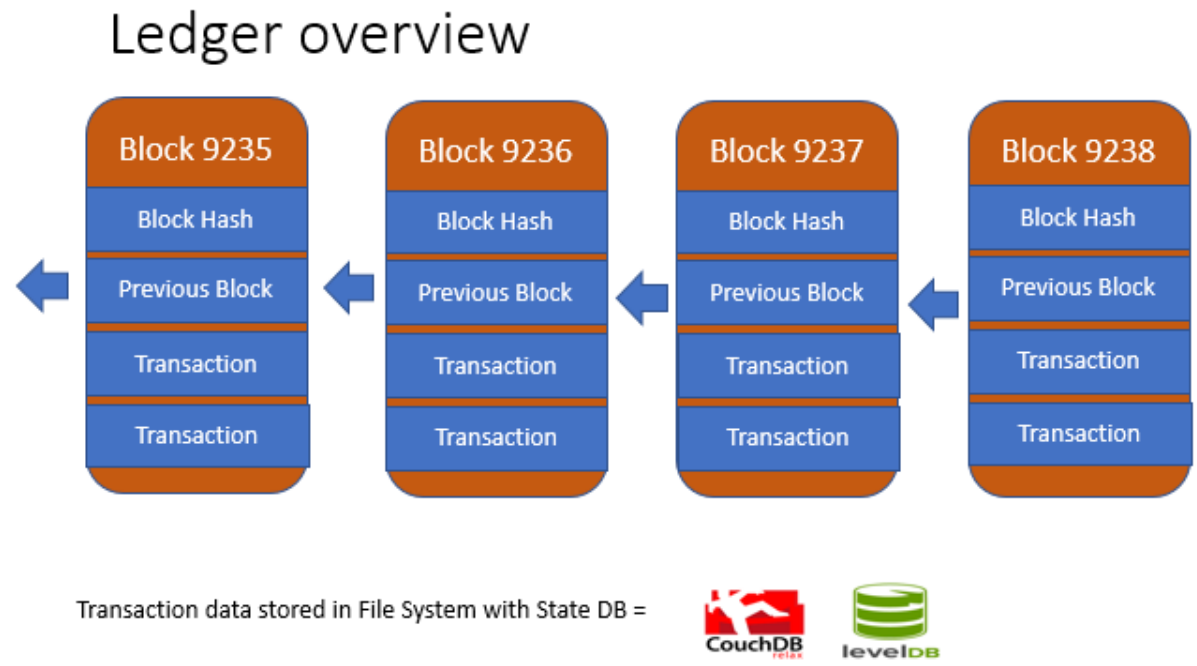
https://hyperledger.github.io/composer/reference/cto_language.html

The Perishable Goods Network architecture:

Grower Shipper Importer







1. Three participants that all access 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 as well as a minimum and maximum penalty factor. Which are all encapsulated in the Smart Contract and visible to all participants in the network, whci have agreed them 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 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
}

/**
  * 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
}
```

perishable-network

Business network description will be previewed here when entered in the basic information section.

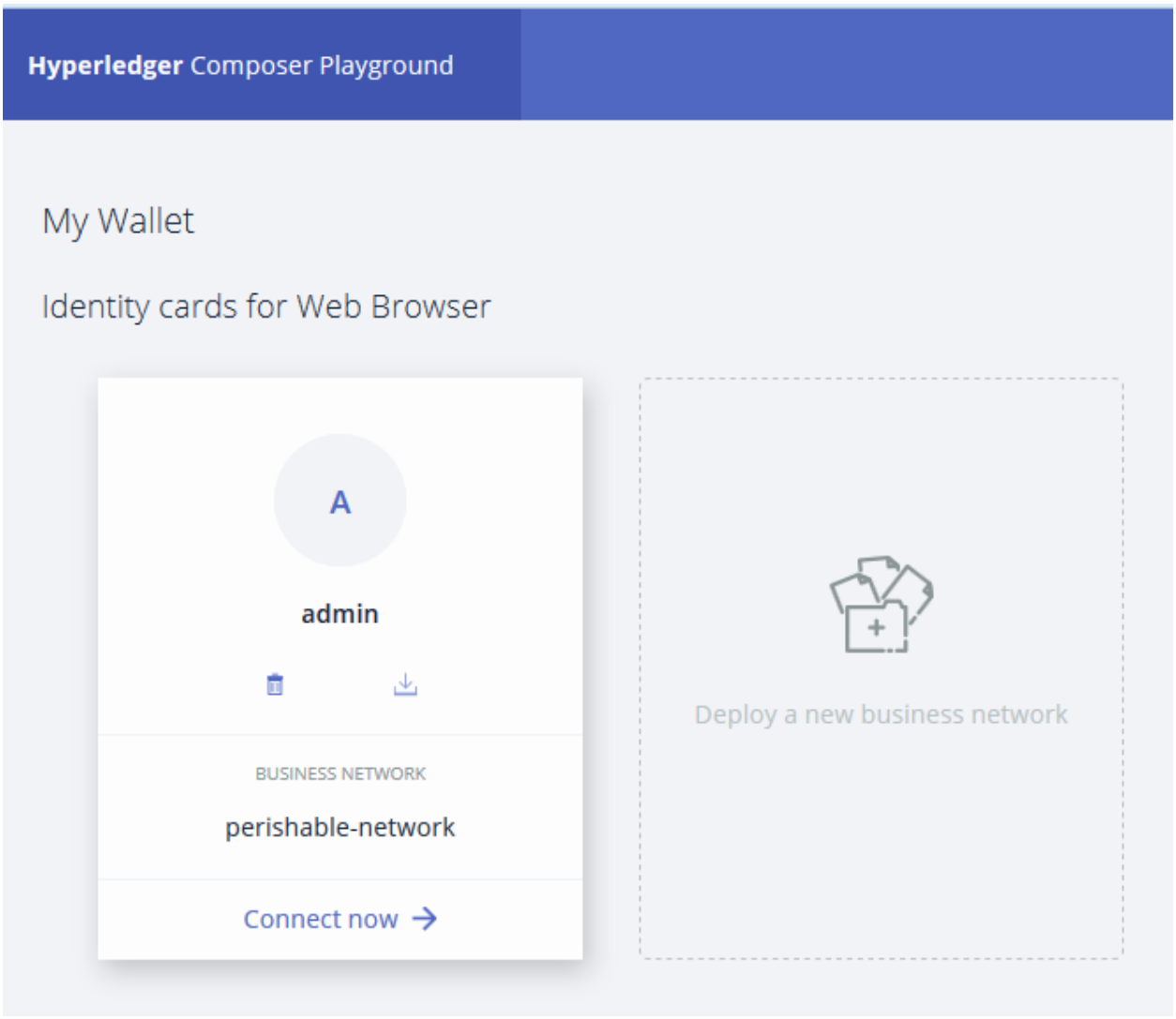
CONNECTION PROFILE

BASED ON
perishable-network

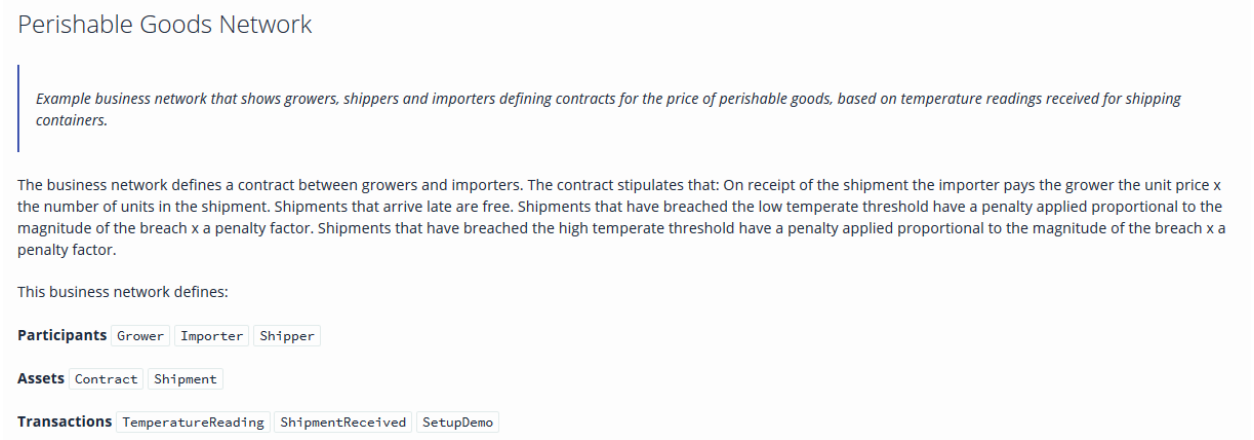
Shipping Perishable Goods Business Network

Contains: 5 Participant Types, 10 Asset Types, and 19 Transaction Types

Deploy



Click on Connect now.



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

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": 0,
      "shipment": "resource:org.acme.shipping.perishable.Shipment#shipmentId:9227"
    }
```

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#shipmentId:2356"
}
```

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!

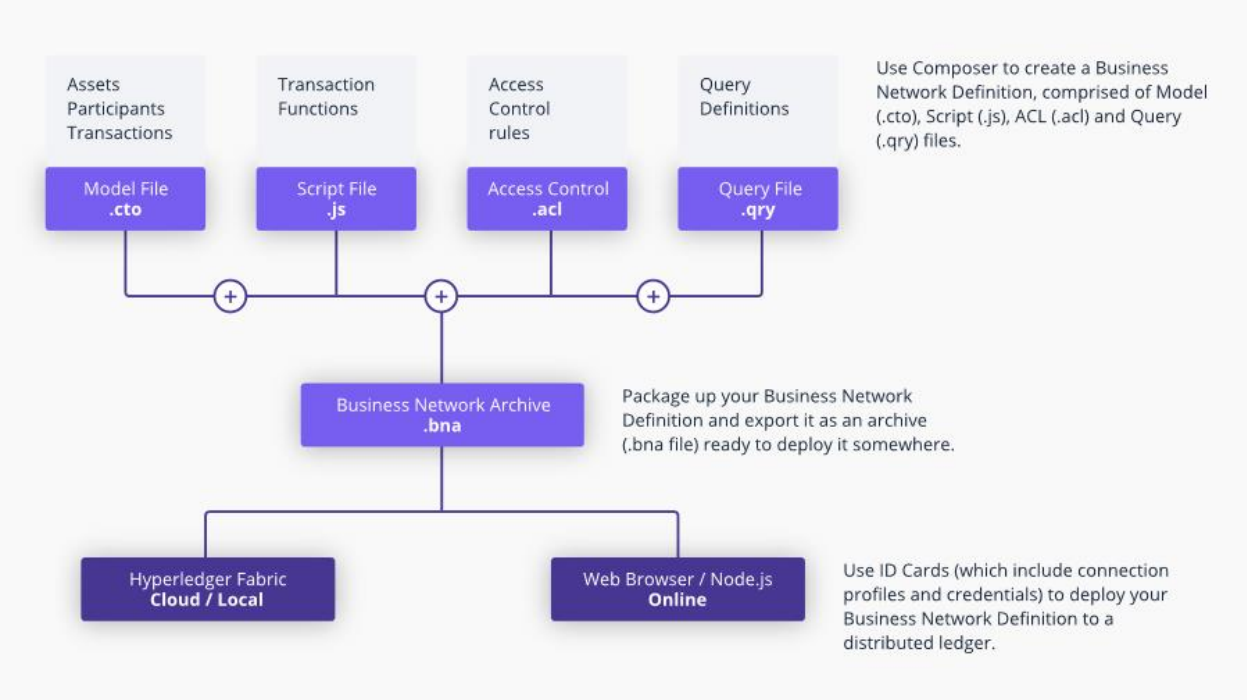
Step 3 Installing the Composer locally

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

Step 4 Where do we go from here?

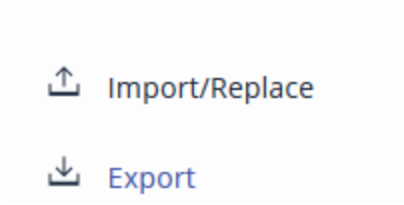
Create your own Business Network Definitions from scratch.

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



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