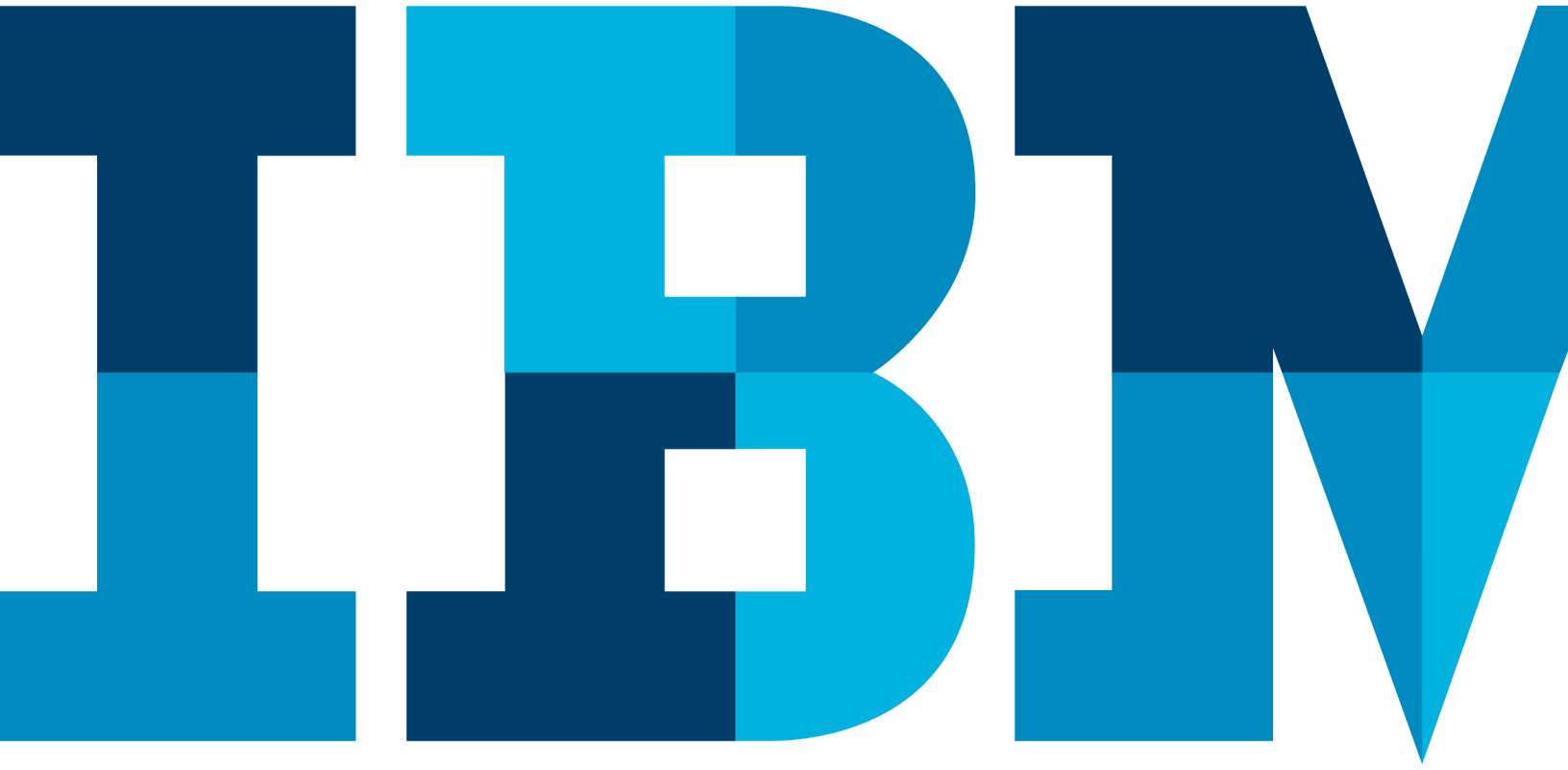
IBM Blockchain
Hyperledger Fabric Hands-On

Develop a smart contract in NodeJS with IBM Blockchain Platform (Hyperledger Fabric 1.4) development tools

Lab



IBM

1 Contents

1 Contents 2

2 Overview 3

2.1 Introduction 3

2.2 Preparation 3

3 SmartContract development and deployment 5

3.1 Create a new SmartContract project 5

4 My first SmartContract 7

4.1 SmartContract folders 7

4.2 SmartContract skeleton 7

5 Implementation of the business use case 12

5.1 Use case 12

5.2 Transactions implementation 12

5.3 Privacy implementation (security) 25

6 Deployment and test on the IBM Blockchain Platform on IBM Cloud 30

6.1 Deploy the SmartContract 30

6.2 Connect the development environment (VSCode) to your IBP2.0 34

Notices 45

Appendix A. Trademarks and copyrights 47

2 Overview

The aim of this lab is to explore the development of SmartContracts (Hyperledger Fabric 1.4) using the IBM Blockchain Platform development tools.

2.1 Introduction

This lab consists in using Visual Studio Code with the IBM Blockchain Platform extension as the IDE for the development and test of a SmartContract.
It requires

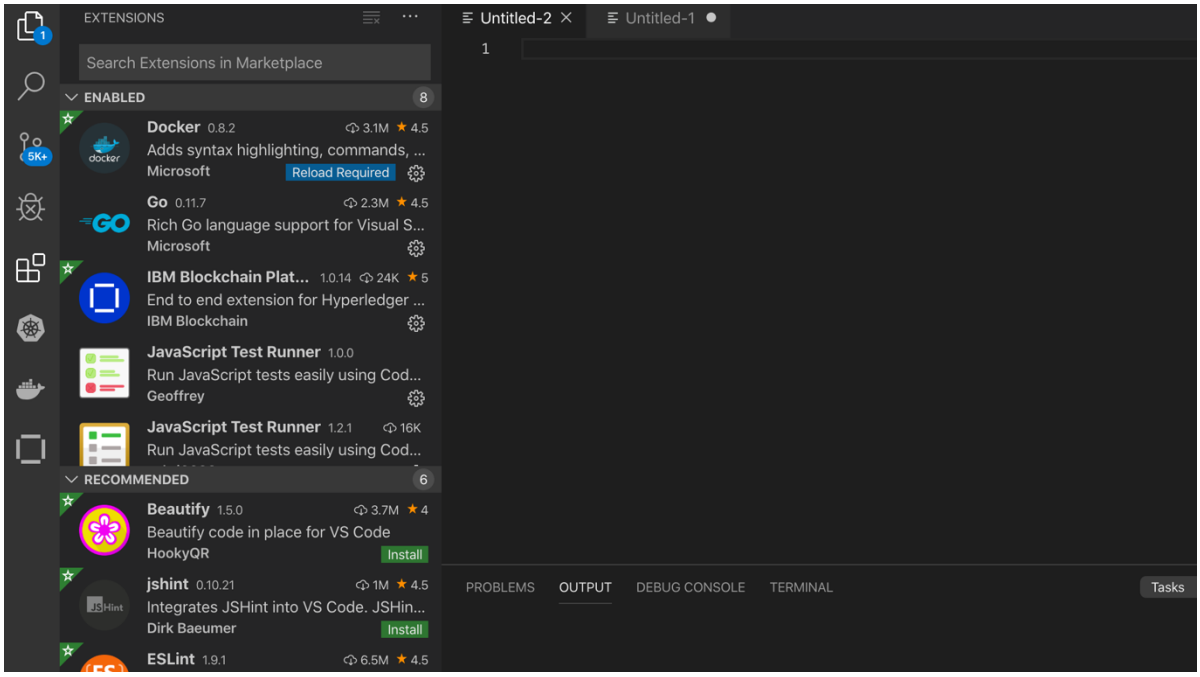
- Nodejs,
- Docker and Docker-compose
- Visual Studio Code.

Either you have installed Visual Studio Code and the prerequisites on your workstation either you will use a VM provided for the lab.
In this last case, you are provided an OpenVPN certificate. You need to install an OpenVPN client and start the VPN connection using the certificate. Then use Microsoft Remote Desktop to access remotely to the VM.

2.2 Preparation

Visual Studio Code (VSC) is the IDE for the blockchain solution development.
First, the extensions required for the development should be added.
Start Visual Studio Code:

In the left side of the VSC windows, click on the Extensions icon (the white square in the following picture), then in the search field type “blockchain” as shown in the following picture.
Then click on the Install button of the IBM Blockchain Platform extension.

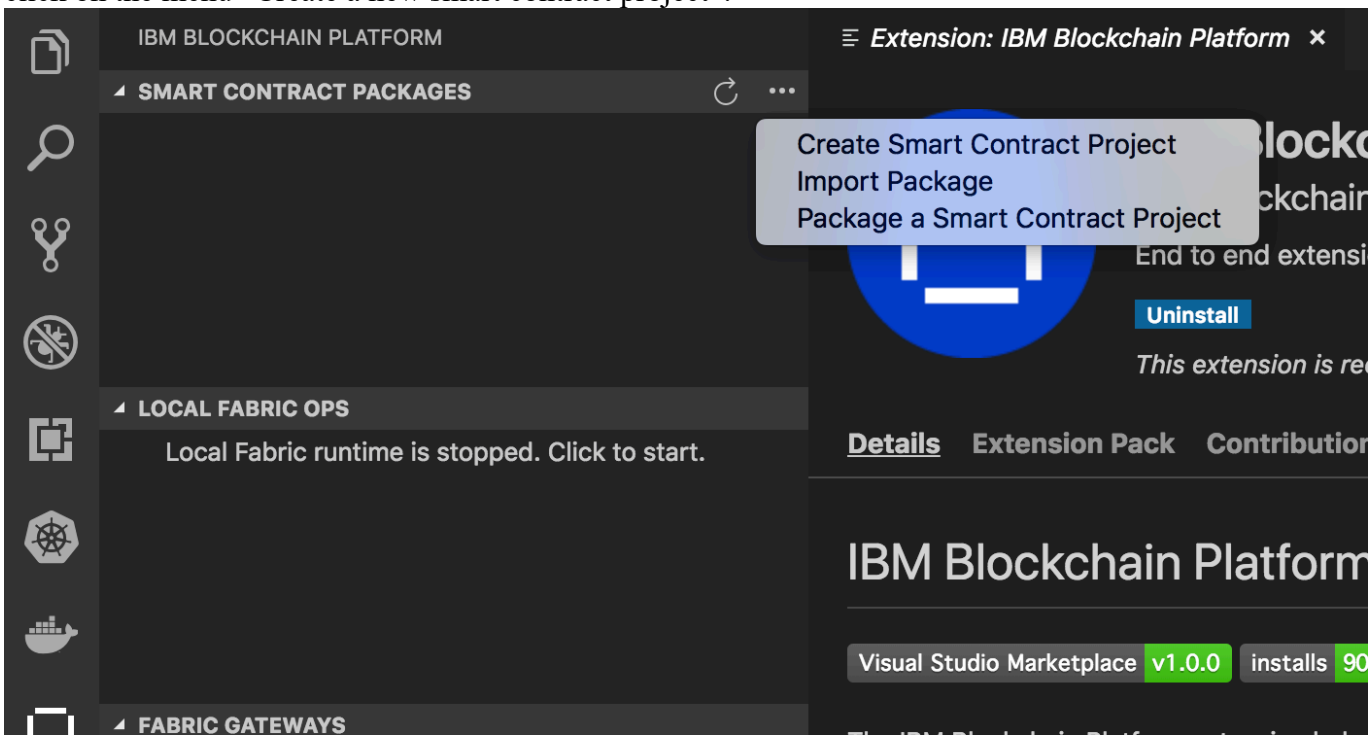


Now you are ready to create your SmartContract.

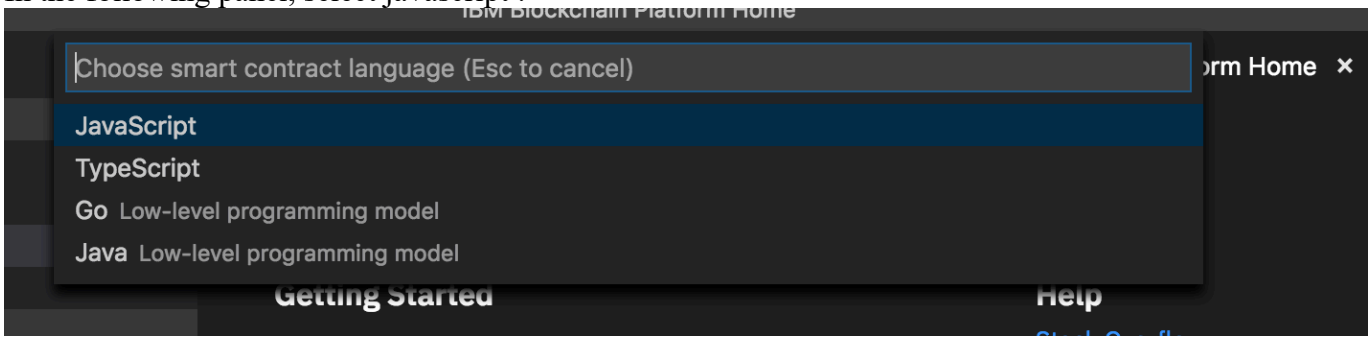
3 SmartContract development and deployment

3.1 Create a new SmartContract project

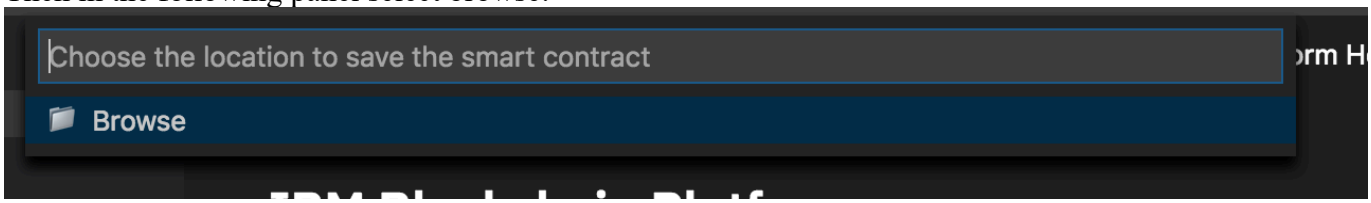
In VSC, click on the blockchain button on the left side menu to access to the Blockchain workspace. Then click on the menu (...)on the SMART CONTRACT PACKAGES frame, at the top left side, then click on the menu “Create a new smart contract project”:



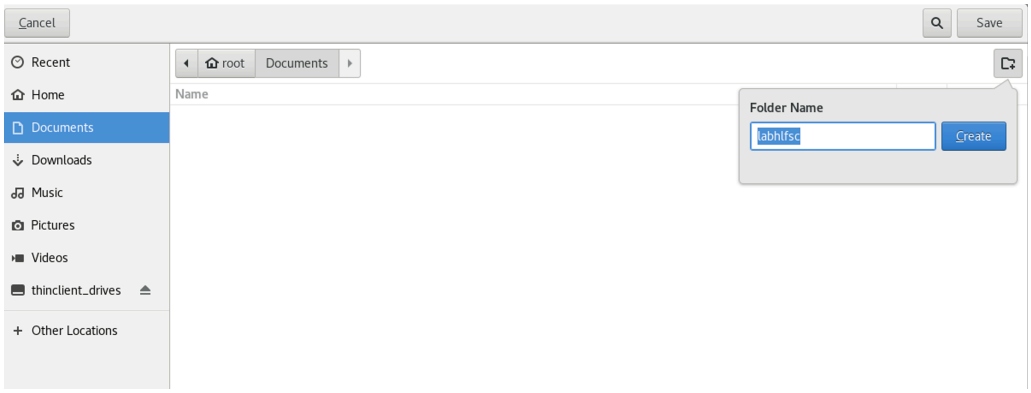
In the following panel, select javascript :



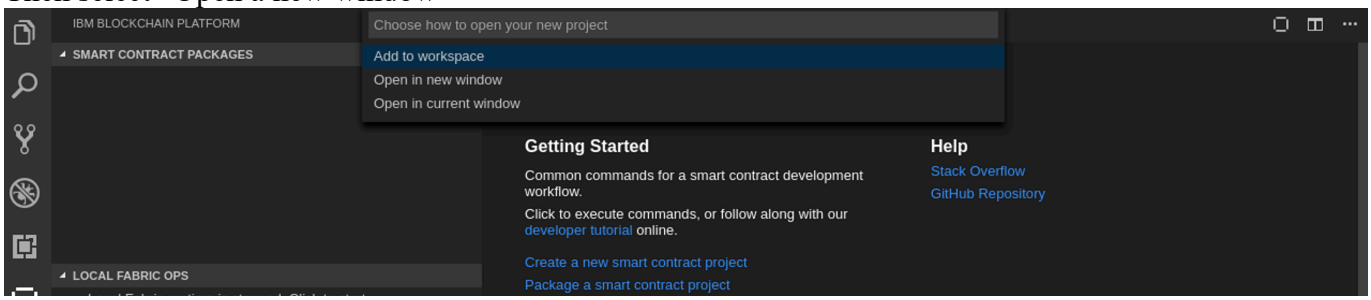
Then in the following panel, put the name of the asset: Pack
Then in the following panel select browse:



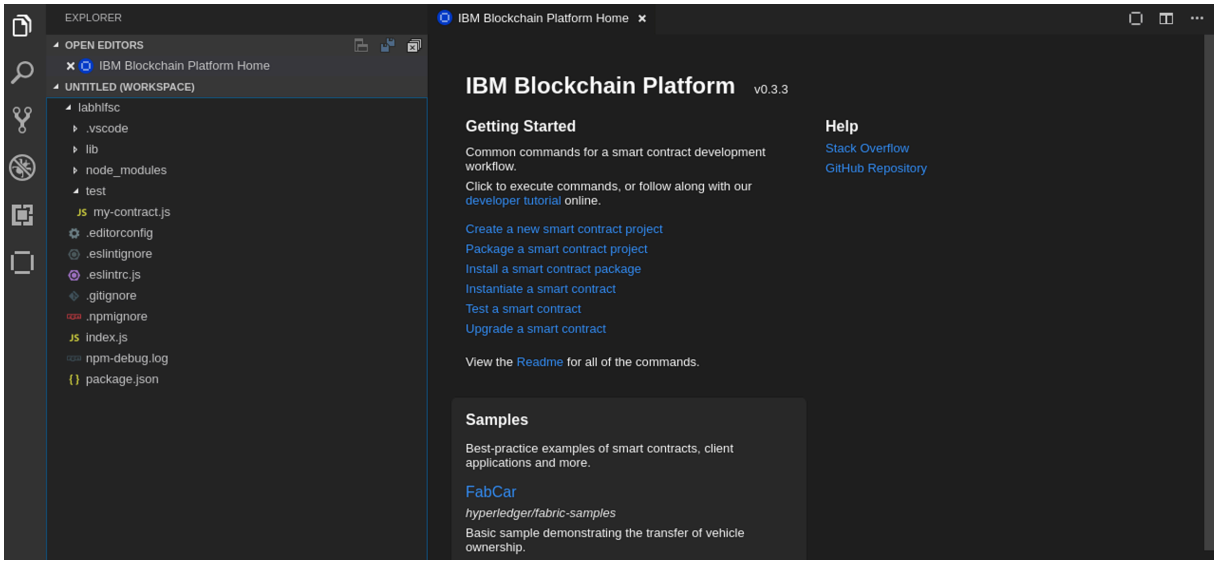
Then in the following panel, select document as folder.
Then click on the ‘add folder’ button to add a folder and type the folder name “ibp2sc”:



Click Create then Save.
Then select “Open a new window”



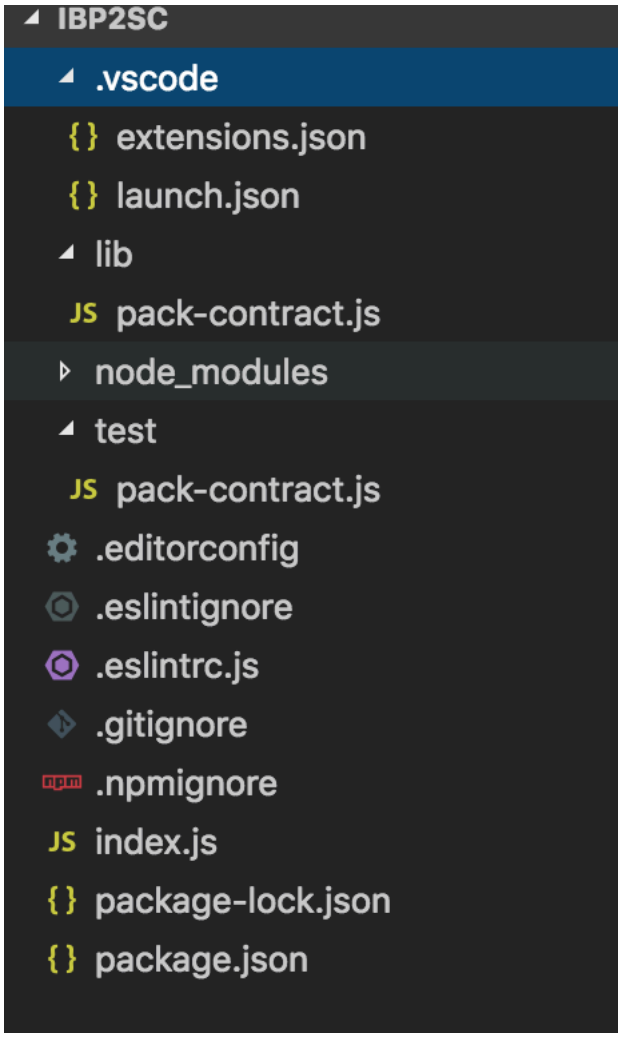
You have created the SmartContract skeleton.



4 My first SmartContract

4.1 SmartContract folders

Let's have a look at the generated package. It consists of the following folders/files:



*The folder **lib** consists of the file for the smart contract: pack-contract.js.*
*The folder **node_modules** consists of the node modules required by the smart contract.*
*The folder **test** consists of the file for the unitary test of the smart contract: pack-contract.js.*
*The file **index.js** specifies how to publish the smart contrat.*
*The file **package.json** specifies how to package the smart contract, especially it describes the name and the version of the smart contract that will be used to prepare the smart contract for the deployment.*
*The **.vscode** folder and the other files are for the internal aspect of Visual Studio Code.*

4.2 SmartContract skeleton

Open the smart contract (file **lib/pack-contract.js)**

The skeleton should be the following:

```
/*
 * SPDX-License-Identifier: Apache-2.0
 */

'use strict';

const { Contract } = require('fabric-contract-api');

class PackContract extends Contract {

    async packExists(ctx, packId) {
        const buffer = await ctx.stub.getState(packId);
        return (!!buffer && buffer.length > 0);
    }

    async createPack(ctx, packId, value) {
        const exists = await this.packExists(ctx, packId);
        if (exists) {
            throw new Error(`The pack ${packId} already exists`);
        }
        const asset = { value };
        const buffer = Buffer.from(JSON.stringify(asset));
        await ctx.stub.putState(packId, buffer);
    }

    async readPack(ctx, packId) {
        const exists = await this.packExists(ctx, packId);
        if (!exists) {
            throw new Error(`The pack ${packId} does not exist`);
        }
        const buffer = await ctx.stub.getState(packId);
        const asset = JSON.parse(buffer.toString());
        return asset;
    }

    async updatePack(ctx, packId, newValue) {
        const exists = await this.packExists(ctx, packId);
        if (!exists) {
            throw new Error(`The pack ${packId} does not exist`);
        }
        const asset = { value: newValue };
        const buffer = Buffer.from(JSON.stringify(asset));
        await ctx.stub.putState(packId, buffer);
    }

    async deletePack(ctx, packId) {
```

```
        const exists = await this.packExists(ctx, packId);
        if (!exists) {
            throw new Error(`The pack ${packId} does not exist`);
        }
        await ctx.stub.deleteState(packId);
    }

}

module.exports = PackContract;
```

This skeleton proposes functions to manage the asset Pack, some functions which are updating the ledger are transactions.
This give you an example and the basis to develop a smart contract handling an asset.

Open the **test script** of the smart contract (file **test/pack-contract.js)** :

```
/*
 * SPDX-License-Identifier: Apache-2.0
 */

'use strict';

const { ChaincodeStub, ClientIdentity } = require('fabric-shim');
const { PackContract } = require('..');
const winston = require('winston');

const chai = require('chai');
const chaiAsPromised = require('chai-as-promised');
const sinon = require('sinon');
const sinonChai = require('sinon-chai');

chai.should();
chai.use(chaiAsPromised);
chai.use(sinonChai);

class TestContext {

    constructor() {
        this.stub = sinon.createStubInstance(ChaincodeStub);
        this.clientIdentity = sinon.createStubInstance(ClientIdentity);
        this.logging = {
            getLogger:
sinon.stub().returns(sinon.createStubInstance(winston.createLogger().constructor)),
            setLevel: sinon.stub(),
        };
    }
```

```
}

describe('PackContract', () => {

    let contract;
    let ctx;

    beforeEach(() => {
        contract = new PackContract();
        ctx = new TestContext();
        ctx.stub.getState.withArgs('1001').resolves(Buffer.from('{"value":"pack 1001
value"}'));
        ctx.stub.getState.withArgs('1002').resolves(Buffer.from('{"value":"pack 1002
value"}'));
    });

    describe('#packExists', () => {

        it('should return true for a pack', async () => {
            await contract.packExists(ctx, '1001').should.eventually.be.true;
        });

        it('should return false for a pack that does not exist', async () => {
            await contract.packExists(ctx, '1003').should.eventually.be.false;
        });

    });

    describe('#createPack', () => {

        it('should create a pack', async () => {
            await contract.createPack(ctx, '1003', 'pack 1003 value');
            ctx.stub.putState.should.have.been.calledOnceWithExactly('1003',
Buffer.from('{"value":"pack 1003 value"}'));
        });

        it('should throw an error for a pack that already exists', async () => {
            await contract.createPack(ctx, '1001', 'myvalue').should.be.rejectedWith(/The pack
1001 already exists/);
        });

    });
```

This script tests the functions implementing the transactions of the smart contract. Each function can be tested independently.

When moving the mouse above the “describe(…” instructions you will see 2 links “Run Test | Debug test”.
So to test the full smart contract, click on “Run test” above the first describe instruction :

5 Implementation of the business use case

5.1 Use case

A consumer buys a product to Company A.
Company A packs the product and asks to Company B to ship the product.
Company A hands over the package to Company B.
Company B ships the package to the consumer location.
The consumer receives the package and acknowledges the reception of the product.

The problem to solve is the tracking of the product shipment.

We are creating one asset, the package with the following attributes
- Identifier
- Description
- Status (Ready, Shipped, Delivered, Received)
- Destination

The transactions will be:
- *OrderShipment*: the supplier (from Company A), once the order received from the consumer and then once the product packed, orders the shipment to the carrier (Company B)
- *Ship(PackageId, status):* the carrier (Company B) accounts the package corresponding to the order shipment, then provide the status of the shipment (Ready, Shipped, Delivered).
- *Acknowledgement(PackageId):* the consumer, as soon as he has received the package, acknowledges the reception. Then the status move to Received.
- *GetPackageStatus(PackageId):* provides the status of the shipment (Ready, Shipped, Delivered, Received).

5.2 Transactions implementation

We are updating the smart contract to take into account the previous use case. Then, we will complete the test module accordingly to validate the implementation.

5.2.1 Implement the OrderShipment function

The OrderShipment consists in the creation of an asset Pack (package) with the status READY.
An asset is stored in the Ledger in a Database also called Worldstate. The asset is stored in JSON format as the value of a (key:value) record.

To store assets in the worldstate ledger, the Hyperledger Fabric API provides the following function:

`PutState(key string, value []byte)` will create or will update a worldstate entry at the key attribute using the value passed as second parameter.

Open the smart contract file (file **lib/pack-contract.js).**

Create the orderShipment transaction : add the following code after the last function :

```
    //Create an order for shipping the package
    async orderShipment(ctx, packageID, description, destination) {
        const pack = {
            packageID,
            description,
            status: 'READY',
            destination
        };

        console.info('****OrderShipment>>>>>', JSON.stringify(pack));
        const exists = await this.createPack(ctx, packageID, pack);
        //await ctx.stub.putState(packageID, Buffer.from(JSON.stringify(pack)));
        console.info('****OrderShipment<<<<<');
    }
```

In this function, we are using the createPack function in order to create the package in the ledger.
Save the file.

5.2.2 Update the unitary test file

Open the test file and add the following block at the end of the file before the last "});" :

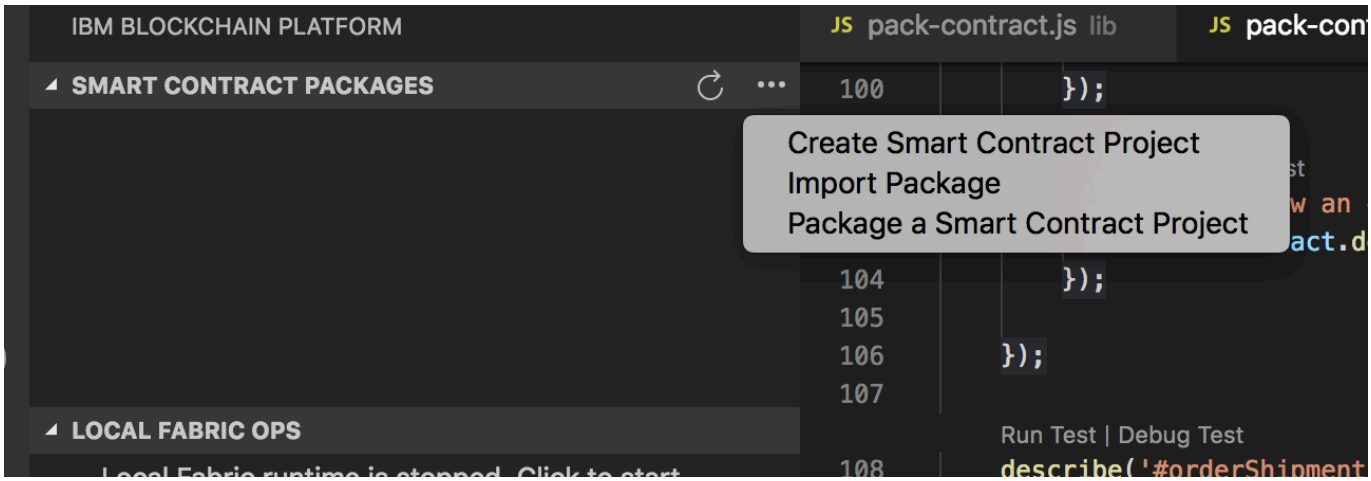
```
   describe('#orderShipment', () => {

        it('should order a shipment', async () => {
            await contract.orderShipment(ctx, 'P001', 'Package number one', 'rue de la vieille
Poste, Montpellier');
        });

    });
```

Then test the smart contract (click on "Run test" on top of "describe('#orderShipment'…").
You should get:

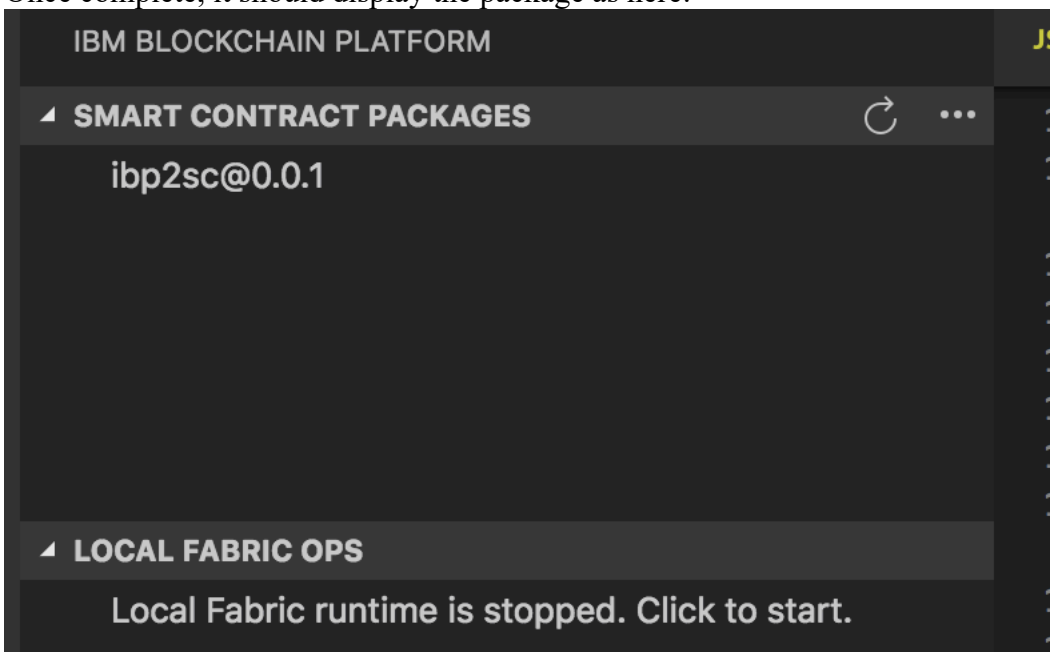
```
  PackContract
    #orderShipment
****OrderShipment>>>>> {"packageID":"P001","description":"Package number one","status":"READY","destination":"rue d
e la vieille Poste, Montpellier"}
****OrderShipment<<<<<
      ✓ should order a shipment


  1 passing (49ms)
```

5.2.3 Prepare the package for the deployment

Open the IBM Blockchain Platform view (click on the blockchain button on the left side menu to access to the Blockchain view).
Click on the menu of the SMART CONTRACT PACKAGES



Click on “Package a Smart Contract Project” menu. (if you have several folders in your workspace, it should ask you to choose the one with the smart contract, else it takes by default the one opened).
Once complete, it should display the package as here:

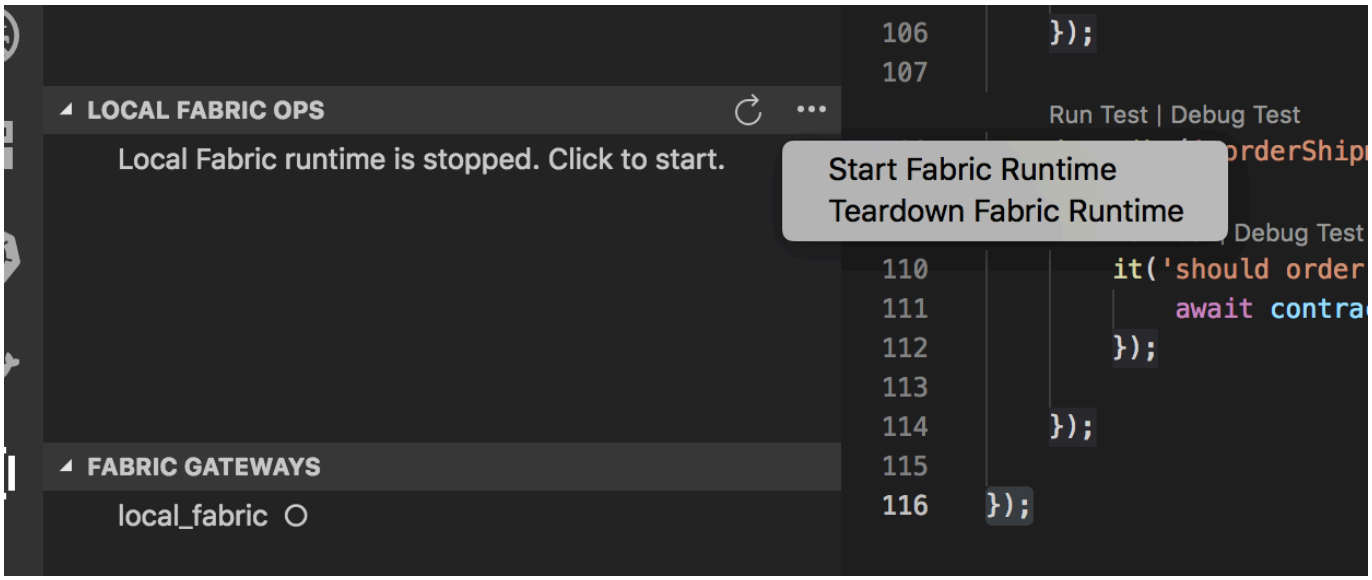


5.2.4 Deploy the smart contract in the local Fabric

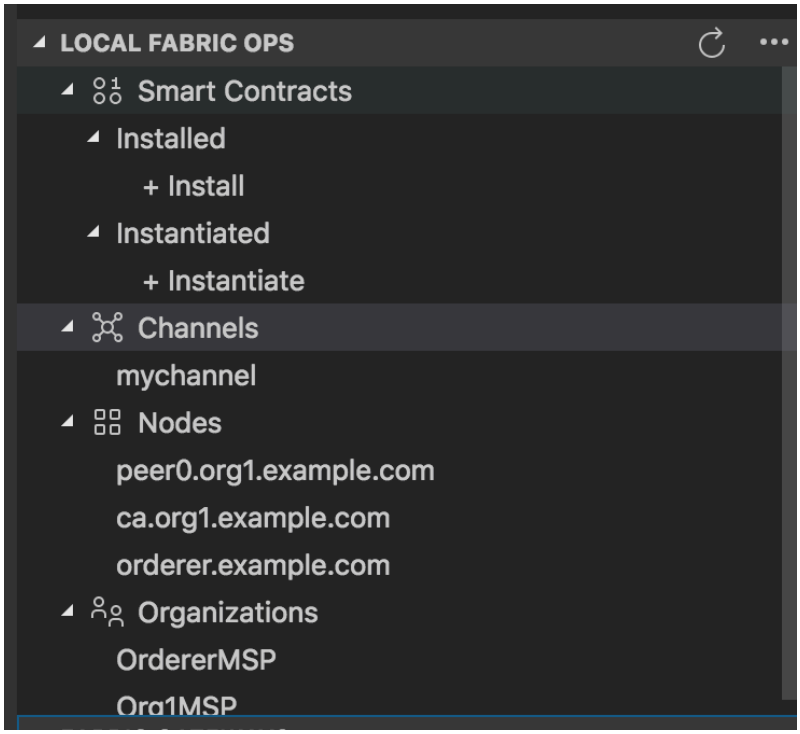
The IBP extension for Visual Studio Code embeds a local Hyperledger Fabric environment which allows to deploy and test the smart contract/blockchain solution locally.

In the following step, we will start the local Fabric, install the Smart Contract in this Fabric, instantiate the smart contract, and test the transactions.

First **start the local Fabric**:
- in the IBP view, click on the ... menu on the **LOCAL FABRIL OPS** line,
- then click on the menu **Start Fabric Runtime**:

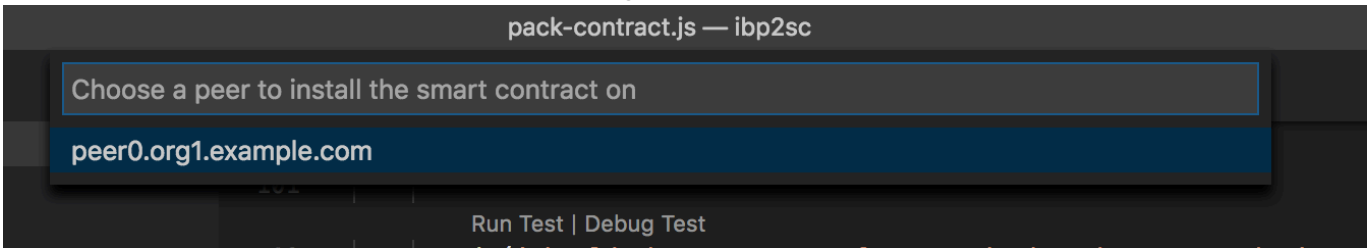


Once the local fabric is started you should see the following panel describing the fabric environment. If you collapse all the folder, you should see this:
- the list of Smart Contracts (installed or installed)
- the list of channels
- the list of nodes
- the list of organizations

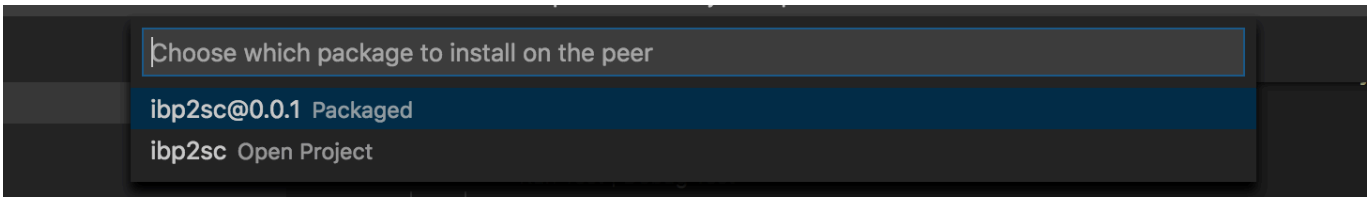


Install the smart contract:

- In the **LOCAL FRABRIC OPS** frame, under **Installed** branch, click on **"+ Install**"
- then choose the peer on which install the package

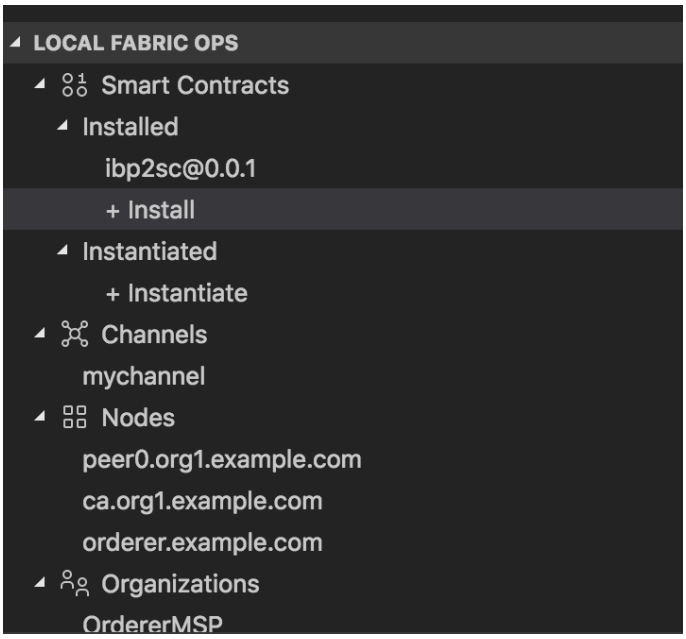


- then select the package previously prepared: ibp2sc@0.0.1



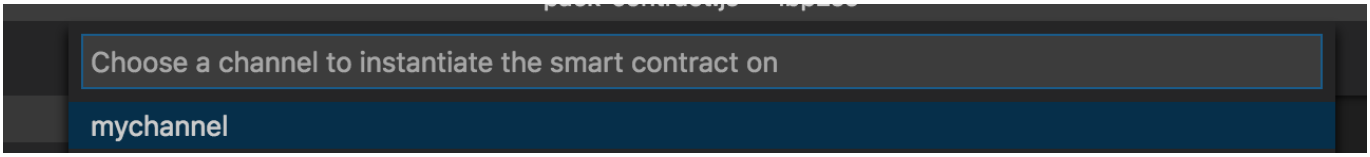
Notice you could select the project ibp2sc; in this case, VSC would have created the package first, before installing it.

- Once installed, it appears in the list of installed packages, in the local fabric.

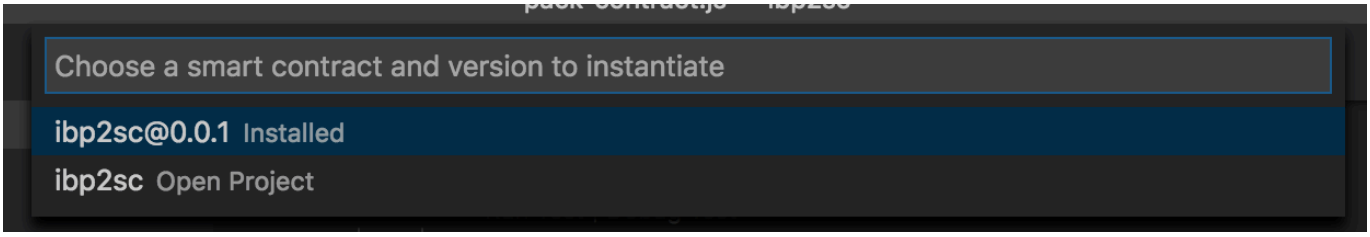


Instantiate the smart contract:

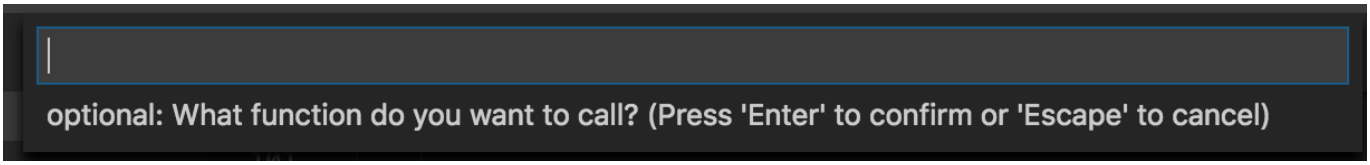
- Click on the “+instantiate” in the local Fabric
- Then select the default channel (mychannel) of the local Fabric



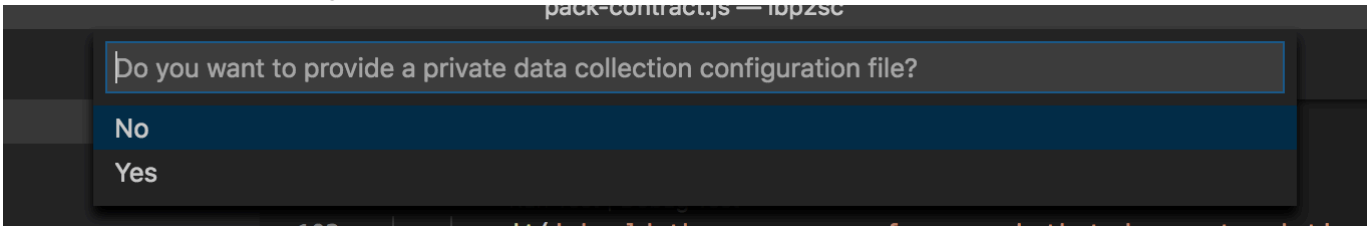
- Select the smart contract previously installed



- Then it propose to call a function of the smart contract at the instantiation step. You can use this way to initialize something in the ledger for example. In our case, no need to initialize anything, so leave the field blank and press enter

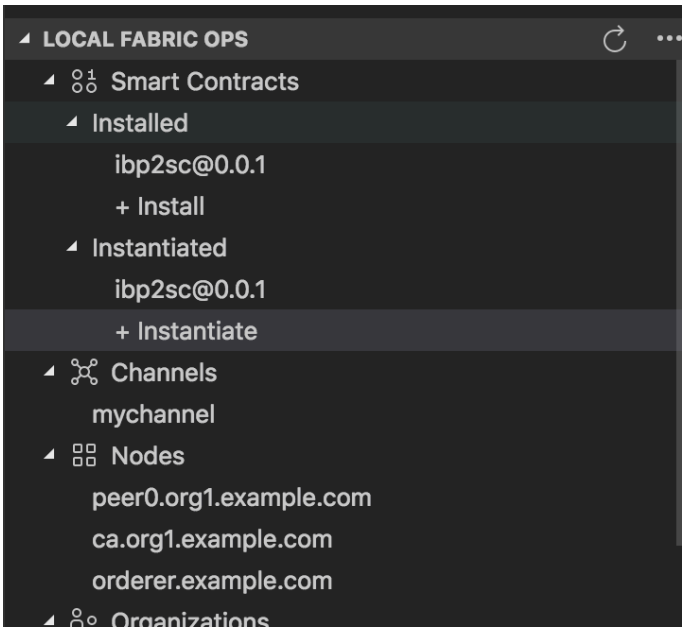


- Then it proposes to configure the private data collection; in our case, no need; answer No



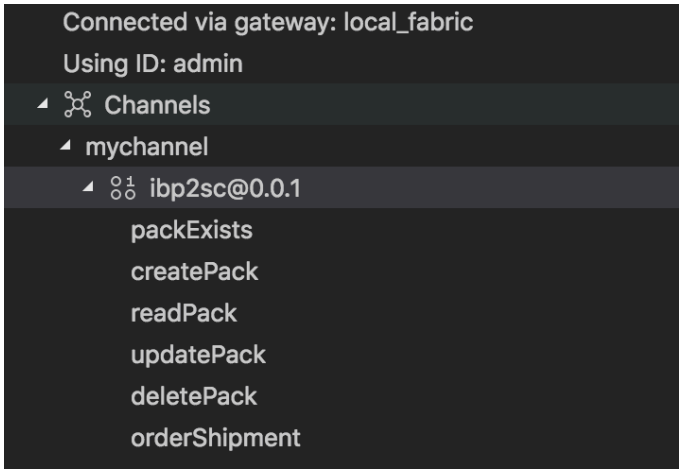
-

- Once instantiated, it will appear in the list of instantiated packages, in the local fabric.



Test the smart contract transactions: you can test directly the smart contract function without client application, through the local_fabric gateway:

- In the **FABRIC GATEWAYS** frame, click on local_fabric>Admin@Org1.example.com
- Then expand the tree under Channels; then you will get the list of function of the ibp2sc@0.0.1 smart contract



- To test a function right click on it; you will have the option to evaluate or submit the transaction. For example right-click on orderShipment and select **Submit transaction.** In the argument field, fill in the arguments required by the orderShipment transaction : ["P001", "Package number one", "rue de la vieille Poste, Montpellier"] then press Enter

You should see the following log on the Output view:

```
[5/13/2019 5:58:58 PM] [INFO] submitTransaction
[5/13/2019 6:00:05 PM] [INFO] submitting transaction orderShipment with args P001,Package
number one,rue de la vieille Poste, Montpellier
[5/13/2019 6:00:07 PM] [SUCCESS] No value returned from orderShipment
```

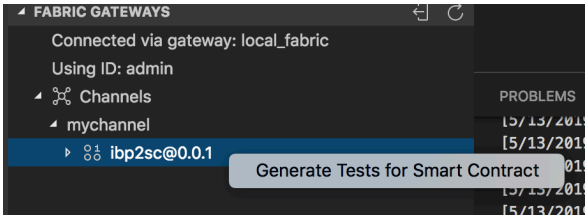
To check the content of the ledger, do the same with the readPack function with argument : ["P001"]
You should get the following result:

```
[5/13/2019 6:02:02 PM] [INFO] submitTransaction
[5/13/2019 6:02:37 PM] [INFO] submitting transaction readPack with args P001
[5/13/2019 6:02:40 PM] [SUCCESS] Returned value from readPack:
{"value":{"description":"Package number one","destination":"rue de la vieille Poste,
Montpellier","packageID":"P001","status":"READY"}}
```

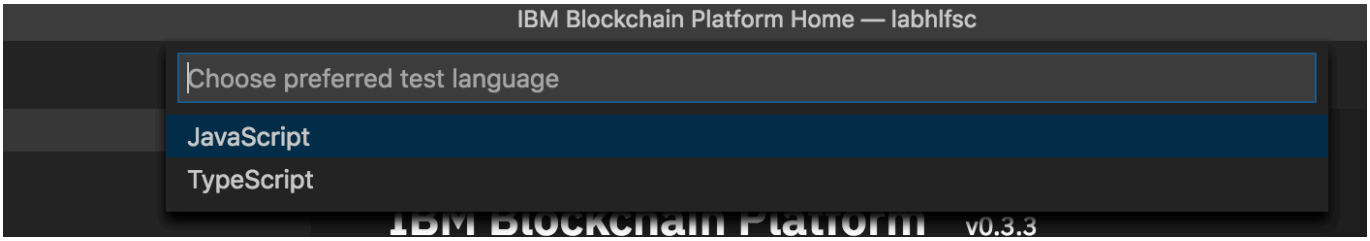
5.2.5 Generate a client application

Now, let's generate a client application which will allow to test the smart contract

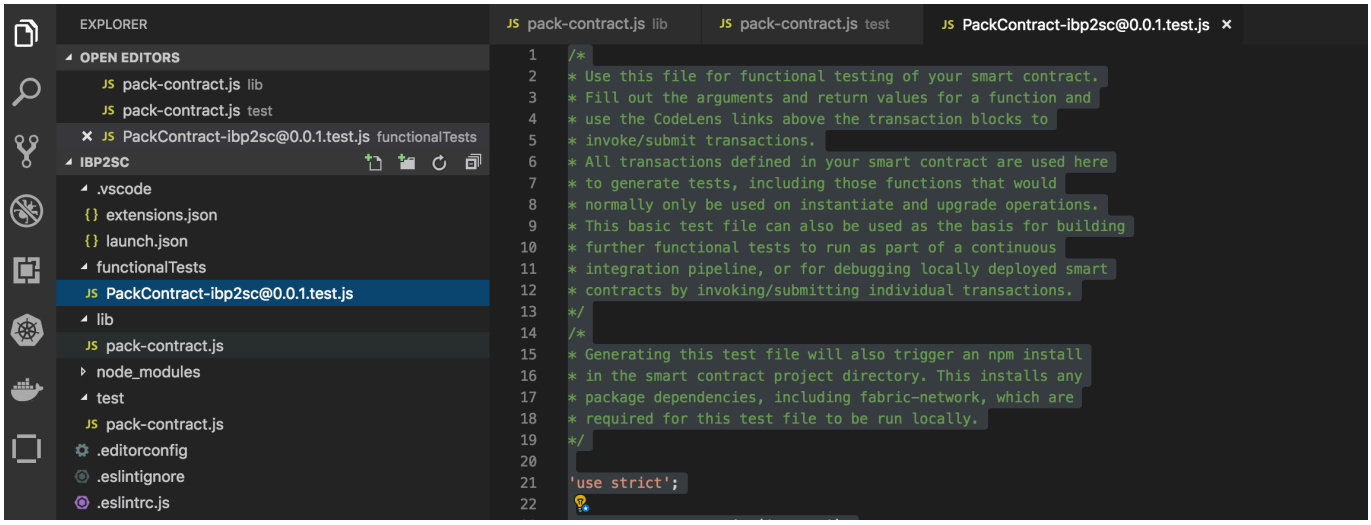
- In the FABRIC GATEWAYS frame, right-click on the smart contract and select the menu Generate Tests for smart contract



- Then choose JavaScript



- it generates an application in the file: functionalTests>PackContract-ibp2sc@0.0.1.test.js



Let’s dig into this application.
First, it retrieves the information of the local fabric in the file .fabric-vscode/runtime/gateways/local_fabric.json to build the **connectionProfile**.
Then it takes the list of identities from the **wallet** (.fabric-vscode/local_fabric_wallet).
It takes the “admin” identity to connect to the fabric (await gateway.connect(connectionProfile, options);).

```
        const homedir = os.homedir();
    const walletPath = path.join(homedir, '.fabric-vscode', 'local_fabric_wallet');
    const gateway = new fabricNetwork.Gateway();
    const wallet = new fabricNetwork.FileSystemWallet(walletPath);
    const identityName = 'admin';
    let connectionProfile;

    before(async () => {
        const connectionProfilePath = path.join(homedir, '.fabric-vscode', 'runtime',
'gateways', 'local_fabric.json');
        const connectionProfileContents = await fs.readFile(connectionProfilePath, 'utf8');
        if (connectionProfilePath.endsWith('.json')) {
            connectionProfile = JSON.parse(connectionProfileContents);
        } else if (connectionProfilePath.endsWith('.yaml') ||
connectionProfilePath.endsWith('.yml')) {
            connectionProfile = yaml.safeLoad(connectionProfileContents);
        };
    });

    beforeEach(async () => {

        const discoveryAsLocalhost = hasLocalhostURLs(connectionProfile);
        const discoveryEnabled = true;
```

```
        const options = {
            wallet: wallet,
            identity: identityName,
            discovery: {
                asLocalhost: discoveryAsLocalhost,
                enabled: discoveryEnabled
            }
        };

        await gateway.connect(connectionProfile, options);
    });

    afterEach(async () => {
        gateway.disconnect();
    });
```

In the following, it runs the transactions.
You have to update the < const args = [];> instruction with the appropriate arguments of the transaction (cf the previous paragraph).

```
    it('orderShipment', async () => {
        // TODO: Update with parameters of transaction
        const args = [];

        const response = await submitTransaction('orderShipment', args); // Returns buffer of
transaction return value
        // TODO: Update with return value of transaction
        // assert.equal(JSON.parse(response.toString()), undefined);
    }).timeout(10000);

    async function submitTransaction(functionName, args) {
        // Submit transaction
        const network = await gateway.getNetwork('mychannel');
        const contract = await network.getContract('ibp2sc', 'PackContract');
        const responseBuffer = await contract.submitTransaction(functionName, ...args);
        return responseBuffer;
    }
```

Update the arguments of the orderShipment test.
You can run the test

5.2.6 Implement the GetPackageStatus function

The GetPackageStatus consist in retrieving the package according to a packageid, then return the status of this package.

To retrieve an asset from the ledger, the Hyperledger Fabric API provides the following function:

`GetState(key string)` *will retrieve an asset according to the key in parameter.*

Open the smart contract file (file **lib/pack-contract.js)** and add the following function at the end of the file before the last curly bracket. Save the file

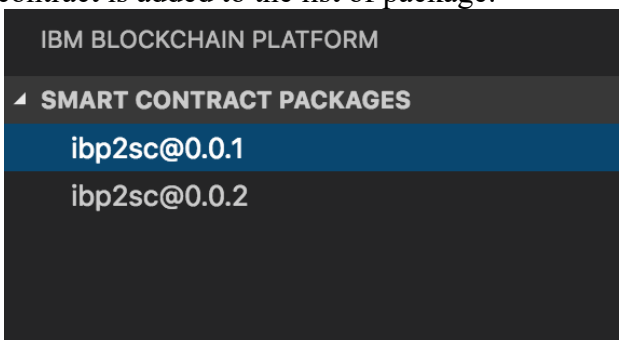
```
  // Retrieve the status of a Package
async getPackageStatus(ctx, packageID) {
   console.info('****GetPackageStatus>>>>>', packageID);
   let packAsBytes = await ctx.stub.getState(key);
   let pack = JSON.parse(packAsBytes);
   console.info('****GetPackageStatus<<<<<', JSON.stringify(pack));
   return pack.status;
 }
```

5.2.7 Update the smart contract

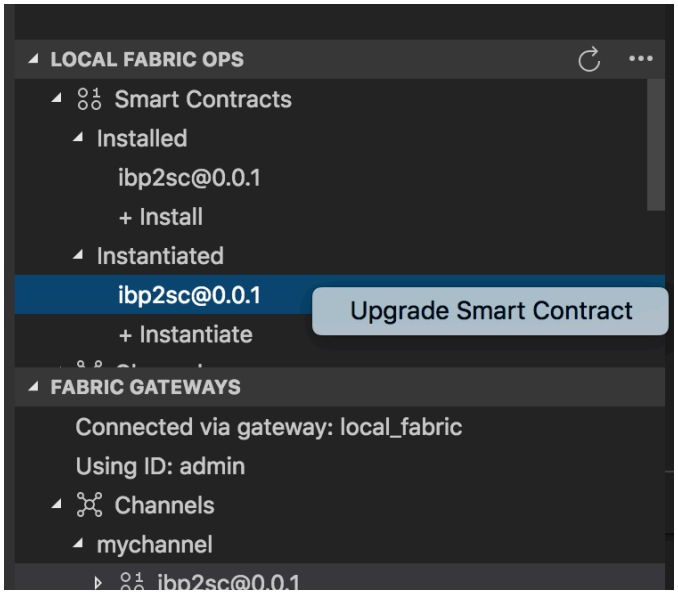
Open the file package.json and increment the version from 0.0.1 to 0.0.2. Save the file

```
   "name": "ibp2sc",
   "version": "0.0.2",
   "description": "My Smart Contract",
```

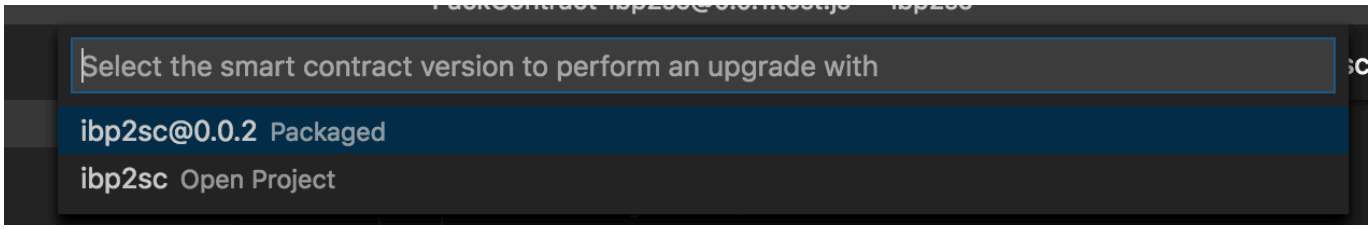
Then go in the IBP view and click on Package a smart contract project. A second version of the smart contract is added to the list of package:



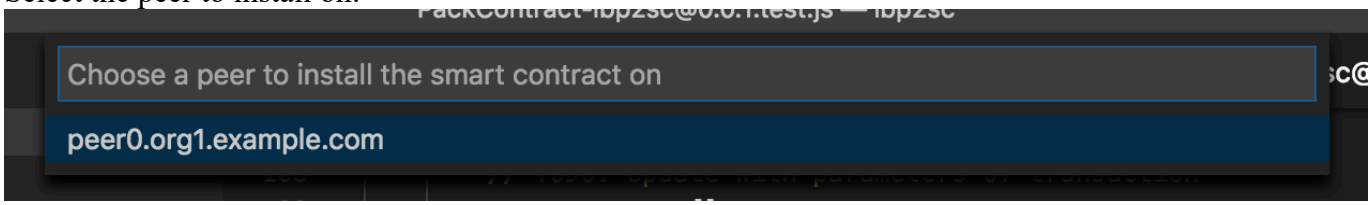
Now, lets update the smart contract. Right-click on the instantiated smart contract and click on the menu “upgrade Smart Contract”



Select the package with which it will be upgraded (…@0.0.2):



Select the peer to install on:



Then leave blank the function to call, press enter, then answer No to the question relative to the private data.

Notice that the instantiated smart contract is now ibp2sc@0.0.2

Notice that you could directly upgrade the smart contract without going through the installation step:

- Right click on the instantiated smart contract,
- Then select Upgrade,
- Then select the project instead of a smart contract package (pay attention to have previously updated the smart contract version in the package.json file)
- Then VSC create the package, install it and instantiate it.

5.2.8 Test your smart contract

Generate a new test application as it was done in Generate a client application.
Then update it with the right parameters for both functions orderShipment and getPackageStatus, and the right test of result :

```
    it('OrderShipment', async () => {
        // TODO: Update with parameters of transaction
        const args = ['P004', 'Package number four', 'rue de la vieille Poste, Montpellier' ];
       …
    }).timeout(10000);

    it('getPackageStatus', async () => {
        // TODO: Update with parameters of transaction
        const args = ['P001'];
       …
        // TODO: Update with return value of transaction
        console.log('****GetPackageStatus>>>>>', JSON.parse(response.toString()));
      assert.equal(JSON.parse(response.toString()), 'READY');
    }).timeout(10000);
```

You can run the test. You should get the similar result:

```
$ node_modules/.bin/mocha functionalTests/MyContract-labhlfsc@0.0.3.test.js --
grep="MyContract-labhlfsc@0.0.2"

  MyContract-labhlfsc@0.0.2
2019-04-30T07:36:48.349Z - info: [TransactionEventHandler]: _strategySuccess: strategy success
for transaction "eefc5bacaf60acf44eb5f7c6fdae2c1d4aa80251c6a6176f8ad41d288773bc23"
    ✓ instantiate (2291ms)
2019-04-30T07:36:50.548Z - info: [TransactionEventHandler]: _strategySuccess: strategy success
for transaction "c997e9013ae48d3b4d4ace197d965661b3beddb5eeff1ca966bd78954497ca61"
    ✓ OrderShipment (2181ms)
2019-04-30T07:36:52.744Z - info: [TransactionEventHandler]: _strategySuccess: strategy success
for transaction "bb2b44c546d82aa7ed5d9180d285073f17b188a5d4cc68942ff5537992d9b90a"
****GetPackageStatus>>>>> READY
    ✓ getPackageStatus (2185ms)

  3 passing (7s)
```

5.2.9 Implement Ship and Acknowledgement transactions

Based on the 2 functions already implemented, complete your smart contract with the remaining transactions :

- Ship(PackageId, status): the carrier (Company B) accounts the package corresponding to the order shipment, then provide the status of the shipment (Ready, Shipped, Delivered).
- Acknowledgement(PackageId): the consumer, as soon as he has received the package, acknowledges the reception.

5.3 Privacy implementation (security)

In this section, we are seeing how to add security aspects to our development in order to restrain transaction usage to authorized profiles.

The principle is to specialize the security certificate of the user adding attributes that will allow to determine the profile of the user: supplier, carrier, customer.
We are naming this attribute “userProfile”.

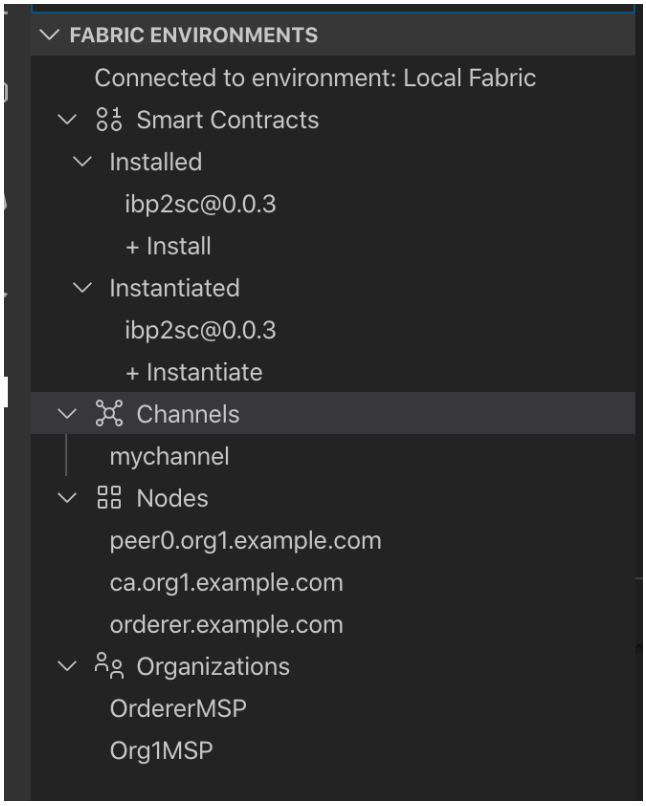
In the transaction, we will test this attribute in order to filter the utilization of the transaction according to the profile. We are also checking the organisation (company) and filtering according it. However, in our test environment, we have a single organization and then we will have the same organization for all the users. In a production environment, the user profile will be distributed between the different organizations.

5.3.1 Implement the privacy in the transactions

We are using the **ClientIdentity** object of the **fabric-shim** package.
The method **assertAttributeValue** will allow to test the value of an attribute.
The method **getMSPID** will allow to retrieve the organization identification in order to filter according to the organization.

Let’s have a look to the fabric environment for the Local Fabric. We observe that in the list of organizations, there are 2 organizations: OrdererMSP and Org1MSP (cf following picture).

The OrdererMSP organization is dedicated to host the Orderer services, while Org1MSP is the the organization that will run the transactions.
In a production environment, we would have an organization for Company A (Org1MSP) and an organization for Company B (Org2MSP). Suppliers and customers would be attached to Company A, while carriers would be to Company B.
Here, we’ll have all the users attached to the same organization.



In the smart contract, add the following declaration of the ClientIdentity Object:

```
const { ClientIdentity } = require('fabric-shim');
```

Then in every transaction that requires it, add the filter on the user profile before running the transaction:

```
    let cid = new ClientIdentity(ctx.stub); // "stub" is the ChaincodeStub object

    //…….

    // Pay attention to put the good user profile : supplier, carrier or customer
    if (cid.assertAttributeValue('userProfile', 'supplier') && cid.getMSPID() == 'Org1MSP') {

    //…….
    }
```

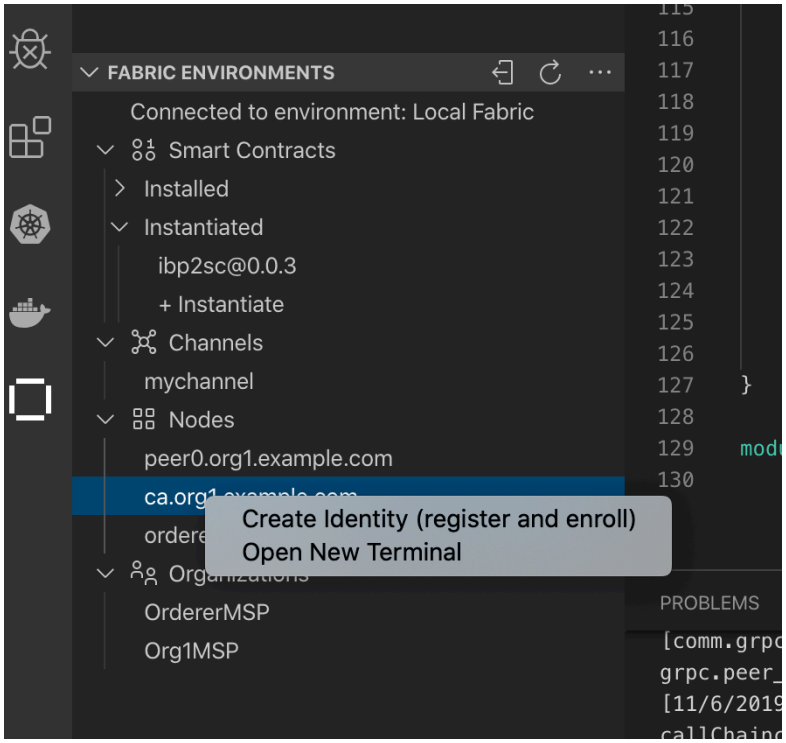
Once the SmartContract is updated with the privacy aspects, you can upgrade it in the Local fabric, creating a new version of the SmartContract.

5.3.2 Register users

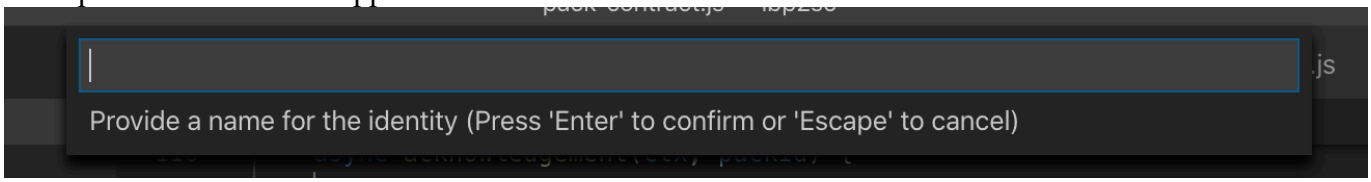
Before testing the transactions, we need to create an identity for the different user profiles to test.

So, we are using the Certificate Authority (CA) to register new users.

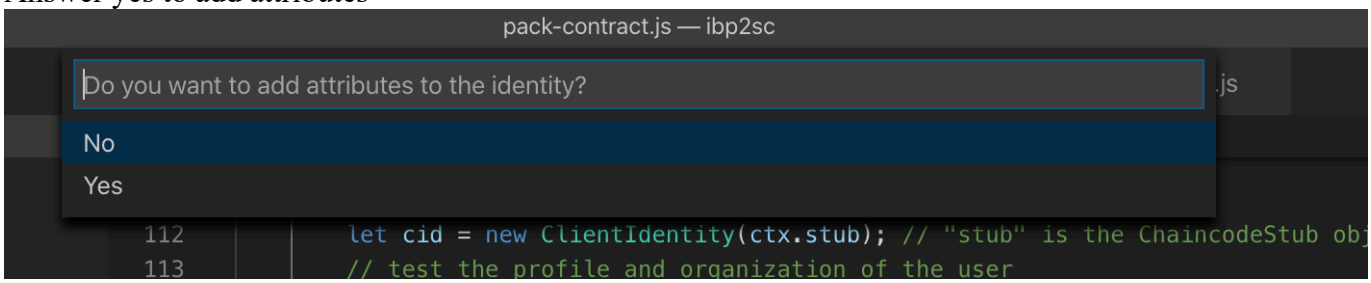
In the “fabric environments” frame, under the Nodes branch, right click on ca.org1.example.com, then click on “Create identity (register and enroll).



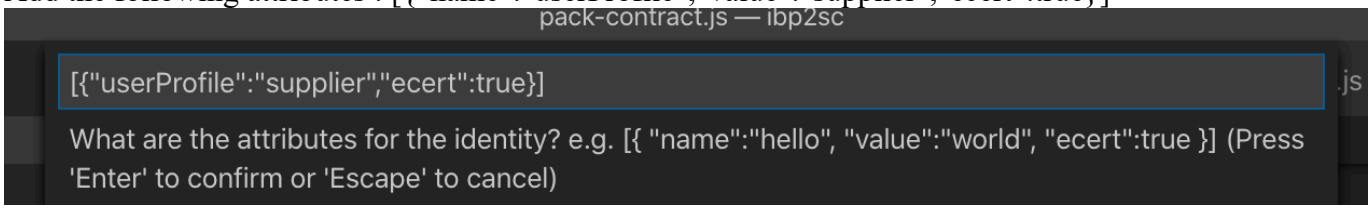
Then provide the name : supplier1



Answer yes to add attributes



Add the following attributes : [{"name":"userProfile","value":"supplier","ecert":true}]

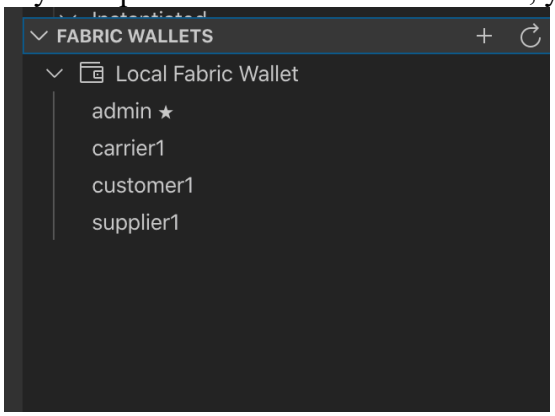


Pay attention to have the **ecert** parameter to true in order to have the attributes retrievable in the smartcontract.

You have to register 3 users:

- Name : supplier1 userProfile : supplier
- Name : carrier1 userProfile : carrier
- Name : customer1 userProfile : customer

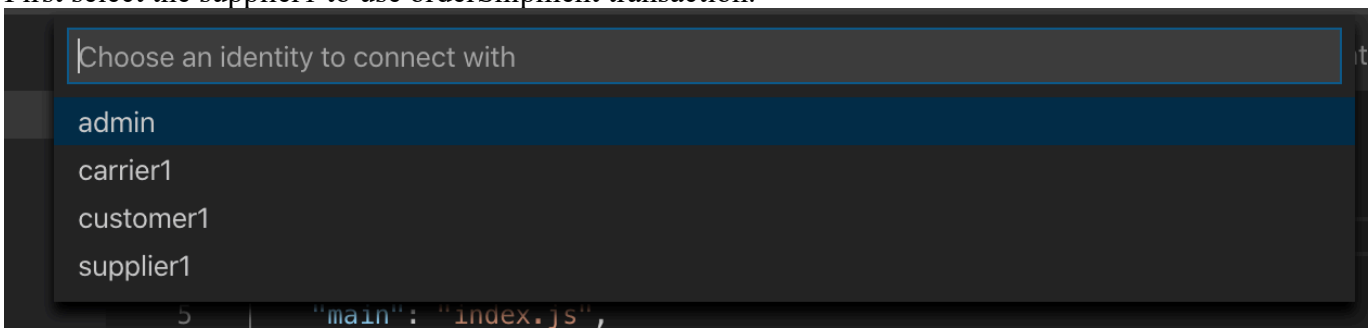
If you expand the “Fabric wallets” frame, you can see the different identities that you have created:



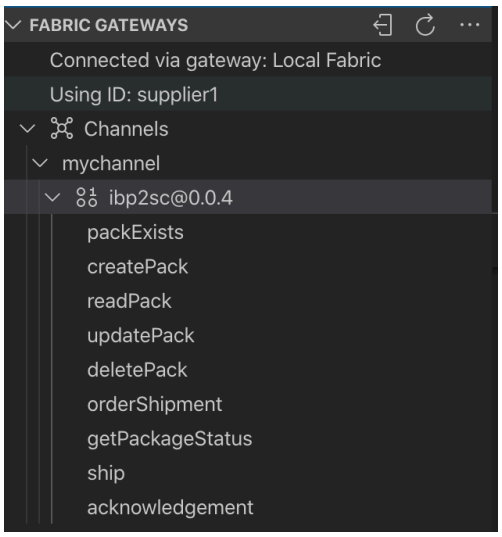
You are ready to test the transactions.

5.3.3 Test the transactions

On the “Fabric gateways” frame, click on the “Local Fabric”; you will be requested to select one identity. First select the supplier1 to use orderShipment transaction.



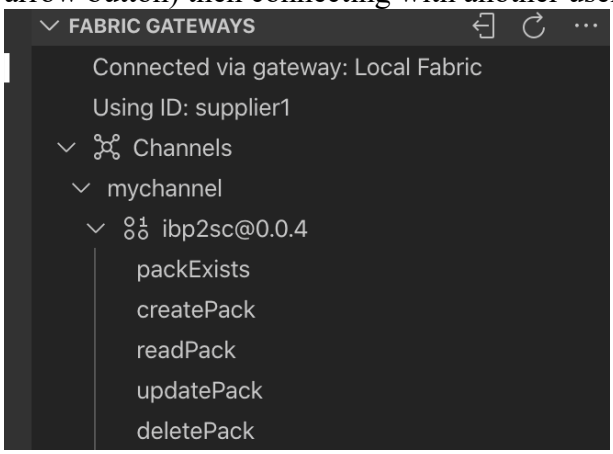
You can see that you are connected to the Local Fabric using the ID supplier1:



Now you can test the transaction **orderShipment**.

If you test the **acknowledgement** transaction, you should get an error since you are not with the right profile.

Switch between the different profile disconnecting from Local Fabric gateway (click on the door with arrow button) then connecting with another user



You can know, generate a new client application (Generate a client application). Then adapt the parameters to use the right user profile with the different transactions.

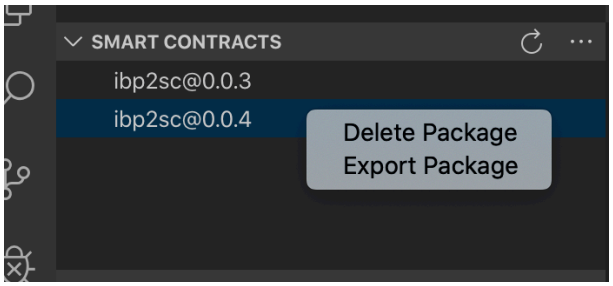
6 Deployment and test on the IBM Blockchain Platform on IBM Cloud

In this section, we will have an overview on

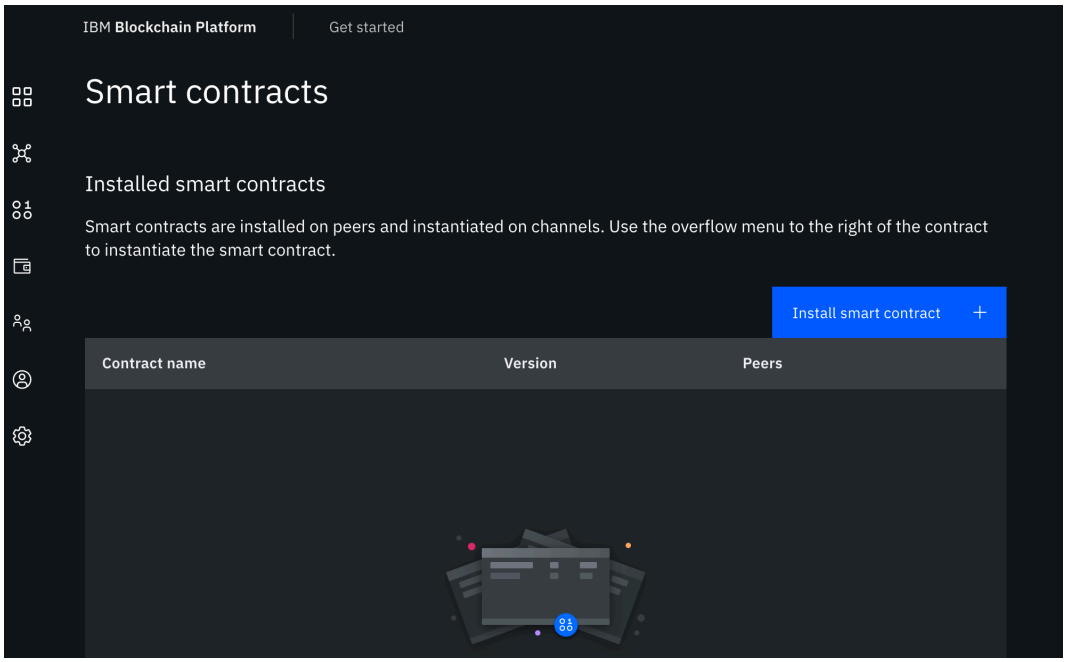
- how deploy the SmartContract on an IBM Blockchain Platform in the IBM cloud
- how to connect the VSCode environment to this IBP (using a gateway)
- how to test the smartcontract remotely

6.1 Deploy the SmartContract

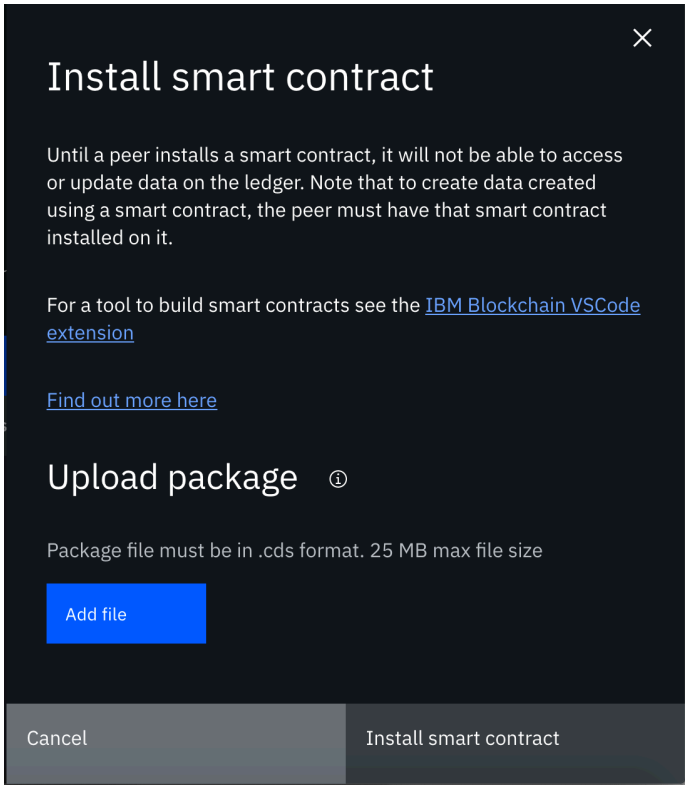
In VSCode, right click on the SmartContract , then select Export Package.



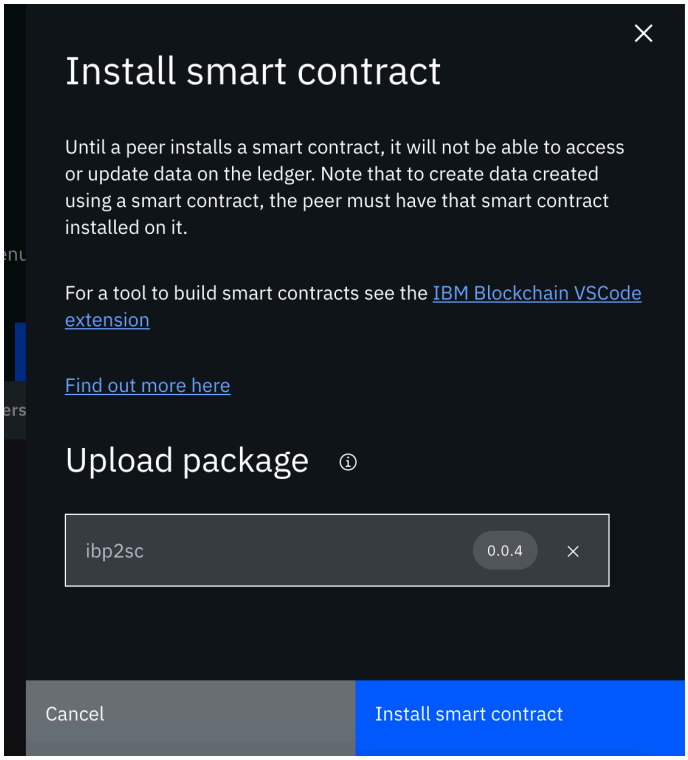
Once it's done, go to the IBM Blockchain Platform web console, in the "Smart contracts" menu, then click on the button "Install smart contract":



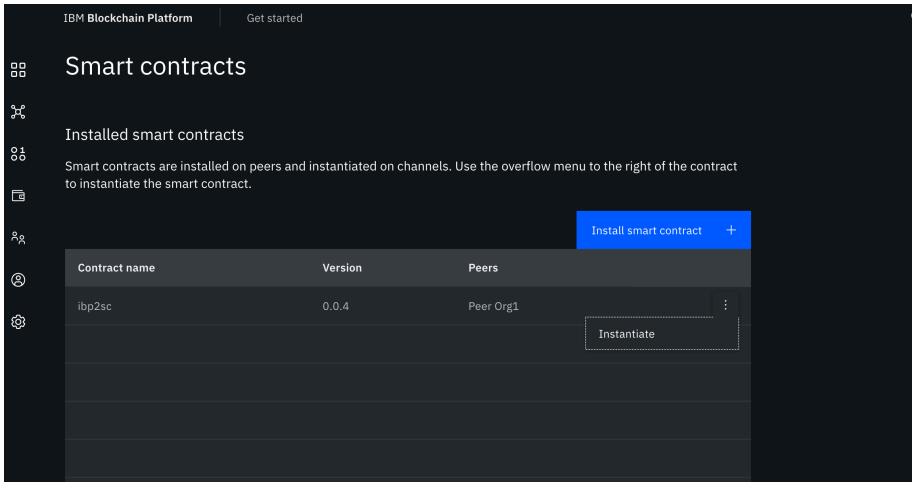
The following panel propose to add a file to upload.



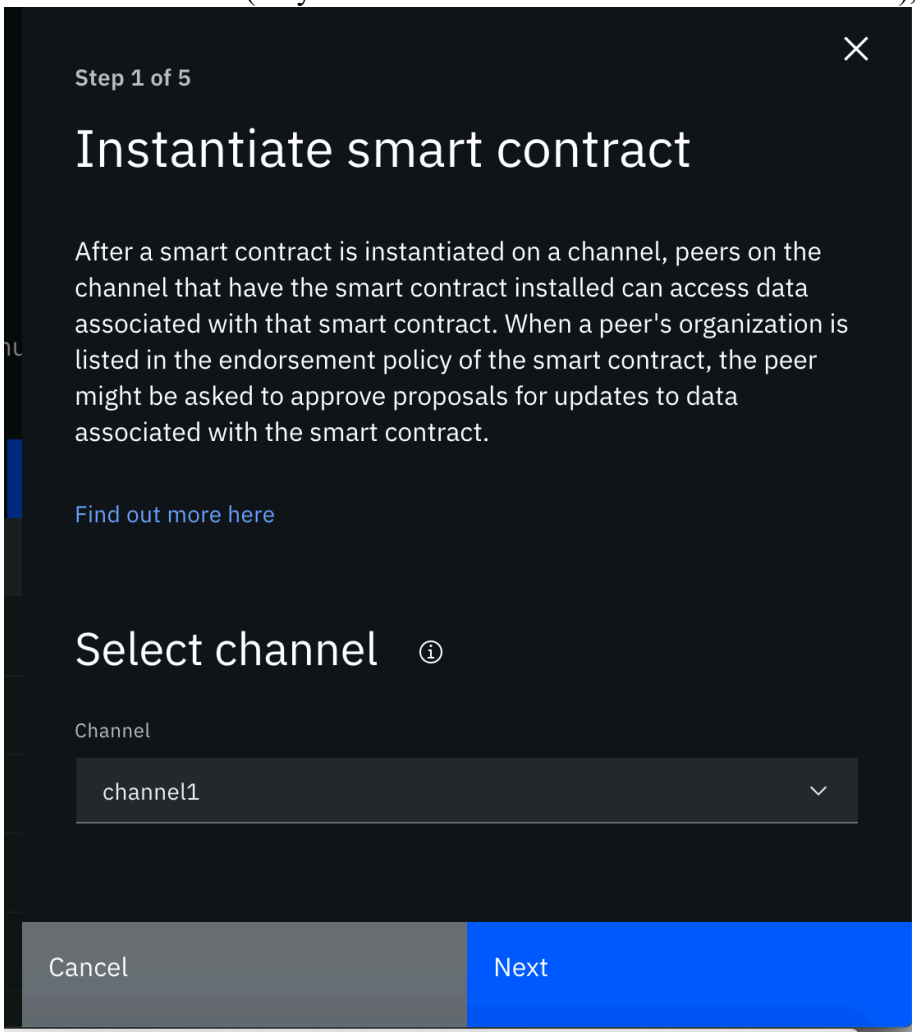
Click on “Add file” button then select the package you have previously exported, then click on “Install smart contract” button.



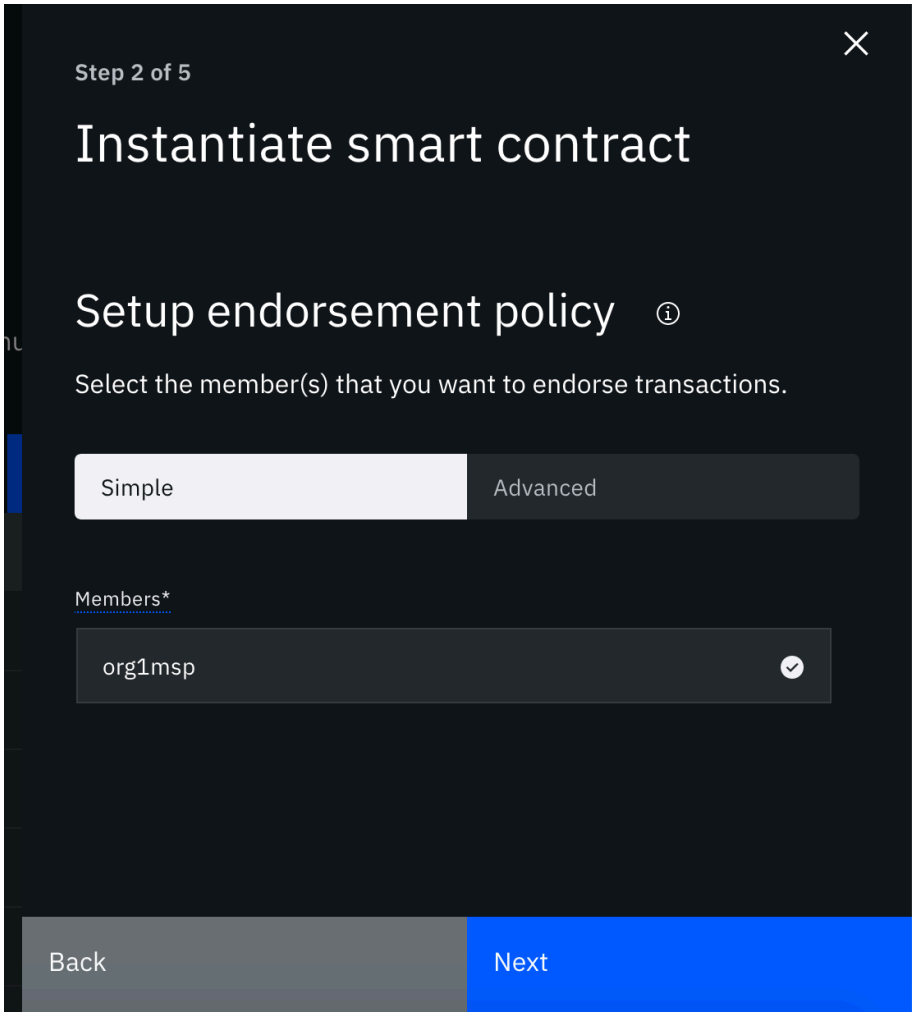
The smart contract has been installed. Now, it should be instantiated.
Click on the menu (…) then select Instantiate:



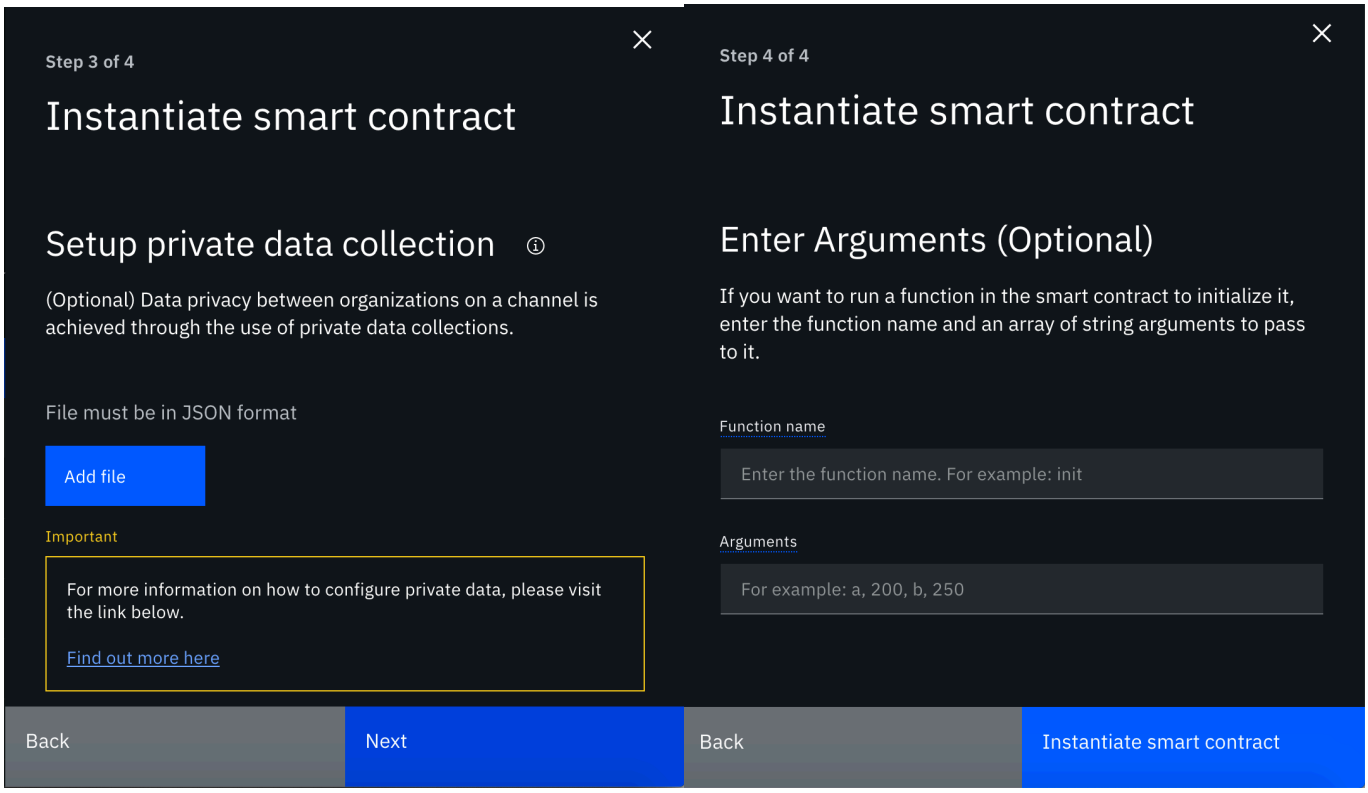
Select the channel (only channel1 is available in this test environment), then click Next.



Select the member org1msp (it corresponds to organization 1) , then click Next.



In the two next panel, let the default options and click next.
You could, when instantiating the smart contract, run a specific function with arguments in order to initialize something if needed.
In our case, no initialization steps.



The smartContract is instantiated.

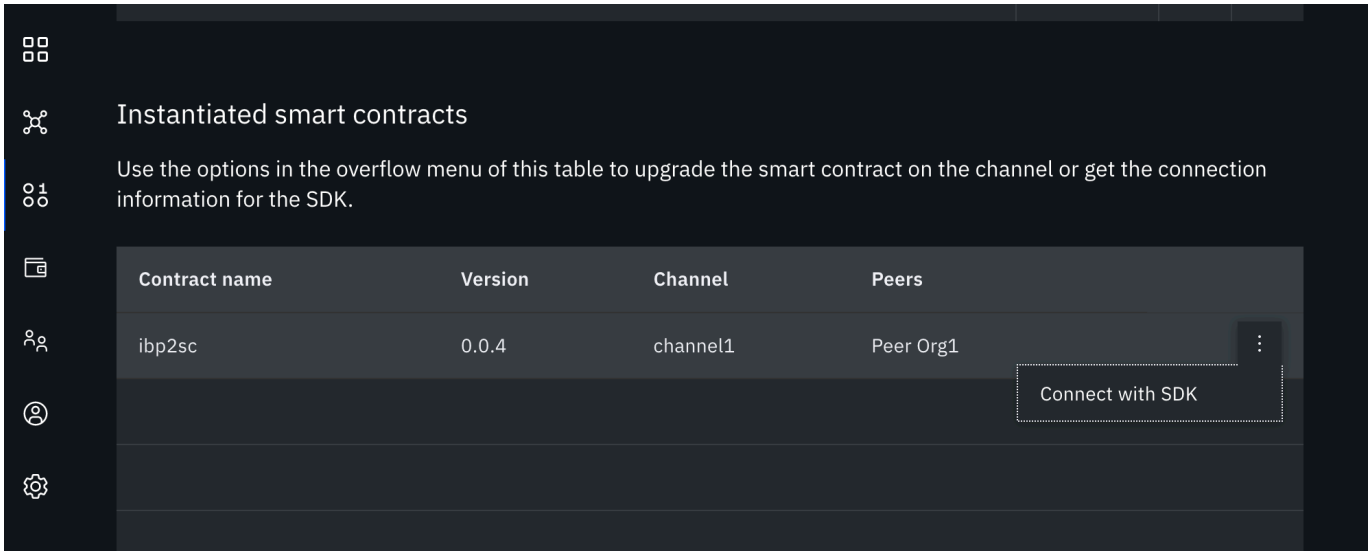
6.2 Connect the development environment (VSCode) to your IBP2.0

Now, we are going to prepare the connection from VSCode to the IBP2.0.

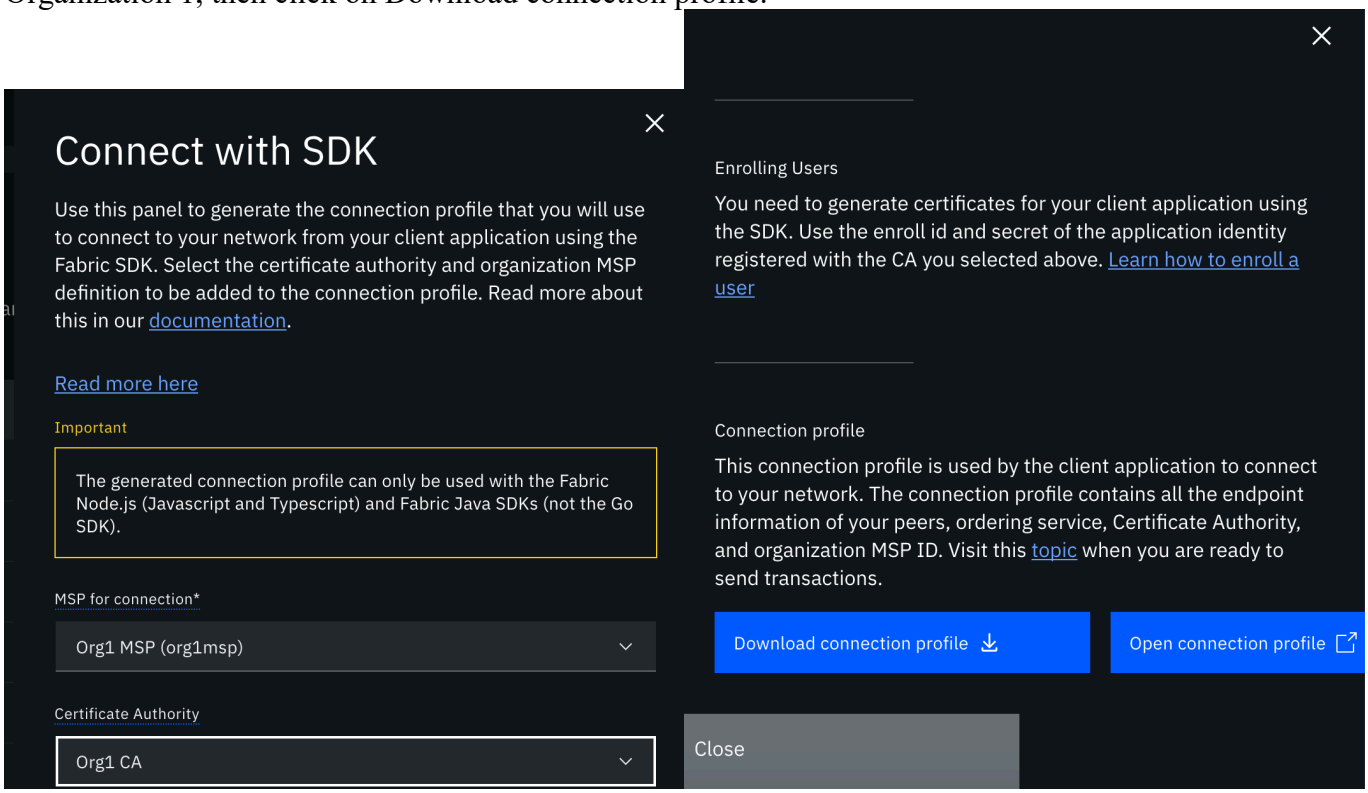
We need to retrieve a connection profile which provides the detail to connect remotely to the blockchain platform, then register users to connect remotely.

6.2.1 Retrieve Connection profile

To retrieve a connection profile, in the IBM Blockchain Platform console, click on the menu (…) on the line of the instantiated smart contract, then select the menu Connect with SDK.



Select the MSP (Membership Service Provider), Org1 MSP, and the CA (Certificate authority) of Organization 1, then click on Download connection profile.



It will download a file with the detail of the connection: the channel, the organisation, the orderer, the peers, the CA:

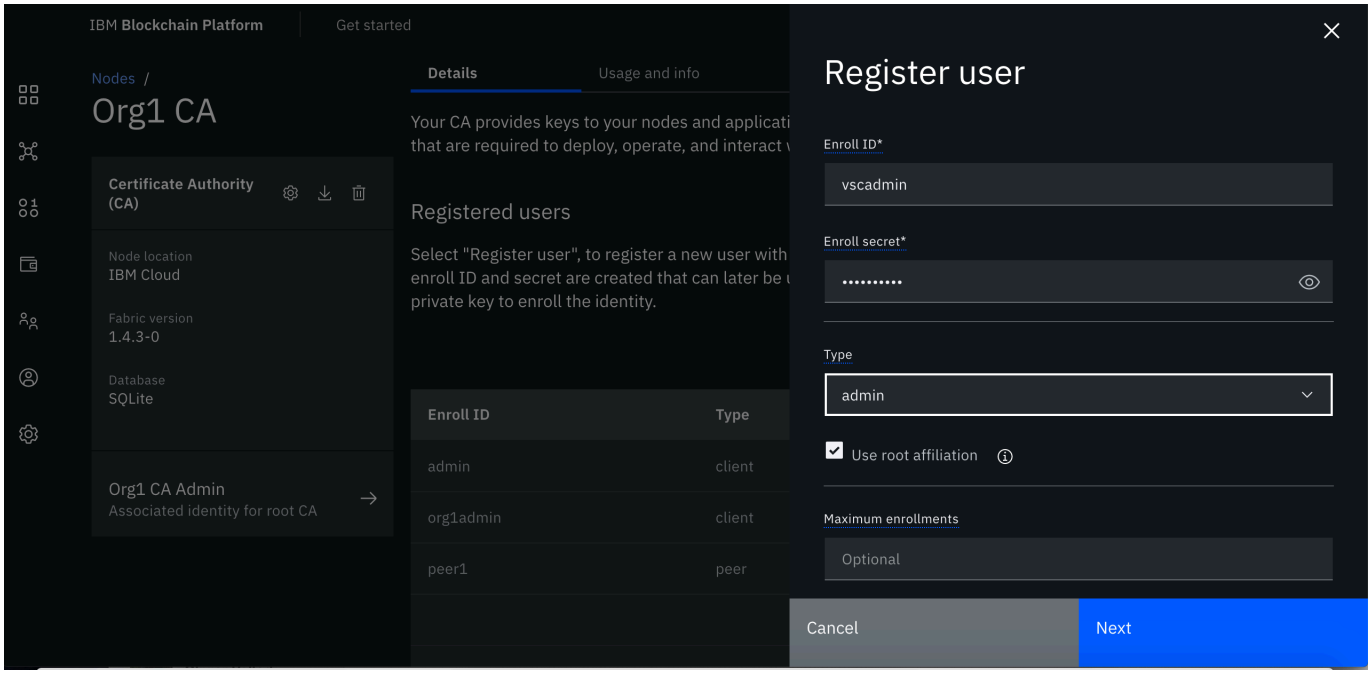
```
{
    "name": "channel1",
    "description": "Network on IBP v2",
    "version": "1.0.0",
    "client": {
        "organization": "org1msp"
    },
    "organizations": {
```

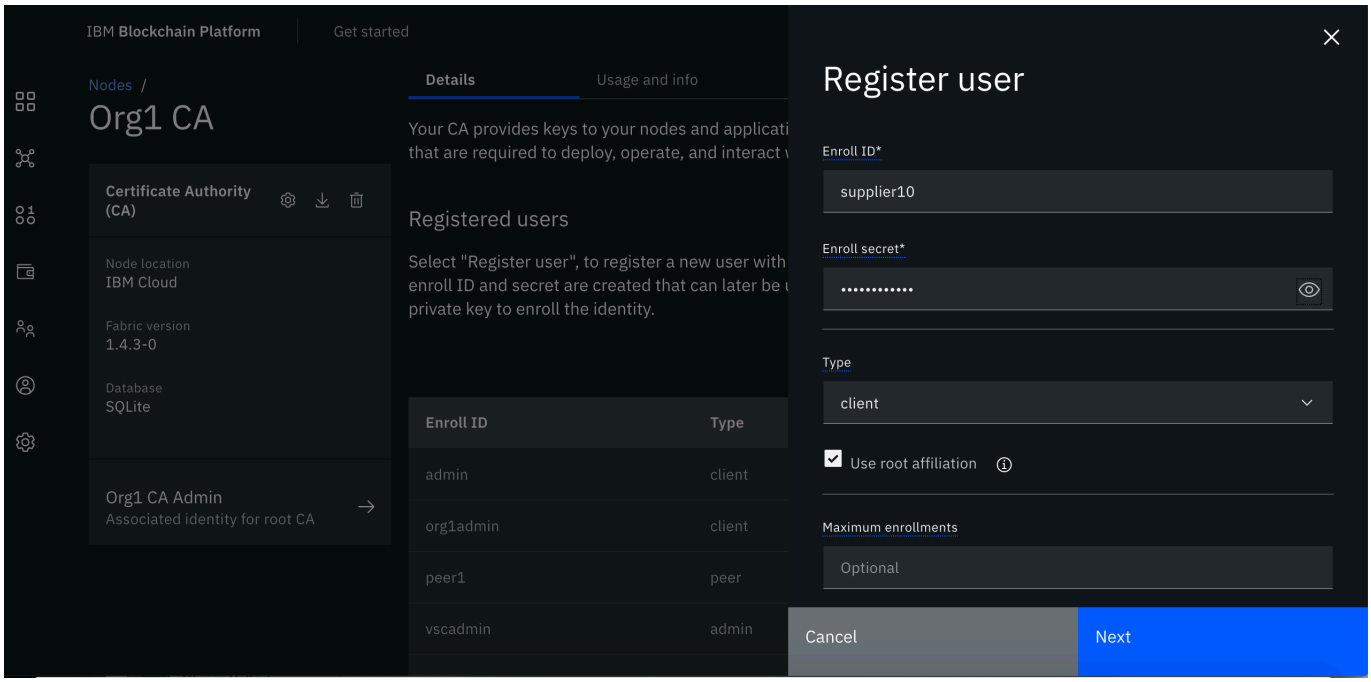
```
        "org1msp": {
            "mspid": "org1msp",
            "peers": [
                "nfcaa4f-peerorg1.ibp2.us-south.containers.appdomain.cloud:7051"
            ],
            "certificateAuthorities": [
                "nfcaa4f-org1ca.ibp2.us-south.containers.appdomain.cloud:7054"
            ]
        }
    },
    "orderers": {
        "nfcaa4f-orderingservice1.ibp2.us-south.containers.appdomain.cloud:7050": {
            "url": "grpcs://nfcaa4f-orderingservice1.ibp2.us-
south.containers.appdomain.cloud:7050",
            "tlsCACerts": {
                "pem": "-----BEGIN CERTIFICATE-----
\nMIICMTCCAdigAwIBAgIURDEh9lCsFdQXJxGk77T+F6RUrFswCgYIKoZIzj0EAwIw\nbTELMAkGA1UEBhMCVVMxFzAVBg
NVBAgTDk5vcnRoIENhcm9saW5hMRQwEgYDVQQK\nEwtIeXBlcmxlZGdlcjEPMA0GA1UECxMGRmFicmljMR4wHAYDVQQDEx
VPcmRlcmlu\nZ1NlcnZpY2VDQS10bHMwHhcNMTkxMTAzMTU0MDAwWhcNMzQxMDMwMTU0MDAwWjBt\nMQswCQYDVQQGEwJV
UzEXMBUGA1UECBMOTm9ydGggQ2Fyb2xpbmExFDASBgNVBAoT\nC0h5cGVybGVkZ2VyMQ8wDQYDVQQLEwZGYWJyaWMxHjAc
BgNVBAMTFU9yZGVyaW5n\nU2VydmljZUNBLXRsczBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABDhtH5BWxgkU\nkI2Ezs
VhutmddbR267svc9ciFyQA+MlkQesNESmmZzlXfsFkZiTi5/GQ0MYCK+p/\n9qE9kq+VKrqjVjBUMA4GA1UdDwEB/wQEAw
IBBjASBgNVHRMBAf8ECDAGAQH/AgEB\nMB0GA1UdDgQWBBSSPk2W51x1lefW4tUcr1dAmqvFJDAPBgNVHREECDAGhwQKXu
Zx\nMAoGCCqGSM49BAMCA0cAMEQCIAkgz5hY4aJmhlC4mDcy2/7kMNPoM5V1ZAbIy9fs\nhFpeAiBucvFHYU3mgVofFL13
PYSSQE6rdrad4RMIakpeuvQ9Yg==\n-----END CERTIFICATE-----\n"
            }
        }
    },
    "peers": {
        "nfcaa4f-peerorg1.ibp2.us-south.containers.appdomain.cloud:7051": {
            "url": "grpcs://nfcaa4f-peerorg1.ibp2.us-south.containers.appdomain.cloud:7051",
            "tlsCACerts": {
                "pem": "-----BEGIN CERTIFICATE-----
\nMIICHDCCAcKgAwIBAgIUU+acMnjqAGr2MUVfjpdYxu5a/FswCgYIKoZIzj0EAwIw\nYjELMAkGA1UEBhMCVVMxFzAVBg
NVBAgTDk5vcnRoIENhcm9saW5hMRQwEgYDVQQK\nEwtIeXBlcmxlZGdlcjEPMA0GA1UECxMGRmFicmljMRMwEQYDVQQDEw
pPcmcxQ0Et\ndGxzMB4XDTE5MTEwMzE1MTkwMFoXDTM0MTAzMDE1MTkwMFowYjELMAkGA1UEBhMC\nVVMxFzAVBgNVBAgT
Dk5vcnRoIENhcm9saW5hMRQwEgYDVQQKEwtIeXBlcmxlZGdl\ncjEPMA0GA1UECxMGRmFicmljMRMwEQYDVQQDEwpPcmcx
Q0EtdGxzMFkwEwYHKoZI\nzj0CAQYIKoZIzj0DAQcDQgAEDNsQBac7xMd5Yf/RNMxiMELJwcw6UgR3UHukLY6I\nFQ/XOA
L49GAFYoeUHICbvBAA0GA4/Q7SAp3unf0xQfq8+KNWMFQwDgYDVR0PAQH/\nBAQDAgEGMBIGA1UdEwEB/wQIMAYBAf8CAQ
EwHQYDVR0OBBYEFDvFxdMcTLaCfQGH\nxBofTc++BNGeMA8GA1UdEQQIMAaHBape5nswCgYIKoZIzj0EAwIDSAAwRQIhAN
PW\nA0tWtqt24fFi8LhwbvtCCuc1YtrFon3eXVHeIhU7AiAGh40WFSQqCRiDag55RgNG\nOQiLlRyj1P013ibvLrd8dg==
\n-----END CERTIFICATE-----\n"
            },
            "grpcOptions": {
                "ssl-target-name-override": "nfcaa4f-peerorg1.ibp2.us-
south.containers.appdomain.cloud"
            }
```

```
        }
    },
    "certificateAuthorities": {
        "nfcaa4f-org1ca.ibp2.us-south.containers.appdomain.cloud:7054": {
            "url": "https://nfcaa4f-org1ca.ibp2.us-south.containers.appdomain.cloud:7054",
            "caName": "ca",
            "tlsCACerts": {
                "pem": "-----BEGIN CERTIFICATE-----
\r\nMIIDADCCAmmgAwIBAgIJEpYY8sKpd1t+MA0GCSqGSIb3DQEBBQUAMIGdMUAwPgYD\r\nVQQDEzduZmNhYTRmLW9yZz
FjYS5pYnAyLnVzLXNvdXRoLmNvbnRhaW5lcnMuYXBw\r\nZG9tYWluLmNsb3VkMQswCQYDVQQGEwJVUzEXMBUGA1UECBMO
Tm9ydGggQ2Fyb2xp\r\nbmExEDAOBgNVBAcTB1JhbGVpZ2gxDDAKBgNVBAoTA0lCTTETMBEGA1UECxMKQmxv\r\nY2tjaG
FpbjAeFw0xOTExMDMxNTIwMDRaFw0yMDExMDIxNTIwMDRaMIGdMUAwPgYD\r\nVQQDEzduZmNhYTRmLW9yZzFjYS5pYnAy
LnVzLXNvdXRoLmNvbnRhaW5lcnMuYXBw\r\nZG9tYWluLmNsb3VkMQswCQYDVQQGEwJVUzEXMBUGA1UECBMOTm9ydGggQ2
Fyb2xp\r\nbmExEDAOBgNVBAcTB1JhbGVpZ2gxDDAKBgNVBAoTA0lCTTETMBEGA1UECxMKQmxv\r\nY2tjaGFpbjCBnzAN
BgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEAyyhdjlPIk/JSv6N3\r\n9DKe2JAcDCJ9gdJyR4Rn8FXuVFMvl+MbT8c/knhdg+
eLibV59Cbtn+egTBGzeqrn\r\n5kGMKWyBFec8LuxjONUd77hv35QEtTpA6J+e46jtMqiGLKHWPDZmhUV4TdvAxQ9O\r\n
lR1poF6EtZmKemRWuPugVbULi60CAwEAAaNGMEQwQgYDVR0RBDswOYI3bmZjYWE0\r\nZi1vcmcxY2EuaWJwMi51cy1zb3
V0aC5jb250YWluZXJzLmFwcGRvbWFpbi5jbG91\r\nZDANBgkqhkiG9w0BAQUFAAOBgQBPySUGDanTQfuLDl46FWjF7vhZ
CtT0O+pncRrd\r\nVpmP4aH8fTrz/PjaWmV5N3QH2CuAmUzZwPz+OVMHXTet8NprAf11N40gLs7osQJ8\r\nKdedGQKArh
IinpDV+RDnm3BPt/TwTrmY88hzpfIOrqYAUIyBldQMektODoLVzpgF\r\nEhbpQA==\r\n-----END CERTIFICATE----
-\r\n"
            }
        }
    }
}
```

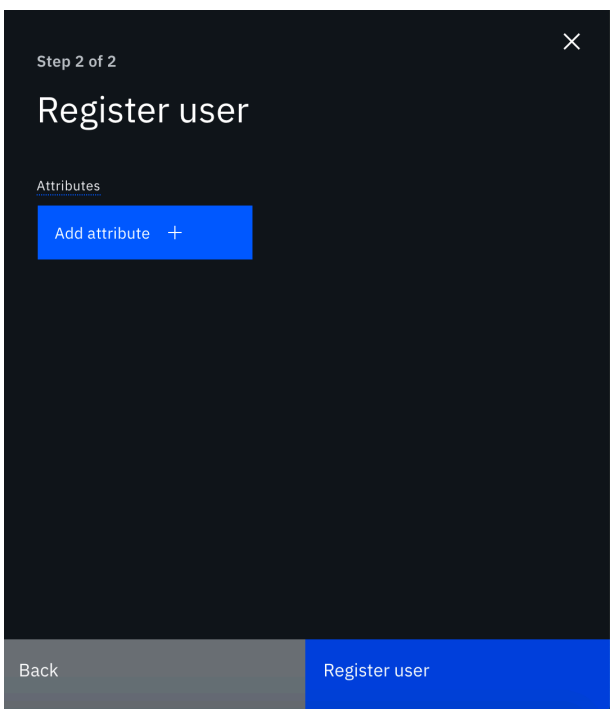

6.2.2 Register users for remote connection

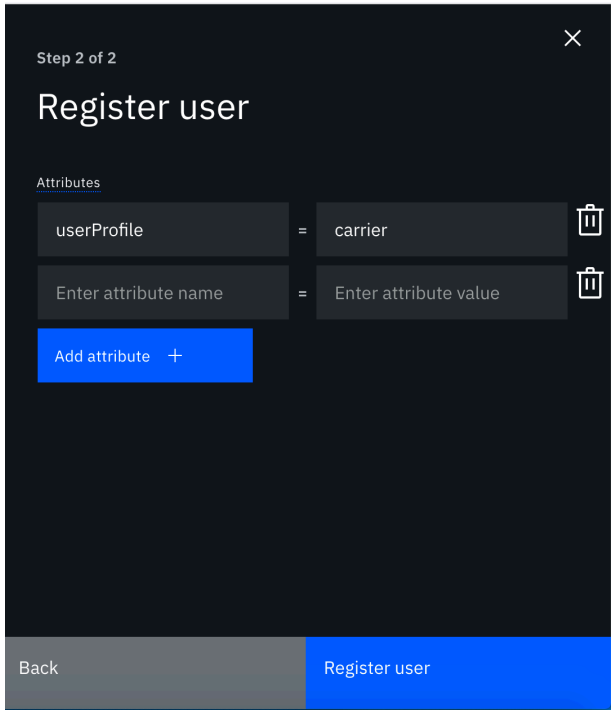
Now, we are registering users for the remote connection.
In the blockchain platform console, go to the nodes, and enter the CA of Org1.
Then click on the button “**Register user**”.



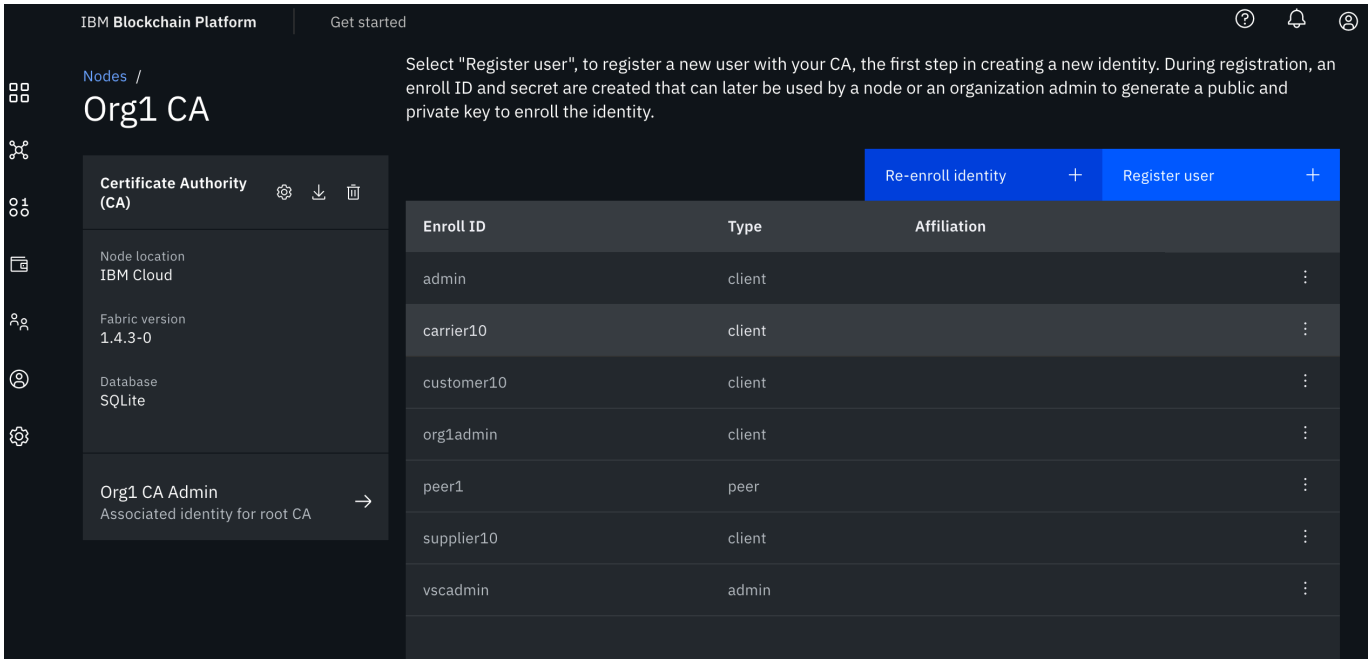


On the next panel, you have the possibility to add attributes.
So we are adding the userProfile attribute., then click on “Register user”.





The users are created

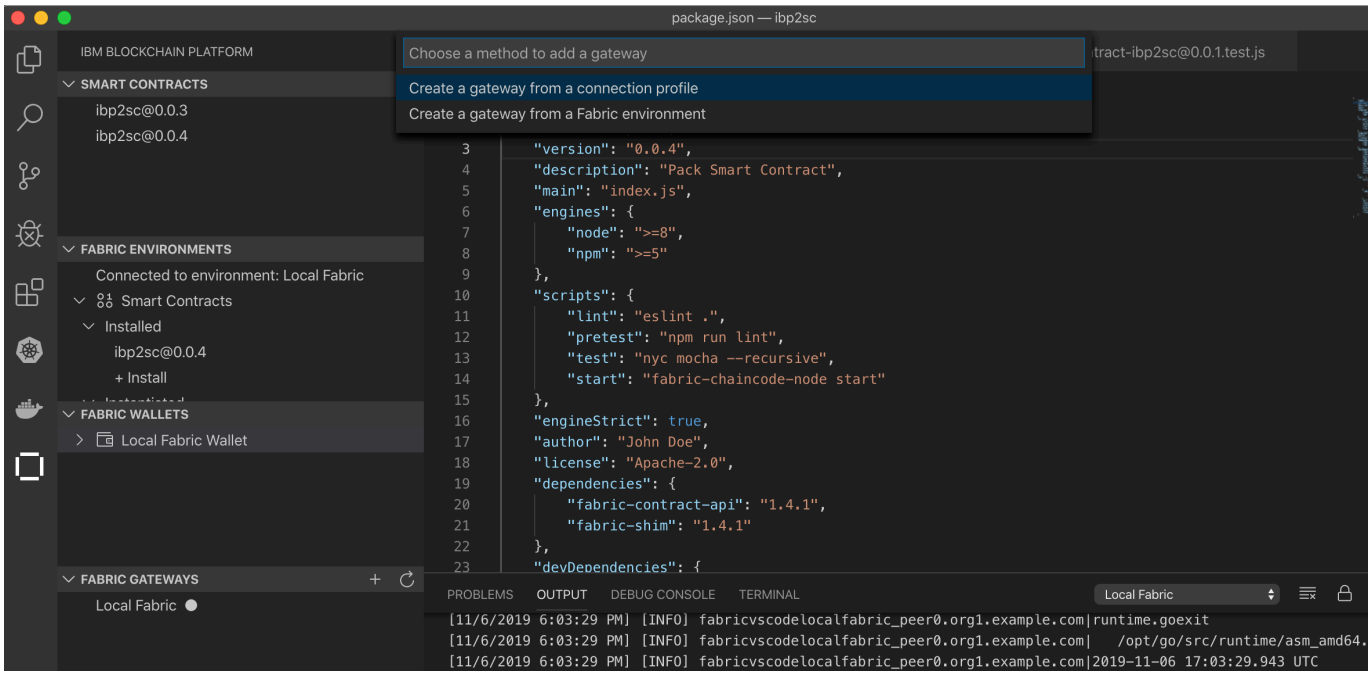


We will use the following user/pwd:

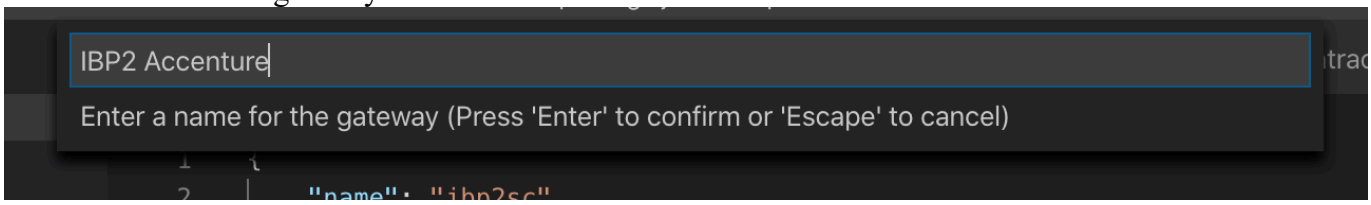
- vscadmin/vscadminpw
- supplier10/ supplier10pw
- carrier10/ carrier10pw
- customer10/customer10pw

6.2.3 Add a gateway on VSCode

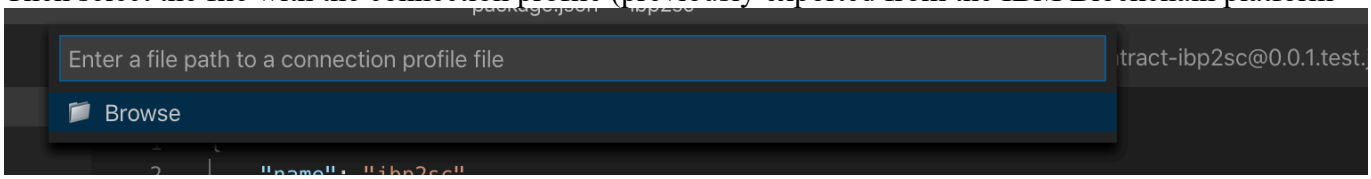
On the frame “FABRIC GATEWAYS”, click on the + button. Then select “Create a gateway from a connection profile”



Enter a name for the gateway : IBP2 Accenture



Then select the file with the connection profile (previously exported from the IBM Blockchain platform

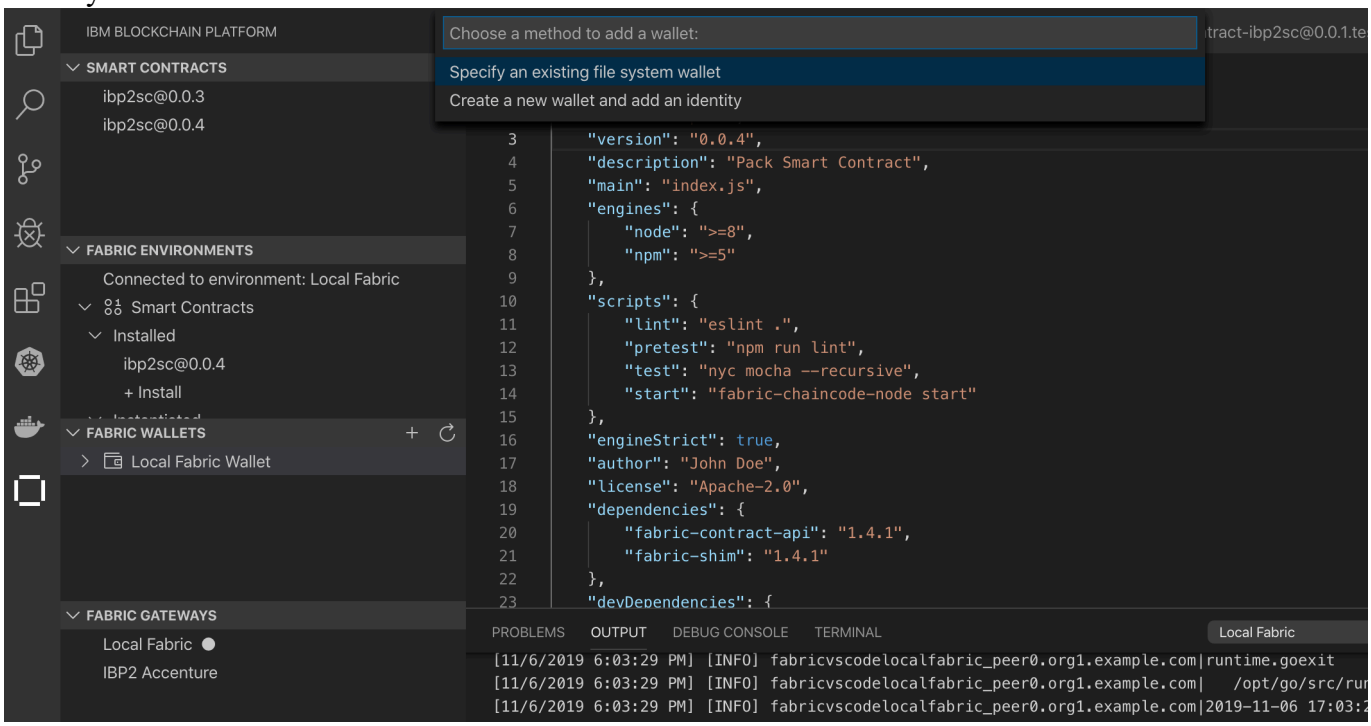


The gateway is created.

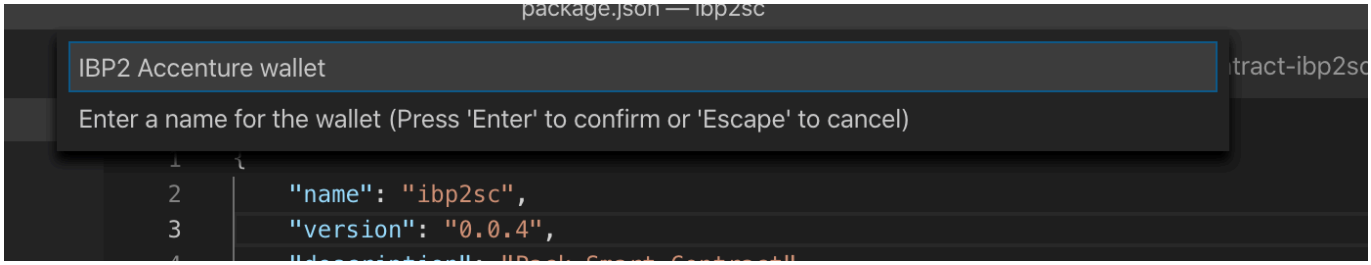
6.2.4 Add a wallet

Now, we are adding the wallet to store the identities that we will use to connect to the IBM Blockchain Platform.

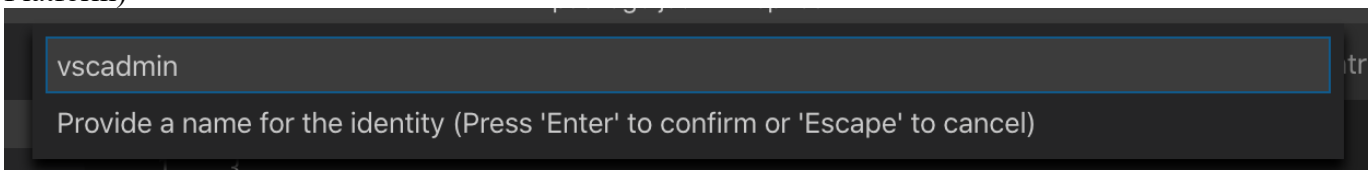
On the frame “FABRIC WALLETS”, click on the + button, then select “Create a new wallet and add an identity”



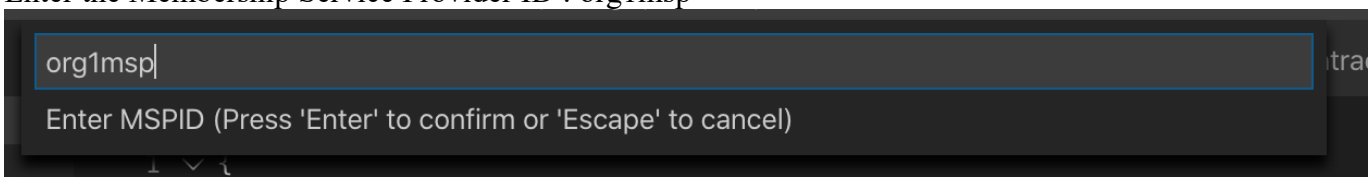
Give the name IBP2 Accenture wallet.



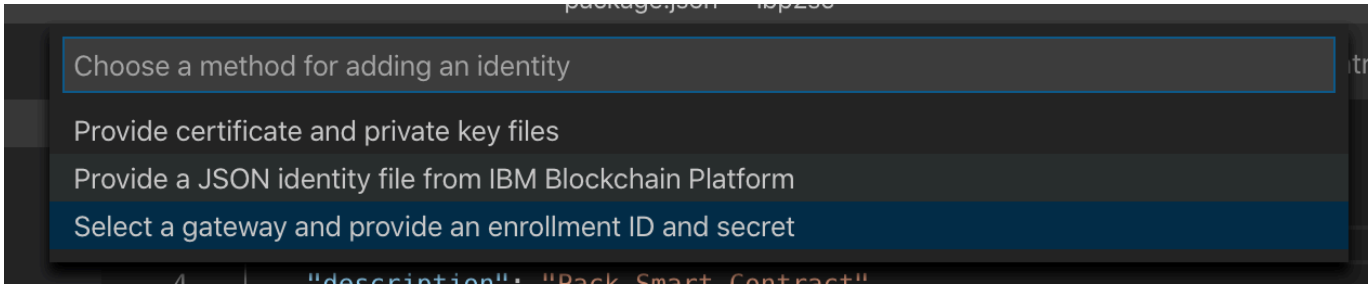
Enter the name of the identity (use one of the identities created previously in the IBM Blockchain Platform)



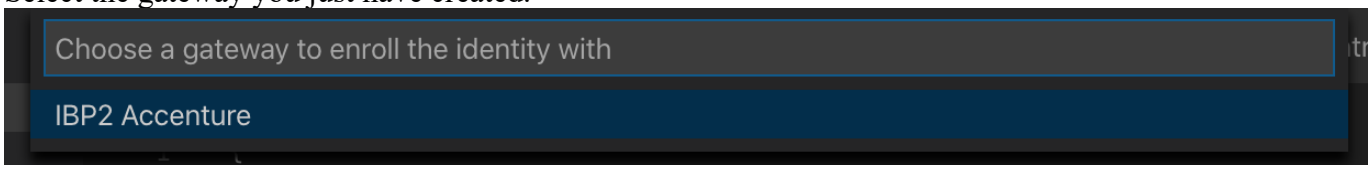
Enter the Membership Service Provider ID : org1msp



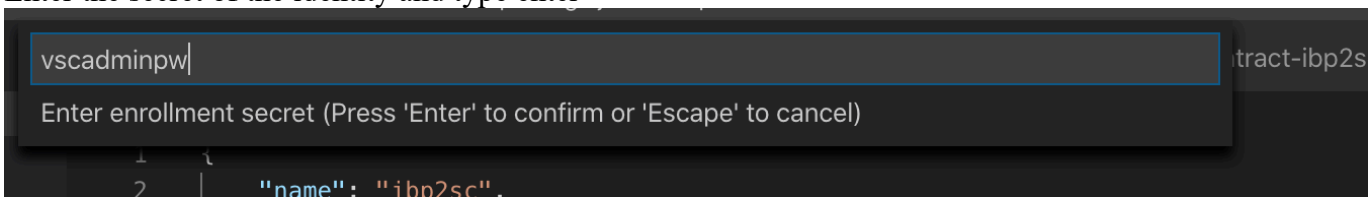
We are using the enrolment option to add the identity. Select the option : “Selet a gateway and provide an enrollment ID and secret”



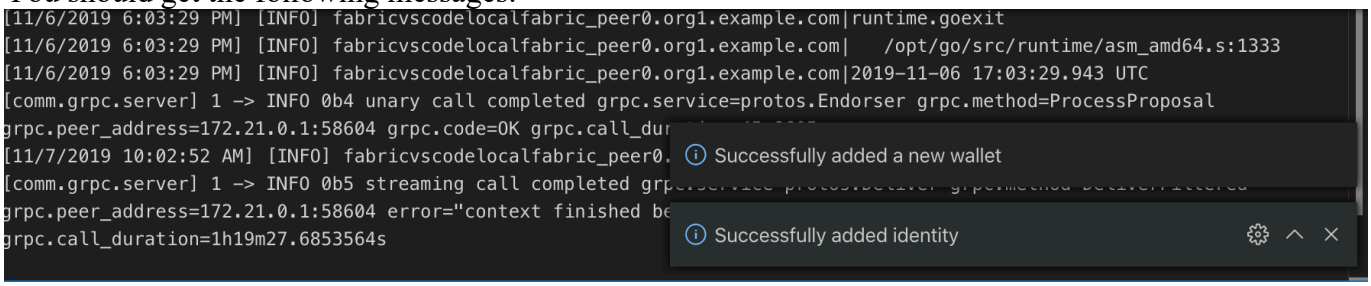
Select the gateway you just have created.



Enter the secret of the identity and type enter

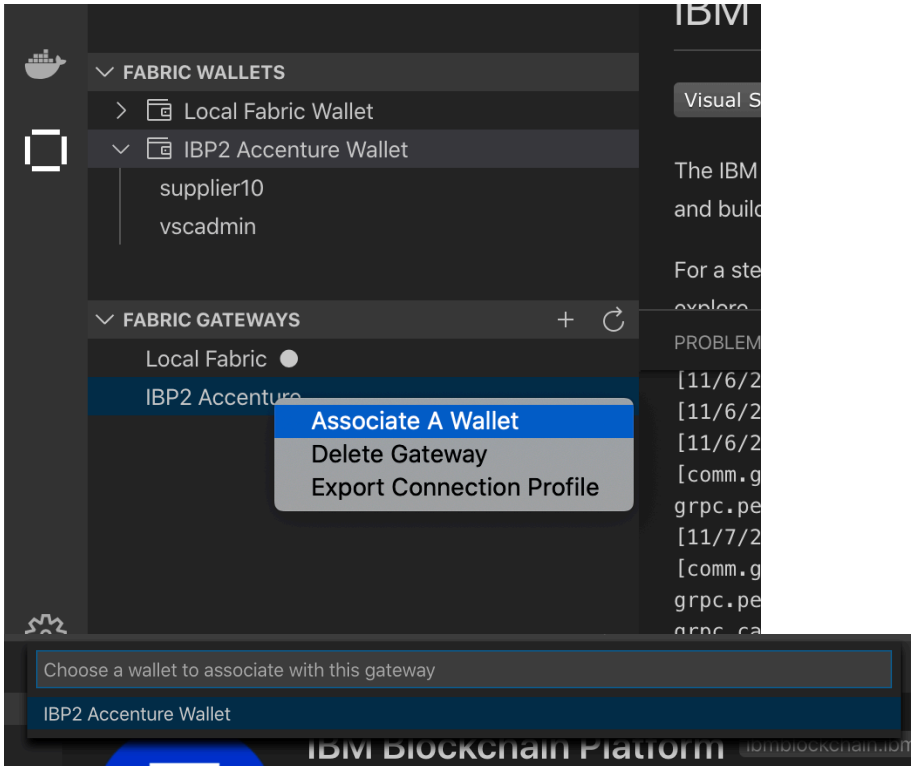


You should get the following messages:

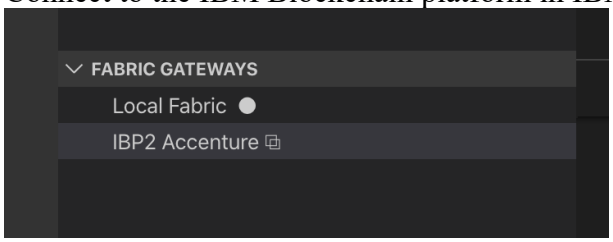


6.2.5 Associate a wallet to the gateway

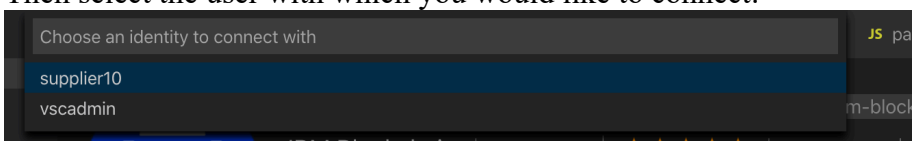
Right click on the IBP2 Accenture gateway, then select Associate a wallet.



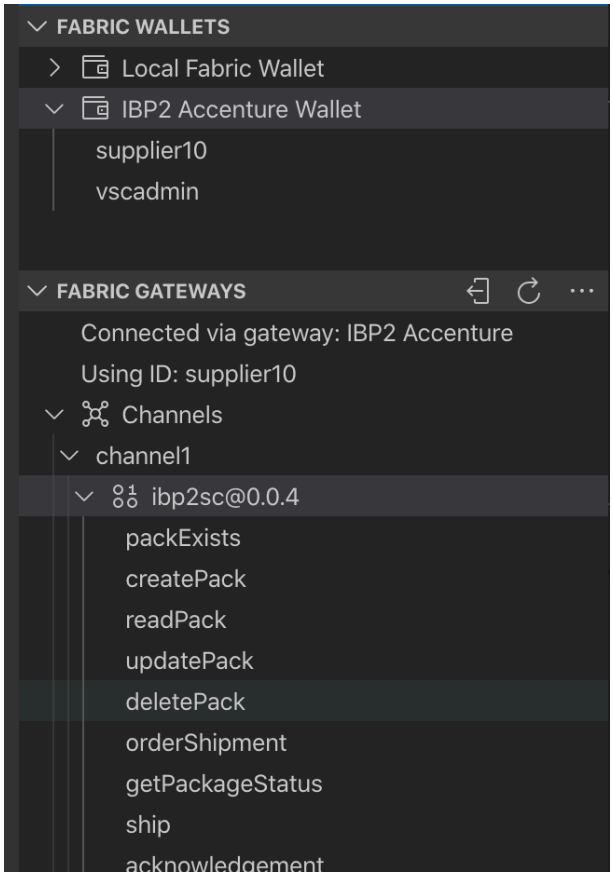
Connect to the IBM Blockchain platform in IBM Cloud: double click on the IBP2 Accenture gateway



Then select the user with which you would like to connect:



Then you can see the channel and smartContract installed:



You can test the functions/transactions as you did it with the local fabric.

Notices

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

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

products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.
All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.
This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental. All references to fictitious companies or individuals are used for illustration purposes only.
COPYRIGHT LICENSE:
This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Appendix A. Trademarks and copyrights

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

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

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

NOTES

NOTES