

Tourism Subsidy - Hyperledger Fabric Documentation

By

[Tarun Kotagiri](https://tarunkotagiri.me/)

Blockchain Developer Intern

Metalok Solutions Pvt. Ltd

Contents

1. [Introduction](#_Introduction)
2. [Overview](#_Overview)
3. [Architecture](#_Architecture)
4. [Installation and Setup](#_Installation_and_Setup)
5. [First Network](#_First_Network)
6. [APIs](#_6._APIs)
7. [Custom Network](#_Custom_Network_1)
8. [Custom Chaincode](#_Custom_Chaincode)
9. [Execution Flow](#_Execution_Flow_1)
10. [Network Configuration](#Network_config)
11. [Channel Configuration](#channel_config)
12. [Docker-Compose – Network Up](#_Docker-compose)
13. [Create channel](#_Create-channel.sh)
14. [Deploy Chaincode](#_Deploy-chaincode.sh)
15. [Chaincode Invocation and Query](#_15._Chaincode_Invoke)

# Introduction

Tourism Subsidy is an innovative project that seeks to address the issue of transparency between the state tourism department and the shooting studio for the processing of tourism subsidies. The traditional method of processing subsidies in the tourism industry has long been plagued by a lack of transparency and accountability, resulting in mistrust and inefficiencies. However, with the incorporation of private blockchain technology like Hyperledger Fabric, Tourism Subsidy has the potential to revolutionize the way subsidies are processed in the tourism industry, making the process more transparent, efficient and secure.

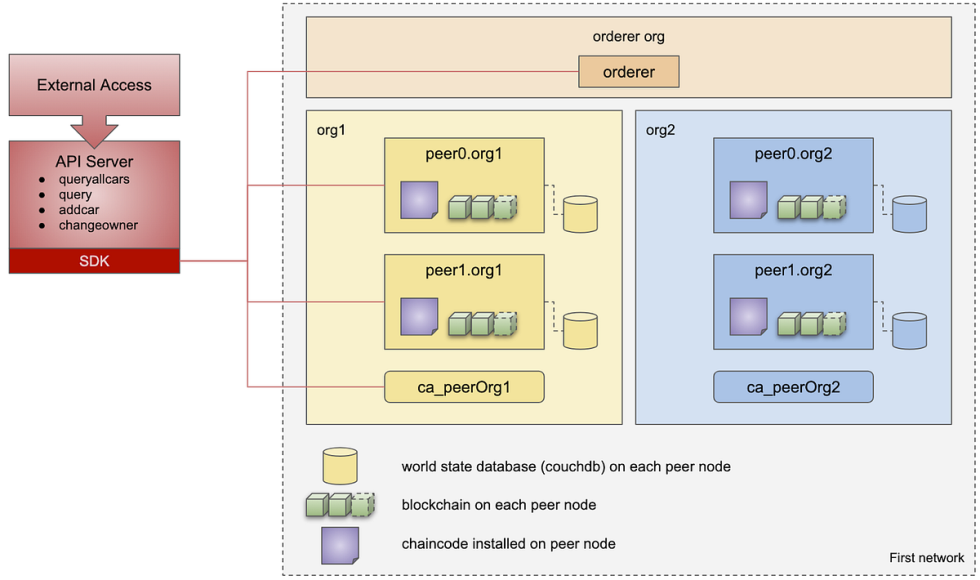
In this documentation, a detailed guide has been provided to set up and configure the Hyperledger Fabric network for Tourism Subsidy. It includes step-by-step instructions to install and configure Hyperledger Fabric, set up channels, develop chaincode, manage users, and maintain the network. By following this documentation, users can gain a thorough understanding of how Hyperledger Fabric can be used to create a secure and transparent network for processing tourism subsidies.

# Overview

The objective of this project is to provide transparency between the involved parties. That is, the information should be accessible to authorized personnel only. To achieve this, instead of using public blockchains like ethereum or polygon, we're using a private blockchain "Hyperledger fabric. This private blockchain facilitates secure, private exchange of information yet transparent among the parties involved using a concept called "channels".

The Hyperledger Fabric is based on an organizational architecture where communication between the organizations is carried out through channels. This means that an organization can only exchange information with another organization if it's part of the same channel that it wants to communicate with. For instance, if there are three organizations A, B, and C, and A and B are part of a channel called 'mychannel', then only A and B will be able to communicate on this channel. This makes the information exchange between A and B completely secure, transparent and private.

# Architecture



API SERVER

Fig. 1.0

This is the architecture of a Hyperledger fabric Network. The network consists of two organizations with two peers each. There is a global ordering service, two organizations with two peers each and each peer has it’s own couch DB instance to maintain the ledger and chaincode installed. Each organization has an instance of Certificate Authority.

**ORDERER**

In the Hyperledger Fabric network, the Ordering Service is responsible for ordering transactions into blocks and distributing them to the peers for validation and commitment to the ledger. The Ordering Service is also known as the Orderer, and it acts as the centralized point of control in the network.

The Orderer node can be implemented in different ways, including Solo, Kafka, and Raft. In the Solo implementation, there is only one Orderer node responsible for ordering transactions. In the Kafka and Raft implementations, multiple Orderer nodes are responsible for ordering transactions in a distributed manner.

In the Tourism Subsidy project, the Ordering Service is implemented using the Raft consensus algorithm. The Raft algorithm ensures that the Ordering Service is fault-tolerant and can continue to function even if some of the nodes fail. The Raft algorithm works by electing a leader node among the Orderer nodes, which is responsible for ordering transactions and ensuring consistency among the nodes.

**ORGANISATION**

In Hyperledger Fabric, an organization is a group of network participants that share a common identity, purpose, and administrative authority. An organization is responsible for managing the identities of its members, controlling access to resources, and participating in transactions.

Each organization in a Hyperledger Fabric network has its own Certificate Authority (CA) server, which is responsible for managing the identities of its members and issuing digital certificates. The CA server ensures that only authorized participants can access the network and perform transactions.

Organizations in Hyperledger Fabric communicate with each other through channels, which are private subnetworks that allow organizations to exchange information securely and transparently. Each channel has its own ledger, and only the organizations that are authorized to access the channel can read or write to its ledger.

The state tourism department and the shooting studio are the two primary organizations involved in the project, and each organization has two peers. The peers are responsible for participating in transactions and maintaining a copy of the ledger for their organization.

The organization's administrative authority is managed by a set of policies, which define the rules for access control and governance. These policies are defined by the organization's administrators and are enforced by the CA server.

**PEER**

In Hyperledger Fabric, a peer is a network node that participates in the transaction processing, validation, and consensus. Peers in Hyperledger Fabric are responsible for endorsing and validating transactions, maintaining the state database, and sharing ledger updates with other peers in the network. Peers also interact with the chaincode, which contains the business logic for the transactions.

There are two types of peers in Hyperledger Fabric: endorsing peers and committing peers. Endorsing peers are responsible for executing the chaincode and endorsing the transaction by adding their signature to the transaction proposal. The endorsement indicates that the transaction has been executed correctly and is valid.

Committing peers are responsible for validating the transaction and updating the ledger with the transaction data. The committing peers ensure that the transaction has been endorsed by the required number of endorsing peers and that it meets the network's consensus rules. Once the transaction is validated, the committing peers update the ledger, making the transaction permanent and immutable.

In the Tourism Subsidy project, each organization has two peers. The peers maintain a copy of the ledger for their organization and participate in the transaction processing and validation. The peers execute the chaincode, endorse the transaction, and validate it before updating the ledger.

**COUCH DB**

In Hyperledger Fabric, the world state database is used to store the current state of the ledger. This database contains the latest values of all the variables stored in the ledger, which are updated with each transaction. Hyperledger Fabric allows for different implementations of the world state database, including CouchDB.

CouchDB is a NoSQL document-oriented database that is used as the default world state database in Hyperledger Fabric. It is a distributed database that is designed to handle large amounts of unstructured data. CouchDB is known for its scalability, flexibility, and fault-tolerance, making it an ideal choice for use in blockchain networks.

In Hyperledger Fabric, the world state database stores the latest state of the ledger, which is updated with each transaction. CouchDB stores this data in the form of documents, which can be queried using a RESTful API. This makes it easy to search and retrieve data from the database.

CouchDB also provides built-in security features, such as user authentication and access control, which can be used to control access to the database. This ensures that only authorized users have access to the data stored in the world state database

**FABRIC CA**

In a Hyperledger Fabric network, the Certificate Authority (CA) is responsible for issuing digital certificates and managing identities for the network participants. The CA ensures that only authorized participants can access the network and perform transactions.

The CA issues certificates to network participants, which contain a public key and other relevant information. These certificates are used to authenticate participants in the network and ensure secure communication. The CA is responsible for revoking certificates in case of a security breach or when a participant leaves the network.

The Hyperledger Fabric network supports different types of CAs, including the Fabric-CA server and the External-CA server. The Fabric-CA server is built-in to the Hyperledger Fabric network and is recommended for most use cases. The External-CA server can be used when an organization already has an existing CA infrastructure that they want to integrate with the Hyperledger Fabric network.

In the Tourism Subsidy project, the Fabric-CA server is used to manage identities and issue certificates for the network participants. Each organization has its own Fabric-CA server that is responsible for managing the identities of its members. The Fabric-CA server is configured to issue certificates with specific roles and permissions, ensuring that only authorized participants can access the network.

# Installation and Setup

**Development Ecosystem:**

The Hyperledger fabric strictly runs only in linux Environment. So preferrable install it in any linux distribution or set up Windows subsystem for linux (WSL) environment in Windows.

1. WSL for windows

<https://learn.microsoft.com/en-us/windows/wsl/install>

1. Ubuntu

<https://www.microsoft.com/store/productId/9PDXGNCFSCZV>

1. Docker Desktop

<https://www.docker.com/products/docker-desktop/>

1. Docker Integration with WSL

<https://www.youtube.com/watch?v=izpYsEn2V4I>

**INSTALLATION AND SETUP**

1. **Prerequisites:**
   1. Docker
   2. Docker-compose
   3. Curl
   4. Go lang
   5. Node js
   6. Npm
   7. Python
2. **Hyperledger fabric, fabric samples and couchdb**

This command install with proper versions installs the Hyperledger fabric, fabric samples and couchdb

`curl sSL https://bit.ly/2ysbOFE | bash -s -- fabric\_version fabric-ca\_version thirdparty\_version`

`curl -sSL https://bit.ly/2ysbOFE | bash -s -- 2.0.1 1.4.6 0.4.18`

No need to explicitly run the above command, there is a shell script which installs the prerequisite and the fabric samples taking care of everything including adding the fabric binaries to the ./bashrc (path).

You can find the script to Set up Hyperledger fabric here:

<https://github.com/Metalok/Tourism-Subsidy-/blob/master/hyperledger%20fabric/fabric_2.0_setup.sh>

After successfully running the above script, you can see fabric images for orgs, peers, couch db and other images required to bring the fabric network up. Also you’ll find a folder named **fabric-samples** in the current working directory, which contains the binaries and required files to work with Hyperledger fabric. The above bash script adds the fabric binaries to system path, so they’re ready to be used by containers in the later stage.

Run **“ peer version ”** in the terminal to verify the binary.

And change the directory to /fabric-samples/test-network using the command **“ cd /fabric-samples/test-network ”**

Now bring the test network up using the command “**./network.sh up”** to verify the installation of Hyperledger fabric.

NOTE:

On Windows: Make sure Docker Desktop is up and running

On linux : Make sure Docker is running. Verify it by using command **docker -v**

# First Network

Now that the fabric binaries are set up, change directory to **fabric-samples/first-network** . To bring the First network up, here’s the command.

*./byfn.sh up -a -l javascript -s couchdb*

Breakdown of the command

**./byfn.sh** : script that stands for build your first network

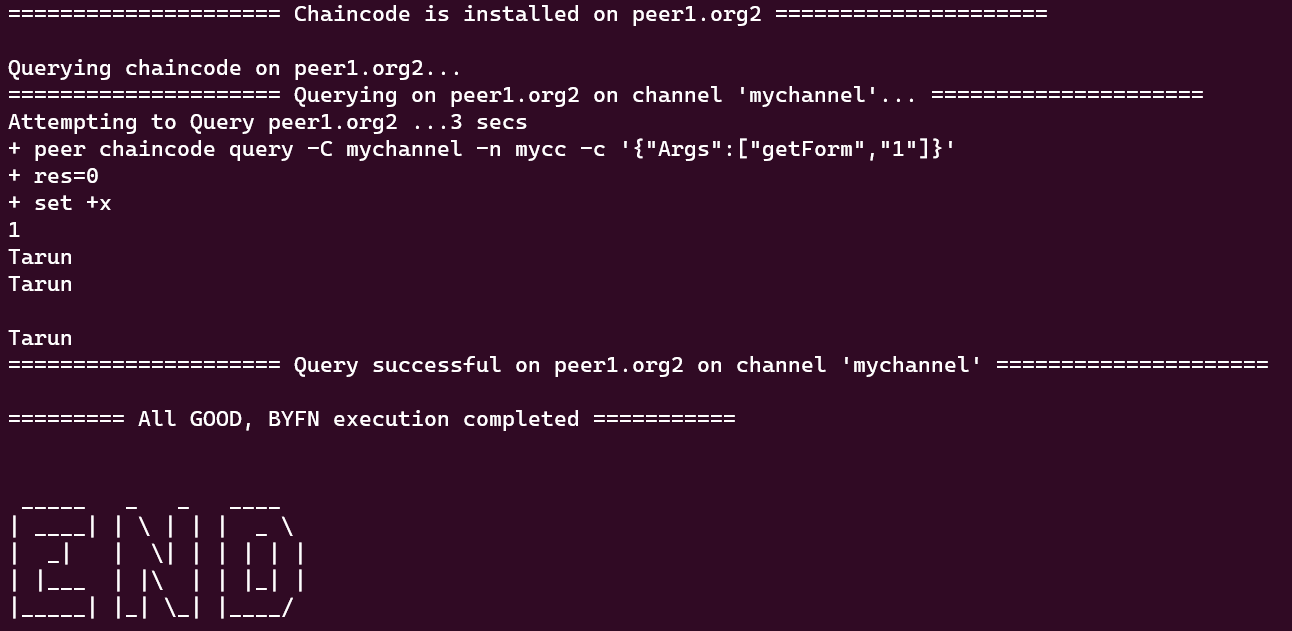
-**a** : switch suggesting ‘all’ - brings up one CA (certificate authority) for each organization

**-l javascript** : specifies source language as javascript

**-s couchdb** : Hyperledger by default uses levelDB, this switch suggests using couchdb. Brings up an instance for each peer of an organization.

Note : Before running the above command, make sure docker desktop is up and running.

./byfn.sh scripts runs an end to end first network, which also tests for invoking and quering the chaincode. A successful execution looks something like this.

****

# 6. APIs

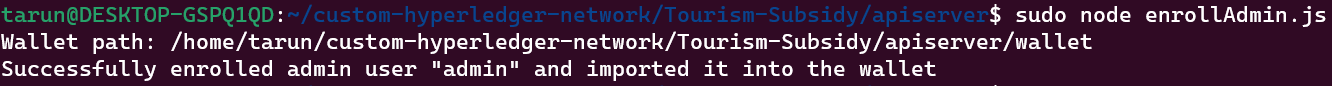
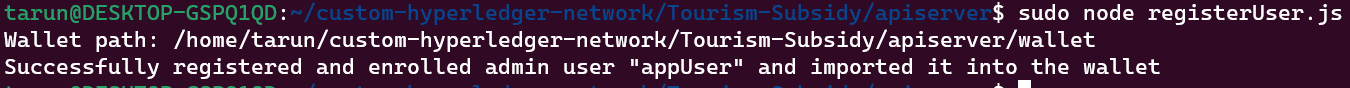
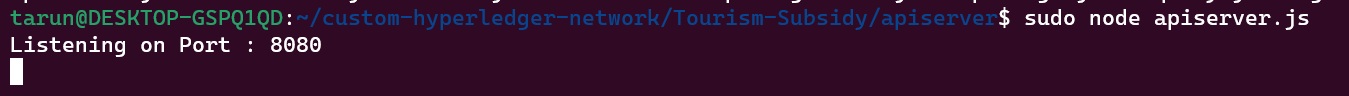
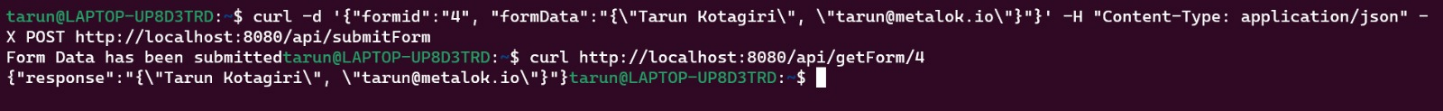
To access the API end points, from the **fabric-samples** directory, cd into **Tourism-Subsidy** directory.

These are the contents of this directory.



To get started, you can run **npm install** which installs the dependencies mentioned in the package.json file.

Once, that’s done, follow these steps.

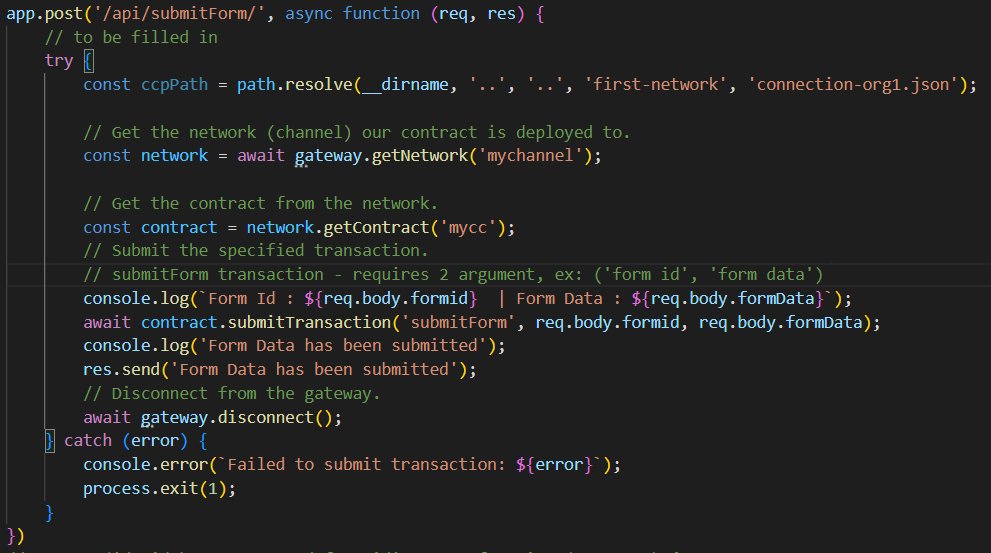
1. sudo node enrollAdmin.js : creates an admin user to manage other users for this network.
2. sudo node registerUser.js : Registers a user to interact with the network
3. sudo node apiserver.js : runs the api server, which listens on port 8080.
4. From another terminal, make api calls to invoke and query the data on Hyperledger Fabric Network
5. Details of the API endpoints:

* **GET /api/getForm/:form\_index**



This endpoint handles a GET request for retrieving a form with a given index (**form\_index**) from the blockchain network. The endpoint first loads the network configuration from a connection file located in the first-network directory. Then, it creates a new file system-based wallet and checks if the user appUser has been enrolled in the wallet. If the user is enrolled, the endpoint creates a new gateway and connects to the peer node using the loaded network configuration and the user's identity. It then retrieves the network/channel (**mychannel**) and the smart contract (**mycc**) deployed on the network, and evaluates the **getForm** transaction on the contract, passing the form\_index as an argument. Finally, the endpoint returns the result of the transaction in JSON format.

* **POST /api/submitForm/**



This endpoint handles a POST request for submitting a form to the blockchain network. The endpoint first loads the network configuration from a connection file located in the first-network directory. Then, it creates a new file system-based wallet and checks if the user appUser has been enrolled in the wallet. If the user is enrolled, the endpoint creates a new gateway and connects to the peer node using the loaded network configuration and the user's identity. It then retrieves the network/channel (**mychannel**) and the smart contract (**mycc**) deployed on the network, and submits the **submitForm** transaction on the contract, passing the **formid** and **formData** as arguments. Finally, the endpoint returns a success message in the response.

# Custom Network

You can find the github reposity here:

<https://github.com/Metalok/Tourism-Subsidy-/tree/master/hyperledger%20fabric/HyperLedger-End2End>

Folder structure:

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No.** | **Name** | **Dir/file** | **Description** |
| 1 | Artifacts | Dir | Contains channel info, chaincode, crypto materials such as ca certs, msp, users for both peer orgs as well as ordering service. |
| 2 | Channel  Artifacts | Dir | Contains channel .block files for all the created channels |
| 3 | Readable-Blocks | Dir | Contains json files of channel and genesis block |
| 4 | Id-Files | Dir | Contains public keys of users |
| 5 | Explorer | Dir | Contains connection profile of Hyperledger Explorer |
| 6 | API-1.4 | Dir | Api implementation using fabric client |
| 7 | API-2.0 | Dir | Api implementation using fabric network |
| 8 | createChannel.sh | file | Used for Creating a channel |
| 9 | Deploychaincode.sh | file | Used for Deploying the chaincode |
| 10 | Network\_up.sh | file | Custom shell script to bring the network up, create channels and join peers of orgs to respective channels. |
| 11 | Network\_down.sh | file | A shell script to bring the network down |
| 12 | Tourism\_forms.tar.gz | file | Packaged chaincode |

# Custom Chaincode

# 

**Chaincode : Tourism forms**

This is a chaincode written in JavaScript using the fabric-contract-api and shim-api libraries for Hyperledger Fabric version 2.0. The purpose of this chaincode is to store and retrieve form data using a composite key.

The fabric-contract-api library is a Node.js package that provides a high-level API for writing chaincode in JavaScript. It abstracts away some of the low-level details of the Fabric API, such as handling transactions and versioning, and provides a simpler interface for defining chaincode methods.

The shim-api library is another Node.js package that provides a low-level API for interacting with the Fabric network. It allows chaincode developers to access the state database, read and write to the ledger, and handle transactions directly. The fabric-contract-api library actually builds on top of the shim-api library to provide a more user-friendly interface.

In this chaincode, the createCompositeKey method of the stub object is used to create a composite key that consists of a prefix ('form') and a unique identifier (formId) for each form. This allows multiple forms to be stored in the state database without conflicting with each other.

The putState method is used to store the form data in the state database.

The chaincode defines a FormDataContract class that extends the Contract class from the fabric-contract-api library.

The class contains two methods:

**submitForm**: This method takes in two parameters - a context object (ctx) and two strings (formId and formData). It creates a composite key using the formId and stores the formData as a JSON object in the state database using the putState method of the stub object. The formData is converted to a Buffer object before storing it in the state database. The method also logs a message indicating the successful storage of the form data.

# **getForm**: This method takes in two parameters - a context object (ctx) and a string (formId). It creates a composite key using the formId and retrieves the form data from the state database using the getState method of the stub object. If the form data is not found or is empty, the method throws an error indicating that the form does not exist. Otherwise, the method returns the form data as a JSON object.

# Execution Flow

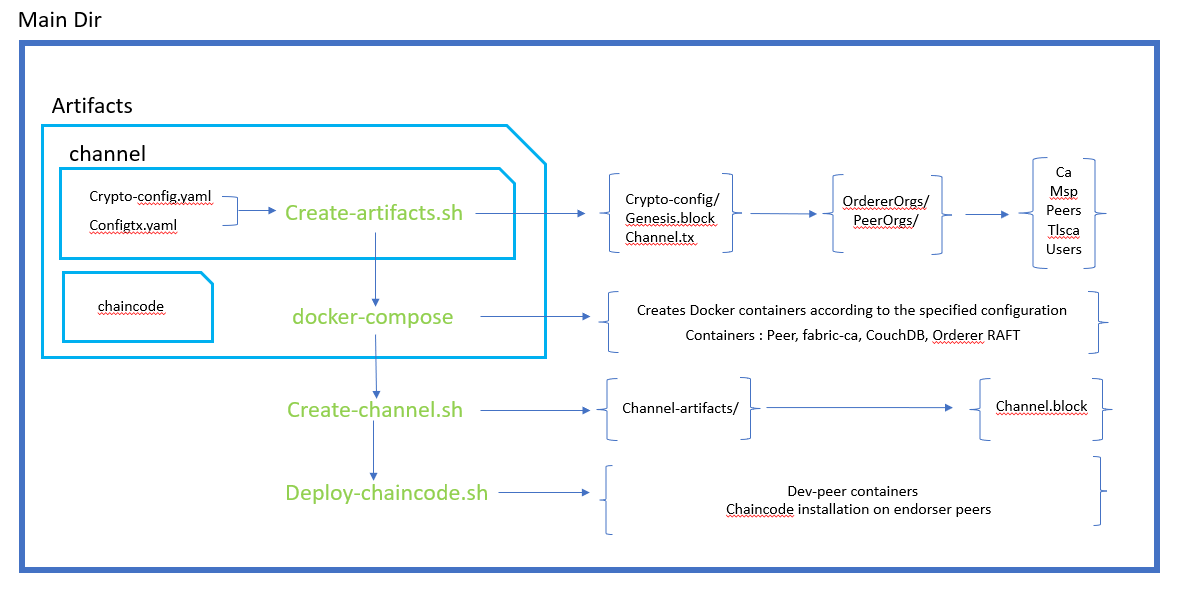


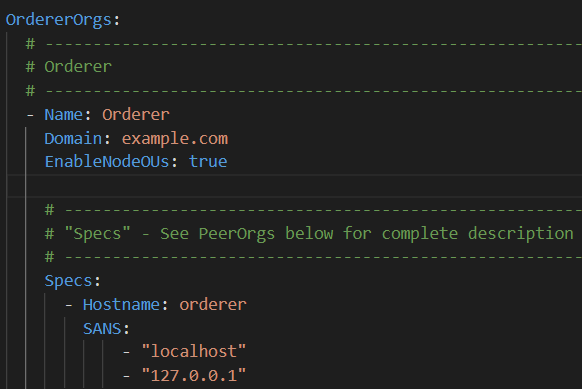
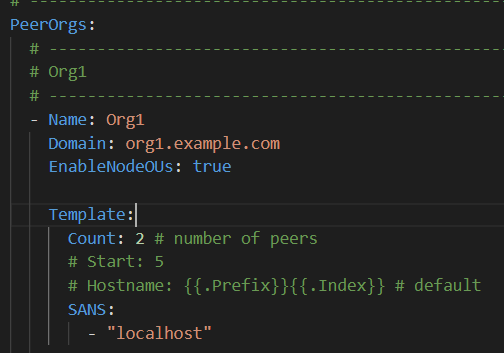
Fig. 1.1

The main directory contains a folder named artifacts which is responsible for generating the crypto materials. This folder contains the configurations for running the network and channel. This folder also contains the chaincode src to be packaged.

The Execution starts from the “**create-artifacts.sh”** file which uses two files for its configuration.

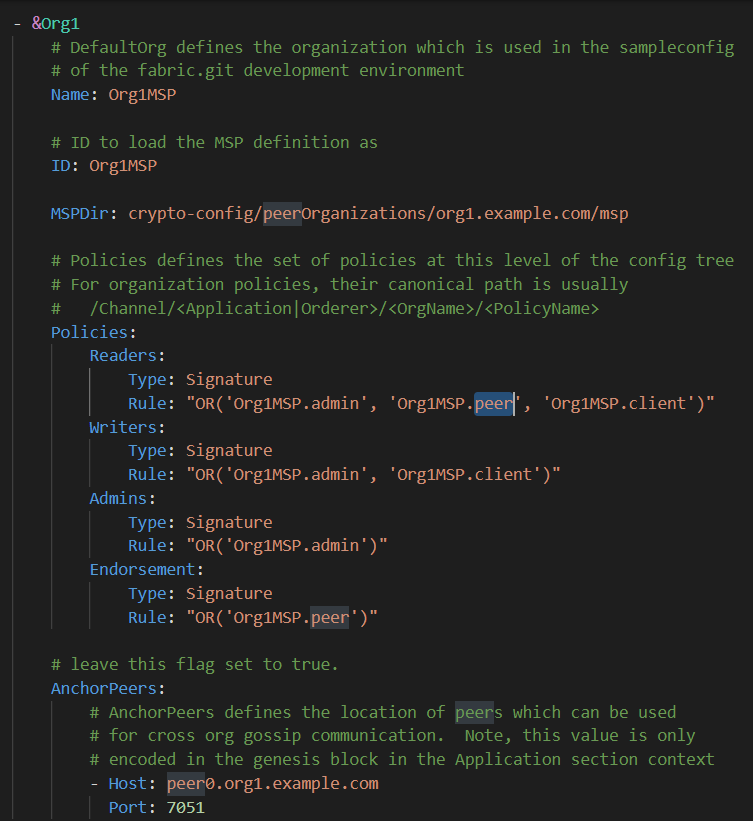
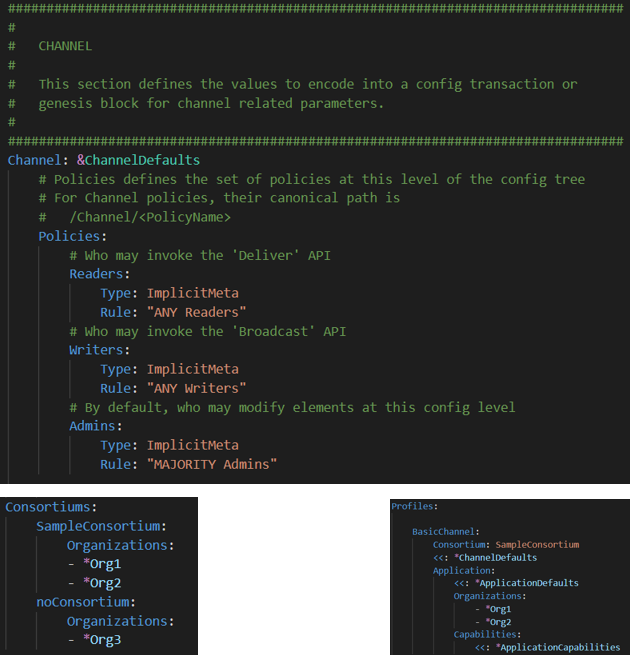
**Network Configuration :**

**Crypto-config.yaml** file contains the network configuration like the number of organizations this network should have, and the number of peers each organization has, number of orderers the RAFT should have and the number of users for each organization.

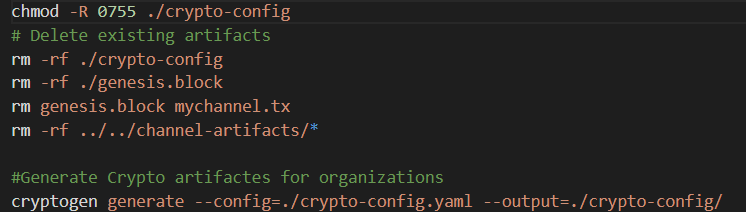
# **Channel Configuration :**

**Config.tx** file has the MSP configuration, Policies and Anchor peers configuration for each organization. It also contains the channel configuration, orderer configuration, Application and consortium configurations.

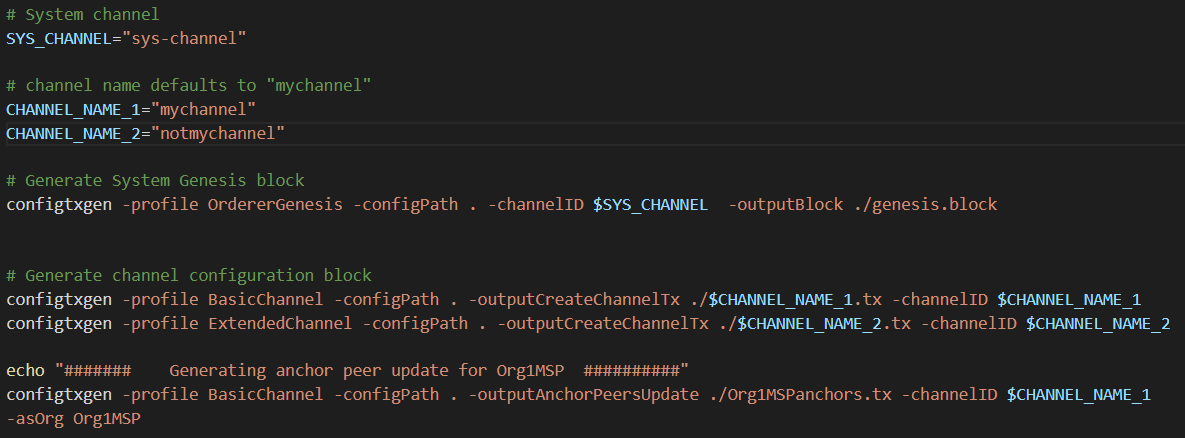
 

**Create-artifacts.sh**

As discussed earlier, the execution starts from this file. It first removes any existing artifacts and uses the above mentioned file crypto-config.yaml to generate crypto artifacts using the **cryptogen** fabric binary. We mention the output to be **crypto-config** folder using the –output flag.

****

Then we use the **configtxgen** fabric binary to generate genesis block and channel.tx utilising the **configtx.yaml** file, system chanel and the desired channels. After this we update one of the peers of the organization to be the anchor peer for this organization. Anchor peers are used for inter-organization communication. i.e., communications between peers of different organizations.

****

# **Docker-compose**

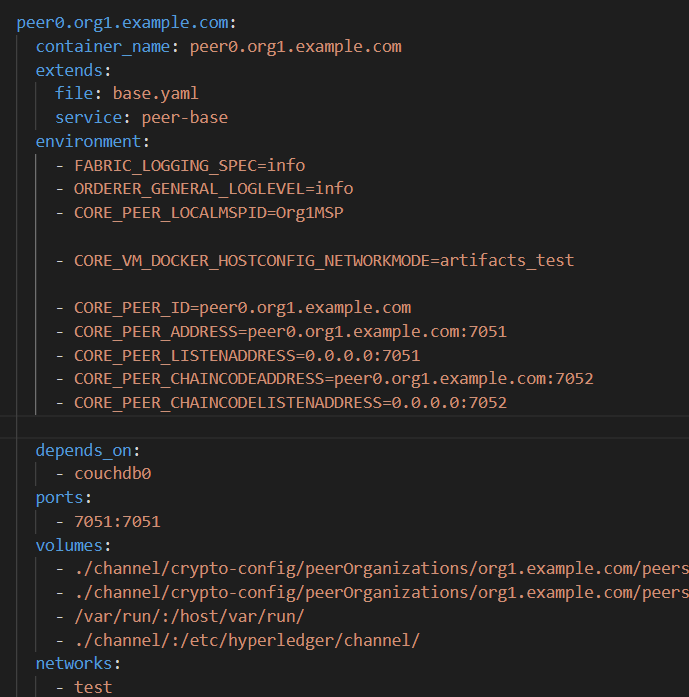
Docker Compose is a tool for defining and running multi-container Docker applications. It allows you to describe the services that make up your application in a YAML file and then spin up all the containers with a single command. Docker Compose provides an easy way to manage the container dependencies, configuration, and networking for multi-container Docker applications.

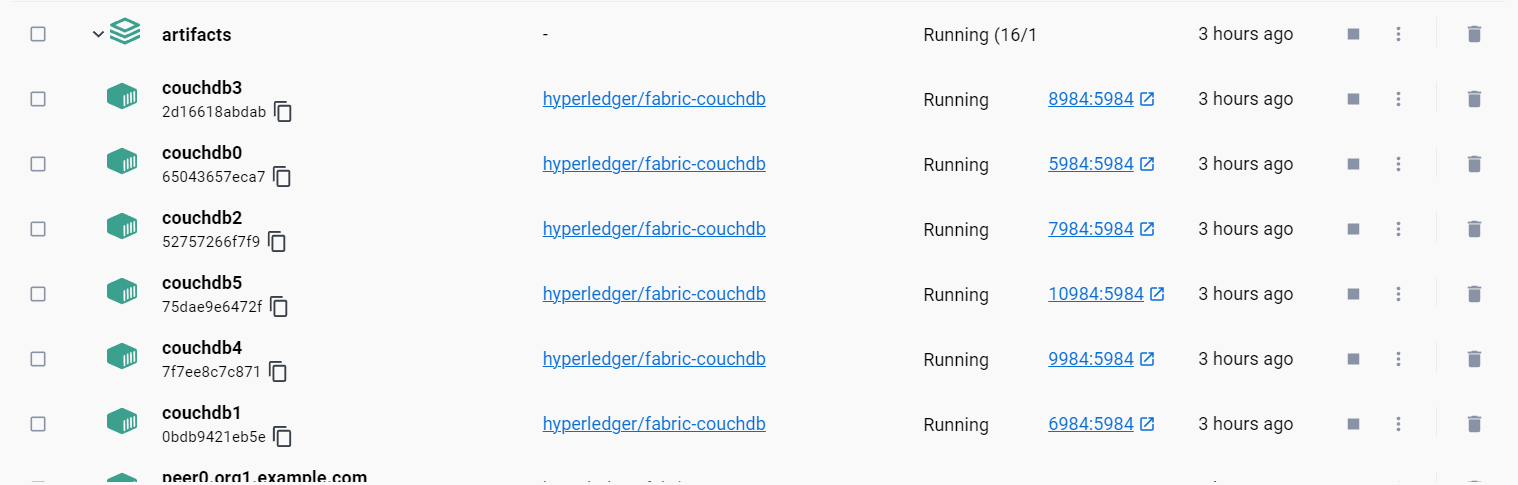
We use docker-compose to bring up the Hyperledger network with generated configurations from the before step. This step runs the following containers.

1. Certificate Authority (CA) – one for each organization
2. Orderer RAFT – default 1 : can be configured in the crypto-config.yaml
3. CouchDB – one for each peer in an organization
4. Peer – default : 2 per org : can be configured in the crypto config.yaml

For every container we mentioned above, we specify the following configurations:

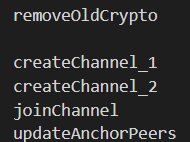
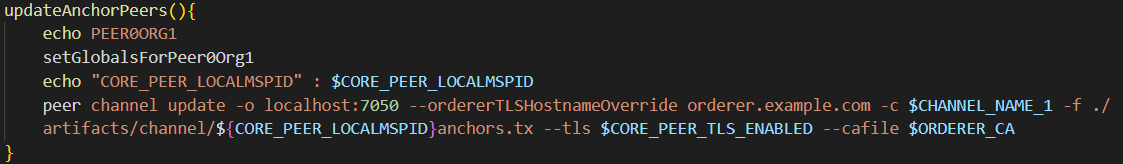
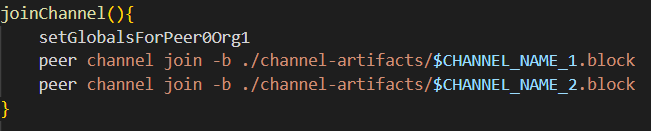
1. Docker Image
2. Environment variables
3. Ports
4. Command to start
5. Volumes to map (if any)
6. Container name
7. Host name
8. Networks
9. Depends on (if a container depends on any running containers)

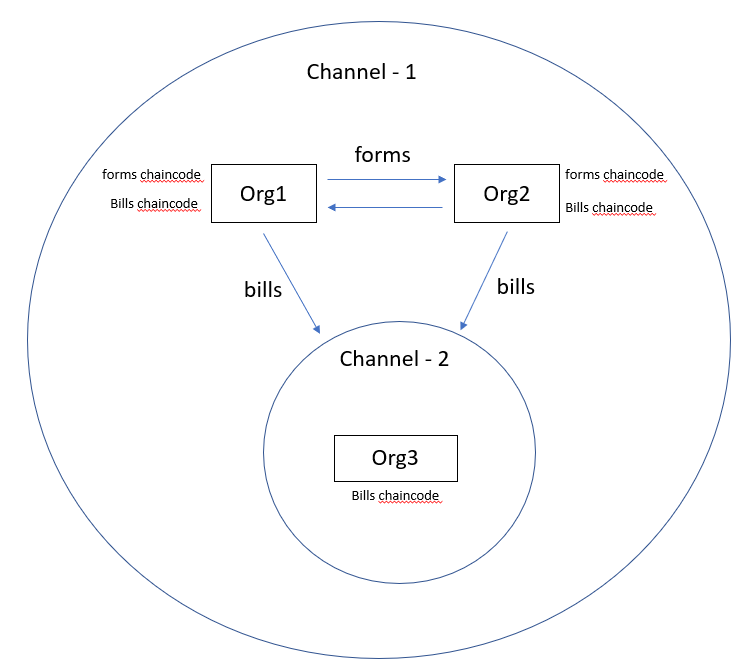




# **Create-channel.sh**

This file removes any existing artifacts found in channel-artifacts directory, creates the channels for each channel.tx file found in the artifacts, joins the peers of organization to specified channels and updates the anchor peers accordingly. These actions are performed on every peer for an organization. So its important to set the global environment variables accordingly.

fig. 1.2

# **Deploy-chaincode.sh**

This file follows a number of steps to deploy chaincode onto the endorsing peers of each organization. We’ll go through each step in detail.

**Steps for Deploying chaincode**

1. packageChaincode 4. approveForMyOrg1

2. installChaincode 5. checkCommitReadyness

3. queryInstalled 6. approveForMyOrg2

7. checkCommitReadyness 14. chaincodeInvokeInit

8. approveForMyOrg3 15.chaincodeInvokeInit3

9. checkCommitReadyness 16. chaincodeInvoke

10. commitChaincode Definition 17. ChaincodeInvoke3

11. queryCommitted 18. chaincodeQuery

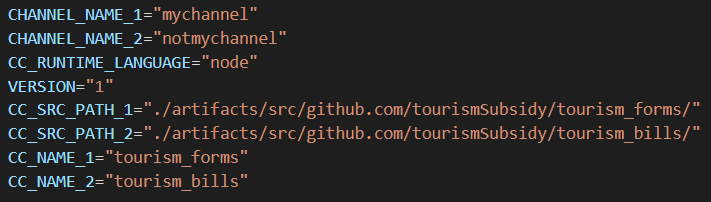
12. commitChaincodeDefination3 19. ChaincodeQuery3

13. queryCommitted3

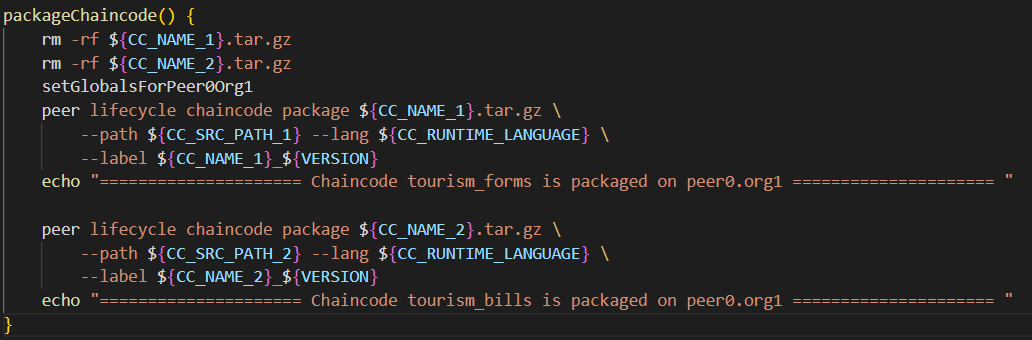
NOTE : The steps with convention [function\_name(3)] are duplicated for debugging. They are to be aggregated once the network is operational and stable.

**EXECUTION**

The globals being used for operations



1. **Package chaincode**

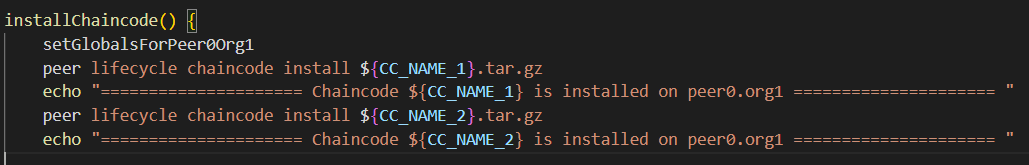


**OUTPUT**

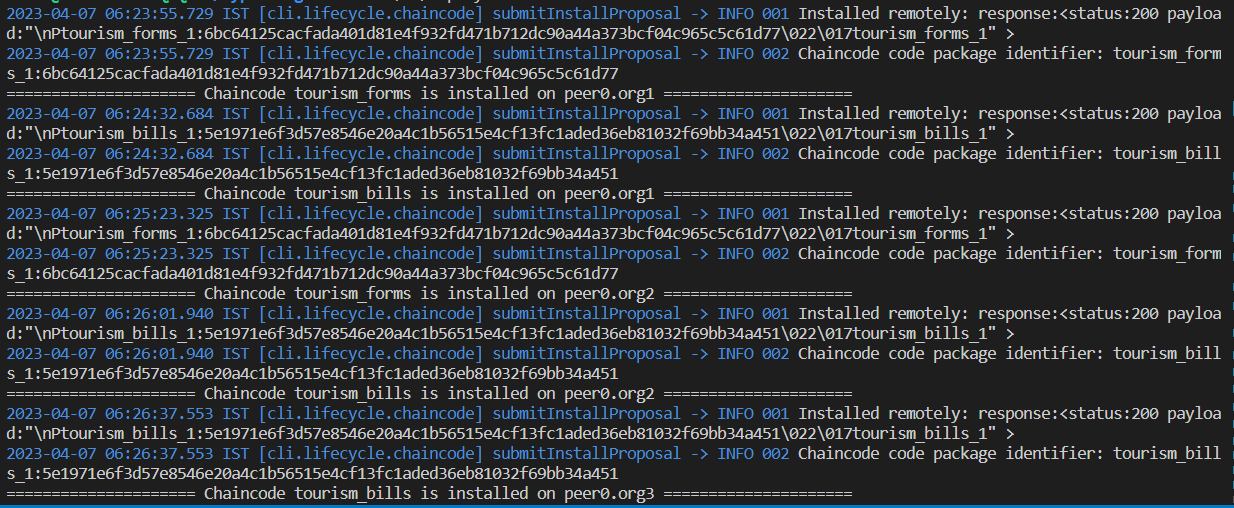
****

1. **Install Chaincode**

We have packaged two chaincodes. Tourism\_forms and tourism\_bills. As showcased previously in fig 1.2 , the organizations one and two with context govt. and studio are being installed with both the chaicodes so as to facilitate the functionality (forms) and third party services (bills). The chaincode is installed using the peer fabric binary.

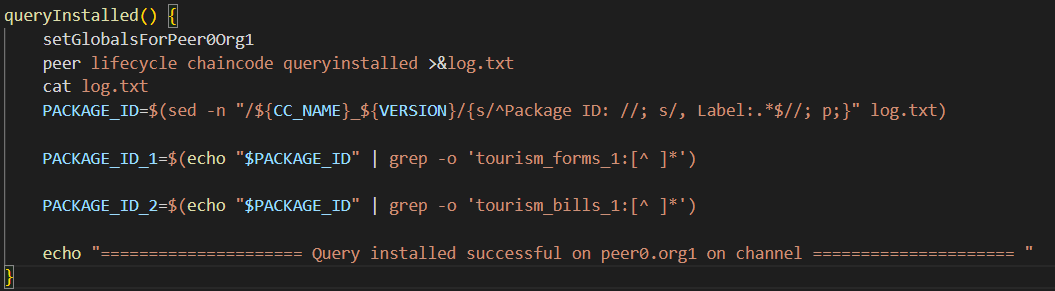


**OUTPUT**

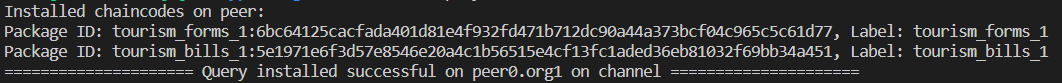
****

1. **QueryInstalled**

This function queries the installed chaincode for package ID, which is used for approving the chaincode on organizations in the next step. The package id is queried using the peer fabric binary. The package id for different chaincodes is parsed into separate variables for ease of access.

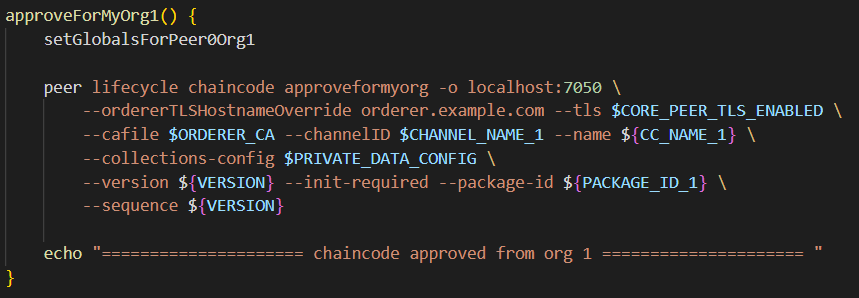


**OUTPUT**

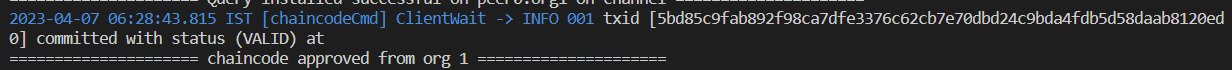


1. **ApproveForOrganization1**

The installed chaincode should be approved by an organization on the channel, for the peers to be able to interact with the chaincode on a channel. The chaincode is approved using the peer fabric binary.

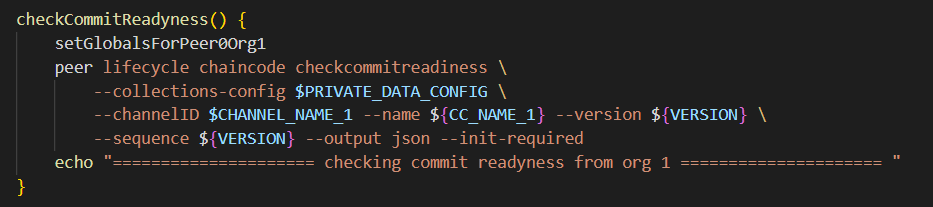
****

**OUTPUT**

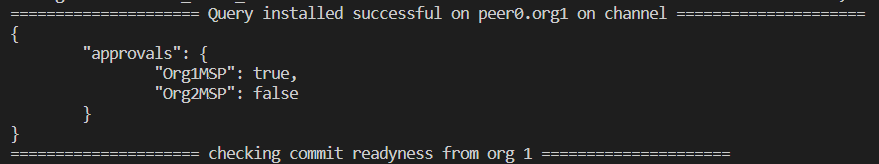
****

1. **CheckCommitReadyness**

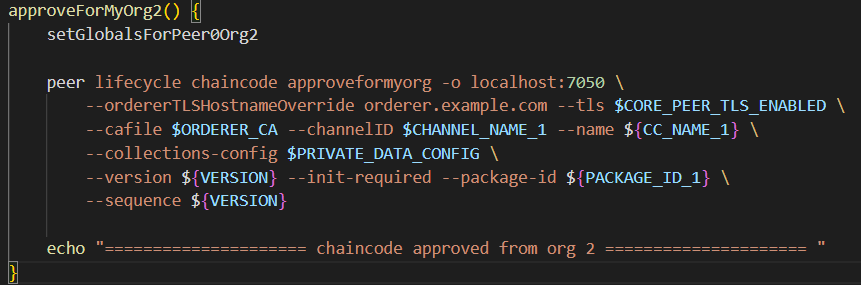
This step is to check the commit readiness of a chaincode. The chaincode should be approved by the organizations before commiting. The number of organizations required to approve a chaincode before it can be committed to the network is determined by the endorsement policy set for the chaincode. By default, the endorsement policy requires that all organizations defined in the channel configuration must endorse a transaction for it to be considered valid. However, this default policy can be customized according to the requirements of the application.

****

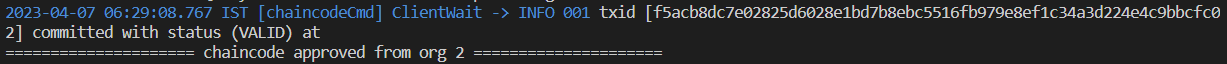
**OUTPUT**

****

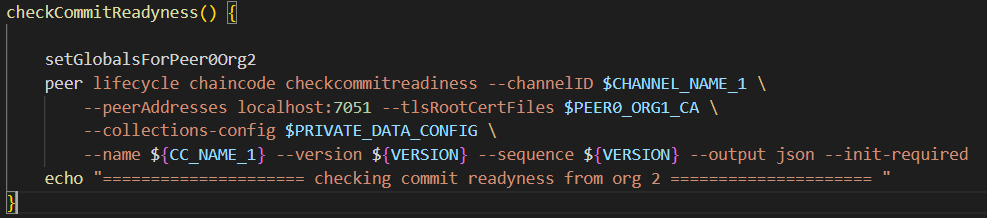
**7.ApproveForOrganization2**

****

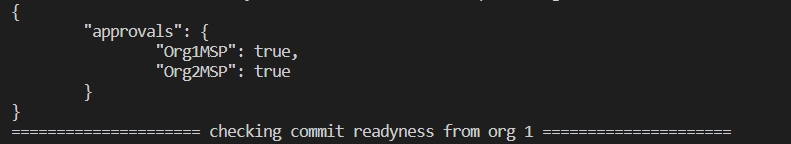
**OUTPUT**

****

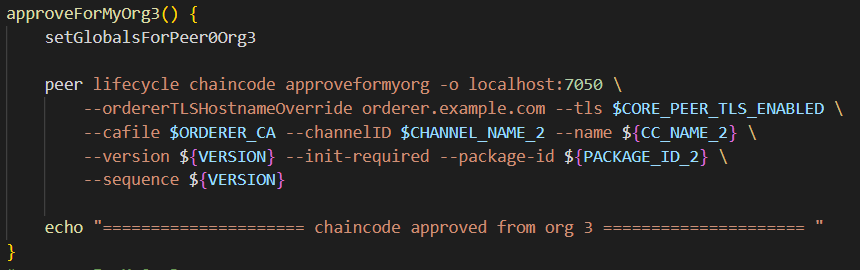
**8. CheckCommitReadyness**

****

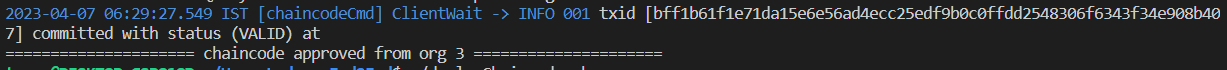
**OUTPUT**

****

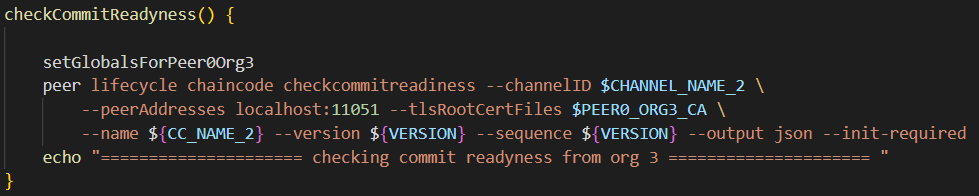
**9.ApproveForOrganization3**

****

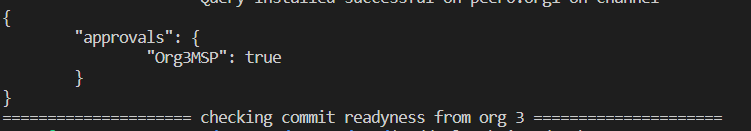
**OUTPUT**

****

**8. CheckCommitReadyness**

****

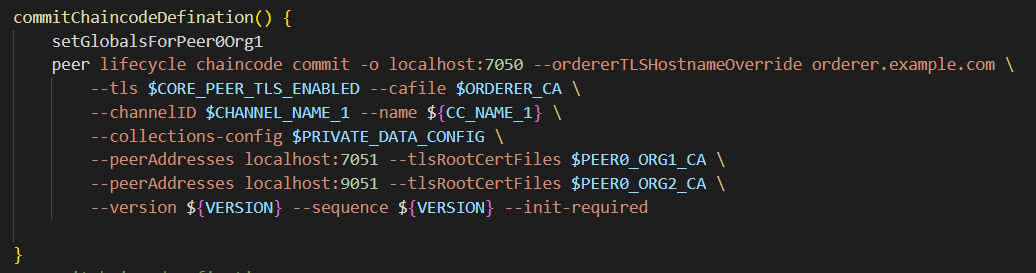
**OUTPUT**

****

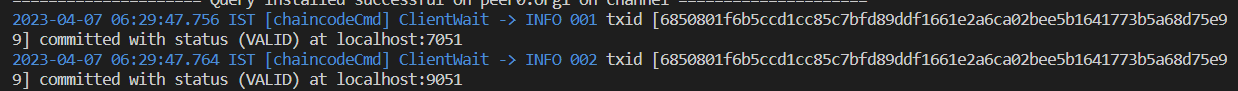
**9.CommitChaincode Definition**

Committing a chaincode in Hyperledger Fabric means that the chaincode is being deployed to the peers in the network and will be available for execution. When a chaincode is committed, it becomes a part of the blockchain ledger, and its code and data are stored on every peer that is a part of the channel.

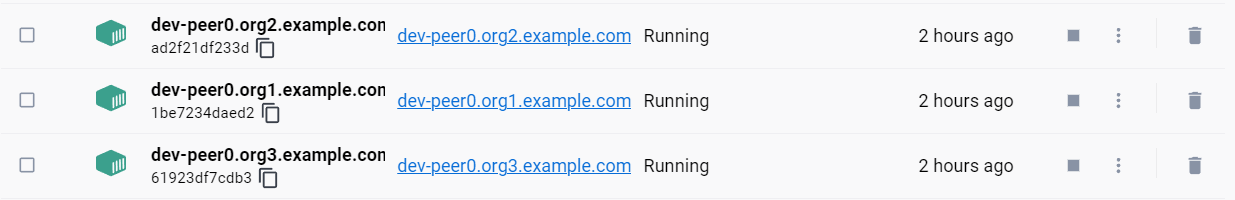
Committing a chaincode in Hyperledger Fabric brings up the development peers, the reason for this is that the peers need to be updated with the new chaincode definition and need to have the latest version of the chaincode container image. In development environments, we use dev peers that are configured to automatically download and install the latest chaincode container image whenever it is updated. This makes it easy to test and iterate on the chaincode during the development process.

****

**OUTPUT:**

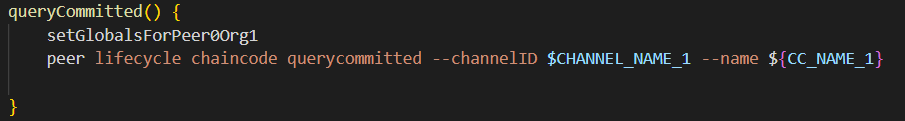
****

We can see the dev-peer containers on docker desktop

****

**10.Query Commited**

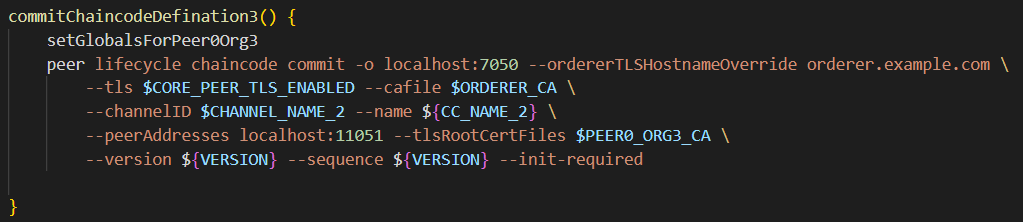
This step queries the status of chaincodecommit from the previous step, performed using the peer fabric binary.

****

**OUTPUT**

****

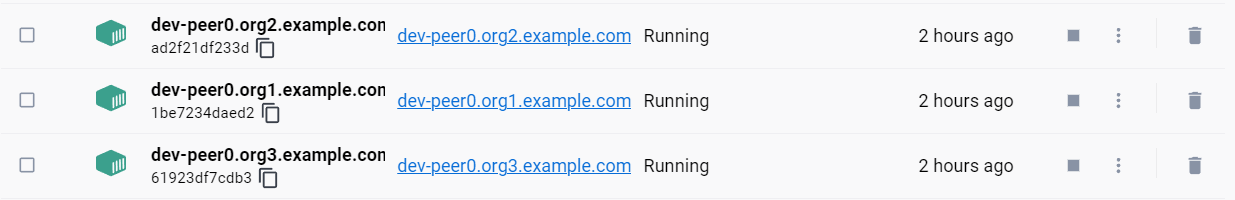
**11.CommitChaincode Definition3**

We perform the commitChaincode operation for organization 3 on channel 2, exactly the same way we did with organizations 1 and 2.****

**OUTPUT:**

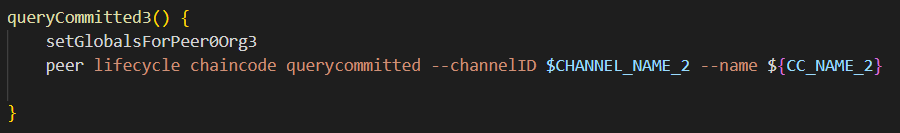
****

We can see the additional dev-peer for org3

****

**12. Query Commited**

This step queries the status of chaincodecommit from the previous step, performed using the peer fabric binary.

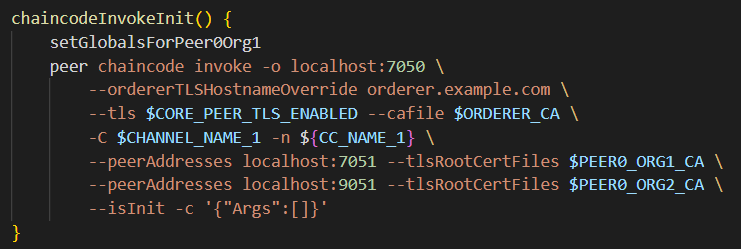
****

**OUTPUT**

****

**13. ChaincodeInvokeInit**

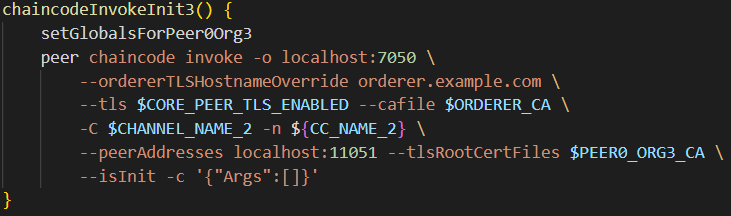
The Init function is a special function that is executed only once when a chaincode is first deployed to the network, and it is used to initialize the state of the chaincode. Once the transaction is endorsed, the chaincodeInvokeInit function returns a TransactionResponse object that can be used to track the status of the transaction. This object contains information about the transaction, such as its ID and the time it was submitted.



**OUTPUT**



**14. ChaincodeInvokeInit3**

We’re performing the same action for organization3 on channel 2.

**OUTPUT**

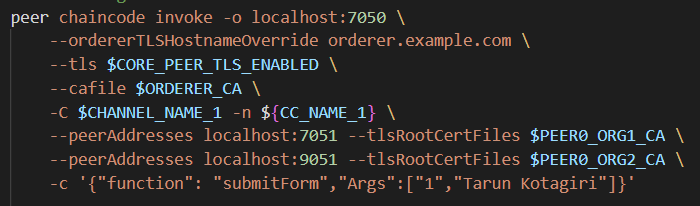


# **15. Chaincode Invoke**

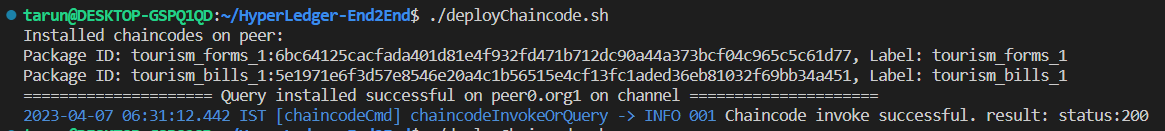
In Hyperledger Fabric, chaincode is the smart contract that runs on the blockchain network. Chaincode invocation is the process of executing the functions defined in the chaincode.

Here, we’re ivoking the function submitForm with form ID as 1 and Form Data as “Tarun Kotagiri”.

Later we’ll replace this with the ipfs cid of a file uploaded to Interplanatary File System (IPFS).

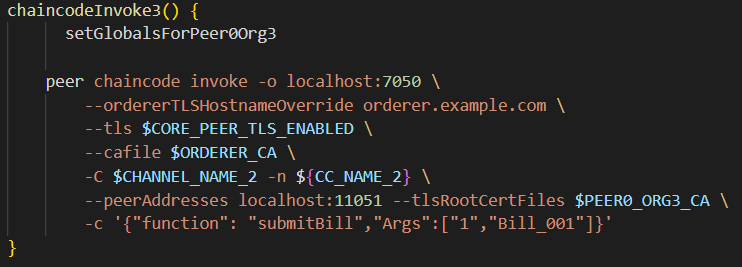


**OUTPUT**

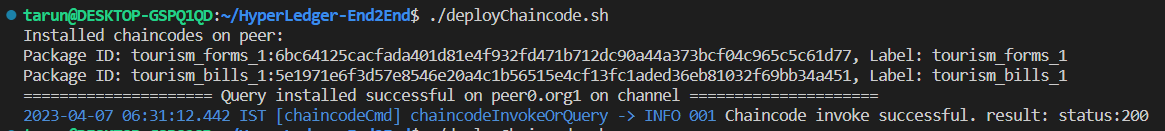


**16. Chaincode Invoke 3**

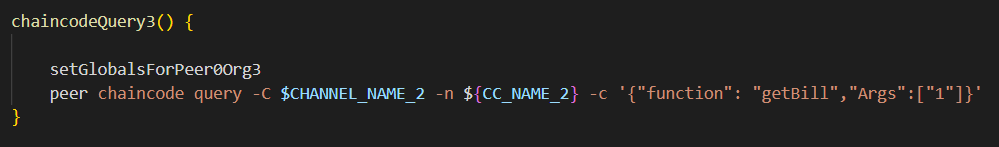
Here, we’re ivoking the function submitBill with Bill ID as 1 and Bill Data as “Bill\_001”.



**OUTPUT**



**17. Chaincode Query 3**

Here we’re quering the function getBill with bill ID as argument.

OUTPUT

