

Práctica 5. Hyperledger Composer en modo local

José Incera. Marzo 2018

1. Objetivos

- Instalar y utilizar un ambiente de desarrollo local y seguro (Playground) del Hyperledger Composer
- Implementar una red de negocio que modele una cadena de suministro entre distintos actores
- Desplegar la red en un blockchain en la nube

2. Introducción

En esta práctica seguiremos los pasos para desplegar el modelo de una red de negocio en un ambiente local conformado por Hyperledger Composer montado sobre Hyperledger Fabric, una arquitectura robusta para desplegar blockchains entre organizaciones de distintas industrias.

Se dejarán los detalles de Hyperledger Fabric para otra práctica; en ésta nos seguiremos familiarizando con Composer y su lenguaje de modelado (CTO) con una red de negocio un poco más compleja en la que se controla una cadena de suministros de productos perecederos (Plátano, Pera y Café).

Los socios de negocio (los participantes) son productores, importadores (o compradores) y transportistas. Los acuerdos entre las partes estipula las condiciones (las reglas de negocio) en que operarán. Específicamente, las reglas de negocio establecerán bajo qué condiciones se harán pagos por importar los productos predederos.

La práctica está inspirada en las siguientes referencias:

- <https://hyperledger.github.io/composer/installing/development-tools.html>
- <https://www.ibm.com/developerworks/cloud/library/cl-model-test-your-blockchain-network-with-hyperledger-composer-playground/index.html>

En este tutorial vamos a instalar Hyperledger Composer en modo local.

Composer todavía no es soportado en Windows, así que utilizaremos una máquina virtual con

Ubuntu. El Playground se ejecuta en un contenedor Docker.

Si no lo ha hecho ya, instale las herramientas. El apéndice contiene la guía de instalación. **Nota.**
En el laboratorio ya se tienen las herramientas instaladas.

3. Desarrollo

Puede encontrar todo este proyecto como una de las redes de negocio en Bluemix, donde creamos nuestra red en la práctica anterior. En este tutorial trataremos de hacerla paso a paso para conocer un poco más cómo definir una red de negocios en el lenguaje de modelado CTO.

Antes de ejecutar el Hyperledger Composer en modo local con toda la infrestructura de Hyperledger Fabric, vamos a seguir trabajando en modo navegador.

3.1.- Composer local a través de un navegador

Una vez que tiene instalado todo, ejecute localmente el Playground. Se desplegará una página web con la interfaz de usuario, esperando comandos en el puerto 8080

```
cmd> composer-playground
```

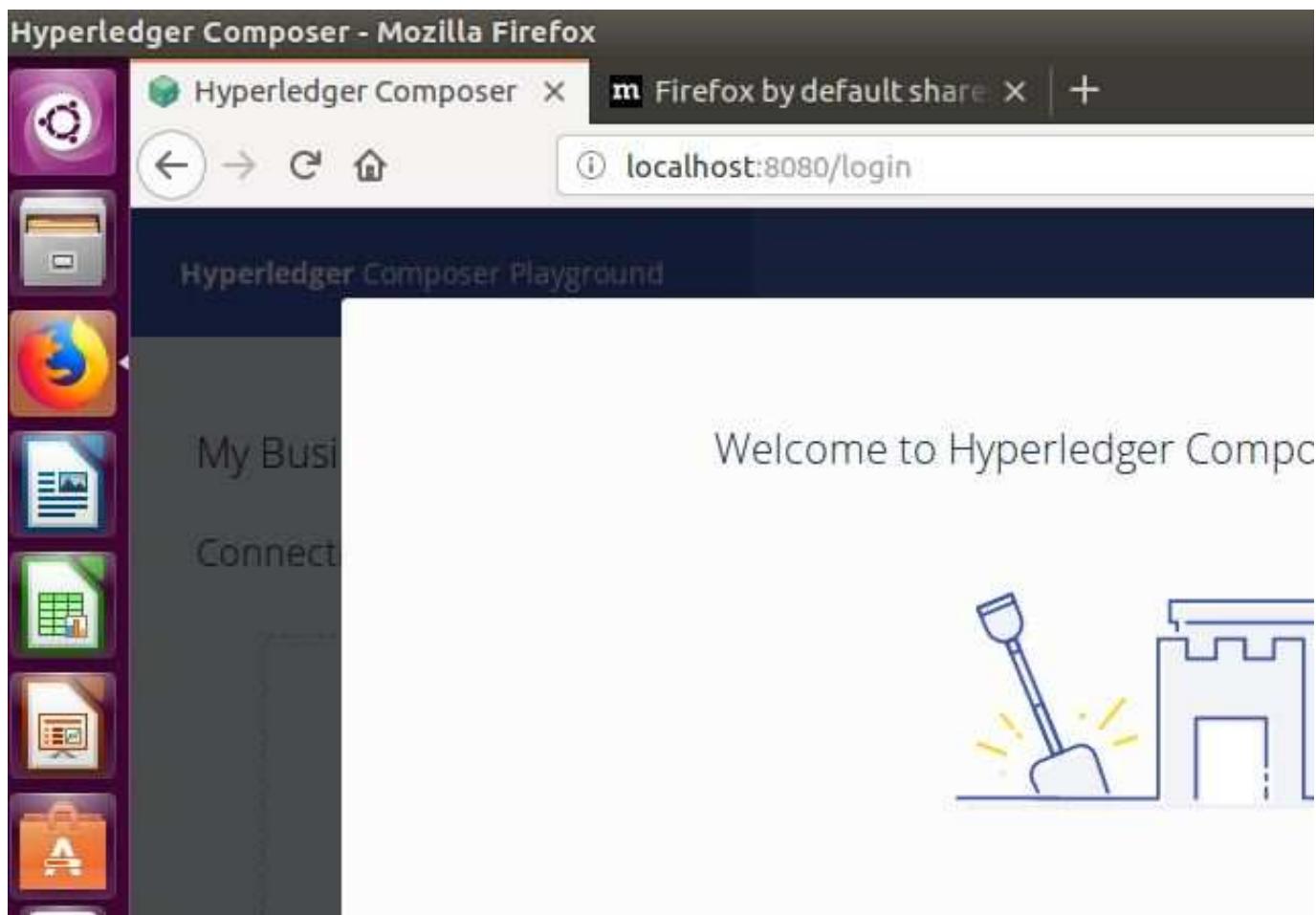


Fig. 1.- Página de bienvenida del Hyperledger Composer en línea

El modelo de nuestra red de negocios se especifica con el lenguaje de modelado y se almacena en un archivo .cto . Tiene definiciones para:

- **namespace.** Un namespace crea una frontera en la que la cobertura de los nombres es única. Cada archivo .cto necesita un namespace.
- **Resource.** Un recurso es cualquiera de los siguientes:
 - *activo*.- Un activo en la red de negocios
 - *participant*.- Un participante en la red
 - *transaction*.- La lógica de negocio
 - *event*.- Una notificación de interés que está ocurriendo en el sistema
 - *enumerated type*.- Un conjunto de valores nombrados
 - *concept*.- Cualquier objeto que se desee modelar y que no sea ninguno de los anteriores

Cada tipo de recurso se corresponde con el modelo con el mismo nombre. Por ejemplo, *asset* se usa para modelar un *Asset* y *participant* un *Participant*.

Un recurso tiene las siguientes propiedades:

- El namespace en el que fue definido

- Un `name` que es único en el `namespace`
 - Si el recurso es un `asset` o `participant`, debe tener un campo identificador, indicado por la palabra reservada `identified by` seguido del nombre del campo
- Opcionalmente puede especificar el tipo de su padre (super type), indicado por la palabra `extends` y el nombre del tipo del padre
- Opcionalmente, la palabra reservada `abstract` si no deseamos que este recurso sea instanciado, en cuyo caso puede ser sólo un super tipo para otros recursos.

En nuestra red de perecederos, un activo de embarque se modela así:

```
/**
 * El embarque tiene un control de temperatura
 * El pago es negociable en función de la temperatura observada
 */
namespace org.acme.cargamento.perecedero
.

.

/**
 * Embarque monitoreado como activo en el ledger
 */
asset Embarque identified by idEmbarque {
    o String idEmbarque
    o TipoProducto tipo
    o StatusEmbarque status
    o Long cuentaUnitaria
    o LecturasTemperatura[] lecturasTemperatura optional
    --> Contract contrato
}
```

Las propiedades de un activo se indican con la letra ‘o’ minúscula. Las propiedades pueden tener un tipo fundamental, como `String` o uno enumerado, como `TipoProducto`, o una transacción como el arreglo `LecturasTemperatura`.

La referencia a `Contract` indicada por el símbolo ‘`-->`’ es una *relationship* y es unidireccional.

Tipos enumerados

Cuando se conoce el conjunto de valores que una propiedad puede tener, conviene modelarlos como *enumerated*, lo que hará su validación más sencilla.

Un ejemplo de declaración de un tipo enumerated es:

```
/**  
 * El perecedero que está siendo embarcado  
 */  
  
enum TipoProducto {  
    o PLATANOS  
    o MANZANAS  
    o PERAS  
    o DURAZNOS  
    o CAFE  
}
```

Concepts

Son las entidades en el modelo que no pueden considerarse asset, participant, transaction o event. Un ejemplo es una dirección:

```
/**  
 * Concept para una dirección simple  
 */  
  
concept Direccion {  
    o String ciudad optional  
    o String pais  
    o String calle optional  
    o String cp optional  
}
```

3.2.- Modelando la red de negocio

Borrar la memoria del host local

Cuando estamos trabajando con el Playground en modo navegador, Composer permite trabajar solamente con un modelo a la vez. Si tenemos otro modelo, tendrá que eliminarse de la memoria local. En condiciones normales Playground removerá el modelo anterior, pero si cometemos errores, conviene empezar desde cero borrando manualmente la memoria local.

Esto varía de según el navegador. Para Chrome se accede a:

```
Settings > Advanced > Content Settings > Cookies > All cookies and site data > localhost
```

... y damos clic en el bote de basura.

3.2.1.- Creación de un nuevo modelo

La plantilla de perecederos ya existe. Búsquela en las plantillas (`perishable-network` en la figura siguiente), llámala `red-lab-perecederos` y de clic en **Deploy**.

En la tarjeta Admin ID de clic en la liga `Connect now`. Si todo está bien, deberá ver una figura como la siguiente:

The screenshot shows the Modeler interface with the following details:

- Web red-lab-perecederos**: The title of the model.
- Define Test**: The tabs at the top.
- FILES**: A sidebar listing files:
 - About**: README.md
 - Model File**: models/perishable.cto
 - Script File**: lib/logic.js
 - Access Control**: permissions.ad
- v0.2.0-20180102082548**: Version number with a edit icon.
- Perishable Goods Network**: The main title of the model.
- Description**: Example business network that shows growers, shippers and importers defining a contract between them.
- Definition**: The business network defines a contract between growers and importers. It specifies a minimum temperature and a penalty for late shipments. Shipments that arrive late are free. Shipments that have breached the high temp limit are charged a penalty.
- Participants**: Grower, Importer, Shipper.
- Assets**: Contract, Shipment.

Fig. 2.- Pantalla de definición de la red de perecederos

Como ya sabemos, el archivo `models/perishable.cto` tiene el modelo de la red y el que está debajo de scripts, `lib/logic.js` tiene los smart contracts que implementan la lógica de negocio.

De clic en el archivo de modelos. Observará que el agricultor y el transportista son ambos hijos del modelo 'Business'.

```
/**  
 * 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 {
}
```

Un contrato de un activo es así:

```

/***
 * 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 penalty factor proportional to the deviation from the min and max
 * negotiated 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
}
```

Siéntase en libertad de analizar el código tanto en perishable.cto como en el de la lógica de negocio.

Cuando lo desee, instancié el model dando clic en `Test` para empezar a jugar con la red.

Web red-lab-perecederos

Define Test

PARTICIPANTS

Grower ID Data

Importer

Shipper

ASSETS

Contract

Shipment

This registry is empty!

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

TRANSACTIONS

All Transactions



Fig. 3.- Pantalla para interactuar con la red de negocio

Vemos los activos y los participantes del lado izquierdo, pero no hay nada en el centro pues todavía no se han creado instancias de ellos.

Para acelerar nuestra interacción con este ejercicio, el ejemplo tiene una función `SetupDemo()` que genera instancias de los activos y participantes, les asigna valores y almacena las instancias en sus respectivos `registries`.

Para ejecutar esta función, damos clic en **Submit Transaction**; en la ventana de diálogo que aparece seleccionamos ``SetupDemo`. Quedará así:

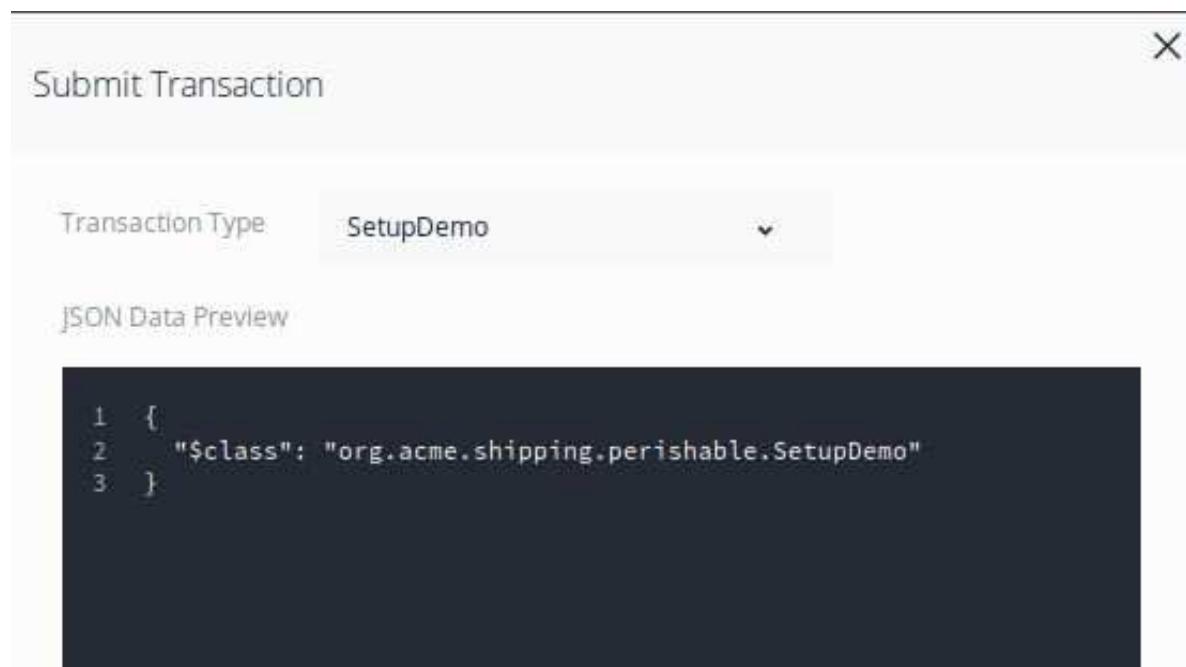


Fig. 4.- Diálogo para enviar una transacción

Seleccione **Grower** en el panel de ASSETS en el menú de la izquierda. Verá que ya está instanciado con ciertos valores. SetupDemo() hizo lo mismo con todos los demás objetos en el modelo.

PARTICIPANTS

Grower
Importer
Shipper

ASSETS

Contract

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

ID	Data
farmer@email.com	{ "\$class": "org.acme.shipping.perishable.Grower", "email": "farmer@email.com", "address": { "\$class": "org.acme.shipping.perishable.Address", "country": "USA" } }

Show All

Fig. 5.- Se verifica que el participante ya ha sido instanciado

3.3.- Transacciones

Hasta ahora hemos hablado y visto activos y participantes. Es momento de trabajar con las transacciones.

La lógica de negocio en el smart contract (o **chaincode** en Hyperledger), generada en `setupDemo()`, estipula las siguientes condiciones:

1. La temperatura dentro del contenedor debe ser de 6 grados celsius todo el tiempo. Si la temperatura está por debajo de un rango acordado (+/- 5 grados), entonces el precio del embarque, que originalmente era de \$0.5 por unidad se reduce \$0.2 por cada grado debajo de la temperatura mínima y \$0.1 por cada grado que rebasa esa temperatura.
2. Si el embarque llega tarde, el contenido es gratuito.

3.4.- Control de acceso

Las reglas de control de acceso se estipulan en el archivo `permissions.acl`. En otro tutorial se trabajará con ellas. Por ahora, simplemente familiarícese con el archivo. Se encuentra en la pestaña **Define** del modelo.

```
/**  
 * 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
}

```

Podrá ver que por ahora no sirven de gran cosa: Otorga permisos de todo a todos. Más adelante pondremos algunas restricciones.

3.5.- Probar el modelo

Ya que el modelo está instanciado, podemos lanzar algunas operaciones sobre él. Esto ejecutará el código JavaScript del smart contract. Veamos un segmento del contrato:

```

// 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

```

Fig. 6.- Un segmento del smart contract

Vamos a simular el siguiente escenario:

1. Las lecturas del sensor de temperatura marcan (en grados celsius):
 1. 5
 2. 7
 3. 3
 4. 1
2. El cargamento llega

La temperatura se checa con la función `temperatureReading()`. Para simular que un sensor en el

mando real envió una lectura, invocamos la función:

1. De clic en **SubmitTransaction**. Asegúrese que la función invocada es `TemperatureReading`
2. Cambie la lectura de 'centigrade' de 0 a 5 en la ventana **JSON Data Preview** (el primer valor que queremos notificar)
3. Asegúrese que el identificador del embarque es `SHIP_001`
4. De clic en **Submit**
5. Repita para las demás lecturas
6. También ejecute la transacción correspondiente a la llegada del barco. No olvide cambiar el identificador a `SHIP_001`.

Al terminar, puede ver que el agricultor ya tiene algo de dinero y que corresponde a la paga que le toca.

4. Refinando la red de negocio

En esta sección aprenderemos cómo se puede modificar una red de negocio ya definida. También aprenderemos a atender **Eventos** en el modelo.

Supongamos que se tiene un sensor GPS en los contenedores que pueden proporcionar la ubicación del barco que está transportando el contenedor. Nos interesa agregar las lecturas de este sensor a nuestro modelo de red. También se nos pide enviar un evento cuando el barco haya llegado a su destino.

Para poder modificar el código, ¡primero hay que tenerlo! Lo vamos a copiar de un repositorio en github de un investigador de IBM.

Cree una carpeta, por ejemplo `HplCompTst` y clone el repositorio:

```
cmd> mkdir HplCompTst
cmd> cd HplCompTst
cmd> git clone https://github.com/makotogo/developerWorks
Cloning into 'developerWorks' ...
...
cmd> ls
developerWorks
```

Muy bien. Si revisa el contenido de la carpeta `developerWorks`, encontrará el código de varios proyectos. El que nos interesa es `perishable-networks`.

4.1 Modificar la definición de la red

Empecemos por agregar el sensor GPS. Con el editor de su preferencia (si no ha instalado ninguno, utilice gedit) abra el archivo perishable.cto que se encuentra en la carpeta **developerWorks/perishable-network/models**.

Agregue un nuevo enum para representar las cuatro coordenadas geográficas justo debajo del enum ShipmentStatus :

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

Es importante acotar los datos entregados por un conjunto de coordenadas GPS. Este enum ayudará a restringirlos para facilitar su validación.

Cada vez que se haga una lectura GPS, ésta se graba en el blockchain como una transacción. Por lo tanto, debemos agregar una transacción al modelo. Justo debajo de la transacción TemperatureReading añada una nueva llamada GpsReading :

```
/**  
 * 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  
}
```

Además de longitud y latitud, una lectura de GPS proporciona fecha y hora.

Para que la transacción se pueda almacenar en el blockchain, su información *debe ser parte de un Activo*. Dado que la lectura del sensor GPS puede considerarse parte de un embarque, es

natural agregar estas lecturas al Activo Embarque tal y como se hizo para las lecturas de temperatura. Agregue la línea 7 resaltada en la figura, a su código:

```
1 asset Shipment identified by shipmentId {  
2   o String shipmentId  
3   o ProductType type  
4   o ShipmentStatus status  
5   o Long unitCount  
6   o TemperatureReading[] temperatureReadings optional  
7   o GpsReading[] gpsReadings optional  
8   --> Contract contract  
9 }
```

*Fig. 7.- Segmento del código Asset Shipment que debe ser modificado

Finalmente, **agregue dos nuevos eventos** al modelo, uno cuando el umbral de temperatura ha rebasado la tolerancia acordada en el contrato, y el otro cuando el barco ha llegado al puerto destino.

```
/**  
 * 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  
}
```

4.2 Agregar *chaincode*

Hasta ahora hemos modelado el sensor GPS y la transacción que permitirá agregar las lecturas a blockchain. Para que esto ocurra, es necesario escribir el código de la transacción (el smart contract). Como recordará, en Composer esto se hace escribiendo una rutina JavaScript (en el archivo lib/logic.js) que se ocupe de registrar la lectura en blockchain. Abra el archivo, tendrá que hacerle varias modificaciones.

En primer lugar, se debe modificar la función `temperatureReading` para que se dispare un evento (`TemperatureThresholdEvent` en el código) cuando se excedan los umbrales contratados. En el siguiente segmento de código se resaltan las líneas que deben agregarse a la función:

```
function temperatureReading(temperatureReading) {

    var shipment = temperatureReading.shipment;
    var NS = "org.acme.shipping.perishable";

    /**** SE AGREGAN LAS SIGUIENTES DOS LÍNEAS ****/
    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];
    }

    /**** SE AGREGAN LAS SIGUIENTES OCHO LINEAS ****/
    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);
    }
    /**** HASTA AQUI ****/

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

Ahora debemos agregar una nueva función que se ocupe de las transacciones generadas por las lecturas del GPS. El código es el siguiente. **Nota: Debe añadirse el comentario porque tiene anotaciones importantes @param y @transaction.**

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

}

El `chaincode` para esta transacción almacena esta lectura en el arreglo `GPSReadings` del activo `Shipment`. Después revisa si esta lectura corresponde al puerto destino; si es así, emite un evento `ShipmentPort`. Finalmente, se actualiza el blockchain con el estado actual del embarque.

5. Desplegar la red en la nube IBM Bluemix

Como se mencionó en la introducción, el despliegue de nuestra red de negocio en Hyperledger Fabric se dejará para la siguiente práctica. Con el fin de verificar que los cambios hechos al código funcionan correctamente, desplegaremos nuestro modelo en el Playground en línea que utilizamos en la práctica anterior (en <https://composer-playground.mybluemix.net/>).

El archivo que se envía a la nube es un *Business Network Archive (BNA)* que se crea al hacer un build del proyecto. Este archivo se importará desde el Playground. Para poder crear este archivo, debemos hacer todavía algunos cambios a nuestro código.

5.1 Modificar chaincode para desplegar mensajes

En las funciones `temperatureReading` y `gpsReading` en el archivo `lib/logic.js`, se debe incluir una línea para desplegar el mensaje del evento en la consola. Esto se hace en las dos funciones justo después de haber formado el mensaje. En los siguientes fragmentos de código se muestra la línea que debe ser agregada en cada función:

```
***** en temperatureReading ****/  
...  
...  
  
    temperatureEvent.message = 'Temperature threshold violated! Emitting Temperature  
Event for shipment: ' + shipment.$identifier;  
    console.log(temperatureEvent.message);  
...  
...  
***** en gpsReading ****/  
...  
...  
  
var message = 'Shipment has reached the destination port of ' + PORT_OF_NEW_YORK;  
shipmentInPortEvent.message = message;  
console.log(message);
```

...

...

Con estas modificaciones, cuando se ejecuten las transacciones `temperatureReading` y `gpsReading` y éstas generen eventos, se podrán ver éstos en la consola de JavaScript.

5.2 Crear el archivo BNA

El `build` para generar el *Business Network Archive* se genera al ejecutar `npm install` en la línea de comandos. Desde una terminal colóquese en la carpeta `perishable-network` y lance el comando:

```
cmd> cd developerWorks/perishable-network/
cmd> npm install

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

Creating Business Network Archive
...
...
Command succeeded
```

5.3 Importar el modelo

Vayamos al Playground en línea utilizado en la práctica anterior. Por si no lo recuerda, se encuentra en <https://composer-playground.mybluemix.net/>. Al entrar, debemos encontrar la pantalla de bienvenida de la *Fig. 1*.

Recuerde que si no ve la pantalla de bienvenida, debe limpiar la memoria del navegador (Settings > Advanced > Content Settings > Cookies > All cookies y site data > localhost, en Chrome).

Como en la práctica anterior, de clic en **Deploy a new business network** para crear una nueva red.

Hyperledger Composer Playground

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

Fig. 8.- Página inicial para desplegar una nueva red o seleccionar una ya creada

En la siguiente pantalla, de clic en **Drop here to upload or browse**.

Hyperledger Composer Playground

Describe what your Business Network will be used for:

Give the network admin card that will be created a name eg. admin@basic-sample-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

Samples on npm

Drop here to upload or browse

Fig. 9.- Seleccionar una red de negocio o cargar (importar) una nueva

Se abrirá una ventana de diálogo; navegue hasta la carpeta `perishable-network/dist` y localice el archivo `perishable-network.bna`. De clic en **Open**. Una vez cargada, de clic en **Deploy** para desplegar el modelo.

5.4 Interactuando con el modelo

Esta parte ya debe resultarle familiar pues es lo que hicimos en la primera parte de esta práctica. De clic en **Connect now** para iniciar.

Recuerde que empezamos por instanciar entidades en el modelo con ayuda de la transacción `SetupDemo`. Invóquela desde el menú `Test/Submit Transaction`.

Ahora abra la consola `JavaScript`. En Chrome, seleccione `View > Developer > JavaScript console`. En Firefox, la consola del navegador atrapa los mensajes de JavaScript y se activa con la combinación `Ctrl-Shift-J`.

Envíe una transacción `TemperatureReading` para el embarque `SHIP_001` con un valor de 12 grados centígrados.

Submit Transaction

Transaction Type TemperatureReading

JSON Data Preview

```
1  {
2    "$class": "org.acme.shipping.perishable.TemperatureReading",
3    "centigrade": 11,
4    "shipment":
5      "resource:org.acme.shipping.perishable.Shipment#SHIP_001"
```

Optional Properties

Just need quick test data? [Generate Random Data](#)

Cancel

Submit

Fig. 9.- Transacción con valor de temperatura fuera de umbrales

Este valor excede las condiciones contractuales, por lo que se dispara un evento. En la consola de JavaScript (o del navegador en Firefox) verá el siguiente mensaje: *Temperature threshold violated! Emitting TemperatureEvent for shipment: SHIP_001.*

Para terminar, envíe una transacción `GpsReading` para el embarque `SHIP_001` con los siguientes parámetros, que corresponden al puerto destino:

```
readingTime = 171000
readingDate = 20180328
latitude = 40.6840
latitudeDir = N
longitude = 74.0062
longitudeDir = W
```

Si efectivamente estas coordenadas coinciden con el puerto destino, se dispara un evento que podrá comprobar con el siguiente mensaje en la consola: *Shipment has reached the destination port of /LAT:40.6840N/LONG:74.0062W.*

¡Felicitaciones. Se está volviendo un experto en Hyperledger Compose!

Apéndice. Instalación del ambiente

Hyperledger Composer y Hyperledger Fabric no pueden ejecutarse directamente en Windows. Se ejecutan adentro de un contenedor de *Docker* y parece ser que éste tiene algunos conflictos con Windows. Por ello, instalaremos el ambiente con una máquina virtual.

Si tiene un ambiente Linux, puede brincar a la sección A3.

A1. Paquetes

Si aún no lo tiene instalado, descargue e instale Virtualbox desde esta liga:

<https://www.virtualbox.org/wiki/Downloads>

También necesitamos una imagen de Ubuntu. La puede descargar de esta liga:

<https://www.ubuntu.com/download/desktop>. Esta imagen es de 1.5 GB... tenga paciencia!

A2. Instalación de Ubuntu

La guía está tomada de esta liga: <https://www.lifewire.com/run-ubuntu-within-windows-virtualbox-2202098>

1. Instalar VirtualBox

Desde la carpeta de descargas, dar doble clic al instalador.

1. Dar **Next** en la pantalla de bienvenida
2. Dejar los componentes por default que recomienda
3. **Next** para ir a la configuración personalizada
4. Seleccione la carpeta donde desee que VirtualBox aparezca y de clic en **Next**
5. Decida si desea un ícono en el escritorio, **Next**
6. Clic en **Install** para instalar el ambiente. Va a tardar un poco y quizás solicite permisos para modificar el ambiente y para apagar el firewall.
7. Seleccione la opción para iniciar VirtualBox cuando se haya terminado la instalación

8. Clic en **Finish** para completar la instalación.

2. Configuración en VirtualBox

Si no se inició automáticamente, lance VirtualBox para crear una nueva máquina virtual (VM).

1.- Elija un nombre descriptivo en **Name**. Le sugerimos *HyperledgerLab*

2.- Seleccione **Linux** como Type

3.- Seleccione **Ubuntu** como Version. NOTA: ASEGÚRESE DE HABER ELEGIDO 64-bit o 32-bit
DEPENDIENDO DE LA ARQUITECTURA DE SU HOST

4.- Clic en **Next**

5.- Asigne memoria a la VM. Nunca menos de lo recomendado, y nunca toda la memoria de su host pues debe dejar espacio para que Windows se ejecute. Si tiene 4GB de RAM o más, opte por 2GB

6.- Seleccione **Create a virtual hard drive now**

7.- Clic en **Create**

8.- Se le preguntará el tipo. Seleccione **VDI**. Clic **Next**

9.- Si tiene suficiente espacio en disco, seleccione *fixed size hard drive*, de lo contrario, seleccione *dynamic sized*

10.- Se le preguntará cuánto espacio en disco desea asignar a la VM. No elija menos del valor por default. Si puede, elija al menos 20 GB

11.- Seleccione dónde quiere guardar la VM y el tamaño. Clic en **Create**

Una vez que la máquina virtual se ha creado, arránquela dando clic en **Iniciar**.

12.- La primera vez que la inicia, deberá elegir un disco de inicio. De clic en el ícono de folder, navegue hasta donde guardó la **Imagen Ubuntu ISO** y de clic en **Start**.

13.- Ahora arranca Ubuntu con un mensaje de bienvenida y pregunta por el idioma de instalación y si se desea probar o instalar Ubuntu. **Seleccione Install**

14.- Si tiene una buena conexión a Internet, seleccione la opción **install updates as you go**. Si puede evitarlo, **NO** seleccione instalar software de terceros. Siempre podrá hacerlo después

15.- Ahora le preguntará cómo particionar el disco duro virtual. Seleccione **Erase disk and install Ubuntu**. No tema, estamos hablando del archivo virtual

16.- Clic en **Install Now**. Inicia la instalación copiando los archivos al disco virtual

17.- Elija localidades (México City) **Continue**. También elija el teclado en función de su lenguaje (Español, Latam). Clic **Continue**

18.- En la pantalla *Who are you* introduzca su nombre, una contraseña (repítala), si desea entrar automáticamente o requiere de *login* (seleccione esto último). También si requiere de una carpeta cifrada (No). Clic **Continue**

Cuando haya terminado la instalación, se le pedirá que reinicie la VM. Hágalo.

19.- Le conviene instalar las **Guest Additions** de esta liga

<https://www.virtualbox.org/manual/ch04.html>

19.1.- Seleccione **Devices**

19.2.- Seleccione **Install Guest Additions** del menu donde corre la VM

Cuando haya terminado, reinicie la máquina virtual y listo. Ya tiene un ambiente Ubuntu funcional.

A3. Instalación de Hyperledger Composer y Fabric

La instalación de los ambientes se tomó de la siguiente guía:

<https://hyperledger.github.io/composer/installing/installing-prereqs.html>

En Linux, siga estos consejos:

- Regístrese como usuario normal, no como root.
- No use `su` para hacerse root
- Al instalar los requisitos, use `curl` y después `unzip` usando `'su'`.
- Ejecute `prereqs-ubuntu.sh` como usuario normal; quizás solicite la contraseña de root
- No use `npm` con sudo o su
- Evite instalar node globalmente como root

A3.1 Instalar requisitos

En Ubuntu podemos descargar los requisitos con los siguientes comandos:

```
cmd> sudo apt install curl
cmd> curl -O https://hyperledger.github.io/composer/prereqs-ubuntu.sh

cmd> chmod u+x prereqs-ubuntu.sh
cmd> ./prereqs-ubuntu.sh
```

A3.2 Instalar componentes

1.- Herramientas CLI

Se tienen varias herramientas útiles en línea de comandos. La más importante es `composer-cli` que tiene todas las operaciones esenciales, por lo que esa será la primera en instalarse. Después se instalarán otras que no son esenciales pero serán útiles durante el tutorial.

```
cmd> sudo npm install -g composer-cli  
cmd> sudo npm install -g composer-rest-server  
cmd> sudo npm install -g generator-hyperledger-composer  
cmd> sudo npm install -g yo
```

2.- Instalar Playground

Instalaremos una interfaz de usuario local como la que se utilizó en la práctica anterior. Es muy útil para probar nuestra red de negocio sin tener que utilizar Hyperledger Fabric.

```
cmd> npm install -g composer-playground
```

3.- Instalar un ambiente de desarrollo

Estrictamente hablando, no es necesario para este taller, por lo que **NO SE INSTALARÁ POR AHORA**

Se puede instalar cualquier IDE. La guía recomienda VSCode que se puede descargar de esta liga: [

Install VSCode from this URL: <https://code.visualstudio.com/download> pues ya tiene la sintaxis de CTO.

Abra VSCode, vaya a Extensions y busque e instale la extensión para Hyperledger Composer extension del Marketplace.

4.- Hyperledger Fabric

Vamos a instalar una implementación local para poder desplegar nuestras reglas de negocio. Conviene crear un directorio para guardar ahí el archivo que recibiremos:

```
cmd> mkdir ~/fabric-tools  
cmd> cd ~/fabric-tools  
  
cmd> curl -O https://raw.githubusercontent.com/hyperledger/composer-tools/master/packages/fabric-dev-servers/fabric-dev-servers.zip  
  
cmd> unzip fabric-dev-servers.zip
```

Ahora utilizaremos algunos de los scripts que acabamos de obtener para descargar el Hyperledger Fabric:

```
cmd> cd ~/fabric-tools  
cmd> sudo ./downloadFabric.sh
```

¡Felicitaciones! Ya tiene todo lo que necesita para trabajar con Hyperledger Composer y Fabric!