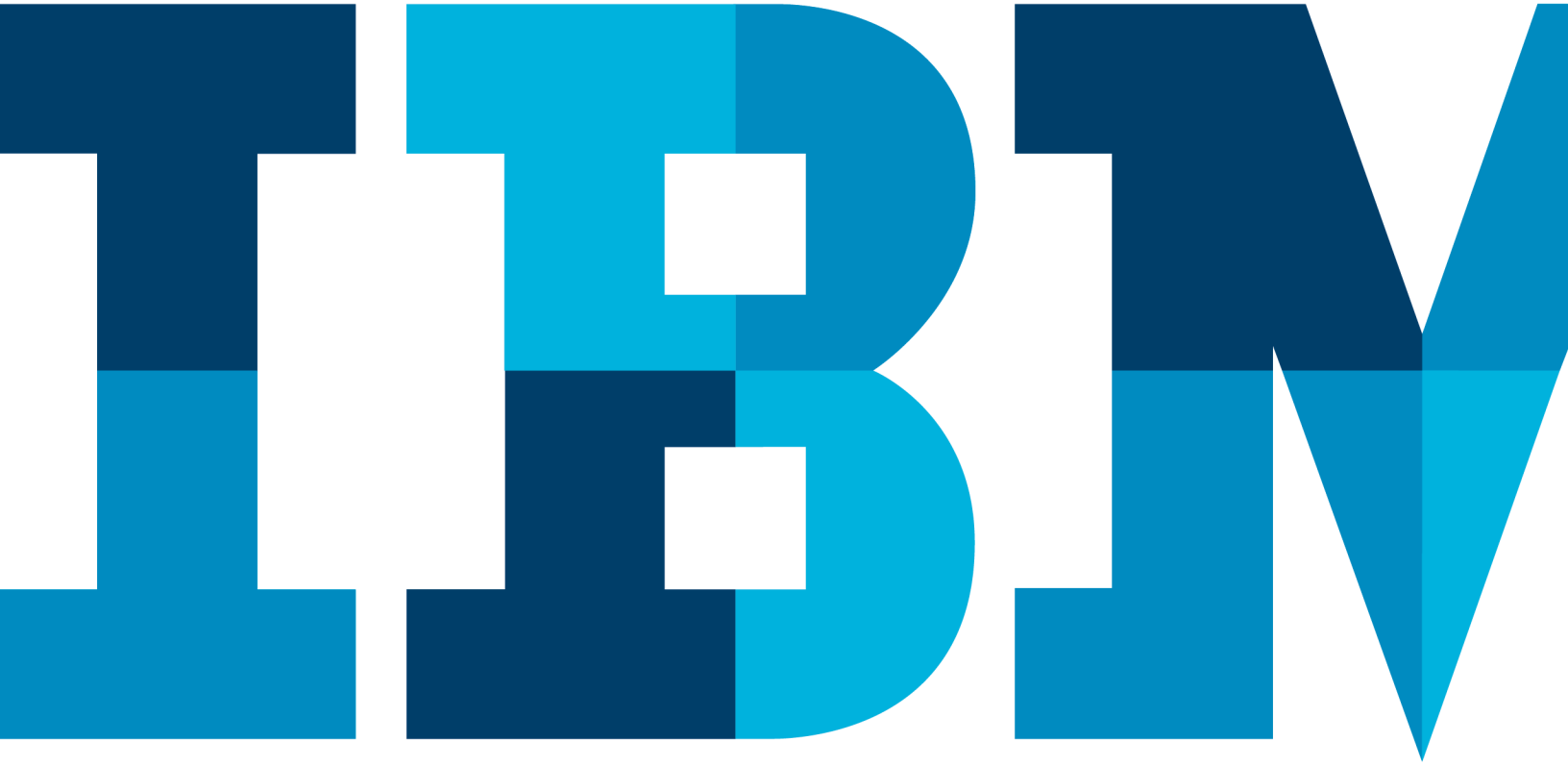
IBM Blockchain Hands-On Composer Development

Lab Two



IBM

Contents

SECTION 1. STARTING THE HYPERLEDGER COMPOSER PLAYGROUND....4
1.1. ACCESSING THE ENVIRONMENT.... 4
SECTION 2. COMPOSER'S MODELLING LANGUAGE....6
2.1. CREATING NEW BUSINESS NETWORK 6
2.2. WRITING A TRANSACTION.... 11
2.3. DEPLOY AND CREATE SOME TEST ASSETS.... 11
SECTION 3. TRANSACTION LOGIC....16
3.1. CREATE THE LOGIC FILE 16
3.2. ADD THE CHANGEOWNER FUNCTION.... 16
3.3. ADD CHANGOWNER'S BODY 17
3.4. TEST CHANGEOWNER 18
SECTION 4. ACCESS CONTROL....21
4.1 ACCESS CONTROL LISTS (ACL) - GRAMMAR 21
4.2 ADDING RULES.... 21
4.3 TESTING THE RULES.... 23
NOTICES 26
APPENDIX A. TRADEMARKS AND COPYRIGHTS....28

Overview

The aim of this lab is to get you familiar with developing Hyperledger Composer business networks. We will do this by exploring the Composer modelling language, how to write transaction processor functions in JavaScript and lastly examine how Access Control is managed in Composer.

The lab will also familiarise you with the Composer Playground, a web-based tool that allows for rapid development and testing of Composer business networks.

It should be noted that while the contents of this lab will predominantly occur within Composer Playground (for the sake of accelerating the learning and development process), the Lab can easily be completed offline and using a text editor such as Visual Studio Code or Atom. In this case please refer to the next Lab for instructions on how to use the command line tools available in Composer.

Section 1. Starting the Hyperledger Composer Playground

1.1. Accessing the environment

After Lab 1 you should have all the environment – Hyperledger Fabric infrastructure and Hyperledger Composer up and running.

Let's open the Composer Playground (using Firefox browser):

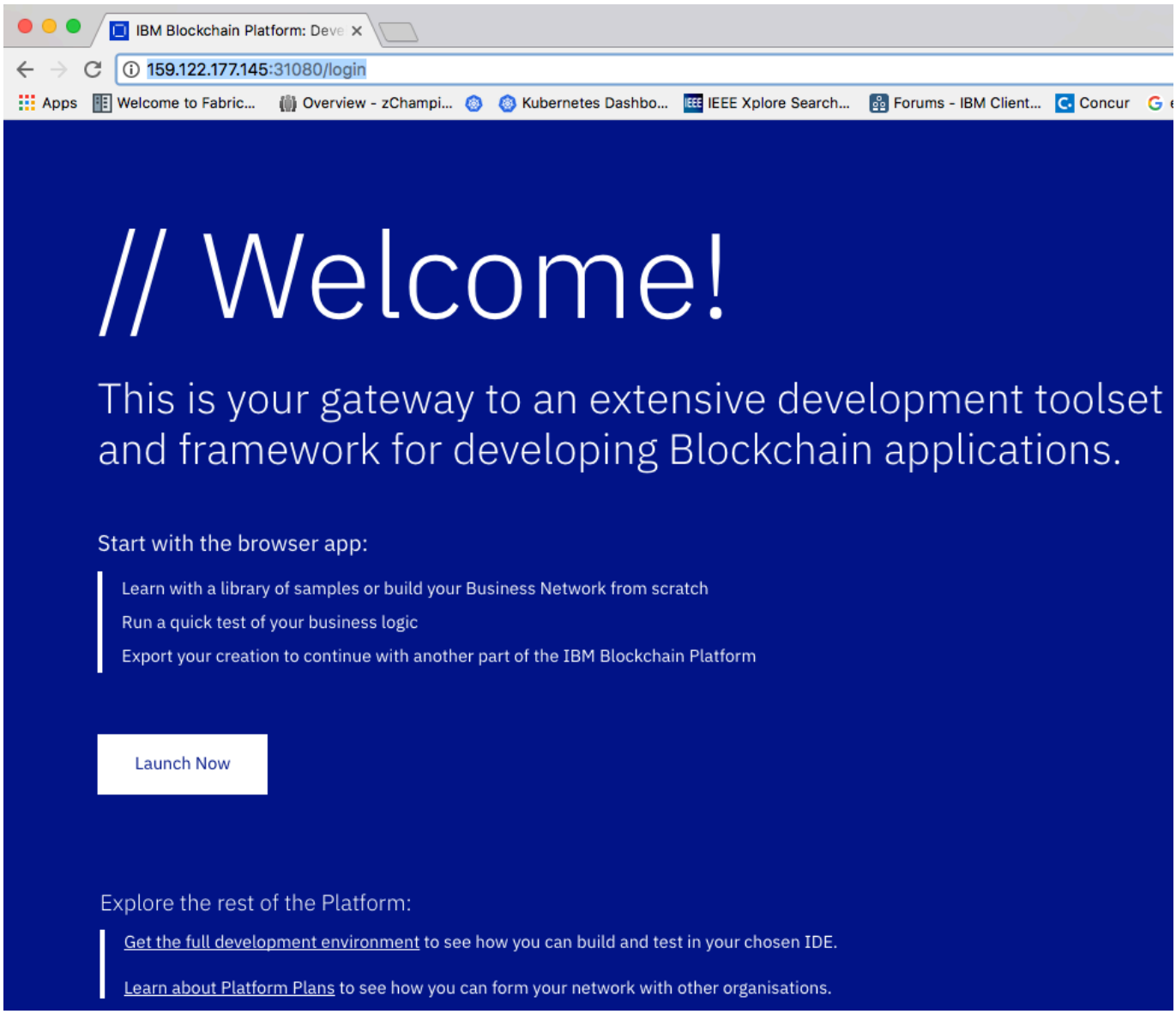
<YOUR PUBLIC IP ADDRESS>:31080

You can retrieve your Public IP address from your terminal using the command:

```
$ bx cs workers <name of the cluster>

ID                                                    Public IP        Private IP
kube-mil01-pa5da85267bee540d0b0954ed59495f0e7-w1      xxx.xxx.xxx.xxx  yy.yyy.yyy.y
```

You should launch Firefox and open the playground web UI, and then press Launch Now:

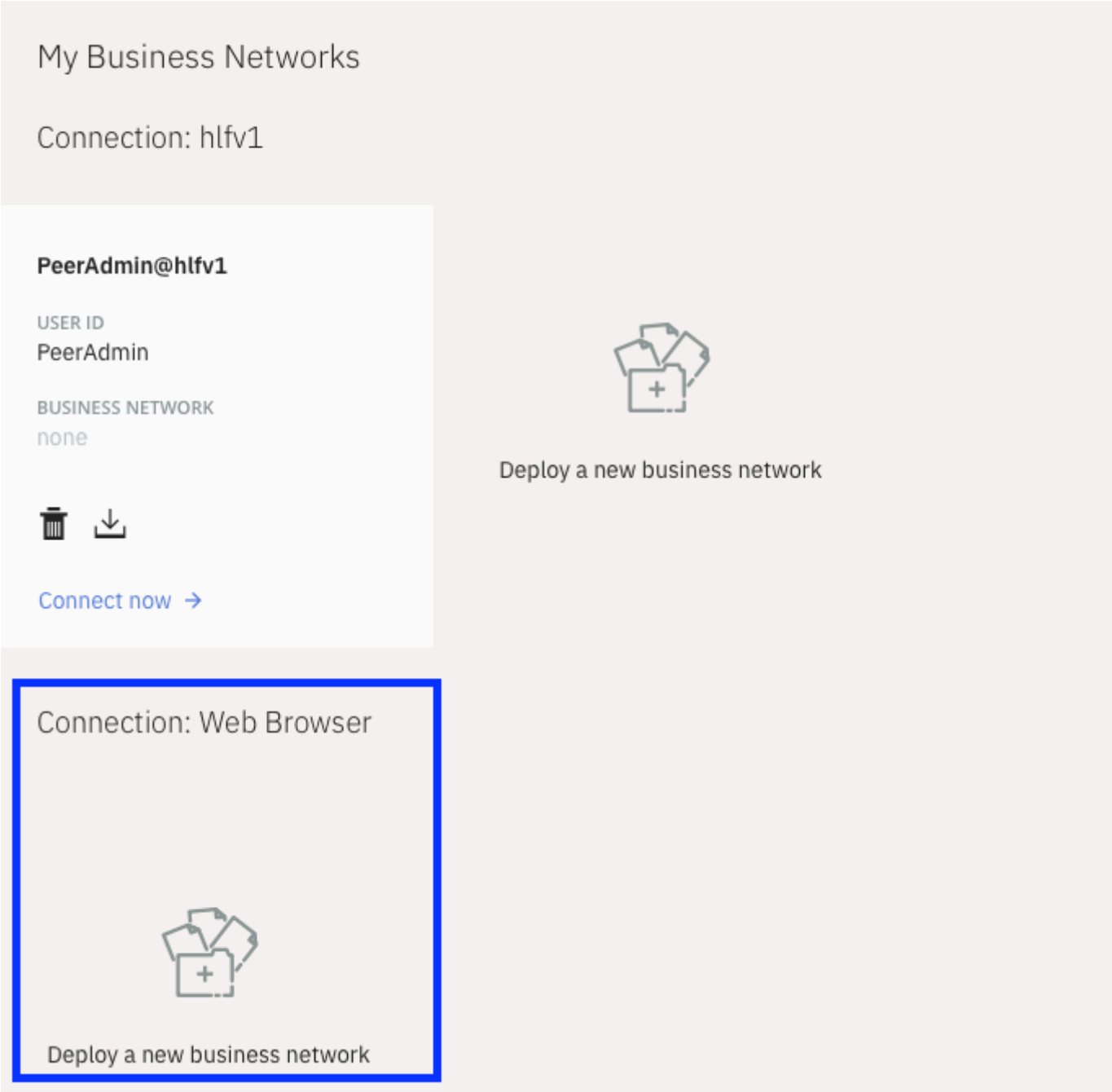


Section 2. Composer's Modelling Language

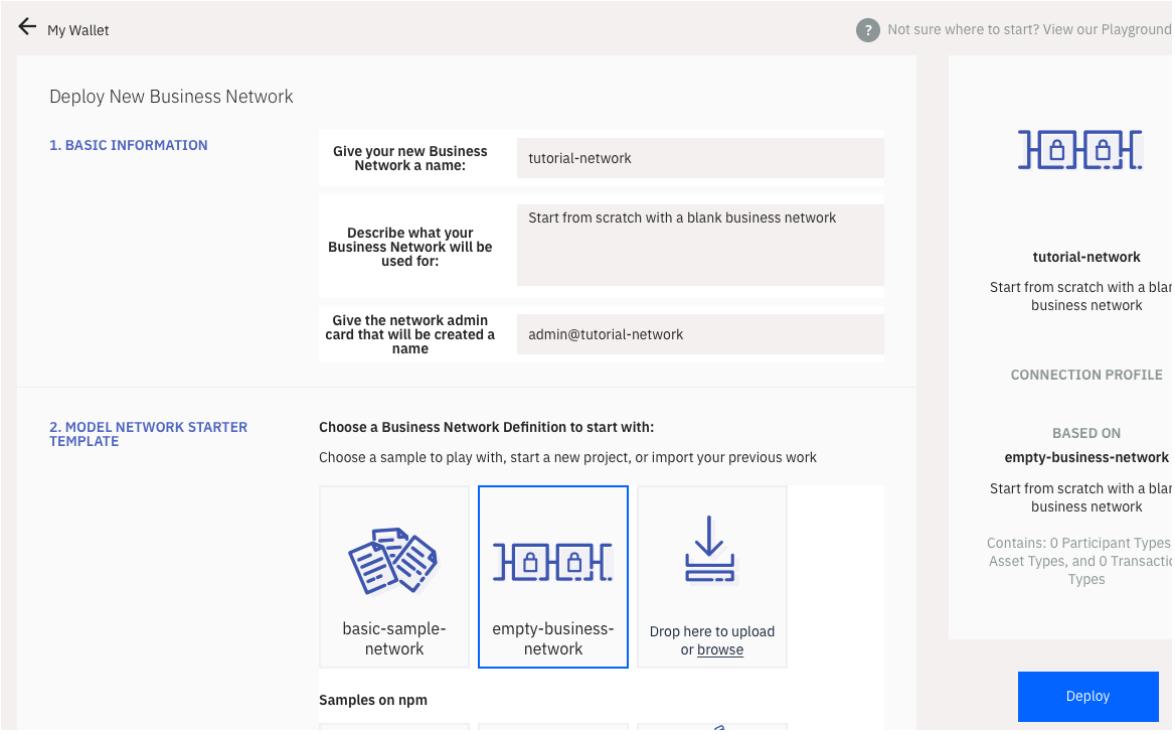
In this section you will learn about the modelling language Hyperledger Composer uses to define resources in its business networks. You will define a basic business network that allows participants to exchange marbles with each other, using this as a base from which to explore the language's features.

2.1. Creating new business network

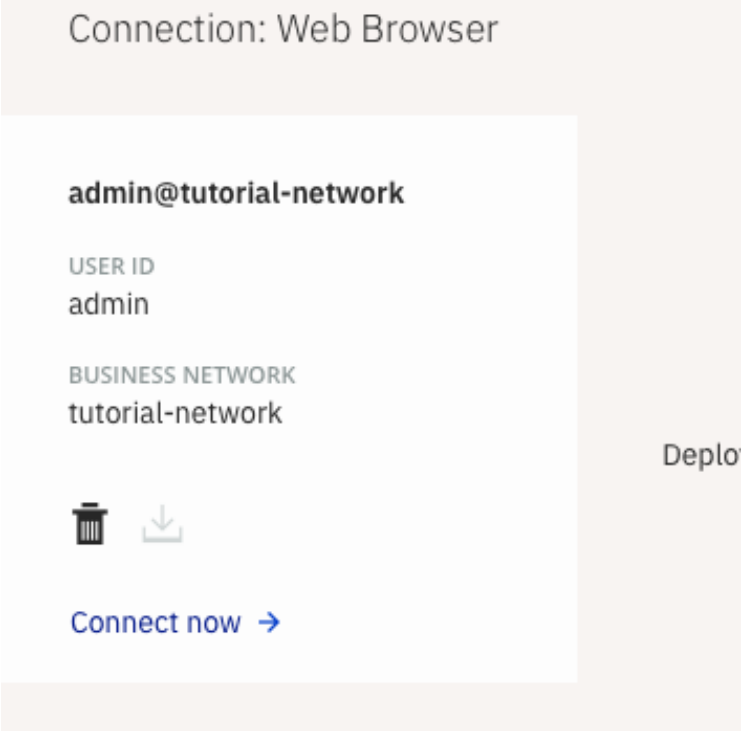
a. Select Connection: Web Browser and then click Deploy a new business network:



Enter the details – name of your new network, description and then the network admin card. Select '**Empty Business Network**' from the list and then click **Deploy**



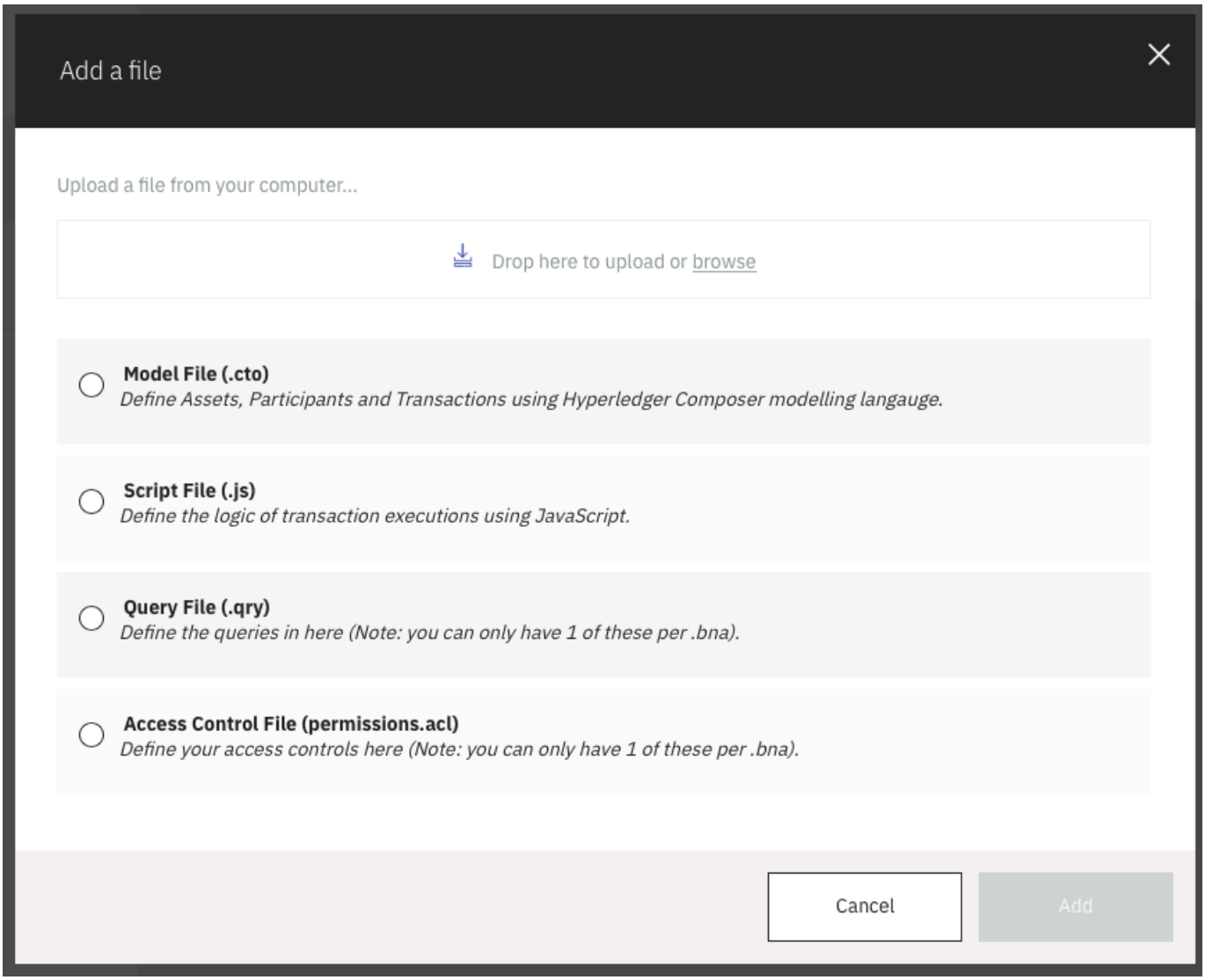
b. Connect to the network that has been created:



Press Connect now

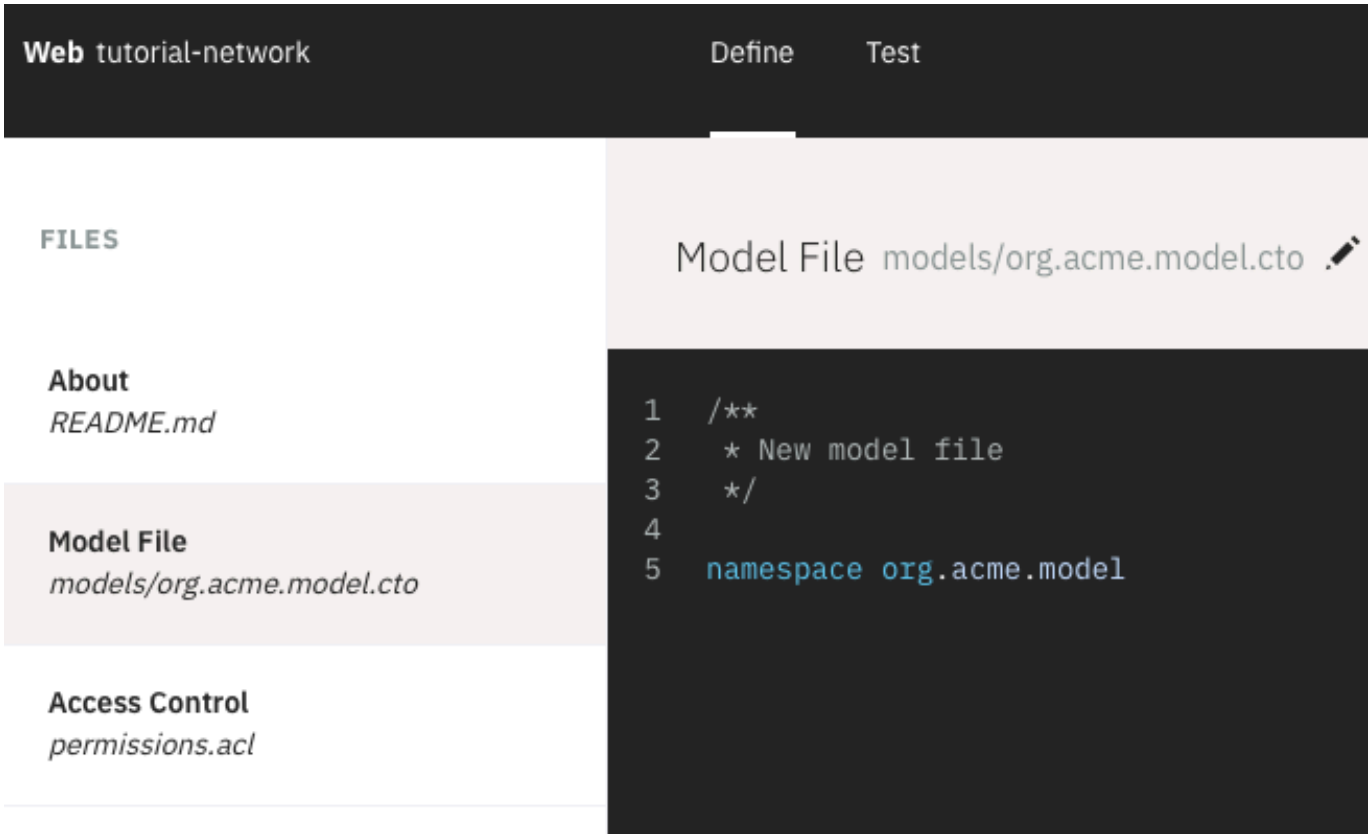
c. Adding files

We will be adding the following files that define the business network, transactions and access control.



d. Add a model file

Let's start with a Model file (.cto)



e. Resources

Composer's modelling language is first and foremost oriented around high level business concepts. As such, the **three top-level resources** that can be defined are as follows:

Assets	Participants	Transactions
Digital representations of assets that are recorded on the ledger.	*Individuals and Organisations that will contribute to and make use of the ledger.*	*Business logic governing the manipulation of assets.*

Additionally, each resource belongs to a namespace (a default namespace is at the top of the newly created file) which acts in a similar manner to how namespaces and packages work in other languages. In much the same way, resources can be imported from other namespaces. Namespace names can be any combination of letters and periods.

The modelling language describes these resources in a similar manner that you would describe a class in another language, this being an entity with attributes.

f. Writing an asset

Below is the **example** on how one can define an asset, participant and transaction. Please follow the lab's scenario to create your own.

Creating the asset `Marble`:

```
asset Marble identified by Id {
      o String Id
}
```

This defines a Marble asset and gives it an identifier to be referred to by (similar to the keys used in Fabric). Let's add the attributes:

```
asset Marble identified by Id {
      o String Id
      --> Collector owner
      o Integer diameter
      o String colour
}
```

You'll have noticed that the attributes in this do not all have the same prefix. The `owner` attribute is preceded by a `-->` instead of a `o`.

The o attributes are 'named fields' – they belong to the resource, for example the Marble will have a size and colour property.

The `-->` attributes are 'relationships' – while they make up the information that describes the resource they are not part of it, for example a Marble will have an owner but the owner is not part of the Marble.

The currently supported attribute types are as follows:

```
String, Boolean, DateTime, Integer, Double, Long
```

g. Writing a participant

Let's see how participants are defined:

```
participant Collector identified by email {
  o String email
}
```

Attributes for participants work in an identical manner to those of Marbles. As such, expand the `Collector` class:

```
participant Collector identified by email {
  o String email
  o String firstname
  o String surname

}
```

2.2. Writing a transaction

Transactions are also declared in the modelling file using the same syntax as with Assets and Participants, add the following to your modelling file:

```
transaction ChangeOwner {

      --> Marble marble
      --> Collector newOwner
}
```

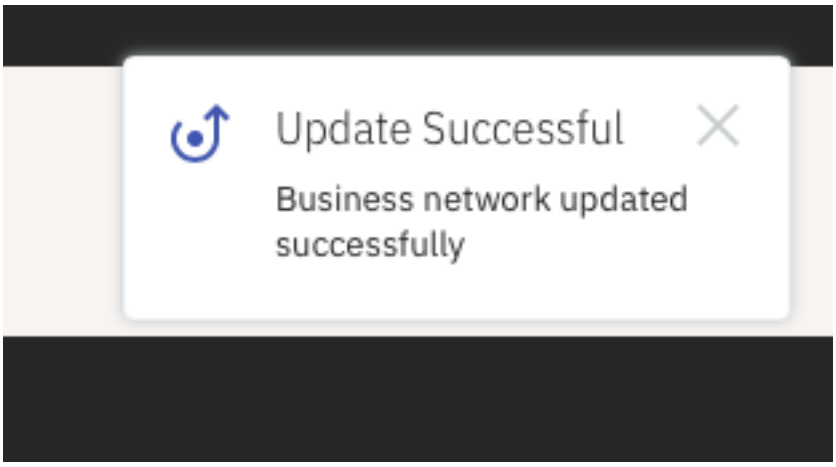
Instead of denoting attributes, the variables within the body of a transaction denote the arguments that the transaction logic function will take (this will be covered in more detail in the next section).

2.3. Deploy and create some test assets

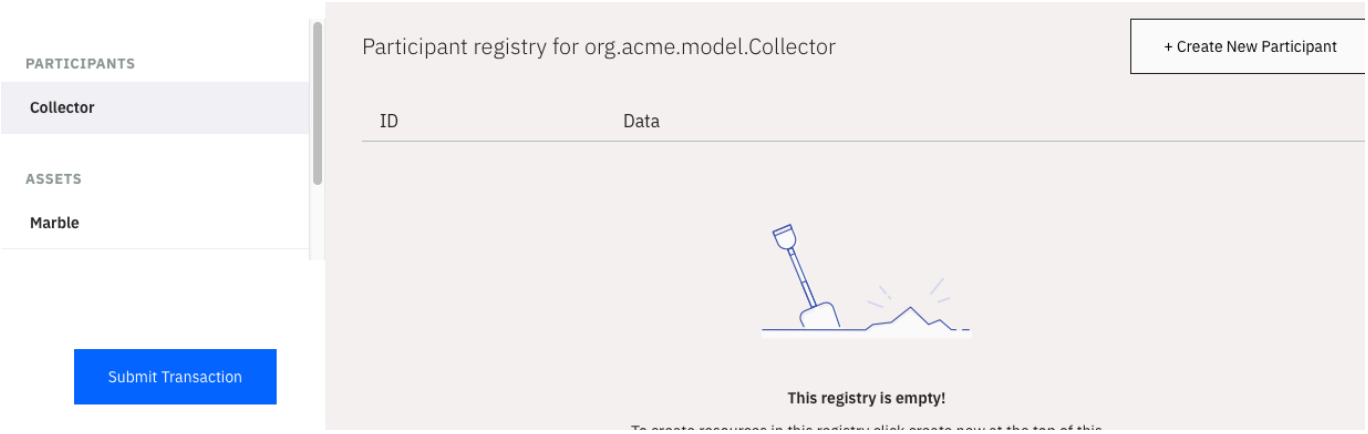
Now let's make sure that there were no syntax errors and let's update the business network with new definitions:

a. **Update the business network**

Click on the button **Update**



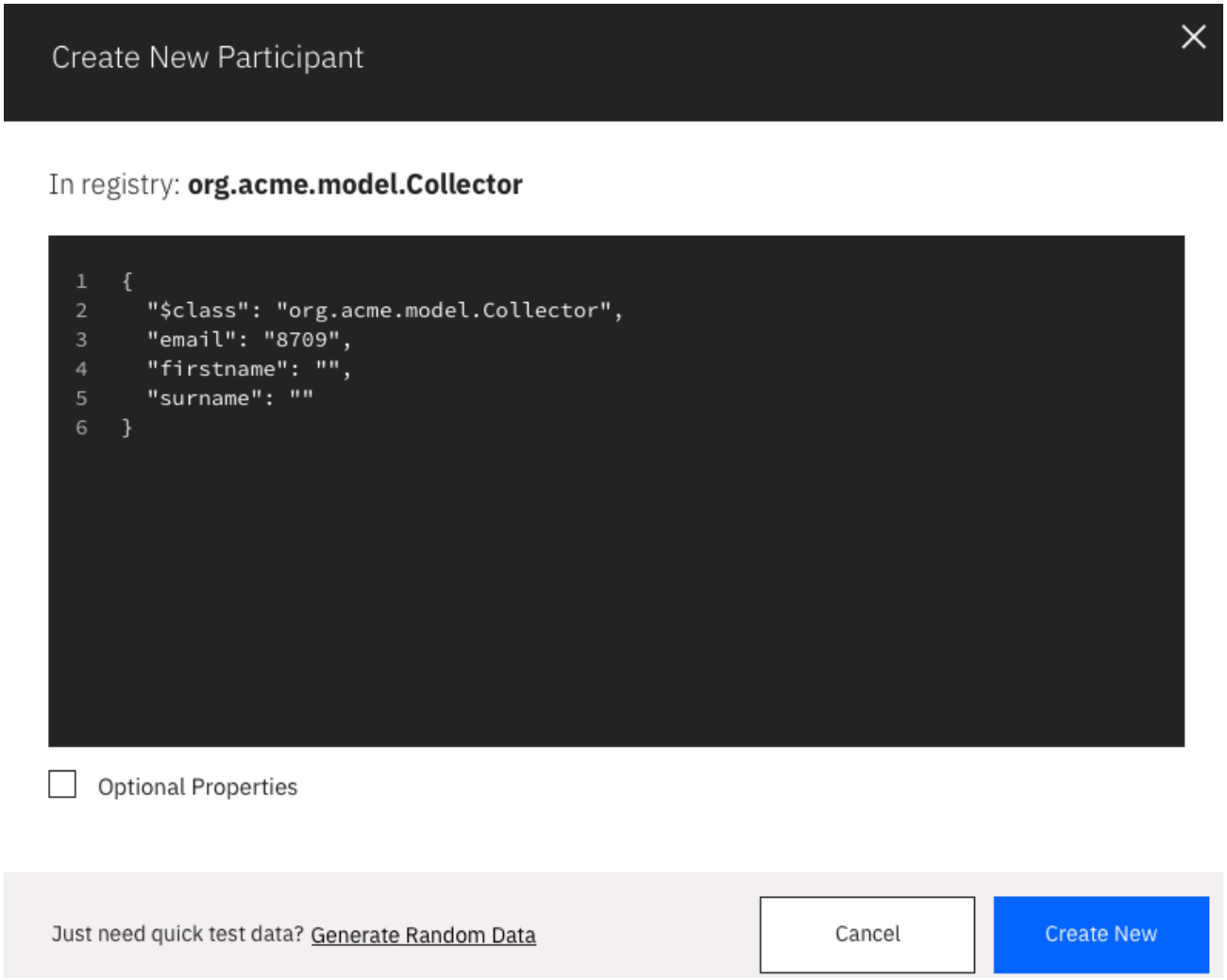
Now we have some asset definitions, hit the Update button on the left side of the screen. On success a small pop-up should appear. Go to the **Test** tab at the top:



Here you can see the assets and participants we've made.

b. Create the assets and participants

Hit **+ Create New Participant** on the top right. A dialogue box will appear prompting you to enter details of the new participant:



Enter the following:

```
{
  "$class": "org.acme.model.Collector",
  "email": "louis@yahoo.com",
  "firstname": "Louis",
  "surname": "Funes"
}
```

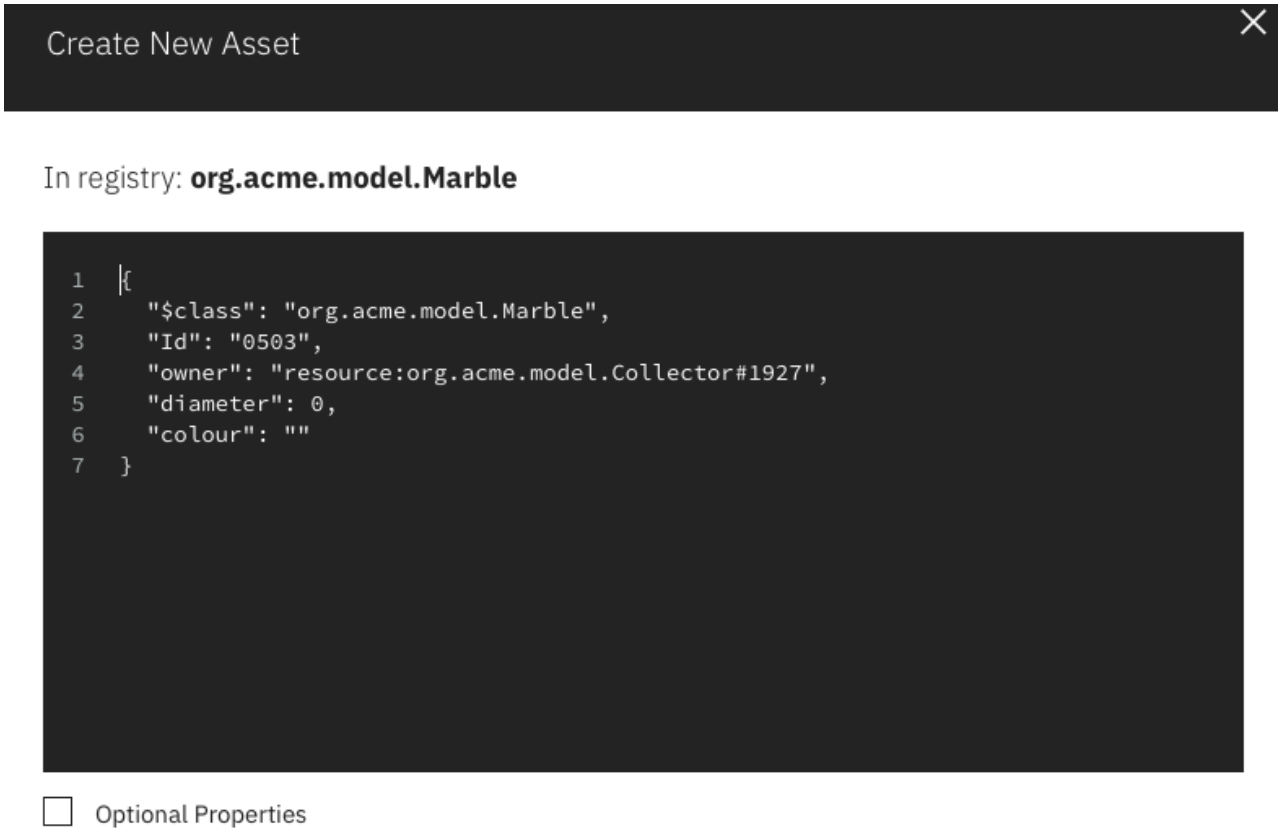
Fill this in and select **Create New**, you will see the new participant appear:

Create a second Collector with the following:

```
{
  "$class": "org.acme.model.Collector",
  "email": "jean@fr.com",
  "firstname": "Jean",
  "surname": "Marais"
}
```

Press **Create New**.

If you swap to the Marble asset and select **+ Create New Asset** you will see a similar dialogue box:



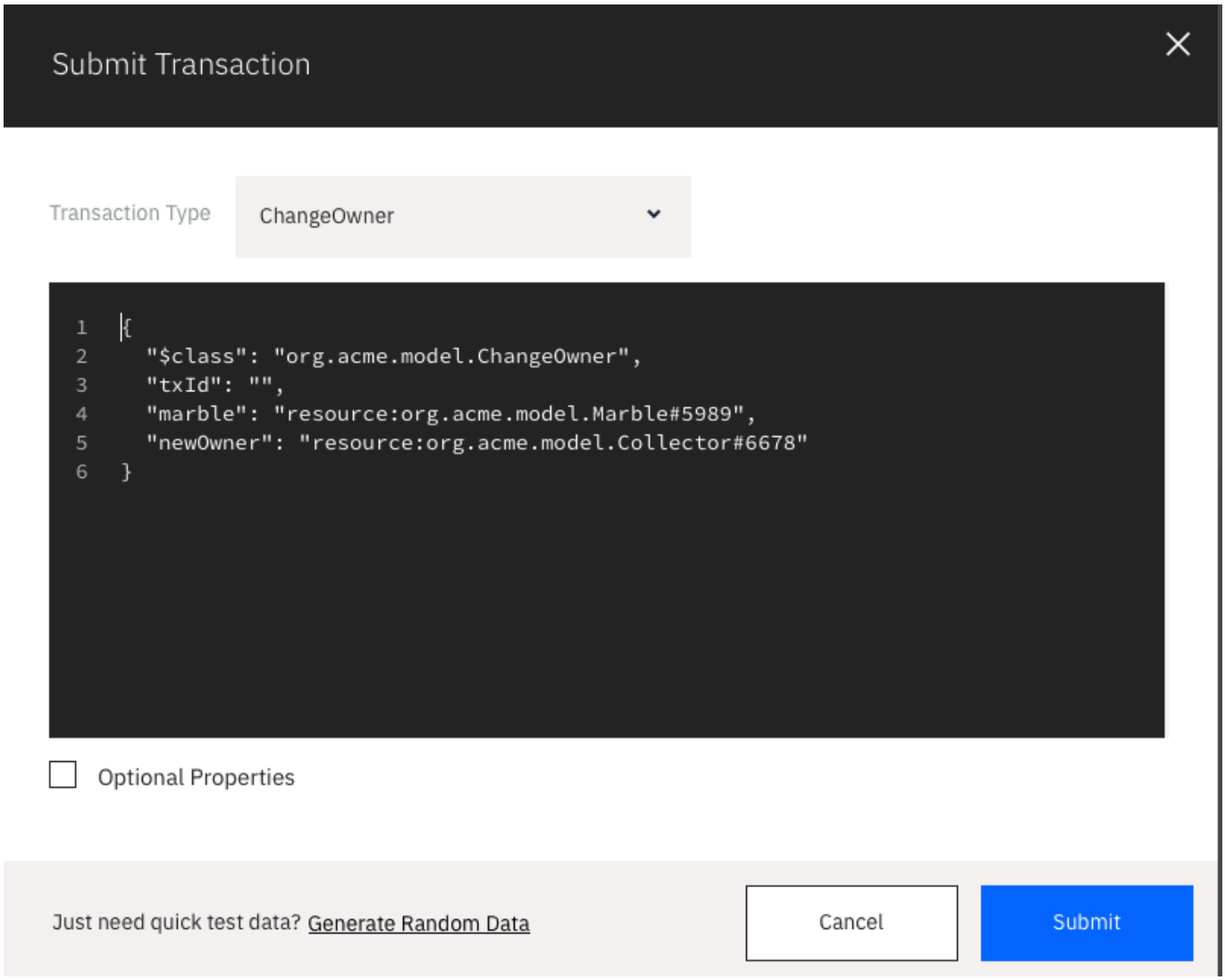
Enter the following. Note that when **filling out relationships**, **you must supply a fully qualified identifier** – this being as follows:

```
resource:<namepace>.<resource name>#identifier
```

```
{
  "$class": "org.acme.model.Marble",
  "Id": "1",
  "owner": "resource:org.acme.model.Collector#jean@fr.com",
  "diameter": 10,
  "colour": "red"
}
```

You will see the new asset appear.

If you select **Submit Transaction** at the bottom left you will see a similar dialogue box:



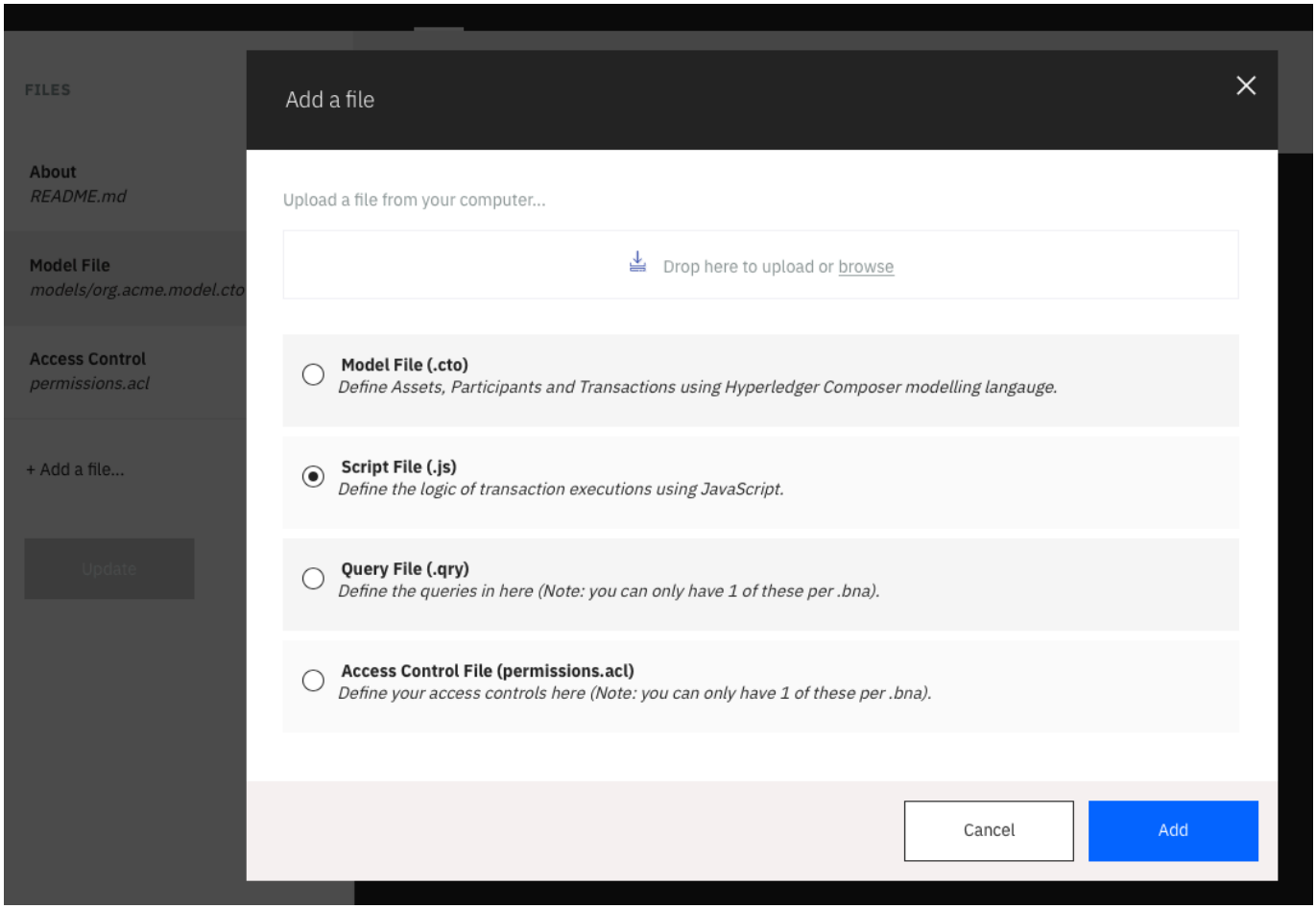
However if you submit this transaction, nothing will happen. We need to define some logic to associate with it. Before moving on to the next section, try creating another participant and an asset.

Section 3. Transaction Logic

In this section we will explore how to write transaction processor functions, these being the business logic that is executed when a transaction is invoked in Hyperledger Composer. Please note, while transaction processor functions are analogous to chaincode in their purpose we are not writing chaincode in this lab. Composer transaction logic, while achieving the same results, is not handled in the same way as chaincode is.

3.1. Create the logic file

Go back to the tab **Define** and select **+ Add a file…** from the left hand side and select **Script File (.js)** from the dialogue:



3.2. Add the changeOwner function

Within the new file add the following:

```
/**
 * @param {org.acme.model.ChangeOwner} args - the changeOwner transaction arguments
 * @transaction
 */
```

```
function changeOwner(args) {

}
```

Transaction processor functions are defined by writing a function with a JS Doc decorator that maps the first argument to the transaction's model definition. The `args` argument represents the incoming transaction, in particular the data packaged in it.

Recall the transaction's definition:

```
transaction ChangeOwner {
      --> Marble marble
      --> Collector newOwner
}
```

Args is an object where the keys are each of the attributes and the values are what has been attached to them during the transaction invocation.
`args` will have a `marble` and a `newOwner` attribute that are accessed the same way attributes are accessed in JS objects: `args.marble` or `args.newOwner`.

Transaction processor functions do not return anything, much like `Invoke` functions in Fabric, they simply execute and finish.

3.3. Add changOwner's body

The scenario of the lab is to change the owner of a marble. You will need to add 2 lines to the body of `changeOwner` function.

When the update has been made to the asset, we need to update the assets record in the world state. Add the following:

```
return getAssetRegistry('org.acme.model.Marble').then(function(marbleRegistry) {
       return marbleRegistry.update(args.marble);
});
```

Registries are indexes used by composer to store resources ; they store a reference to every instance of that particular resource. To update a resource, we get the registry (`getAssetRegistry(…)`) from its type and call the update function with the new version of the resources we want to update (composer will find it within the registry and update it for us).

Participants also have registries and are updated in the same way (although with `getParticipantRegistry`).

Notice that, while the language composer (used for its transaction process functions) is JavaScript, it only supports up to ES5, as such features like `() => {}` functions are not permitted. This is due to the Otto JavaScript engine that is currently used by composer. Otto is set to be replaced by an embedded version of node.js in a future release.

3.4. Test changeOwner

a. Create the assets and participants

Deploy the code and go to the Test tab. We are going to transfer a Marble between two Collectors. If you don't have 2 `Collectors` or a `Marble` follow the steps in 2.3 to create them:

Asset

ID	Data
1	{ "$class": "org.acme.model.Marble", "Id": "1", "owner": "resource:org.acme.model.Collector#jean@fr.com", "diameter": 10, "colour": "red" } Collapse

Participants

Define Test admin

Participant registry for org.acme.model.Collector + Create New Participant

ID	Data
jean@fr.com	{ "$class": "org.acme.model.Collector", "email": "jean@fr.com", "firstname": "Jean", "surname": "Marre" }
louis@yahoo.com	{ "$class": "org.acme.model.Collector", "email": "louis@yahoo.com", "firstname": "Louis", "surname": "Funes" }

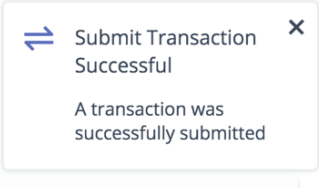
b. Submit the transaction

Select **Submit Transaction** from the sidebar and fill in the fields accordingly to select your marble and the Collector who is not the owner:

{

```
  "$class": "org.acme.model.ChangeOwner",
  "marble": "resource:org.acme.model.Marble#1",
  "newOwner": "resource:org.acme.model.Collector#louis@fr.com"
}
```

Select **Submit** to issue the transaction.
If successful, the following dialogue will appear:



A transaction entry will also appear: click on **All Transactions** menu, then click on **view record** in front or the first record of the list, which **Entry type** should be **ChangeOwner**. Then you will see the following window.

If you go back to the Marble, you will find its record has also been updated:

ID	Data

1	{
	"$class": "org.acme.model.Marble",
	"Id": "1",
	"owner": "resource:org.acme.model.Collector#louis@fr.com",
	"diameter": 10,
	"colour": "red"
	}
	Collapse

Section 4. Access Control

In this section, we will explore how Hyperledger Composer restricts access to the resources on the network and the ability to modify them.

4.1 Access Control Lists (ACL) - Grammar

We now have some digital assets defined and the ability to move them between users. However, in a real system, it would likely be the case that the `Marble` objects would not be available for all to see and if they were they would not be available for just anyone to change the ownership of.

ACL Rules are of the following format:

```
rule <Rule Name> {
    description: <description of the rule>
    participant(p): <namespace and name of the participant performing the action>
    operation: <operation the participant wishes to perform>
    resource(r): <resources the operation is being performed on>
    condition: (<condition under which this rule applies>)
    action: <does this rule allow an operation or deny it>
}
```

In more detail:

Participant is the person or entity that has submitted the transaction.

Operation is what they wish to do to this resource, supported operations are `CREATE, READ, UPDATE, DELETE, ALL`

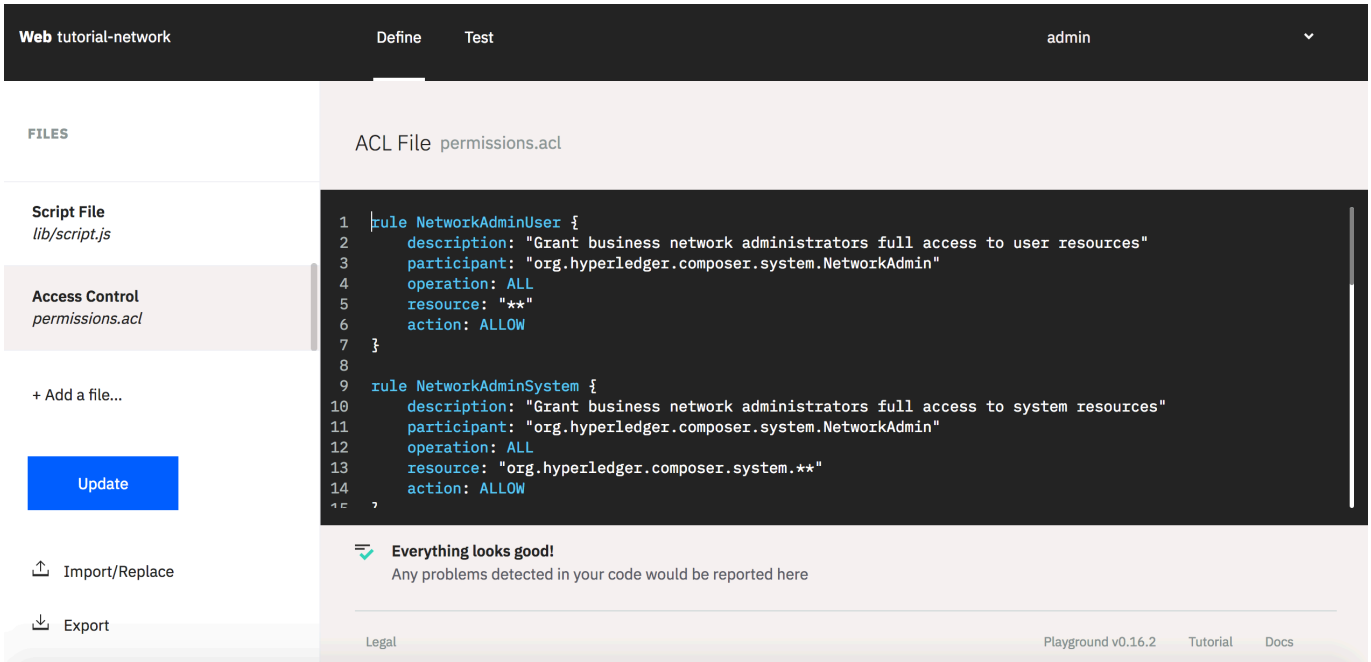
Resource is the asset that the transaction is being applied to. Resources (and indeed Participants) can simply be namespaces in which case they apply to all participants/resources in that namespace.

Condition is a JavaScript statement that can examine the participant and resource to check for certain conditions. Anything valid for use in an `if` statement is valid here.

Action is a simple ALLOW or DENY, as the name suggests this allows or denies the transaction.

4.2 Adding rules

Go to the **Define** tab, and select the **Acces Control (permission.acl)** file:



Add the following rule:

```
rule OnlyOwnerCanUpdateAMarble {
    description: "Only an owner can edit a marble"
    participant(p): "org.acme.model.Collector"
    operation: UPDATE
    resource(r): "org.acme.model.Marble"
    condition: (r.owner.getIdentifier() == p.getIdentifier())
    action: ALLOW
}
```

This rule ensures that only the owners of `Marble` resources are able to edit them. It does this by `ALLOW`ing an `UPDATE` to `org.acme.model.Marble` resources only when the identifier of the `participant` and the `resource`'s owner are the same.

By default, all action is restricted unless explicitly permitted. As such while we do have a rule allowing updates of a `Marble` resource even the owner would be unable to read it. Add the following:

```
rule AnyoneCanReadMarbles {
    description: "All the participants can read the marble"
    participant(p): "org.acme.model.Collector"
    operation: READ
    resource(r): "org.acme.model.Marble"
    condition: (true)
    action: ALLOW
}
```

This rule allows all `Collector` participants to `READ` all `Marbles`.
We'll also need a rule to let `Collectors` read each other. The `ChangeOwner` transaction requires a submission of the identifier of a new `owner` which will not be possible if `Collectors` cannot read each other:

```
rule AnyoneCanReadCollectors {
    description: "Only an owner can edit a marble"
    participant(p): "org.acme.model.Collector"
    operation: READ
    resource(r): "org.acme.model.Collector"
    condition: (true)
    action: ALLOW
}
```

To update the Marble through the ChangeOwner transaction, another rule is needed. In particular we need to allow participants to create change owner transactions:

```
rule AnyoneCanIssueChangeOwner {
    description: "The participants can use the ChangeOwner transaction to update the
marble"
    participant(p): "org.acme.model.Collector"
    operation: CREATE
    resource(r): "org.acme.model.ChangeOwner"
    condition: (true)
    action: ALLOW
}
```

Even with this rule, the transaction could be created by a non-owner but would still be rejected as they lack update access.

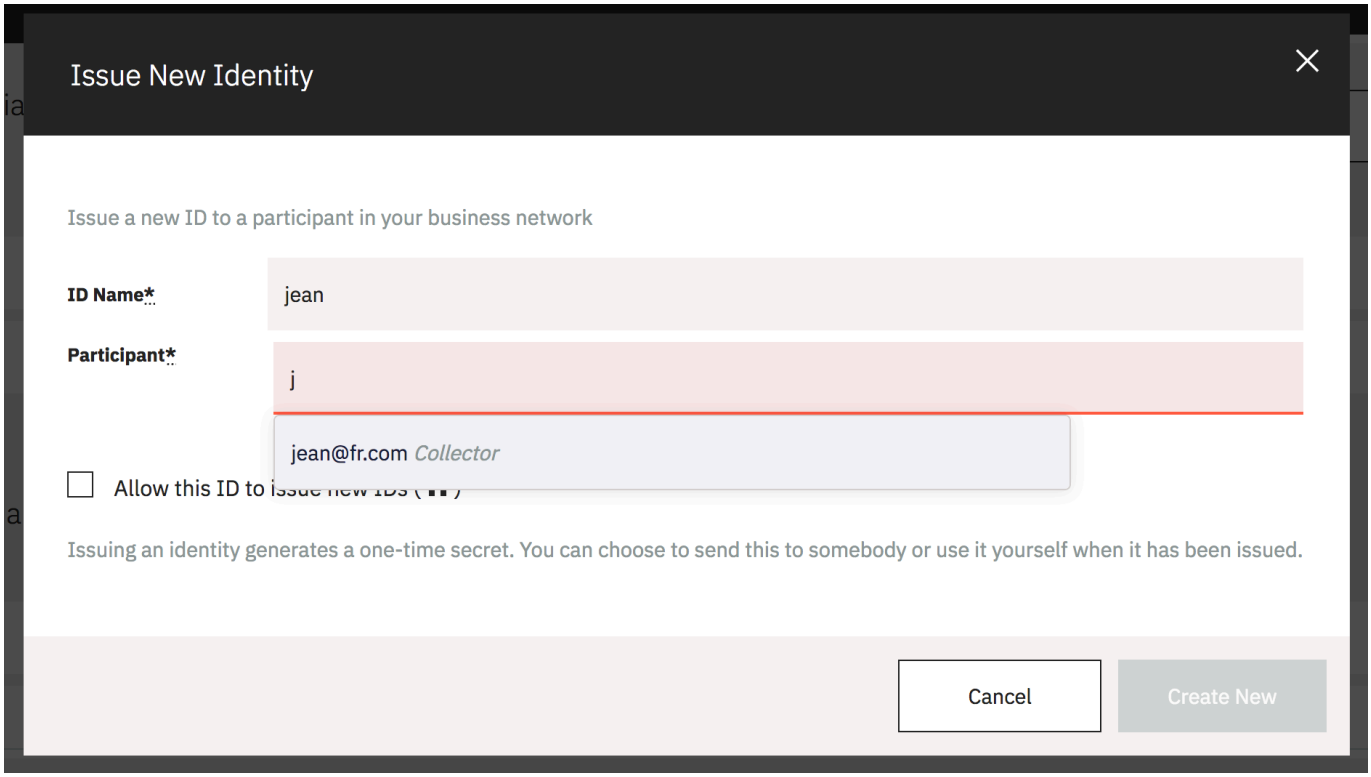
And finally, we are adding a rule in order to let the participants access to the objects through the composer for the purpose of the coming test :

```
rule ParticipantCanReadNetwork {
    description: "Participant can read the business network"
    participant(p): "org.acme.model.Collector"
    operation: ALL
    resource(r): "org.hyperledger.composer.system.**"
    condition: (true)
    action: ALLOW
}
```

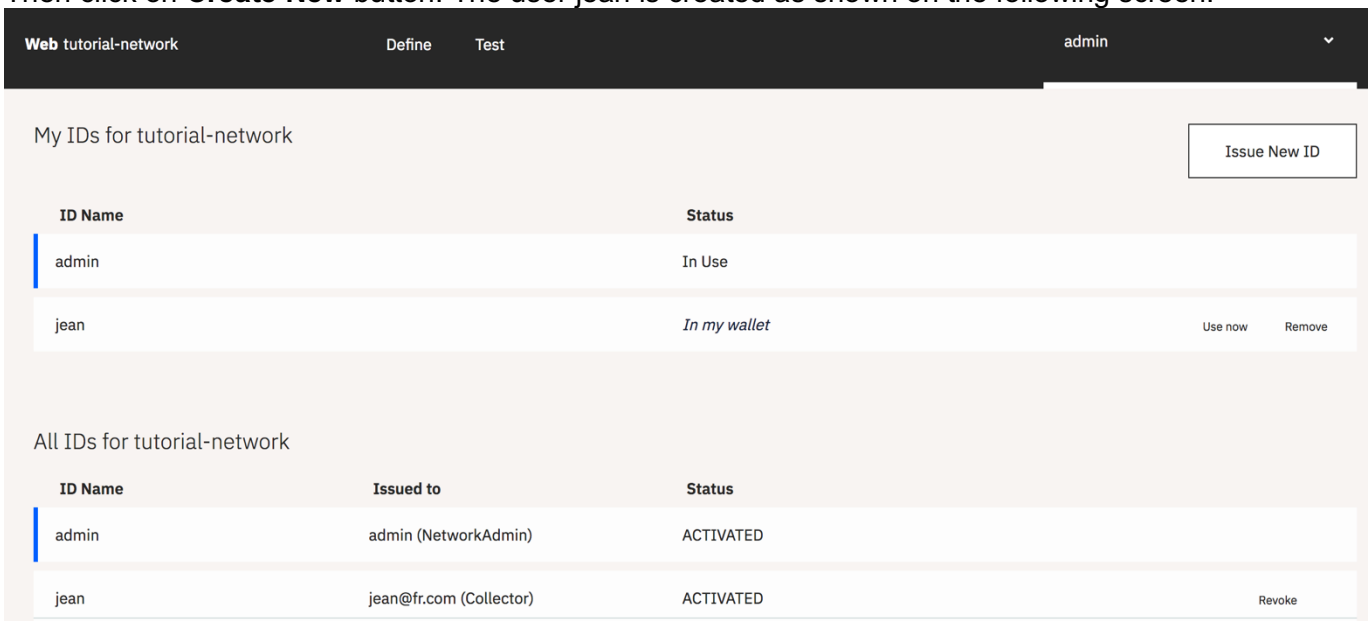
Now, you have completed the ACL changes.
Click on the **Update** button to take into account the ACL.

4.3 Testing the rules

Now, we will create a new user for the coming test: click on the top right menu (beside the user “admin”), then click on the **Id Registry** menu.
Create a new user clicking on the top right button : **Issue New ID**
On the Issue New Identity window, fill in
ID Name : jean
Participant : jean@fr.com



Then click on **Create New** button. The user jean is created as shown on the following screen.



Then click on the line “jean” in the “My IDs...” list. It will select **jean** as the new user (**jean** is displayed instead of **admin** in the top right).

Go to the test window (Click on Test) then click on the **Submit Transaction** button.
Fill in the following info then click on submit :

```
{
  "$class": "org.acme.model.ChangeOwner",
  "marble": "resource:org.acme.model.Marble#1",
```

```
  "newOwner": "resource:org.acme.model.Collector#jean@fr.com"
}
```

You should get an error since the Marble #1 is owned by louis@fr.com and there is a rule which restricts the update of Marble to the owner.
So change from **jean** to **admin** user :
- click on the top right menu (beside the user “jean”), then click on the **Id Registry** menu,
- Then click on the line “admin” in the “My IDs...” list. It will select **admin** as the new user (**admin** is displayed instead of **jean** in the top right).

Go to the test window (Click on Test) then click on the **Submit Transaction** button.
Fill in the following info then click on submit :

```
{
  "$class": "org.acme.model.ChangeOwner",
  "marble": "resource:org.acme.model.Marble#1",
  "newOwner": "resource:org.acme.model.Collector#jean@fr.com"
}
```

The transaction is successful and the marble #1 is now owned by jean.

So switch back from admin to jean :
- click on the top right menu (beside the user “jean”), then click on the **Id Registry** menu,
- Then click on the line “jean” in the “My IDs...” list. It will select **jean** as the new user (**jean** is displayed instead of **admin** in the top right).

Go to the test window (Click on Test) then click on the **Submit Transaction** button.
Fill in the following info then click on submit :

```
{
  "$class": "org.acme.model.ChangeOwner",
  "marble": "resource:org.acme.model.Marble#1",
  "newOwner": "resource:org.acme.model.Collector#louis@fr.com"
}
```

The transaction is successful and the marble #1 is now owned by louis.

Notices

This information was developed for products and services offered in the U.S.A.
IBM may not offer the products, services, or features discussed in this document in other countries.

Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.
IBM may have patents or pending patent applications covering subject MGKer described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.
For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:
IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan
The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.
This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.
Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.
IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.
Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.
Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM

Appendix A. Trademarks and copyrights

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM	AIX	CICS	ClearCase	ClearQuest	Cloudscape
Cube Views	DB2	developerWorks	DRDA	IMS	IMS/ESA
Informix	Lotus	Lotus Workflow	MQSeries	OmniFind	
Rational	Redbooks	Red Brick	RequisitePro	System i	
System z	*Tivoli*	*WebSphere*	*Workplace*	*System p*	

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.
IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency which is now part of the Office of Government Commerce.
Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.
Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.
Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.
ITIL is a registered trademark, and a registered community trademark of The Minister for the Cabinet Office, and is registered in the U.S. Patent and Trademark Office.
UNIX is a registered trademark of The Open Group in the United States and other countries.
Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.
Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.
Linear Tape-Open, LTO, the LTO Logo, Ultrium, and the Ultrium logo are trademarks of HP, IBM Corp. and Quantum in the U.S. and other countries.

NOTES

NOTES