

Fundamentos básicos do Hyperledger Composer, Parte 1: Modele e teste sua rede de blockchain

Comece a usar o blockchain de maneira rápida e fácil no Hyperledger Composer Playground

J Steven Perry

Principal Consultant

Makoto Consulting Group, Inc.

08/Fev/2018

O Hyperledger Composer é uma ferramenta para desenvolver redes de negócios de blockchain e criar protótipos de aplicativos de blockchain rapidamente. Este tutorial orienta como começar a usá-lo.

Desenvolva seu primeiro aplicativo de blockchain rapidamente!

Com o Hyperledger Composer, é possível criar aplicativos de blockchain da sua maneira preferida: on-line, localmente ou na nuvem. Veja como **começar a desenvolver gratuitamente**.

Este tutorial orienta como começar a desenvolver uma rede de blockchain. Ele apresentará o **Hyperledger Composer** e sua interface com o usuário, o **Hyperledger Composer Playground** no qual é possível modelar e testar facilmente a sua rede usando apenas o Docker e o navegador da web.

Pré-requisitos

Para acompanhar este tutorial, os seguintes softwares devem estar instalados em seu computador:

- [Docker Engine 17.03 ou superior](#)
- Navegador da web

O que é o Hyperledger Composer?

Um dos projetos [Hyperledger](#) hospedado pelo Linux Foundation, o [Hyperledger Composer](#) é um conjunto de ferramentas que facilita o desenvolvimento de aplicativos de blockchain e consiste em:

O que significa CTO?

Não está correlacionado com nada atualmente. Os arquivos receberam a extensão CTO porque o projeto original era chamado de "Concerto", de acordo com a explicação da committer do projeto [Simon Stone em Rocket.Chat](#). (Será necessário usar sua [ID do Linux Foundation](#) para efetuar login.)

- Uma **linguagem de modelagem chamada CTO** (uma homenagem ao nome do projeto original, *Concerto*)
- Uma **interface com o usuário chamada Hyperledger Composer Playground** para acelerar a configuração, a implementação e os testes de uma rede de negócios
- **Ferramentas da interface da linha de comandos (CLI)** para integrar as redes de negócios usando o Hyperledger Composer com uma instância em execução da rede de blockchain do Hyperledger Fabric

Nesta parte da série de tutoriais, vou apresentar a linguagem de modelagem CTO e o Hyperledger Composer Playground. Vou deixar a CLI para a Parte 2 desta série, na qual vou mostrar tudo sobre a CLI (e muito mais).

Linguagem de modelagem do Hyperledger Composer (CTO)

O Hyperledger Composer tem sua própria linguagem de modelagem (chamada de CTO) usada para modelar a rede de negócios. Na Parte 1, vou mostrar como usar o CTO para modelar a amostra [Rede de Mercadorias Perecíveis](#). Esta amostra de rede de negócios demonstra como produtores, transportadoras e importadores definem os contratos de preço das mercadorias perecíveis, com base nas leituras de temperatura recebidas para os contêineres de remessa.

Hyperledger Composer Playground

O Hyperledger Composer Playground é uma interface baseada em navegador que pode ser usada para modelar a rede de negócios: quais itens de valor (ativos) são trocados, quem participa (participantes) da troca, como o acesso é protegido (controle de acesso), qual lógica de negócios (transações) é envolvida no processo e muito mais.

O Hyperledger Composer Playground (Playground, daqui para frente) usa o armazenamento local do navegador para simular o armazenamento do estado da rede de blockchain, o que significa que não é necessário executar uma rede de par de validação real para usar o Playground.

O que você vai fazer neste tutorial:

- Conhecer os conceitos de rede de negócios
- Execute o Playground no computador usando o Docker
- Familiarizar-se com a linguagem de modelagem
- Usar a linguagem de modelagem para modelar ou descrever a rede de negócios
- Testar a rede de negócios

Na Parte 2, vou mostrar como instalar o conjunto completo de utilitários de desenvolvimento do Hyperledger Composer, como trabalhar com recursos mais avançados da linguagem CTO (incluindo eventos), como testar a unidade de contratos inteligentes JavaScript e como interagir com uma rede de blockchain real do Hyperledger Fabric usando a interface da linha de comandos (CLI).

Na Parte 3, vou fornecer uma visão avançada do Composer: como gerar uma interface REST e uma GUI usando Yeoman, e como implementar o aplicativo da rede de blockchain na IBM Cloud.

Conceitos de rede de negócios



Obtenha uma rodada mensal do que há de melhor em ferramentas gratuitas, treinamentos e recursos gratuitos da comunidade para ajudá-lo a fazer o blockchain funcionar.

[Problema atual](#) | [Assinar](#)

Em linhas gerais, uma rede de negócios é um grupo de entidades que trabalham juntas para realizar determinados objetivos. Para atingir esses objetivos, deve haver um contrato entre os membros da rede de negócios, em relação a:

- As mercadores e os serviços que são trocados
- Como a troca deve ocorrer (incluindo as regras que controlam pagamentos e multas)
- Quais membros no grupo têm permissão de participar e quando

Na próxima seção, vou introduzir a terminologia de rede comum. Primeiro, eu quero falar sobre o problema de negócios que a sua primeira rede de negócios de blockchain vai resolver: a remessa de bens perecíveis. Ainda mais importante, como a Internet das Coisas, sensores de temperatura e a nuvem são usados para assegurar que os perecíveis sejam enviados em condições ideais (e o que acontecerá se eles não forem).

A rede Mercadorias perecíveis

O que IoT tem a ver com isso?

Parece que a Internet das Coisas está em toda parte. A rede Mercadorias perecíveis obtém as leituras de temperatura de uma array de sensores de IoT que transmitem a temperatura nos contêineres de carga para a IBM Cloud, onde os dados do sensor são armazenados no blockchain.

Saiba mais sobre [convergência de IoT e blockchain](#).

A [rede Mercadorias perecíveis](#) baseada na Internet das Coisas é uma rede que envolve:

- Itens perecíveis como bananas, peras e café
- Parceiros de negócios como produtores, transportadoras e importadores
- Remessas de mercadorias perecíveis
- Contratos entre as partes de negócios que estipulam as condições dos contratos
- Confirmação de recebimento de mercadorias e serviços

Vou usar essa rede de negócios como exemplo em toda esta série de tutoriais. Conforme você avançar na série, poderá notar que aumentará a complexidade do aplicativo à medida que eu apresentar mais conceitos do Hyperledger Composer e mostrar como eles estão relacionados ao blockchain e à IBM Cloud.

Ativos

Um nó *ativo* é qualquer coisa de valor que possa ser trocada entre as partes em um contrato de negócios. Isso significa que um ativo pode ser qualquer coisa. Exemplos:

- Um barco
- Uma quantidade de estoque
- Uma casa
- Um caixote de bananas
- Uma remessa de bananas
- Um contato para remessa de mil caixotes de bananas pelo preço X com base nas condições {X, Y, Z}

Nomeie-o. Se ele tiver um valor observado e puder ser trocado entre partes, será um ativo. Na rede Mercadorias perecíveis, os ativos incluem as próprias mercadorias perecíveis, as remessas dessas mercadorias e os contratos que controlam as atividades executadas durante a troca.

Participantes

Uma *participante* é um membro da rede de negócios. Para a rede Mercadorias perecíveis, isso inclui os produtores que produzem as mercadorias perecíveis, os transportadoras, que as transportam dos produtores para os portos e os importadores, que recebem a entrega das mercadorias nos portos. Obviamente, esse modelo está extremamente simplificado, mas dará uma noção de como os aplicativos reais são modelados usando a terminologia da rede de negócios.

Controle de acesso

Em uma rede de negócios, nem todos os participantes têm acesso a tudo. Por exemplo, um produtor não tem acesso ao contrato entre a transportadora e o importador. *Controle de acesso* é usado para limitar quem tem acesso a o que (e sob quais condições).

Transações

Quando um ativo é "tocado", essa interação pode afetar o estado no livro-razão do blockchain. A interação é modelada no Hyperledger Composer com uma *transação*.

Exemplos de transações na rede Mercadorias perecíveis incluem:

- Um sensor de IoT no contêiner de remessa registra uma leitura de temperatura
- Um importador recebe uma remessa de mercadorias perecíveis
- Um sensor de GPS de IoT registra o local atual do contêiner de remessa

As transações são a lógica de negócios (ou os contratos inteligentes) do sistema.

Eventos

Um nó *evento* é uma notificação produzida pelo aplicativo de blockchain e consumida por uma entidade externa (como um aplicativo) no modo publicar/assinar.

Embora o livro-razão do blockchain seja atualizado e os ativos sejam trocados no sistema, este é um processo interno (sistema). No entanto, há momentos em que uma entidade externa precisa ser notificada de que o estado do livro-razão mudou ou que alguma outra coisa significativa ocorreu (ou não ocorreu mas deveria ter ocorrido) no sistema. O aplicativo de blockchain poderia usar um evento nesse caso.

Exemplos de eventos na rede Mercadorias perecíveis podem incluir:

- A leitura de temperatura excedeu x vezes um limite superior ou inferior, (isso pode indicar um problema com o contêiner de remessa, por exemplo).
- Uma remessa chegou.
- Uma remessa chegou ao porto (um sensor de GPS de IoT poderia relatar isso, por exemplo).

Na próxima seção, vou mostrar como modelar a rede Mercadorias perecíveis de ativos, participantes e transações. Vou falar um pouco sobre controle de acesso e deixar os eventos para a Parte 2 desta série.

Execute o Playground no computador usando o Docker

O que é o Playground?

O Playground é um ambiente que permite criar e testar redes de negócios de blockchain rapidamente. Ele não precisa de uma rede de blockchain em execução e, assim, reduz a complexidade de definir, validar e testar uma rede de negócios.

O Playground é executado em um contêiner do Docker e pode ser instalado no computador de dois modos:

- Com uma rede de par de validação do Hyperledger Fabric
- No modo somente de navegador

Com uma rede de par de validação do Hyperledger Fabric

Esse modo instala o Playground com a rede de par de validação do Hyperledger Fabric e inclui o contêiner do Docker para o Playground, juntamente com todos os outros contêineres do Docker para executar uma rede de validação do Hyperledger Fabric.

Ao se aprofundar no tutorial nas Partes 2 e 3, você precisará de uma rede de par de validação do Hyperledger Fabric completa. Neste momento, isso não é necessário porque nenhuma das atividades que serão executadas na Parte 1 exigem isso.

Modo somente de navegador

Usando o modo somente de navegador, é possível modelar e testar a rede de negócios usando um livro-razão de blockchain simulado que reside no armazenamento local do navegador.

Essa é a abordagem que será usada na Parte 1.

Execute o Playground

Em uma janela do terminal (Mac/Linux) ou em um prompt de comandos (Windows), execute este comando:

```
docker run --name composer-playground --publish 8080:8080 hyperledger/composer-playground
```

O Docker será iniciado de forma interativa. Eu gosto de executar o Playground dessa forma porque posso verificar o que é registrado no STDOUT:

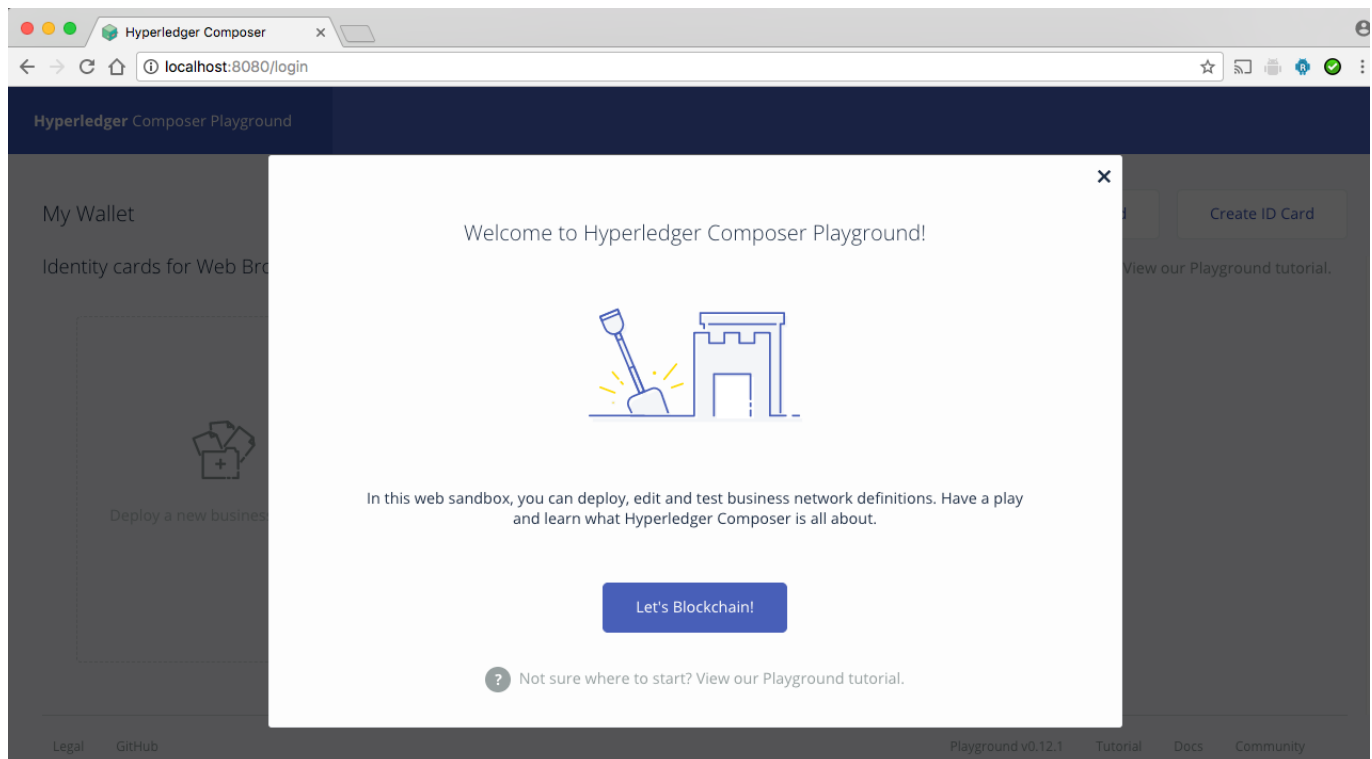
```
$ docker run --name composer-playground --publish 8080:8080 hyperledger/composer-playground
0|composer | PlaygroundAPI           :createServer()           > 8080
0|composer | ConnectionProfileManager :constructor()           Created a new ConnectionProfileManager
{"fs":{"constants":
{"O_RDONLY":0,"O_WRONLY":1,"O_RDWR":2,"S_IFMT":61440,"S_IFREG":32768,"S_IFDIR":16384,"S_IFCHR":8192,"S_IFBLK":24576,"
0|composer | PlaygroundAPI           :createServer()           Playground API started on port 8080
0|composer | PlaygroundAPI           :createServer()           <
0|composer | Composer               :main()                 >
```

Se você desejar executar o Playground de forma desanexada, inclua `--detach`:

```
docker run --name composer-playground --publish 8080:8080 --detach hyperledger/composer-playground
```

Agora, abra um navegador e acesse `http://localhost:8080` e será exibida uma tela como a Figura 1:

Figura 1. Tela de boas-vindas do Playground



Ao concluir a execução do Playground, `ctrl+c` para encerrar o contêiner no modo interativo. Se você estiver executando no modo desanexado, execute este comando:

```
docker stop composer-playground
```

Agora limpe o contêiner do Docker (ou o Docker reclamará se você tentar executá-lo novamente):

```
docker rm --force composer-playground
```

Vídeo: Executando o playground, tour da UI

Neste vídeo, vou mostrar como executar o Playground usando o Docker e fornecer um tour da UI do Playground.

Para assistir ao vídeo, **Tour do Hyperledger Composer Playground**, por favor acesse a versão online deste artigo.

A linguagem de modelagem do Hyperledger Composer

Para seja possível usar o Hyperledger Composer para testar e implementar uma rede de negócios de blockchain, é necessário criar um modelo. O Hyperledger Composer tem sua própria linguagem de modelagem para fazer isso.

Ah, legal, outra linguagem para aprender, certo? Felizmente, a linguagem de modelagem CTO é simples (e intuitiva se você já tiver trabalhado com conceitos de orientação a objetos).

A [Linguagem de modelagem CTO](#) é extremamente focada (para modelar redes de negócios) com apenas algumas palavras-chave, portanto, não há muito que aprender. O modelo para a rede de negócios reside em um arquivo que tem uma extensão de arquivo `.cto` e contém definições para os seguintes elementos:

- Namespace
- Recursos
- Importa de outros namespaces, conforme o necessário

Se o modelo for muito grande, poderá haver vários arquivos de modelo `.cto`, conforme o necessário. Cada arquivo de modelo `.cto` deve incluir um único namespace e pelo menos uma definição de recurso.

Namespace

Um namespace cria um limite cujos nomes de escopo são considerados exclusivos. Cada arquivo de modelo `.cto` requer um namespace, o que significa que cada nome em um arquivo de modelo `.cto` precisa ser exclusivo.

Você já está familiarizado com o conceito de namespace, mesmo que não tenha percebido. Os sistemas de arquivos usam o diretório como um namespace, assim, dois arquivos no mesmo

diretório não podem ter o mesmo nome. No entanto, dois arquivos em diretórios diferentes podem ser o mesmo nome, pois estão em *namespaces diferentes*.

A conclusão: dois recursos em um arquivo de modelo CTO (o limite de namespace) não podem ter o mesmo nome.

Resource

Um recurso é um dos seguintes:

- ativo - um ativo de rede de negócios
- participante - um participante de rede de negócios
- transação - lógica de negócios
- evento - uma notificação de algo interessante acontecendo no sistema
- tipo enumerado - um conjunto de valores nomeados
- conceito - qualquer objeto que você deseje modelar que não seja um dos outros tipos

Cada tipo de recurso corresponde ao seu tipo de modelo do mesmo nome (por exemplo, ativo é usado para modelar um ativo, `participant` modela um participante, etc.).

Um recurso tem as seguintes propriedades:

- Um namespace no qual ele é definido
- Um nome, que deve ser exclusivo no namespace
 - Se o recurso for um ativo ou participant, ele deverá ter um campo identificado indicado por `identified by` seguido do nome do campo
- (Opcional) Seu tipo pai (supertipo), se disponível, indicado por `extends` seguido pelo nome do tipo pai
- (Opcional) A palavra-chave `abstract`, se você não desejar que o recurso seja instanciado, mas sim usado como um supertipo para outros recursos desse tipo

Na rede Mercadorias Perecíveis, um ativo Remessa é modelado desta forma:

```
/**
 * 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
.
.
/**
 * 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
}
```


Vamos analisar o exemplo acima e eu vou apontar algumas coisas.

O namespace é `org.acme.shipping.perishable`.

A palavra-chave `ativo` indica que `shipment` é um ativo. A propriedade (indicada pela letra minúscula "o") `shipmentId` é o tipo `Sequência` e identifica exclusivamente uma `shipment` (conforme a indicação de `identified by`).

Cada uma de suas propriedades têm um tipo, que pode ser um tipo fundamental (como `Sequência`) ou um tipo enumerado (como `ProductType`) ou uma transação (como uma array de `TemperatureReading`).

A referência (indicada por `-->`) a `contract` é chamada de *relacionamento* e é unidirecional.

Tipos enumerados

Quando o conjunto de valores que uma propriedade específica pode ter é conhecido, é necessário modelar como um tipo enumerado. Isso facilita a restrição de valores, o que simplifica a validação.

Um tipo enumerado é declarado desta forma:

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

Conceitos

Quando uma entidade existe no modelo de negócio, mas não é um ativo, um participante, uma transação ou um evento, ela deve ser modelada como um *conceito*.

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

O conceito de endereço é importante em um modelo de negócio, mas não se encaixa exatamente em nenhuma das outras categorias, portanto, é modelado como um `concept`.

Importações

As importações são usadas em um arquivo de modelo `.cto` para indicar um relacionamento entre uma entidade nesse arquivo de modelo e uma entidade em outro arquivo de modelo.

Não vou falar de importações neste tutorial, porque o modelo com o qual vamos trabalhar é muito pequeno. O conceito é semelhante a `import` na linguagem Java™ e `#include` em C++.

Referência de CTO

Depois que você aprender a sintaxe básica (que, na maioria dos casos, é óbvia devido ao contexto), a linguagem de modelagem ficará fácil de entender. Se você já tiver trabalhado com conceitos de orientação a objetos, será ainda mais fácil.

Se você deseja saber mais, eu recomendo a leitura da [documentação completa da linguagem de modelagem CTO](#).

Modele a rede de negócios

No campo [vídeo anterior](#), eu mostrei como criar um modelo de negócio vazio usando o Playground. Nesta seção, você vai modelar a rede Mercadorias perecíveis no Playground (não se preocupe, o modelo integrado `perishable-network` o ajudará).

Exclua o armazenamento do navegador do host local

Se for exibida uma tela de boas-vindas como [Figura 1](#), estará tudo certo e você poderá acessar diretamente "[Crie um novo modelo no Playground](#)."

No modo somente de navegador, somente o Hyperledger Composer permite trabalhar com um modelo de cada vez. Se houver outro modelo carregado, poderá ser necessário excluir o armazenamento local do navegador antes de carregar outro modelo.

O Playground poderá substituir o modelo atual pelo novo. No entanto, se ocorrerem erros, será possível começar desde o início, excluindo o armazenamento local do navegador. O procedimento varia de navegador para navegador.

No Chrome, por exemplo, em `Configurações > Avançado > Configurações de conteúdo > Cookies > Todos os cookies e dados de site > host local`, clique no ícone de lixeira para remover o armazenamento local. Se você estiver usando um navegador diferente, siga as instruções específicas desse navegador e exclua todo o armazenamento local.

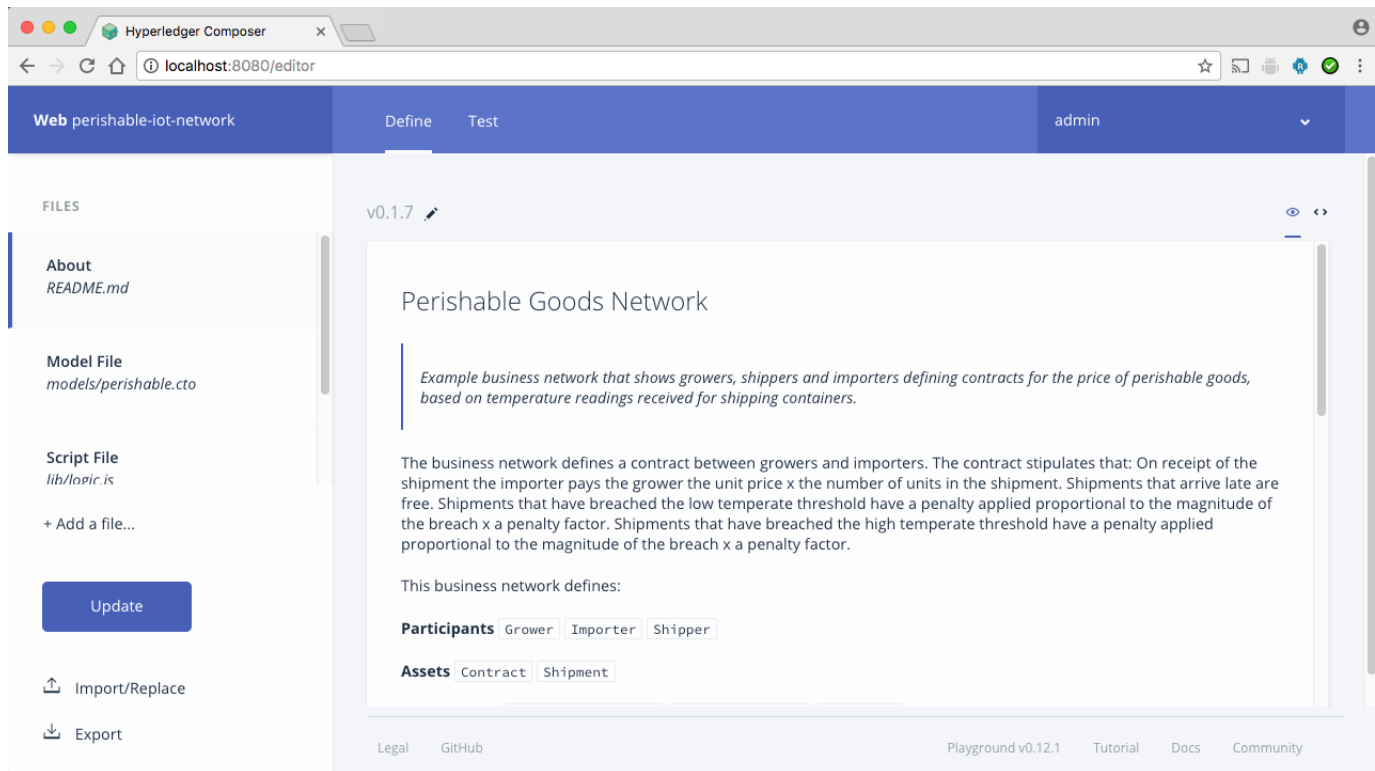
Crie um novo modelo no Playground

No campo [vídeo anterior](#), eu mostrei como criar uma nova rede de negócios vazia no Playground e também os fundamentos básicos de como entrar no Playground, então, não vou repetir essa parte. Se você não conseguiu assistir ao vídeo, confira-o antes de continuar.

Clique no botão **Vamos executar o blockchain** para começar (veja a [Figura 1](#)). Em seguida, crie uma rede de negócios usando o modelo `perishable-network`. Chame-o de `perishable-iot-network` e clique em **Implementar**.

No cartão de ID de administrador, clique em **Conectar agora**. Será exibido algo como a [Figura 2](#).

Figura 2. A rede Mercadorias perecíveis



Em FILES, observe o seguinte:

- `README.md` - esse arquivo Markdown fornece uma visão geral rápida da rede Mercadorias perecíveis
- `models/perishable.cto` - contém o modelo de negócio
- `lib/logic.js` - contém o código da lógica de negócios (contrato inteligente), incluindo a implementação da transação

Quando você selecionar um dos arquivos em FILES, ele será aberto na janela do editor do lado direito. Abra o arquivo de modelo (`perishable.cto`), que contém o modelo.

Um ativo `grower` é modelado da seguinte forma:

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

e um `shipper` desta forma:

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

Um ativo `contract` desta forma:

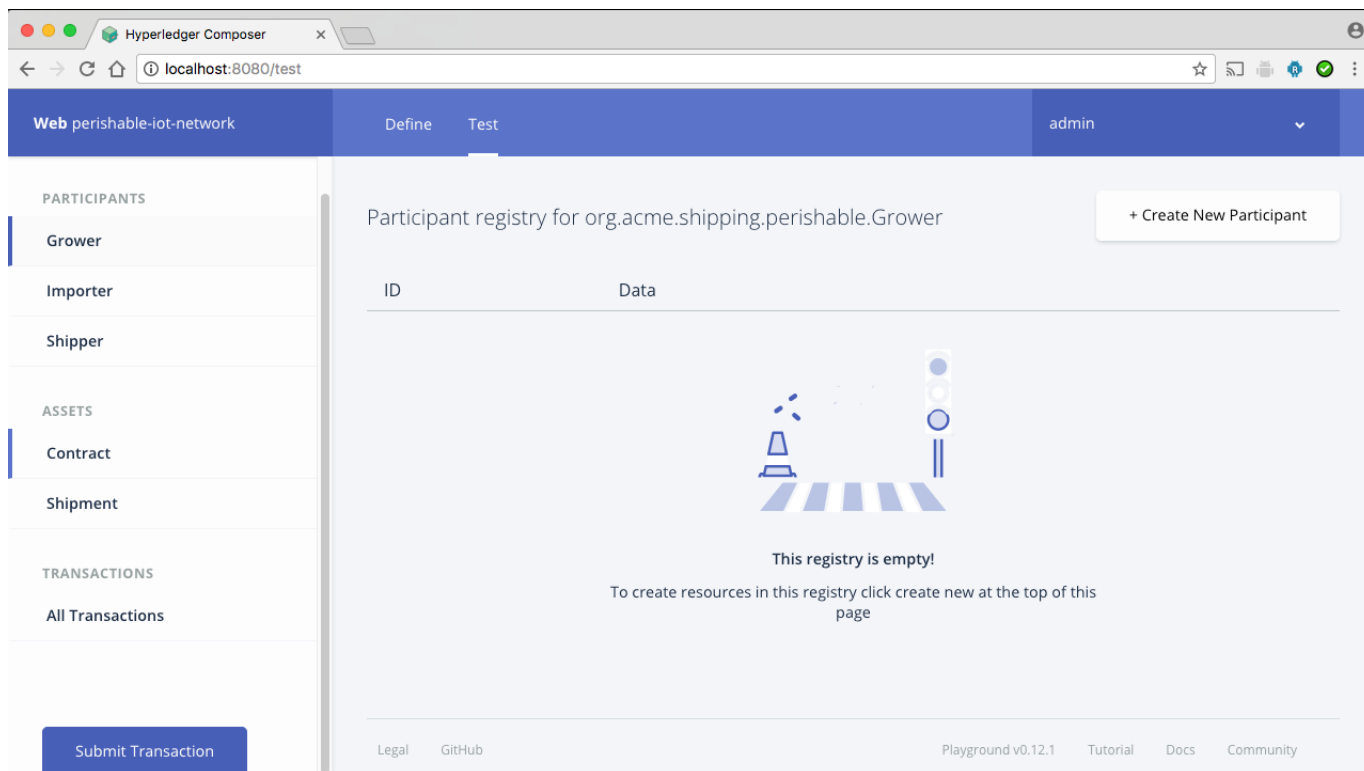
```
/**
 * Defines a contract between a Grower and an Importer to ship using
 * a Shipper, paying a set unit price. The unit price is multiplied by
 * a penalty factor proportional to the deviation from the min and max
 * negotiated temperatures for the shipment.
 */
asset Contract identified by contractId {
  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
}
```

É interessante que você se acostume com o modelo e com a aparência dos diversos recursos dentro do editor. Faça o mesmo para `lib/logic.js` e também se familiarize com o código JavaScript.

Instancie o modelo

Clique na guia **Teste** na parte superior da tela e você verá algo como a Figura 3.

Figura 3. perishable-network - Guia Teste



Observe que ativos e participantes do modelo aparecem no lado esquerdo da tela, mas no centro da tela há uma mensagem informando que o registro está vazio. O que está acontecendo?

Como eu mostrei no [vídeo](#), quando a rede de negócios é criada inicialmente, os registros de ativo e participante ficam vazios. É necessário criar as *instâncias* Ativos e Participantes que residirão no registro.

Na próxima seção, vou mostrar como instanciar e testar o modelo.

Testar a rede de negócios

Os modelos são ótimos para funcionar como um tipo de projeto para o aplicativo que está sendo criado, mas um modelo de algo somente será bom se ele resultar (em algum momento) em algo *real*. Por exemplo, um conjunto de projetos de um arranha-céu é fundamental para construir um prédio, mas somente será útil se em algum momento ele for realmente usado para construir (instanciar) um prédio real!

Voltando ao modelo de negócio, que para ser útil, precisará ser *instanciado*. Mas o que isso significa para um aplicativo de blockchain?

Os registros de ativo e participante

Mais cedo você viu o modelo para o participante `Grower`, o ativo `Shipment` e assim por diante. Agora, chegou o momento de instanciar esses recursos e, então, suas instâncias passarão a

residir em seus respectivos registros. Assim, as instâncias de ativo ficarão no registro de ativo e as instâncias de participante ficarão no registro de participante.

O modelo `perishable-network` inclui uma transação implementada como uma função JavaScript no módulo `lib/logic.js` chamada `setupDemo()` que pode ser usada para instanciar o modelo e criar entradas nos registros de ativo e de participante. Essa função é fornecida como uma forma de fazer com que a rede de negócios do modelo comece a funcionar rapidamente em vez de inserir o modelo manualmente.

Eu não vou mostrar a função `setupDemo()` aqui, mas gostaria de informar que ela faz três coisas:

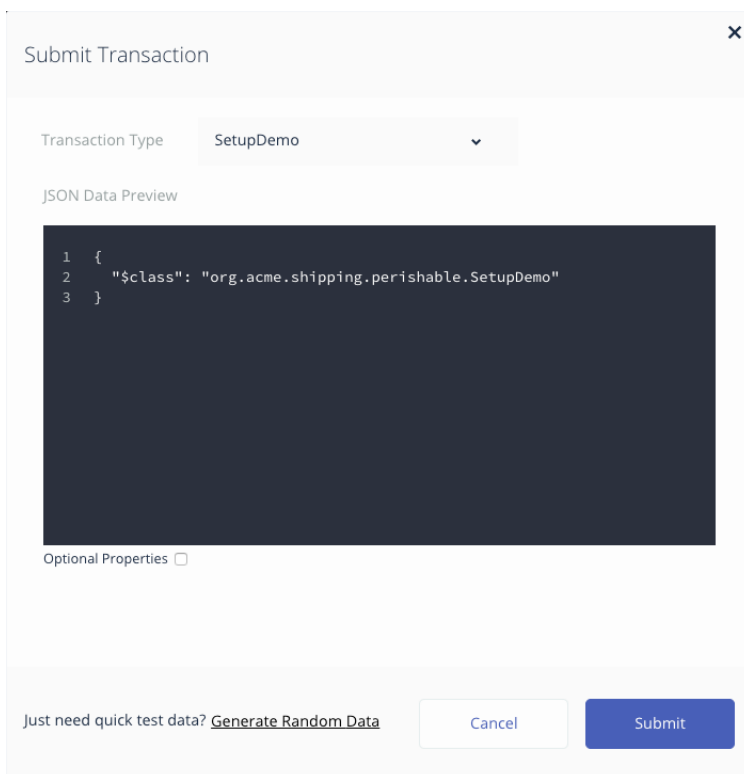
1. Cria instâncias de todos os ativos e participantes do modelo
2. Configura os valores de propriedade nessas instâncias
3. Armazena as instâncias em seus respectivos registros

É interessante que você abra o arquivo `lib/logic.js` no editor e verifique-o você mesmo.

Instancie o modelo

Para executar a transação `SetupDemo`, clique no botão **Enviar transação** e será exibido um diálogo modal semelhante ao da Figura 4.

Figura 4. SubmitTransaction - SetupDemo



Submit Transaction

Transaction Type: SetupDemo

JSON Data Preview

```
1 {  
2   "$class": "org.acme.shipping.perishable.SetupDemo"  
3 }
```

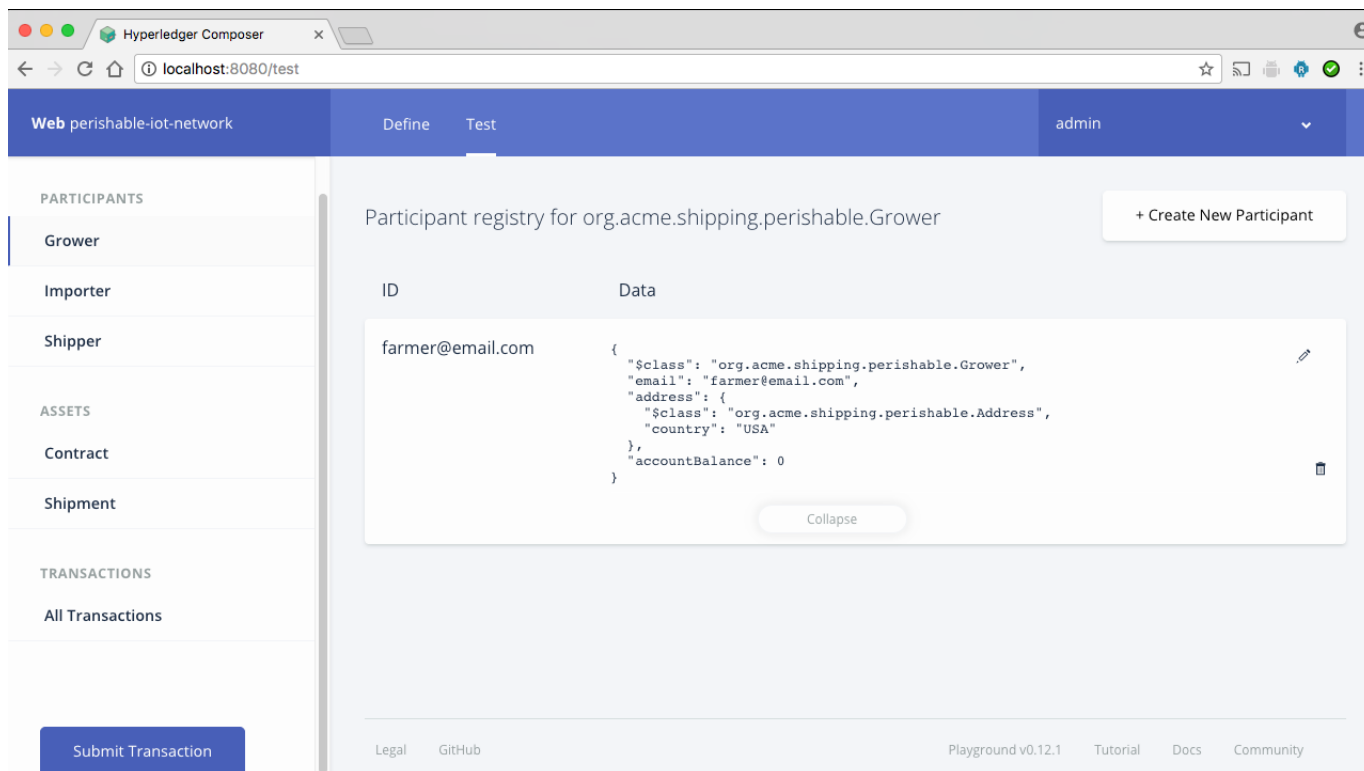
Optional Properties ☐

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

Assegure-se de que `SetupDemo` aparece na lista suspensa **Tipo de transação** e clique no botão **Enviar**. Quando a transação for executada com sucesso, será exibida uma mensagem de notificação informando isso.

Selecione **Produtor** na área de janela ASSETS à esquerda e todas as suas instâncias serão exibidas do lado direito (Figura 5). O mesmo também vale para os outros recursos (vá em frente, experimente!).

Figura 5. Uma instância do ativo Produtor



Agora que você tem uma rede de negócios definida e os ativos e participantes estão em seus respectivos registros, é possível testar a rede.

E as transações?

Até agora nesta seção, eu falei sobre ativos e participantes, mas e as transações no modelo de negócio? Onde elas vão aparecer?

Para responder à primeira pergunta: as transações representam a lógica de negócios do aplicativo (contratos inteligentes ou chaincode). A lógica de negócios cumprida pelo contrato inteligente gerada por `setupDemo()` estipula as seguintes condições:

1. A temperatura dentro do contêiner de remessa deve ser 6 graus Celsius o tempo todo. Se a temperatura da remessa ficar fora do intervalo acordado (+/- 5 graus), o preço da remessa (\$0,50/unidade) será reduzido para \$0,20/unidade para cada grau abaixo e para \$0,10 para cada grau acima.
2. Se a remessa chegar com atraso, o Produtor não receberá o pagamento da remessa.

Ok, então onde as transações aparecem? Uma transação não é *instanciada* em si, mas é mostrada como um código Javascript em `lib/logic.js`.

A Figura 6 mostra o código do contrato inteligente (de lib/logic.js) que cumpre a segunda estipulação (não pagamento para uma remessa em atraso) do contrato.

Figura 6. Contrato inteligente: multa de pagamento em atraso

Script File lib/logic.js

```

16  /**
17   * A shipment has been received by an importer
18   * @param {org.acme.shipping.perishable.ShipmentReceived} shipmentReceived - the ShipmentReceived transaction
19   * @transaction
20   */
21  function payOut(shipmentReceived) {
22
23      var contract = shipmentReceived.shipment.contract;
24      var shipment = shipmentReceived.shipment;
25      var payOut = contract.unitPrice * shipment.unitCount;
26
27      console.log('Received at: ' + shipmentReceived.timestamp);
28      console.log('Contract arrivalDateTime: ' + contract.arrivalDateTime);
29
30      // set the status of the shipment
31      shipment.status = 'ARRIVED';
32
33      // if the shipment did not arrive on time the payout is zero
34      if (shipmentReceived.timestamp > contract.arrivalDateTime) {
35          payOut = 0;
36          console.log('Late shipment!');
37      } else {
38          // find the lowest temperature reading
39          if (shipment.temperatureReadings) {
40              // sort the temperatureReadings by centigrade
41              shipment.temperatureReadings.sort(function (a, b) {

```

E o controle de acesso?

O controle de acesso é controlado por um arquivo no modelo chamado `permissions.acl`. Na seção [Conceitos de rede de negócios](#), vou apresentar o controle de acesso como um dos conceitos principais. Nas Parte 2 e 3, vou abordar melhor este tópico extremamente importante e como controlar o acesso ao aplicativo de blockchain.

Vamos analisar `permissions.acl` na guia **Definir** do modelo. O modelo `perishable-network` incluído no arquivo da lista de controle de acesso (ACL) que é semelhante a este:

```

/**
 * 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
}

```

O arquivo da ACL contém regras que permitem controlar o acesso aos recursos no aplicativo de blockchain. Neste momento, basta dizer que o Hyperledger Composer cuida de toda a segurança e, nos próximos tutoriais desta série, eu vou mostrar tudo sobre isso.

Agora, as regras de controle de acesso definidas acima concedem acesso amplo, o que está bom para este momento, pois você está apenas começando com o Hyperledger Composer.

Teste o modelo

Agora que o modelo foi instanciado, chegou a hora de testá-lo e isso significa executar o código! Nesse caso, isso significa executar o código JavaScript de `lib/logic.js`.

Antes de configurar o teste, vamos revisar o contrato [que foi instanciado na função `setupDemo()`] para ver os termos. A Figura 7 mostra o código Javascript usado para instanciar o ativo de contrato:

Figura 7. Contrato inteligente: termos e condições

Script File `lib/logic.js`

```
148 var shipper = factory.newResource(NS, 'Shipper', 'shipper@email.com');
149 var shipperAddress = factory.newConcept(NS, 'Address');
150 shipperAddress.country = 'Panama';
151 shipper.address = shipperAddress;
152 shipper.accountBalance = 0;
153
154 // create the contract
155 var contract = factory.newResource(NS, 'Contract', 'CON_001');
156 contract.grower = factory.newRelationship(NS, 'Grower', 'farmer@email.com');
157 contract.importer = factory.newRelationship(NS, 'Importer', 'supermarket@email.com');
158 contract.shipper = factory.newRelationship(NS, 'Shipper', 'shipper@email.com');
159 var tomorrow = setupDemo.timestamp();
160 tomorrow.setDate(tomorrow.getDate() + 1);
161 contract.arrivalDateTime = tomorrow; // the shipment has to arrive tomorrow
162 contract.unitPrice = 0.5; // pay 50 cents per unit
163 contract.minTemperature = 2; // min temperature for the cargo
164 contract.maxTemperature = 10; // max temperature for the cargo
165 contract.minPenaltyFactor = 0.2; // we reduce the price by 20 cents for every degree below the min temp
166 contract.maxPenaltyFactor = 0.1; // we reduce the price by 10 cents for every degree above the max temp
167
168 // create the shipment
169 var shipment = factory.newResource(NS, 'Shipment', 'SHIP_001');
170 shipment.type = 'BANANAS';
171 shipment.status = 'IN_TRANSIT';
172 shipment.unitCount = 5000;
```

Agora, está tudo pronto para testar o contrato. Com o Playground em execução no navegador, clique na guia Testar na parte superior da UI do Playground.

Vamos testar o cenário a seguir:

1. Os sensores de temperatura de IoT fornecem as seguintes leituras (os números estão em graus Celsius):
 1. 5
 2. 7
 3. 1
 4. 4
2. A Remessa é recebida.

Vamos analisar os componentes deste cenário um de cada vez, começando com os dados do sensor de temperatura.

Em um aplicativo real, os sensores de temperatura de IoT enviariam esses dados para a IBM Cloud, na qual o código do contrato inteligente seria chamado em relação ao blockchain para registrar essas transações.

No Playground, o blockchain é mantido no armazenamento local do navegador, mas o código de transação executado é o mesmo, independentemente de onde o blockchain reside (o que faz do Playground um local perfeito para testar, certo?).

Confira a função `temperatureReading()` dentro de `lib/logic.js`:

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

No aplicativo real, quando um sensor de IoT no contêiner de remessa deseja enviar uma leitura, ele a envia para a nuvem (por meio da rede de envio de carga), onde ela é recebida por uma função sem servidor em execução no OpenWhisk, que chama a `temperatureReading()`.

Para simular isso no Playground:

1. Clique no botão **Enviar transação** [como você fez para chamar a função `setupDemo()`].
2. Certifique-se de que a opção **TemperatureReading** aparece na lista suspensa **Tipo de transação**.
3. Altere a leitura "centigrade" de 0 para 5 (a primeira leitura que desejamos enviar) na janela **Visualização de dados JSON**.
4. Assegure-se de que o ID de remessa seja configurado como SHIP_001.
5. Clique em **Enviar**.
6. Repita para as três leituras restantes.

Para receber a remessa do aplicativo real, um aplicativo em execução no dispositivo portátil do importador poderia indicar ao aplicativo em execução na IBM Cloud (ou uma função sem servidor em execução no OpenWhisk) que a remessa foi recebida e esse aplicativo calcularia o pagamento a ser remetido para o produtor.

Para simular o recebimento da remessa no Playground, execute a transação **ShipmentReceived** no Playground, assegure-se de fornecer o ID da remessa e clique em **Enviar**.

Vídeo: Testando o modelo, vendo transações

Eu sei que foram abordados vários assuntos. Mas não se preocupe: neste próximo vídeo, eu vou mostrar como testar o modelo e ver os resultados de todas as transações executadas no console Javascript do navegador.

Para assistir ao vídeo, **Hyperledger Composer Playground - Crie e teste uma rede de blockchain**, por favor acesse a versão online deste artigo.

Trabalhando com modelos

Então você tem esse ótimo modelo de negócio. E agora?

Exporte o modelo

Vamos supor que você deseje disponibilizar o modelo para outros membros da equipe ou precise exportar o modelo para implementá-lo em uma rede de blockchain real em produção (vou mostrar como fazer isso na Parte 2).

O Playground tem um recurso de exportação que permite criar um arquivo Business Network Archive (BNA) que pode ser compartilhado. Para exportar o modelo, na guia Definir, clique no link **Exportar** na parte inferior esquerda da tela e o Playground gerará o arquivo Business Network Archive (BNA) e fará seu download no computador.

Importe um modelo

Vamos supor que você tenha um arquivo Business Model Archive (BNA) — que pode ter sido fornecido por um membro da equipe — e você deseje usá-lo no Playground.

Como colocar um arquivo BNA no Playground? Primeiro, assegure-se de que o Playground esteja funcionando. Em seguida, na guia Definir, escolha **Importar/substituir**. Na tela Importar/substituir rede, clique em **Soltar aqui para fazer upload ou buscar**, use o diálogo de busca para localizar e selecionar o arquivo BNA que deseja importar e escolha **Abra**. Em seguida, clique em **Importar** e confirme se deseja substituir o modelo atual por aquele que você deseja importar.

Observação: se houver algum problema para importar o arquivo BNA, poderá ser necessário limpar o armazenamento local do navegador e tentar a importação novamente. Veja a seção ["Exclua o armazenamento do navegador do host local"](#) anterior neste tutorial para obter mais informações.

Conclusão da Parte 1

A Parte 1 desta série de tutoriais mostrou como executar o Playground usando o Docker e introduziu os conceitos básicos da linguagem de modelagem (CTO) do Hyperledger Composer. Você também viu como é um modelo de rede de negócios simples no Playground criando uma nova rede de negócios usando o modelo perishable-network, fornecido pela imagem do Docker do Playground.

Em seguida, o tutorial mostrou como testar esse modelo armazenando várias transações de leitura de temperatura no blockchain e como o contrato inteligente usou essas transações para resolver os termos do contrato quando a remessa foi recebida.

Por último, eu mostrei como compartilhar modelos com colegas e outras pessoas exportando e importando modelos de/para o Playground.

Fique ligado na Parte 2, que mostrará como criar e refinar a rede de negócios existente da Parte 1, além de realizar o teste de unidade e implementá-la na IBM Cloud.

Temas relacionados: [Documentação do Hyperledger Composer](#) [Rede Mercadorias perecíveis \(GitHub\)](#) [Mais definições de rede de negócios de amostra \(GitHub\)](#) [Comece a usar o blockchain rapidamente com o Hyperledger Composer](#) [Criando uma PoC de blockchain em dez minutos usando o Hyperledger Composer](#) [Integre dados de dispositivo a contratos inteligentes no IBM Blockchain](#) [IBM Blockchain Developer Center](#) [IBM Blockchain Platform](#) [IBM Blockchain Platform para desenvolvedores](#) [Cursos de blockchain para desenvolvedores](#)

Sobre o autor

J Steven Perry



Steve Perry is a software developer, architect, and general Java nut who has been developing software professionally since 1991. His professional interests range from the inner workings of the JVM to UML modeling and everything in-between. Steve has a passion for writing and mentoring; he is the author of *Java Management Extensions* (O'Reilly), *Log4j* (O'Reilly), and the IBM developerWorks articles "[Joda-Time](#)" and "[OpenID for Java Web applications](#)." In his spare time, he hangs out with his kids, plays ice hockey, and practices yoga.

© Copyright IBM Corporation 2018. Todos os direitos reservados.

(www.ibm.com/legal/copytrade.shtml)

[Marcas Registradas](#)

(www.ibm.com/developerworks/br/ibm/trademarks/)