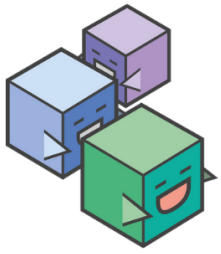
create first then learn

PLAYTIME WITH HYPERLEDGER COMPOSER



Create a supplychain management project in Blockchain using Hyperledger Composer

SHUBHAM CHADOKAR

License

2020

Preface

For the last 2 years, I am working as a Blockchain developer. In the last 2 years, I came across many blockchain platforms and blockchain development tools. Among all, Hyperledger Composer stand out from the rest.

> **The Hyperledger Composer is deprecated on 18th August 2019.**

Even it is deprecated, it is still a great tool to create a Proof of Concept (POC) on Hyperledger Fabric.

For a student, creating a project on latest technology like blockchain is great for his profile and experience.

What you should expect from this eBook?

This eBook is more about hands-on and less about theory. This eBook is not for someone who is seeking how composer actually works or an in-depth explanation of the composer's concepts.

This eBook covered the basic definition of all the concepts used while developing the project.

You will learn, how to create a virtual machine in the cloud, setting up the composer, composer design, developing the project, deploying and testing the project in composer playground, generating the REST APIs for the project using the composer-rest-server and instructions on how to create the frontend of the project and hosting it on GitHub Pages.

Should have

- You should have a basic understanding of blockchain.
- Basics of JavaScript. In the composer, JavaScript knowledge is very essential.
- Understanding of Objects and Promises in JavaScript. Check out the references section to learn about Objects and Promise.

*****

About Me

Hello, I am ***Shubham Kumar Chadokar***.

I am a Software Engineer and in my short career of almost 4 years, I've had the opportunity to work on Blockchain, Nodejs, Golang, and Docker.

I've learned about other tech as well, but these are my primary focus. I love to write articles and tutorials on new tech by following a hands-on approach. This is my first book.

Front end development isn't my specialty, and that's why I didn't include it in the book.

If you have any queries or questions, please feel free to drop me an email.

✉ hello@schadokar.dev
schadokar.dev
github.com/schadokar

*****

Thank You 🙏

I especially want to thanks the entire Hyperledger Composer Community for creating such an amazing tool. Many developers entered into the blockchain domain because of the simplicity of the composer.

It is unfortunate that it is deprecated but it sets a great example of easy automation, wrapping a complex Hyperledger Fabric into the easy to use Hyperledger Composer.

Table of Contents

*****

What is Blockchain and Hyperledger Fabric?

Before we start exploring Hyperledger Composer, it is better to know about Blockchain, Hyperledger and Hyperledger Fabric.

What is Blockchain?

Blockchain is a distributed database. The data is stored in a cryptographical secured block. These blocks are linked to each other in a sequence. There is no going back in the blockchain. All the blocks and data in it is time-stamped. All the operations like create, update or delete the data append after the latest block. In this way, it is easy to verify the change in the state of the data.

To understand how blockchain is different than the existing database, consider the existing database system as ***write using pencil*** and blockchain as ***write using pen***.

In ***write using pencil***, the information on a paper written using a pencil can be erased, update or delete without any trace. It is very difficult to examine the authenticity of the information.

While ***write using pen***, the information can't be easily updated or deleted. Even if it updates, it is easy to determine the change.

> It is an example for understanding the concept. There is no such ***write using pen*** concept in the current system. But, similar to this can be achieved. Create multiple copies of the data ***write using pencil*** and distribute with all the participants. To create the data, every copy needs to create it. To update the data, every copy needs to update it. If someone tries to update the data without informing the others, now it is easy to verify the compromised copy.

What is Hyperledger?

Hyperledger is a project of The Linux Foundation Project. Hyperledger is an open-source community focused on developing a suite of stable frameworks, tools, and libraries for enterprise-grade blockchain frameworks.

- **Frameworks:** Hyperledger Fabric, Hyperledger Sawtooth, Hyperledger Indy etc.
- **Tools:** Hyperledger Composer, Hyperledger Explorer, Hyperledger Cello etc.
- **Libraries:** Hyperledger Aries, Hyperledger Quilt, Hyperledger URSA etc.

What is Hyperledger Fabric?

Hyperledger Fabric is an enterprise-grade, distributed ledger platform that offers modularity and versatility for a broad set of industry use cases. The modular architecture for Hyperledger Fabric accommodates the diversity of enterprise use cases through the plug and play components, such as consensus, privacy and membership services.

*****

Introduction of Hyperledger Composer

Hyperledger Composer is an extensive, open development toolset and framework to make developing blockchain applications easier.

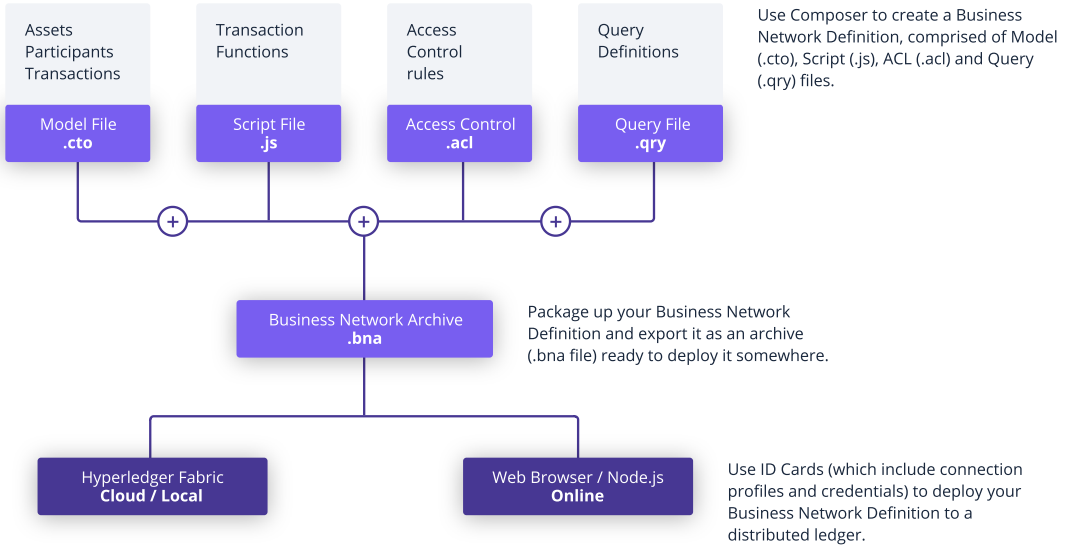
> Hyperledger Composer supports the existing Hyperledger Fabric blockchain infrastructure and runtime, which supports pluggable blockchain consensus protocols to ensure that transactions are validated according to the policy by the designated business network participants. Learn more.

> Hyperledger Composer supports only Hyperledger Fabric v1.1 and v1.2.

How Hyperledger Composer Works?

Hyperledger Composer has 3 basic unit:

1. **Assets:** Shipment, House, or any object
2. **Participants:** Buyer, Seller or anyone who can own the asset
3. **Transactions:** Buying, Selling, Receiving or actions between two or more participants on the asset.



Business Network Archive (BNA)

BNA is a package which packages the complete business logic and dependency in it.
`.bna` is an extension.

There are 4 types of files in the bna:

1. **Model File** (.cto): In the model file, we use modelling language to define business logic. Its extension is `cto`. In the model file, we declare all the assets, participants, and transactions required in the business logic. We only declare them, no functionality.
2. **Transaction File** (.js): In the transaction file, we define the functionality of the transaction which is declared in the model file. The transaction file is in javascript.
3. **Access Control** (.acl): In the access control file, we define the roles and permissions of the participants. Hyperledger Composer includes an access control language (ACL) that provides declarative access control over the elements of the domain model. By defining ACL rules you can determine which users/roles are permitted to create, read, update or delete elements in a business network's domain model. Learn more.
4. **Query File** (.qry): A query is a request for the data from the data set. Queries in Hyperledger Composer are written in a bespoke query language. Queries are defined in a single query file called (queries.qry) within a business network definition.

*****

Environment Requirement

As per the documentation, Hyperledger Composer pre-requisites can be installed on Ubuntu or macOS. Windows is not officially supported.

System Requirement:

1. Operating Systems: Ubuntu Linux 14.04 / 16.04 LTS (both 64-bit), or Mac OS 10.12
2. At least 4GB RAM
3. Minimum 40GB free space

Pre-requisites

- **Docker Engine:** Version 17.03 or higher
- **Docker-Compose**: Version 1.8 or higher
- **Node**: 8.9 or higher (note version 9 and higher is not supported)
- **npm**: v5.x
- **git**: 2.9.x or higher
- **Python**: 2.7.x

Windows user

If you are using Windows, don't waste your time installing it. It will give you lots of trouble. Instead of that, you can use one of these options:

1. Creating a Linux virtual machine on windows using **Hyper-V** or using similar tools like **VirtualBox**.

   > ⚠ Please don't install VirtualBox without reading the instructions for windows. On many Windows machine, it is not supported and it might conflict with the **Hyper-V**. Use **Hyper-V** to save yourself from trouble.

2. Creating a Linux Virtual machine on Cloud. This option is more promising than the above one. First, it will save you from setting up the VM. Second, many cloud providers are very generous to give free credits. If used properly, these free credits are more than enough for the project. And last you will get some exposure to Cloud.

In this e-book, we're using 2nd option the cloud. You will find all the necessary instructions and knowledge require to set up a VM in the cloud.

You can create a VM in any of the below cloud providers. In the next sections, instructions are given for each listed cloud providers.

1. Google Cloud Provider (GCP)
2. Amazon AWS
3. Microsoft Azure

> 💡 I recommend to use **GCP** as it gives ***300 USD*** value of free credits for ***12 months***.

*****

Create a Linux VM in Azure

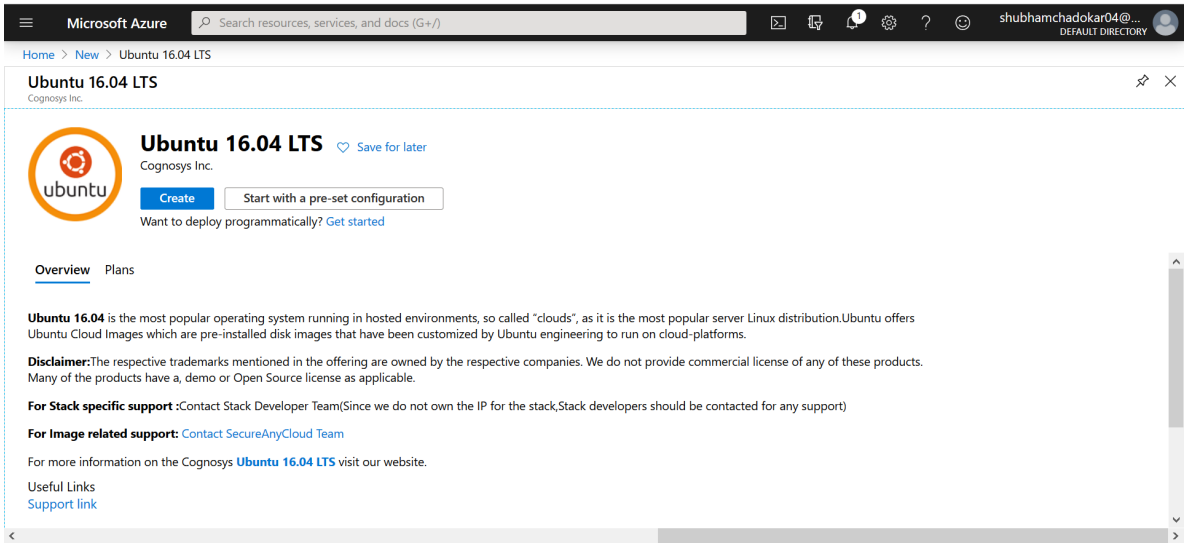
Create a new account in Microsoft Azure. For new users, Azure gives free credits for 30 days and many services free of cost upto a limit for 12 months.

For students, Microsoft also give away many resources and azure credits. You just have to reach to them by their email.

Login to your azure account.

Step 1 Search for Ubuntu 16.04 LTS

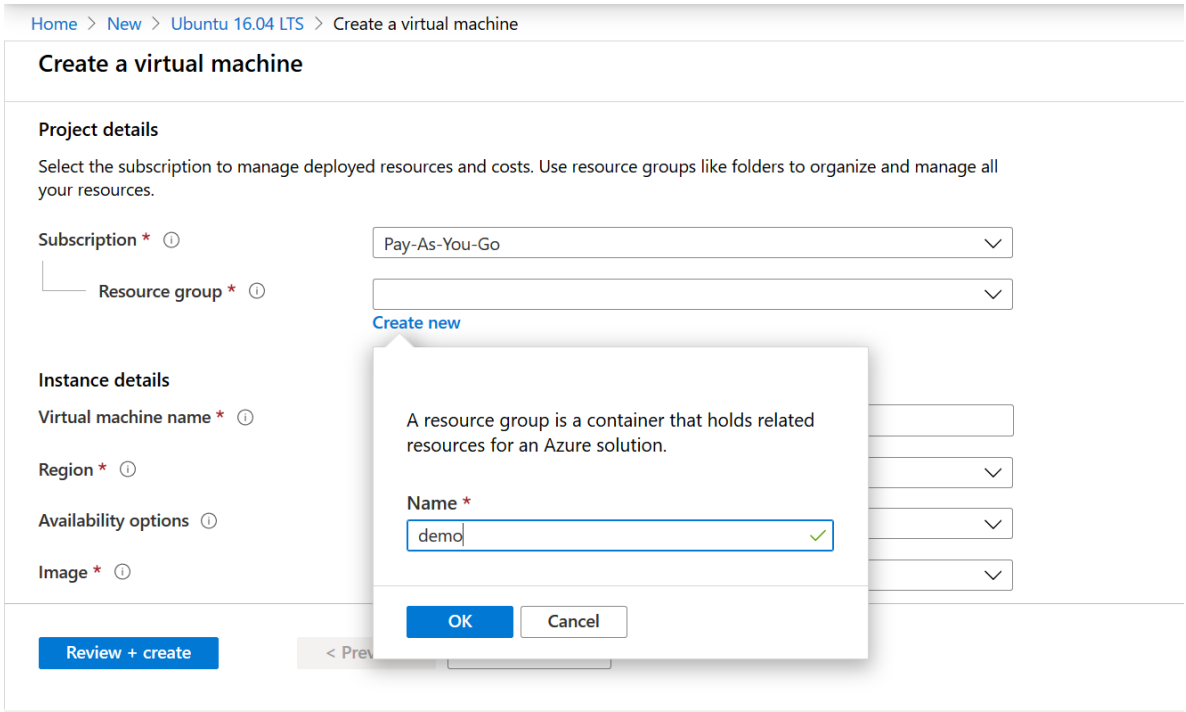
In the search bar, search **Ubuntu 16.04 LTS** and select create.



Step 2 Enter the Project details

In azure, all the related resources must be in a container, this container is known as Resource Group.

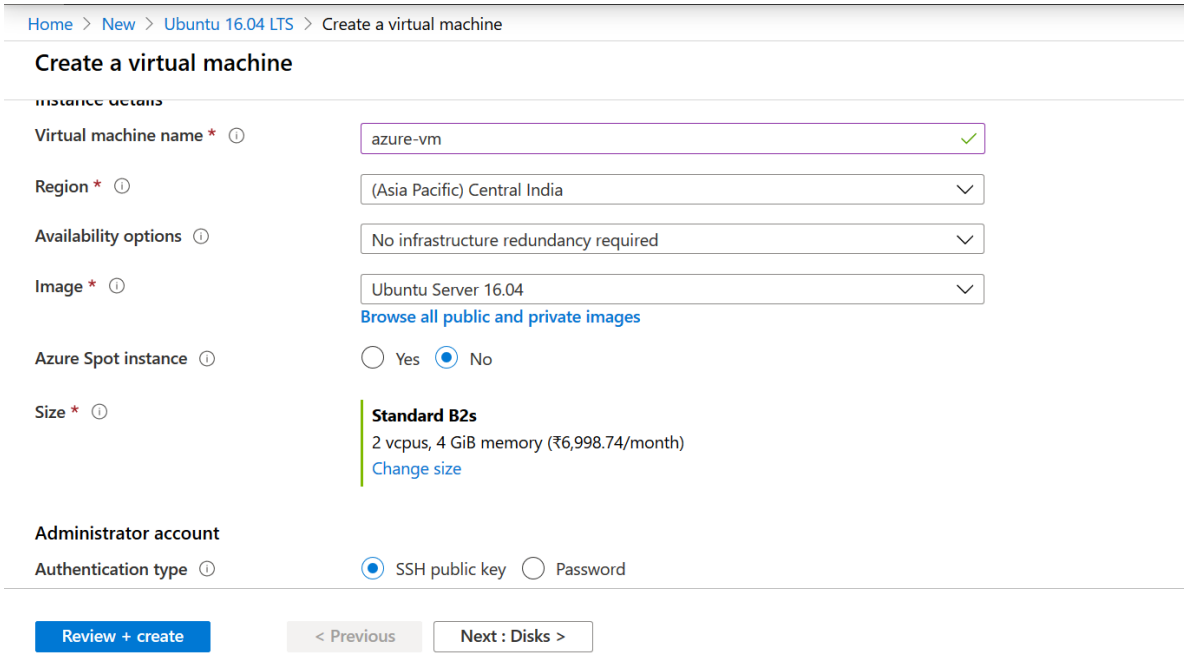
Create a new resource group **demo**.



Enter Instance Details

Enter the virtual machine name **azure-vm**.

Change the VM size to **Standard B2s**.



> The VM's price or default configuration may vary region to region.

Administrator account

There are 2 options to login in the VM, using the SSH key or the password.

> Azure currently supports SSH protocol 2 (SSH-2) RSA public-private key pairs with a minimum length of 2048 bits.

Let's setup the SSH key.

Open any command line tool. Like bash shell.

Use `ssh-keygen` command to create keys. By default it saves these keys in `~/.ssh` directory.

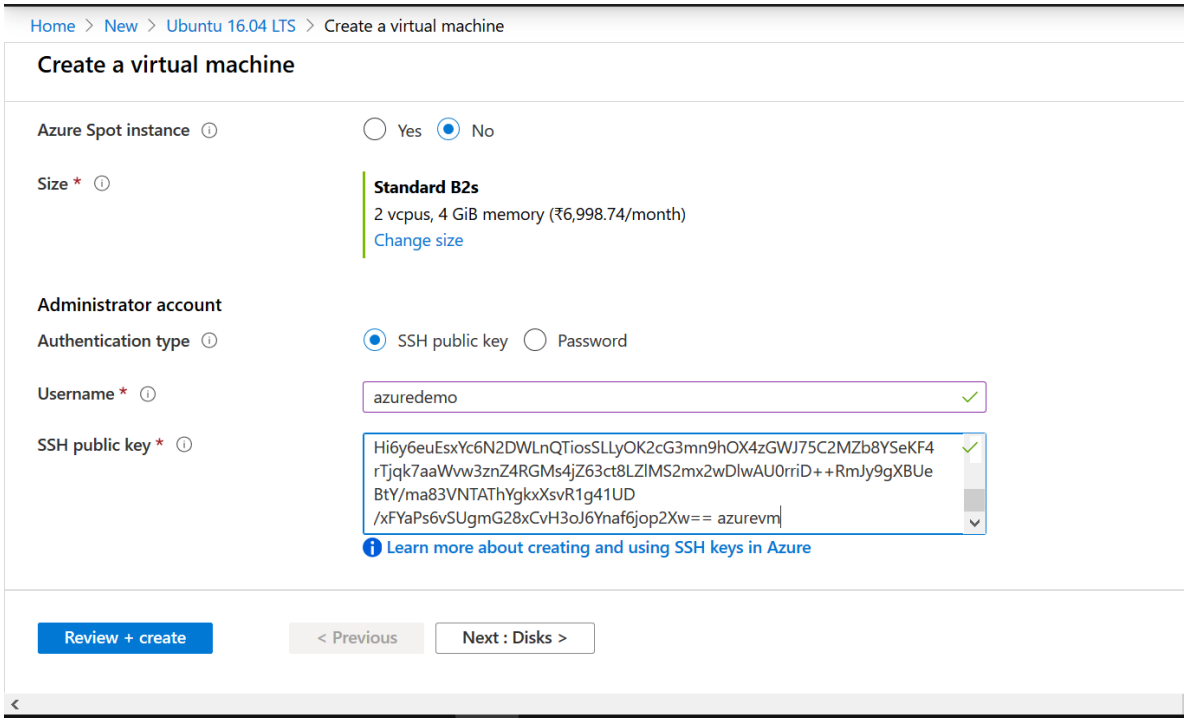
```
$ ssh-keygen -m PEM -t rsa -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/admin/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/admin/.ssh/id_rsa.
Your public key has been saved in /c/Users/admin/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:9Hl7Wfo6Et4+QfQ7sKrZr2yaO7Dgb4IVLSIpgsY9ftE admin@LAPTOP-BAVLQDM1
The key's randomart image is:
+---[RSA 4096]----+
|                 |
|              .  |
|o o   o .    . . |
|+= + + E . . .. .|
|+ o o + S o ..o o|
|   . + .   ..o.* |
|    = . o  .oo+..|
```

```
|    . o o .=oo.+. |
|       +. **+o+o+.|
+----[SHA256]-----+
```

Enter the absolute path where you want to save the keys or skip it and press Enter.

Passphrase is required to access the private key while login in the VM.

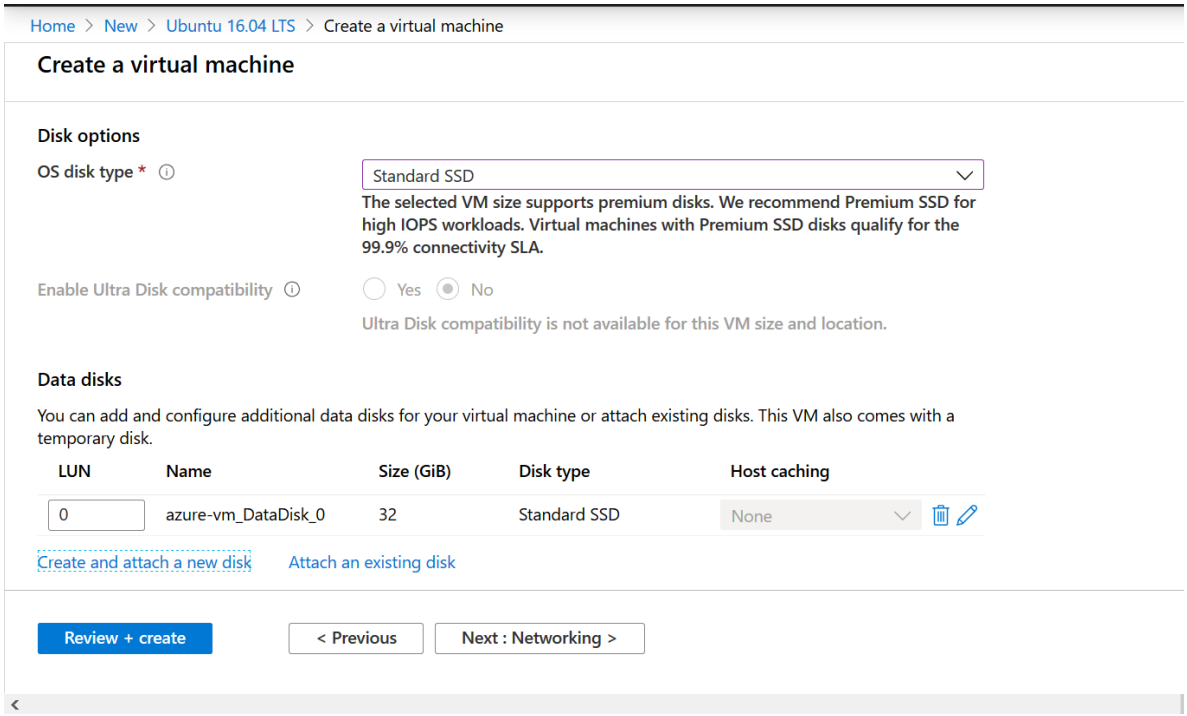
Enter Username `azuredemo`.
Open the `id_rsa.pub` file. This is the public key.
Copy the key and paste it in **SSH public key**.



Click **Next**.

Step 3 Disk

In the Disk section, you can add extra disk space as per your requirement. The Standard B2s VM comes with 32 GB temporary storage.



> For the demo purpose, we don't need extra storage.

Click **Next** to Networking.

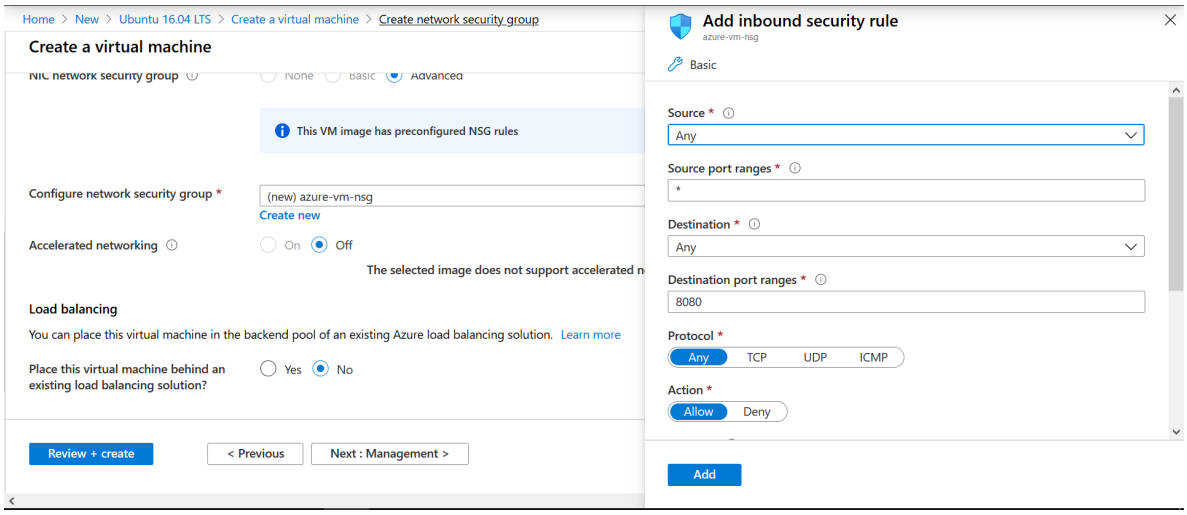
Step 4 Networking

In networking, you can define network connectivity for the virtual machine.

For the demo purpose, we are not defining network restriction. Anyone can reach the VM.

Create a new **Network Security Group**.

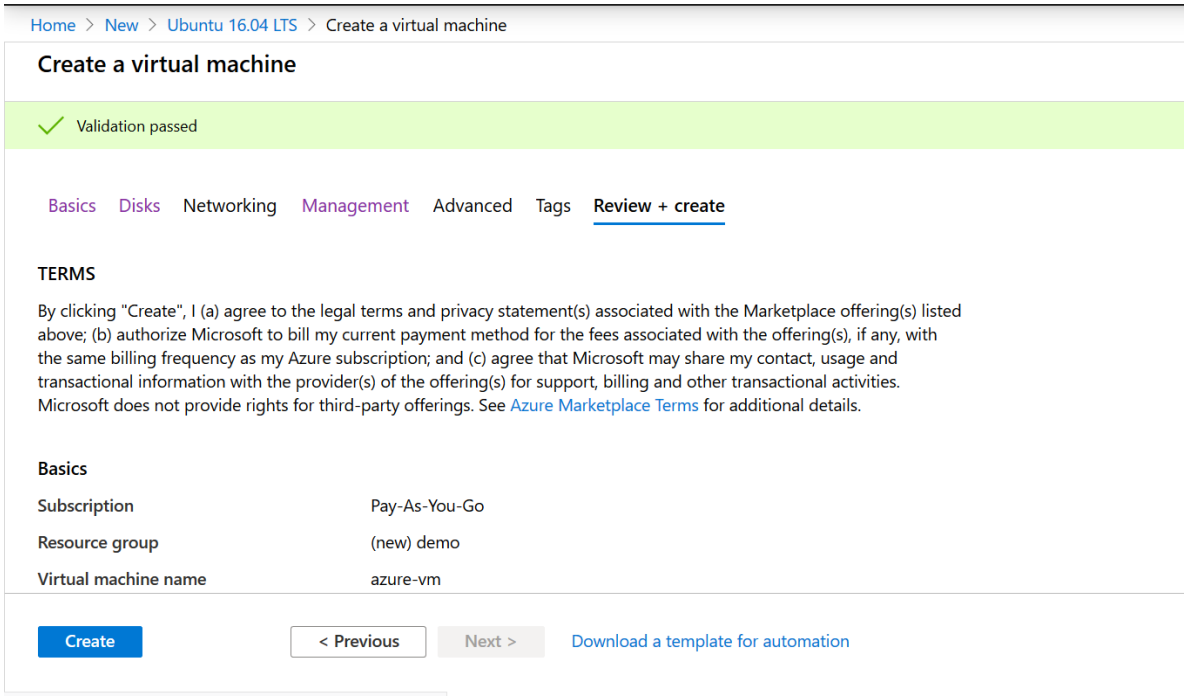
Add an inbound rule where any IP can access the VM.



Leave the remaining to defaults setting and click **Review + create**.

Step 5 Review + create

In the review, it will take some time to validate all the configuration. On the pass, you can see **Validation Passed** message.



Click **Create**. It will take some time to create a virtual machine.

Step 6 Connect to the VM

Go to the VM overview and click **Connect** and select **SSH**.

Open the bash or any other command line.

```
ssh -i <private key path> azuredemo@52.172.178.61

ssh -i ~/.ssh/id_rsa azuredemo@52.172.178.61
```

Enter passphrase.

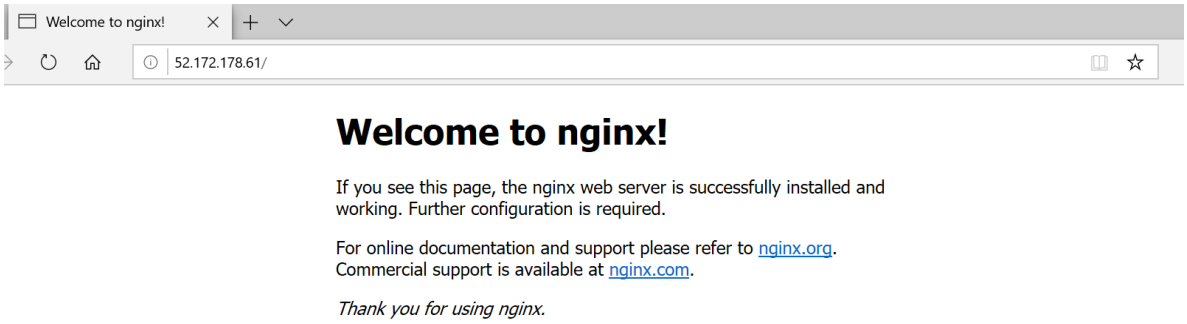
You're connected to the VM.

Step 7 Test it.

Install `nginx` web server in it.

```
sudo apt-get -y update
sudo apt-get -y install nginx
```

It will take a few seconds to install it. Once it install.
Open the browser and enter the VM IP address.



Step 8 Stop the VM

You can save some money by stopping the VM when you're not using it. Go to VM overview and stop it.

Next time when you'll start it only the public IP of the VM will change.

*****

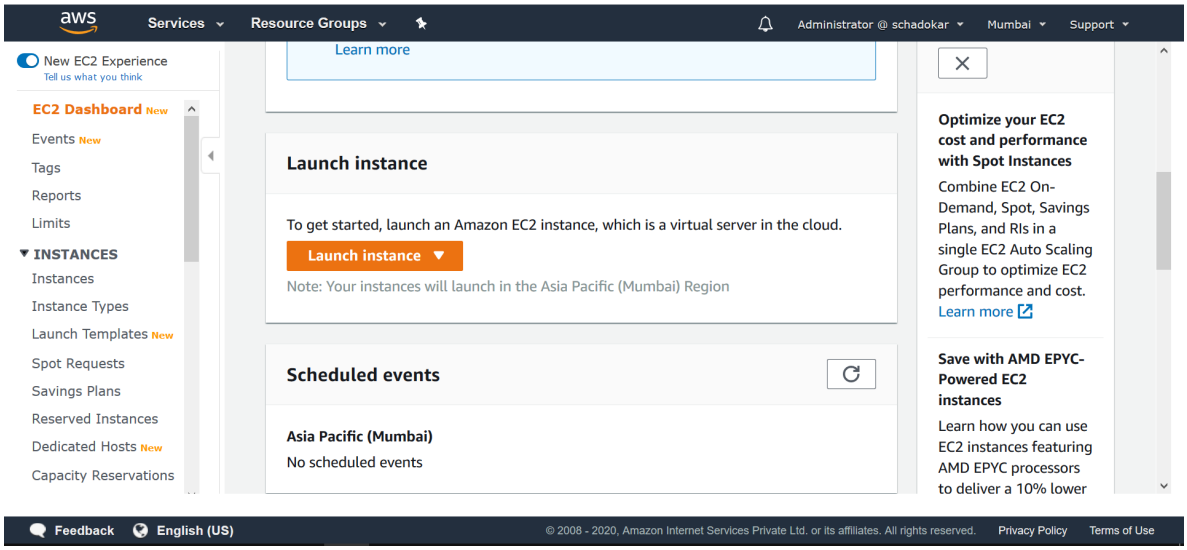
Create a Linux VM in AWS

Create a new aws account.

Login to your AWS account.
Select Services on the top left and search **EC2** and select it.

EC2 Dashboard

Scroll down to ***Launch Instance*** section and select **Launch Instance**.

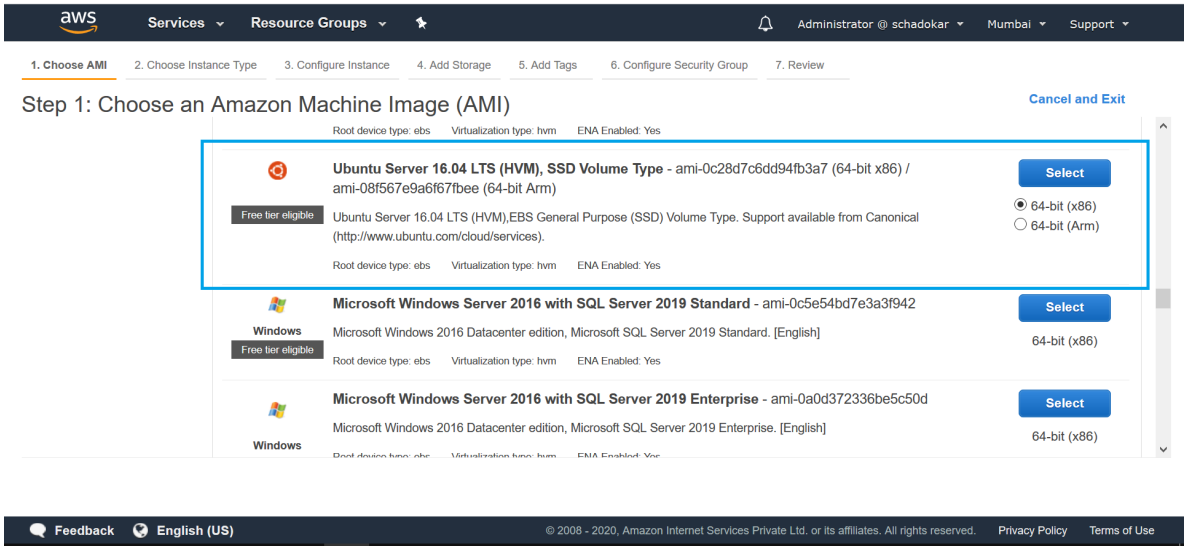


Configure Instance

Follow the steps to create a Ubuntu 16.04 VM.

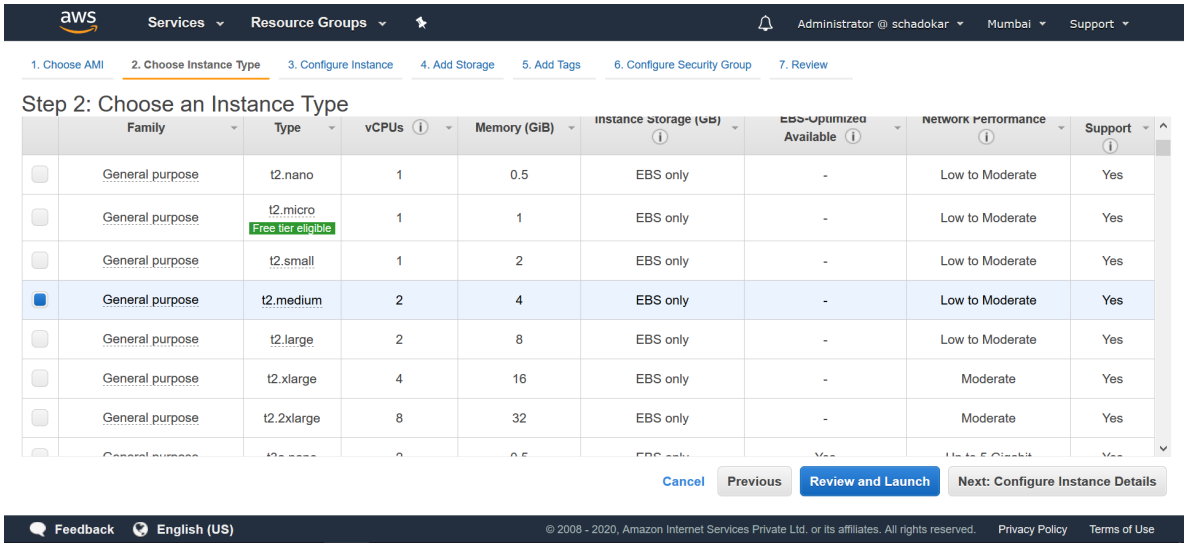
Step 1 Choose an Amazon Machine Image (AMI)

Scroll down and select **Ubuntu Server 16.04 LTS (HVM)**.



Step 2 Choose an Instance Type

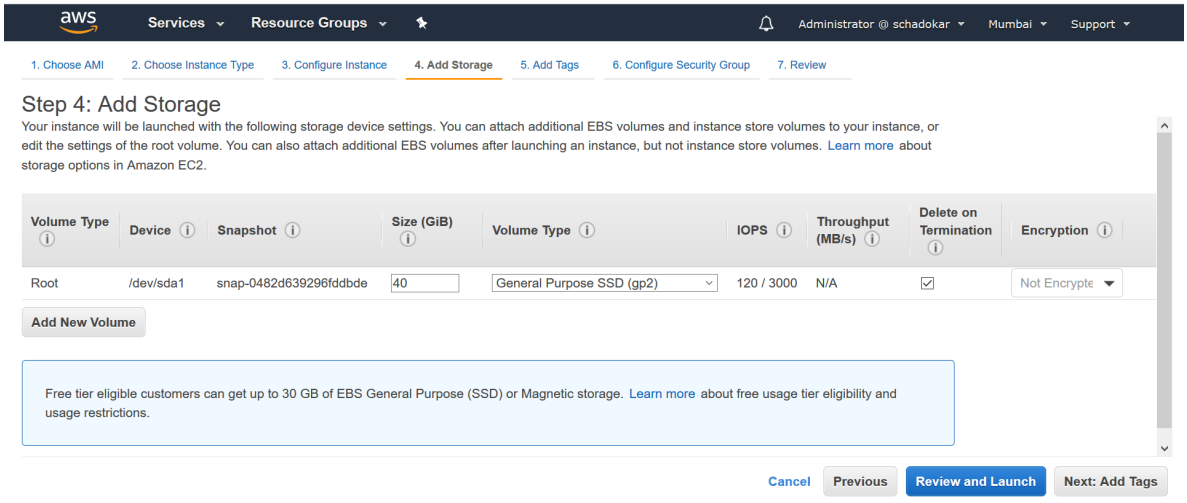
Select **t2.medium** and click **Configure Instance Details**.



Select next step till **Add Storage**.

Step 3 Add Storage

Change the storage to 40GB.

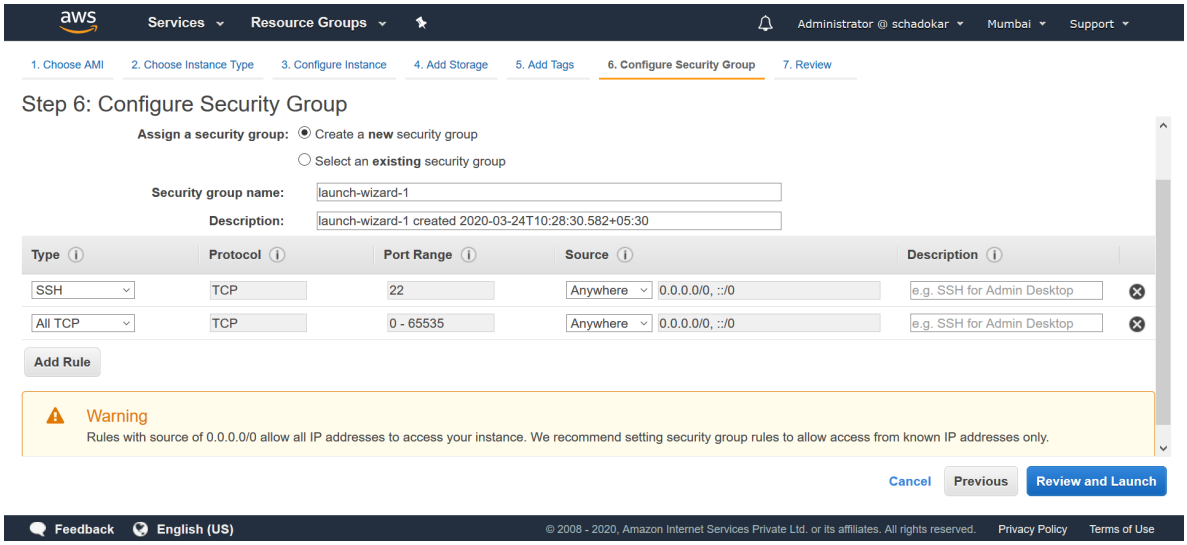


Select next step till **Configure Security Group**.

Step 4 Configure Security Group

Security group allow you to restrict the access. As for the learning, keep it simple and open it for everyone.

Add 2 rules, **SSH** and **All TCP**. Change source to **Anywhere**.



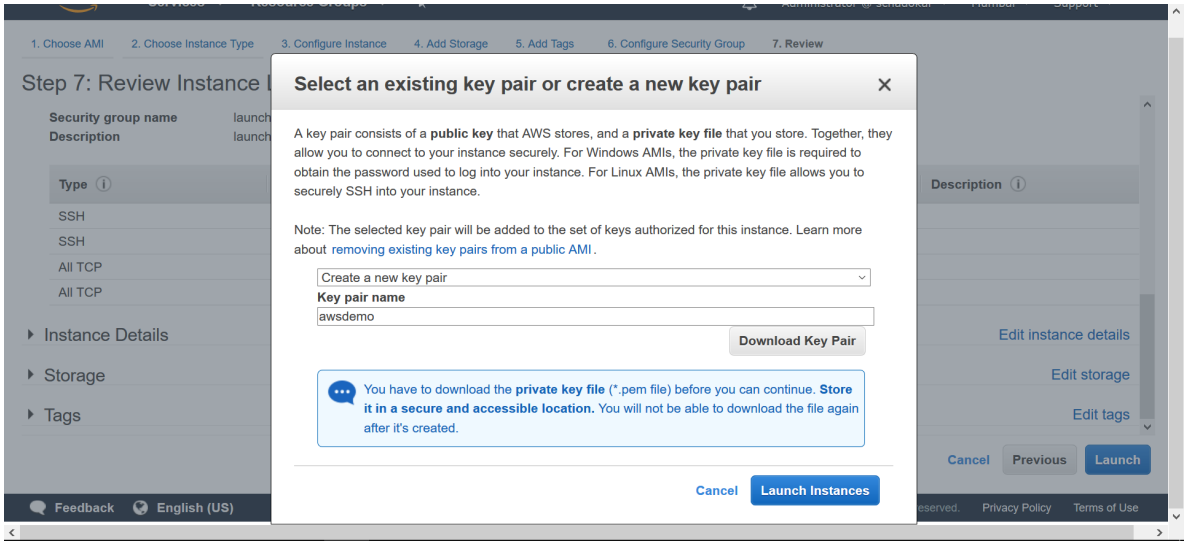
Select **Review and Launch**.

Step 5 Launch

Scroll down and select **Launch**.

Create a new key-pair. It is a public-private key pair. AWS stores the **public key** while **private key** is saved by you. While connecting to the instance you need this private key to connect.

Give a Keypair a name.

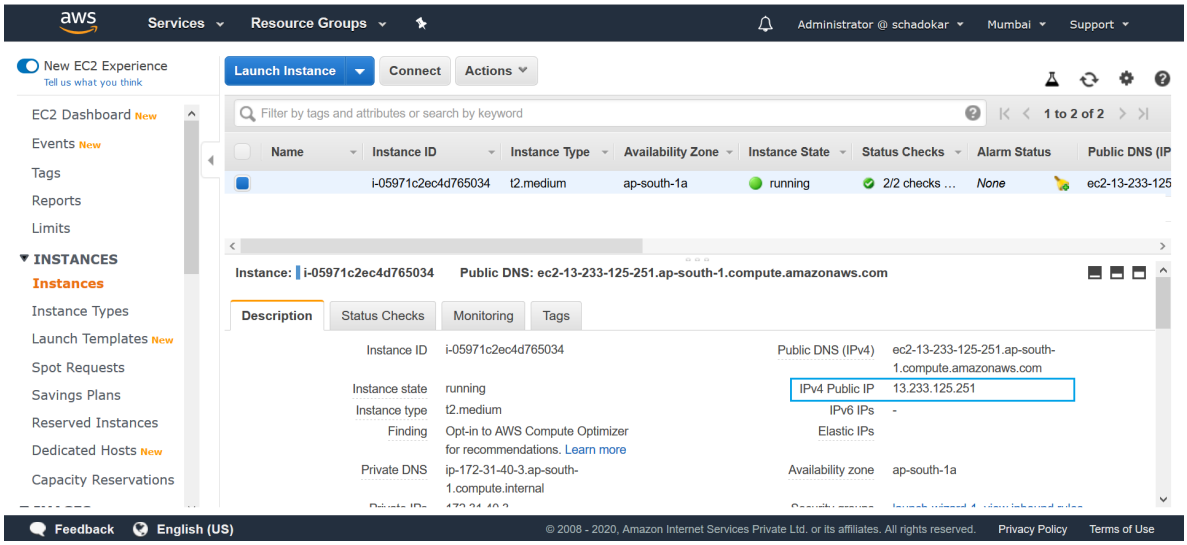


Select **Launch Instance**. It will take some time to launch the instance.

To check the instance, either you can scroll down and select **View Instance** or go to **EC2 Dashboard** and select **Instance** from the left menu.

Connect to the Instance

Once the instance is ready. Its **Instance State** is changed to **running**.
Copy the **IPv4 Public IP**.



Open the command line.

You can use any terminal such as `cmd`, `bash`, `terminal` etc.
Open the `terminal` and go to the directory where you saved the private key.

Use the `ssh` command to connect to the VM.

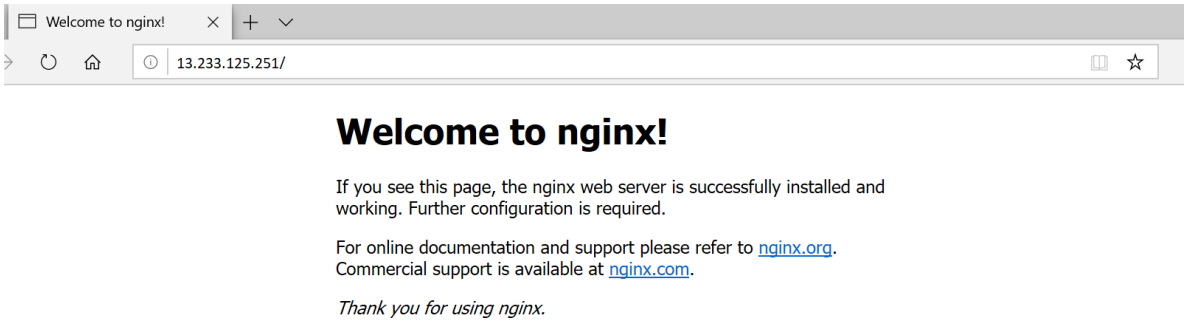
```
ssh -i path/privatekey.pem ubuntu@ipaddress

ssh -i awsdemo.pem ubuntu@13.233.125.251
```

Test the VM

Install `nginx` web server in it.

```
sudo apt-get -y update
sudo apt-get -y install nginx
```

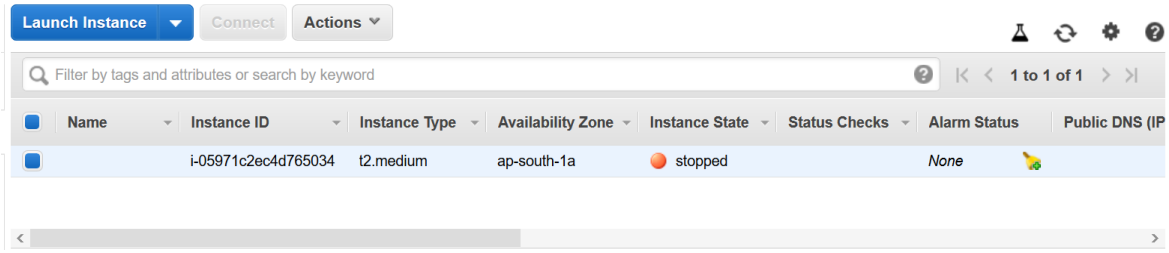
It will take a few seconds to install it. Once it install.
Open the browser and enter the VM IP address.



Stop the VM

You can stop the VM when you're not using it. When you restart it, that time the IP address will change.

> Go to Actions > Instance State > Stop.



*****

Create a Linux VM in Google Cloud Platform (GCP)

Create a new account on GCP. For new users, GCP credits $300 dollars in the account for 12 months and with many free services.

Login to your GCP account.

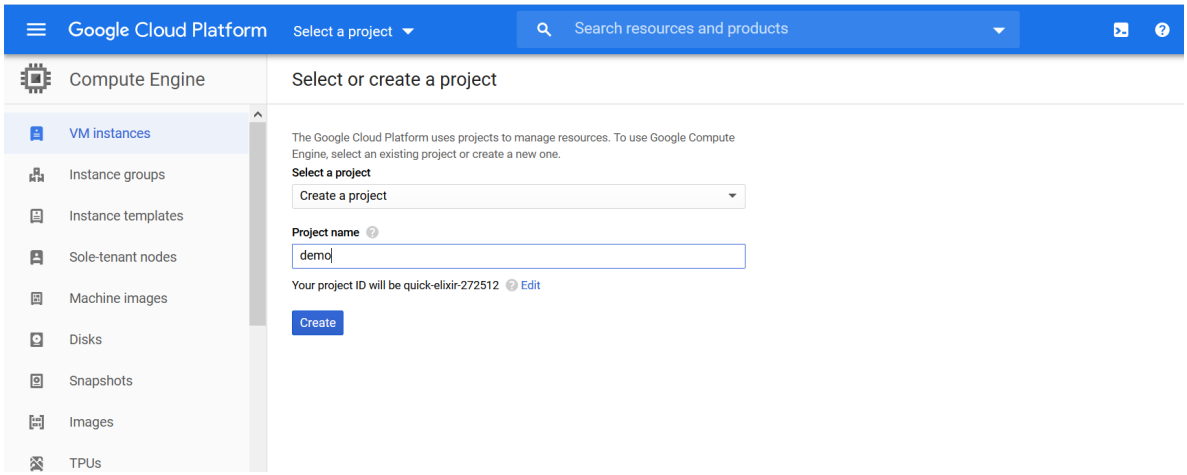
Create a new virtual machine

Select **Compute Engine** from the top left menu.

Menu > Compute > Compute Engine

Create a new project

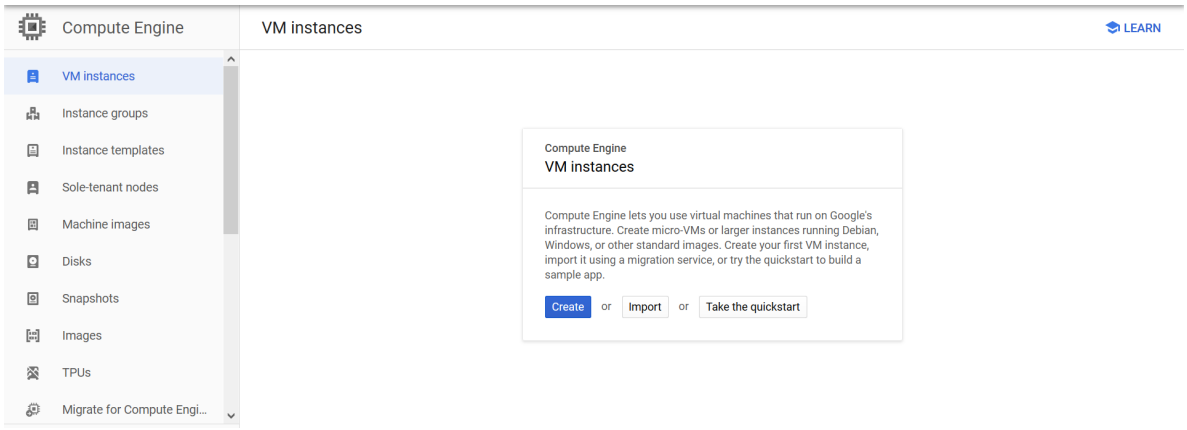
In GCP, a project is a resource manager. All the related resources of a project place inside it. Create a new project **demo**. It will take some time to create the project.



Create VM instance

Click on **Create**.

> **Take the quickstart** it is a small tutorial on how to create the VM.

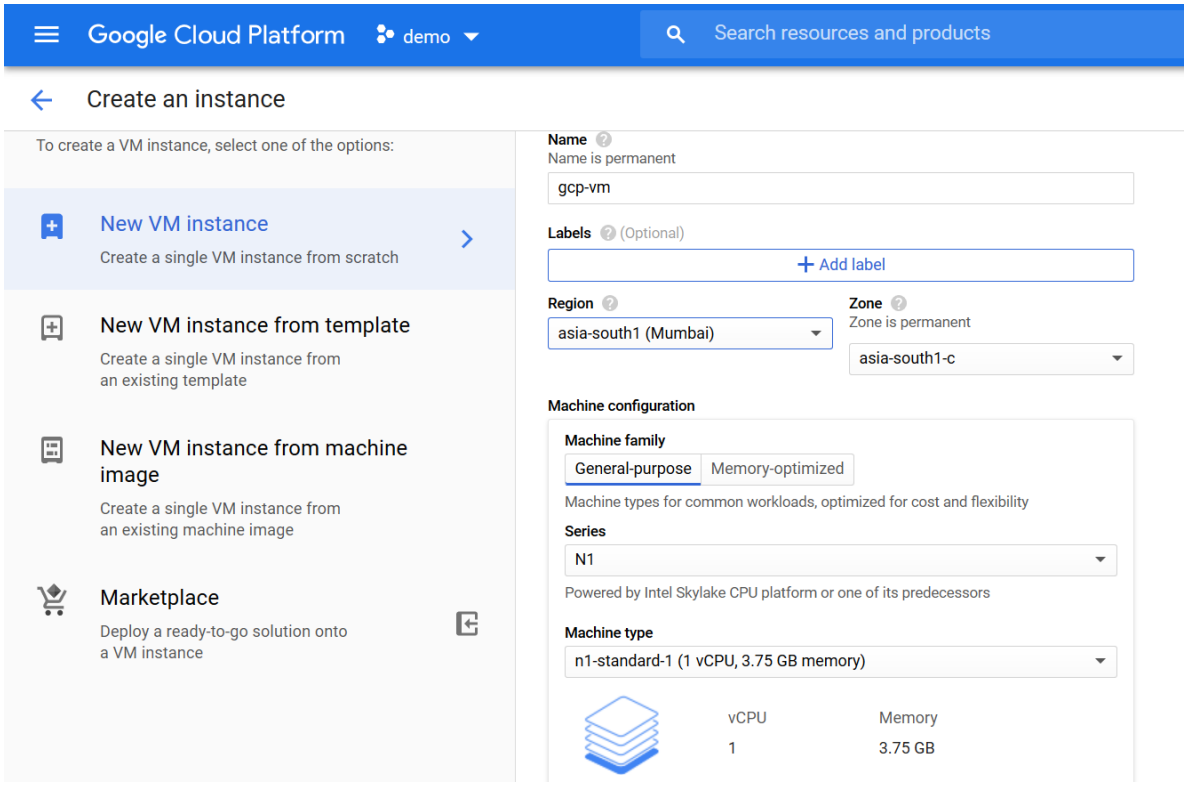


Basic Details

Enter the virtual machine name. `gcp-vm`.

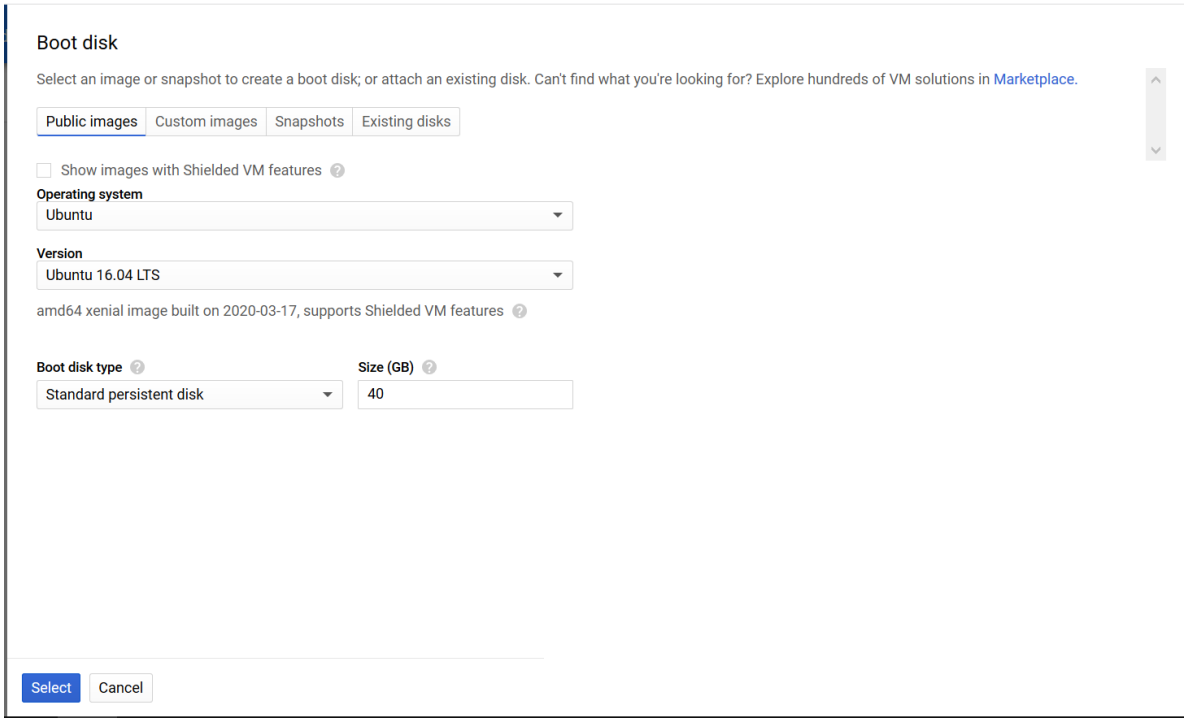
Select the nearest region.

Select the Series **N1** and machine type **n1-standard-1**.



Select the OS and Disk size

Change the OS Image to **Ubuntu 16.04 LTS**.
Increase the disk size to 40GB.



Firewall

Allow traffic from **http** and **https**.

> This step is optional. If you don't want to manage the private key then you can skip this step. If this is not setup then GCP manages the private key on your behalf. This private key is required to remotely connect to the VM.

Click on **security**.

Configure SSH to remotely connect to the VM using the command line.

Open the bash or any command line.

Use `ssh-keygen` command to create keys. By default, it saves these keys in `~/.ssh` directory.

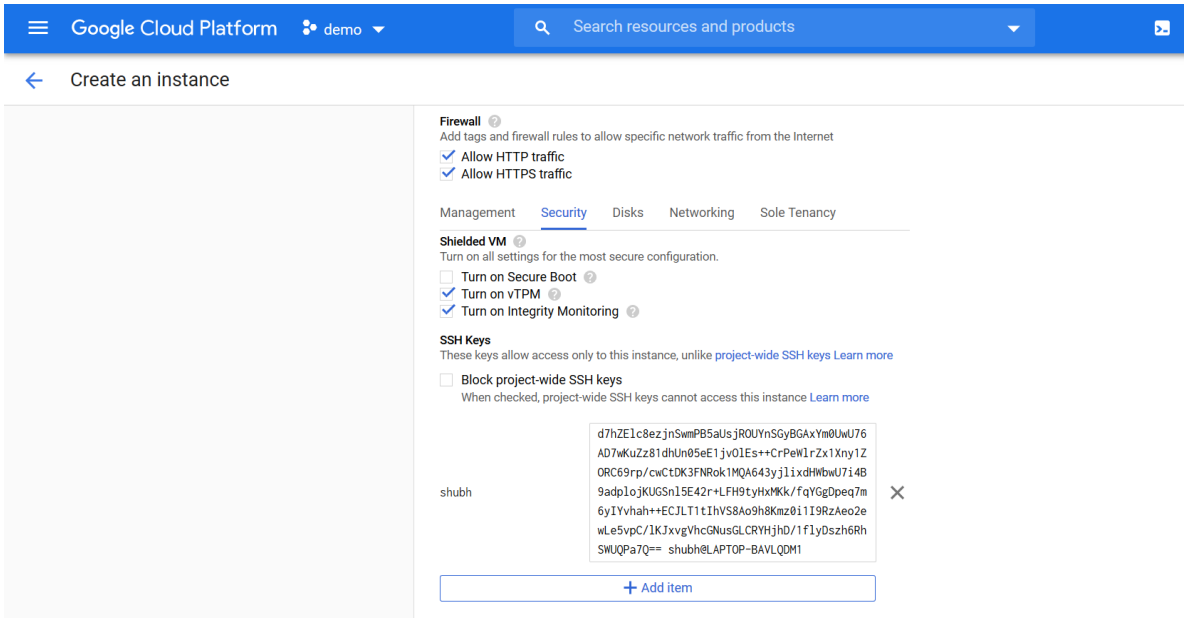
```
$ ssh-keygen -m PEM -t rsa -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/shubh/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/shubh/.ssh/id_rsa.
Your public key has been saved in /c/Users/shubh/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:5C+NIr1JjM8Oy5R6lgCyRm0EXqhHwToQQPYtN7f0qDY shubh@LAPTOP-BAVLQDM1
The key's randomart image is:
+---[RSA 4096]----+
|=*+o             |
|+.=..            |
|o+oo + o.        |
|*..oo +o+        |
|o=.    oS.       |
|...  =.  +       |
|.  .*E= o o      |
|   +=B.+ .       |
|  .oo.*          |
+----[SHA256]-----+
```

Enter the absolute path where you want to save the keys or skip it and press Enter.

The passphrase is required to access the private key while login in the VM. This step is optional. Passphrase length should be 5 or more. Note it down.

Open the `id_rsa.pub` file. This is the public key.
Copy the key and paste it in **Enter public SSH key**.

> Remove any leading or trailing spaces while pasting.

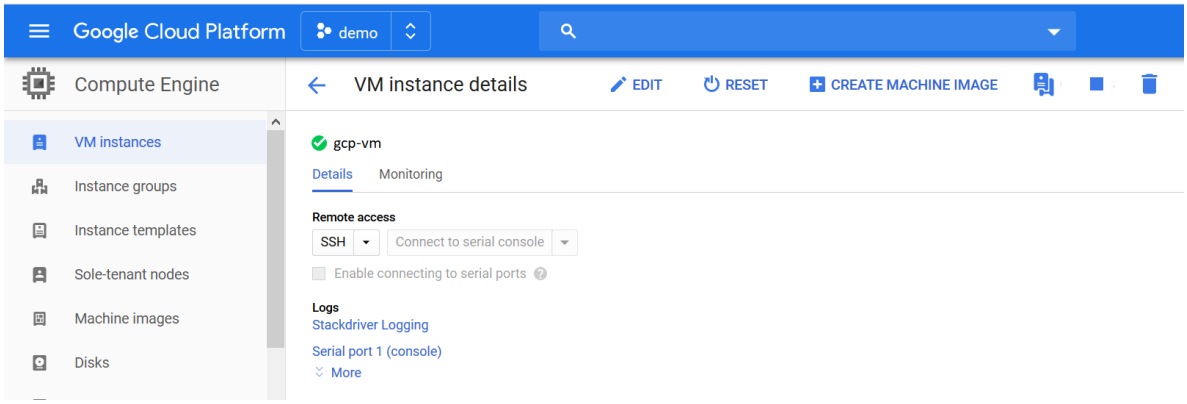


Click **Create**. It will take some time to create the VM.

Connect to the VM

Open the VM. It will show the VM details.

If you haven't configured the SSH then you can connect to the VM using the SSH provided by the GCP.
Click on **SSH**.



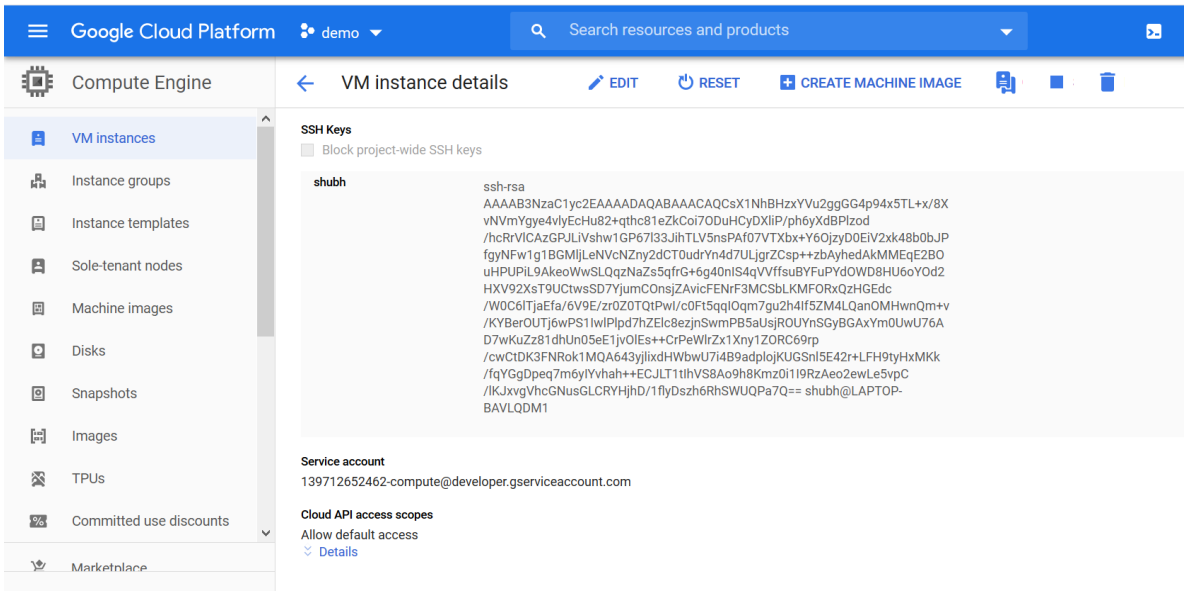
For SSH configured

Scroll down and copy the **External IP**.

Open the bash shell or any command line.

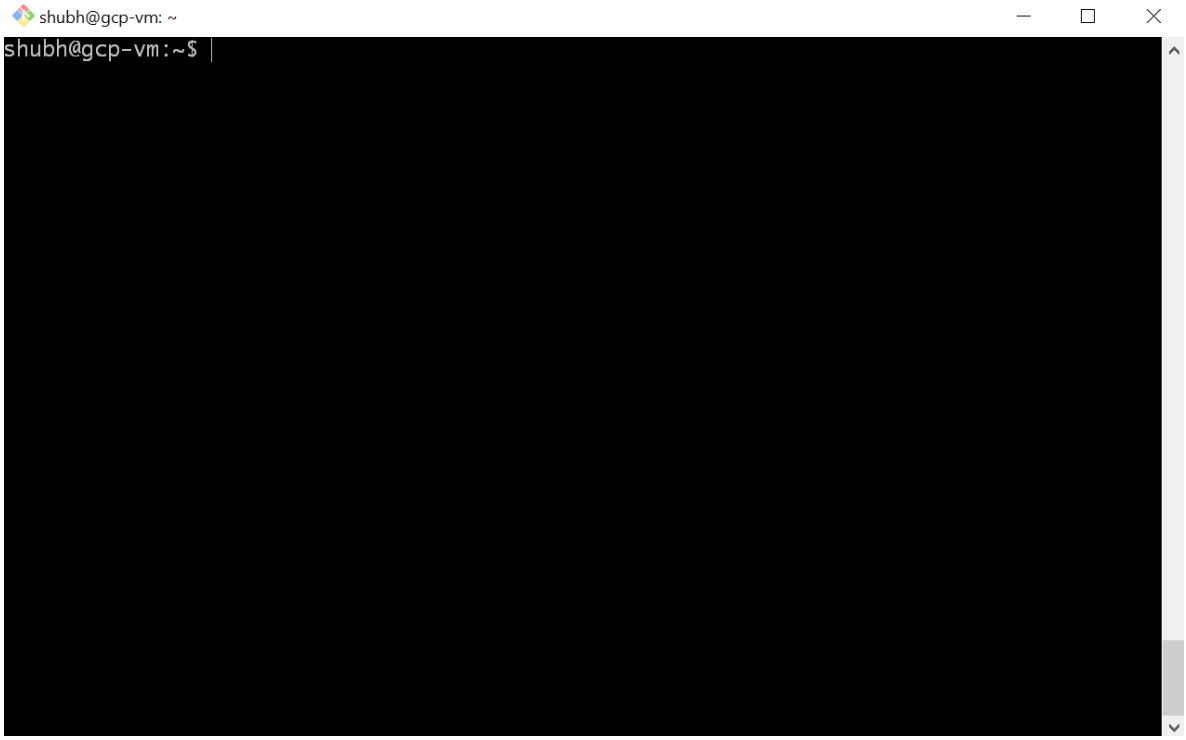
```
ssh -i <absolute-private-key-path> username@vmipaddress
```

Username is the key to the SSH.




```
ssh -i /c/Users/shubh/.ssh/id_rsa shubh@34.93.244.120
```

Enter the passphrase if you have set up it while creating the SSH keys.

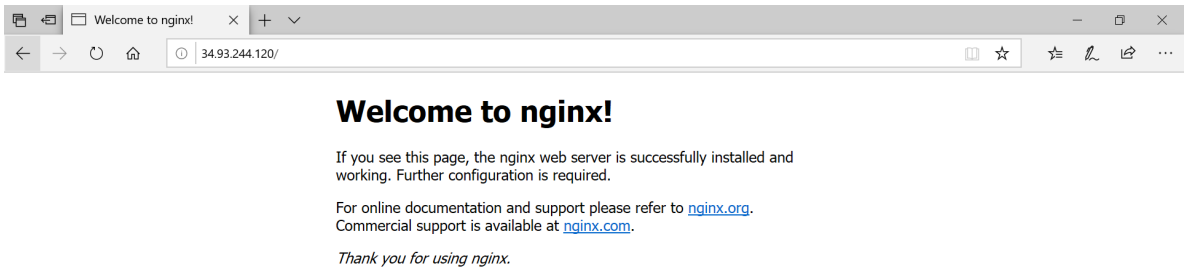


Test the VM

Install the **nginx** web server.

```
sudo apt-get -y update
sudo apt-get -y install nginx
```

It will take a few seconds to install it. Once it installs.
Open the browser and enter the VM IP address.

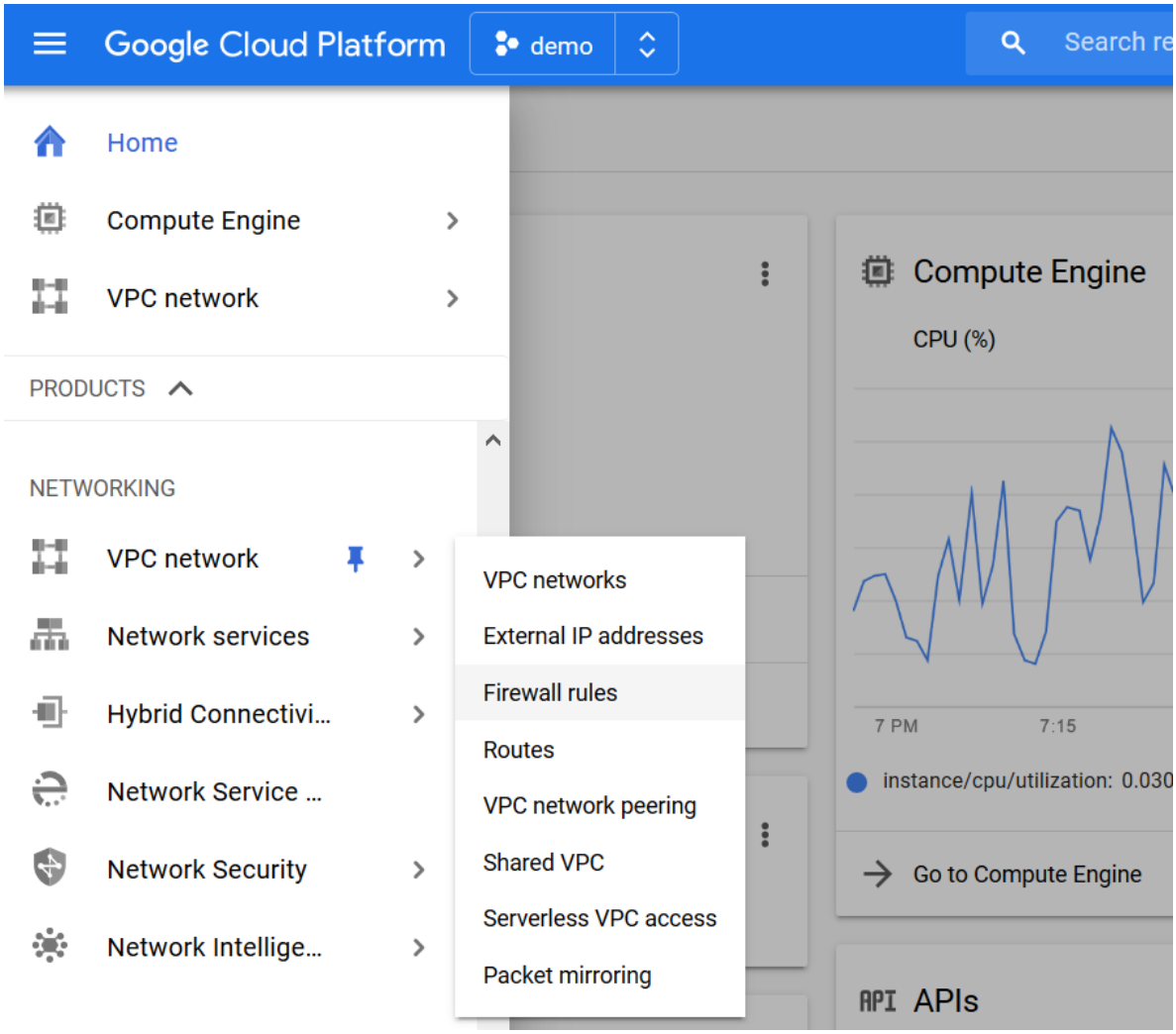


Allow Ports

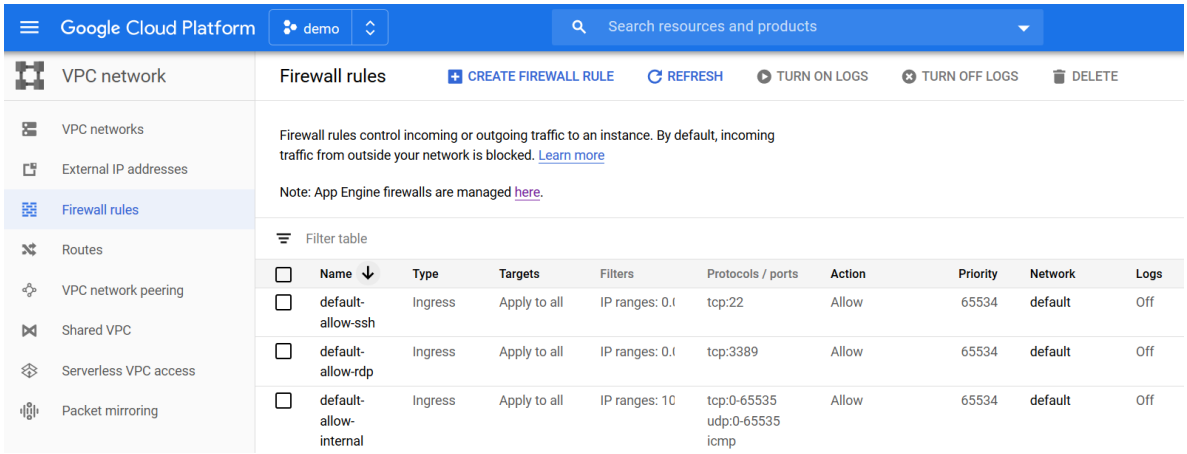
By default, all the requests to ports are blocked by the firewall. You have to explicitly create a firewall rule to allow various ports or allow all the ports.

Create a firewall rule to allow all the ports.

Open **Menu > VPC network > Firewall rules**.

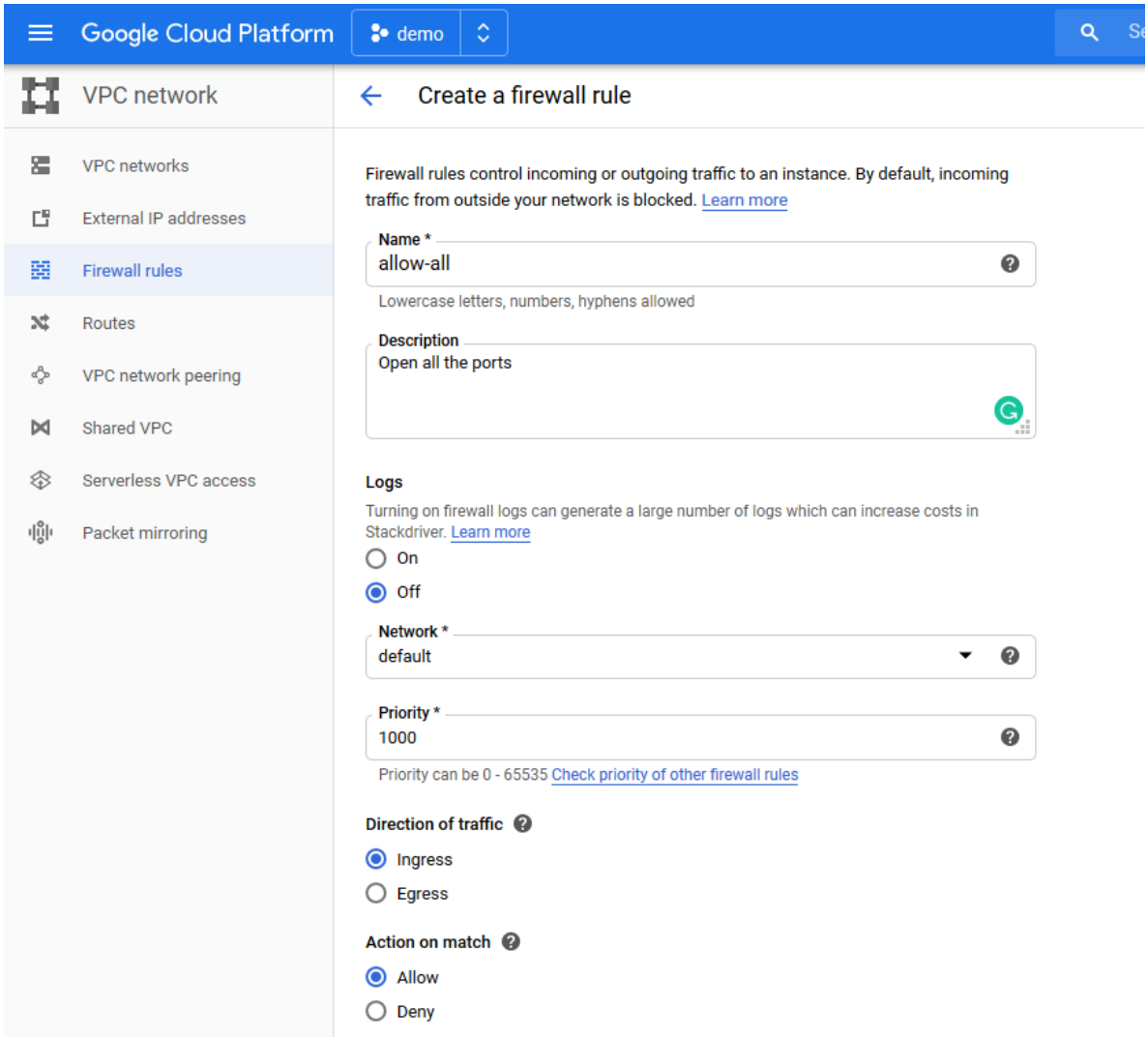


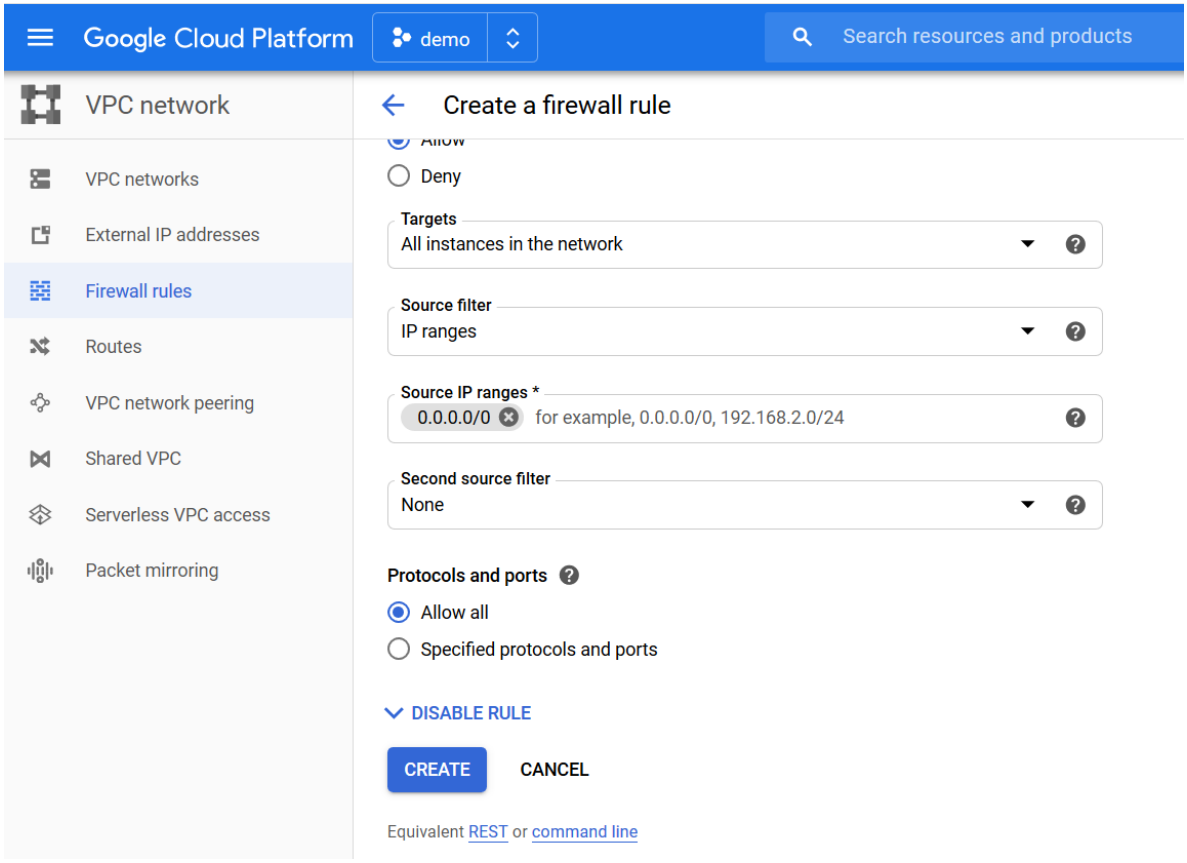
Click on **Create Firewall Rule** to create a new firewall.




```
Enter the name as allow-all
Select Targets as All instances in the network
Enter Source IP ranges as 0.0.0.0/0
Select Protocols and Ports as Allow all
```

Keep all the settings as default.



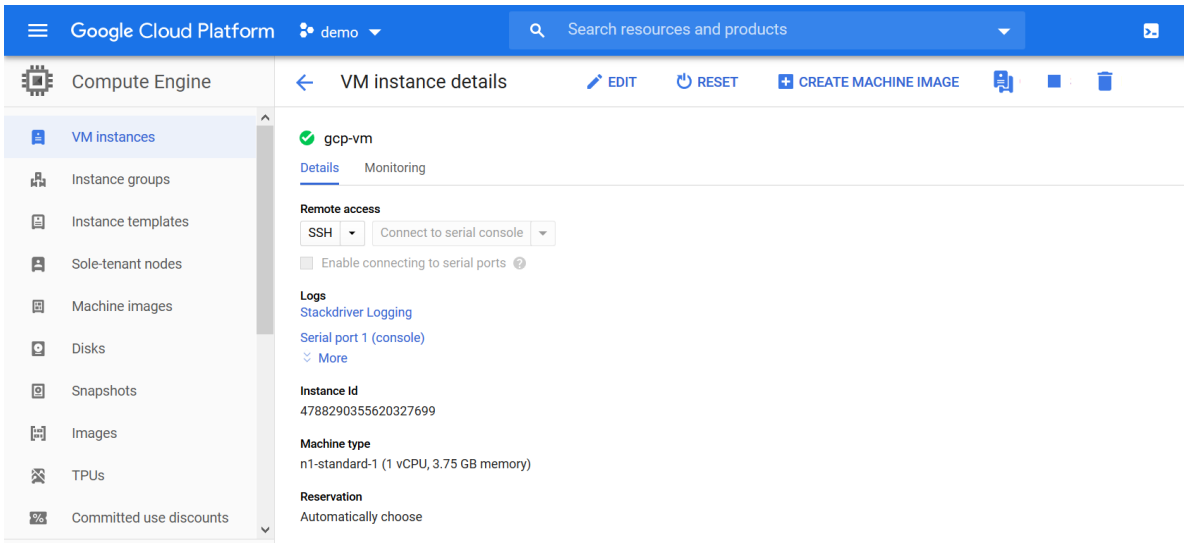


Click on **Create**.

Stop the VM

You can save some money by stopping or deleting the VM when you're not using it.

From the top, you can **Stop** or **Delete** the VM.



Next time when you'll start it only the public IP of the VM will change.

*****

Hyperledger Composer Installation Instruction

These instructions are applicable on UNIX based OS like Linux or MacOS.

Connect to your Linux VM. To use hyperledger composer, first, we have to install some pre-requisites.

Warnings for Linux Users

It is recommended that Hyperledger Fabric and Composer shouldn't be installed as a root user.

Instructions to create a new user

Open the command line or terminal.

Add a new user *composeruser*

You can have any name.

```
sudo adduser composeruser
```

Add the *composeruser* to the sudo groups

```
sudo usermod -aG sudo composeruser
```

Login to *composeruser*

```
su composeruser
```

Follow this link for more information.

Pre-requisites

Hyperledger Composer has provided a shell script which downloads and install all the dependencies.

```
curl -O https://hyperledger.github.io/composer/latest/prereqs-ubuntu.sh

chmod u+x prereqs-ubuntu.sh

./prereqs-ubuntu.sh
```

Install Composer Components

1. Essential CLI tools:

```
npm install -g composer-cli@0.20
```

2. Composer REST Server

The Hyperledger Composer REST Server can be used to generate a REST API from a deployed blockchain business network that can be easily consumed by HTTP or REST clients.

```
npm install -g composer-rest-server@0.20
```

3. Install Composer Playground

The Hyperledger Composer Playground provides a user interface for the configuration, deployment and testing of a business network.

```
npm install -g composer-playground@0.20
```

4. Install Hyperledger Fabric
Enterprise grade permissioned distributed ledger platform that offers modularity and versatility for a broad set of industry use cases.

```
mkdir ~/fabric-dev-servers && cd ~/fabric-dev-servers

curl -O https://raw.githubusercontent.com/hyperledger/composer-
tools/master/packages/fabric-dev-servers/fabric-dev-servers.tar.gz

tar -xvf fabric-dev-servers.tar.gz
```

Open the folder and download the **Hyperledger Fabric v1.2** images.

```
cd ~/fabric-dev-servers

export FABRIC_VERSION=hlfv12

./downloadFabric.sh
```

Start the Network

1. Clean the network if it was running before:

```
cd ~/fabric-dev-servers

export FABRIC_VERSION=hlfv12

./stopFabric.sh

./teardownFabric.sh

./downloadFabric.sh

./startFabric.sh
```

The `startFabric.sh` script will start a hyperledger fabric network. The network will start with a peer organisation, an orderer organisation and a certificate authority. It will also create one channel and joins the peer organisation to the channel.

2. Create Peer Admin Card
A Business Network Card provides all of the information needed to connect to a blockchain business network.

A peer admin business network card must be created in order to deploy a Hyperledger Composer business network to a Hyperledger Fabric instance.

```
./createPeerAdminCard.sh
```

3. Start Composer Playground

```
composer-playground
```

Open the composer playground in the browser.

Composer playground is started locally on the VM.

To access the playground on your web browser, instead of `http://localhost:8080`
Use `ipaddress_of_vm:8080`.

*****

Project Objective

We are building a supplychain management project.
Project's objective is to track the package while it is moving from one participant to another.

The is a simple project in which ownership of the package keeps changing from supplier to shipper and from shipper to buyer.

Project Structure

- **Participants**: There are 3 participants in the network. ***Seller***, ***Shipper***, and ***Buyer***.
- **Assets**: There is 1 asset as ***Package***.
- **Transactions**: There is only 1 transaction, which will transfer the ownership of the package from one participant to another.

Project Flow

When a package is created, the seller will be its owner and its status is ***Package_Ready***.
When shipper picks the package its status will change to ***Package_Picked***.
Once the package is delivered to buyer its status will change to ***Package_Delivered***.

*****

Project Setup in Composer

In this section, we will set up the composer environment for the project.

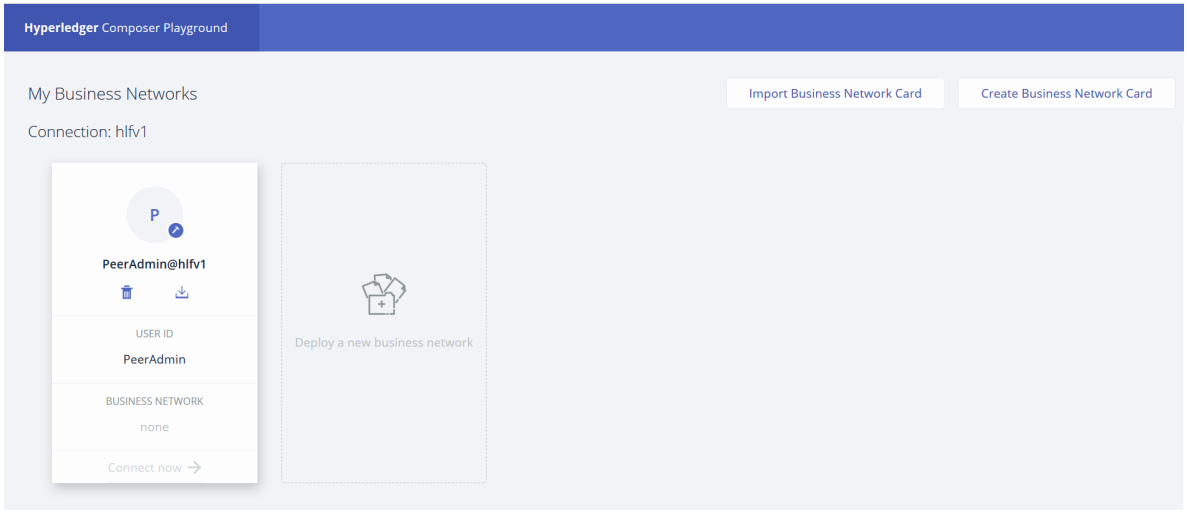
Open the terminal and start the composer playground.

```
composer-playground
```

This command will start the `composer-playground` on `localhost:8080`.

> If you're using your Linux machine then you can open the browser and search for `localhost:8080`.

Open the browser and search `<YOUR VM IP>:8080` to access the `composer-playground`.

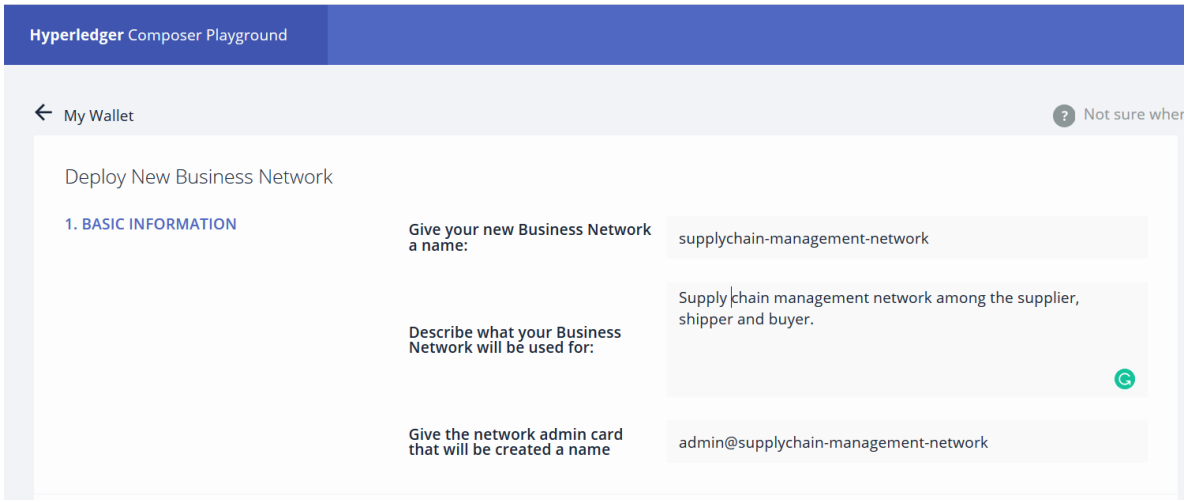


Click on **Deploy a new business network** to start a new business network.
Composer provides many network templates to start with. For this project, we are learning everything from scratch.
We'll start with the **empty business network**.

Enter the basic details of the network

Enter the business network name, description and network admin card.

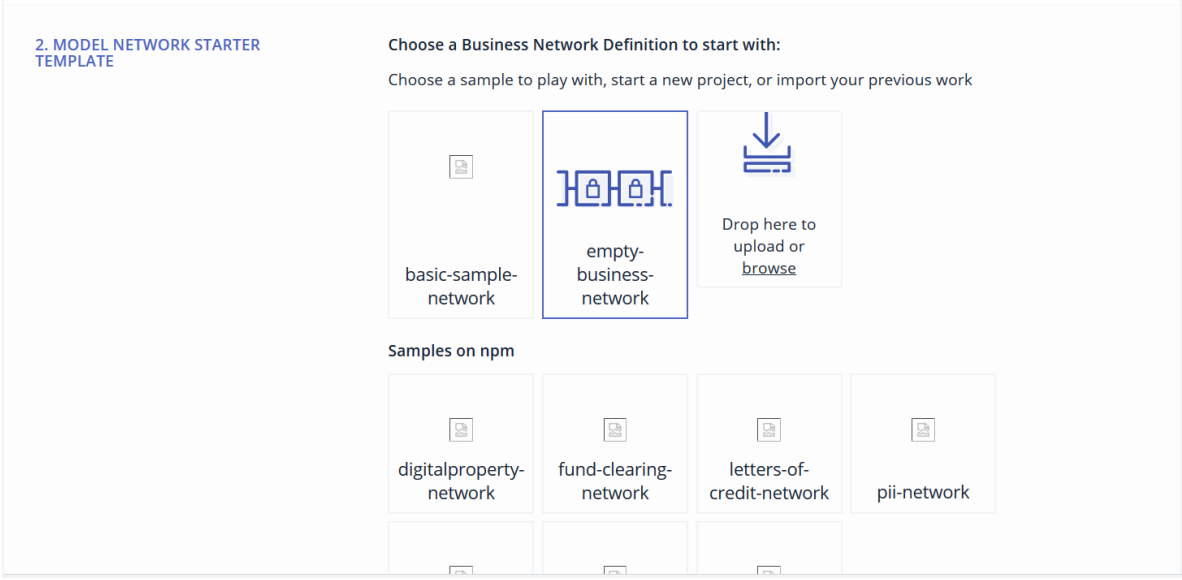
Using the network admin card you can interact with the network. It is like a key to the business network. If you don't enter any card name then by default it takes `admin@network-name`.



Select the empty business network

Select the empty business network. This business network has nothing in it except the default ACL and Model file.

> There are other business networks available with different real-world scenarios.

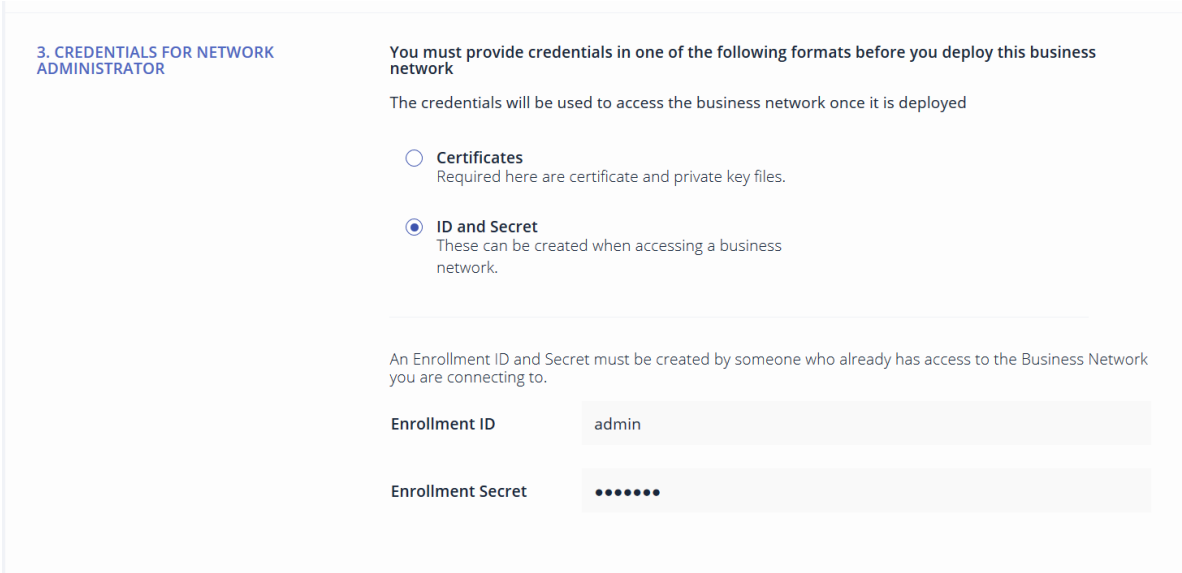


Enter the credential for network administrator

Enter the default admin id and secret. These secrets are required to deploy the business network on Hyperledger Fabric.

Select the **ID and Secret**.
Enter the default **Enrollment ID** and **Enrollment Secret**.

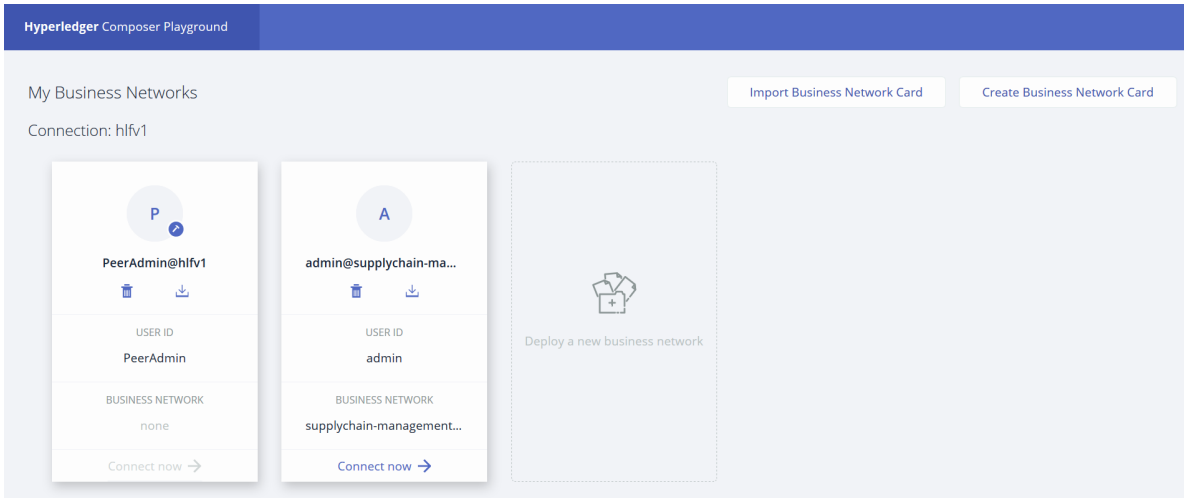
```
Enrollment ID: admin
Enrollment Secret: adminpw
```



Deploy the supplychain management business network

Click on **Deploy**. It will take some time to deploy the business network on the hyperledger fabric.

> **Note:**
> In the background, the business network translates into the hyperledger fabric compatible smart contract language. Later, this smart contract deploys on a channel. The smart contract deploys as a docker container. Once the deployment completes, open the terminal and enter `docker ps`. This command will return all the containers running in the hyperledger fabric.



Now, you can connect to the network and check the default files.

In the next sections, we'll develop supplychain management project.

*****

Model File

Hyperledger Composer includes an object-oriented modelling language that is used to define the business logic. The model file has a `.cto` extension.

Modeling Language

A Hyperledger Composer CTO file is composed of the following elements:

1. A single namespace. All resources related to single domain consist in a single namespace. In this way, different domains are isolated and can be included when needed.
2. asset, participant, transaction, event, enum and concept are resources.
3. Optional import declarations that import resources from other namespaces.

Let's understand the basics of the modelling language.

Data Types

These are Primitive Data Types

1. **String:** a UTF8 encoded string.
2. **Double:** a double-precision 64-bit numeric value.
3. **Integer:** a 32 bit signed the whole number.
4. **Long:** a 64 bit signed the whole number.
5. **DateTime:** an ISO-8601 compatible time instance, with optional time zone and UTZ offset.
6. **Boolean:** a Boolean value, either true or false.

Array

All the primitive types can be declared as an array.

Syntax

```
Integer[] integerArray
```

Asset

An asset is the objects on which different participants can do the transaction.

For Ex. Consider a car as an asset. Seller owns this car. Buyer can buy this car from the seller. Participant Buyer did a transaction with participant seller on asset car.

Syntax

```
asset Vehicle identified by vehicleid {
  o String vehicleid
  o Double price
  o String manufacturer
}
```

Participant

A participant is a member of the network. There can be multiple members of a network.

For Ex. In a supply chain network, there is a manufacturer, supplier, seller, and buyer. These all are a different participant with some role.

Syntax

```
participant User identified by userid {
  o String userid
  o String username
}
```

Transaction

The transaction is an action on asset or/and participant. Transactions are used to change the state of the asset or/and participant.

For Ex. Buying a car, changing the participant name.

Syntax

```
transaction buyVehicle {
  o String vehicleid
  --> User from
  --> User to
}
```

Relationships

Relationships are the reference for a particular resource. A relationship is represented using `-->`. This is unidirectional.

For Ex. A vehicle has an owner. To represent this relationship between a user and a vehicle.

```
asset Vehicle identified by vehicleid {
  o String vehicleid
  --> User owner
}
```

Event

Events are like notifications. If a particular thing happens then inform me.
Using events, on a particular scenario an event will publish and all the subscribers to this scenario will receive an event.

For Ex. Whenever a state of an asset changes publishes an event. In e-commerce whenever a user places an order, an event will trigger as **order places**. When it dispatch, an event will trigger as **order dispatch**. All these activities are events which can be published and subscribe.

Enum

Enum is used to define a category with **N** possible values. The benefit of using enum is to restrict the possible values.

For Ex. Days[Sunday, Monday, Tuesday,..., Saturday] Days is an enum with 7 possible values.

Syntax

```
/**
 * An enumerated type
 */
enum Vehicle {
  o Car
  o Bike
  o Truck
}

participant User identified by userid {
  o String userid
  o Vehicle vehicletype
}
```

Concept

Concepts are abstract classes that are not assets, participants or transactions. They are typically contained by an asset, participant or transaction.

Concepts are like static data fields among different assets, participants or transactions. Using concept there is no need to rewrite the same fields.

For Ex. below an abstract concept Address is defined, and then specialized into a UnitedStatesAddress. Note that concepts do not have an identified by field as they cannot be directly stored in registries or referenced in relationships.

```
abstract concept Address {
  o String street
  o String city default ="Winchester"
  o String country default = "UK"
  o Integer[] counts optional
}

concept UnitedStatesAddress extends Address {
  o String zipcode
}
```

You can then use this concept, for example

```
participant Farmer identified by farmerId {
    o String farmerId
    o UnitedStatesAddress address
    o ProductType primaryProduct
}
```

Optional

By default, all the fields in the asset and participant are mandatory. Using the `optional` keyword, the field can be optional.

Syntax

```
asset Vehicle identified by vehicleid {
  o String vehicleid
  o String manufacturer
  o String colour optional
}
```

Default

Any field in the asset or the participant can have an optional default field.

Syntax

```
asset Vehicle identified by vehicleid {
  o String vehicleid
  o String manufacturer
  o String color default="white"
}
```

Project Code

The code namespace is `org.supplychain.management`.
Package is an asset. Each asset has a reference of a member as an owner.
Member is a participant. Every member has a role.
Role is an enum. There are 3 roles for the members, **Seller**, **Buyer**, and **Shipper**.
Every package has 3 states, **Package_Ready**, **Package_Picked** and **Package_Delivered**. These states are represented as **Status**. **Status** is an enum.
transferOwnership is a transaction. It transfers the ownership of the package from member to member.

```
namespace org.supplychain.management

// role of the participant
enum Role {
  o Seller
  o Buyer
  o Shipper
}

// Status of the package
enum Status {
  o Package_Ready
  o Package_Picked
  o Package_Delivered
}

// package created by seller
asset Package identified by packageid {
  o String packageid
  o Status status
  o DateTime date
  --> Member owner
}

// different members in the network
// while creating define the role of the member
participant Member identified by memberid {
  o String memberid
  o String name
  o Role role
}

// transfering the ownership of the package
// from one member to another
transaction transferOwnership {
  o String packageid
  o Status status
  --> Member to
}
```

*****

Script File

In the script file, we define the functionality of the transactions defined in the model file.

The transaction definition using the javascript language.

The functions in the script file are known as ***Transaction Processor Function***. The bridge between the Transaction Processor Function and the transaction defined in the model file is known as ***Transaction Processor Structure***.

Transaction Processor Structure

The structure of transaction processor functions includes decorators and metadata followed by a JavaScript function, both parts are required for a transaction processor function to work.

All the transaction function must have a transaction processor structure defined. The transaction processor structure must be defined just above the transaction function.

After the comments are the JavaScript function which powers the transaction. The function can have any name but must include the parameter name defined in the comment as an argument.

Syntax

```
/**
 * A transaction processor function description
 * @param {namespace.transactionName} parameters A human description of the
parameters
 * @transaction
 */

async function transactionFunctionName(parameters) {}
```

Transaction Processor Function

Hyperledger Composer manages an individual registry for each resource in the blockchain network. To manage resources hyperledger composer has provided different registry APIs. In the transaction processor function using these APIs CRUD operations can be done on the resources.

There are 4 types of registry

Name	Description
AssetRegistry	The AssetRegistry is used to manage a set of assets stored on the Blockchain
ParticipantRegistry	The ParticipantRegistry is used to manage a set of participants stored on the blockchain.
TransactionRegistry	The TransactionRegistry is used to store a set of transactions on the blockchain.
IdentityRegistry	The IdentityRegistry is used to store a set of identities on the blockchain.

Following are the available methods in the registry

Name	Returns	Description
add	Promise	Adds a new resource to the registry
addAll	Promise	Adds a list of new resources to the registry
exists	Promise	Determines whether a specific resource exists in the registry
get	Promise	Get a specific resource in the registry
getAll	Promise	Get all of the resources in the registry
remove	Promise	Remove an asset with a given type and id from the registry
removeAll	Promise	Removes a list of resources from the registry
resolve	Promise	Get a specific resource in the registry, and resolve all of its relationships to other assets, participants, and transactions
resolveAll	Promise	Get all of the resources in the registry, and resolve all of their relationships to other assets, participants, and transactions
update	Promise	Updates a resource in the registry
updateAll	Promise	Updates a list of resources in the registry

Project Code

There is only one transaction processor function in the script file.
First, declare the ***Transaction Processor Structure*** then define the ***Transaction Processor Function***.

In the structure `@param {org.supplychain.management.transferOwnership`, `org.supplychain.management` is the namespace and `transferOwnership` is the transaction defined in the model file. `ownershipTx` is a parameter which is passed in the transaction function. The parameter is a Javascript Object. To access the arguments in the parameter using a `.` dot operator.

For ex. to access the `packageid` use, `ownershipTx.packageid`.

The Registry APIs are in asynchronous and returns promise. `async-await` is a method to handle the promise easily.

`TransferOwnershipf` is the processor function name which accepts the `ownershipTx` parameter.

There are 2 ways to pass the parameters, as a value or as a relationship.
In the `transferOwnership` transaction both are used for the demo.
The `packageid` is passed as value and `to` is passed as a relationship.

```
transaction transferOwnership {
  o String packageid
  o Status status
  --> Member to
}
```

To make any changes on the package with `packageid`, first resolve package relationship. `Package` is an asset and `packageid` is a string with no relationship to the `Package` asset.

Get the Package asset registry using `getAssetRegistry` API.

```
const packageRegistry = await getAssetRegistry(
  "org.supplychain.management.Package"
);
```

`org.supplychain.management` is the namespace and `Package` is the asset.

`packageRegistry` is the registry of all the Packages. To get a specific package use `get` method.

```
// Get the package from the package registry using packageid
const package = await packageRegistry.get(ownershipTx.packageid);
```

Now, update the `package` owner and `status`.
Let's first review the `Package` asset.

```
// package created by seller
asset Package identified by packageid {
  o String packageid
  o Status status
  o DateTime date
  --> Member owner
}
```

`owner` is declared as `relationship`. It requires a resolved relationship to update it.
In the transaction, `to` argument is a relationship. The relationship doesn't require to be resolved, composer resolves it.

> The arguments `packageid` and `to` are passed as value and relationship respectively. It depends on use-case and on your choice, how to pass the arguments.

The `package` variable is a `Package` asset object. To update the `owner` and `status` use `.` dot operator.

```
// transfer the ownership of the package
package.owner = ownershipTx.to;

// update the status
package.status = ownershipTx.status;
```

To reflect the changes in the `packageRegistry`, use the `update` method of the registry.

```
// update the package registry with the changes
await packageRegistry.update(package);
```

Now, the changes will reflect in the blockchain.

Let's club everything in a file.

```
/**
 * A transaction processor function description
 * @param {org.supplychain.management.transferOwnership} ownershipTx Transfering
the ownership of the package from one member to another
 * @transaction
 */

async function TransferOwnershipf(ownershipTx) {
  // get the package asset registry
  const packageRegistry = await getAssetRegistry(
    "org.supplychain.management.Package"
  );
  // Get the package from the package registry using packageid
  const package = await packageRegistry.get(ownershipTx.packageid);

  // transfer the ownership of the package
  package.owner = ownershipTx.to;

  // update the status
  package.status = ownershipTx.status;

  // update the package registry with the changes
  await packageRegistry.update(package);
}
```

*****

Query File

Hyperledger Composer has provided a simple and powerful query language to query the data stored in the Hyperledger Fabric blockchain. A query file has a `.qry` extension.

Query Language

The native query language is capable of doing a simple query using `SELECT` and `WHERE` statement.
A query must have a description and a statement.

Syntax

```
query QueryName {
  description: "Description of the Query"
  statement: SELECT Query Statement
}
```

description

The `description` property is the definition of the query.

statement

The `statement` property contains the query.

- `SELECT`: is an operator to get the resource an asset or a participant. The resource name is given as `namespace.resource`.
- `FROM` is an optional operator which defines a different registry to query.
- `WHERE` is an optional operator which defines the conditions to be applied to the registry data.
- `AND` is an optional operator which defines additional conditions.
- `OR` is an optional operator which defines alternative conditions.
- `CONTAINS` is an optional operator that defines conditions for array values
- `ORDER BY` is an optional operator which defines the sorting or results. For ascending `ASC` and `DESC` for descending.
- `_$variable` using `_$` sign, variables can be passed in the query.

Example

```
query EmployeeQuery {
  description: "Employee Details"
  statement:
    SELECT org.mnc.xyz.Employee
      WHERE ((age < 35 OR occupation == 'Blockchain Developer') AND (experience
>= _$experience))
        ORDER BY [name ASC]
}
```

Project Code

There will be 3 queries in the project.

1. Query all the members by their role.
2. Query all the packages by their status.
3. Query all the packages by their owner.

```
query QueryMembersByRole{
  description: "Get all the members by their roles. Seller, Buyer and Shipper."
    statement:
      SELECT org.supplychain.management.Member
        WHERE (role == _$role)
          ORDER BY [name ASC]
}

query QueryPackageByStatus {
  description: "Get all the packages by their current status. Package_Ready,
Package_Picked and Package_Delivered."
    statement:
      SELECT org.supplychain.management.Package
        WHERE (status == _$status)
}

query QueryPackageByOwner {
  description: "Get all the packages by their current owner."
    statement:
      SELECT org.supplychain.management.Package
        WHERE (owner == _$owner)
}
```

*****

ACL File

Hyperledger Composer has provided an access control language using which rules and permissions can be defined on the resources described in the Model File. By defining the ACL rules for the users, the access can be restricted on the resources. Like, Create, Read, Update and Delete operations. For ex. Create operations can be restricted to one user and only read access is provided to all users in the network.

The extension of the access control file is `.acl`.

> There are 2 types of access control, first access control on resources and second on the Network. These are out of the scope for this eBook. To learn more about them please check out the references.

Syntax

```
rule RuleName {
    description: "Description of the ACL rule"
    participant(m): "org.example.SampleParticipant"
    operation: READ, CREATE, UPDATE, DELETE or ALL
    resource(v): "org.example.SampleAsset"
    transaction(tx): "org.example.SampleTransaction"
    condition: (v.owner.getIdentifier() == m.getIdentifier())
    action: ALLOW
}
```

Project Code

In the access control file, there are default network rules defined. You don't have to add anything.

```
rule NetworkAdminUser {
    description: "Grant business network administrators full access to user
resources"
    participant: "org.hyperledger.composer.system.NetworkAdmin"
    operation: ALL
    resource: "**"
    action: ALLOW
}

rule NetworkAdminSystem {
    description: "Grant business network administrators full access to system
resources"
    participant: "org.hyperledger.composer.system.NetworkAdmin"
    operation: ALL
    resource: "org.hyperledger.composer.system.**"
    action: ALLOW
}
```

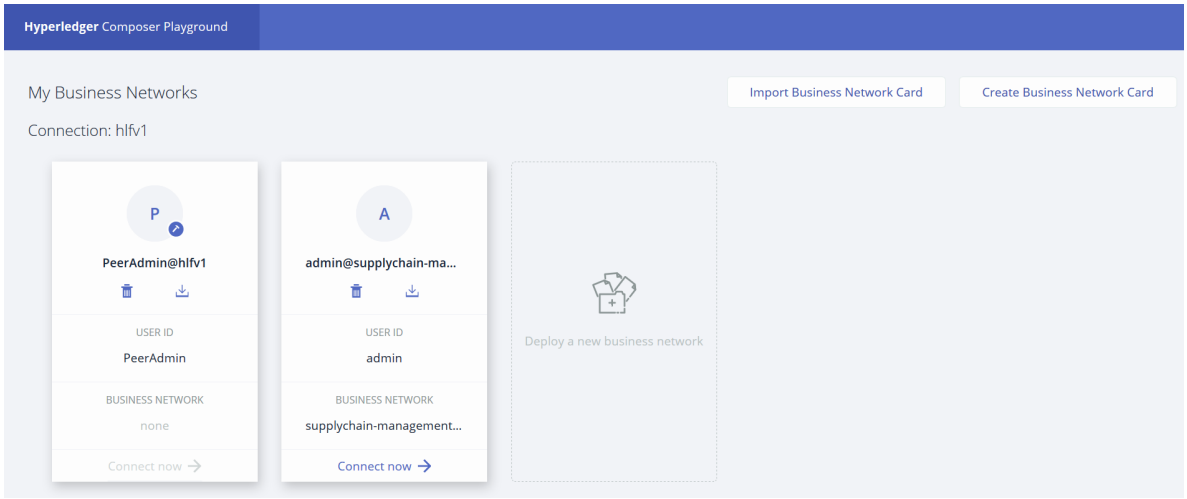
*****

Deployment in Composer Playground

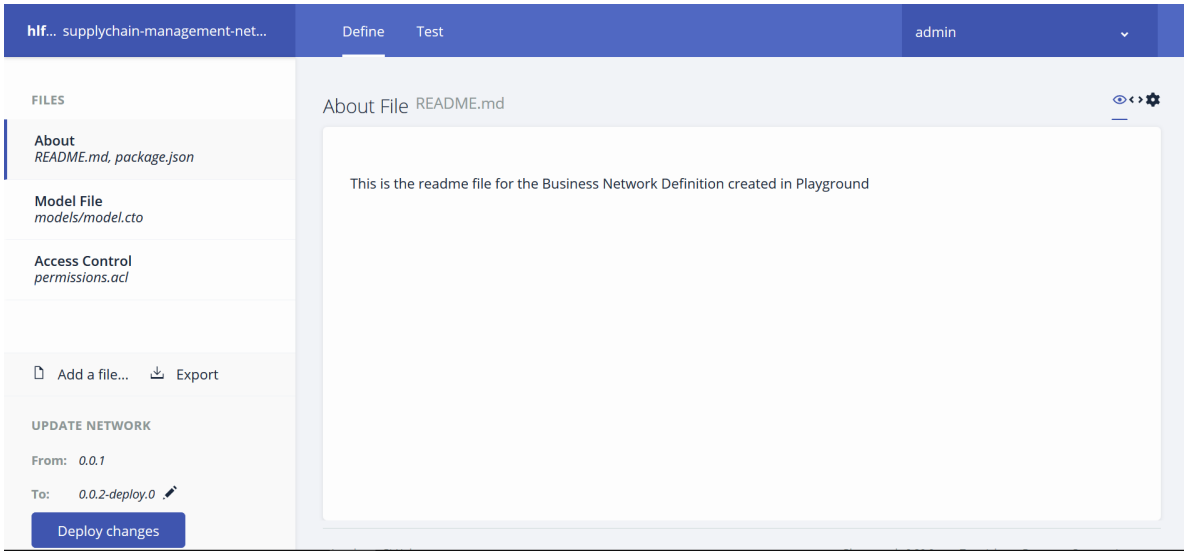
In this section, we'll deploy the supplychain management project.
We'll integrate the business network logic.

Open the composer playground in the browser.

We already set up the empty supplychain business network.



Connect to the `supplychain-management` business network.



About File

This file is a `README.md` file where you can enter all the detailed description of the network.
To edit the `readme` click on `<>` in the top right corner.

Click on settings ⚙ to check the project settings.
You can edit the configuration. Only edit author and description if you really want to.

> ⚠ **Do not edit project settings**
> The dependencies are the libraries with specific supported version to the hyperledger fabric and hyperledger composer. Changing them can cause trouble.


```
{
  "name": "supplychain-management-network",
  "author": "author",
  "description": "Supply chain management network among the supplier, shipper and buyer.",
  "version": "0.0.2-deploy.0",
  "devDependencies": {
    "browserfs": "^1.2.0",
    "chai": "^3.5.0",
    "composer-admin": "latest",
    "composer-cli": "latest",
    "composer-client": "latest",
    "composer-connector-embedded": "latest",
    "eslint": "^3.6.1",
    "istanbul": "^0.4.5",
    "jsdoc": "^3.4.1",
    "mkdirp": "^0.5.1",
    "mocha": "^3.2.0",
    "moment": "^2.19.3"
  },
  "keywords": [],
  "license": "Apache 2.0",
  "repository": {
    "type": "e.g. git",
    "url": "URL"
  },
  "scripts": {
    "deploy": "./scripts/deploy.sh",
    "doc": "jsdoc --pedantic --recurse -c jsdoc.conf",
    "lint": "eslint .",
    "postlicchk": "npm run doc",
    "postlint": "npm run licchk",
    "prepublish": "mkdirp ./dist && composer archive create  --sourceType dir --sourceName . -a ./dist/unnamed-network.bna",
    "pretest": "npm run lint",
    "test": "mocha --recursive",
    "start": "start-network"
  },
  "dependencies": {
    "composer-common": "0.20.9",
    "composer-runtime-hlfv1": "0.20.9"
  }
}
```

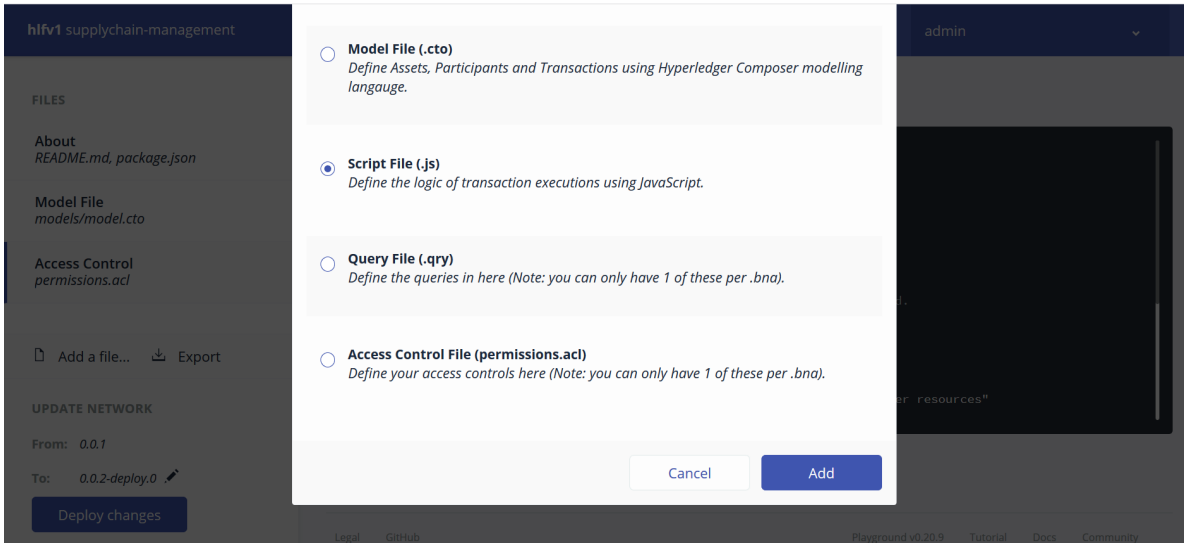
Model File

Open the Model file `.cto`.
Copy the below code and paste in it. This is the same code which we walkthrough in the **Model File** Section.

```
namespace org.supplychain.management

// role of the participant
enum Role {
  o Seller
  o Buyer
  o Shipper
}

// Status of the package
enum Status {
  o Package_Ready
  o Package_Picked
  o Package_Delivered
}

// package created by seller
asset Package identified by packageid {
  o String packageid
  o Status status
  o DateTime date
  --> Member owner
}

// different members in the network
// while creating define the role of the member
participant Member identified by memberid {
  o String memberid
  o String name
  o Role role
}

// transfering the ownership of the package
// from one member to another
transaction transferOwnership {
  o String packageid
  o Status status
  --> Member to
}
```

Script File

Click on **Add a file**.
Select **Script File(.js)**.

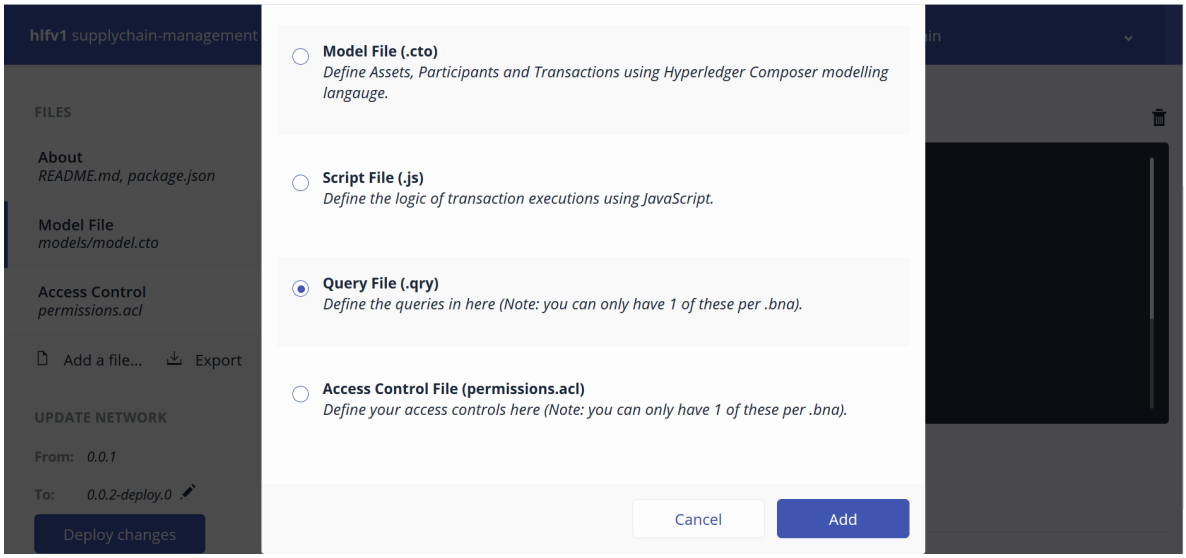


Copy the below code and paste in it.

```
/**
 * A transaction processor function description
 * @param {org.supplychain.management.transferOwnership} ownershipTx Transfering
the ownership of the package from one member to another
 * @transaction
 */

async function TransferOwnershipf(ownershipTx) {
  // get the package asset registry
  const packageRegistry = await getAssetRegistry(
    "org.supplychain.management.Package"
  );
  // Get the package from the package registry using packageid
  const package = await packageRegistry.get(ownershipTx.packageid);

  // transfer the ownership of the package
  package.owner = ownershipTx.to;

  // update the status
  package.status = ownershipTx.status;

  // update the package registry with the changes
  await packageRegistry.update(package);
}
```

Query File

Click on **Add a file**.
Select **Query File(.qry)**.



Open the **Query File**.
Copy and paste the below code in it.

```
query QueryMembersByRole{
  description: "Get all the members by their roles. Seller, Buyer and Shipper."
    statement:
      SELECT org.supplychain.management.Member
        WHERE (role == _$role)
          ORDER BY [name ASC]
}

query QueryPackageByStatus {
  description: "Get all the packages by their current status. Package_Ready,
Package_Picked and Package_Delivered."
    statement:
      SELECT org.supplychain.management.Package
        WHERE (status == _$status)
}

query QueryPackageByOwner {
  description: "Get all the packages by their current owner."
    statement:
      SELECT org.supplychain.management.Package
        WHERE (owner == _$owner)
}
```

ACL File

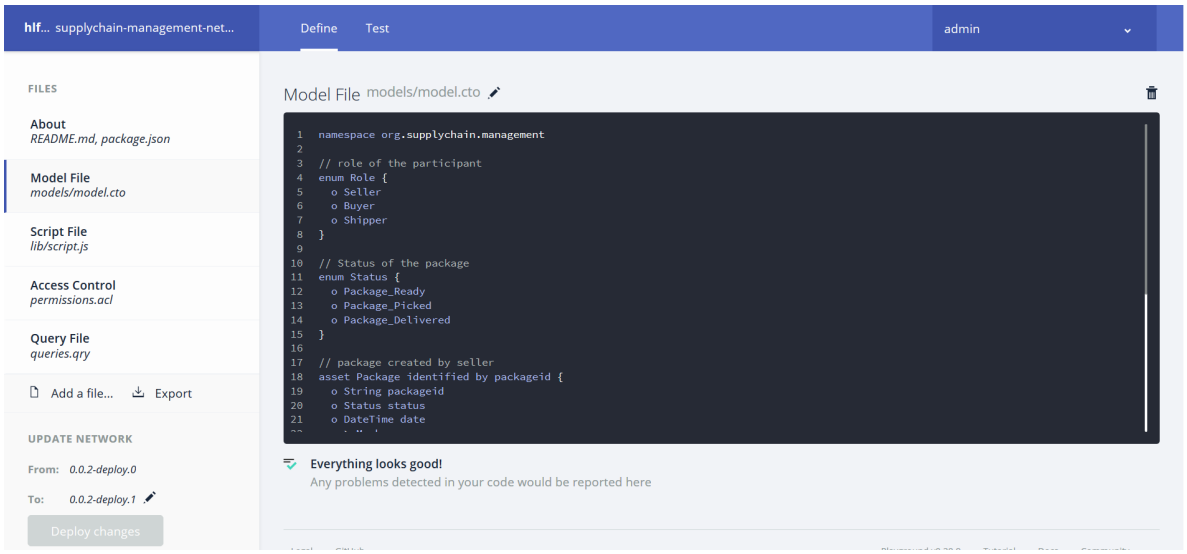
Open the **Access Control** File. Copy and paste the below code if it is empty.

```
rule NetworkAdminUser {
    description: "Grant business network administrators full access to user resources"
    participant: "org.hyperledger.composer.system.NetworkAdmin"
    operation: ALL
    resource: "**"
    action: ALLOW
}

rule NetworkAdminSystem {
    description: "Grant business network administrators full access to system resources"
    participant: "org.hyperledger.composer.system.NetworkAdmin"
    operation: ALL
    resource: "org.hyperledger.composer.system.**"
    action: ALLOW
}
```

Deploy

Click on **Deploy Changes**.
It will ask for confirmation to upgrade the network. Click on **Upgrade**.

It will deploy the business network on the Hyperledger Fabric blockchain as a Smart Contract.
It will take some time to instantiate.



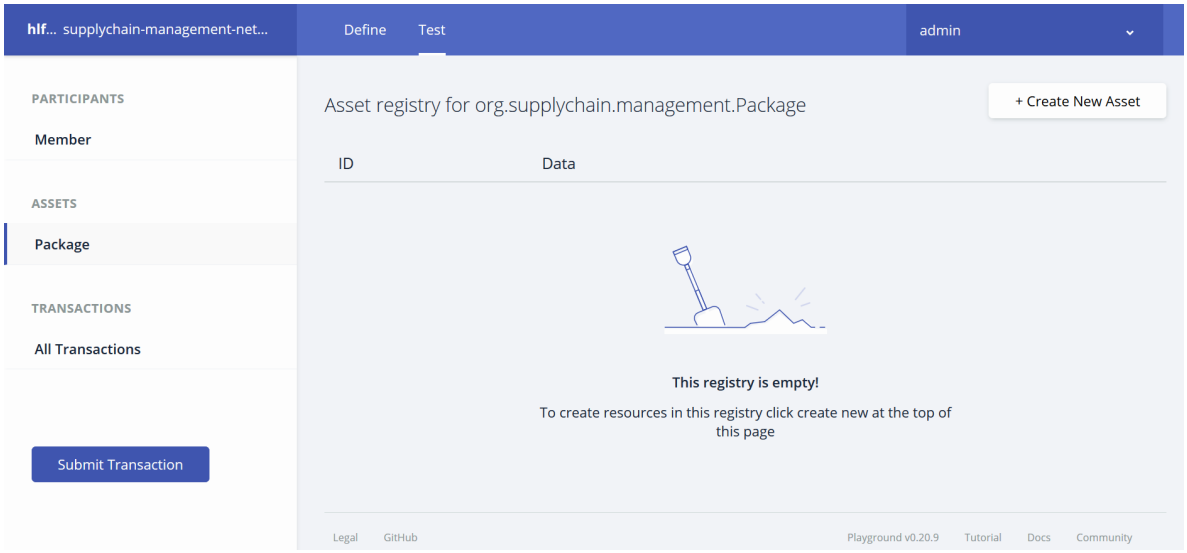
Check the network version in the bottom left.
It is deployed from version `0.0.2-deploy.0` to version `0.0.2-deploy.1`.
Now, the next deployment will be from `0.0.2-deploy.1` to `0.0.2-deploy.2`.

*****

Testing in Composer Playground

The composer playground provides a test interface, where you can test your deployed business network behaviour.

Click on **Test** tab in the composer playground.



In the test section, you can easily create an asset and participant and submit the transaction.

Let's test the supplychain management business network.

Create Participants

Click on **Member** under the *Participants* section on the left side.

> In the model file Member is defined as Participants.

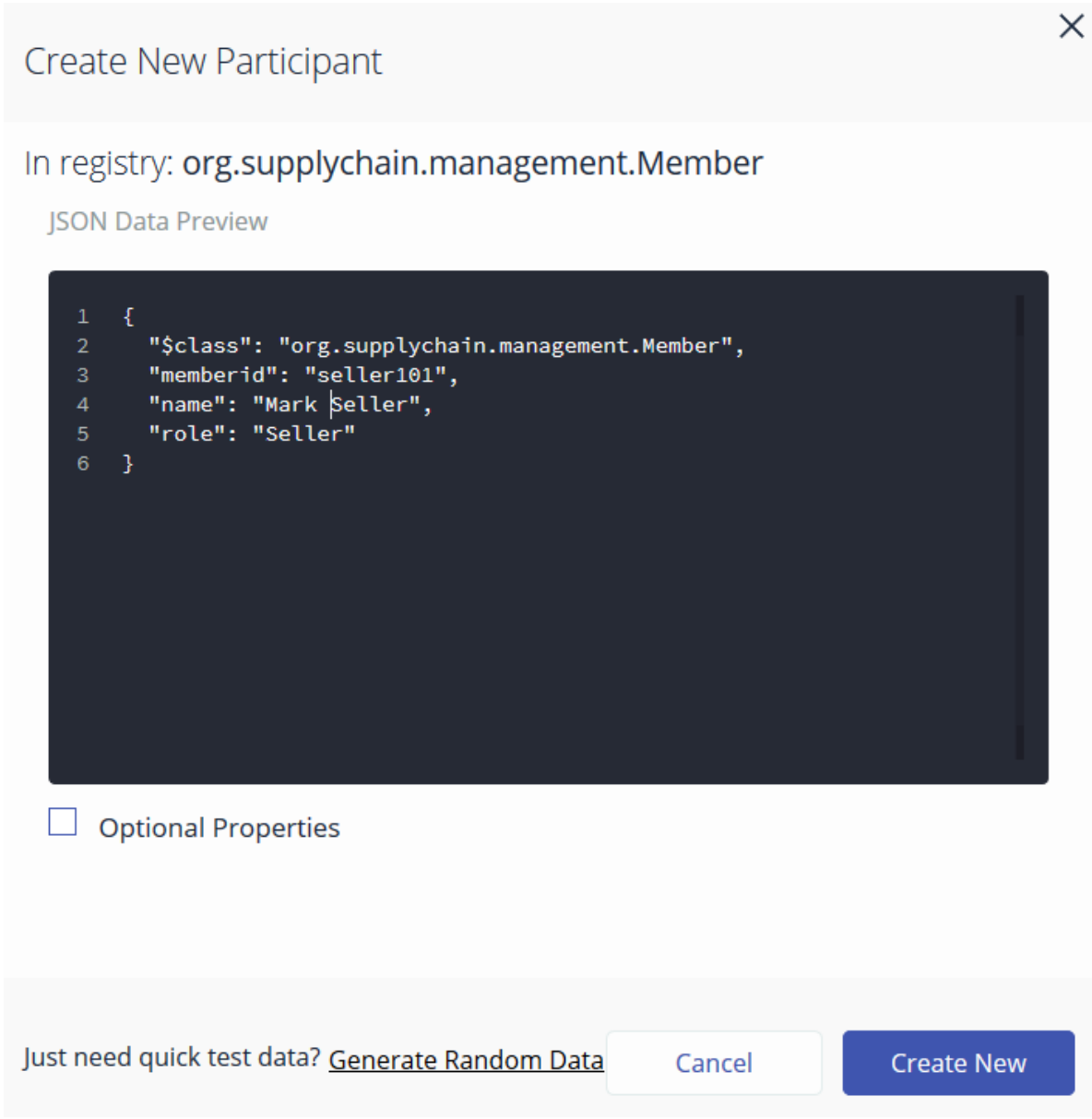
To create new participants, click on **+ Create New Participant** in the top-right section.

Create Seller

Click on **+ Create New Participant**.

```
Enter the memberId as seller101.
Enter the name as Mark Seller.
Enter the role as Seller.
```

Click on **Create New**.

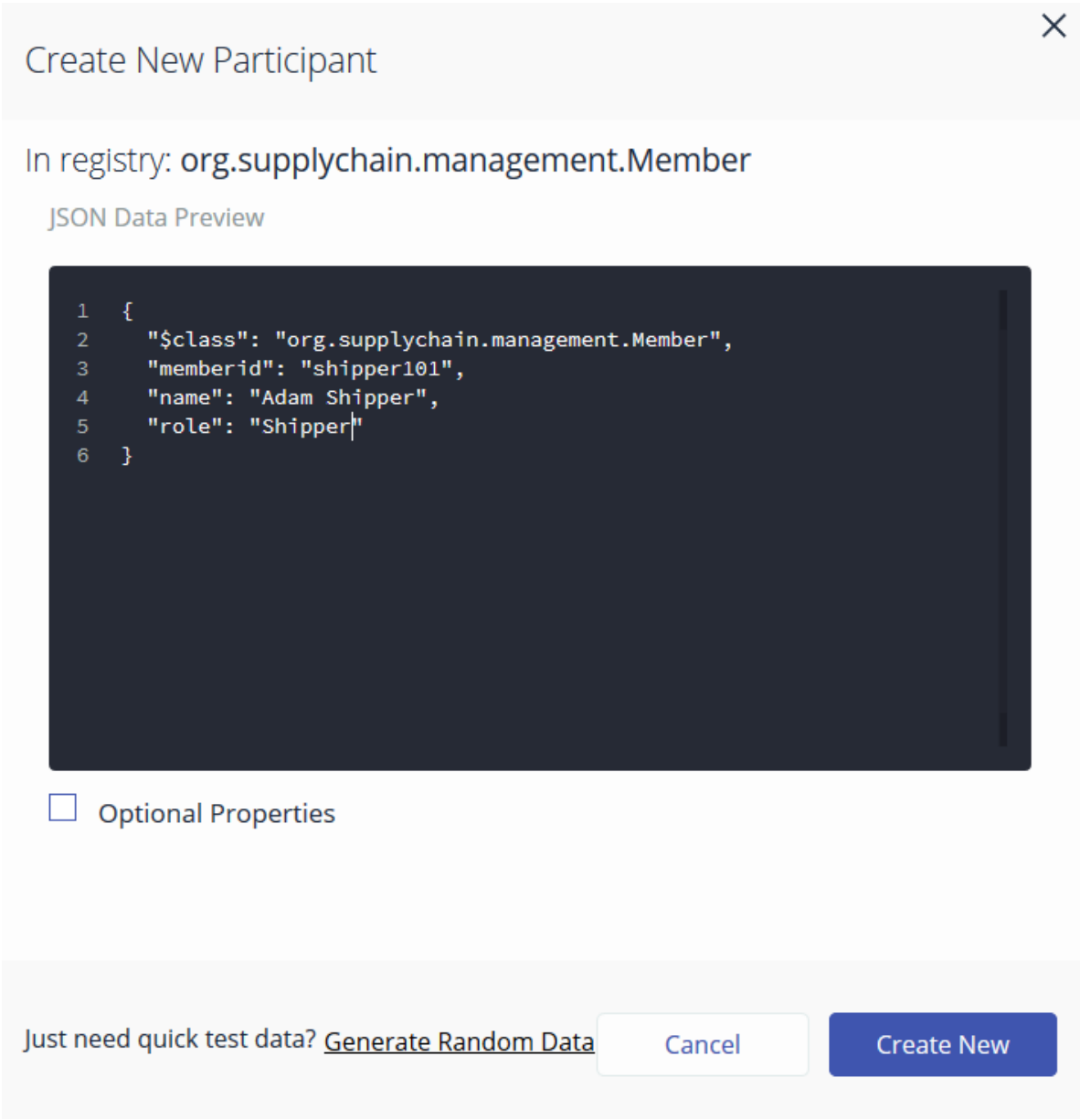


Create Shipper

Click on **+ Create New Participant**.

```
Enter the memberId as shipper101.
Enter the name as Adam Shipper.
Enter the role as Shipper.
```

Click on **Create New**.

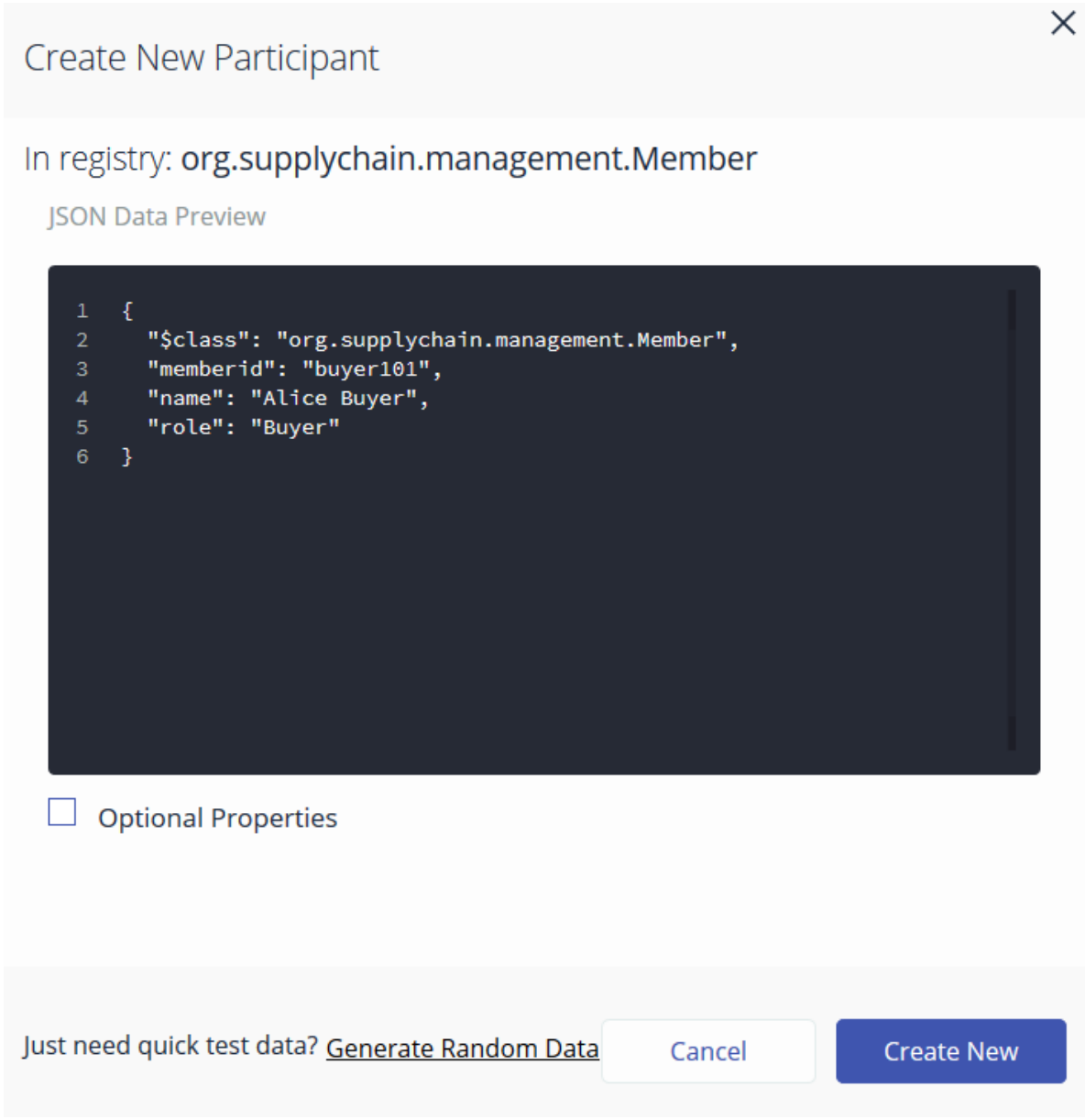


Create Buyer

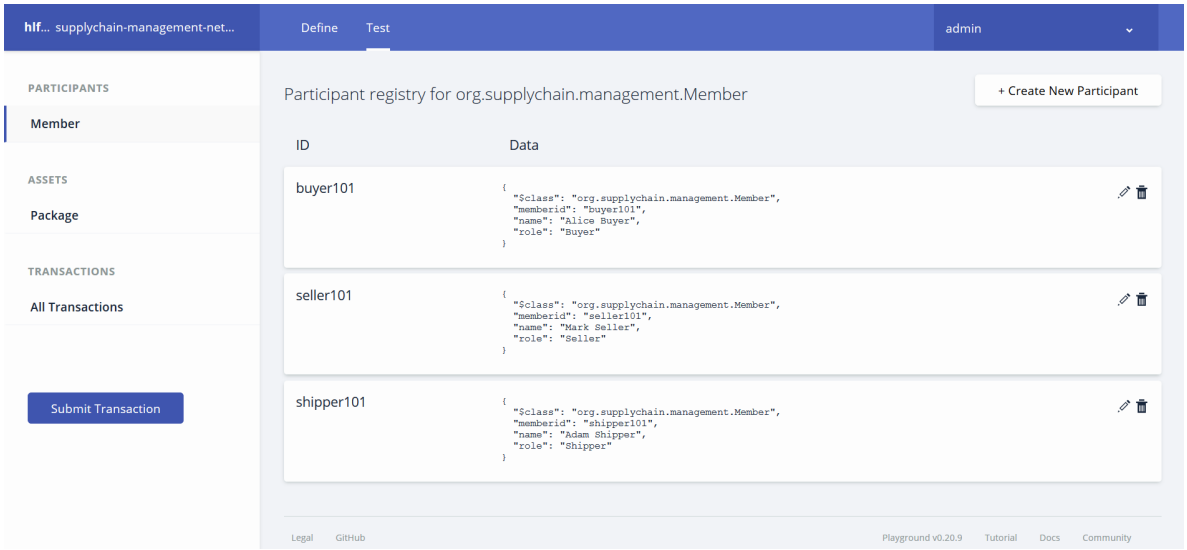
Click on **+ Create New Participant**.

```
Enter the memberId as buyer101.
Enter the name as Alice Buyer.
Enter the role as Buyer.
```

Click on **Create New**.



All the participants of the network are created.



Create Asset

Click on **Package** under the *Assets* section on the left side.

> In the model file Package is defined as Assets.

To create new assets, click on **+ Create New Asset** in the top-right section.

Create a package with **Seller** `seller101` as the owner.

```
Enter the packageId as package101.
Enter the status as Package_Ready.
Enter the owner as a resource:org.supplychain.management.Member#seller101.
```

For a date field, by default, it takes the current date and time of the machine in which the composer playground is running.

If you remember, the owner field in the package is a relationship. We have to provide a resolved relationship with the member.

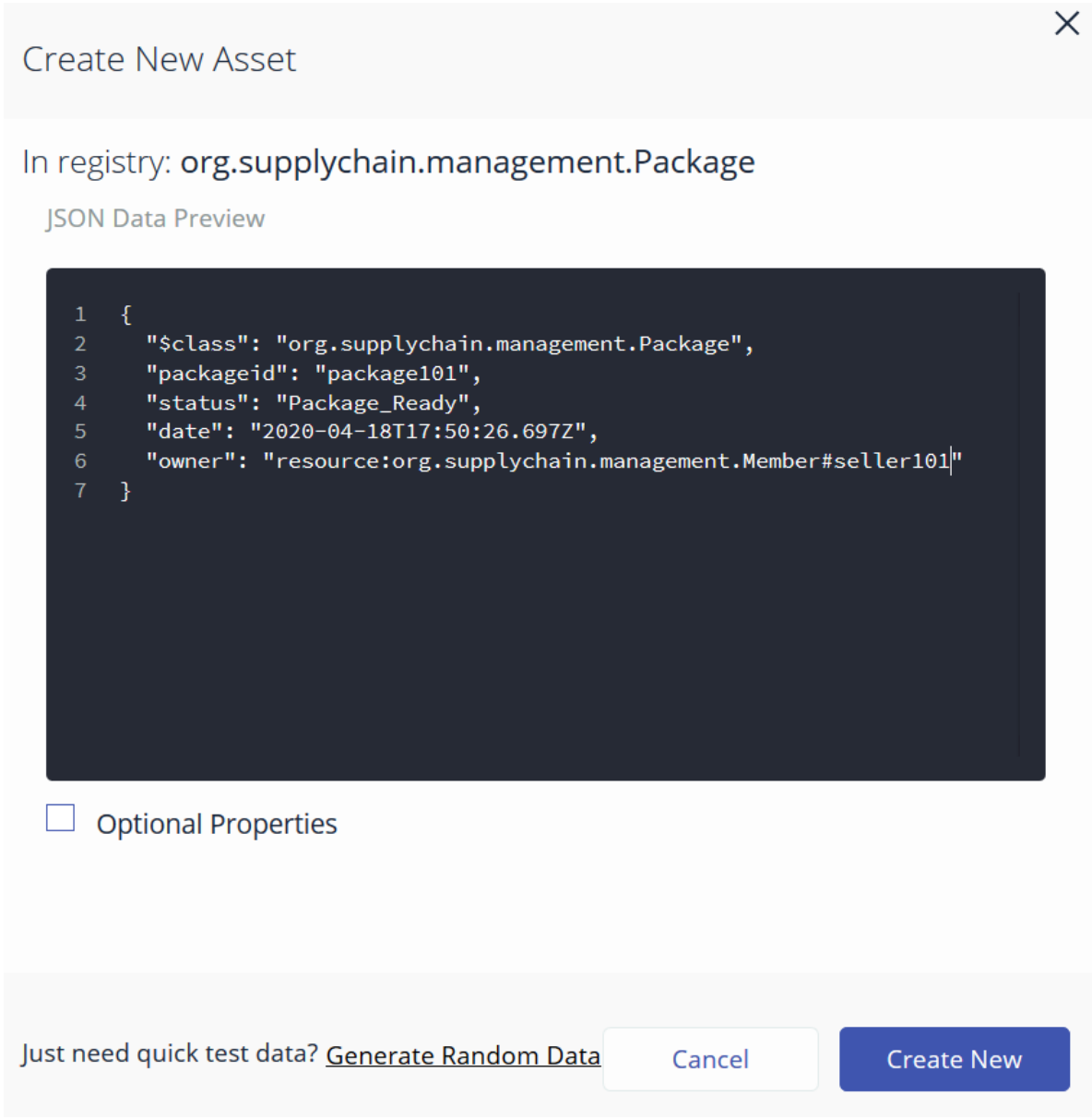
Resolved Relationship Syntax is simple.

```
resource:<namespace
  >.<resourcename>#<resourceID></resourceID></resourcename
></namespace>
```

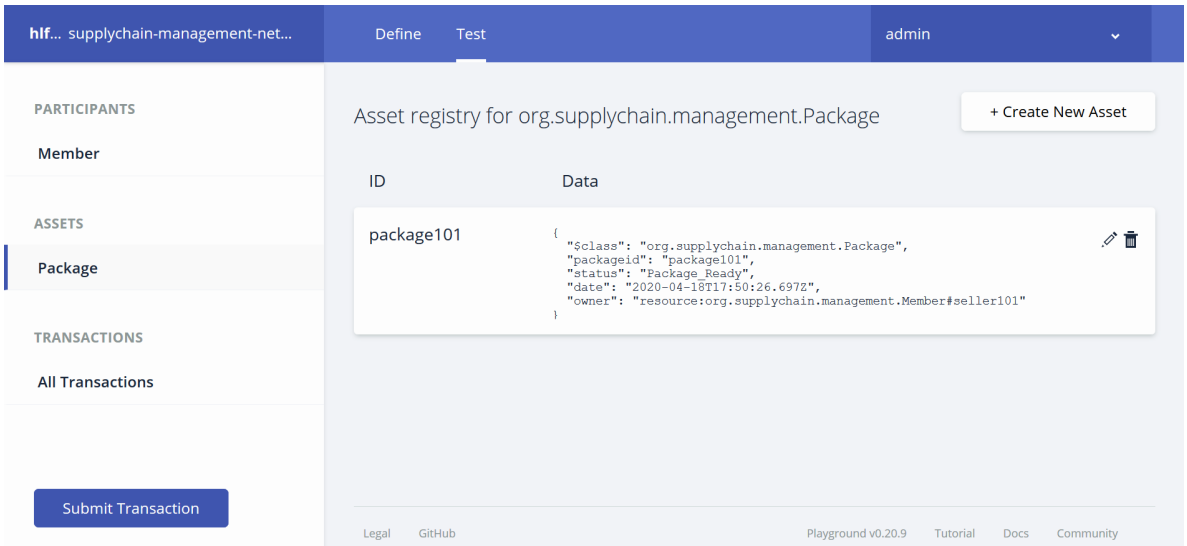
For Seller Member, the relationship is

```
resource:org.supplychain.management.Member#seller101
```

Click on **Create New**.



Now, the package is created with `package101` id and its owner is `seller101`.



Transfer the Ownership of the package

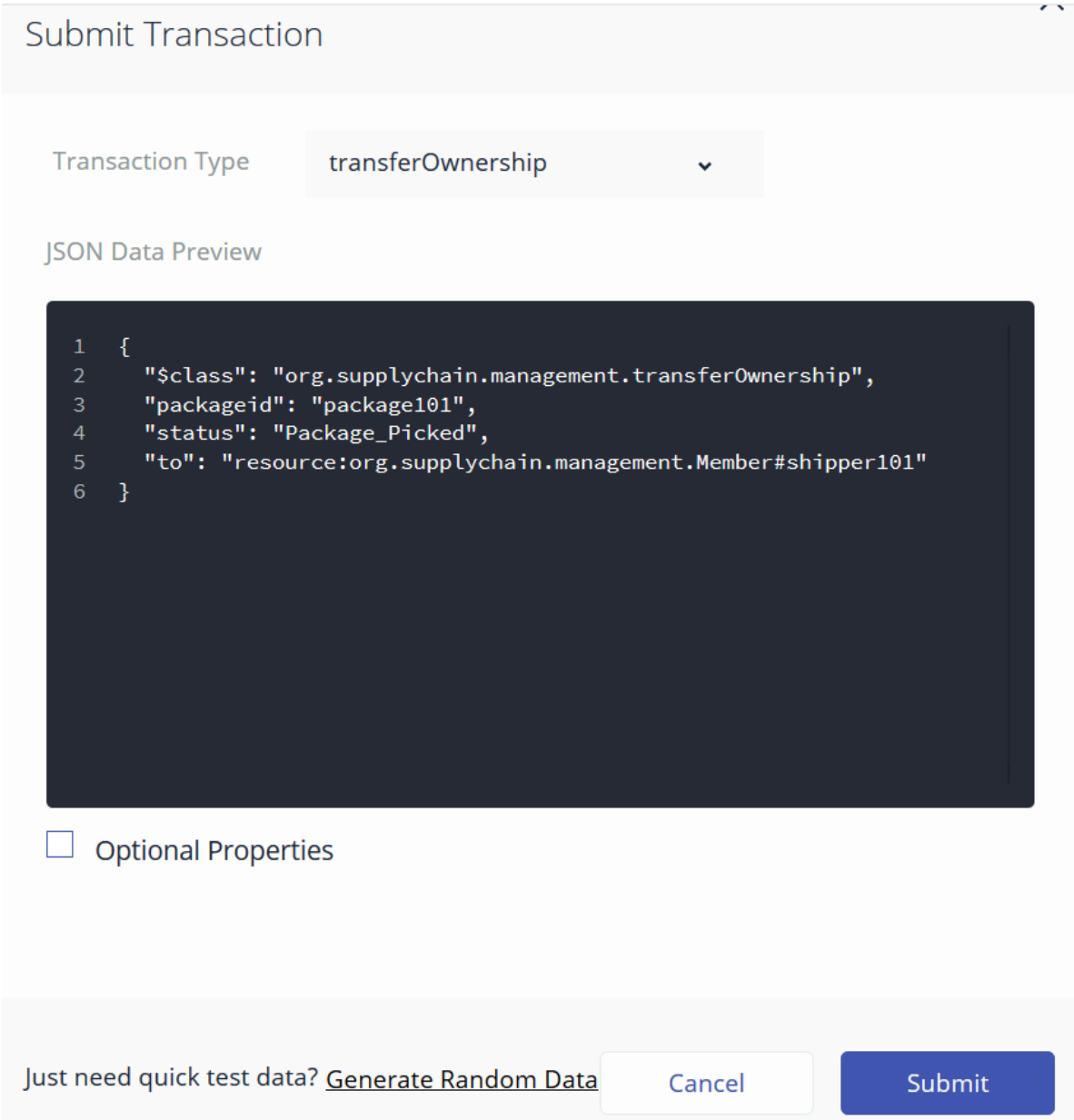
Click on **Submit Transaction** (bottom-left) to access all the transactions declared in the *Model File*.

From the **Transaction Type**, you can select the available transactions. In this case, we only have one transaction **transferOwnership**.

Move the package from the Seller to Shipper

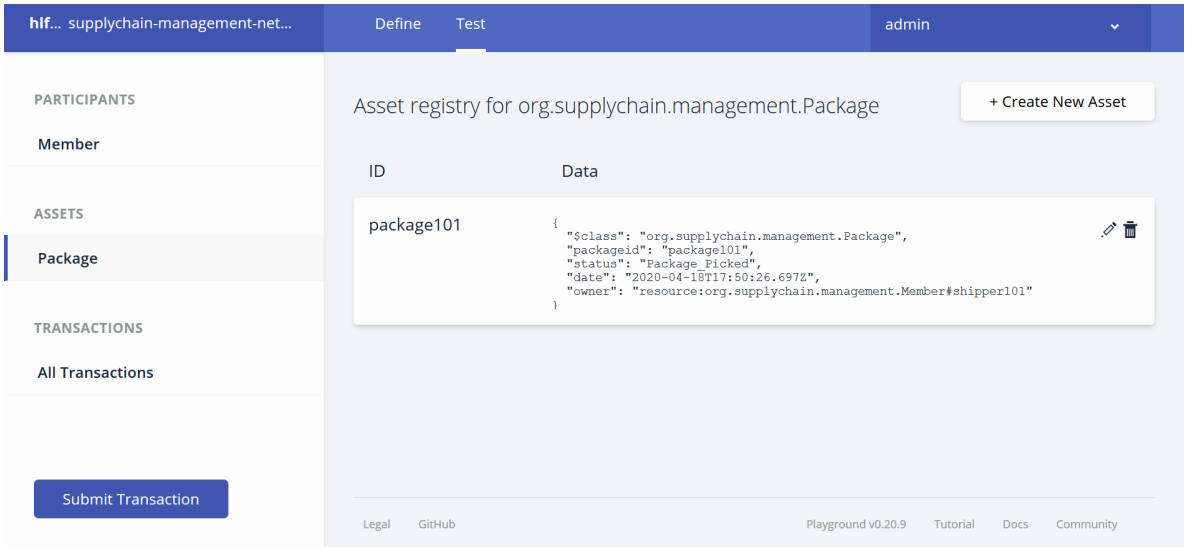
```
Enter the packageId as package101.
Enter the status as Package_Picked.
Enter the to as resource:org.supplychain.management.Member#shipper101.
```

> `to` field is a relationship with Member.



Submit the transaction.

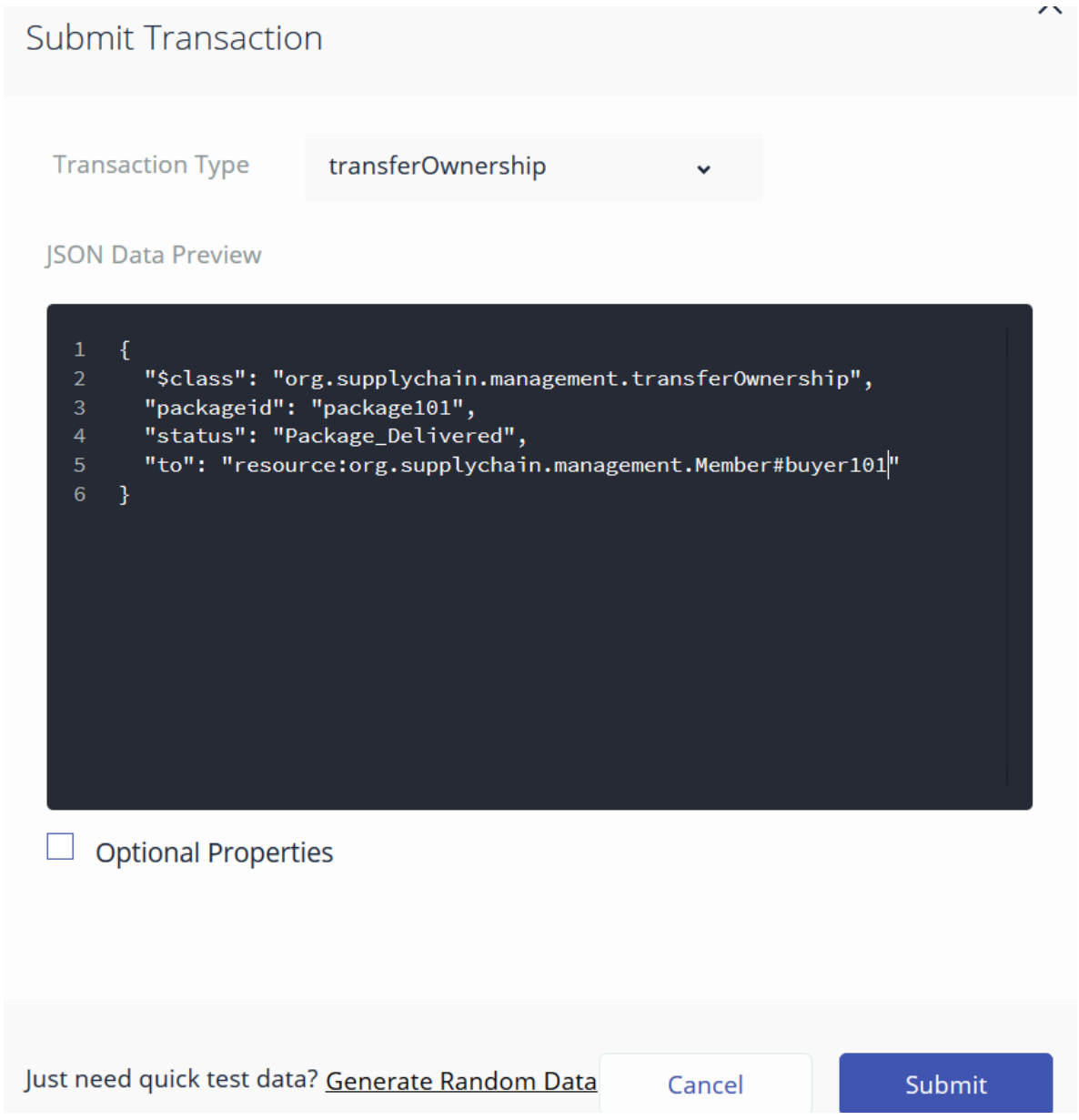
Now, check the package owner and package status, it is changed to *shipper101* and *Package_Picked* respectively.



Move the package from the Shipper to Buyer

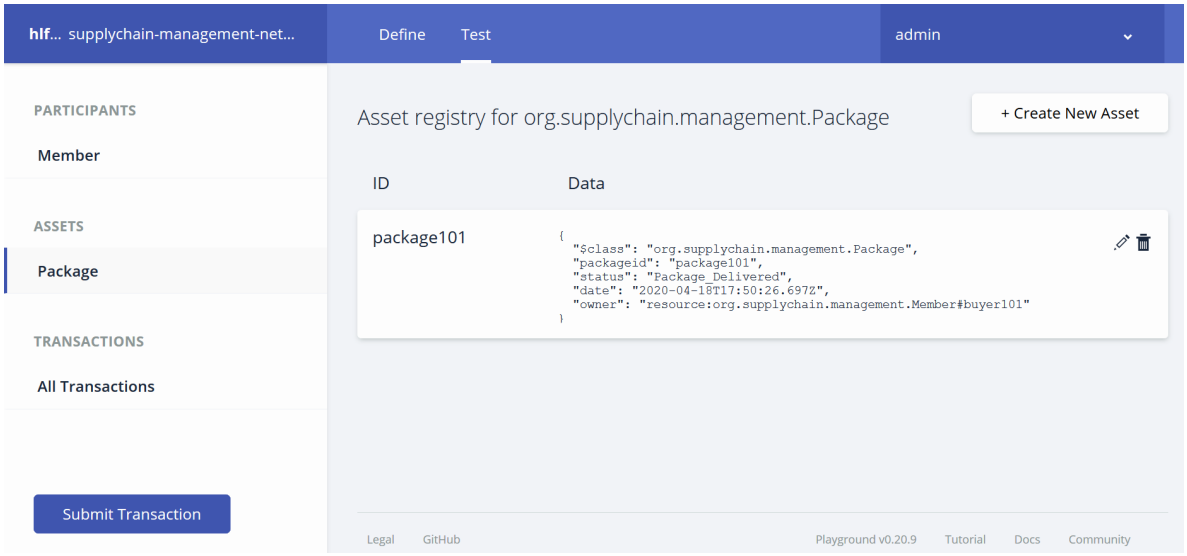
```
Enter the packageId as package101.
Enter the status as Package_Delivered.
Enter the to as resource:org.supplychain.management.Member#buyer101.
```

> `to` field is a relationship with Member.



Submit the transaction.

Now, check the package owner and package status, it is changed to *buyer101* and *Package_Delivered* respectively.



*****

Export the bna

In software development, it is always recommended to save efforts as much as possible.

While in development it is necessary to make the changes in the business logic, fix the bugs or add a new feature. At the same time, testing is an essential part of any project. To check if business logic is behaving expectedly or not.

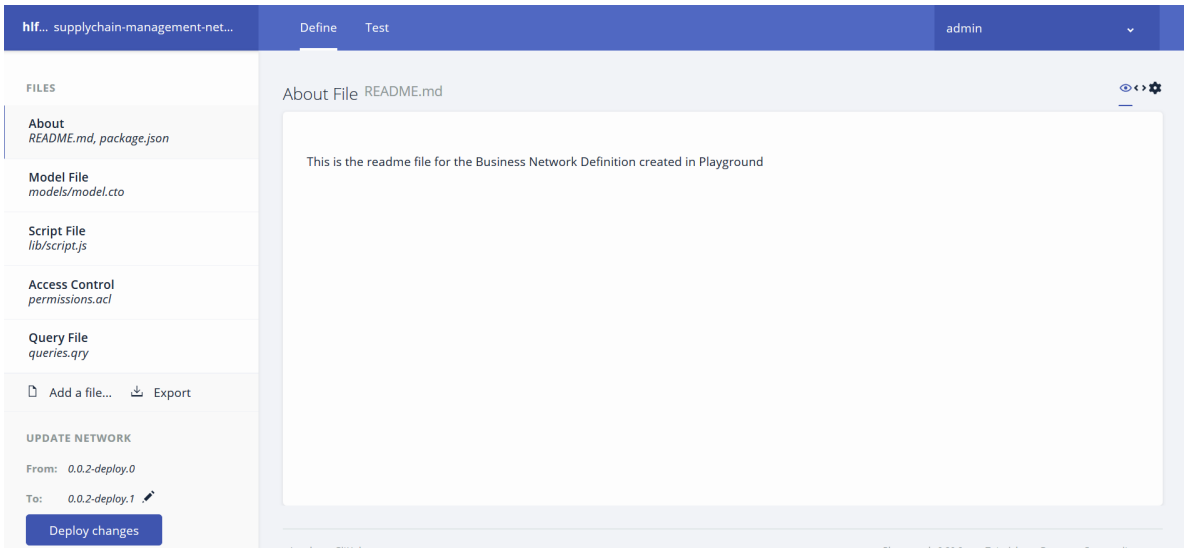
After making all the changes and testing you don't want to copy-paste your business logic every time you start the composer playground.

To save your extra efforts, composer playground provides you to export the business network in the `.bna` format. So, next time when you want to set up the project, you can easily import the business network with minimal efforts.

The composer playground exports the current version of the deployed network. If the current version is `0.0.2-deploy.1` then the `.bna` will export the `0.0.2-deploy.1` version. When you will deploy the `.bna` then its upgrade to the next version `0.0.2-deploy.2`.

Click on **Define** tab on the top.

Click on **Export** from the left bar. It will export the business network in the `.bna` format.



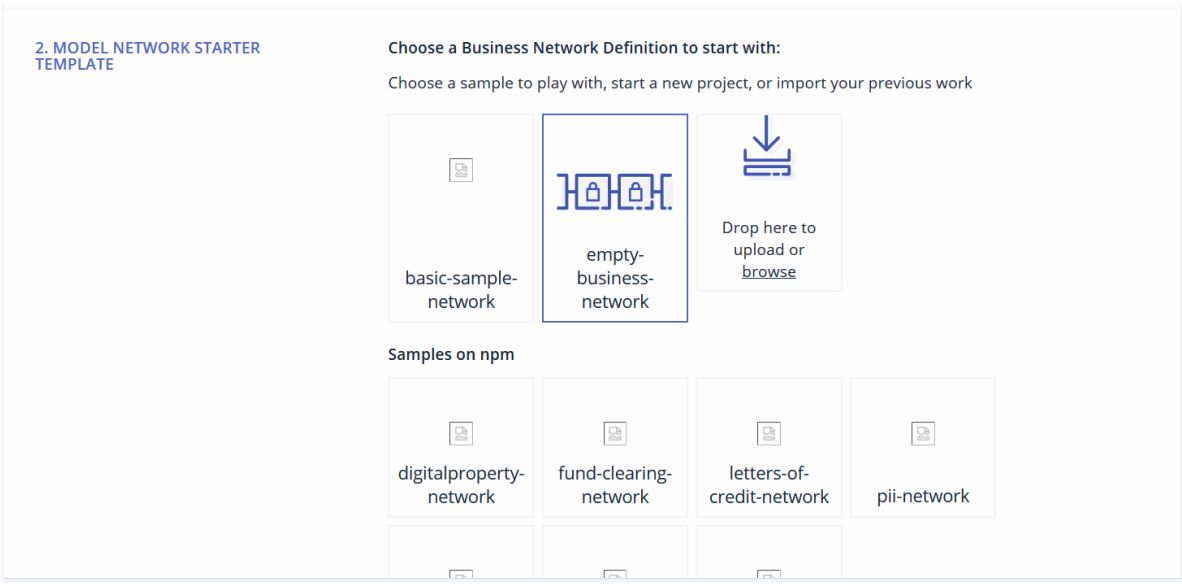
Import the business network

Go to the composer playground dashboard.

To go the dashboard from the connected network.
Click on **admin** drop down from the top-right.
Select **My Business Networks**.

Click on **Deploy a new business network**.

To import the `.bna`.
From the **Model Network Starter Template** click on **Drop here to upload or browse**.



This will import all the files and basic details of the business network.

You can update the basic details.

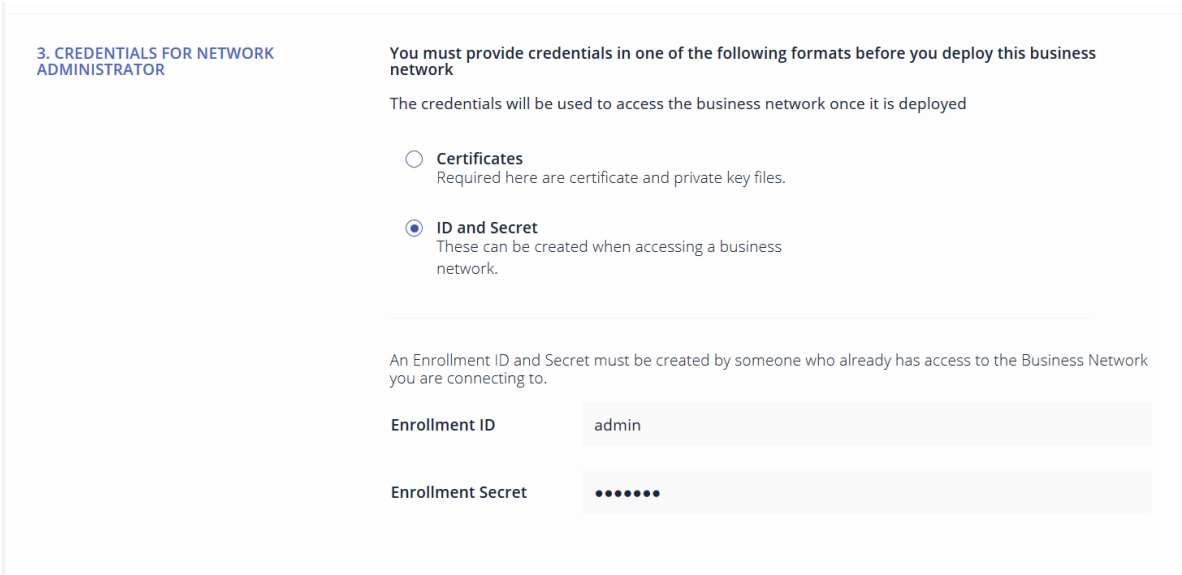
> If the same network is already deployed then you have to change the network name and admin card. It will not allow deploying two networks with the same name.

Enter the credential for network administrator

Enter the default admin id and secret. These secrets are required to deploy the business network on Hyperledger Fabric.

Select the **ID and Secret**.
Enter the default **Enrollment ID** and **Enrollment Secret**.

```
Enrollment ID: admin
Enrollment Secret: adminpw
```



Deploy

Click on **Deploy**. It will take some time to deploy on the network.

*****

Composer Rest Server

Hyperledger composer provides a `composer-rest-server` CLI tool. The `composer-rest-server` generates the REST APIs of the business network and serve them on port `3000`. This not only generates the API and serve them, but it also documents the APIs using Swagger now OpenAPI.

API is an application programming interface. API is the bridge between the backend and the user interface. The UI requires these APIs to interact with the backend.
Representational state transfer (REST) is a software architectural style that defines a set of constraints to be used for creating Web services.

For Example:
In the supplychain management project, the `Package` is an asset.
The `composer-rest-server` reads the model file and generates the Create, Read, Update and Delete rest APIs for the Package. These APIs are the interface to interact and modify the state of a package from outside.

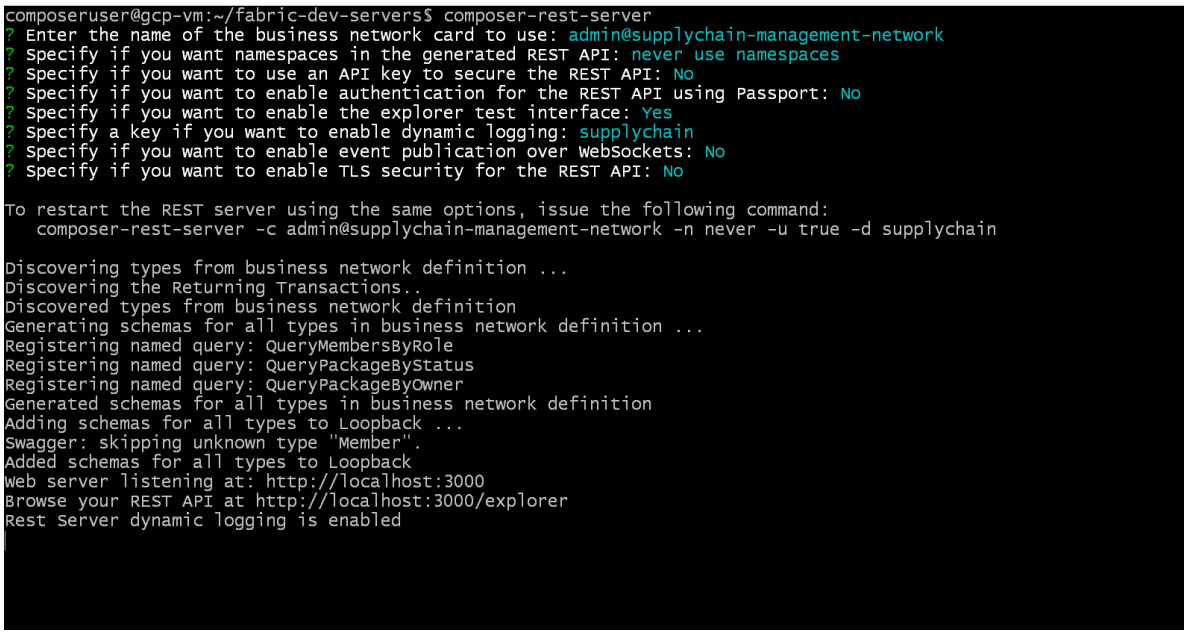
Setup composer rest server

Open a new terminal, if the playground is still running.
Composer rest server requires some configuration to setup.

```
composer-rest-server
```

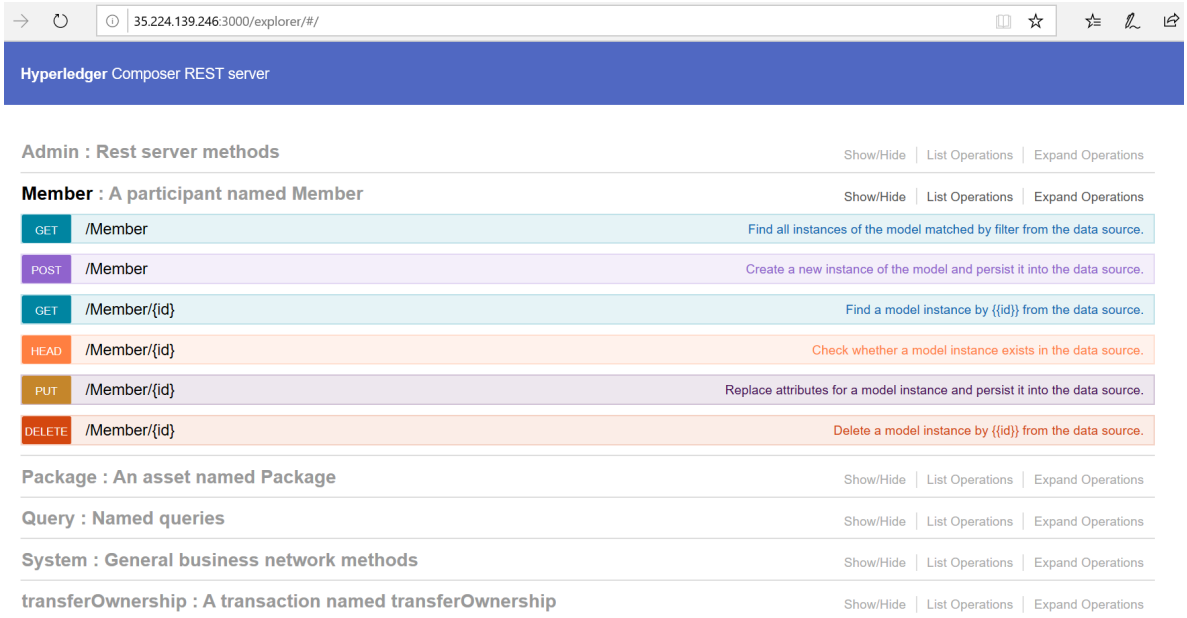
Enter the configuration as demonstrated in the screenshot.
All the configuration are self-explanatory.

Business Network Server Configuration



Explorer

Open the browser and go to `vmipaddress:3000/explorer`. If you're using your machine go to `localhost:3000/explorer`.



Create a member using Explorer APIs

Click on **Member**.
Click on **POST** request.

Copy the Example Value and Paste in the `data` field.

```
{
  "$class": "org.supplychain.management.Member",
  "memberid": "seller102",
  "name": "Jack Seller",
  "role": "Seller"
}
```

Click on **Try it out!**.

On success, the **Response Code** is **200**.

Get all the member

Click on **GET** request in **Member**.
Click on **Try it out!**.

This will return all the **Members** as an array of objects.

Create a package

Click on **Package** and **POST** request.

Copy the example value and paste it in the data field.

> If you remember `owner` is a Member relationship. So, we have to provide the resolved relationship.

```
{
  "$class": "org.supplychain.management.Package",
  "packageid": "package102",
  "status": "Package_Ready",
  "date": "2020-04-19T14:43:06.891Z",
  "owner": "resource:org.supplychain.management.Member#seller101"
}
```

Click on **Try it out!**.

This request will create a package with id `package102`. To check the package try the **GET** request of the **Package**.

transferOwnership

Transfer the ownership of the `package102` from `seller101` to `shipper101`.

Click on `transferOwnership` and select **POST** request.

Copy the example value and paste it in the data.

Remove `transactionId` and `timestamp`. The system will generate it.

```
{
  "$class": "org.supplychain.management.transferOwnership",
  "packageid": "package102",
  "status": "Package_Picked",
  "to": "resource:org.supplychain.management.Member#shipper101"
}
```

> Remember `to` is a Member relationship. So, we have to provide the resolved relationship.

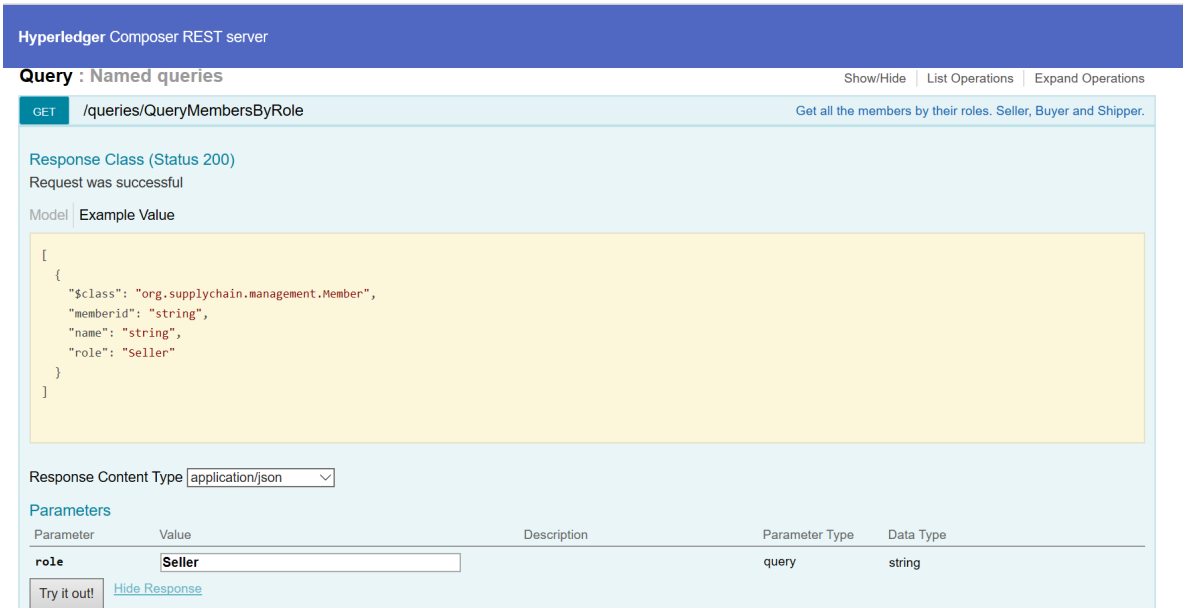
Now, to check, go to **Package** section and try **GET/{id}** request. In the parameter enter the `package102`. It will return only the `package102`.

Query

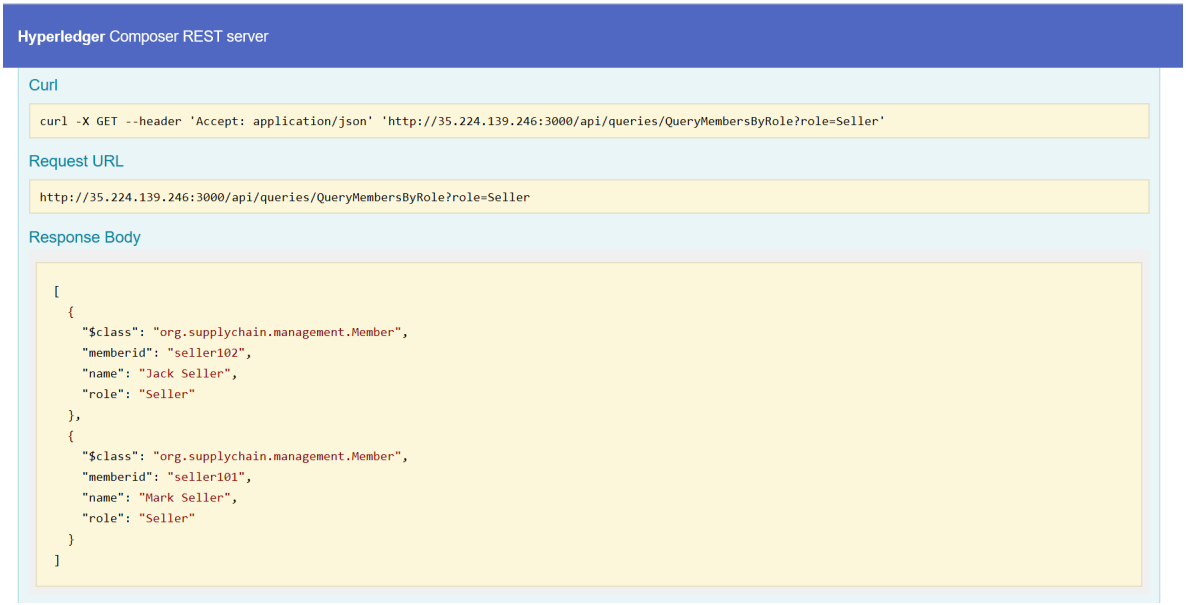
Query members by their role.

Click on **Query**.
Select **queries/QueryMembersByRole**.

In the **parameters** section, enter the role **Seller**.



It will return all the Sellers.



Similarly, you should try all the queries.

System

These requests help to interact with the composer system. For example, Get a history of the transactions. Every action in the blockchain which is writing something in the blockchain is a transaction.

Use screen

`screen` is a very special command in Linux. It allows to run multiple terminal sessions from a single SSH session.

Right now, the `composer-rest-server` is running in a terminal. If you want to close the terminal then you have to stop the server also. This is not an ideal situation when you want the server up and running 24/7 and exit the terminal.

Using `screen`, you can create a new screen or terminal session which you attach or detach anytime without stopping the process.

Stop the `composer-rest-server` using `Ctrl+C`.

Create a new screen

Syntax

```
screen -S nameOfScreen
```

Create a **rest-server** screen

```
screen -S rest-server
```

> If you're unable to create a screen, try with `sudo`.

Now, you're in a new screen.
Start the `composer-rest-server`.

Detach the screen

To detach the screen, use `Ctrl + a d`.
Press `Ctrl` all the time, press `a` and then press `d`.

The screen is detached and the process is still running. Check the explorer in the browser.

List the screen
To list the running screens.

Syntax

```
screen -ls
```

It will list all the screens with their ids. These are important to attach to the screen.

Attach the screen
To attach to the screen, copy the screen id using the list command.

Syntax

```
screen -r screenid
```

You are attached to the screen.

Terminate the screen
Stop the running service.
Use `exit` to terminate the screen.

*****

Frontend

A simple UI can make an application more appealing and useful.
You can't ask your users to use APIs or playground to interact with your business network.

Including a UI section and describing that is out of scope for this ebook.

Below are available options, there are many and you are free to get your hand dirty.
This UI will use the `composer-rest-server` APIs to interact with the business network.
Make sure `composer-rest-server` up and running. Use `screen` command to run the rest server in the background.

- The easiest way is by using the `expressjs` framework with any view engines like **pug**, **ejs**, **handlebar** etc.
- Any UI Framework like **Angular**, **Reactjs** or **Vuejs**. The learning curve will be a little steep but the result will be satisfying. There are many CSS library available for each framework. These libraries include many UI components like Header, Footer etc. with default CSS.
- If you are not good in **CSS**, then don't worry about that. You can use **Bootstrap**, **SemanticUI** etc.

The above options require a basic understanding of Javascript.

There are many frameworks available in each language but JS frameworks are most popular.

Hosting on Github Pages

It is good to become friend with git and GitHub as soon as possible. Github is the perfect place to showcase your work to the world.

GitHub not only provides you with the facility to save your code but also can host free of cost.
You can host your UI on GitHub Pages. GitHub Pages only host static websites.

For this you require:

- A GitHub account
- Create a repository and push the project in it
- From the repository, settings use GitHub Pages to host

> It will host on .github.io/repositoryName

There are many great tutorials on the internet which can guide you on how to host on GitHub Pages.

If you're using frameworks like Angular, Reactjs or Vuejs, they provide a facility to build the UI.
This build option creates a static content which you can host using GitHub Pages.

*****

Conclusion

The purpose of this ebook is to deliver a basic understanding of the Hyperledger Composer. For simplicity, we used the supplychain management. Now, as you know the basic concepts of Hyperledger Composer, you can build any solution in industries like Healthcare, Food Tracking, Real Estate, Document Stamping etc.

If you're interested, you can add more features to this project.
Try to add below functionalities:

Add temperature reading on the package

Create a test API which will return the temperature reading. Call this API every 5-10 mins or any interval and update the package temperature. Whenever the temperature rises above a threshold temperature, trigger a transaction that package is spoiled. For threshold temperature, you can add a variable in the package.
Add functionality of accepting the package by the buyer. Buyer can only accept the package if the package is fresh.

Instead of creating a test API, you can also use the weather APIs. There are many weather APIs which are free to consume up to a certain limit.

Add IoT sensor

This is an advance option. Instead of weather APIs or a test API, use the IoT sensors to read the temperature. The simple option is using a breadboard and a temperature sensor.

Logistic Blockchain Network

Earlier I had created a similar project. You can checkout it here.

*****

References

- Hyperledger Projects
- Hyperledger Fabric
- Hyperledger Composer
- Query Language
- Query Syntax
- Registry
- Transactio Processor Function
- Classes in Composer
- GitHub Pages
- View Engine EJS
- Basics of Javascript
- Javascript Handbook by Flavio Copes
- Objects in Javascript
- Promises in Javascript
- async await in Javascript

*****

Thank you for Reading