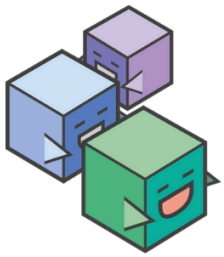
create first then learn

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 have worked as a Blockchain developer. During that time, I came across many blockchain platforms and development tools. Among all of them, Hyperledger Composer stands out from the rest.

> **The Hyperledger Composer is deprecated on 18th August 2019.**
> Even though it is deprecated, it is still a great tool to create a Proof of Concept (POC) on Hyperledger Fabric.

For a student, building a project on a latest technology like blockchain is a great addition to your profile and experience.

What you should expect from this eBook?

This eBook focuses on hands-on learning and covers less theory. It is not for someone seeking a deep dive into how Composer works internally or an in-depth explanation of its concepts.

This eBook covers the basic definition of all the concepts you will use while developing the project.

You will learn how to:

- Create a virtual machine in the cloud
- Set up Composer
- Design and develop the project
- Deploy and test the project in Composer Playground
- Generate REST APIs using composer-rest-server
- Create the frontend of the project and host it on GitHub Pages

Should have

Before you begin, make sure you have the following:

- A basic understanding of blockchain.
- Basics of JavaScript — JavaScript knowledge is essential when working with Composer.
- An 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 have had the opportunity to work on Blockchain, Nodejs, Golang, and Docker.

I have learned about other technologies 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 is not my specialty, which is why I did not 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 thank the entire Hyperledger Composer Community for creating such an amazing tool. Many developers entered the blockchain domain because of the simplicity of Composer.
It is unfortunate that it is deprecated, but it sets a great example of easy automation, wrapping the complex Hyperledger Fabric into the easy-to-use Hyperledger Composer.

Table of Contents

*****

What is Blockchain and Hyperledger Fabric?

Before you start exploring Hyperledger Composer, it helps to understand Blockchain, Hyperledger, and Hyperledger Fabric.

What is Blockchain?

Blockchain is a distributed database that stores data in cryptographically secured blocks. These blocks link to each other in sequence, and the chain never goes backward. Every block and its data carry a timestamp, and all operations — create, update, or delete — append after the latest block. This design makes it straightforward to verify any change in the state of the data.

To understand how blockchain differs from a conventional database, think of a conventional database as ***write using pencil*** and blockchain as ***write using pen***.

With ***write using pencil***, anyone can erase, update, or delete information on a page without leaving a trace, making it very difficult to verify the authenticity of that information.

With ***write using pen***, you cannot easily update or delete information. Even when a change does occur, the original mark remains visible, making the change easy to detect.

> This is an example for understanding the concept. No literal ***write using pen*** mechanism exists in current systems, but you can achieve something similar. Create multiple copies of the data ***write using pencil*** and distribute them to all participants. To create data, every copy must create it. To update data, every copy must update it. If one participant updates their copy without informing the others, you can immediately identify the compromised copy.

What is Hyperledger?

Hyperledger is a project of The Linux Foundation. It is an open-source community focused on developing a suite of stable frameworks, tools, and libraries for enterprise-grade blockchain solutions. Its offerings fall into three categories:

- **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. Its modular architecture accommodates the diversity of enterprise requirements through plug-and-play components such as consensus, privacy, and membership services. Hyperledger Composer builds directly on top

of Fabric, so understanding Fabric's structure will help you reason about everything that follows.

*****

Introduction of Hyperledger Composer

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

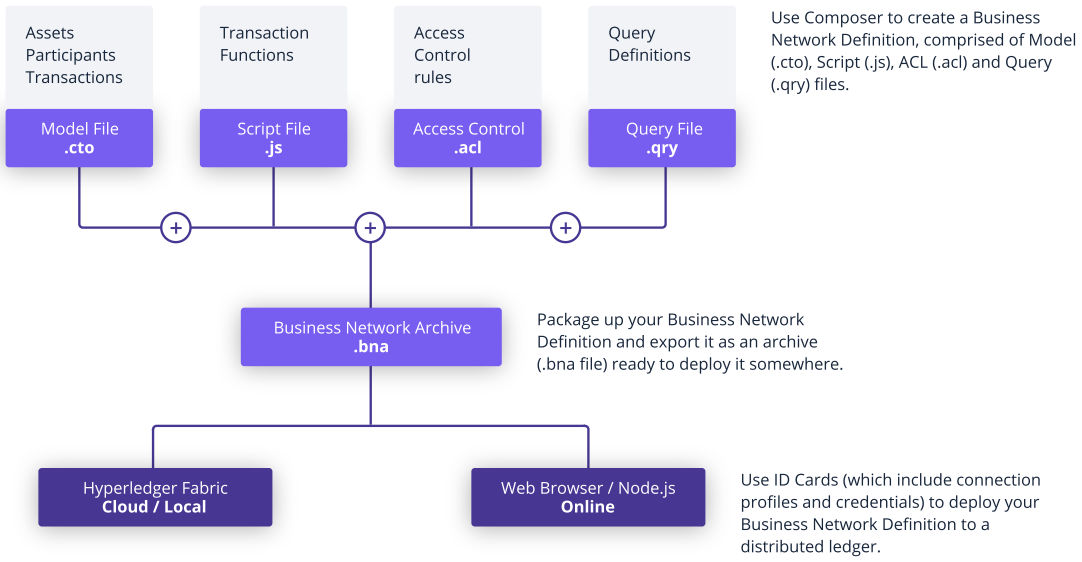
> Hyperledger Composer supports the existing Hyperledger Fabric blockchain infrastructure and runtime, which supports pluggable blockchain consensus protocols to ensure that transactions are validated according to the policy by the designated business network participants. Learn more.

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

How Hyperledger Composer Works?

Hyperledger Composer organizes a business network around 3 basic units:

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)

A BNA is a package that bundles the complete business logic and its dependencies together. The `.bna` extension identifies these archive files.

A BNA contains 4 types of files, each serving a distinct purpose in the business network definition:

1. **Model File** (.cto): The model file uses a modelling language to define business logic. Its extension is `cto`. In the model file, you declare all the assets, participants, and transactions the business logic requires — declarations only, with no functionality.
2. **Transaction File** (.js): The transaction file defines the functionality of each transaction declared in the model file. You write the transaction file in JavaScript.
3. **Access Control** (.acl): The access control file defines 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 data from the data set. You write queries in Hyperledger Composer using a bespoke query language. Define all queries in a single query file called `queries.qry` within a business network definition.

Together, these four file types give you everything needed to specify, secure, and query a complete business network.

*****

Environment Requirement

The Hyperledger Composer documentation supports installation on Ubuntu or macOS only. 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

Before you begin, ensure your system has the following tools installed at the required versions:

- **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, skip the native installation — it causes significant compatibility issues. Instead, use one of these options:

1. Create a Linux virtual machine on Windows using **Hyper-V** or a similar tool like **VirtualBox**.

   > ⚠️ Do not install VirtualBox without reading the instructions for Windows first. Many Windows machines do not support it, and it may conflict with **Hyper-V**. Use **Hyper-V** to avoid these conflicts.

2. Create a Linux virtual machine on a cloud provider. This option offers several advantages: it eliminates local VM setup, many cloud providers offer generous free credits, and you gain practical exposure to cloud environments.

This e-book uses the second option — the cloud. You will find all the instructions and background knowledge you need to set up a cloud VM in the sections that follow.

You can create a VM on any of the cloud providers listed below. The next sections provide step-by-step instructions for each one:

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

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

*****

Create a Linux VM in Azure

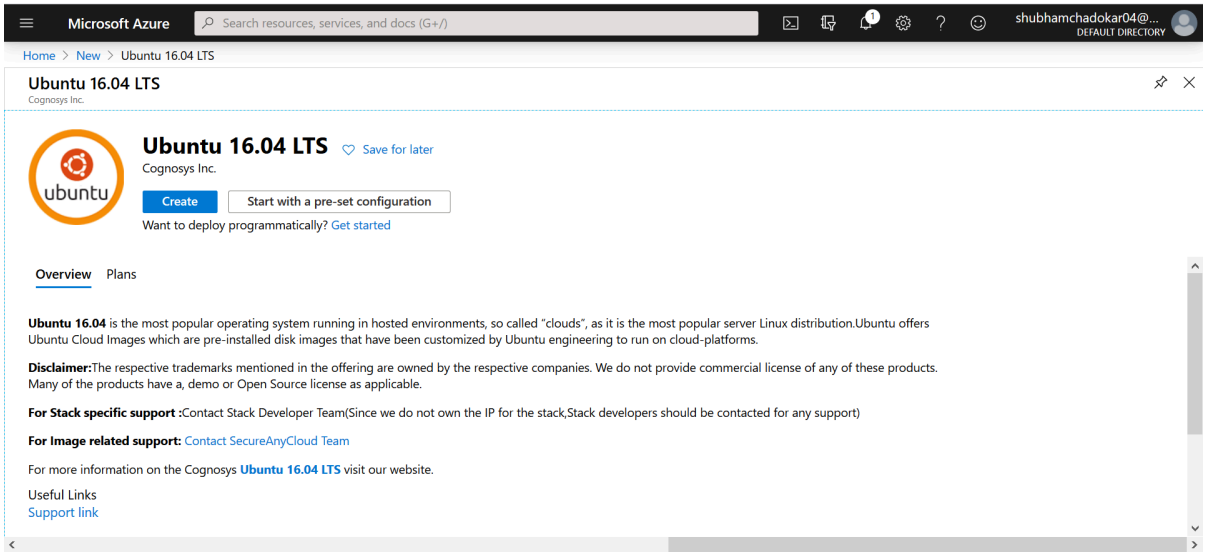
Create a new account in Microsoft Azure. Azure gives new users free credits for 30 days and many services free of cost up to a limit for 12 months.

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

Log in to your Azure account.

Step 1 Search for Ubuntu 16.04 LTS

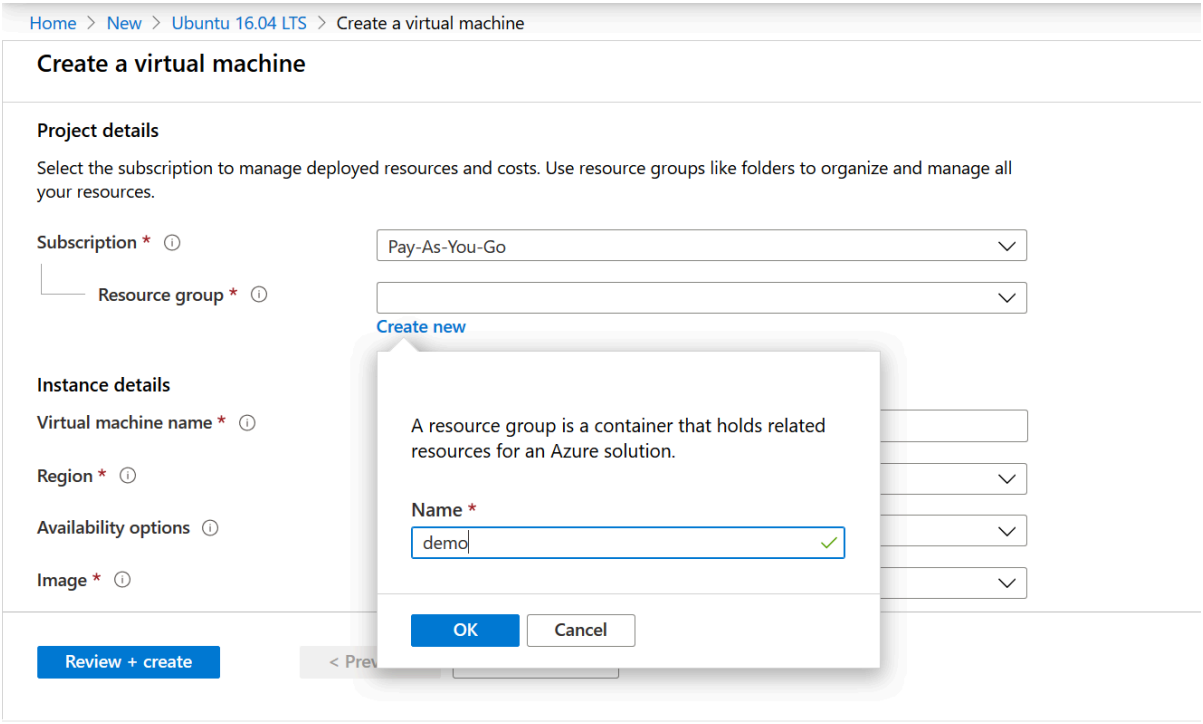
In the search bar, search **Ubuntu 16.04 LTS** and select create.



Step 2 Enter the Project details

In Azure, all related resources must live in a container called a Resource Group.

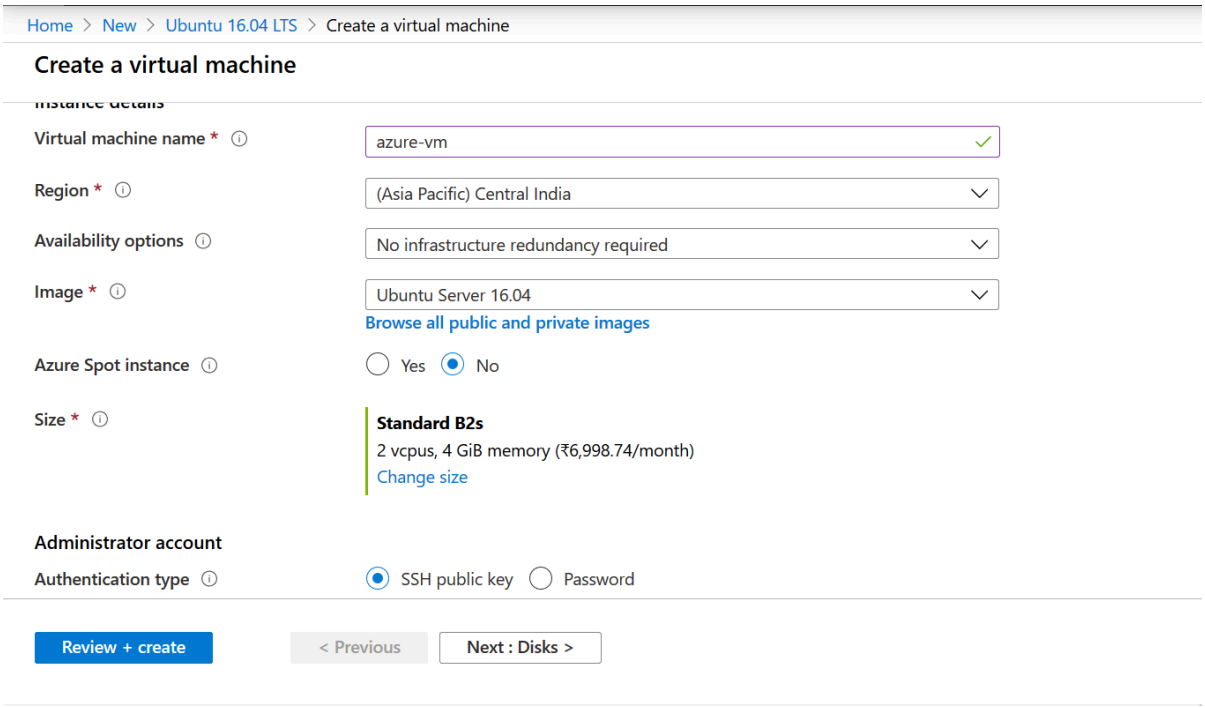
Create a new resource group **demo**.



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

Azure offers two options for logging in to the VM: using an SSH key or a password.

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

Let's set up the SSH key.

Open any command line tool, such as a bash shell.

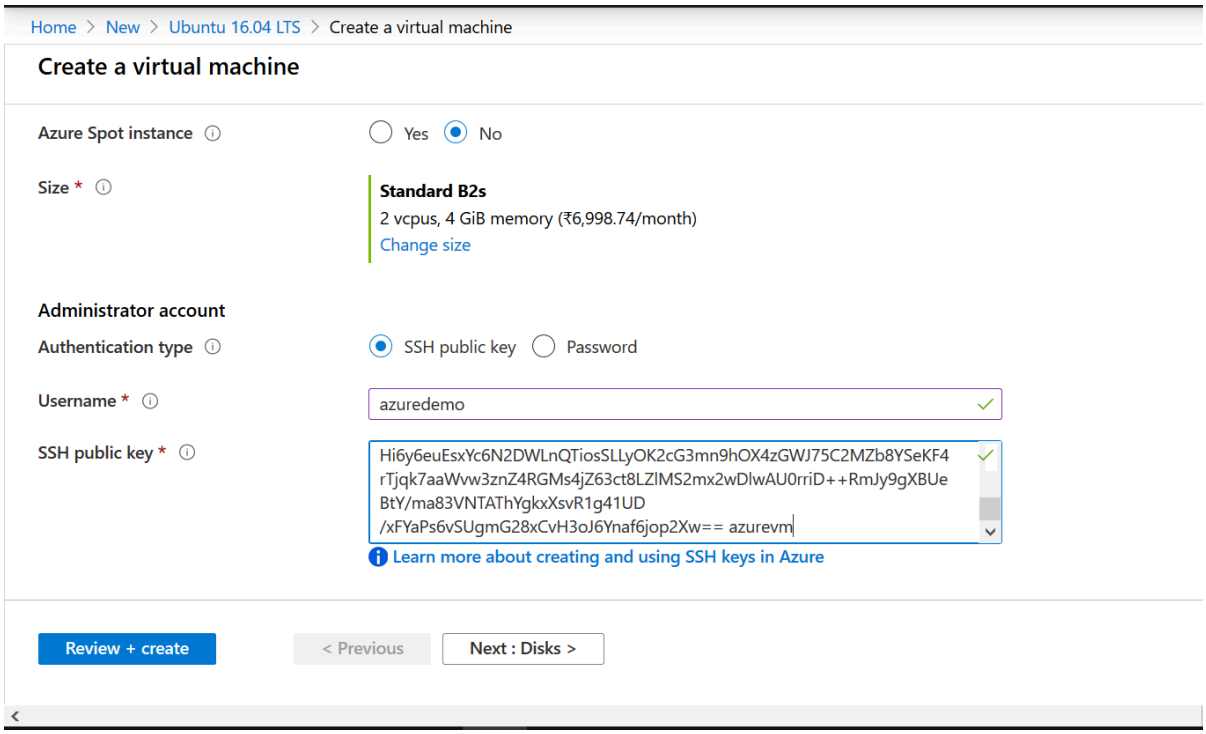
Use the `ssh-keygen` command to create keys. By default, it saves these keys in the `~/.ssh` directory.

```
$ 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-BAVLQ[
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.

A passphrase protects your private key when you log in to the VM. With the keys generated, fill in the following fields:

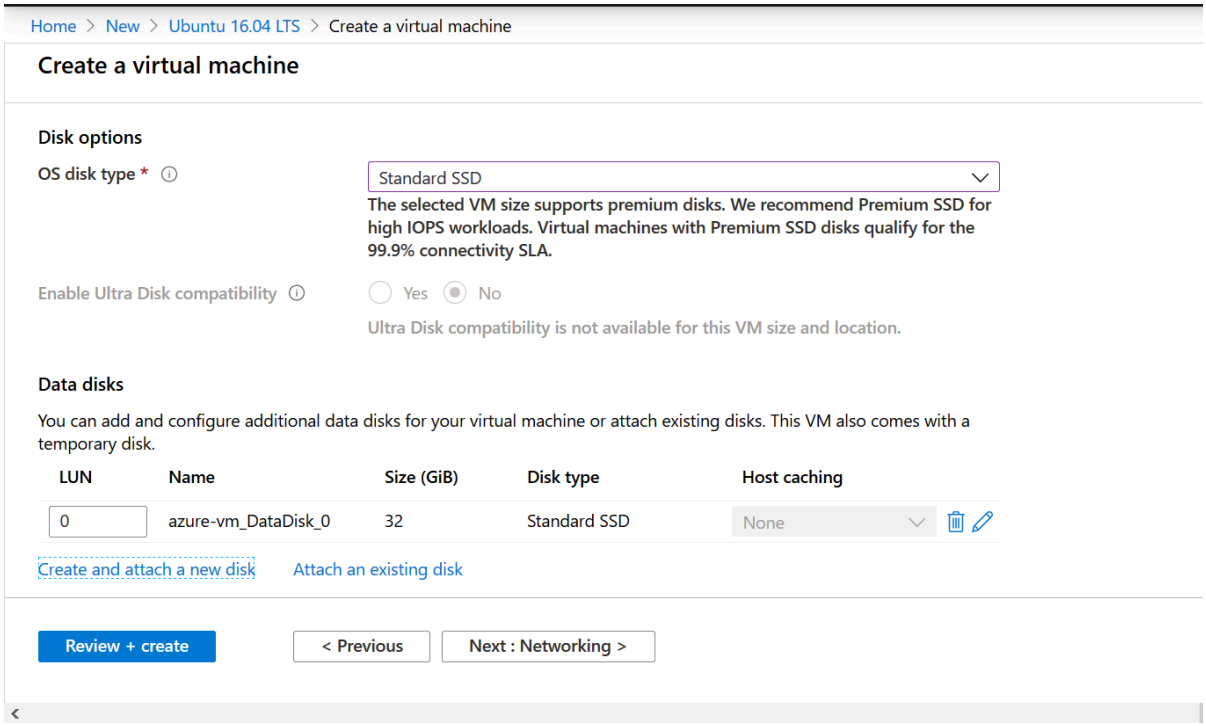
- Username: `azuredemo`
- Open the `id_rsa.pub` file to view your public key.
- Copy the key and paste it into **SSH public key**.


Home > New > Ubuntu 16.04 LTS > Create a virtual machine
Create a virtual machine
Azure Spot instance
Yes
No
Size *
Standard B2s
2 vcpus, 4 GiB memory (₹6,998.74/month)
Change size
Administrator account
Authentication type
SSH public key
Password
Username *
azuredemo
SSH public key *
Hi6y6euEsxYc6N2DWLnQTiosSLLyOK2cG3mn9hOX4zGWJ75C2MZb8YSeKF4
rTjqk7aaWvw3znZ4RGMs4jZ63ct8LZlMS2mx2wDlwAU0rriD++RmJy9gXBUe
BtY/ma83VNTAThYgkxXsvR1g41UD
/xFYaPs6vSUgmG28xCvH3oJ6Ynaf6jop2Xw== azurevm
Learn more about creating and using SSH keys in Azure
Review + create
< Previous
Next : Disks >

Click **Next**.

Step 3 Disk

The Disk section lets you add extra disk space to meet your requirements. The Standard B2s VM comes with 32 GB of temporary storage.



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

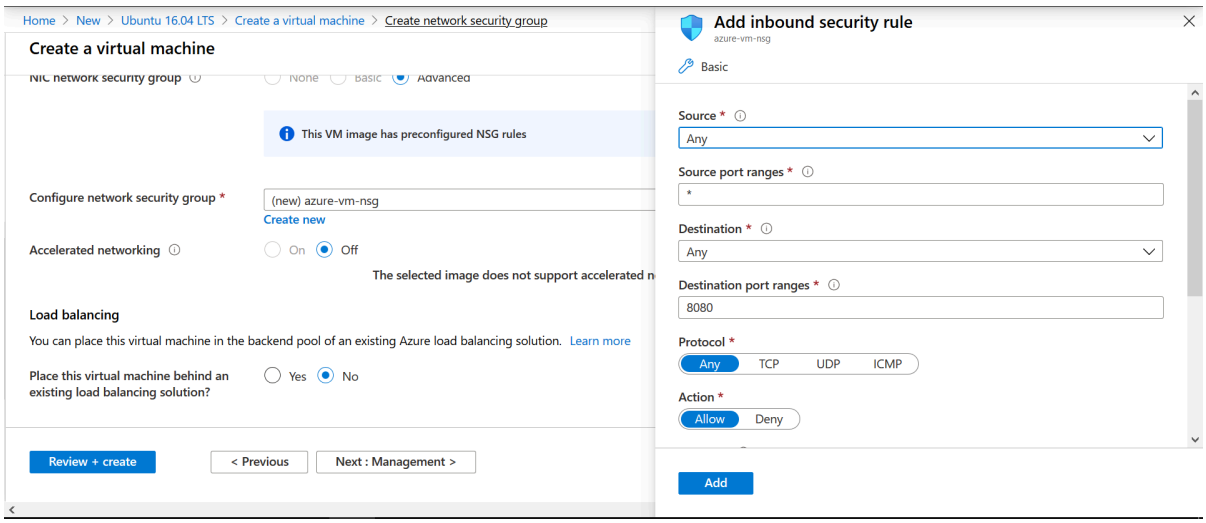
Click **Next** to proceed to Networking.

Step 4 Networking

The Networking section lets you define network connectivity for the virtual machine. For this demo, you will skip network restrictions so that anyone can reach the VM.

Create a new **Network Security Group**.

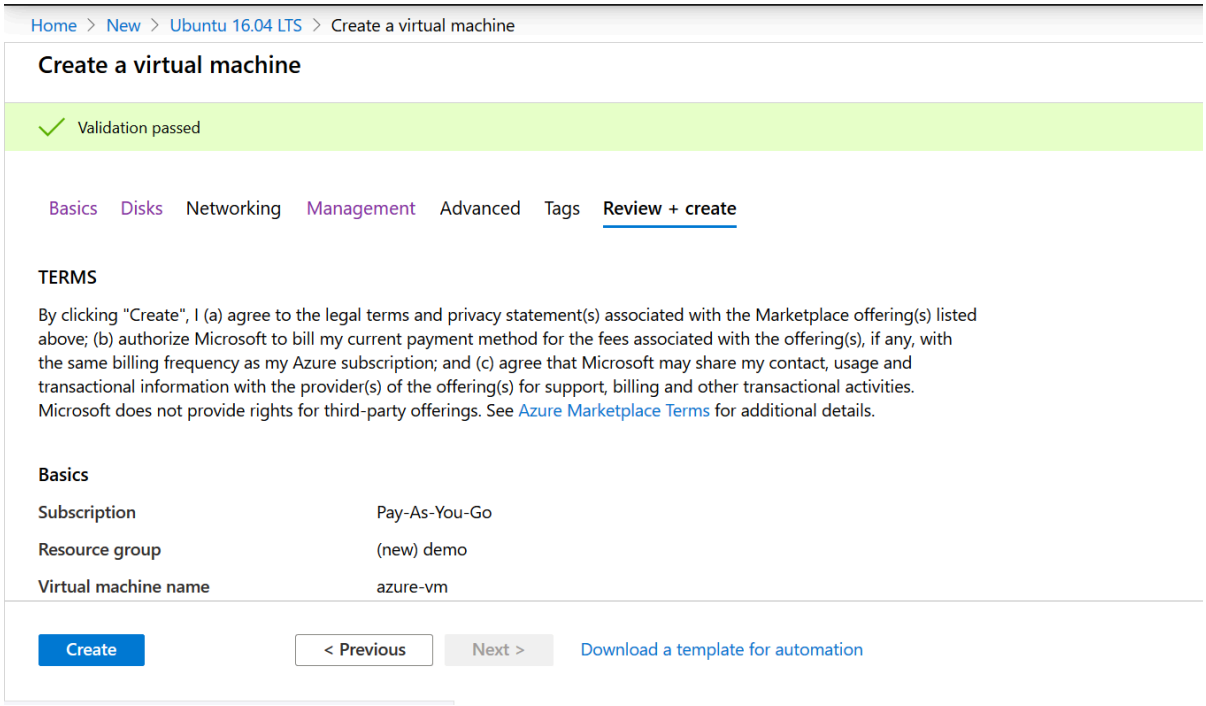
Add an inbound rule that allows any IP to access the VM.



Leave the remaining settings at their defaults and click **Review + create**.

Step 5 Review + create

Azure validates your configuration before creating the VM. After a few moments, you will see the **Validation Passed** message.



Click **Create**. Azure takes a short time to provision the virtual machine.

Step 6 Connect to the VM

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

Open bash or any other command line tool, then run the following command, replacing the IP address with your VM's public IP:

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

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

Enter your passphrase when prompted.

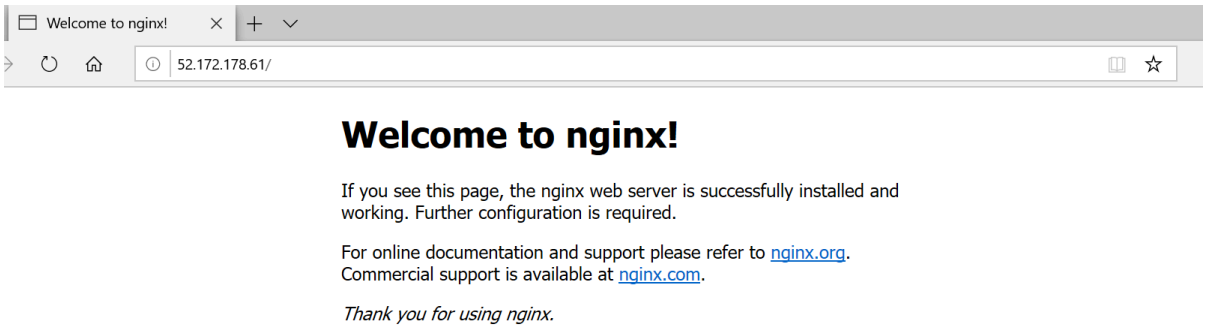
You are now connected to the VM.

Step 7 Test it.

To verify the VM is working, install the `nginx` web server.

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

Installation takes a few seconds. Once it completes, open your browser and enter the VM's IP address.



Step 8 Stop the VM

You can save money by stopping the VM when you are not using it. Go to the VM overview and stop it. Note that the public IP address will change each time you restart the VM.

*****

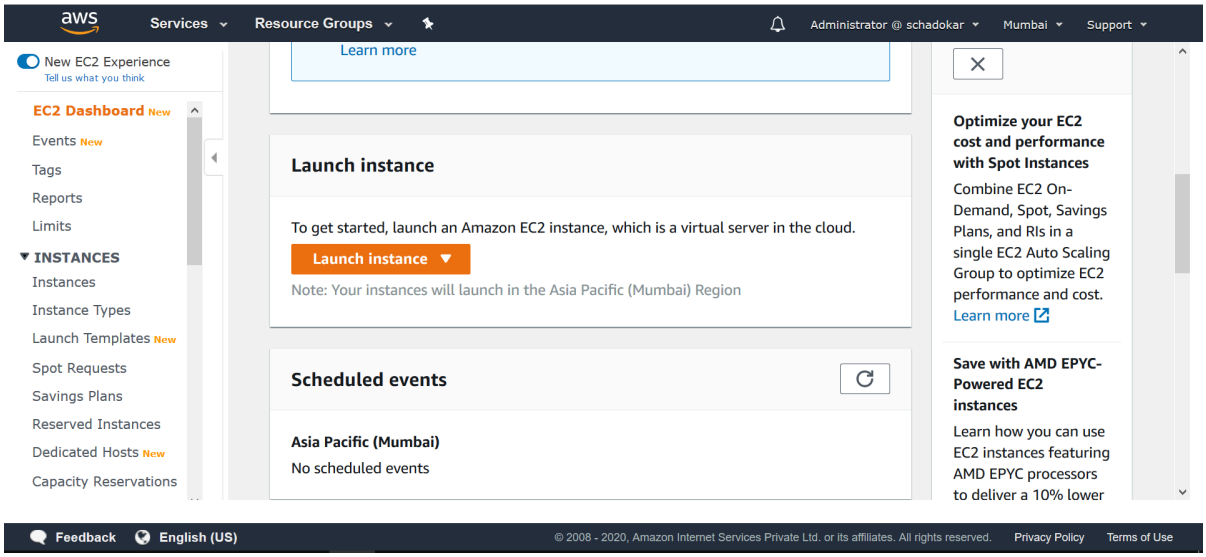
Create a Linux VM in AWS

Start by creating a new aws account.

Log in to your AWS account. Select **Services** in the top left, search for **EC2**, and select it.

EC2 Dashboard

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

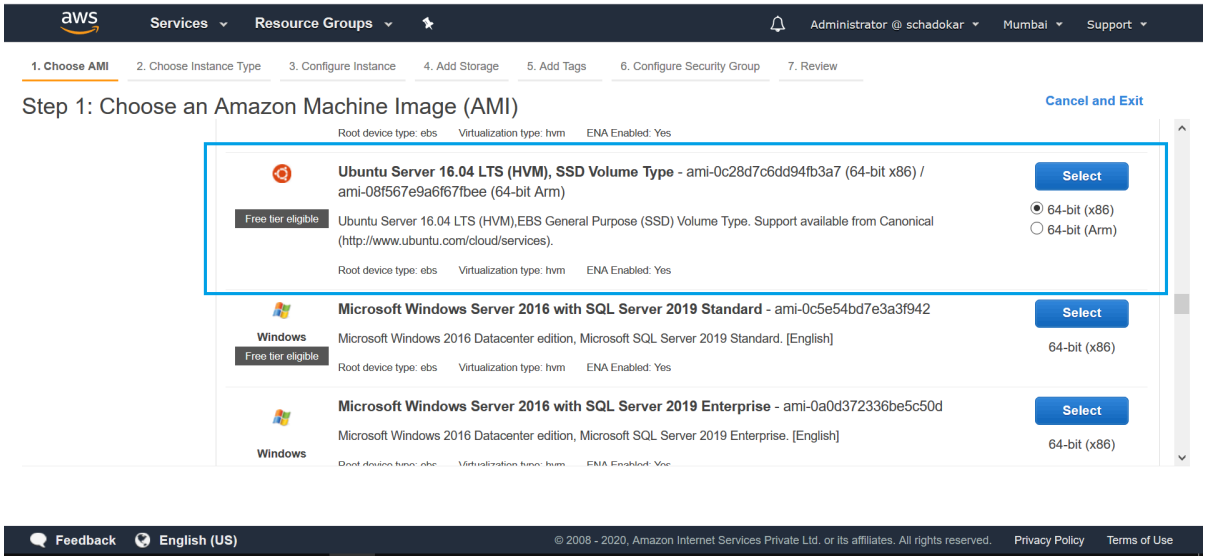


Configure Instance

Follow the steps below to create a Ubuntu 16.04 VM. Each step maps to a page in the AWS instance configuration wizard.

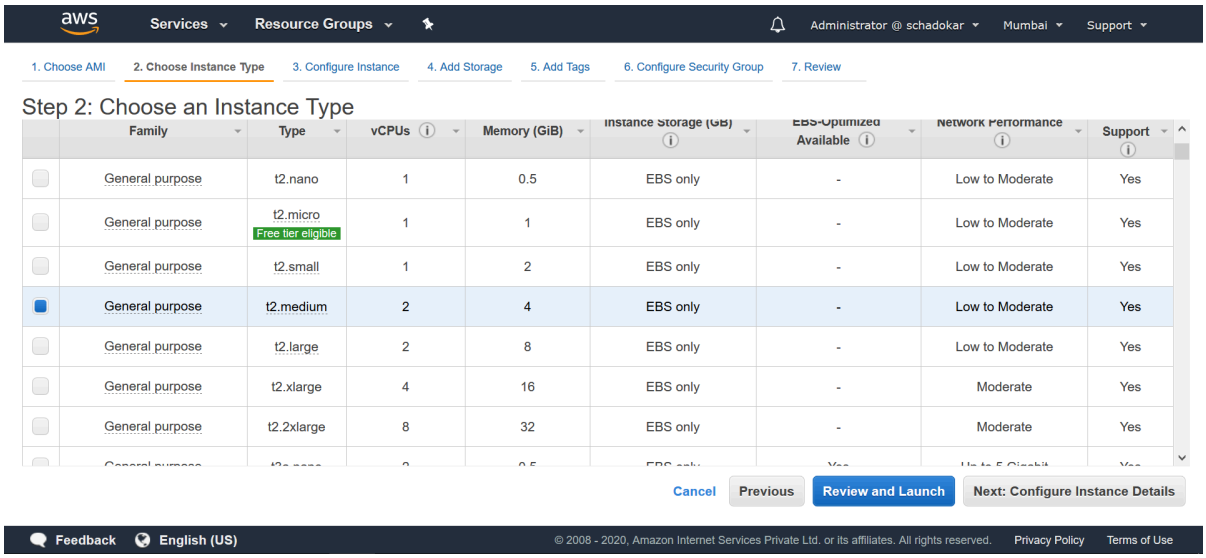
Step 1 Choose an Amazon Machine Image (AMI)

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



Step 2 Choose an Instance Type

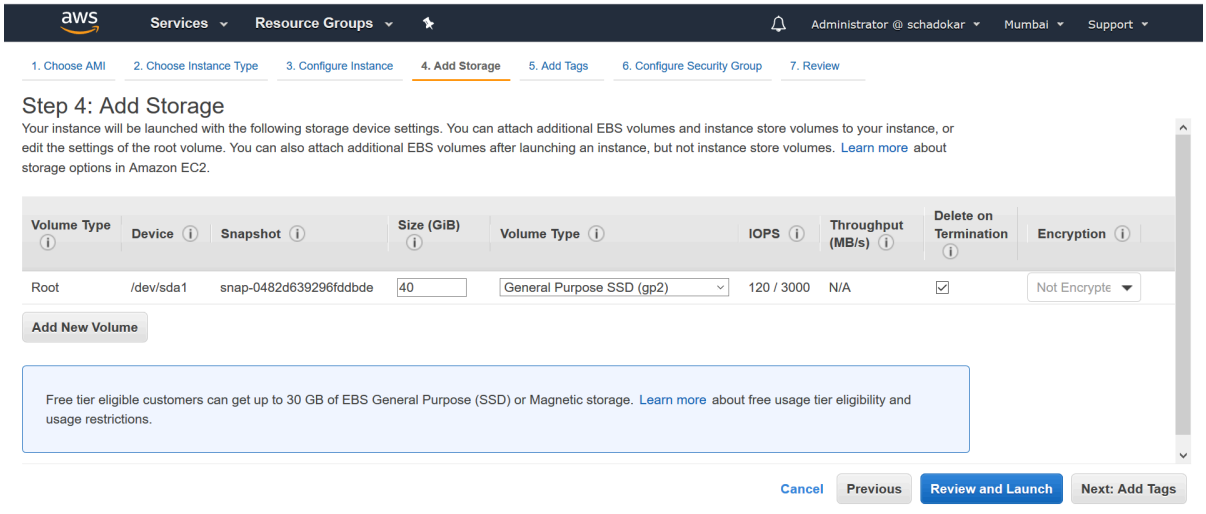
Select **t2.medium** and click **Configure Instance Details**.



Continue clicking through the next steps until you reach **Add Storage**.

Step 3 Add Storage

Change the storage size to 40GB.



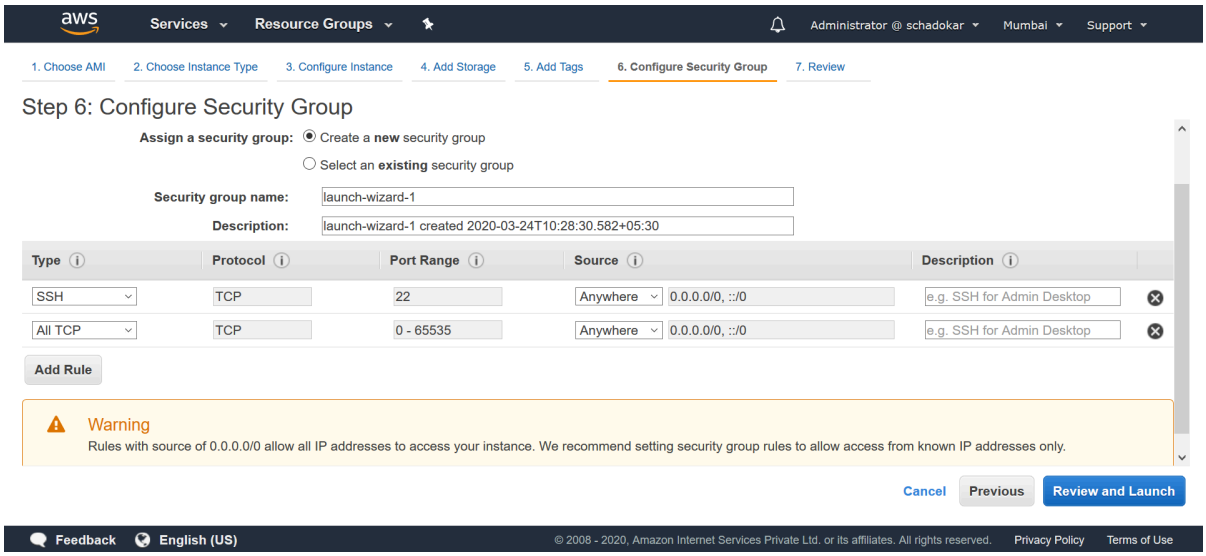
Continue clicking through the next steps until you reach **Configure Security Group**.

Step 4 Configure Security Group

A security group controls which traffic can reach your instance. For learning purposes, keep the configuration open and unrestricted.

Add 2 rules with the following settings:

- **SSH** — set Source to **Anywhere**
- **All TCP** — set Source to **Anywhere**



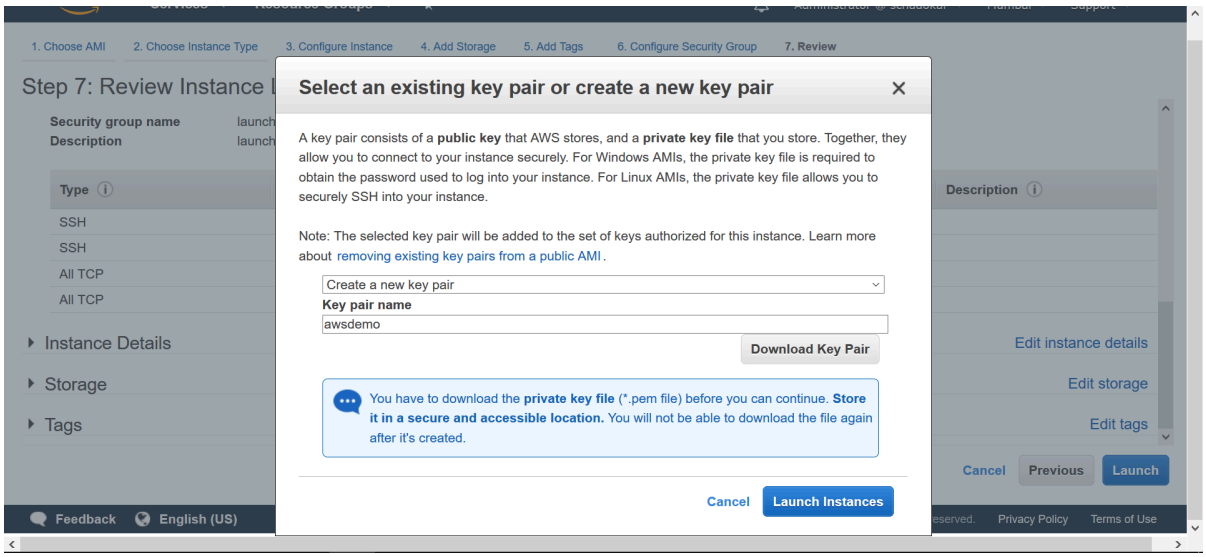
Select **Review and Launch**.

Step 5 Launch

Scroll down and select **Launch**.

When prompted, create a new key pair. AWS uses a public-private key pair for authentication: AWS stores the **public key**, and you download and keep the **private key**. You will need this private key every time you connect to the instance.

Give the key pair a name.

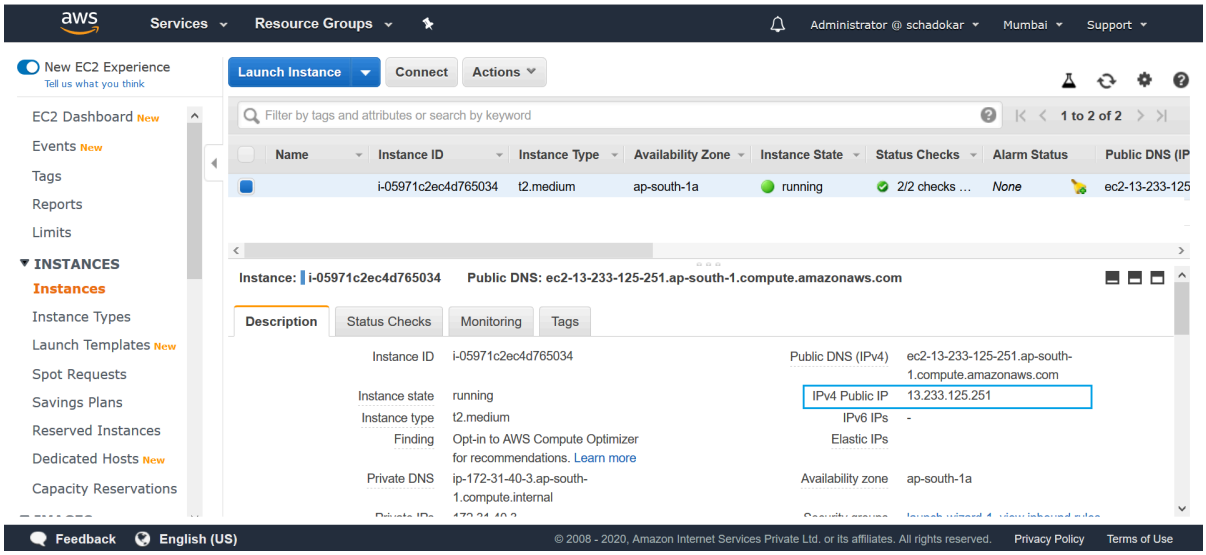


Select **Launch Instance**. AWS will take a moment to bring the instance online.

To check the instance status, either 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** changes to **running**. Copy the **IPv4 Public IP** shown in the instance details.



Open the command line.

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

Run the `ssh` command to connect to the VM:

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

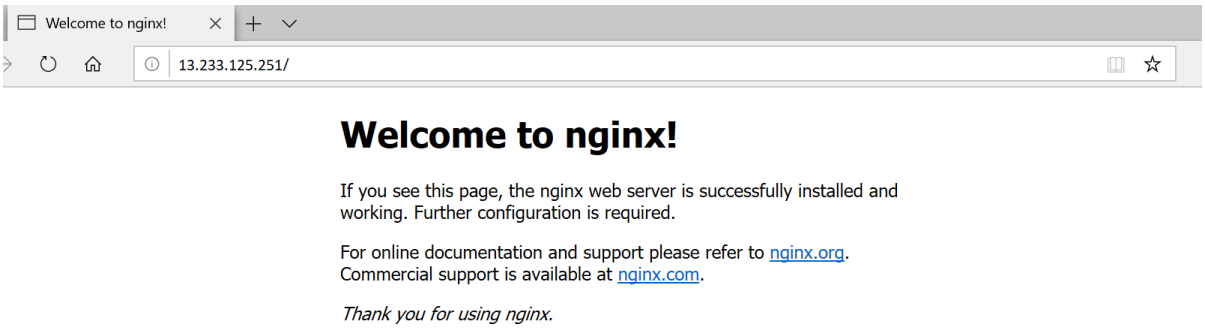
```
ssh -i awsdemo.pem ubuntu@13.233.125.251
```

Test the VM

To verify the VM is working, install the `nginx` web server by running the following commands:

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

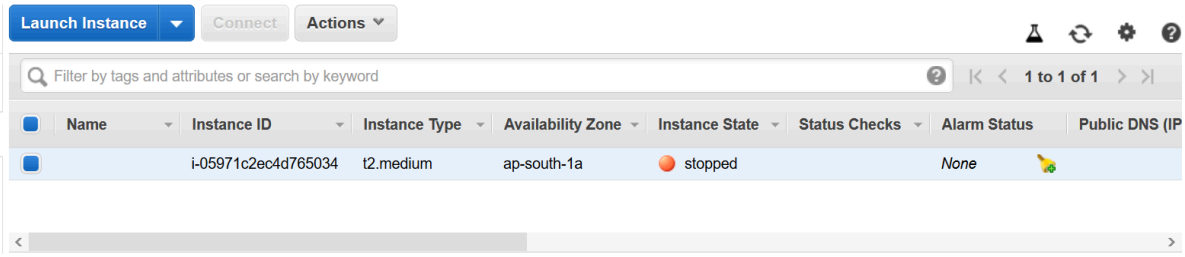
Installation takes a few seconds. Once it completes, open a browser and enter the VM's IP address — you should see the default nginx welcome page.



Stop the VM

You can stop the VM whenever you are not using it to avoid unnecessary charges. Note that the IP address changes each time you restart the instance.

> Go to Actions > Instance State > Stop.



*****

Create a Linux VM in Google Cloud Platform (GCP)

Start by creating a new account on GCP. New users receive $300 in credits valid for 12 months, along with access to many free services.

Login to your GCP account.

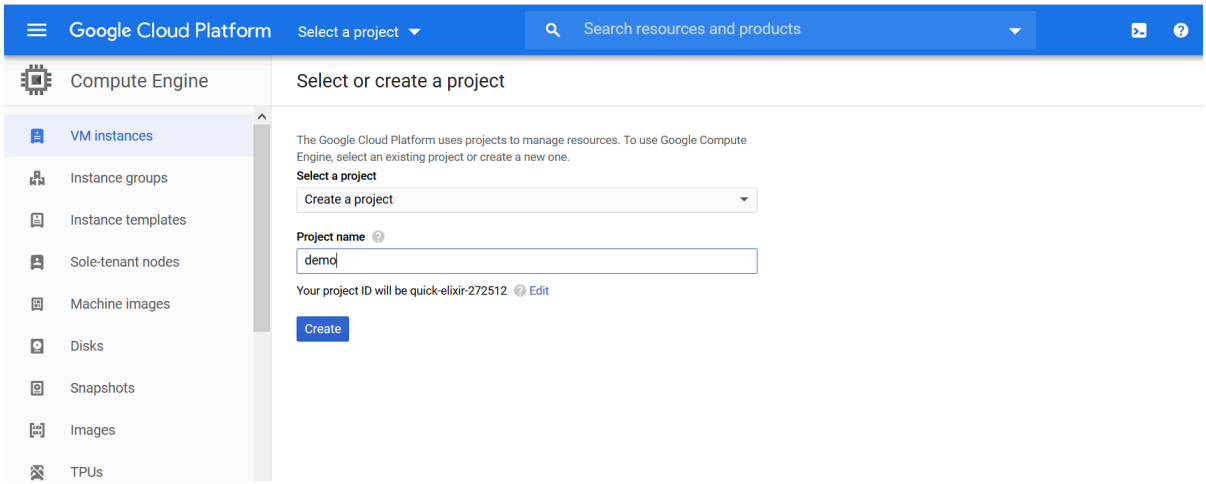
Create a new virtual machine

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

Menu > Compute > Compute Engine

Create a new project

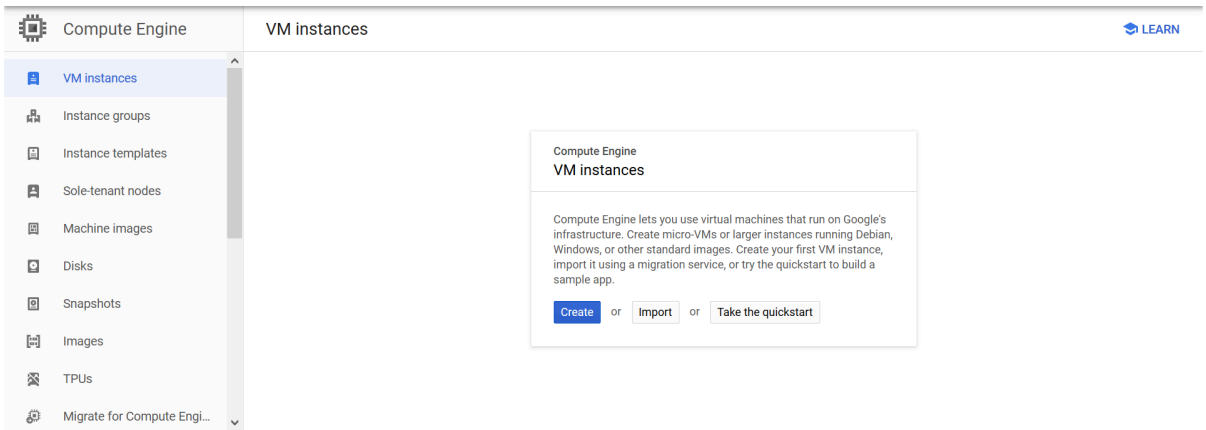
In GCP, a project acts as a resource manager — all related resources for a project live inside it. Create a new project named **demo**. The platform will take a moment to finish creating the project.



Create VM instance

Click on **Create**.

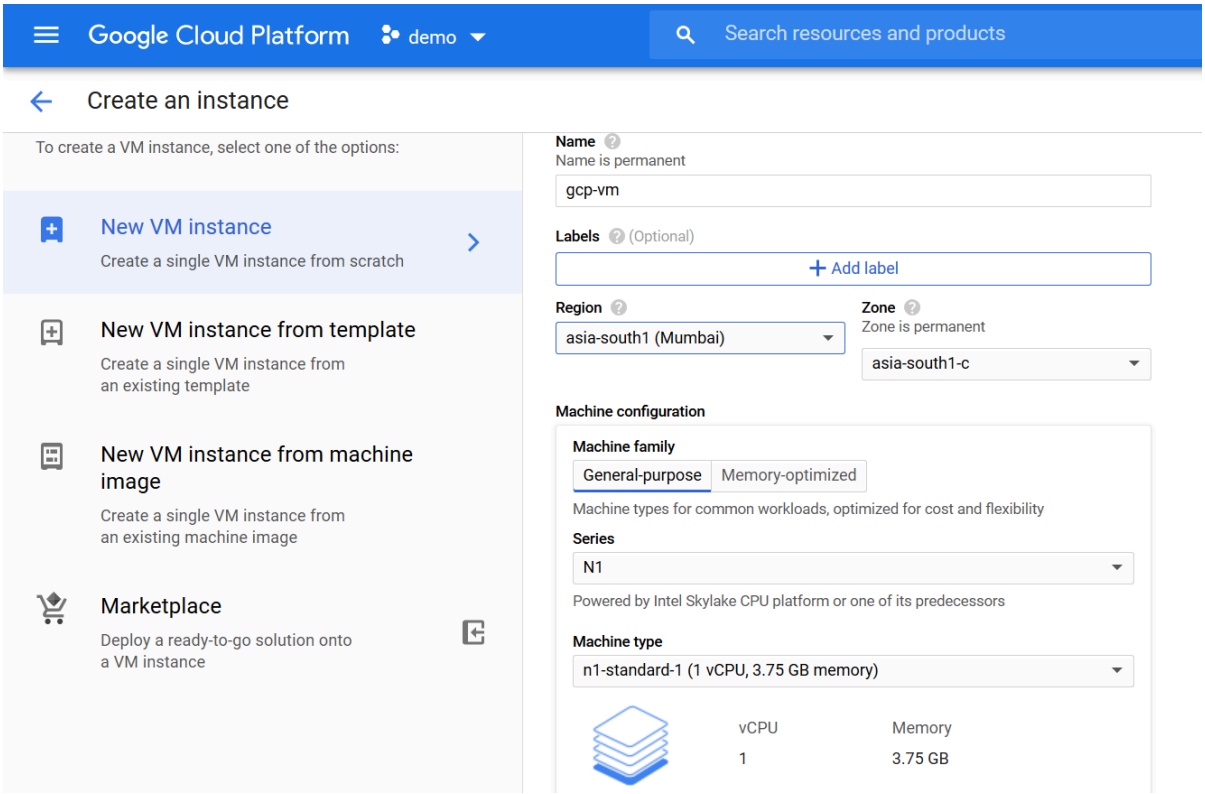
> **Take the quickstart** it is a small tutorial on how to create the VM.



Basic Details

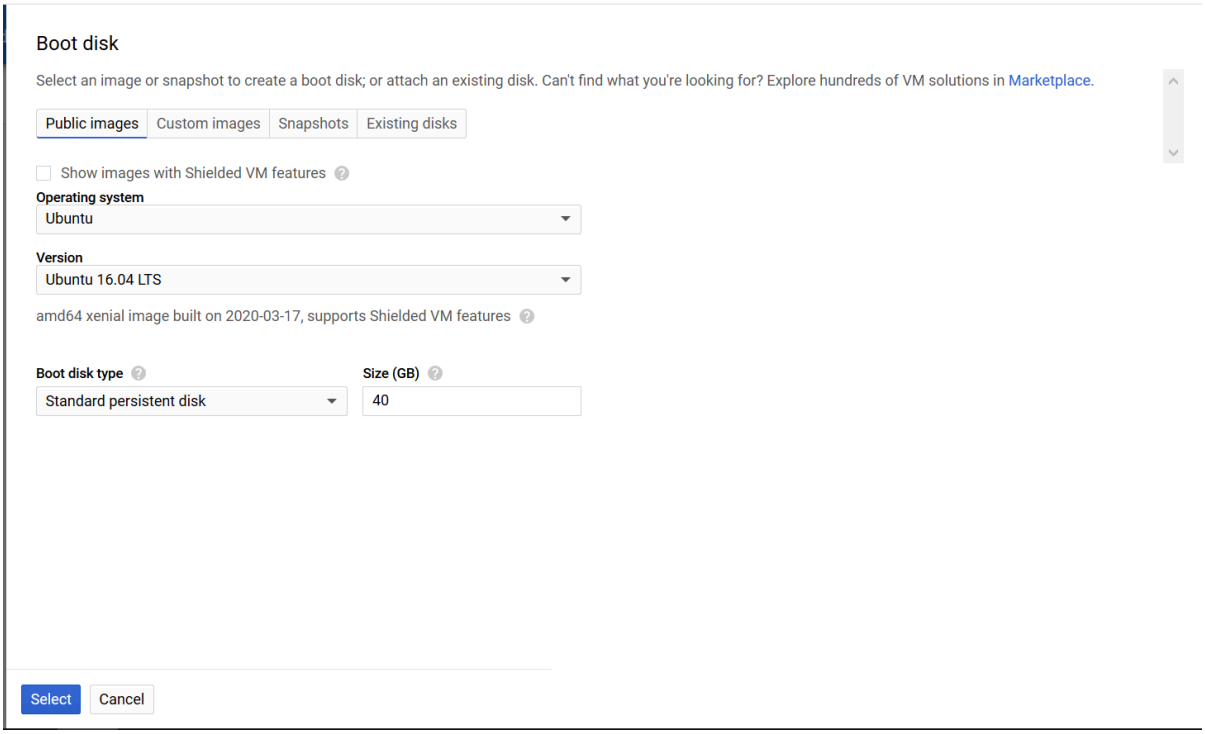
Fill in the following details to configure your virtual machine:

- 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** and 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, you can skip this step. If you skip it, GCP manages the private key on your behalf. You need this private key to remotely connect to the VM.

Click on **security**.

To configure SSH for remote command-line access to the VM, open a bash shell or any command line and run the `ssh-keygen` command to create keys. By default, it saves these keys in the `~/.ssh` directory.

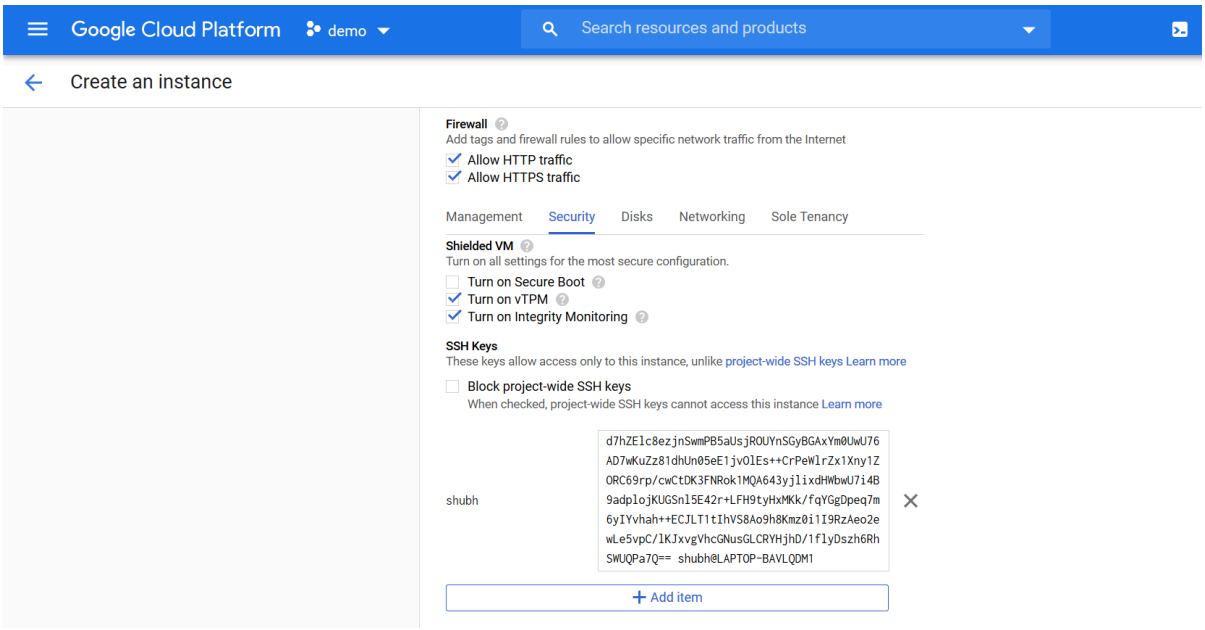
```
$ ssh-keygen -m PEM -t rsa -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/shubh/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/shubh/.ssh/id_rsa.
Your public key has been saved in /c/Users/shubh/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:5C+NIr1JjM8Oy5R6lgCyRm0EXqhHwToQQPYtN7f0qDY shubh@LAPTOP-BAVLQI
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 press Enter to accept the default.

A passphrase protects access to your private key when you log in to the VM. This step is optional, but if you set one, the passphrase must be at least 5 characters long. Note it down somewhere safe.

Open the `id_rsa.pub` file — this is your public key. Copy the key and paste it into **Enter public SSH key**.

> Remove any leading or trailing spaces while pasting.

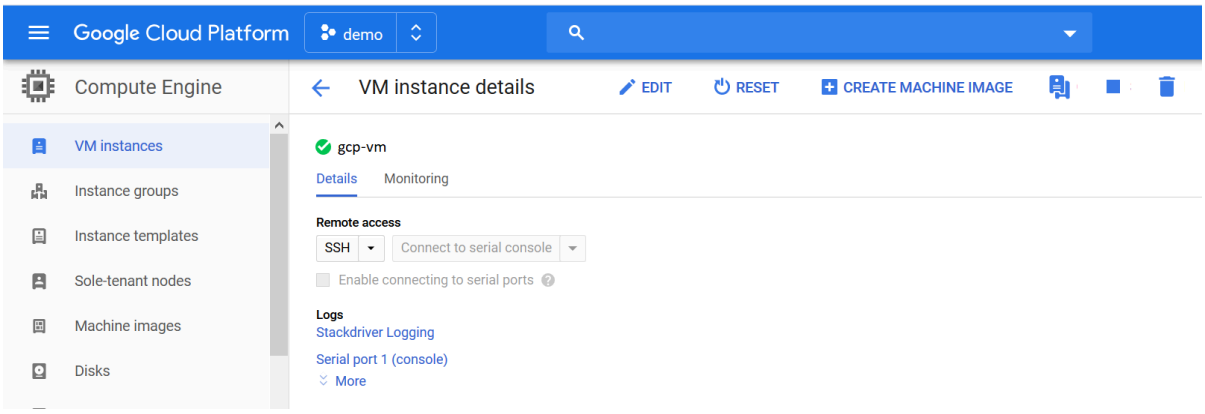


Click **Create**. The VM will take a moment to become ready.

Connect to the VM

Open the VM to view its details.

If you did not configure SSH, you can connect to the VM using the SSH option provided by GCP. Click on **SSH**.

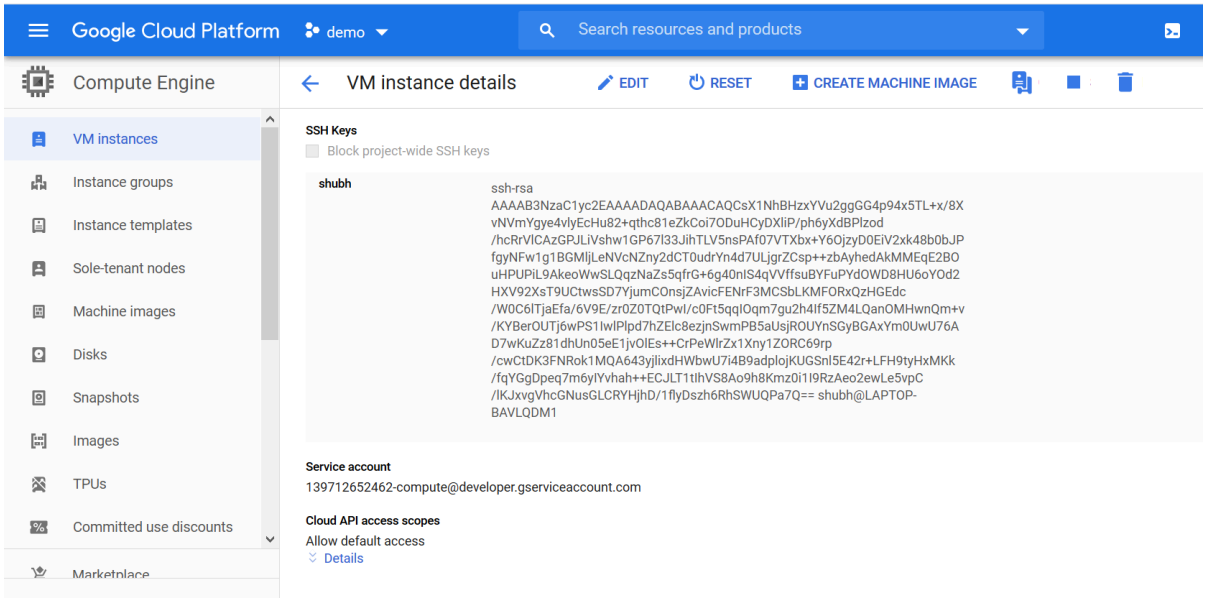


For SSH configured

Scroll down and copy the **External IP**, then open a bash shell or any command line and connect using the following format:

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

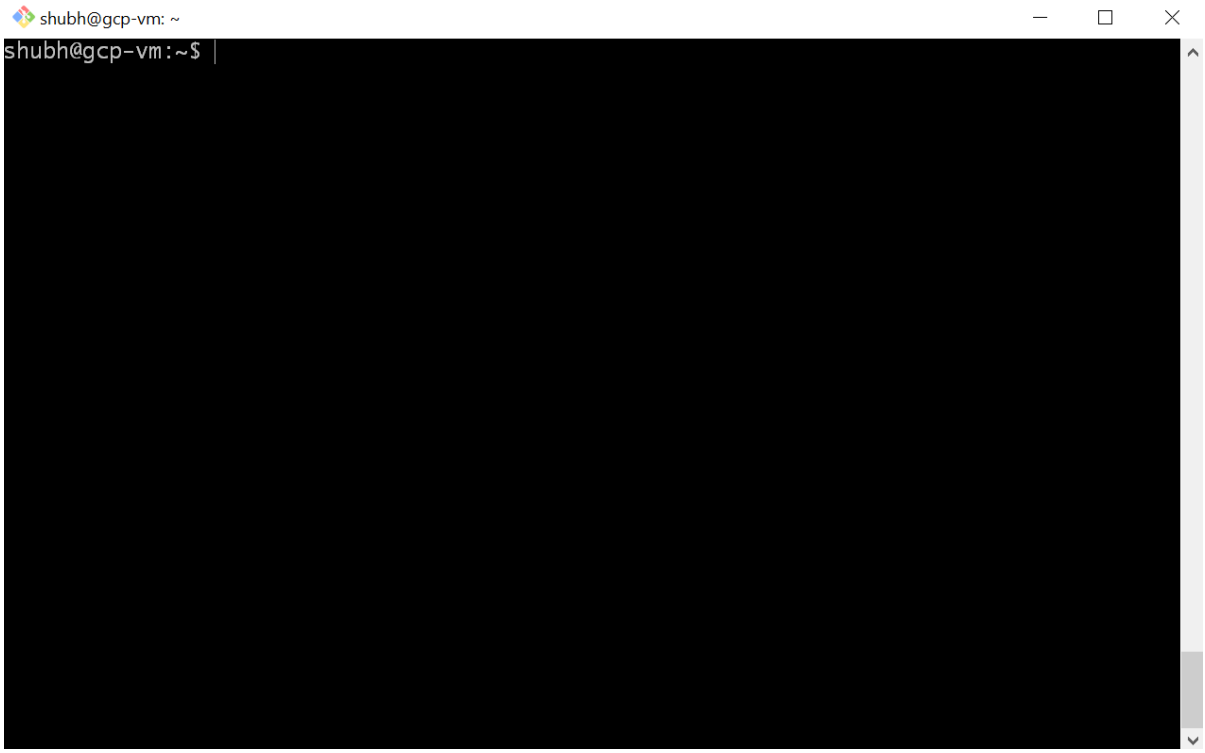
The username corresponds to the account associated with your SSH key.



For example:

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

Enter the passphrase if you set one during SSH key creation.

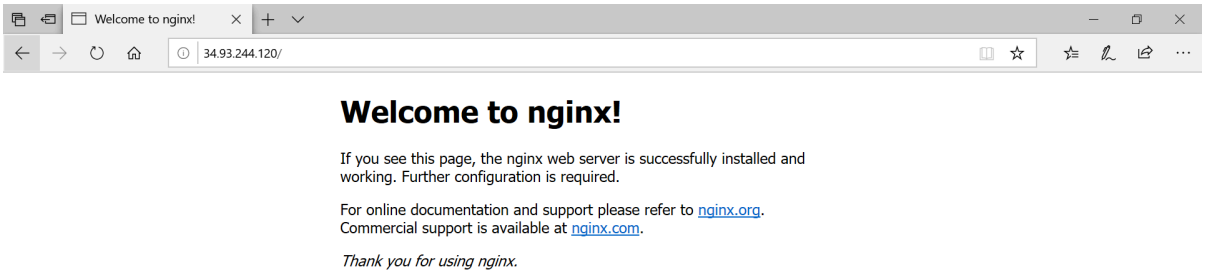


Test the VM

To verify the VM is running correctly, install the **nginx** web server with the following commands:

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

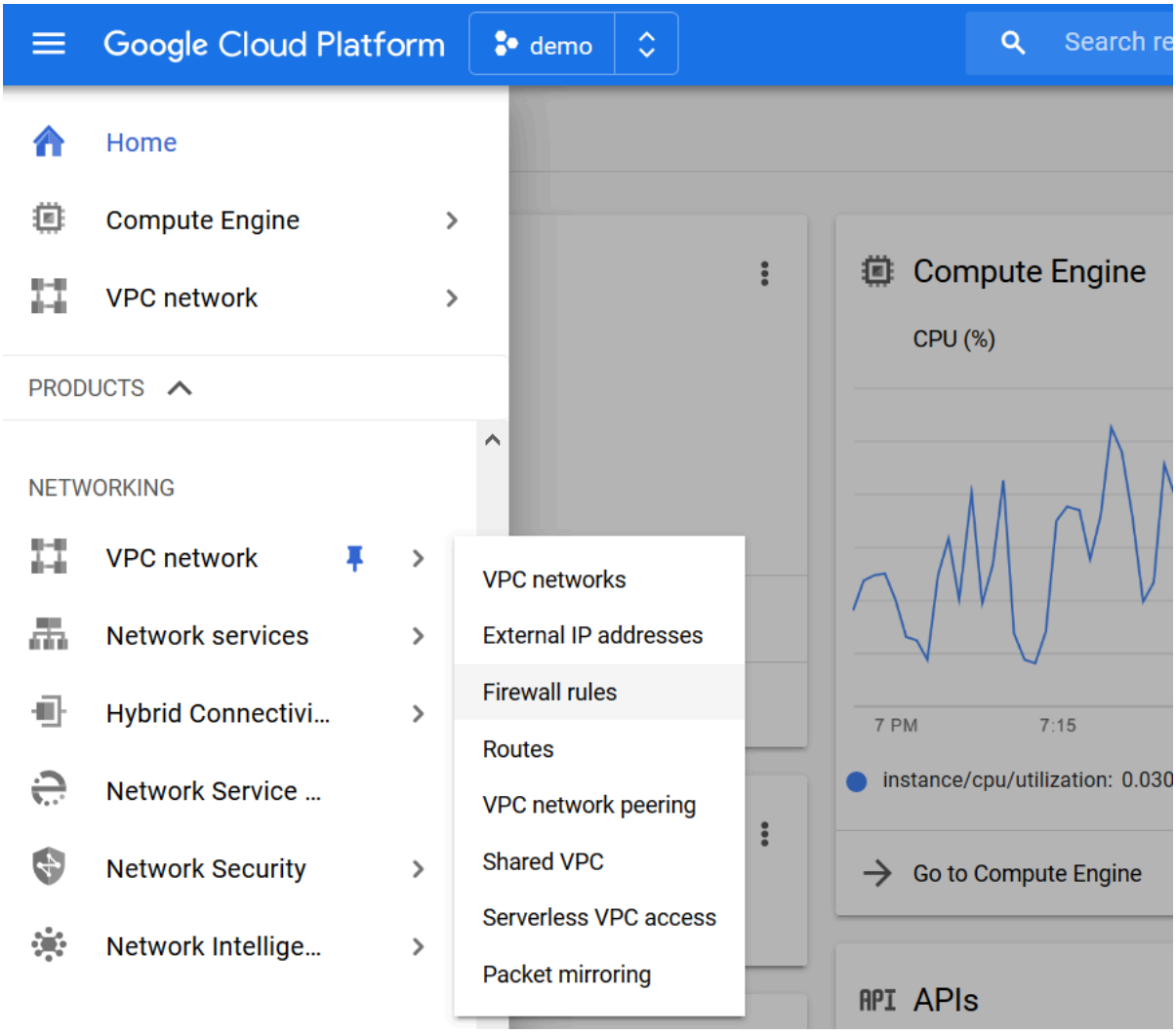
Installation takes a few seconds. Once it completes, open a browser and enter the VM IP address — you should see the nginx welcome page.



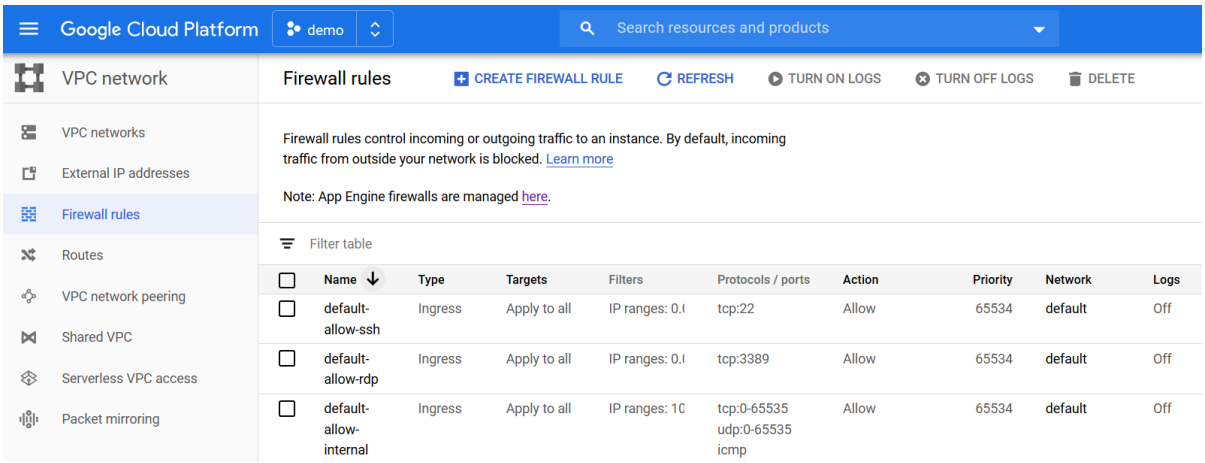
Allow Ports

By default, the firewall blocks all requests to ports. You must explicitly create a firewall rule to allow specific ports or all ports.

To allow all ports, open **Menu > VPC network > Firewall rules**.



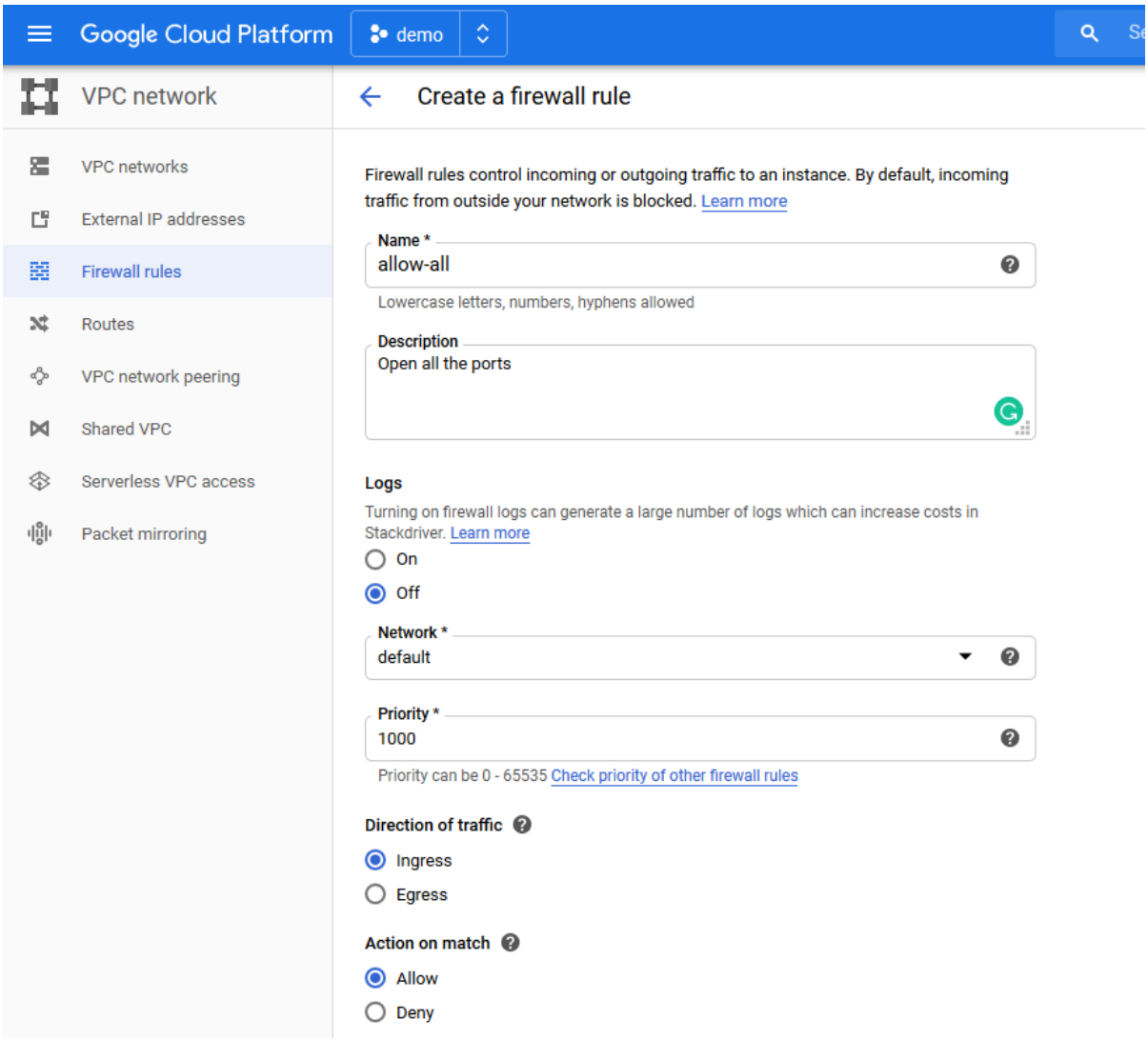
Click on **Create Firewall Rule** to begin configuring a new firewall.

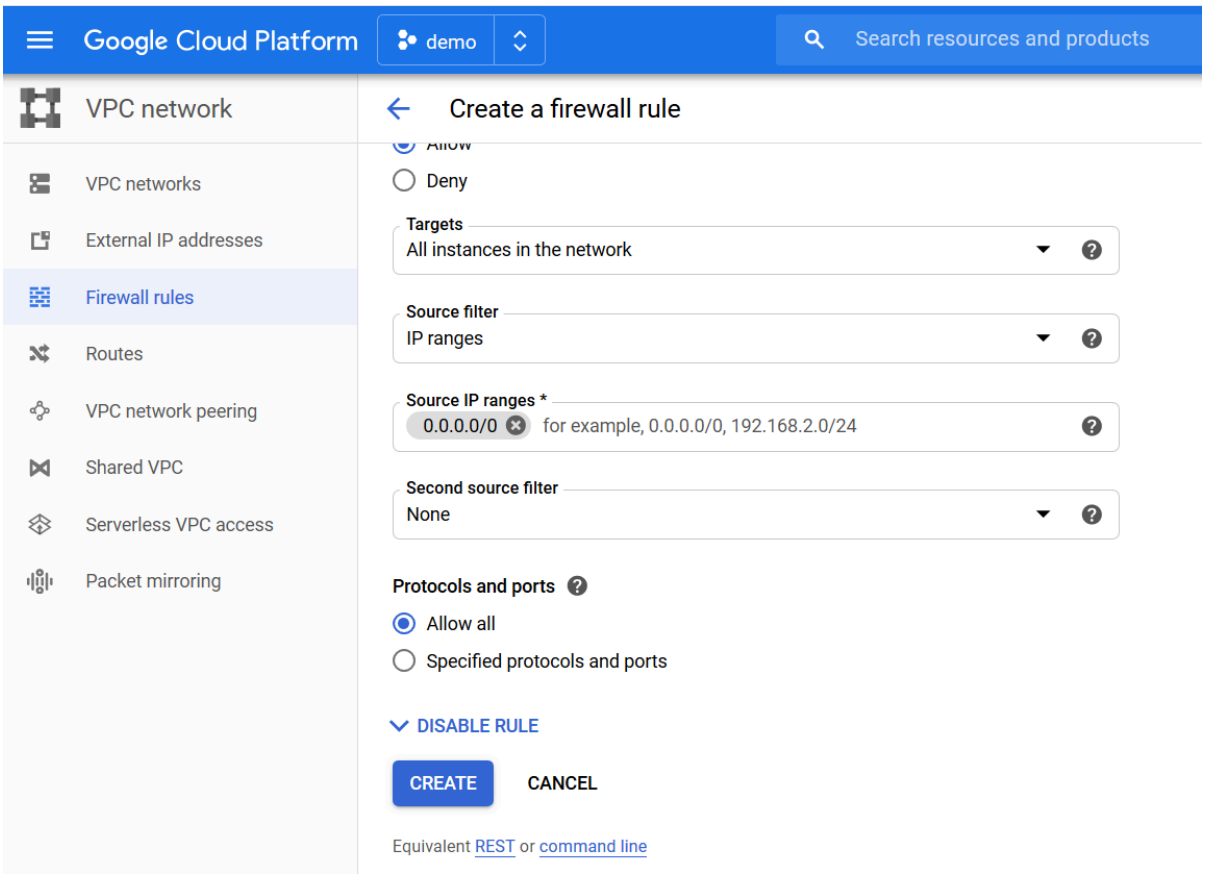


Configure the rule with these settings:

```
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 remaining settings as default.



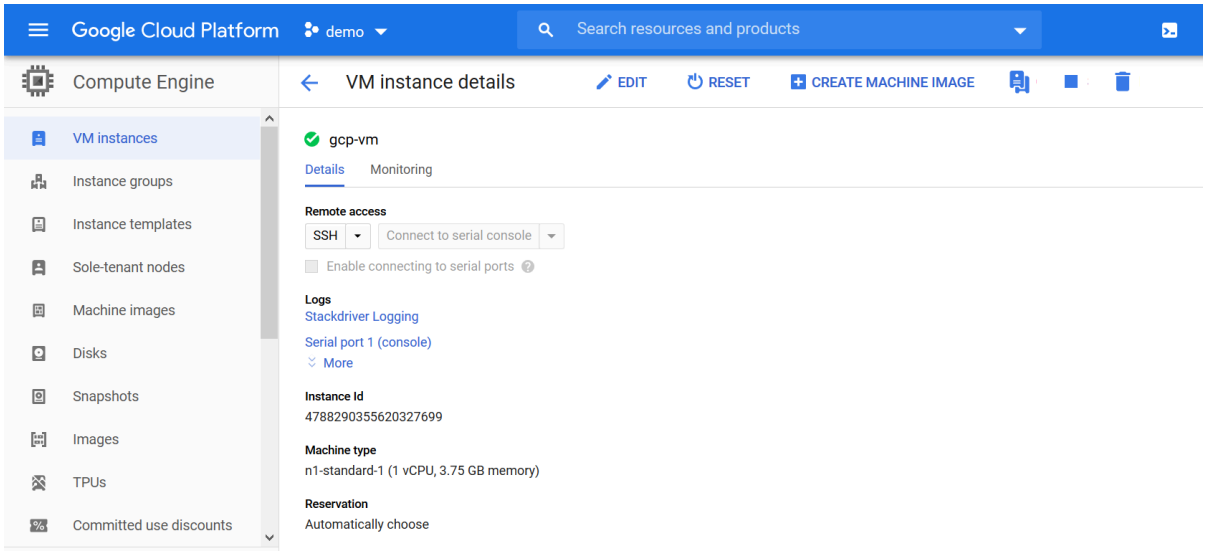


Click on **Create**. Your VM now accepts traffic on all ports.

Stop the VM

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

Use the **Stop** or **Delete** buttons at the top of the VM details page.



Note that the next time you start the VM, its public IP address will change.

*****

Hyperledger Composer Installation Instruction

These instructions apply to UNIX-based operating systems such as Linux and macOS.

Connect to your Linux VM. Before using Hyperledger Composer, you must install several pre-requisites.

Warnings for Linux Users

Do not install Hyperledger Fabric and Composer as the root user.

Instructions to create a new user

Open the command line or terminal.

Add a new user *composeruser*

You can use any name you prefer.

```
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 on managing users.

Pre-requisites

Hyperledger Composer provides a shell script that downloads and installs all the dependencies. Run the following commands to download and execute the script:

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

chmod u+x prereqs-ubuntu.sh

./prereqs-ubuntu.sh
```

Install Composer Components

With the pre-requisites in place, install each Composer component using npm. The following steps walk you through each required package.

1. Essential CLI tools:

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

2. Composer REST Server

The Hyperledger Composer REST Server generates a REST API from a deployed blockchain business network, which HTTP or REST clients can consume easily.

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

3. Install Composer Playground

The Hyperledger Composer Playground provides a user interface for configuring, deploying, and testing a business network.

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

4. Install Hyperledger Fabric

Hyperledger Fabric is an enterprise-grade permissioned distributed ledger platform that offers modularity and versatility for a broad set of industry use cases. Create the development server directory, then download and extract the Fabric tools:

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

curl -O https://raw.githubusercontent.com/hyperledger/composer-tools/r

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

Before starting the network, ensure any previous instance is fully stopped and torn down. The steps below handle cleanup and startup in sequence.

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 starts a Hyperledger Fabric network with a peer organisation, an orderer organisation, and a certificate authority. It also creates one channel and joins the peer organisation to that channel.

2. Create Peer Admin Card

A Business Network Card provides all of the information needed to connect to a blockchain business network. You must create a peer admin business network card before you can deploy a Hyperledger Composer business network to a Hyperledger Fabric instance.

```
./createPeerAdminCard.sh
```

3. Start Composer Playground

Run the following command to start the Composer Playground locally on the VM:

```
composer-playground
```

To access the playground in your web browser, use `ipaddress_of_vm:8080` instead of `http://localhost:8080`. Your environment is now fully set up and ready for use.

*****

Project Objective

This project builds a supply chain management system. Its objective is to track a package as it moves from one participant to another.

This is a simple project in which the ownership of the package keeps changing — from seller to shipper, and from shipper to buyer.

Project Structure

The network consists of three types of elements:

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

Project Flow

The package moves through the network in three stages:

When a package is created, the seller becomes its owner and its status is set to ***Package_Ready***. When the shipper picks up the package, its status changes to ***Package_Picked***. Once the package is delivered to the buyer, its status changes to ***Package_Delivered***.

Together, these three status transitions represent the complete lifecycle of a package through the supply chain.

*****

Project Setup in Composer

In this section, you will set up the Composer environment for the project.

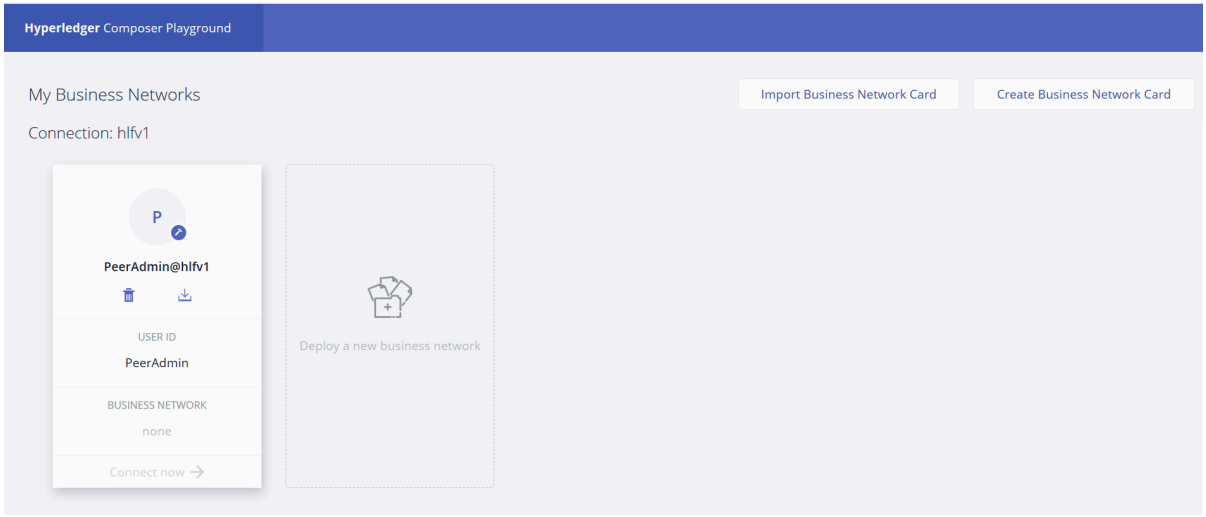
Open the terminal and start the Composer Playground.

```
composer-playground
```

This command starts `composer-playground` on `localhost:8080`.

> If you are using your Linux machine, open the browser and navigate to `localhost:8080`.

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

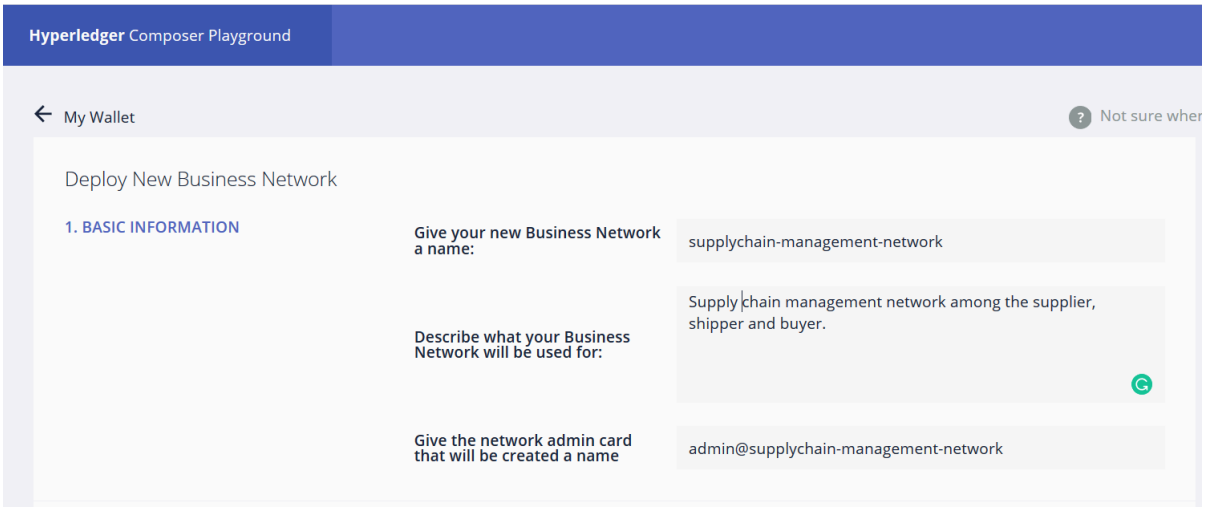


Click on **Deploy a new business network** to start a new business network. Composer provides many network templates to start with, but for this project you will build everything from scratch, so select the **empty business network**.

Enter the basic details of the network

Before deploying, provide the basic identity information for your network. Enter the business network name, description, and network admin card.

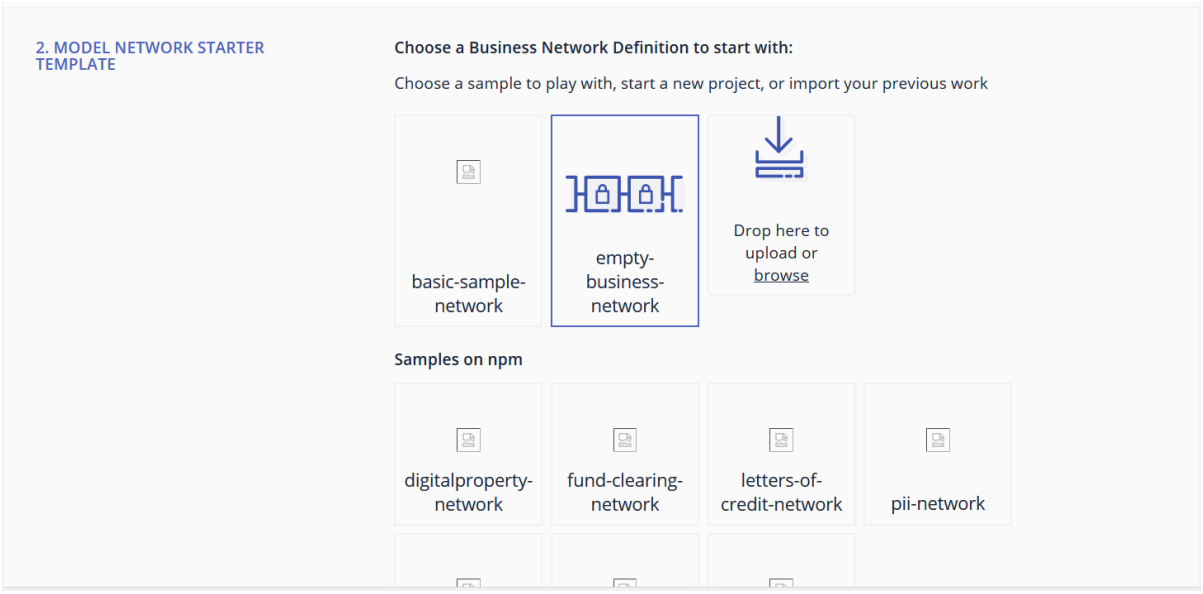
The network admin card lets you interact with the network — it acts as a key to the business network. If you leave the card name blank, Composer defaults to `admin@network-name`.



Select the empty business network

Select the empty business network. This network contains only the default ACL and Model file, giving you a clean foundation to build on.

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



Enter the credential for network administrator

Next, provide the credentials that Hyperledger Fabric requires to deploy the business network. Select **ID and Secret**, then enter the default **Enrollment ID** and **Enrollment Secret**.

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

3. CREDENTIALS FOR NETWORK ADMINISTRATOR

You must provide credentials in one of the following formats before you deploy this business network

The credentials will be used to access the business network once it is deployed

- ○ **Certificates**
  Required here are certificate and private key files.
- ◉ **ID and Secret**
  These can be created when accessing a business network.

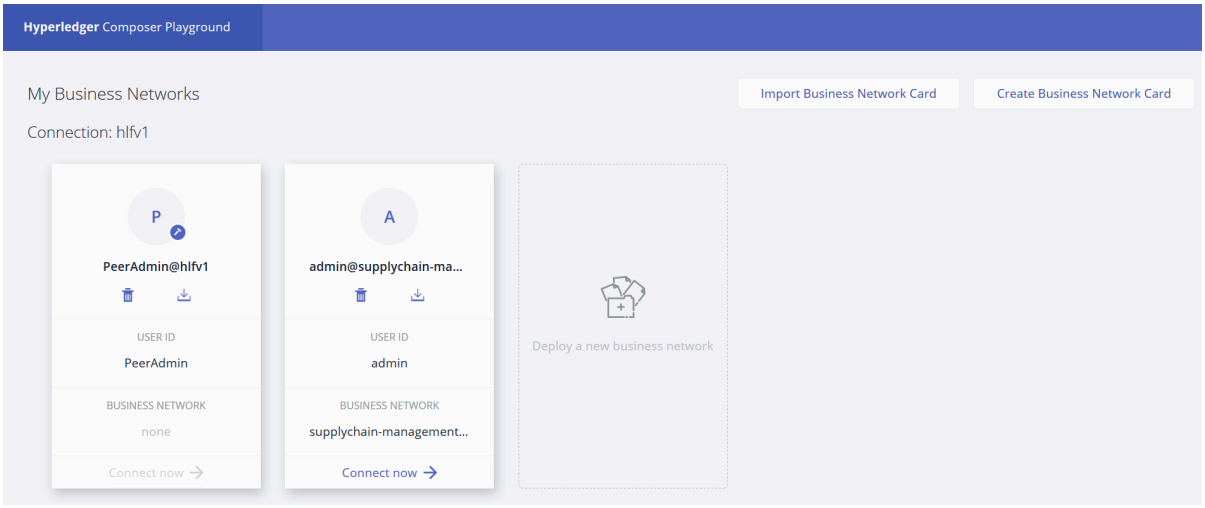
An Enrollment ID and Secret must be created by someone who already has access to the Business Network you are connecting to.

Enrollment ID	admin
Enrollment Secret	•••••••

Deploy the supplychain management business network

With all details filled in, click **Deploy**. The deployment may take a moment because Composer translates the business network into a Hyperledger Fabric-compatible smart contract, then deploys it as a Docker container on a channel.

> 💡 **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.



Once deployment finishes, you can connect to the network and inspect the default files. In the next sections, you will develop the supplychain management project.

*****

Model File

Hyperledger Composer includes an object-oriented modelling language that you use 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 a single domain reside in a single namespace. This way, different domains stay isolated and you can include them when needed.
2. asset, participant, transaction, event, enum and concept are resources.
3. Optional import declarations that import resources from other namespaces.

The following sections cover the basics of the modelling language.

Data Types

The modelling language supports these primitive data types:

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

Array

You can declare all primitive types as an array.

Syntax

```
Integer[] integerArray
```

Asset

An asset is an object on which different participants can perform transactions.

For example, consider a car as an asset. A seller owns the car. A buyer can purchase the car from the seller. The participant Buyer executes 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. A network can have multiple members, each playing a distinct role.

For example, in a supply chain network, the members include a manufacturer, supplier, seller, and buyer. Each of these is a different participant with a specific role.

Syntax

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

Transaction

A transaction is an action on an asset, a participant, or both. Transactions change the state of an asset, a participant, or both.

For example: buying a car, or changing a participant's name.

Syntax

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

Relationships

A relationship is a reference to a particular resource. You represent a relationship using `-->`. Relationships are unidirectional.

For example, to represent the relationship between a user and a vehicle where the vehicle has an owner:

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

Event

Events act as notifications: when a particular thing happens, all subscribers receive a message. When a specific scenario occurs, the system publishes an event, and every subscriber to that scenario receives it.

For example, whenever the state of an asset changes, the system publishes an event. In an e-commerce application, when a user places an order, the system triggers an **order placed** event. When the order ships, it triggers an **order dispatched** event. You can publish and subscribe to all of these activities as events.

Enum

An enum defines a category with **N** possible values. Using an enum restricts a field to only those defined values.

For example, `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. An asset, participant, or transaction typically contains them.

Concepts act as shared static data fields across different assets, participants, or transactions. By using a concept, you avoid rewriting the same fields in multiple places.

For example, the code below defines an abstract concept `Address` and then specializes it into a `UnitedStatesAddress`. Note that concepts do not have an `identified by` field because you cannot store them directly in registries or reference them 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 fields in an asset and participant are mandatory. Use the `optional` keyword to mark a field as non-required.

Syntax

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

Default

Any field in an asset or participant can carry an optional default value.

Syntax

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

Project Code

This section presents the complete model file for the supply chain project. The code namespace is `org.supplychain.management`.

- **Package** is an asset. Each asset holds a reference to a member as its owner.
- **Member** is a participant. Every member has a role.
- **Role** is an enum. Members can hold one of three roles: **Seller**, **Buyer**, or **Shipper**.
- **Status** is an enum representing the three states a package can have: **Package_Ready**, **Package_Picked**, and **Package_Delivered**.
- **transferOwnership** is a transaction. It transfers ownership of the package from one member to another.

```
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, you define the functionality of the transactions declared in the model file.

You write transaction definitions 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.

Every transaction function must have a transaction processor structure defined directly above it.

Below the comments sits the JavaScript function that 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
 * @transaction
 */

async function transactionFunctionName(parameters) {}
```

The structure ties the decorator metadata to the JavaScript function, enabling Hyperledger Composer to route transactions correctly.

Transaction Processor Function

Hyperledger Composer manages an individual registry for each resource in the blockchain network. To manage resources, Hyperledger Composer provides different registry APIs. Inside the transaction processor function, you can use these APIs to perform CRUD operations on 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

Together, these registries and methods give you full control over the state of your blockchain network within any transaction.

Project Code

The script file contains a single transaction processor function. You define the ***Transaction Processor Structure*** first, then the ***Transaction Processor Function*** below it.

In the structure, `@param {org.supplychain.management.transferOwnership}` breaks down as follows: `org.supplychain.management` is the namespace and `transferOwnership` is the transaction defined in the model file. `ownershipTx` is the

parameter passed into the transaction function. The parameter is a JavaScript object, and you access its properties using the `.` dot operator.

For example, to access the `packageid` field, use `ownershipTx.packageid`.

The Registry APIs are asynchronous and return promises. `async-await` is the approach used here to handle those promises cleanly.

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

There are two ways to pass parameters to a transaction — as a value or as a relationship. The `transferOwnership` transaction uses both to illustrate the difference:

- `packageid` is passed as a value.
- `to` is passed as a relationship.

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

To make any changes to the package identified by `packageid`, you first need to resolve the package relationship. `Package` is an asset, and `packageid` is a plain string with no direct relationship to the `Package` asset, so you must retrieve it explicitly.

Start by getting the Package asset registry using the `getAssetRegistry` API.

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

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

`packageRegistry` now holds the registry of all Package assets. To retrieve a specific package, use the `get` method.

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

Next, update the `package` owner and `status`. Review the `Package` asset definition to understand its fields.

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

`owner` is declared as a relationship, so it requires a resolved relationship to update it. The `to` argument in the transaction is already a relationship — Composer resolves it automatically, so no additional resolution step is needed.

> 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. Update its `owner` and `status` using the `.` dot operator.

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

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

To persist the changes back to the `packageRegistry`, call the `update` method.

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

The changes now reflect in the blockchain.

Here is the complete script file with everything assembled.

```
/**
 * A transaction processor function description
 * @param {org.supplychain.management.transferOwnership} ownershipTx
 * @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 provides 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 supports simple queries using `SELECT` and `WHERE` statements. Every query must have a description and a statement.

Syntax

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

description

The `description` property defines what the query does.

statement

The `statement` property contains the query logic. The following operators are available:

- `SELECT`: retrieves a resource — an asset or a participant. Provide the resource name as `namespace.resource`.
- `FROM`: an optional operator that defines a different registry to query.
- `WHERE`: an optional operator that defines the conditions to apply to the registry data.
- `AND`: an optional operator that defines additional conditions.
- `OR`: an optional operator that defines alternative conditions.
- `CONTAINS`: an optional operator that defines conditions for array values.
- `ORDER BY`: an optional operator that defines the sorting of results. Use `ASC` for ascending and `DESC` for descending.
- `_$variable`: use the `_$` prefix to pass variables into the query.

The following example shows a query that combines multiple operators to filter and sort employees.

Example

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

Project Code

The project defines 3 queries to support the supply chain application:

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

The following code shows all three queries as they appear in the `.qry` file.

```
query QueryMembersByRole{
  description: "Get all the members by their roles. Seller, Buyer and
    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_
    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)
}
```

With the query file in place, the project can retrieve members and packages by role, status, or owner using parameterized queries.

*****

ACL File

Hyperledger Composer provides an access control language that lets you define rules and permissions on the resources described in the Model File. By defining ACL rules for participants, you can restrict access to resources for Create, Read, Update, and Delete operations. For example, you can restrict Create operations to one participant while granting only Read access to all other participants 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

Each ACL rule follows a consistent structure. The syntax below shows all available fields:

```
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

The access control file ships with default network rules already in place. You do not need to add anything.

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

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

These two rules together grant the `NetworkAdmin` participant full access to all user and system resources in the business network.

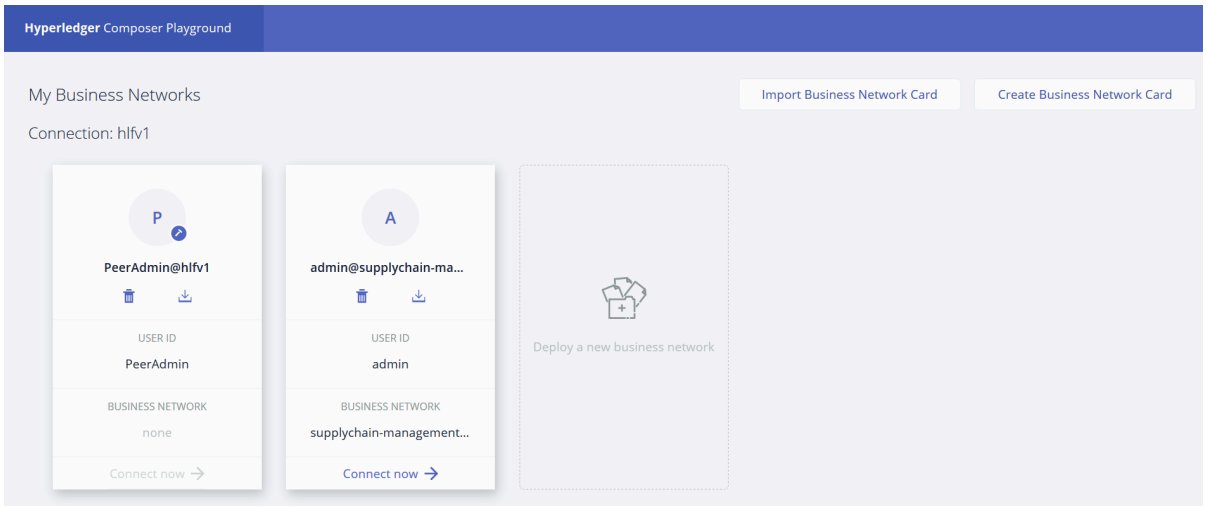
*****

Deployment in Composer Playground

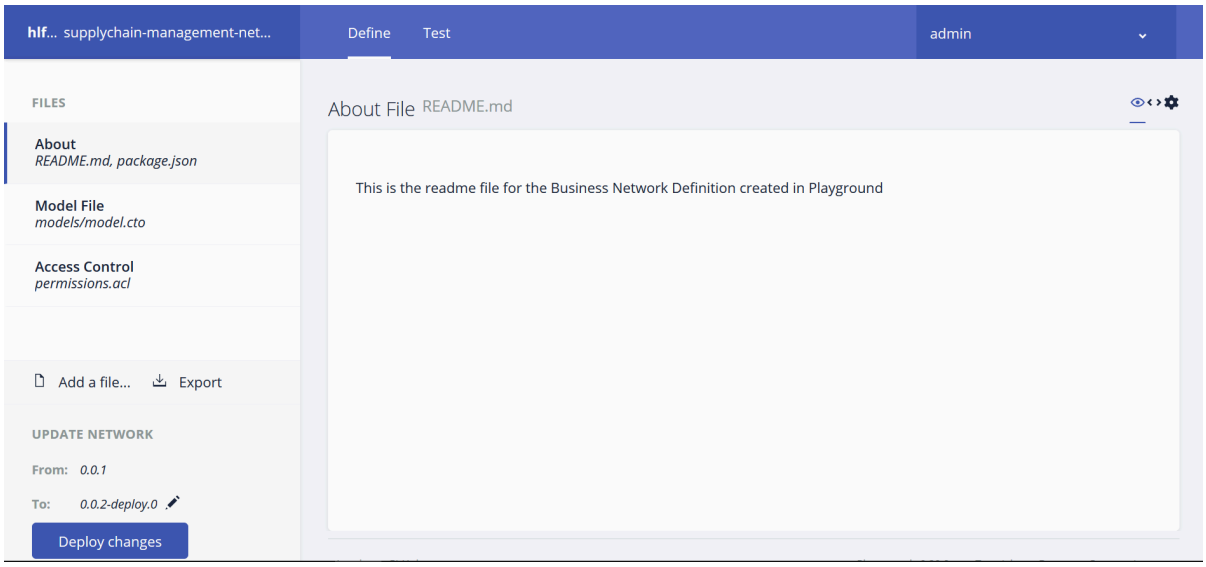
In this section, you will deploy the supplychain management project and integrate the business network logic.

Open the composer playground in the browser.

You 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 a 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, but only modify the author and description fields if necessary.

> ⚠️ **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,
  "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  --source
    "pretest": "npm run lint",
    "test": "mocha --recursive",
    "start": "start-network"
  },
  "dependencies": {
    "composer-common": "0.20.9",
    "composer-runtime-hlfv1": "0.20.9"
  }
}
```

The About file gives collaborators context about your network's purpose and configuration before they connect to it.

Model File

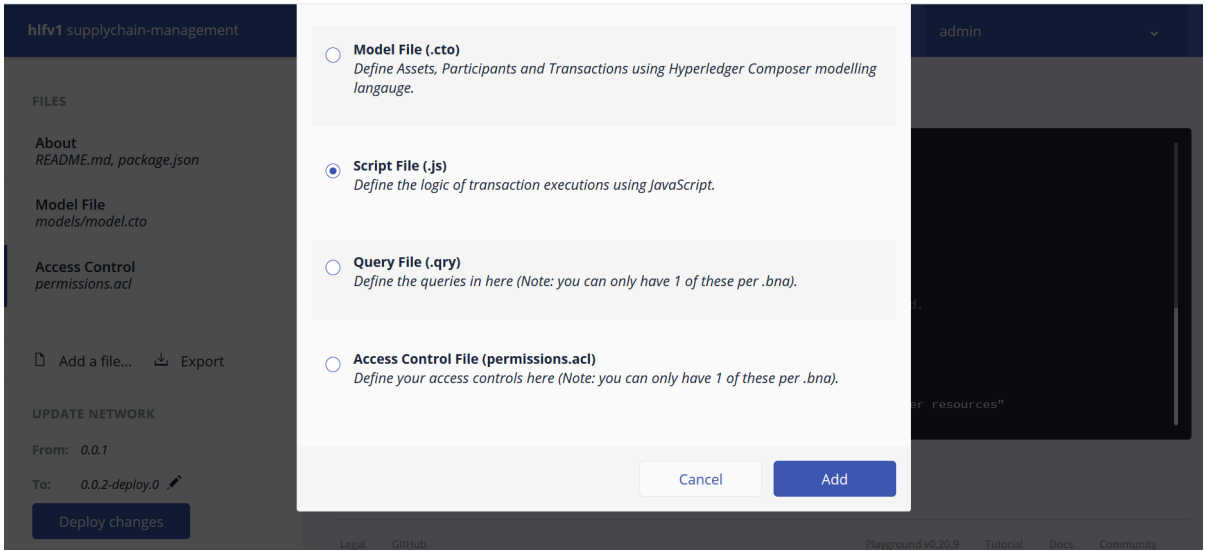
The Model file defines the assets, participants, and transactions that make up your business network. Open the Model file `.cto`, then copy and paste the code below. This is the same code covered in the **Model File** section.

```
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

The Script file contains the transaction processor functions that run when a transaction executes. To add it, follow these steps:

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



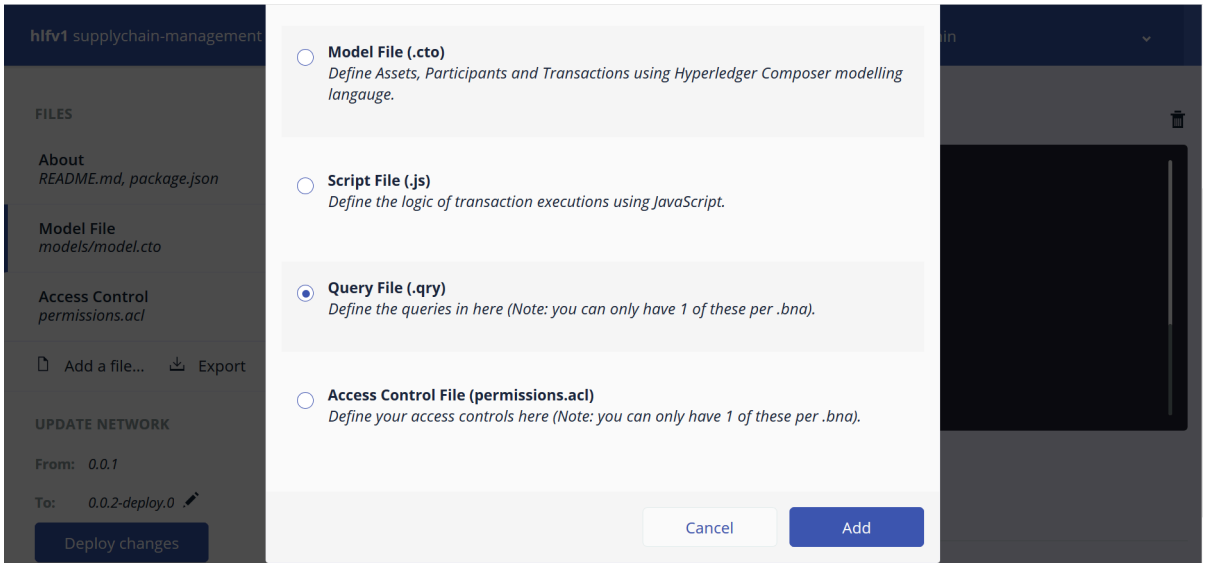
Copy the code below and paste it into the file.

```
/**
 * A transaction processor function description
 * @param {org.supplychain.management.transferOwnership} ownershipTx
 * @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

The Query file defines named queries that retrieve assets and participants based on specific criteria. To add it, follow these steps:

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



Open the **Query File**, then copy and paste the code below into it.

```
query QueryMembersByRole{
  description: "Get all the members by their roles. Seller, Buyer and
    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_
    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

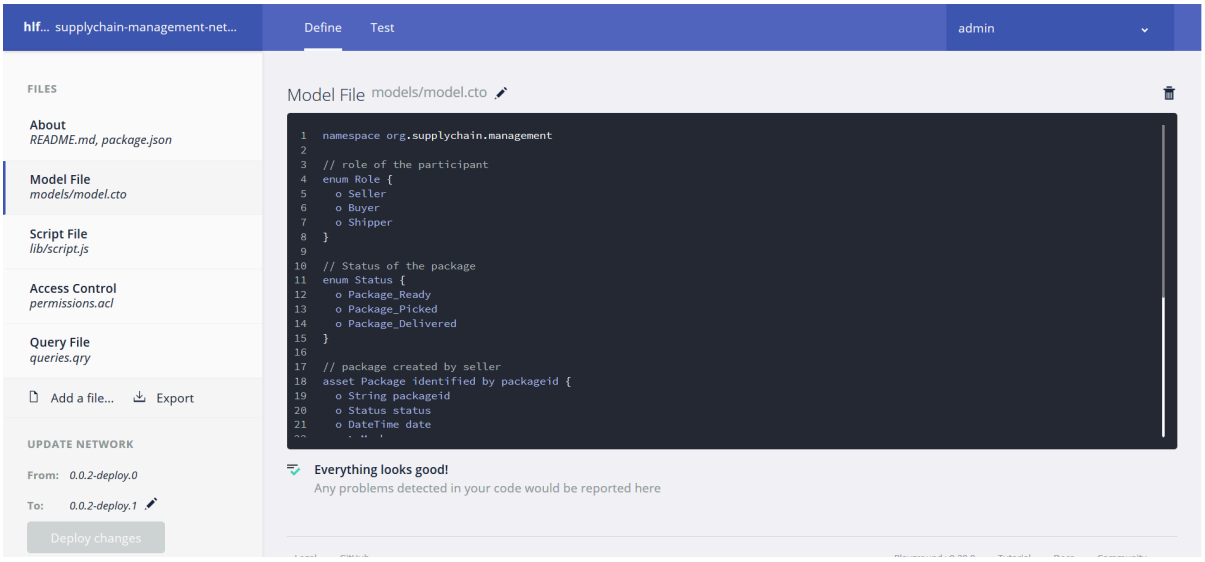
The Access Control file defines the permissions that govern what each participant can do within the network. Open the **Access Control** file and copy and paste the code below if it is empty.

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

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

Deploy

With all files in place, you are ready to deploy the business network to Hyperledger Fabric. Click on **Deploy Changes**. When prompted, click **Upgrade** to confirm.

Composer deploys the business network on the Hyperledger Fabric blockchain as a Smart Contract. The instantiation takes a moment to complete.



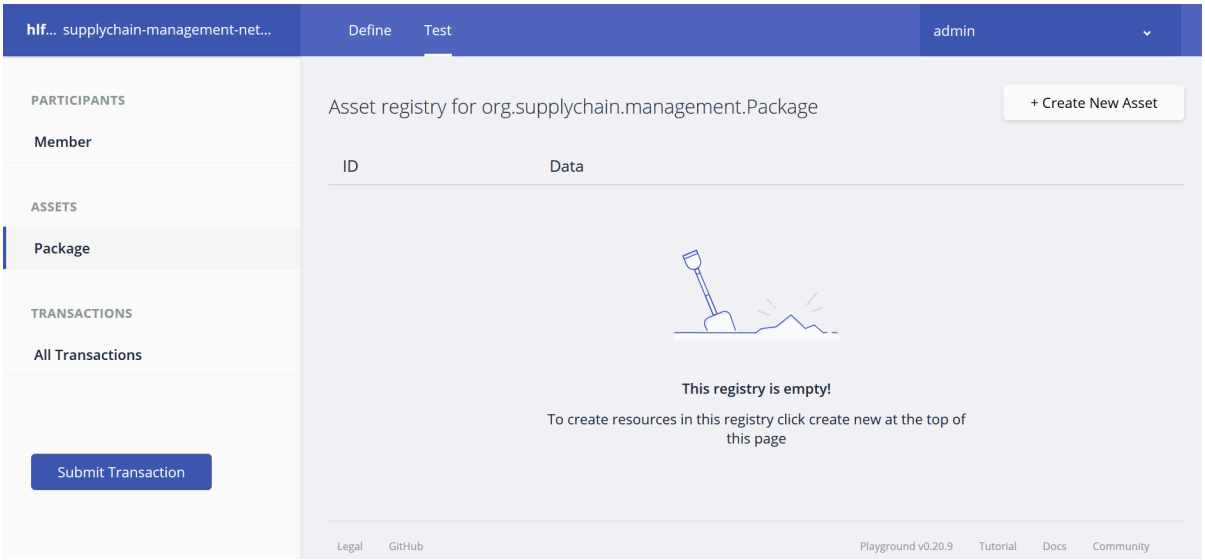
Check the network version in the bottom left. The version advances from `0.0.2-deploy.0` to `0.0.2-deploy.1`, and each subsequent deployment increments this counter by one. The business network is now live and ready to test.

*****

Testing in Composer Playground

The Composer Playground provides a test interface where you can test your deployed business network behaviour.

Click on the **Test** tab in the Composer Playground.



The test section lets you create assets and participants and submit transactions. The following steps walk you through testing 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.

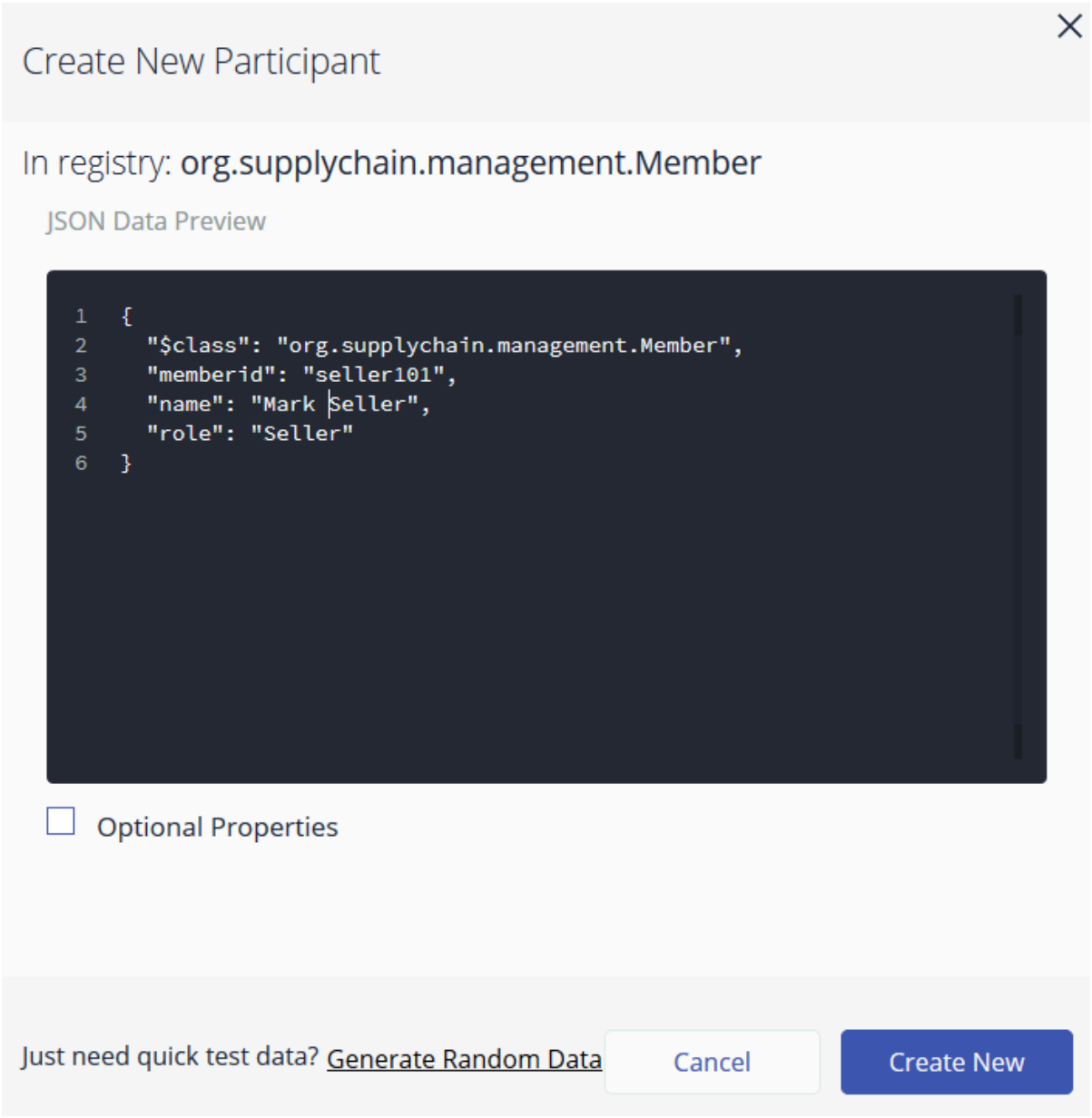
To create new participants, click on **+ Create New Participant** in the top-right section.

Create Seller

Click on **+ Create New Participant** and fill in the following fields:

- **memberId:** seller101
- **name:** Mark Seller
- **role:** Seller

Click on **Create New**.

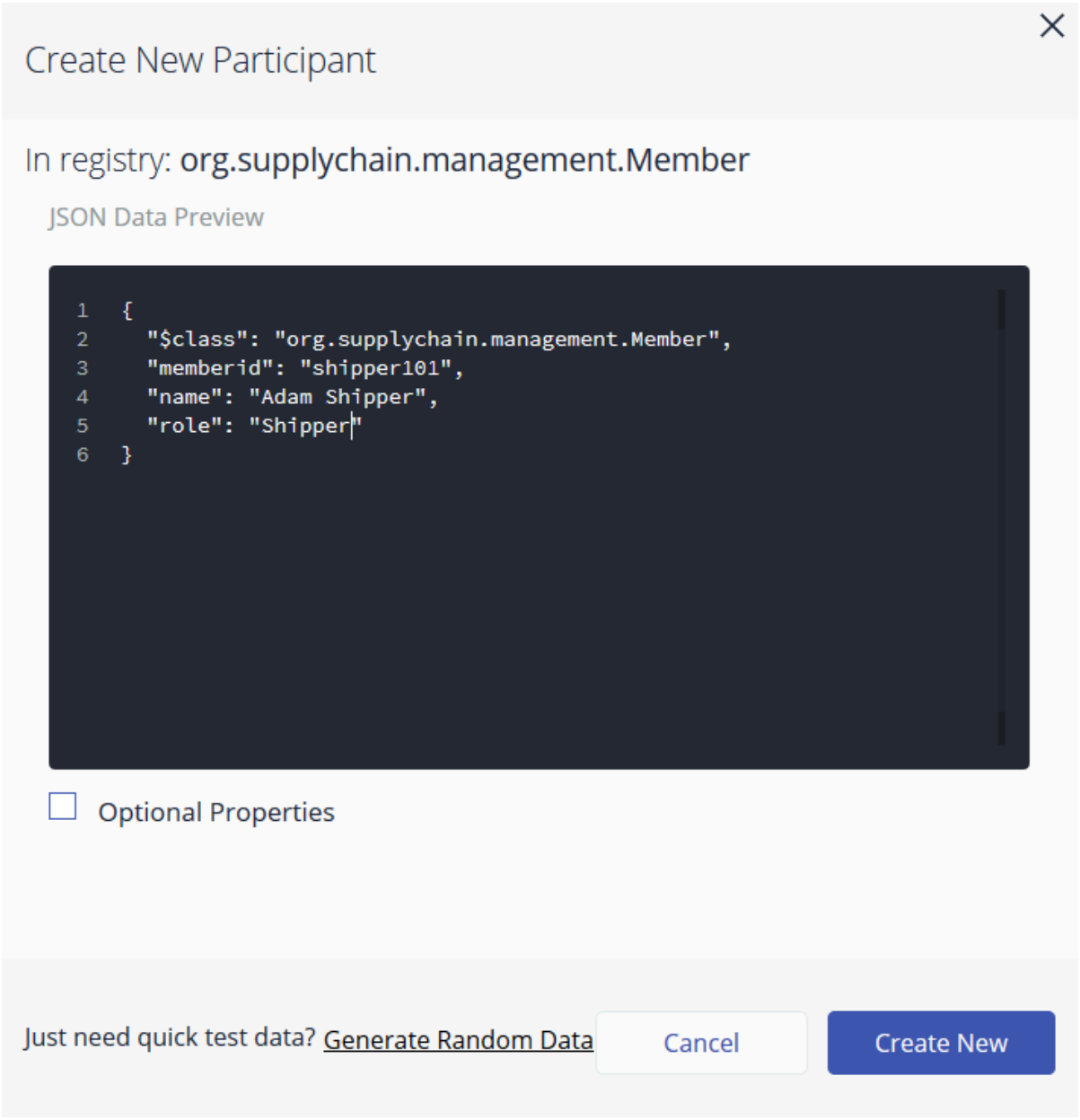


Create Shipper

Click on **+ Create New Participant** and fill in the following fields:

- **memberId:** shipper101
- **name:** Adam Shipper
- **role:** Shipper

Click on **Create New**.

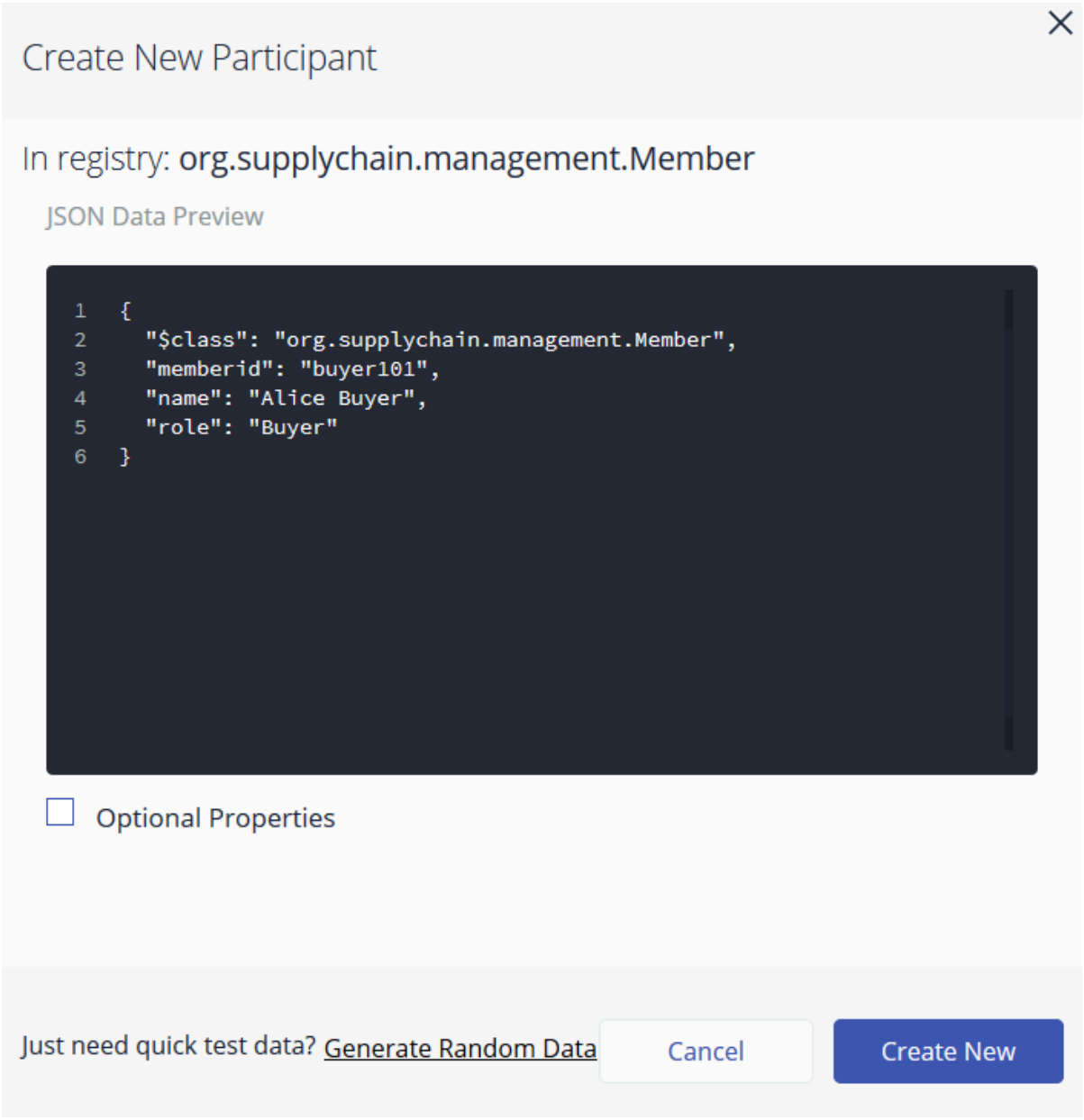


Create Buyer

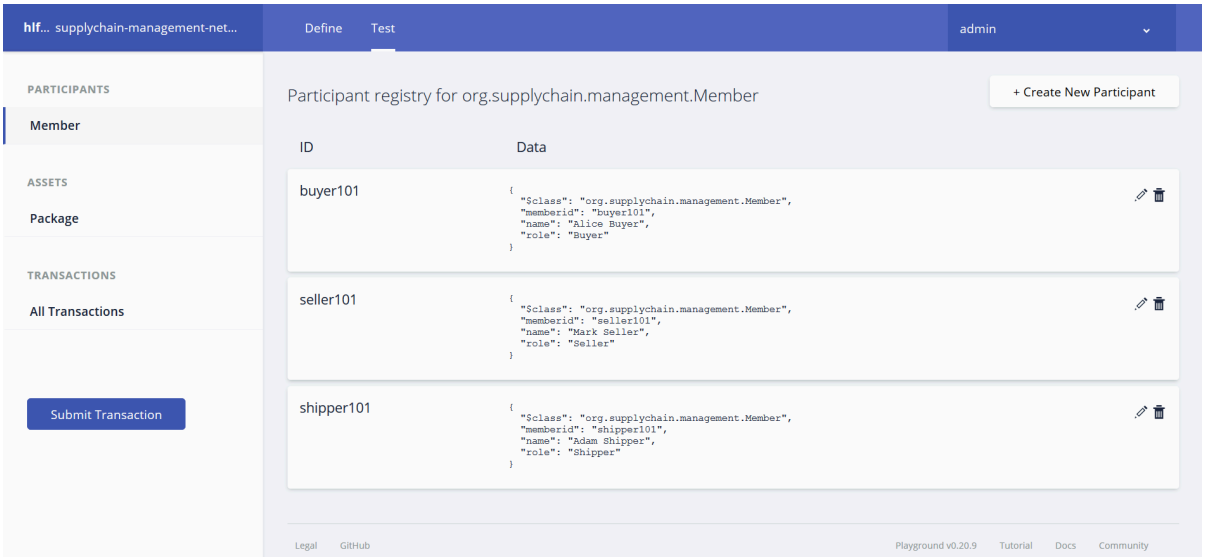
Click on **+ Create New Participant** and fill in the following fields:

- **memberId:** buyer101
- **name:** Alice Buyer
- **role:** Buyer

Click on **Create New**.



You have now created all the participants of the network.



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. Fill in the following fields:

- **packageId:** package101
- **status:** Package_Ready
- **owner:** resource:org.supplychain.management.Member#seller101

For a date field, the Playground uses the current date and time of the machine on which it is running.

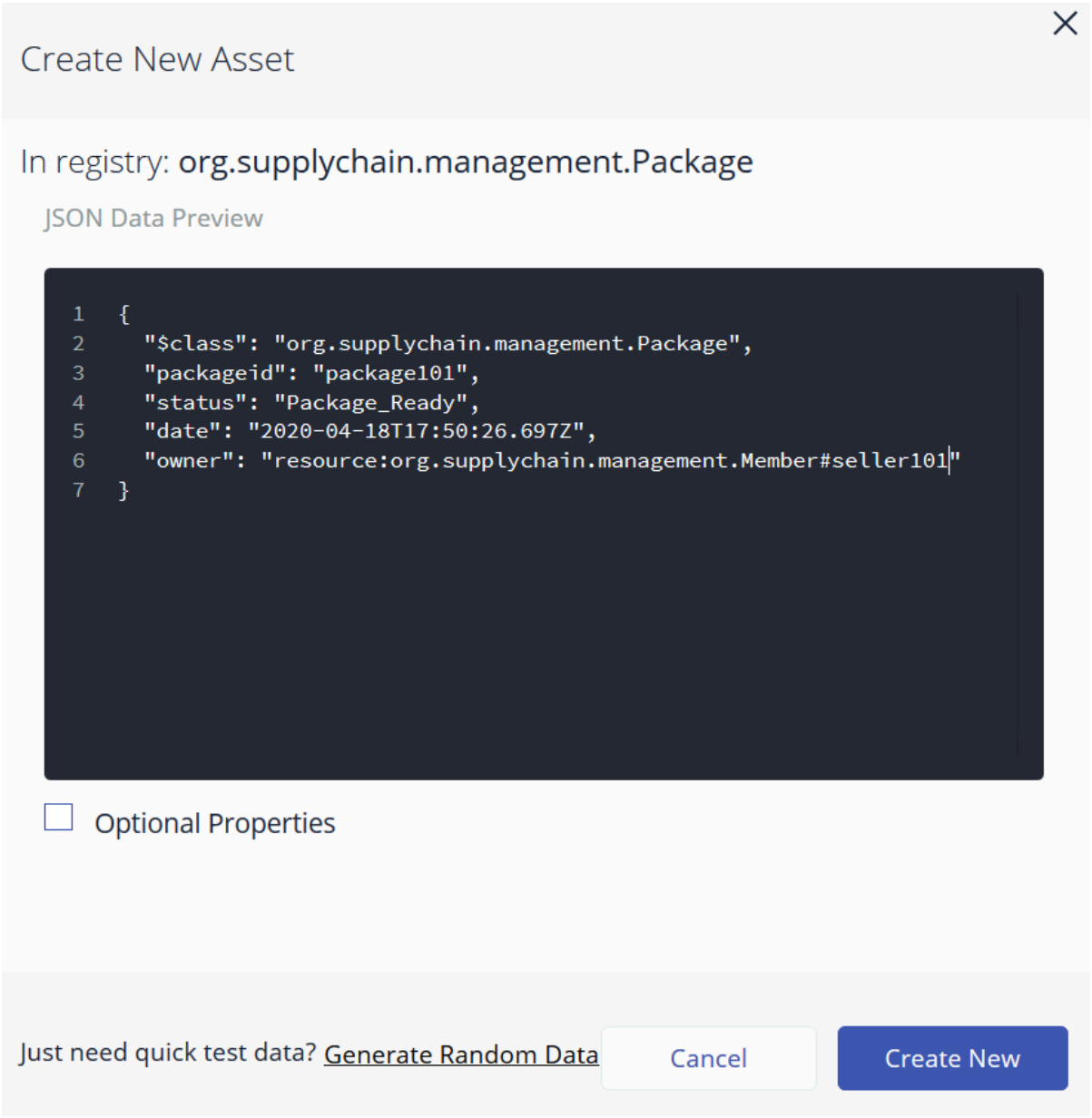
The owner field in the package is a relationship, so you must provide a resolved relationship with the member. The resolved relationship syntax follows this pattern:

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

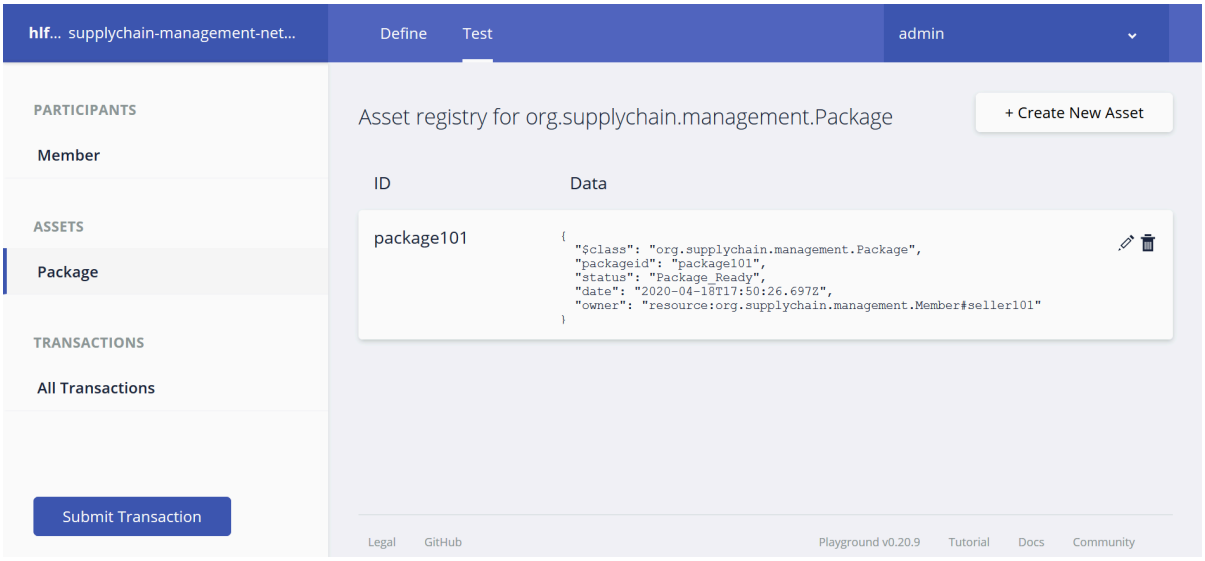
For the Seller Member, the relationship resolves to:

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

Click on **Create New**.



The Playground creates the package with id `package101` and sets its owner to `seller101`.



Transfer the Ownership of the package

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

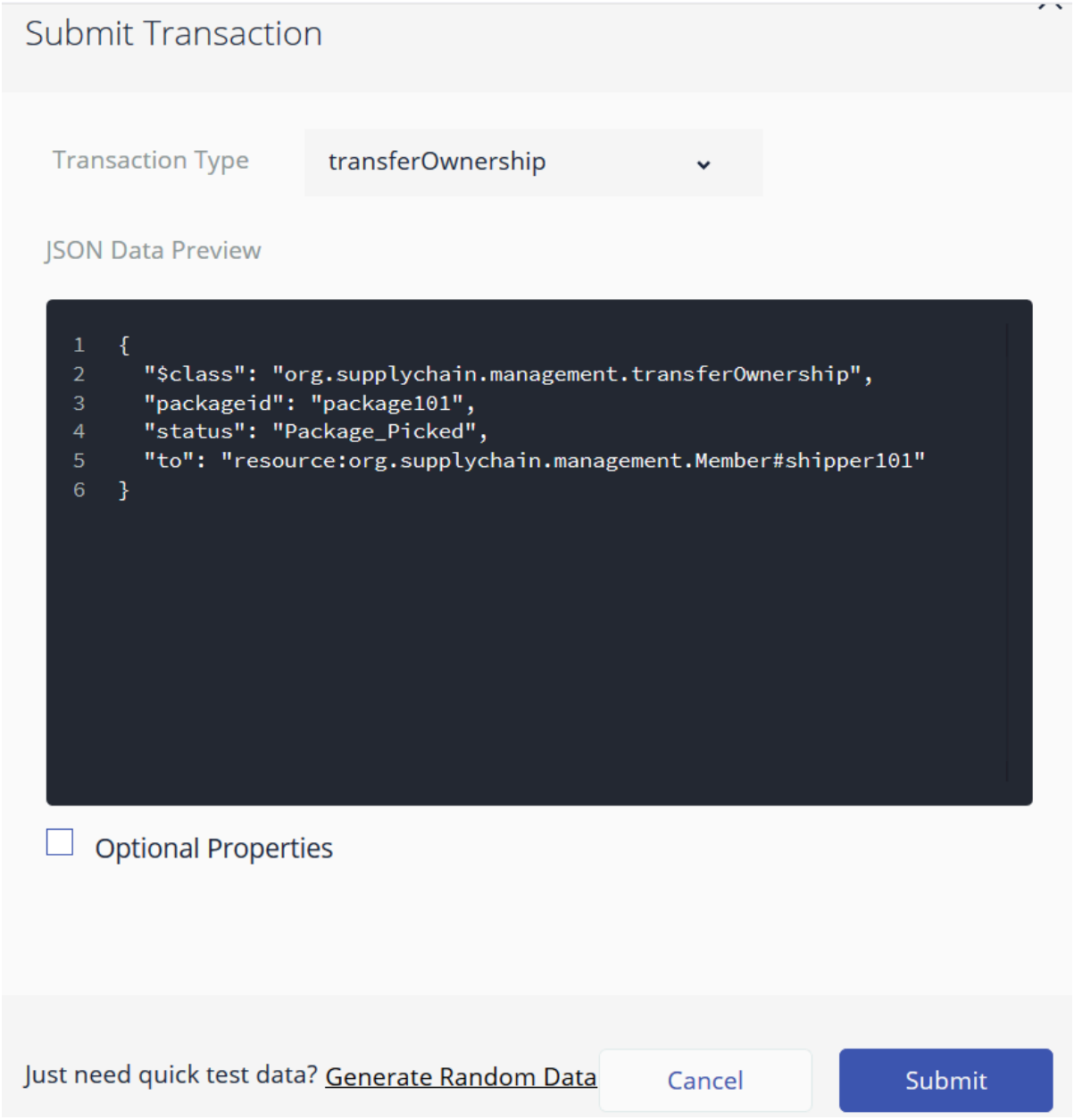
From the **Transaction Type** dropdown, you can select any available transaction. In this case, the only available transaction is **transferOwnership**.

Move the package from the Seller to Shipper

Fill in the following fields to transfer the package to the shipper:

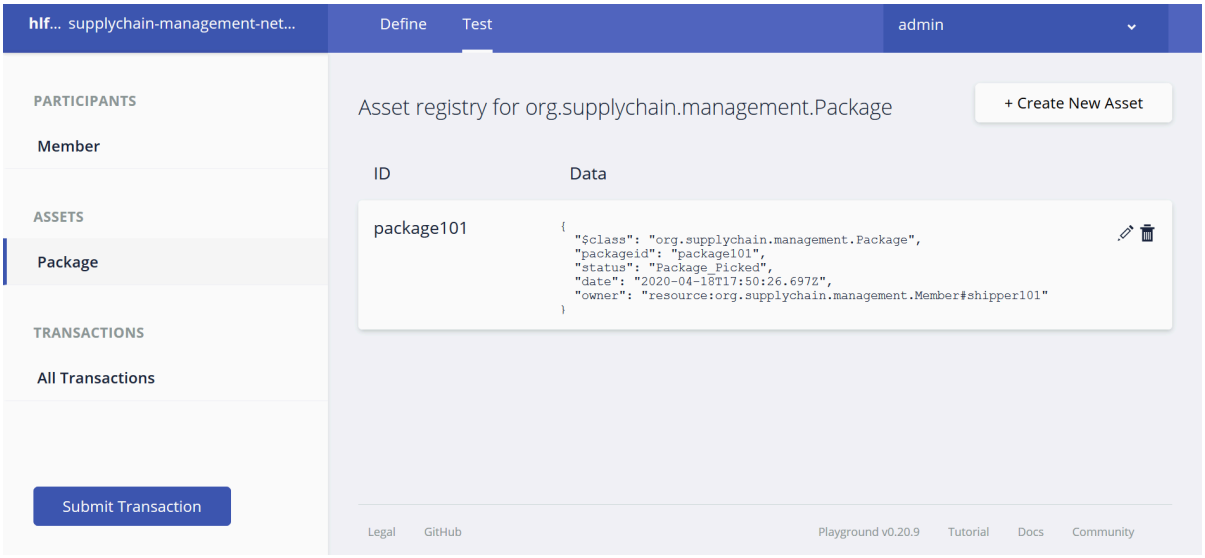
- **packageId:** package101
- **status:** Package_Picked
- **to:** resource:org.supplychain.management.Member#shipper101

> `to` field is a relationship with Member.



Submit the transaction.

Check the package owner and status — they change to *shipper101* and *Package_Picked* respectively.

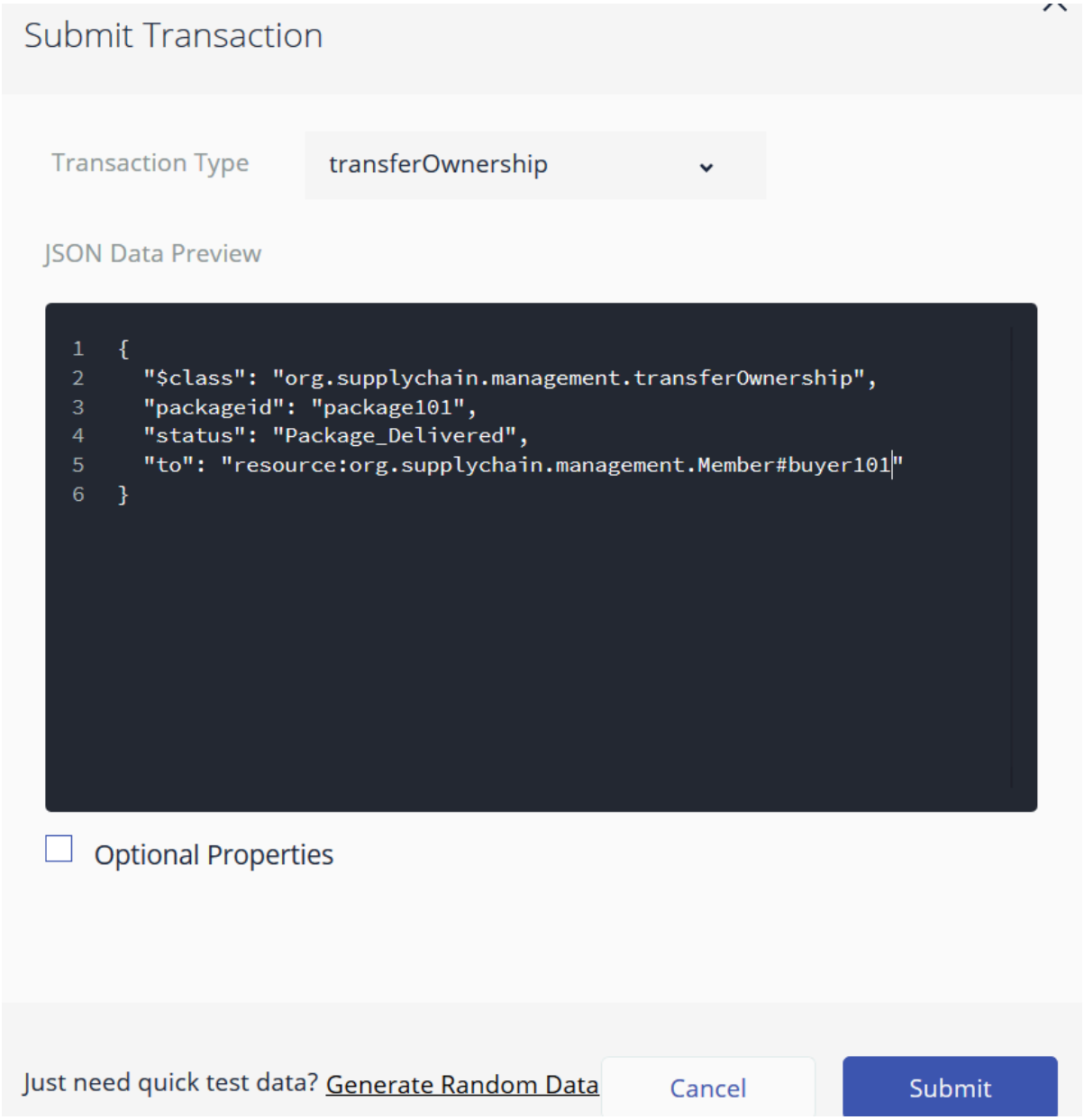


Move the package from the Shipper to Buyer

Fill in the following fields to transfer the package to the buyer:

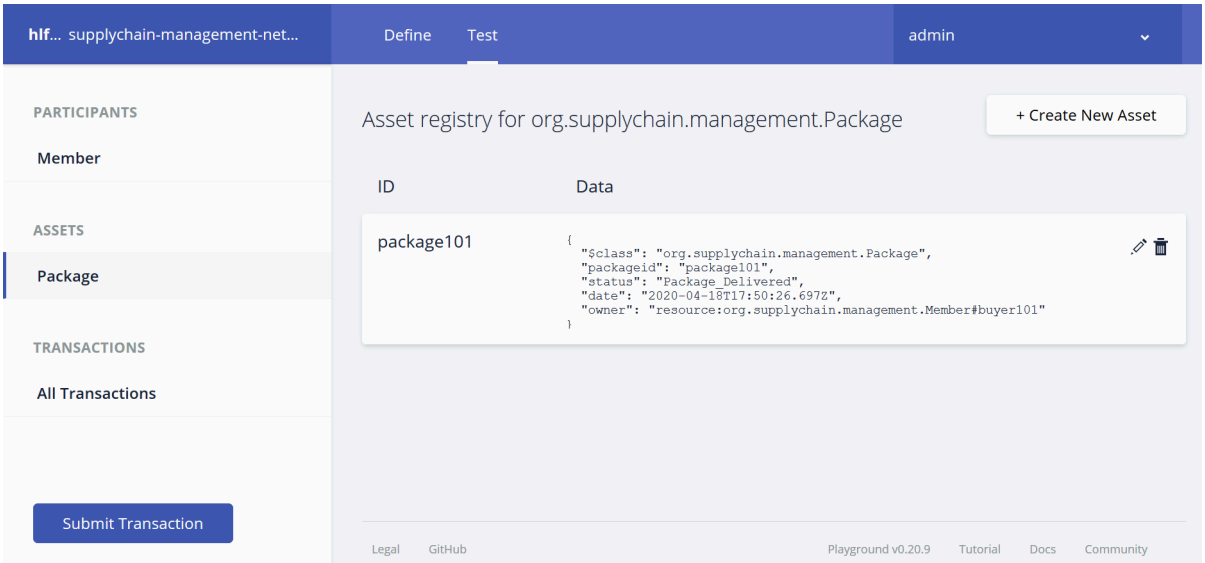
- **packageId:** package101
- **status:** Package_Delivered
- **to:** resource:org.supplychain.management.Member#buyer101

> `to` field is a relationship with Member.



Submit the transaction.

Check the package owner and status — they change to *buyer101* and *Package_Delivered* respectively, confirming the full ownership transfer cycle is complete.



*****

Export the bna

In software development, saving effort wherever possible is always a good practice.

During development, you often need to change business logic, fix bugs, or add new features. Testing is equally essential — you need to verify that the business logic behaves as expected before moving forward.

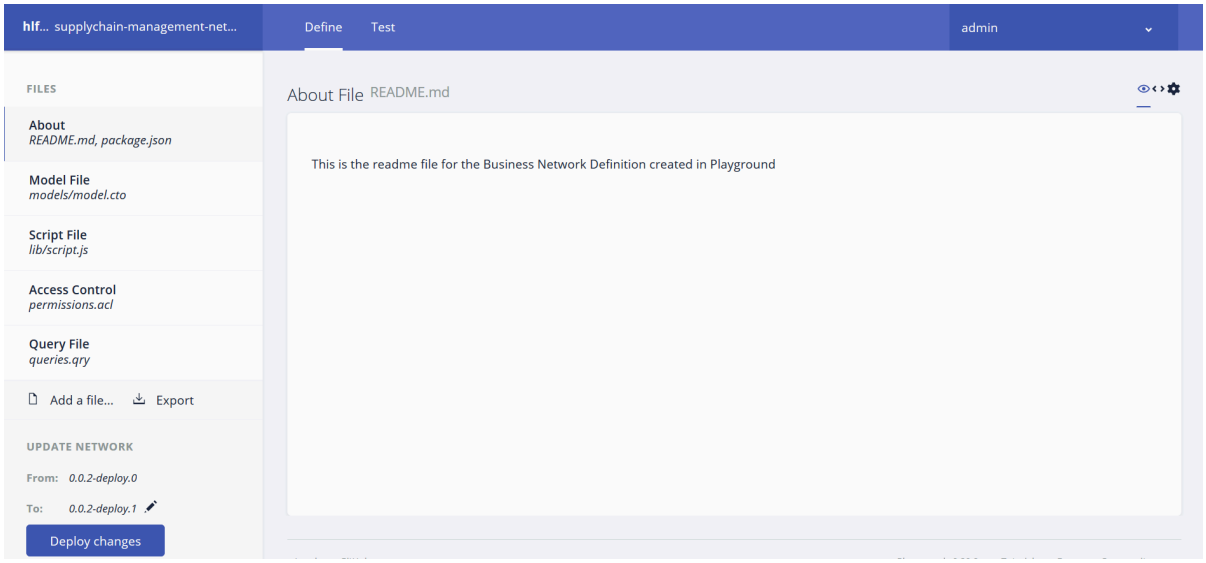
After making all your changes and running tests, you should not have to copy-paste the business logic every time you start Composer Playground.

To save that extra effort, Composer Playground lets you export the business network in the `.bna` format. Next time you want to set up the project, you can import the business network with minimal effort.

Composer Playground exports the current version of the deployed network. For example, if the current version is `0.0.2-deploy.1`, the `.bna` file captures that version. When you deploy the `.bna`, the network upgrades to the next version, `0.0.2-deploy.2`.

To export the business network, follow these steps:

1. Click the **Define** tab at the top.
2. Click **Export** in the left bar. Composer Playground downloads the business network as a `.bna` file.



Import the business network

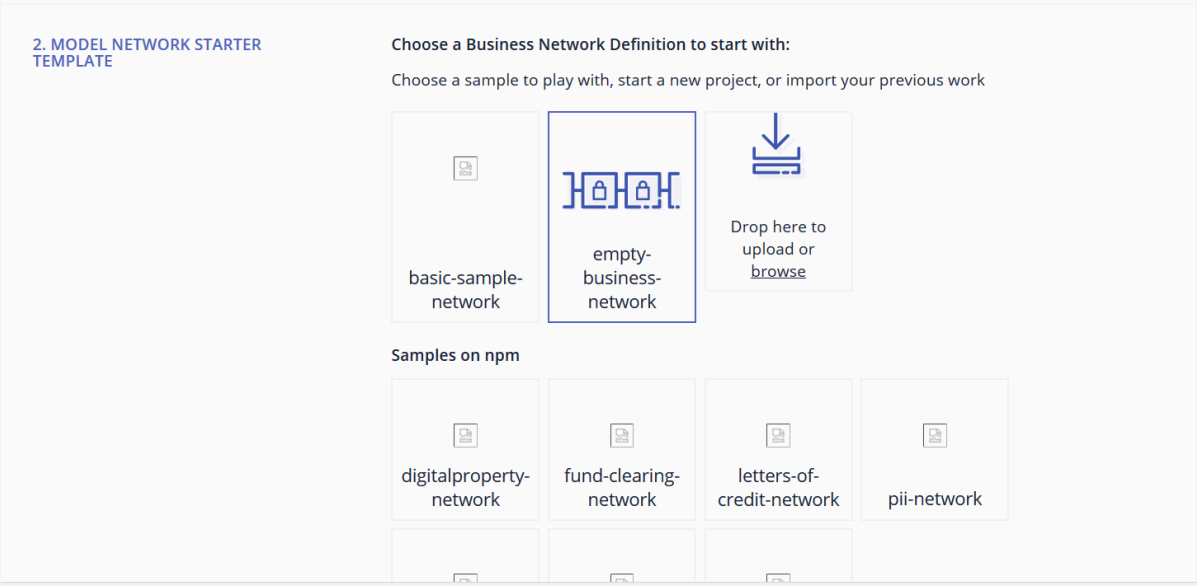
Once you have a `.bna` file, you can import it into any Composer Playground instance from the dashboard.

To reach the dashboard from a connected network:

- Click the **admin** drop-down in the top-right corner.
- Select **My Business Networks**.

Click **Deploy a new business network**.

To upload the *.bna* file, click **Drop here to upload or browse** under the **Model Network Starter Template** section.



Composer Playground reads the file and populates all the files and basic details of the business network. You can update those basic details before deploying.

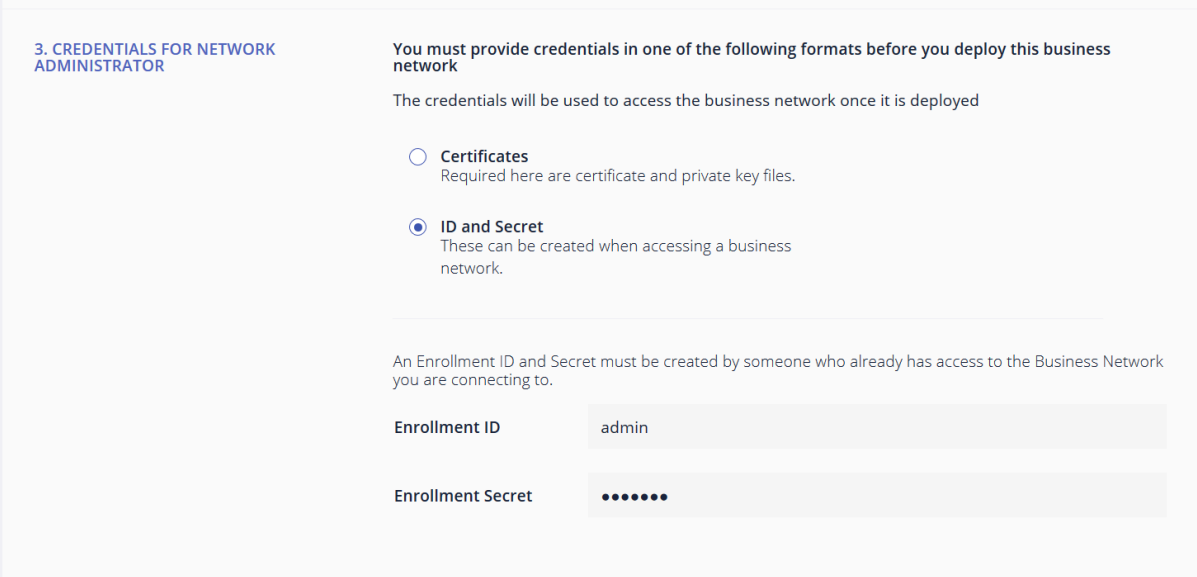
> If the same network is already deployed, you must change the network name and admin card. Composer Playground does not allow two networks with the same name.

Enter the credential for network administrator

Before deploying, you need to provide the network administrator credentials. These credentials authenticate the deployment against Hyperledger Fabric.

Select **ID and Secret**, then enter the default **Enrollment ID** and **Enrollment Secret**:

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



Deploy

Click **Deploy**. The deployment takes a few moments to complete 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 serves them on port `3000`. It not only generates and serves the APIs but also documents them using Swagger, now OpenAPI.

An API (application programming interface) acts as the bridge between the backend and the user interface. The UI relies on these APIs to interact with the backend. Representational State Transfer (REST) is a software architectural style that defines a set of constraints for creating web services.

For Example: In the supply chain management project, `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 serve as the interface for interacting with and modifying the state of a package from outside the network.

Setup composer rest server

Open a new terminal if the playground is still running. The composer rest server requires some configuration before it starts.

```
composer-rest-server
```

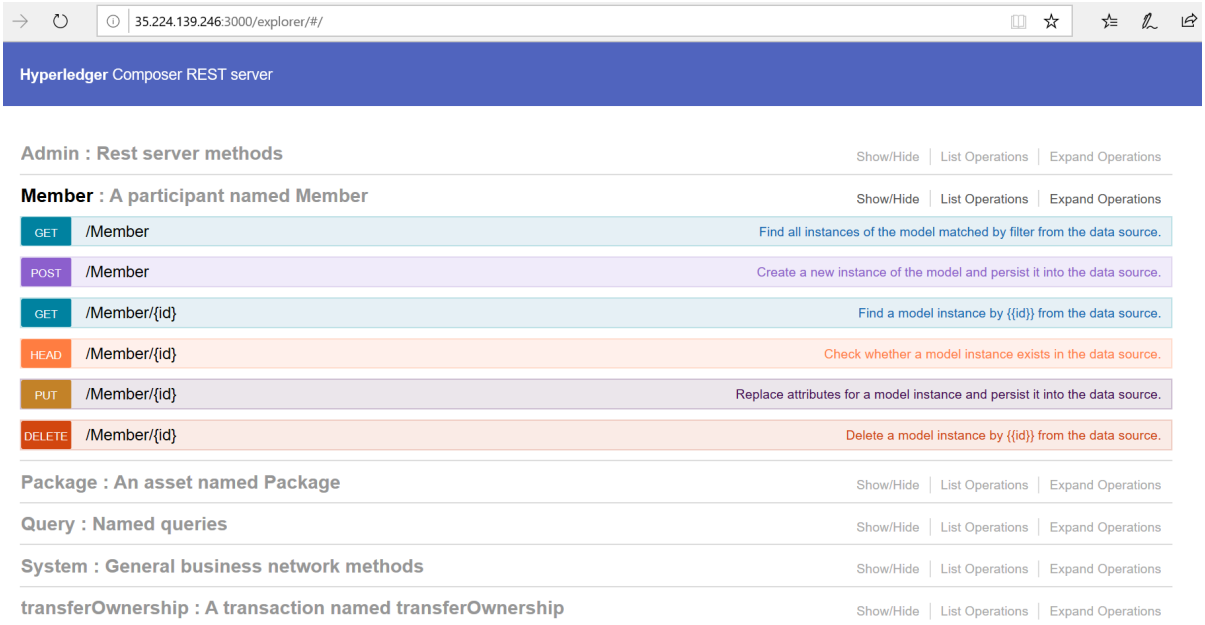
Enter the configuration as demonstrated in the screenshot. All the configuration fields are self-explanatory.

Business Network Server Configuration

```
composeruser@gcp-vm:~/fabric-dev-servers$ composer-rest-server
? Enter the name of the business network card to use: admin@supplychain-management-network
? Specify if you want namespaces in the generated REST API: never use namespaces
? Specify if you want to use an API key to secure the REST API: No
? Specify if you want to enable authentication for the REST API using Passport: No
? Specify if you want to enable the explorer test interface: Yes
? Specify a key if you want to enable dynamic logging: supplychain
? Specify if you want to enable event publication over WebSockets: No
? Specify if you want to enable TLS security for the REST API: No

To restart the REST server using the same options, issue the following command:
   composer-rest-server -c admin@supplychain-management-network -n never -u true -d supplychain

Discovering types from business network definition ...
Discovering the Returning Transactions..
Discovered types from business network definition
Generating schemas for all types in business network definition ...
Registering named query: QueryMembersByRole
Registering named query: QueryPackageByStatus
Registering named query: QueryPackageByOwner
Generated schemas for all types in business network definition
Adding schemas for all types to Loopback ...
Swagger: skipping unknown type "Member".
Added schemas for all types to Loopback
Web server listening at: http://localhost:3000
Browse your REST API at http://localhost:3000/explorer
Rest Server dynamic logging is enabled
```

Explorer

Open the browser and go to `vmipaddress:3000/explorer`. If you are using your own machine, go to `localhost:3000/explorer`.



Create a member using Explorer APIs

The Explorer interface lets you invoke each API directly from the browser. To create a member, follow these steps:

- Click on **Member**.
- Click on **POST** request.
- Copy the example value and paste it into 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

To retrieve all existing members, follow these steps:

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

This returns all **Members** as an array of objects.

Create a package

To create a package, use the Package endpoint. Follow these steps:

- Click on **Package** and **POST** request.
- Copy the example value and paste it into the data field.

> 💡 If you remember, `owner` is a Member relationship. So, you must 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 creates a package with id `package102`. To verify the package, try the **GET** request of the **Package**. This section covers a single ownership transfer to demonstrate the transaction flow.

transferOwnership

Transfer the ownership of `package102` from `seller101` to `shipper101`. The `transferOwnership` transaction endpoint lets you update the owner and status in a single call.

- Click on `transferOwnership` and select **POST** request.
- Copy the example value and paste it into the data field.
- Remove `transactionId` and `timestamp` — the system generates these automatically.

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

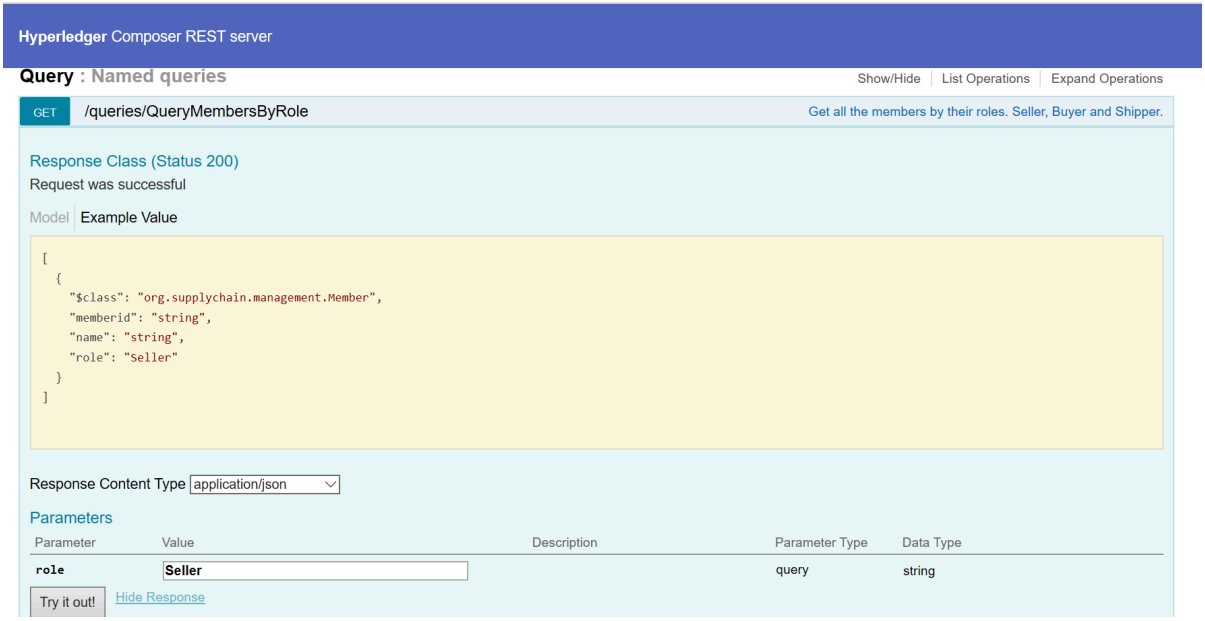
> 💡 Remember that `to` is a Member relationship. So, you must provide the resolved relationship.

To verify the result, go to the **Package** section and try the **GET/{id}** request. Enter `package102` in the parameter field — the response returns only `package102`.

Query

The Query endpoint lets you filter data by specific criteria. To query members by their role, follow these steps:

- Click on **Query**.
- Select **queries/QueryMembersByRole**.
- In the **parameters** section, enter the role **Seller**.



This returns all the Sellers.

Similarly, you should try all the queries to explore the full range of filtering options.

System

These requests let you interact with the Composer system directly. For example, you can retrieve a history of transactions — every action that writes to the blockchain counts as a transaction.

Use screen

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

Right now, the `composer-rest-server` runs in a terminal. If you close that terminal, you stop the server — which is not ideal when you need the server running continuously and want to exit the session.

Using `screen`, you can create a terminal session that you can attach to or detach from at any time without stopping the running 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 are unable to create a screen, try with `sudo`.

You are now in a new screen. Start the `composer-rest-server`.

Detach the screen

To detach the screen, use `Ctrl + a d`. Hold `Ctrl`, press `a`, and then press `d`.

The screen detaches and the process continues running. Verify this by checking the explorer in the browser.

List the screen

To list all running screens, use the following command.

Syntax

```
screen -ls
```

This lists all screens with their IDs, which you need to reattach to a session.

Attach the screen

To attach to a screen, copy the screen ID from the list command.

Syntax

```
screen -r screenid
```

You are now attached to the screen.

Terminate the screen

Stop the running service, then use `exit` to terminate the screen.

*****

Frontend

A simple UI can make an application more appealing and useful.
You cannot ask your users to interact with your business network through APIs or the playground directly.

Including a UI section and describing it in detail is out of scope for this ebook.

Several options are available to you, and you are free to explore whichever fits your skill set. Each option uses the `composer-rest-server` APIs to interact with the business network. Make sure `composer-rest-server` is up and running. Use the `screen` command to run the rest server in the background.

- The easiest way is to use the `expressjs` framework with a view engine such as **pug**, **ejs**, or **handlebar**.
- You can also use a UI framework like **Angular**, **Reactjs**, or **Vuejs**. The learning curve is steeper, but the results are more polished. Each framework has CSS libraries available that provide pre-built UI components such as headers and footers with default styling.
- If you are not comfortable with **CSS**, you can rely on **Bootstrap** or **SemanticUI** to handle the styling for you.

All of the options above require a basic understanding of JavaScript. JavaScript frameworks are currently the most widely used, though many options exist across other languages as well.

Hosting on Github Pages

You should become familiar with Git and GitHub as early as possible, since GitHub is the ideal place to showcase your work to the world.

GitHub not only lets you save your code but also hosts static websites free of charge.
You can host your UI on GitHub Pages, which supports static websites only.

To get started, you need the following:

- A GitHub account
- A repository with your project pushed to it
- GitHub Pages enabled through the repository's Settings

> It will host on .github.io/repositoryName

Many tutorials on the internet can guide you through hosting on GitHub Pages.

If you use a framework like Angular, Reactjs, or Vuejs, each provides a build command that produces static output you can host directly on GitHub Pages.

*****

Conclusion

This ebook set out to give you a foundational understanding of Hyperledger Composer, using supply chain management as a practical example. Now that you know the core concepts, you can apply them to build solutions across industries such as Healthcare, Food Tracking, Real Estate, and Document Stamping.

If you want to take this project further, try adding one of the features below.

Add temperature reading on the package

Create a test API that returns a temperature reading. Call this API every 5–10 minutes, or at any interval you prefer, and update the package temperature accordingly. Whenever the temperature rises above a threshold value, trigger a transaction that marks the package as spoiled. You can store the threshold value as a variable on the package itself.

Also consider adding a package-acceptance feature for the buyer: the buyer should only be able to accept the package if it is still fresh.

Instead of a test API, you can use a weather API. Many weather APIs are free to consume up to a certain request limit.

Add IoT sensor

This is an advanced option. Rather than relying on a weather API or a test API, use IoT sensors to read the temperature directly. A simple starting point is a breadboard paired with a temperature sensor.

Logistic Blockchain Network

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

*****

References

The following resources provide additional context on Hyperledger Composer, Hyperledger Fabric, and the JavaScript concepts used throughout this book.

- Hyperledger Projects
- Hyperledger Fabric
- Hyperledger Composer
- Query Language
- Query Syntax
- Registry
- Transaction 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