Pre-requirements

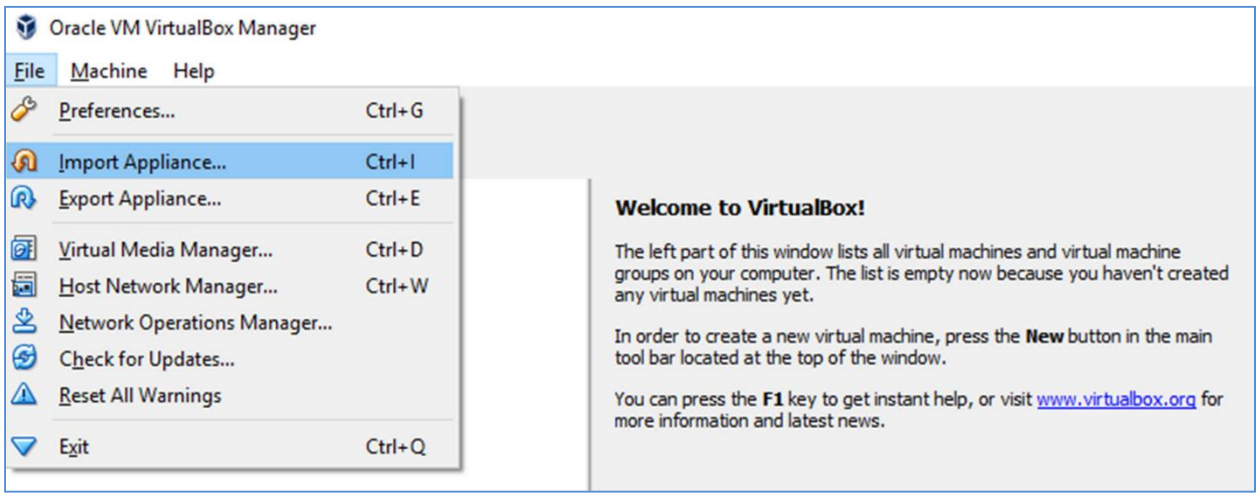
Before we begin we the actual hands-on lab we have set up a starting environment so you don't need to install the required software (i.e. Hyperledger Fabric, Visual Code (with composer plugin). We have made a Virtual Box VM running Ubuntu 16.04 LTS, which will be distributed in class. So the only pre-requirement for your local system is installing Virtual Box 5.2 or higher.

Downloading Virtual Box:

Windows: http://download.virtualbox.org/virtualbox/5.2.0/VirtualBox-5.2.0-118431-Win.exe
OS X: http://download.virtualbox.org/virtualbox/5.2.0/VirtualBox-5.2.0-118431-OSX.dmg
Linux: https://www.virtualbox.org/wiki/Linux_Downloads

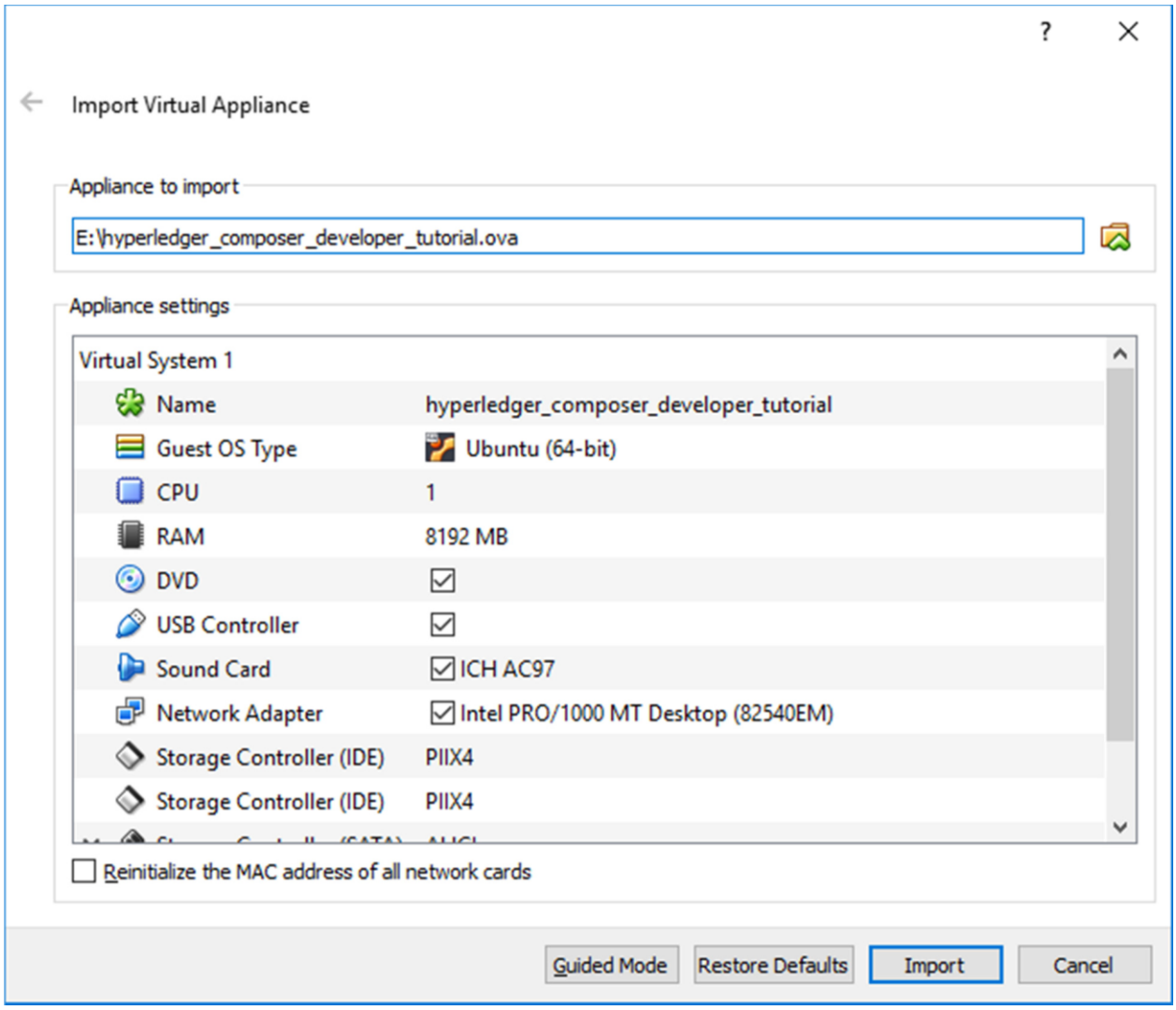
Import OVA file into Virtual Box:

If you received the OVA file from one of the Lab leaders you can import the file by opening the VirtualBox Manager application and in the top menu choosing the option Machine -> Add.



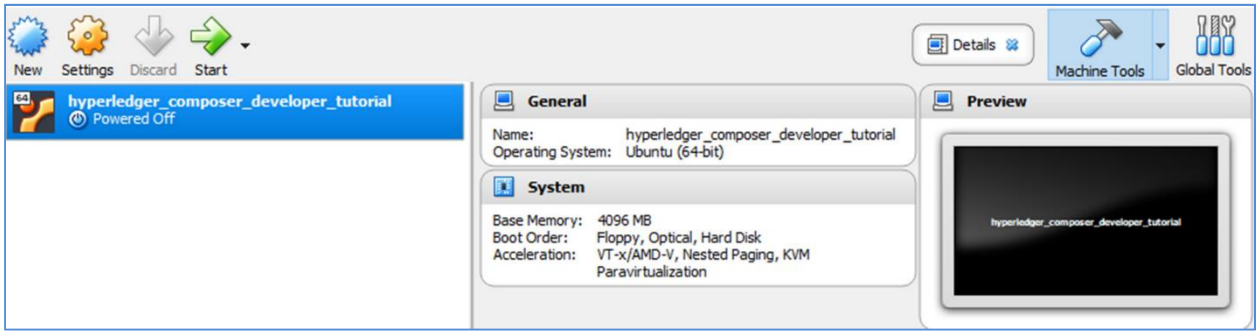
And find and select the file you want to import i.e. "Hyperledger_composer_developer_tutorial.ova"

(Alternatively you can double click on the OVA file in a file explorer to open it directly in Virtual Box)



If you want to control the location of the VM image(s) you can edit the value of the field "*Virtual Disk Image*" of the Virtual System 1 settings. Click on the Import button to accept the appliance settings.

After a few minutes up to 20 minutes (depending on speed of storage) the import is done and you can start the image by clicking on the start button.



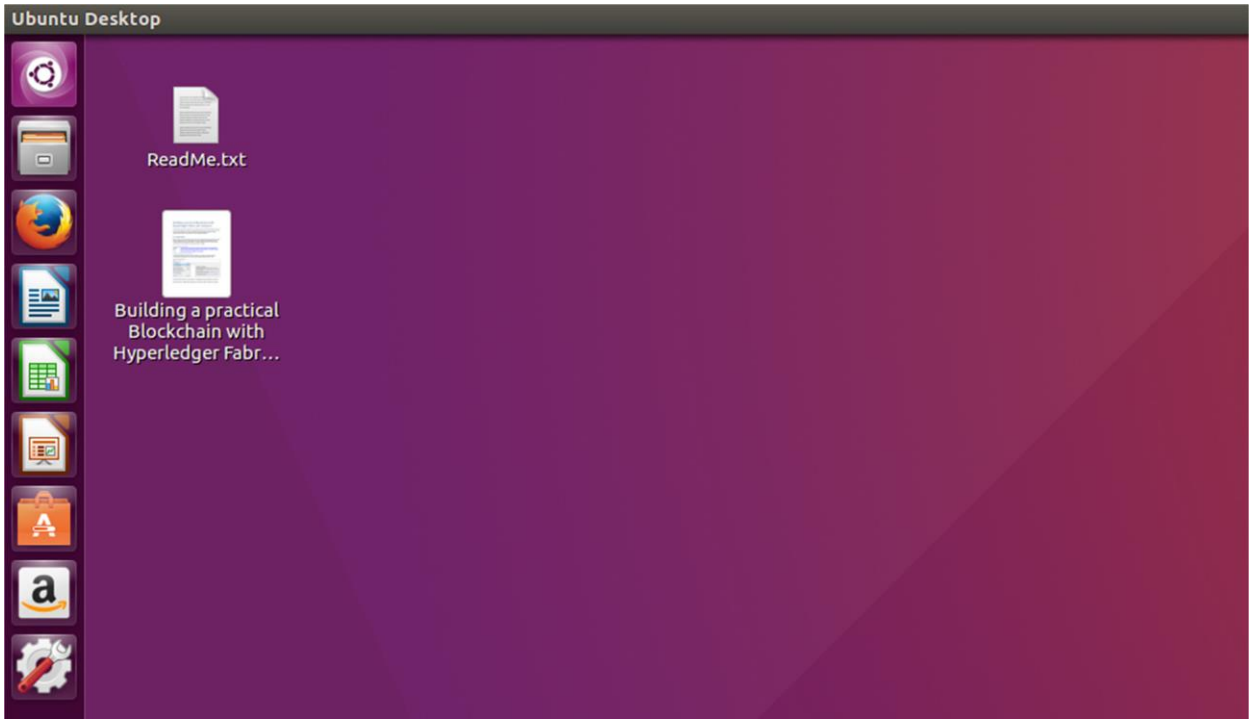
Building your network with Hyperledger Composer

In the introduction presentation we talked about Hyperledger Composer in more detail, but in short it is an extensive, open development toolset and framework to make developing blockchain applications easier. You can use Composer to rapidly develop use cases and deploy a blockchain solution in weeks rather than months. Composer allows you to model your business network and integrate existing systems and data with your blockchain applications.

After starting the VM you will be welcomed by the login screen.

➔ *Use the password **hyperledger** to login*

After logging in you are presented with the desktop. Currently no service or applications is running in the background. On the desktop you will also find this document, a small readme file with URLs and starter instructions.

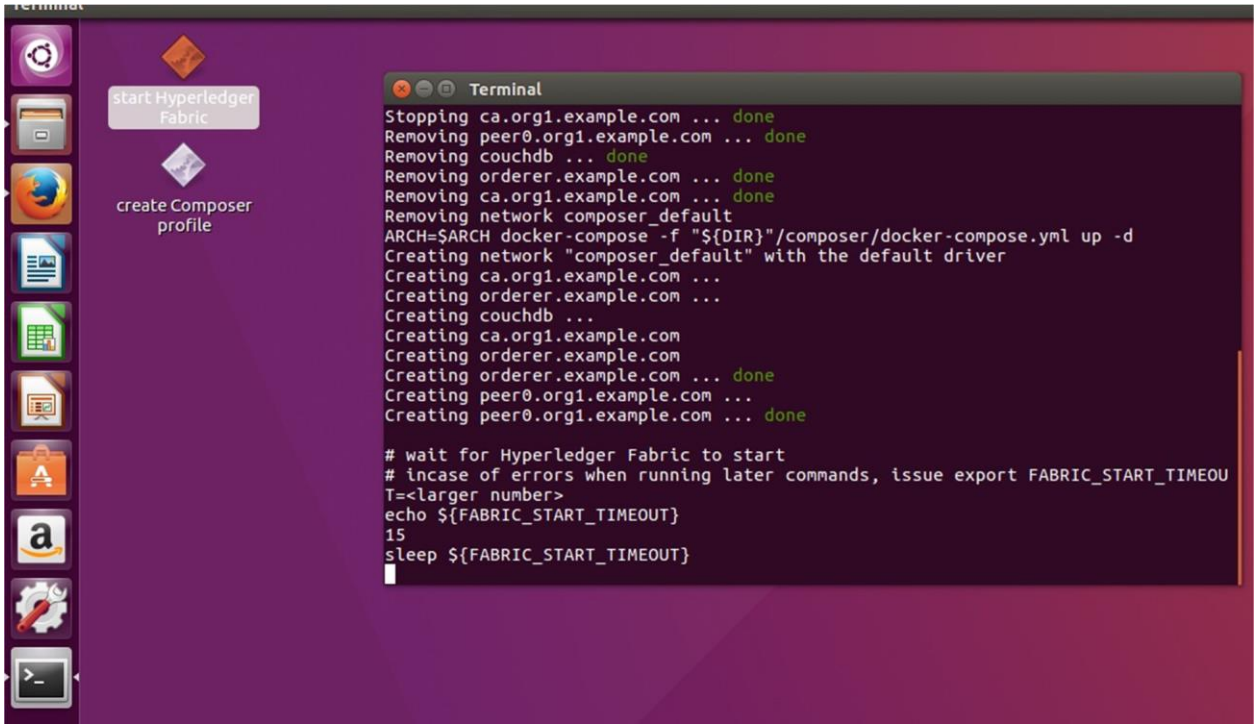


Deploying your network to Hyperledger Fabric

In part one you have created a business network definition that can be used to register car accidents in the Hyperledger Composer Playground, and exported the network file.

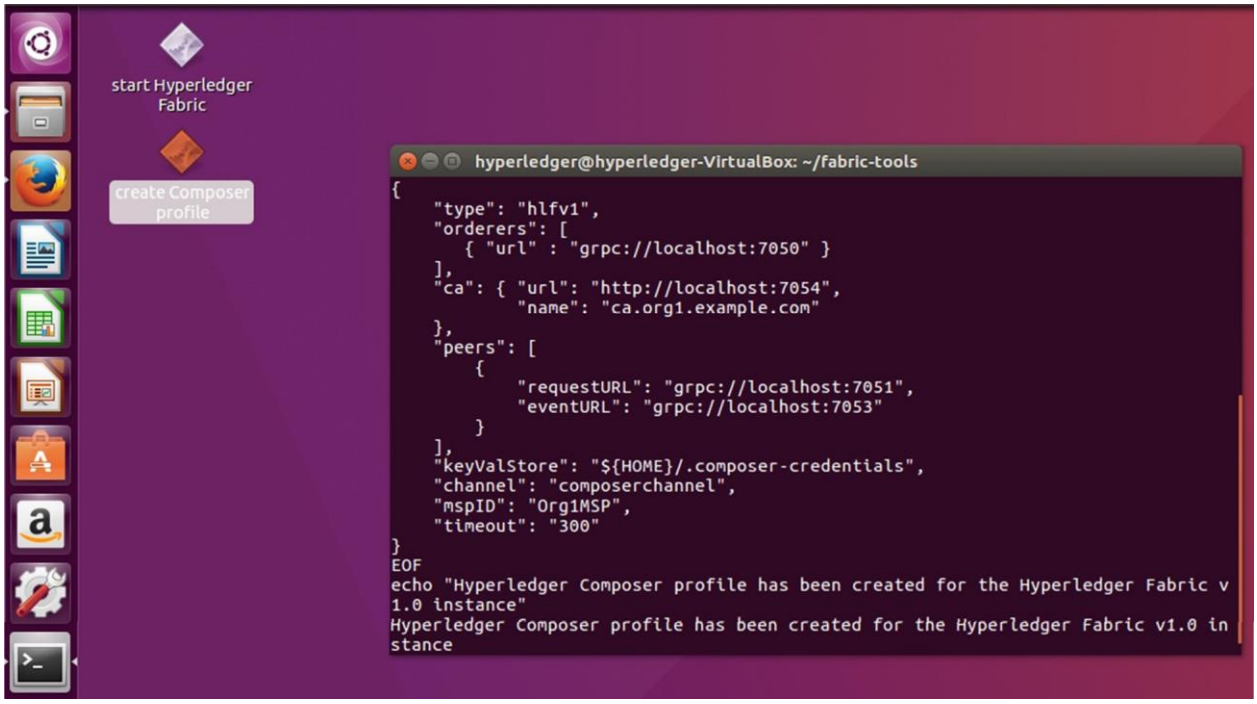
In part two you will use your network file and deploy it locally to the Hyperledger Fabric, to create your very own Blockchain solution.

➔ *Run "start Hyperledger Fabric" on the desktop*



Running start Hyperledger Fabric will start the Hyperledger Fabric in your VM. This will take a minute. Next up is creating a profile that will connect to the Hyperledger Fabric.

➔ *Run "create Composer profile" on the desktop*



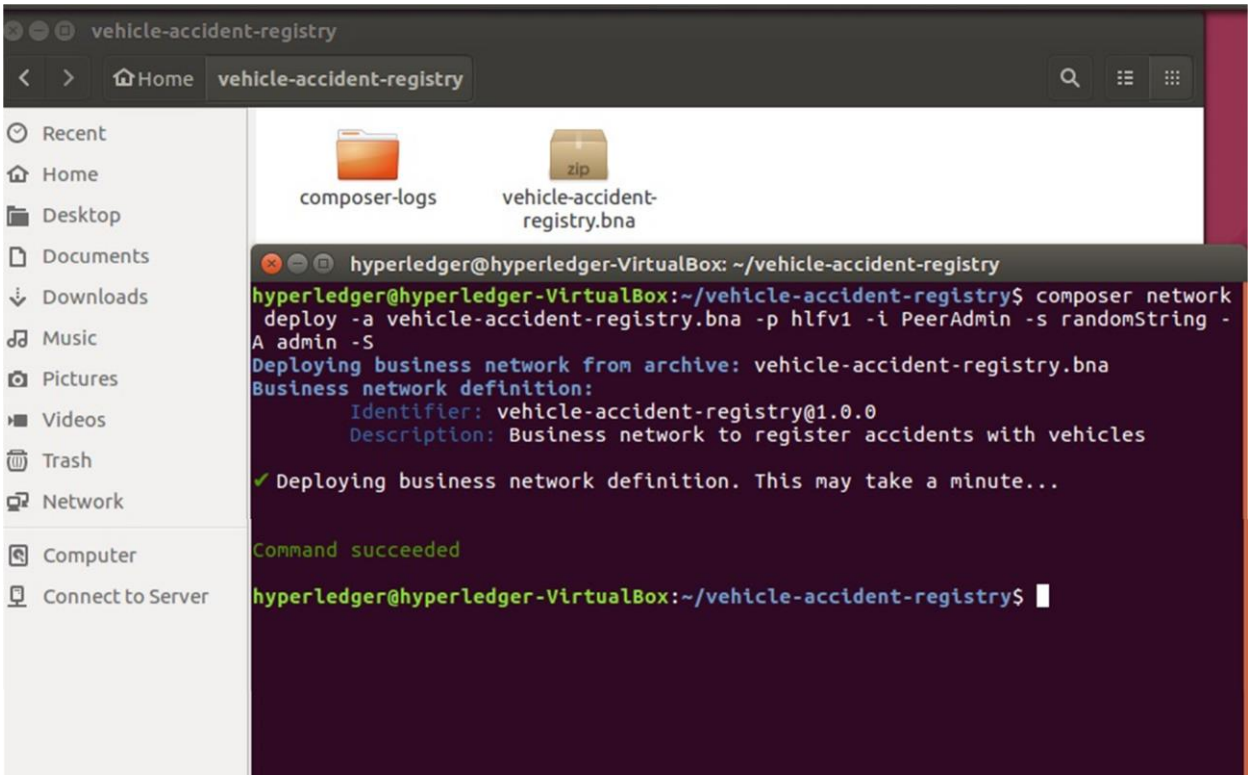
The terminal will probably close too quickly to be able to read what's printed inside.

Hyperledger Fabric is now up and running, and you can deploy a network to it. You have already created a business network in the previous steps, and it is a prime network to be deployed. If you have not yet downloaded your network as a .bna file, now would be the time to do it. Place the .bna file in a folder with a fitting name. The tutorial will use the folder "vehicle-accident-registry" under Home. Nagivate to the folder where the .bna file has been saved and open a terminal there.

➔ *Right click in the folder and choose "Open in Terminal"*

Deploy your network to the Fabric by inserting the full command into the terminal:
make sure to substitute vehicle-accident-registry for the name of your .bna file

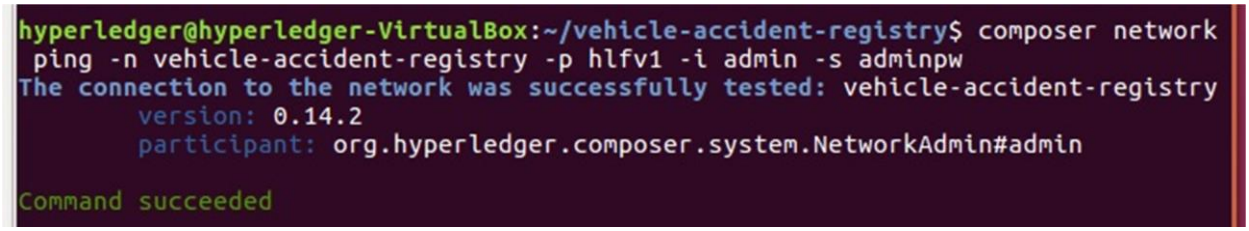
```
composer network deploy -a vehicle-accident-registry.bna -p hlfv1 -i
PeerAdmin -s randomString -A admin -S
```



This will again take a short while to complete. Afterwards your network will have been deployed to the Fabric, and you can start using it. Verify that the deployment succeeded by inserting into the terminal:

```
composer network ping -n vehicle-accident-registry -p hlfv1 -i admin -s adminpw
```

This will test the connection to the network using the admin account.



Create a REST API and a web application

In this final chapter you will create a web application that will connect to a new REST API. Enter into the terminal:

```
yo hyperledger-composer
```

There will be a dialogue, answer the questions as shown in the screenshot below:

```
hyperledger@hyperledger-VirtualBox: ~/vehicle-accident-registry
hyperledger@hyperledger-VirtualBox:~/vehicle-accident-registry$ yo hyperledger-c
omposer
Welcome to the Hyperledger Composer project generator
? Please select the type of project: Angular
You can run this generator using: 'yo hyperledger-composer:angular'
Welcome to the Hyperledger Composer Angular project generator
? Do you want to connect to a running Business Network? Yes
? Project name: vehicle-app
? Description: Vehicle Accident Registry
? Author name: Some Name
? Author email: Some Email
? License: Apache-2.0
? Business network identifier: vehicle-accident-registry
? Connection profile: hlfv1
? Enrollment ID: admin
? Enrollment secret: adminpw
? Do you want to generate a new REST API or connect to an existing REST API?  Ge
nerate a new REST API
? REST server port: 3000
? Should namespaces be used in the generated REST API? (Use arrow keys)
  Always use namespaces
❯ Never use namespaces
```

After submitting the final question the web application will be built, this will take a couple of minutes to complete. Yeoman will generate a new folder inside your current folder that contains the necessary files to start your REST API and angular application. After completion your terminal should look something like this:

```
hyperledger@hyperledger-VirtualBox: ~/vehicle-accident-registry
lled via remote

> protobufjs@6.6.3 postinstall /home/hyperledger/vehicle-accident-registry/vehic
le-app/node_modules/protobufjs
> node scripts/postinstall


> node-sass@4.5.3 postinstall /home/hyperledger/vehicle-accident-registry/vehicl
e-app/node_modules/node-sass
> node scripts/build.js

Binary found at /home/hyperledger/vehicle-accident-registry/vehicle-app/node_mod
ules/node-sass/vendor/linux-x64-48/binding.node
Testing binary
Binary is fine
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.1.2 (node_modules/fse
vents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@
1.1.2: wanted {"os":"darwin","arch":"any"} (current: {"os":"linux","arch":"x64"}
)

added 1666 packages in 203.351s
hyperledger@hyperledger-VirtualBox:~/vehicle-accident-registry$
```

When the building of your angular application has completed go to the directory of your web application and start it by entering into the terminal:

if you have given your application a different name, substitute it here!

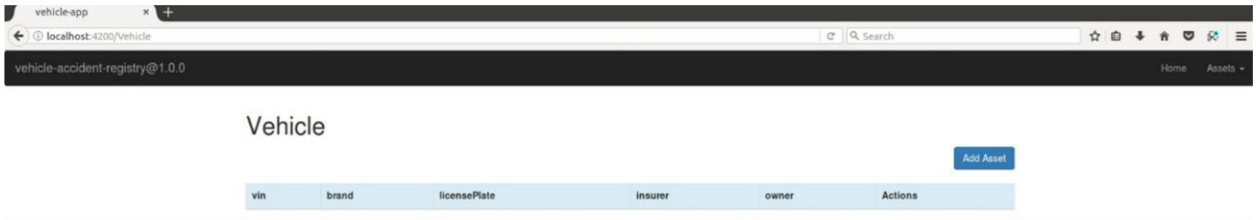
```
cd vehicle-app
npm start
```

```
hyperledger@hyperledger-VirtualBox:~/vehicle-accident-registry$ cd vehicle-app
hyperledger@hyperledger-VirtualBox:~/vehicle-accident-registry/vehicle-app$ npm
start

> vehicle-app@0.0.1 start /home/hyperledger/vehicle-accident-registry/vehicle-ap
p
> concurrently "ng serve --host 0.0.0.0" "npm run app"

[1]
[1] > vehicle-app@0.0.1 app /home/hyperledger/vehicle-accident-registry/vehicle-
app
[1] > composer-rest-server -n vehicle-accident-registry -p hlfv1 -i admin -s adm
inpw -N never -P 3000
[1]
```

Note that this creates a REST API at port 3000 for you! If you would like to create a REST API without the angular application you could simply enter composer-rest-service in the terminal, which will create a dialogue and a REST API for you!

```
hyperledger@hyperledger-VirtualBox: ~/vehicle-accident-registry/vehicle-app
[0]  91% additional asset processing

[0]  92% chunk asset optimization

[0]  94% asset optimization

[0]  95% emitting

[0] Hash: e65f815feb626006fd8c
[0] Time: 35123ms
[0] chunk    {0} polyfills.bundle.js, polyfills.bundle.js.map (polyfills) 265 kB
{5} [initial] [rendered]
[0] chunk    {1} main.bundle.js, main.bundle.js.map (main) 92.5 kB {4} [initial]
[rendered]
[0] chunk    {2} styles.bundle.js, styles.bundle.js.map (styles) 183 kB {5} [ini
tial] [rendered]
[0] chunk    {3} scripts.bundle.js, scripts.bundle.js.map (scripts) 435 kB {5} [
initial] [rendered]
[0] chunk    {4} vendor.bundle.js, vendor.bundle.js.map (vendor) 3.83 MB [initia
l] [rendered]
[0] chunk    {5} inline.bundle.js, inline.bundle.js.map (inline) 0 bytes [entry]
[rendered]
[0] webpack: Compiled successfully.
```

This will start your web application, and you can visit it at http://localhost:4200. Explore the application, you will notice that there is no data at all. Time to add some!



The angular application does not allow you to create participants, instead you'll need to directly access the REST API.

➔ *Go to http://localhost:3000*

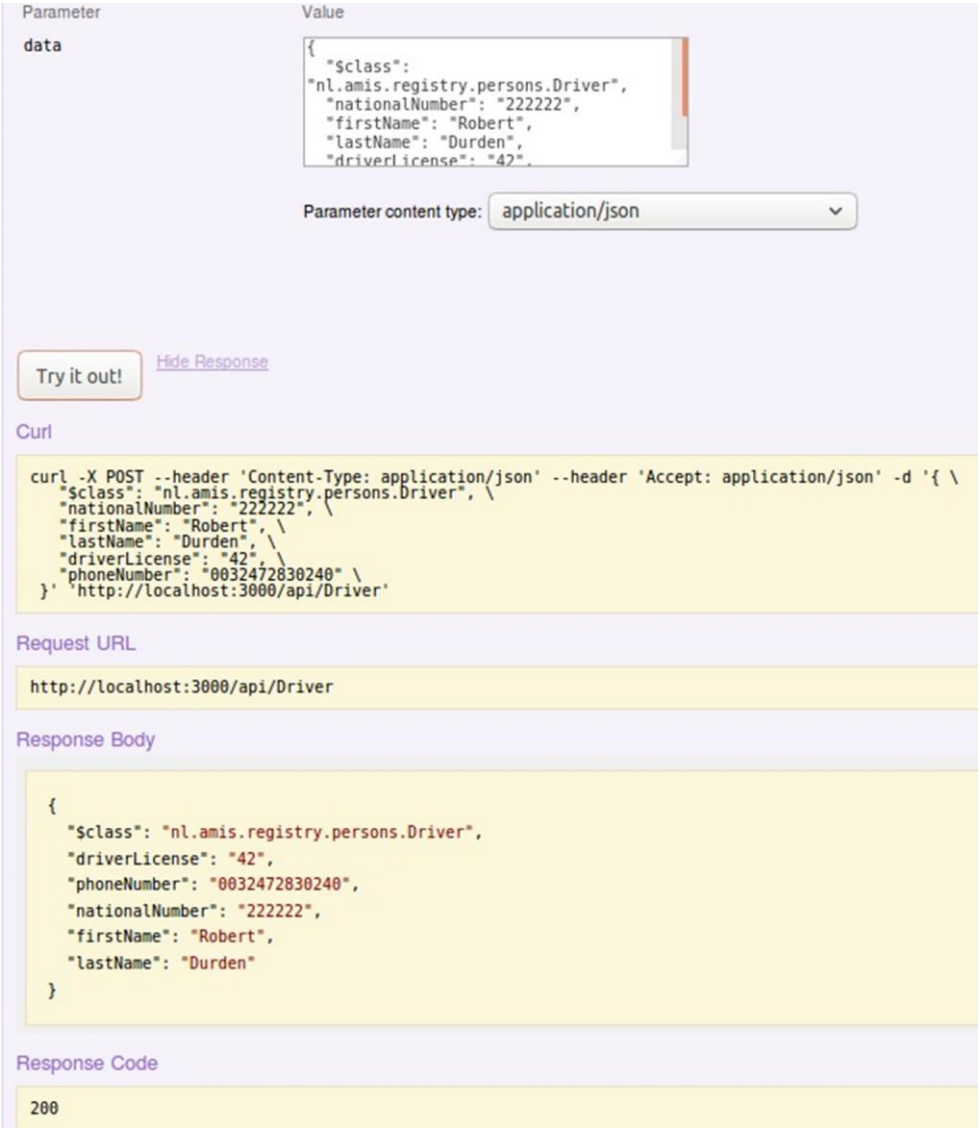
Here you can POST new participants and assets, and you can GET the existing participants and assets. There are several things you will need to POST to be able to complete the examples later on in this tutorial. First we will create a driver.

➔ *Click on Driver and then on POST*

Now you can enter the information of a driver. Examine the page presented to you. You will need to supply an entire JSON construction in the data field. Notice that there is an example that will show you what type of class this participant/asset is, how many variables it has and which types of variables they are. The tutorial will add the driver Robert Durden, with national number 222222.

```
{
  "$class": "nl.amis.registry.persons.Driver",
  "nationalNumber": "222222",
  "firstName": "Robert",
  "lastName": "Durden",
  "driverLicense": "42",
  "phoneNumber": "0032472830240"
}
```

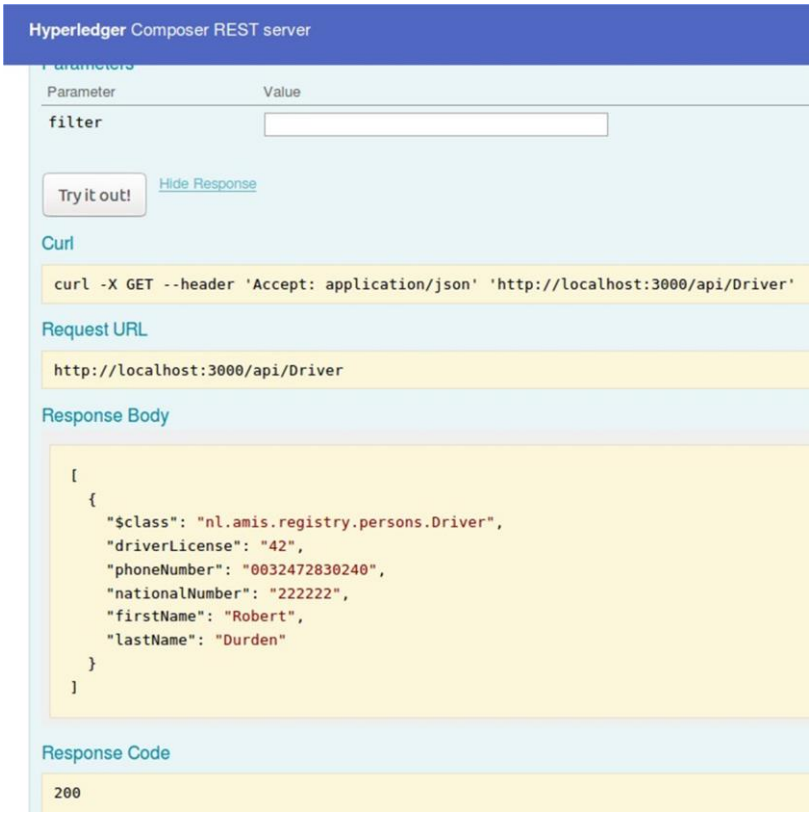
➔ *Click on Try it out!*



You should see a HTTP 200 Response, meaning that the POST was successful. Test whether Robert Durden was added correctly.

➔ *Click on GET and then on Try it out!*

You should see a JSON response with Robert Durden's data.



Now it's your turn to add more data: add a CaseWorker Tyler Paulson, national number 111111 and phone number 0032468470436. Add a Person Marcus Aurelius, national number 333333. Add a Responder John Doe, national number 444444, batchNumber 0. You are of course free to invent other names. Time to add the Emergency Service and the Insurance Company.

```
{
  "$class": "nl.amis.registry.companies.EmergencyService",
  "responders": [
    "resource:nl.amis.registry.persons.Responder#444444"
  ],
  "name": "ResQ"
}
```

The Insurance company is called BeSure and its contact person is Marcus Aurelius, add this one yourself. At the moment Robert Durden is a vehicle-less driver, so add a vehicle using him as an owner, insured by BeSure:

```
{
  "$class": "nl.amis.registry.vehicles.Vehicle",
  "vin": "1",
  "brand": "SomeBrand",
  "licensePlate": "0-GXS-248",
  "insurer":
"resource:nl.amis.registry.companies.InsuranceCompany#BeSure",
  "owner": "resource:nl.amis.registry.persons.Driver#222222"
}
```

Great, you've added a Driver, an InsuranceCompany and its contact Person and an EmergencyService with a Responder, and a CaseWorker. You now have the necessary elements to register an Accident that will happen to Robert Durden, and process it. First you could verify that the vehicle and companies have been registered correctly.

➔ *Go to http://localhost:4200*

This is your Angular web application. It is not nearly as functional as the REST API at port 3000 but more easy-view. Click on assets in the top right corner and verify that there is no accident, but there are the companies and there is the vehicle you've registered in the REST API.

➔ *Go back to the REST API at http://localhost:3000*

Now is the time to create the registry of an Accident that has somehow happened to Robert Durden. Using the REST API there are two ways to do this, the first is by directly creating an Accident in the registry, the second is by calling the transaction that participants will use. This tutorial will call the transaction that will create an Accident in the registry.

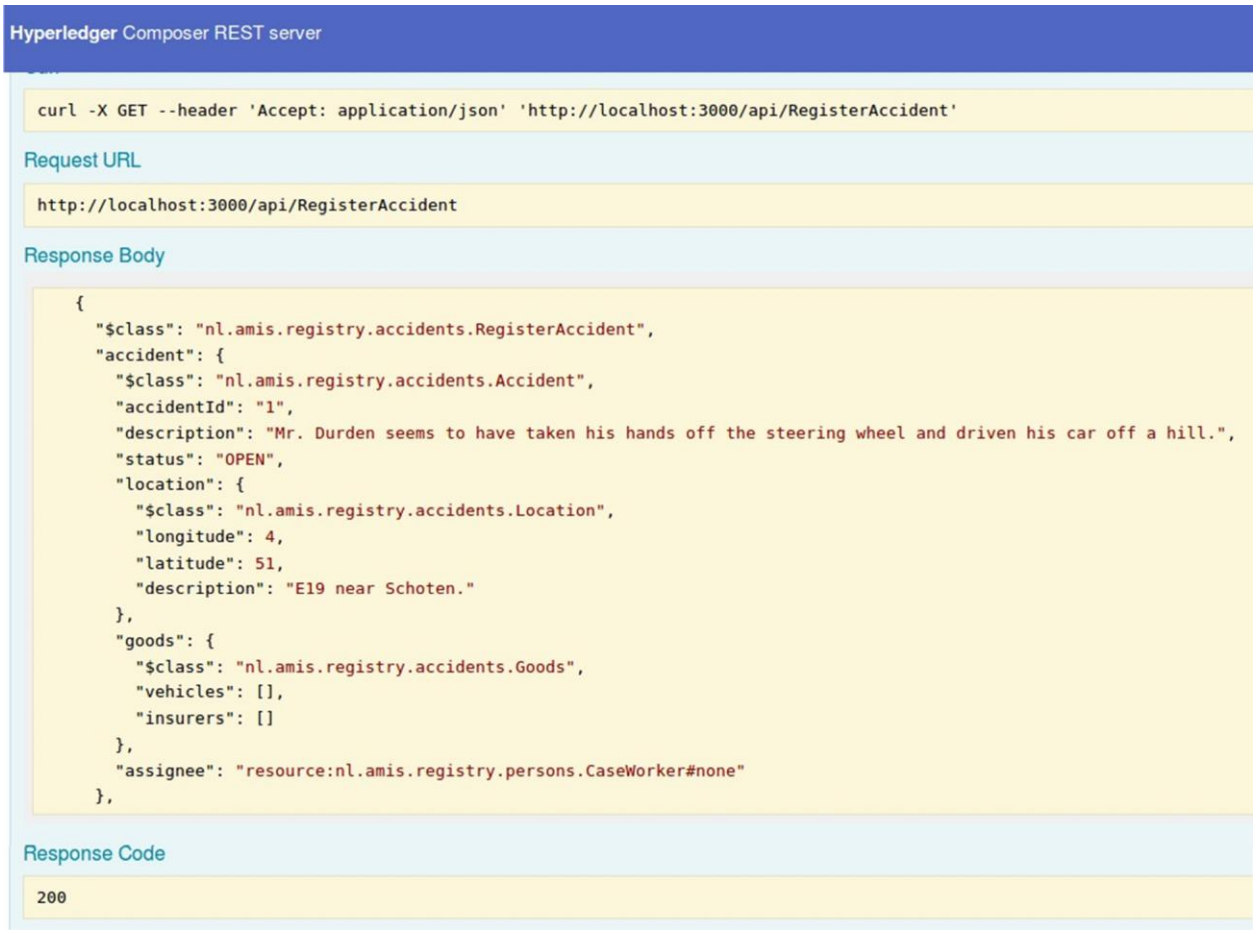
➔ *Click on RegisterAccident*

Now the JSON you need to enter is more complex than the previous JSONs. Notice that this actually creates a registry of the transaction, while the transaction itself is what creates an Accident in the Accident registry. The transactionId will be automatically generated so you can leave it empty. The timestamp however is not generated automatically. You will need to supply a value. The easiest way to get a valid and current timestamp is to copy the timestamp shown in the Example Value JSON. The example timestamp should look something like "2016-09-27T03:48:12.642Z".

```
{
  "$class": "nl.amis.registry.accidents.RegisterAccident",
  "accident": {
    "$class": "nl.amis.registry.accidents.Accident",
    "accidentId": "1",
    "description": "Mr. Durden seems to have taken his hands off the
steering wheel and driven his car off a hill.",
    "status": "OPEN",
    "location": {
      "$class": "nl.amis.registry.accidents.Location",
      "longitude": 4.31,
      "latitude": 51.18,
      "description": "E19 near Schoten."
    },
    "goods": {
      "$class": "nl.amis.registry.accidents.Goods",
      "vehicles": [],
      "insurers": []
    },
    "assignee": "none"
  },
  "transactionId": "",
  "timestamp": "2017-11-03T15:16:08.860Z"
}
```

➔ *Click on Try it out!*

It should give a HTTP 200 Response Code. Verify whether the accident has correctly been registered using the REST API or the web application (http://localhost:4200).



Great, now that there is a registered accident, you can begin handling it. The first step is to assign the case to a CaseWorker.

➔ *Click on AssignToCaseWorker*

```
{
  "$class": "nl.amis.registry.accidents.AssignToCaseWorker",
  "accident": "1",
  "assignee":
"nl.amis.registry.persons.CaseWorker#nationalNumber:111111",
  "transactionId": "",
  "timestamp": "2017-11-03T15:16:08.860Z "
}
```

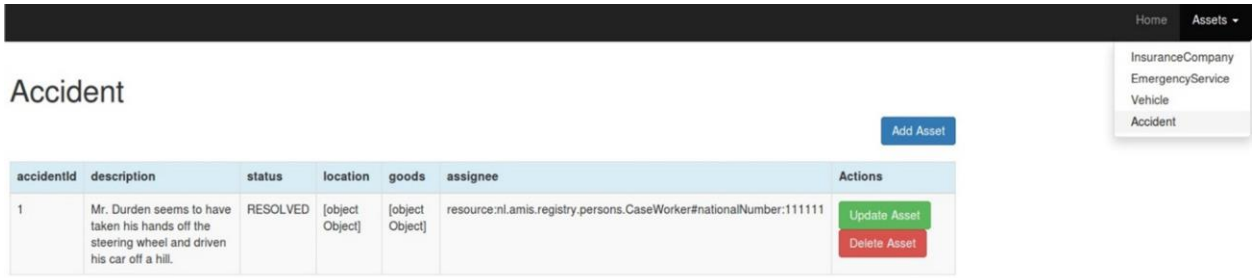
Now that a CaseWorker has been assigned, you can send the case to the InsuranceCompany BeSure.

➔ *Click on SendToInsurance*

This time figure out the JSON yourself. Use the example value given by the REST API to help you out. Once you've send the case to an InsuranceCompany there's only one step left: to invoke ResolveAccident which will list the case as resolved.

➔ *Click on ResolveAccident*

Now, browse to your web application (http://localhost:4200) and verify that your accident is listed correctly under Accidents and that the case has been RESOLVED. It should look like this if you have followed this tutorial to the letter:



Congratulations on finishing the tutorial! You've successfully created your very own Blockchain solution and added data to it. You've created a REST API to communicate with the Blockchain and made a rudimentary web application to access the REST API. If you have time left, you could try to expand or improve our example Car Accident network, or think of your own!

If you would wish to shut down the Hyperledger Fabric, run ./stopFabric.sh from /home/fabric-tools, and if you would wish to completely erase the network (destroying all data you've entered during this tutorial) run ./teardownFabric.sh from the same directory.