Capítulo 4: Construye y prueba la red

Aprendiendo Bluemix y Blockchain

Bob Dill, IBM Distinguished Engineer, CTO Global Technical Sales
David Smits, Senior Certified Architect, IBM Blockchain

El plan: capítulos de 30 minutos con una o dos horas de práctica

Capítulo 1	¿Qué es Blockchain? Visión general del concepto y la arquitectura
Capitulo 2	¿Cuál es la historia que vamos a construir?
Capítulo 2.1	Arquitectura para la historia
Capítulo 3	Configurar el entorno de desarrollo Hyperledger Fabric V1 local
Capítulo 4	**Construye y prueba la red**
Capítulo 5	Experiencia de usuario de administración
Capítulo 6	Soporte del comprador y experiencia del usuario
Capítulo 7	Soporte del vendedor y experiencia del usuario
Capítulo 8	Soporte de proveedores y experiencia del usuario
Capítulo 9	Soporte del remitente y experiencia del usuario
Capítulo 10	Soporte de la compañía financiera y experiencia del usuario
Capítulo 11	Combinando para la demostración
Capítulo 12	Eventos y automatización para demostración

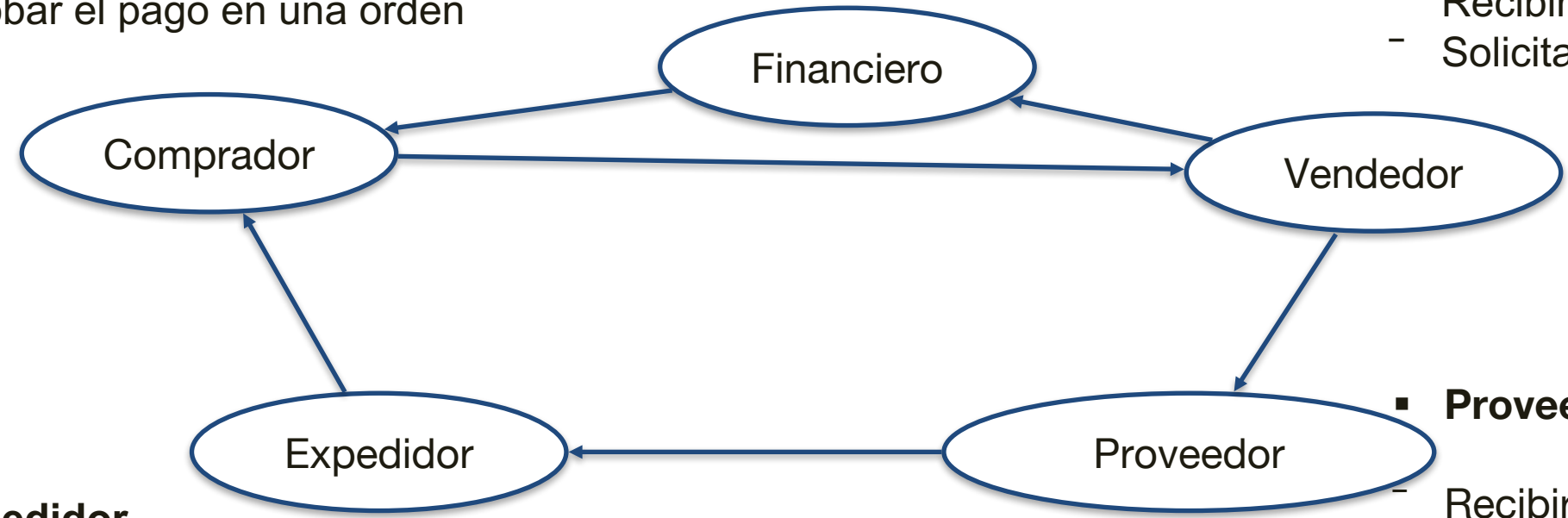
¿Quiénes son los participantes y qué pueden hacer?

Define los miembros de la red, las transacciones y quién debe aprobar las transacciones

- **Comprador**
  - Crear una orden
  - Enviar una orden
  - Recibir un envío
  - Disputa una orden
  - Aprobar el pago en una orden

- **Finance Company**
  - Recibir solicitud de pago
  - Aprobar y pagar
  - Recibir notificación de disputa
  - Resolver disputa

- **Vendedor**
  - Aceptar una orden
  - Enviar un pedido a un proveedor
  - Recibir notificación de entrega
  - Solicitar pago



- **Expedidor**
  - Recibir solicitud de entrega
  - Notificación posterior a la entrega

- **Proveedor**
  - Recibir una solicitud de artículo
  - Emitir, resolver pedido pendiente
  - Solicitar entrega
  - Recibir notificación de entrega

Definir miembros

- En nuestra red simple, los miembros tienen:
  - Un nombre de compañía
  - Un identificador, que implementaremos como una dirección de correo electrónico

- Definimos un tipo abstracto de miembro con un solo campo llamado "companyName"

```
18    namespace composer.base
19
20    abstract participant Member {
21      o String companyName
22    }
```

- Luego, ampliamos este tipo abstracto para cada uno de los tipos de miembros.

```
participant Buyer identified by buyerID extends Member{
    o String buyerID
}
```

- El objetivo es introducir tipos abstractos y nuestra capacidad para separar diferentes tipos de definiciones en archivos separados, para fines de mantenimiento, al tiempo que combinamos todo fácilmente cuando terminamos.

Definiendo activos

- El único activo del que trataremos en este tutorial es un "Pedido", que se define como se muestra aquí.
- Muchos de los campos son para almacenar fechas, por lo que podemos decir qué sucedió cuando. Las fechas y razones se actualizan a través de transacciones, que estarán limitadas por el participante.
- Tenga en cuenta los corchetes, estos denotan matrices
- Tenga en cuenta las flechas (- ->), que denotan referencias a otros, definido previamente, las clases de red.

```
asset Order identified by orderNumber {
    o String orderNumber
    o String[] items
    o String status
    o Integer amount
    o String created
    o String bought
    o String ordered
    o String dateBackordered
    o String requestShipment
    o String delivered
    o String disputeOpened
    o String disputeResolved
    o String paymentRequested
    o String orderRefunded
    o String paid
    o String[] vendors
    o String dispute
    o String resolve
    o String backorder
    o String refund
    --> Buyer buyer
    --> Seller seller
```

Definiendo transacciones

- Las transacciones usan el mismo lenguaje de modelo que los activos y los miembros.

- Aquí nombraremos una transacción e identificaremos qué debe acompañar una solicitud para que se procese esta transacción
- Esta es una clase de transacción

- Se llama CreateOrder

- Tiene un campo (cantidad entera)

- Se refiere a otras 3 instancias
- Una orden
- Un comprador
- Un vendedor

```
transaction CreateOrder {
  o Integer amount
  --> Order order
  --> Buyer buyer
  --> Seller seller
}
```

Tomando esta información, defina lo siguiente…

- Miembros:
  - Comprador, Vendedor, Proveedor, Remitente, FinanceCo
- Bienes:
  - Orden
- Actas:
  - CreateOrder, Buy, OrderFromSupplier, RequestShipping, Deliver, BackOrder, Dispute, Resolve, Request Payment, Pay, Refund

Echemos un vistazo a la red

- Paso 1, actualiza los archivos del modelo
- Paso 2, crear y archivar y desplegarlo
- Paso 3, carga el compositor y pruébalo

- Paso 1, las respuestas están en la carpeta Documentos / respuestas
- Paso 2, ejecuta el siguiente comando desde la carpeta Chapter04
  - **buildAndDeploy**

- Paso 3, ve a:
  - Importar el archivo de modelo de **`Chapter04/network/dist/zerotoblockchain-network.bna`**
  - Pruebe el modelo
  - Notarás en la inspección que no pasa mucho con el objeto Orden, eso es lo siguiente

Escribir el código para implementar las transacciones

- Cada transacción necesita implementar lógica. Por ejemplo, la transacción Crear orden existe para permitir que un comprador cree un pedido y lo guarde antes de enviarlo a un vendedor. El código se muestra a la derecha.

- Puede ver que la definición de clase (abajo a la derecha) incluye un enlace al Pedido, el Importe del pedido y el Comprador. En este código de transacción, la información del vendedor no se utiliza, ya que aún no se ha realizado el pedido con el vendedor

```
/**
 * create an order to purchase
 * @param {org.acme.Z2BTestNetwork.CreateOrder} purchase - the order to be processed
 * @transaction
 */
function CreateOrder(purchase) {
    purchase.order.buyer = purchase.buyer;
    purchase.order.amount = purchase.amount;
    purchase.order.created = new Date().toISOString();
    purchase.order.status = "Order Created";
    return getAssetRegistry('org.acme.Z2BTestNetwork.Order')
        .then(function (assetRegistry) {
            return assetRegistry.update(purchase.order);
        });
}
```

```
transaction CreateOrder {
  o Integer amount
  --> Order order
  --> Buyer buyer
  --> Seller seller
}
```

Escribir código para probar las transacciones

- Estamos utilizando el servicio mocha para probar esta aplicación, el código se ve así:
- Exploraremos este código de forma interactiva en un momento
- Cuando terminemos, podemos decirle a npm que pruebe lo que hemos creado, que debería ofrecer resultados como los siguientes:

```
Finance Network
  #createOrder
    ✓ should be able to create an order (82ms)
  #issueBuyRequest
    ✓ should be able to issue a buy request (40ms)
  #issueOrderFromSupplier
    ✓ should be able to issue a supplier order (50ms)
  #issueRequestShipment
    ✓ should be able to issue a request to ship product (47ms)
  #issueDelivery
    ✓ should be able to record a product delivery (39ms)
  #issueRequestPayment
    ✓ should be able to issue a request to request payment for a product (58ms)
  #issuePayment
    ✓ should be able to record a product payment (48ms)
  #issueDispute
    ✓ should be able to record a product dispute (63ms)
  #issueResolution
    ✓ should be able to record a dispute resolution (48ms)
  #issueBackorder
    ✓ should be able to record a product backorder (53ms)


10 passing (1s)
```

```
describe('#createOrder', () => {

    it('should be able to create an order', () => {
        const factory = businessNetworkConnection.getBusinessNetwork().getFactory();

        // create the buyer
        const buyer = factory.newResource(NS, 'Buyer', buyerID);
        buyer.companyName = 'billybob computing';

        // create the seller
        const seller = factory.newResource(NS, 'Seller', sellerID);
        seller.companyName = 'Simon PC Hardware, Inc';

        // create the order
        let order = factory.newResource(NS, 'Order', orderNo);
        order = createOrderTemplate(order);
        order = addItems(order);
        order.orderNumber = orderNo;

        // create the buy transaction
        const createNew = factory.newTransaction(NS, 'CreateOrder');

        order.buyer = factory.newRelationship(NS, 'Buyer', buyer.$identifier);
        order.seller = factory.newRelationship(NS, 'Seller', seller.$identifier);
        createNew.order = factory.newRelationship(NS, 'Order', order.$identifier);
        createNew.buyer = factory.newRelationship(NS, 'Buyer', buyer.$identifier);
        createNew.seller = factory.newRelationship(NS, 'Seller', seller.$identifier);
        createNew.amount = order.amount;
        // the buyer should of the commodity should be buyer
        //order.buyer.$identifier.should.equal(buyer.$identifier);
        order.status.should.equal('Order Created');
        order.amount.should.equal(orderAmount);
        createNew.amount.should.equal(orderAmount);
        createNew.order.$identifier.should.equal(orderNo);
```

Invocar el compositor-resto-servidor

- Del Capítulo04
- Ejecute el siguiente comando:
  buildAndDeploy
- Esto cargará tu red completa en la ventana acoplable
- Ejecute el siguiente comando
  ./start_rest_server.sh

```
[? Enter your Fabric Connection Profile Name: hlfv1
[? Enter your Business Network Identifier : zerotoblockchain-network
[? Enter your Fabric username : admin
[? Enter your secret: adminpw
 ? Specify if you want namespaces in the generated REST API: always use namespaces
[? Specify if you want the generated REST API to be secured: No
[? Specify if you want to enable event publication over WebSockets: Yes
[? Specify if you want to enable TLS security for the REST API: No
```

- Cuál aplicará las siguientes respuestas:
- Ir a **`localhost:3000/explorer`** e inspeccionar y probar sus nuevas API REST

El plan: capítulos de 30 minutos con una o dos horas de práctica

Capítulo 1	¿Qué es Blockchain? Visión general del concepto y la arquitectura
Capitulo 2	¿Cuál es la historia que vamos a construir?
Capítulo 2.1	Arquitectura para la historia
Capítulo 3	Configurar el entorno de desarrollo Hyperledger Fabric V1 local
Capítulo 4	**Construye y prueba la red**
Capítulo 5	Experiencia de usuario de administración
Capítulo 6	Soporte del comprador y experiencia del usuario
Capítulo 7	Soporte del vendedor y experiencia del usuario
Capítulo 8	Soporte de proveedores y experiencia del usuario
Capítulo 9	Soporte del remitente y experiencia del usuario
Capítulo 10	Soporte de la compañía financiera y experiencia del usuario
Capítulo 11	Combinando para la demostración
Capítulo 12	Eventos y automatización para demostración