Redbooks

ibm.com/redbooks

Developing a Blockchain Business Network with Hyperledger Composer using the IBM Blockchain Platform Starter Plan

Vance Morris

Rohit Adivi

Ratnakar Asara

Matthew Cousens

Nick Gupta

Nicholas Lincoln

Barry Mosakowski

Hong Wei Sun

Cloud

System Networking

IBM

Redpaper

International Technical Support Organization

Developing a Blockchain Business Network with Hyperledger Composer using the IBM Blockchain Platform Starter Plan

May 2018

REDP-5492-00

Note: Before using this information and the product it supports, read the information in “Notices” on page v.

First Edition (May 2018)

This edition applies to Hyperledger Fabric v1.1.0, Hyperledger Composer v0.19.4, and the beta version of IBM Blockchain Platform Starter Plan.

This document was created or updated on May 30, 2018.

Contents

Notices

Trademarks

Preface

Blockchain has emerged as a disruptive technology in the areas of trading assets and sharing information. It has the capability to transform many industries, professions, and aspects of life. The focus of this IBM® Redbooks® publication is to help developers build blockchain solutions and use IBM Blockchain Platform to start, test, and move applications into production.

This publication covers some blockchain for business use cases. It also describes how to get started in defining, developing, and deploying a Hyperledger Composer business network to Hyperledger Fabric, both locally on a workstation and remotely on the IBM Blockchain Starter Plan. A fund clearing business network is used as an example scenario for blockchain and this source code is available for download, testing, and use.

This paper is part one of a series of papers and educational materials. Later materials will describe how to use IBM Blockchain Platform to test and scale your business network, to integrate more completely with a COBOL business application running in IBM CICS®, and to manage changes to your business network in a production environment.

Authors

This paper was produced by a team of specialists from around the world working at the International Technical Support Organization, Raleigh Center.

Vance Morris is a Staff Software Engineer at the IBM Systems Integrated Solutions Test group. Having held customer advocate positions in the IBM Z® Dallas ISV Center, IBM Ecosystem Development, and Z Academic Initiative, he remains dedicated to the success of every client. He is passionate about mainframes and Linux, and loves to find ways to bridge the two together.

Rohit Adivi is a consultant for IBM USA. He has been working on Hyperledger Fabric and IBM Blockchain Platform for the past year. He has helped develop content for tutorials in Hyperledger Fabric documentation. He holds a Bachelor's degree from Osmania University in India.

Ratnakar Asara is a consultant for IBM USA. He has two years of experience in Hyperledger Fabric and IBM Blockchain Platform technologies. He has a Bachelor degree in Computer Science from Jawaharlal Nehru Technological University, India. His areas of expertise include Fabric Blockchain, Mobile Platforms, and IOT.

Matthew Cousens has spent over 15 years at IBM, working on testing z/OS® in every test phase. His current position is with the z/OS Platform Evaluation Test team, where he is responsible for the strategy, design, and development of integration test workloads and applications. He holds a Master of Business Administration degree from Marist College with a certificate in Executive Leadership. Matt presents on software testing as often as he can and is particularly passionate about working with the next generation of technical experts.

Nick Gupta is an IBM Blockchain Business Development Executive - Focusing on Ecosystem Development, located in Brooklyn, New York, US. Nick has been working on blockchain since April 2016. Previously, he worked for IBM Systems, with prior experience in Information Management and data science.

Nicholas Lincoln is a software engineer working on Hyperledger Composer at the Hursley Laboratories in the UK. He holds a PhD in multi-agent systems and a Master's Degree in Aerospace Engineering, both from the University of Southampton (UK). Nick joined IBM in 2012 where he began his career working on IBM Integration Bus, which is where he fostered his passion for business systems. Nick is a test evangelist and a prolific inventor.

Barry Mosakowski is Lead Quality Engineer for IBM Blockchain Platform and the Hyperledger Fabric Project. He has over 20 years with IBM working on TCP/IP in the Communications Server organization and as a lead on the IBM DataPower® integration product. He has been working on blockchain for over two years, and has helped design the Continuous Integration strategy as well as function, system, and performance testing.

Hong Wei Sun is an advisory software engineer at the IBM China System Lab. He joined IBM in 2011 and has over 18 years of experience in software development and customer support. He has a Master's degree in computer science from Beijing University of Technology, China. His areas of expertise include system and application software architecture, IBM z/OS workloads, application integration, performance analysis, IBM Z, Linux, IBM WebSphere® Liberty, CICS, Spark, and Hyperledger Fabric.

Thanks to the following people for their contributions to this project:

Deana Coble
Project Leader, IBM Redbooks, Technical Content Services at Raleigh Center

Ernest A. Keenan
Content Specialist, IBM Redbooks Austin Center

Scott Loveland
Senior Technical Staff Member (STSM), IBM Poughkeepsie Center

Ryan Parece
Staff Software Engineer for Linux on Z and Blockchain, IBM Poughkeepsie Center

Mark Parzygnat
Program Director, IBM Blockchain, Raleigh Center

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our papers to be as helpful as possible. Send us your comments about this paper or other IBM Redbooks publications in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:

  ibm.com/redbooks

- Send your comments in an email to:

  redbooks@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- Find us on Facebook:

  `http://www.facebook.com/IBMRedbooks`

- Follow us on Twitter:

  `http://twitter.com/ibmredbooks`

- Look for us on LinkedIn:

  `http://www.linkedin.com/groups?home=&gid=2130806`

- Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

  `https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm`

- Stay current on recent Redbooks publications with RSS Feeds:

  `http://www.redbooks.ibm.com/rss.html`

1

Overview

Blockchain has emerged as a disruptive technology for trading assets and sharing information. It has the capability to transform many industries, professions, and aspects of life. This paper can help developers build blockchain solutions and use IBM Blockchain Platform to launch, test, and move applications into production.

It covers everything that you need to get started, including the following topics:

- Why blockchain is used for business applications
- Key blockchain concepts and the components of Hyperledger Fabric, the codebase that underpins IBM Blockchain Platform networks
- Descriptions of a number of blockchain use cases and an in-depth analysis of developing a business network for bilateral funds clearing and settlement between banks
- How to develop a business network using Hyperledger Composer
- How to deploy to a live blockchain network using the IBM Blockchain Platform Starter Plan
- First steps in designing a system integration between a blockchain business network and an existing business application running in IBM CICS on z/OS

This paper is part one of a series of papers and educational materials. Later materials describe how to use IBM Blockchain Platform to test and scale your business network, to integrate more completely with a COBOL business application running in CICS, and to manage changes to your business network in a production environment.

This chapter includes the following sections:

- Why blockchain?
- What is blockchain?
- Use cases for blockchain in business
- Linux Foundation’s Hyperledger Project
- Hyperledger Fabric
- Hyperledger Composer
- IBM Blockchain Platform

1.1 Why blockchain?

Ledgers are at the core of every business. Ledgers store the flow of assets, such as payments to suppliers, taxes owed, and goods delivered. They also store the firm’s current balances. All of a firm’s revenue, relationships, and assets are recorded on a core system of ledgers. For centuries, these records were stone, clay, or paper-based. However, these records were among the first systems to be automated and moved to computing systems.

Even with the digitization of ledgers, many transaction-related tasks remain manual and outdated. Each shipping container moved by freight requires a stack of paperwork that adds significant processing time. Each time that you open a new bank account or visit a new doctor, you must share personal information using redundant and insecure forms. The transfer of financial assets between parties often takes days to settle.

If ledgers, financial records, personal records, and inventory have been digitized, why do these inefficiencies remain? The reason is that the move to digitization has generally only occurred within an organization, rather than between organizations. In addition, the systems that were created often cannot communicate with each other. The result is that modern business-to-business processes carry on with inefficiencies that date from when ledgers were on paper.

1.2 What is blockchain?

Blockchains are distributed ledgers that allow multiple parties to access and update a single version of a ledger while maintaining shared control. A *business network* describes any group of organizations or individuals that connect with a desire to transfer or share assets. Those assets can be tangible, such as food or manufactured goods, or digital, such as music or data. By tracking items using a common, shared ledger that is distributed across the business network, assets can be transferred between members, with each member having a record of the transaction and access to the latest version of the ledger.

Blockchains establish trust across a business network through the combination of a distributed ledger, smart contracts, and consensus. The ledger contains the current state of assets and the history of all transactions. Transactions can only be added to the ledger, not removed. Past transactions are protected with cryptography so that they cannot be successfully tampered with. The blockchain also makes changes to the ledger final and immutable, allowing the ledger to be the source of truth within the network.

The strongest use cases for blockchain involve business networks, which add additional trust through the benefits of *immutability*, *provenance*, *consensus*, and *finality*. Immutability means that the historical record of transactions cannot be altered. Provenance means that the origin of any assets contained in the ledger is known. Changes to the ledger require approval by participants according to an agreed upon endorsement policy. This goal is achieved through consensus, in which participants of the network endorse that the transaction is valid and come to agreement on the updated state of the ledger. Finality provides ease of mind because each participant is assured that their copy of the ledger matches all other copies, and that transactions contained in the blockchain have been committed faithfully.

A *smart contract* is an application that contains business logic. Smart contracts define all access to the ledger and are invoked by participants to query or update values of assets.

1.3 Use cases for blockchain in business

Many industries use blockchain technology today, including supply chain, trade finance, healthcare, identity access management, and banking:

- Trade finance along with supply chain: Financial institutions, logistics companies, and IBM are working together on a new global system for trade finance using blockchain technology that is designed to track goods through the supply chain. Current payment resolution is cumbersome and largely a paper-based process that leaves companies waiting weeks or longer before they are paid for their goods. Because all of the members of a blockchain network have an identical ledger and smart contracts can trigger ledger updates automatically, payments can be triggered automatically upon delivery, making common disputes a thing of the past.
- Healthcare: Blockchain reduces the barriers that are involved in complex data sharing agreements between hospitals, physician providers, public health departments, and the Centers for Disease Control (CDC), allowing organizations to quickly and securely move patient data in a transparent and legally compliant manner. Healthcare use cases often converge with identity use cases (noted in more detail below), allowing individuals to easily approve the sharing of pertinent information without sharing non-essential or unneeded information. For example, if an X-ray is required, the parties interested in the result might include the insurance provider, a general physician, the X-ray technician, and the individual. However, the X-ray technician does not need to know that the insurance is provided by the employer. All the technician needs to know is you are a valid customer and are covered for the procedure.
- Identity: Blockchain provides encryption of data in the ledger and provides fine-grained controls that determine who can access the data. Imagine being able to decide exactly who can see every bit of data that you owned. For example, many people carry identification such as a driver's license, which contains information like name, address, and other data. At times, we provide this identification to prove our age to someone, and they need only to know one piece of this information: Our birthday. Keeping your identity on a blockchain allows you to delegate access of pertinent information when needed, while keeping the remainder private, thus reducing the risk of stolen or misused credentials.
- Fund Clearing: Multi-currency cash settlement systems are complex, involving a central clearing house and many disparate systems that must interact to detect the exchange rates and settle currency transfers. This paper focuses on implementing a fund clearing application as an example. We use a blockchain to resolve the exchange of funds in a timely, secure, and transparent manner.

  The Composer business network allows network participants to process inter-participant payments, thus replacing a traditional clearing house. We assume that the participants within the network are financial institutions that accumulate and submit one or more transfer request items to other participants. These transfer requests are placed into the business network, and when a batch transaction is invoked, all transfer requests for each banking participant are accumulated into a series of aggregated batch transfers.

  Each batch transfer details the required net settlement between each banking participant for the contained array of transfer request items, based on the currency of the creditor bank. Because of the popularity of IBM Z in banking, we describe integrating with a full-featured banking application running in CICS.

Note: Chapter 2, “Use case development process” on page 7 begins the implementation scenario for the fund clearing application.

1.4 Linux Foundation’s Hyperledger Project

The Linux Foundation’s Hyperledger Project is commissioned to incubate blockchain technology built for business. Since its founding in 2016, the Hyperledger Project contains almost a dozen projects, and its roster has grown to over 230 companies and organizations. In addition to being a founding and premier member of the project, IBM is a major active contributor to many projects therein. Hyperledger Fabric and Hyperledger Composer are two projects that IBM contributes to, and are great places to get started building your first application.

1.5 Hyperledger Fabric

Hyperledger Fabric is a platform for deploying blockchain networks. The ledger, smart contracts, and consensus between members are maintained by using Fabric protocols. Fabric is designed for business use cases where the blockchain is operated by a set of known, identified, and often vetted participants. This capability is known as a *permissioned blockchain*. A permissioned blockchain provides a way to secure the interactions among a group of entities that know each other, have common business interests, and want to manage a decentralized network (rather than turning management of their ledgers over to a single party). By relying on the identities of the peers, a permissioned blockchain can use traditional crash fault tolerant (CFT) or Byzantine fault tolerant (BFT) consensus protocols that are used by many other distributed programs.

Hyperledger Fabric offers high levels of performance, protection, and transaction privacy. Fabric is the leader in blockchain for business, and can enable modular components to suit various use cases. The IBM Blockchain Platform is built upon Hyperledger Fabric and, as a result, a basic understanding of Fabric's components and architecture is important for developers and decision makers to understand.

Fabric consists of the following basic components:

- A *ledger* on Fabric consists of the world state and the blockchain. The world state contains the status of all assets that are tracked on the ledger (who owns a particular asset, for example), while the blockchain contains a history of all state changes. Ledgers are replicated across a channel (more on channels below) and stored on peers.
- *Peers* are the transaction endpoints for organizations and make up much of the physical structure of a network. They are maintained by members (organizations) whose identities are known by the blockchain network. Peers can maintain multiple ledgers (they have one for every channel they are a member of) and endorse transactions.
- A *channel* contains a subset of network members who want to communicate and transact privately. Ledgers are channel specific (that is, every channel has a separate ledger). Only the peers on a channel can see the assets and transactions for its ledger. As a result, channels ensure privacy for participants within the network.
- Hyperledger Fabric smart contracts are implemented in *chaincode*. When an application needs to interact with the ledger, it invokes these contracts by sending transactions into the Fabric network. This is the case because chaincode predominately interacts only with the database component of the ledger and not the historical transaction log.
- The *Ordering Service*, usually composed of multiple orderers, provides consensus and ordering of transaction. It does so by bundling transactions into blocks, which are then added to the blockchain.

- The *certificate authority (CA)* identifies all entities in the network: Peers, the ordering service, and the participants who are submitting transactions and accessing the ledger. These identities are provided and secured by using a public key infrastructure (PKI). Peers use the CA to cryptographically sign transactions and contracts, whereas participants use the CA to prove that they have a right to access the network.
- The *Hyperledger Fabric Client SDKs* enable interaction between your client application and your blockchain network. With support for multiple languages, the SDK contains APIs that allow an app to connect to and to access the smart contracts and the ledger for the channel the peer is on.

1.6 Hyperledger Composer

Composer is an open source development toolset and framework that accelerates the time it takes to write a blockchain application. Instead of developing smart contracts from scratch, Composer provides a convenience layer and business-level abstractions to implement smart contracts on Fabric. Composer also makes it easier for you to connect to your business network from a web or mobile application.

Composer helps you define your business network using a custom modeling language and a small set of primitives, and then deploy the business network to Hyperledger Fabric. The business network consists of *participants*, *assets*, and the *transactions*. Also included are *access control lists* that restrict access to transactions or data.

This paper guides you through developing a fund clearing business network using a modern IDE, and a Composer Playground.

1.7 IBM Blockchain Platform

IBM Blockchain Platform (IBP) is an IBM Cloud™ offering that is built upon Fabric. It automates the deployment of Fabric and provides tools for governing your network, taking you from initial development of your business network to putting your application in production. The IBM Blockchain Platform helps you create a blockchain network with a few clicks and provides an easy-to-use interface for creating channels and deploying chaincode. When you are ready to grow your network, IBP also makes it easy to invite new members, create channels on your network, customize governance policies, manage the identity credentials of network participants, and much more.

The IBM Blockchain Platform comes with multiple service levels appropriate for different stages of development. The Starter Plan, currently in beta, is ideal for those who are developing and testing their business network. IBP Starter Plan provides a fully configured Fabric network with two-member organizations. Each organization contains one peer, a CA, and an ordering service that is shared by the network. It offers sample applications to help you get started and quickly experience trading assets on a live network. This paper takes you through how to use the Starter Plan to start your own network and to deploy the fund clearing business network.

The Starter Plan is a great tool for development, but should not be used for production. This paper shows you how to use the Starter Plan to develop, test, and harden your blockchain business network. You can then move your network into production using the IBP Enterprise Plan. The Enterprise Plan offers the high availability and live updates that you will need for a production network, runtime isolation, and end-to-end tamper protection to properly secure your intellectual property.

2 Use case development process

This chapter provides a description of how you might get started in developing a fund clearing business network using Hyperledger Composer. It starts with the generation of a template network to build from, then adds tests to define the intended function of the business network and iteratively adds code to fulfill the intended function. The addition of access control lists (ACLs) is reserved for the final stage of developing a business network because the application of ACL rules might hide errors within the transaction logic.

This application is available for download and can be used in Proof of Concept demonstrations and testing scenarios. As described in 1.3, “Use cases for blockchain in business” on page 3, multi-currency cash settlement systems are quite complex. They involve a central clearing house and many disparate systems that must interact to realize the exchange rates and settle currency transfers. This application uses blockchain to resolve the exchanges in a timely, secure, and transparent movement of the funds.

The Composer business network allows network participants to process inter-participant payments, thus replacing a traditional clearing house. We assume that participants within the network are financial institutions that accumulate and submit one or more transfer request items to other participants. These transfer requests are placed into the business network. When a batch transaction is invoked, all transfer requests for each banking participant are accumulated into a series of aggregated batch transfers. Each batch transfer details the required net settlement between each banking participant for the contained array of transfer request items, based on the currency of the creditor bank.

This chapter includes the following sections:

- Developing With Hyperledger Composer
- Generating the Business Network Template
- Model file definition
- Test-based development
- Implementing the business network logic
- Adding ACL rules
- Debugging

2.1 Developing With Hyperledger Composer

Hyperledger Composer provides multiple tools to facilitate the development of business networks. The tools permit both online and local development.

The Hyperledger Composer Playground provides a user interface for the configuration, deployment, and manual testing of a business network. Advanced Playground features permit users to manage the security of the business network, invite participants to business networks, and connect to multiple blockchain business networks.

Hyperledger Composer Playground is not intended to facilitate source code version control or the development of automated testing suites for use within build automation. For these requirements, you can use various version control tools and code editors, which have tools to provide syntax highlighting and scenario-based testing.

2.2 Generating the Business Network Template

A business network definition has the layout shown in Figure 2-1.

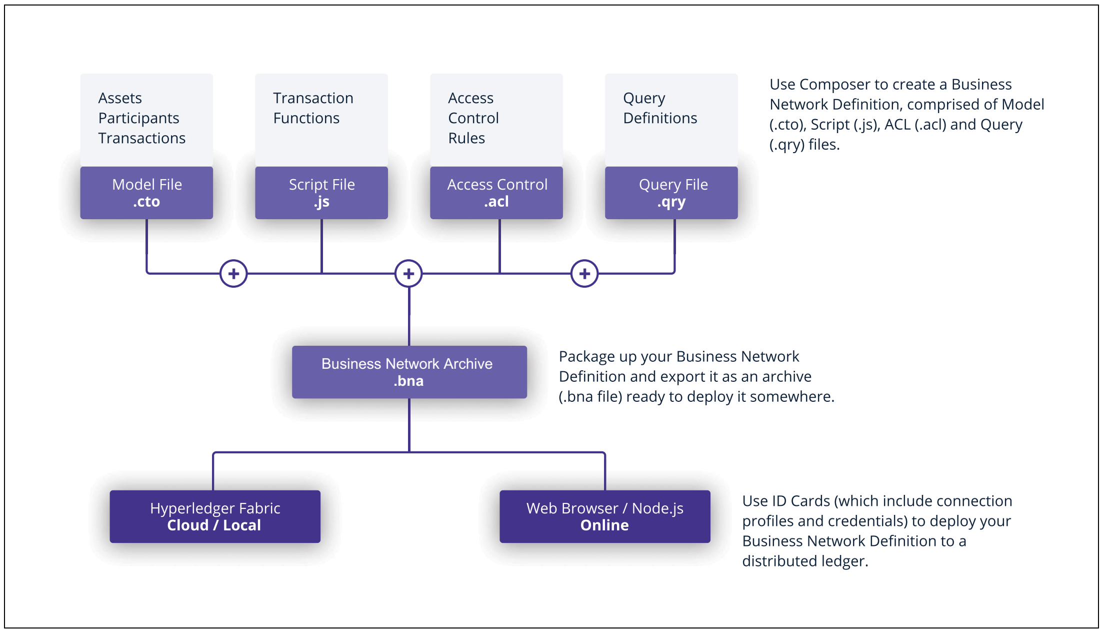

Assets
Participants
Transactions
Transaction
Functions
Access
Control
Rules
Query
Definitions
Use Composer to create a Business Network Definition, comprised of Model (.cto), Script (.js), ACL (.acl) and Query (.qry) files.
Model File .cto
Script File .js
Access Control .acl
Query File .qry
Business Network Archive .bna
Package up your Business Network Definition and export it as an archive (.bna file) ready to deploy it somewhere.
Hyperledger Fabric Cloud / Local
Web Browser / Node.js Online
Use ID Cards (which include connection profiles and credentials) to deploy your Business Network Definition to a distributed ledger.

Figure 2-1 Business layout diagram

The fastest way to create a new business network definition is to either `git clone` an example, or to use the Hyperledger Composer Yeoman generator to create a skeleton business network.

You can read more about creating the business network structure with Yeoman in the online Developer tutorial for creating a Hyperledger Composer solution.

After the business network structure has been generated, open it with a code editor to continue development. The preferred code editor is VisualStudio, which has a plug-in available to provide syntax highlighting.

2.3 Model file definition

Hyperledger Composer uses the abstraction of *Assets*, *Participants*, and *Transactions*. We capitalize these concepts because they are represented by concrete classes within Composer. You can read more about these data types in the Composer API. Within a business network, Participants use Transactions to modify the state of Assets. Therefore, all development proceeds from the definition of these abstract types, which capture the essence of the business network being investigated.

The fund clearing network considers the bilateral clearing of funds between Participants. Therefore, the example has a single participant type named `BankingParticipant`.

The business use case requires the creation of records to indicate a monetary transfer between two `BankingParticipants`. These two can be netted to create a single batch transfer to maximize the liquidity of all member `BankingParticipants`. Thus, we have two assets: `TransferRequest` and `BatchTransferRequest`.

A series of transactions enable `BankingParticipants` to create `TransferRequest` assets, net all relevant `TransferRequests` into a `BatchTransferRequest`, and then fulfill the settlement of the `BatchTransferRequest`. The transactions proposed to facilitate this process are `SubmitTransferRequest`, `CreateBatch`, `MarkPreProcessComplete`, `CompleteSettlement`, and `MarkPostProcessComplete`.

Because `BankingParticipants` need to be notified of any relevant `BatchTransferRequests` that have been created, an event `BatchCreatedEvent` is determined to be necessary.

Additional information regarding key concepts can be found on the Hyperledger Composer website.

2.4 Test-based development

The preferred Composer development methodology is test-driven development (TDD). Beginning a project by first outlining the required business logic in test cases allows you to then implement business logic, ensuring that all the tests pass. Because the model file has been defined, it is possible to progress in this manner.

Testing of the business network relates to these objectives:

- Testing the logic files that define the system transactions and provide concrete implementations
- Testing of the query files
- Testing the access permissions defined by the ACL rules

These objectives can be achieved though any combination of unit, functional, and scenario-based tests.

2.4.1 Standard unit testing

It is possible to apply standard unit test practices to the defined logic files by using the Mocha test framework and Sinon.js to stub runtime functions. However, unit testing in this fashion cannot test query files or test the correct application of ACL rules. This testing requires a connection to a blockchain target run time using functional verification tests.

Note that when testing transaction processor functions, it is not possible to export functions for direct testing. An elegant way to gain access to non-exported functions for testing is the use of the npm module `rewire`, which adds special `setter` and `getter` functions to modules and consequently enables unit testing.

Example 2-1 shows using the npm module `rewire` from the unit testing of the internal function `netTransfers`.

Example 2-1 Using the rewire npm module

```
const rewire = require('rewire');
const clearingRewire = rewire('../lib/clearing');
const testFunction = clearingRewire.__get__('netTransfers');
```

Example 2-1 uses `rewire` instead of `require` to import the module for testing, then uses the getter `__get__` to bind the non-exported function to a variable. This variable can then be treated as the function in subsequent testing, such as this example:

```
testFunction([transferRequest], partyId1, []).should.be.equal(100);
```

2.4.2 Functional verification tests

Hyperledger Composer can connect to any blockchain target run time by using connectors. One such connector is the "embedded-connector" that connects to an embedded memory-based run time designed to be targeted by unit tests.

Two testing frameworks can be used to test a business network: Mocha and Cucumber. Mocha is a standard testing framework for JavaScript, and Cucumber is a scenario-based testing framework intended to test higher-level system behavior through "given-when-then" scenarios defined in feature files. Hyperledger Composer provides a "composer-cucumber-steps" npm module that eases the generation of such behavior-driven tests.

Note that the creation of Formal Verification (FV) tests does not require the transactions to be coded. All specification is completed through knowledge of the model file only, which describes the available assets and transactions.

2.4.3 Mocha tests

You can set up an FV suite using the Composer APIs within a Mocha framework, which enables the driving of Participant actions using a specified identity. This suite enables isolated testing of these objects:

- Access to Assets
- Access to other Participants
- Ability to submit Transactions
- Correct action of Transactions

Examples of such testing are readily available in the Hyperledger Composer sample networks repository. As a sample, the testing of the `MarkPreProcessComplete` transaction is shown in Example 2-2.

Example 2-2 Testing the MarkPreProcessComplete transaction

```
describe('MarkPreProcessComplete Transaction', () => {
        it('should mark all TransferRequests from issuing party as 
PRE_PROCESS_COMPLETE', async () => {
            // Get the factory for the business network.
            await businessNetworkConnection.connect(bank0);
            factory = businessNetworkConnection.getBusinessNetwork().getFactory();
            const transaction = factory.newTransaction(namespace, 
'MarkPreProcessComplete');
            transaction.batchId = '0';

            // Submit the transaction
            await businessNetworkConnection.submitTransaction(transaction);

            // Retrieve and check
            const batchAssetRegistry = await 
businessNetworkConnection.getAssetRegistry(namespace + '.BatchTransferRequest');
            const transferAssetRegistry = await 
businessNetworkConnection.getAssetRegistry(namespace + '.TransferRequest');
            const batchAsset = await batchAssetRegistry.get(transaction.batchId);

            const transferRequests = batchAsset.transferRequests;

            for (let i=0; i<transferRequests.length; i++) {
                let transferRequest = transferRequests[i];
                let tr = await 
transferAssetRegistry.get(transferRequest.getIdentifier());
                if(tr.fromBank.getIdentifier() === bank0){
                    tr.fromBankState.should.equal('PRE_PROCESS_COMPLETE');
                }
            }
        });
});
```

The test in Example 2-2 shows the use of a specified `BankParticipant` issuing the transaction that contains the defined transaction body, then a check being made on the expected (new) state of the `TransactionRequest` within the asset registry.

2.4.4 Cucumber tests

Cucumber testing enables Behavior Driven Development (BDD) by enabling the specification of functional behavior in a natural language that customers can understand. This configuration enables testing of feature documentation written in a business-facing text.

Use of Cucumber testing provides the same capability of FV testing as a custom built Mocha framework, but has the added benefit of abstracting the tests being run into a living feature document. Therefore, the use of Cucumber testing is advantageous for merging business requirements with test confidence. By taking advantage of the provided `composer-Cucumber-steps` module, you can quickly create tests that detail the intended feature.

Example 2-3 shows a sample Cucumber test that describes the action of a `CreateBatch` transaction.

Example 2-3 Sample Cucumber test

```
Scenario: When I submit a CreateBatch transaction, a BatchTransferRequest is
created for each unique paring of banks and an event is emitted that identifies
each newly created BatchTransferRequest.
        When I use the identity bank1
        And I submit the following transaction
"""
            [
            {"$class":"org.clearing.CreateBatch",
            "batchId":"batch1", "usdRates": [
            {"$class":"org.clearing.UsdExchangeRate","to":"EURO","rate":0.75},

{"$class":"org.clearing.UsdExchangeRate","to":"STERLING","rate":1.75}]}
            ]
            """
        Then I should have received the following events of type
org.clearing.BatchCreatedEvent
            | batchId   |
            | batch1:bank11 |
            | batch1:bank12 |
        And I should have the following assets
            """
            [

{"$class":"org.clearing.BatchTransferRequest","batchId":"batch1:bank11",

"settlement":{"$class":"org.clearing.Settlement","amount":1749.9999999999998,"curr
ency":"EURO","creditorBank":"org.clearing.BankingParticipant#bank2","debtorBank":"
org.clearing.BankingParticipant#bank1"},
            "state":"PENDING_PRE_PROCESS",

"parties":["org.clearing.BankingParticipant#bank1","org.clearing.BankingParticipan
t#bank2"],

"transferRequests":["org.clearing.TransferRequest#reqid1","org.clearing.TransferRe
quest#reqid3"]},

{"$class":"org.clearing.BatchTransferRequest","batchId":"batch1:bank12",

"settlement":{"$class":"org.clearing.Settlement","amount":1750,"currency":"STERLIN
G","creditorBank":"org.clearing.BankingParticipant#bank3","debtorBank":"org.cleari
ng.BankingParticipant#bank1"},
            "state":"PENDING_PRE_PROCESS",

"parties":["org.clearing.BankingParticipant#bank1","org.clearing.BankingParticipan
t#bank3"],
            "transferRequests":["org.clearing.TransferRequest#reqid2"]}
            ]
            """
```

The test shown in Example 2-3 on page 12 describes submitting a `CreateBatch` transaction and the expected output. Similar tests can be written to fully encapsulate the intended functionality of the business network.

2.5 Implementing the business network logic

A transaction processor function is the logical operation of a transaction that is defined in a model file. Hyperledger Composer uses Node 8 syntax to define the business network logic, which means it is possible to use `async/await` within your functions.

Within a transaction processor function, you can use both Hyperledger Composer and Hyperledger Fabric APIs. If you are using Hyperledger Fabric APIs, then all Hyperledger Composer access control rules are bypassed.

Within a transaction processor function, all runtime APIs are within scope and can be called directly. For example, to obtain and work with the participant registry that contains all `TransferRequest`(s), issue this command:

```
const transferAssetRegistry = await getAssetRegistry(namespace +
'.TransferRequest');
```

At this point, the registry is available to work with the asset registry APIs.

When working within a transaction processor function to create new resources, use a *factory* to enable direct programmatic creation of resources without the performance degradation of serialization. A factory can be obtained by using the runtime API:

```
const factory = getFactory();
```

At this point, the factory can be used to generate resources that match those defined within the model file.

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

The first line of comments above a transaction processor function contains a human readable description of what the transaction processor function does (Example 2-4). The second line must include the `@param` tag to indicate the parameter definition. The `@param` tag is followed by the resource name of the transaction that triggers the transaction processor function. This resource name takes the format of the namespace of the business network, followed by the transaction name. After the resource name is the parameter name that references the resource. This parameter must be supplied to the JavaScript function as an argument. The third line must contain the `@transaction` tag, which identifies the code as a transaction processor function and is required.

Example 2-4 Transaction processor function

```
**
 * Submit a TransferRequest
 * @param {org.clearing.SubmitTransferRequest} tx passed transaction body
 * @transaction
 */
async function submitTransferRequest(tx) {
```

Example 2-4 on page 13 indicates that the function is a transaction (`@transaction`) that runs when the transaction `org.clearing.SubmitTransferRequest` is issued. It takes the transaction body `tx` as defined in the model file.

We will continue the development of the `org.clearing.SubmitTransferRequest`, which is used by participants to create a `TransferRequest` in the transfer request asset registry. The model file defines the transaction, as shown in Example 2-5.

Example 2-5 Defining the transaction

```
transaction SubmitTransferRequest {
 o String transferId
 o String toBank
 o TransferRequestState state
 o Transfer details
}
```

Example 2-5 shows that we intend to submit the transaction with an argument containing these items:

- A transfer identifier
- The identifier of the `BankingParticipant` to which the transfer is to be made
- The `TransferRequestState`, which is an enumeration of `PENDING | PROCESSING | PRE_PROCESS_COMPLETE | COMPLETE | ERROR`
- The transfer details that define the to/from account, currency, and amount

The passed values are the minimal set required to generate a `TransferRequest`, as defined in the model file, as shown in Example 2-6.

Example 2-6 Passed values to generate TransferRequest

```
asset TransferRequest identified by requestId {
  o String requestId
  o Transfer details
  o TransferRequestState fromBankState default = 'PRE_PROCESS_COMPLETE'
  o TransferRequestState toBankState
  o TransferRequestState state
  --> BankingParticipant fromBank
  --> BankingParticipant toBank
}
```

Because we want to create a resource (the `TransferRequest`), and will be referencing participants, we must obtain the participant registry, asset registry, and a factory, as shown in Example 2-7.

Example 2-7 Obtaining participant registry, asset registry and a factory

```
// Required registries for this transaction
    const participantRegistry = await getParticipantRegistry(namespace +
'.BankingParticipant'); // eslint-disable-line no-undef
    const transferAssetRegistry = await getAssetRegistry(namespace +
'.TransferRequest'); // eslint-disable-line no-undef

    // Use a factory for creation of asset
    const factory = getFactory(); // eslint-disable-line no-undef
    const transferRequest = factory.newResource(namespace, 'TransferRequest',
tx.transferId);
```

We then populate the elements of the `transferRequest` using the passed transaction body and wanted initialization states of PENDING, as shown in Example 2-8.

Example 2-8 Populating transferRequest

```
// tx aspects
    transferRequest.details = tx.details;
    transferRequest.state = 'PENDING';

    // Participant aspects
    const fromBankRef = factory.newRelationship(namespace, 'BankingParticipant',
getCurrentParticipant().getIdentifier()); // eslint-disable-line no-undef
    transferRequest.fromBank = fromBankRef;
    transferRequest.fromBankState = tx.state;

    const toBank = await participantRegistry.get(tx.toBank);
    const toBankRef = factory.newRelationship(namespace, 'BankingParticipant',
toBank.getIdentifier());
    transferRequest.toBank = toBankRef;
    transferRequest.toBankState = 'PENDING';
```

After the creation of the transfer request is complete, add it to the asset registry:

```
await transferAssetRegistry.add(transferRequest);
```

The complete function is shown in Example 2-9.

Example 2-9 Completed TransferRequest function

```
/**
 * Submit a TransferRequest
 * @param {org.clearing.SubmitTransferRequest} tx passed transaction body
 * @transaction
 */
async function submitTransferRequest(tx) { // eslint-disable-line no-unused-vars
    // Required registries for this transaction
    const participantRegistry = await getParticipantRegistry(namespace +
'.BankingParticipant'); // eslint-disable-line no-undef
    const transferAssetRegistry = await getAssetRegistry(namespace +
'.TransferRequest'); // eslint-disable-line no-undef

    // Use a factory for creation of asset
    const factory = getFactory(); // eslint-disable-line no-undef
    const transferRequest = factory.newResource(namespace, 'TransferRequest',
tx.transferId);

    // tx aspects
    transferRequest.details = tx.details;
    transferRequest.state = 'PENDING';

    // Participant aspects
    const fromBankRef = factory.newRelationship(namespace, 'BankingParticipant',
getCurrentParticipant().getIdentifier()); // eslint-disable-line no-undef
    transferRequest.fromBank = fromBankRef;
    transferRequest.fromBankState = tx.state;

    const toBank = await participantRegistry.get(tx.toBank);
    const toBankRef = factory.newRelationship(namespace, 'BankingParticipant',
toBank.getIdentifier());
    transferRequest.toBank = toBankRef;
    transferRequest.toBankState = 'PENDING';

    // Add to asset registry
    await transferAssetRegistry.add(transferRequest);
}
```

When run, this function results in the creation of a `TransactionRequest` in the asset registry, with the `fromBank` listed as the participant that submitted the transaction, the `toBank` as the participant identified in the transaction argument body, and all states in PENDING except for the `fromBank` state (which is taken from the passed argument).

At this point, any functional verification tests that are written to demonstrate the correct operation of the transaction should pass.

2.6 Adding ACL rules

Hyperledger Composer includes an ACL that provides declarative access control over elements of the domain model. By defining ACL rules, you can determine which users/roles are permitted to create, read, update, and delete elements in a business networks domain model.

An ACL rule can be unconditional or conditional, and is written in a JSON format. A sample is shown in Example 2-10.

Example 2-10 Sample ACL rule

```
rule EverybodyCanCreateBatches {
    description: "Allow all participants to submit CreateBatch transactions"
    participant: "org.clearing.BankingParticipant"
    operation: CREATE
    resource: "org.clearing.CreateBatch"
    action: ALLOW
}
```

In Example 2-10, the `participant` defines the person or entity that the rule is applicable to, and must relate to a defined participant in the model file. The `operation` declares the operation that is being considered, and can be CREATE, READ, UPDATE, DELETE, or ALL. The `resource` defines what the ACL rule applies to, such as a class or all classes under a namespace that exist in the model file. The `action` identifies the action of the rule and must be ALLOW or DENY.

Therefore, the rule that is defined in Example 2-10 enables all `BankingParticipants` to perform the CREATE action on the transaction `CreateBatch`.

A conditional rule applies additional logic within the processing of an ACL rule. This process is performed by declaring references on the `participant` and `resource`, which are used within a `condition`. The `condition` must evaluate to a Boolean, and can only be generated by code that is containable within an 'if' condition block. To simplify the presentation of conditional ACL rules, you can use a `helper` class that contains all the required logic to evaluate the Boolean response (Example 2-11). However, it is not possible to perform asynchronous operations within such functions.

Example 2-11 Example of a helper class

```
rule ParticipantsRestrictionOnBatchTransferRequest {
    description: "Allow participants involved with an Asset to interact with it"
    participant(p): "org.clearing.BankingParticipant"
    operation: ALL
    resource(r): "org.clearing.BatchTransferRequest"
    condition: (partyWithinBatchTransferRequest(r, p))
    action: ALLOW
}
```

In Example 2-11, the condition relates to the evaluation of the function `partyWithinBatchTransaferRequest(r, p)`. This is a function declared within a `permissionsHelper`, and used to inspect the `resource(r)` being interacted with by the `participant(p)`.

Note: Avoid adding ACL rules until the required business logic has been implemented and tested because access restrictions can hide errors in logic. When ready, add access control tests to lock down the business network.

2.7 Debugging

Debugging a network operation is achieved through the inspection of logs and by the attachment of a debugger.

2.7.1 Logging inspection

To enable debug log messages, create a folder named `config` in the top level of your network definition directory and create a file within named `default.json`. Enter the contents shown in Example 2-12 into the JSON file. When the Composer application is started, such as by using **`npm test`**, the environment variables are read in and the debug settings are applied to the logger. The `config` file results in all debug information being written to a log file in the local directory and also printed to console. To disable debugging, change `"composer[debug]:*"` to `"composer[none]:*"`, and `"maxLevel": "error"` to `"maxLevel": "none"`.

Example 2-12 Sample default.json file for debugging

```
{
    "composer": {
        "log": {
          "debug": "composer[debug]:*",
          "console": {
            "maxLevel": "error"
          },
          "file": {
            "filename" : "./log.txt"
          }
        }
    }
}
```

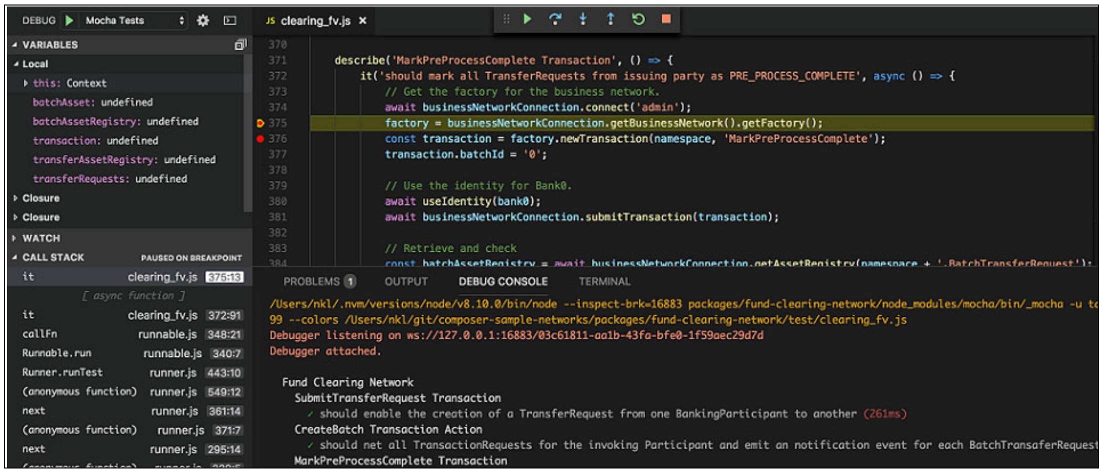
2.7.2 Debugging against the embedded connector

VSCode has built-in debugging support for the `Node.js` runtime. It can debug JavaScript, TypeScript, and any other language that gets transpiled into JavaScript. In the case of the fund clearing sample network, the `launch.json` shown in Example 2-13 was used.

Example 2-13 Sample launch.json

```
{
    "version": "0.2.0",
    "configurations": [
        {
            "type": "node",
            "request": "launch",
            "name": "Mocha Tests",
            "program": 
"${workspaceFolder}/packages/fund-clearing-network/node_modules/mocha/bin/_mocha",
            "args": [
                "-u",
                "tdd",
                "--timeout",
                "999999",
                "--colors",
                
"${workspaceFolder}/packages/fund-clearing-network/test/clearing_fv.js"
            ],
            "internalConsoleOptions": "openOnSessionStart"
        }
    ]
}
```

This `launch.json` gives instant access to the ability to debug the business network through use of the defined test suites, as shown in Figure 2-2.



```
this: Context
batchAsset: undefined
batchAssetRegistry: undefined
transaction: undefined
transferAssetRegistry: undefined
transferRequests: undefined
```

```
it                     clearing_fv.js  375:13
           [ async function ]
it                     clearing_fv.js  372:91
callFn                 runnable.js     348:21
Runnable.run           runnable.js     340:7
Runner.runTest         runner.js       443:10
(anonymous function)   runner.js       549:12
next                   runner.js       361:14
(anonymous function)   runner.js       371:7
next                   runner.js       295:14
```

```

    describe('MarkPreProcessComplete Transaction', () => {
        it('should mark all TransferRequests from issuing party as PRE_PROCESS_COMPLETE', async () => {
            // Get the factory for the business network.
            await businessNetworkConnection.connect('admin');
            factory = businessNetworkConnection.getBusinessNetwork().getFactory();
            const transaction = factory.newTransaction(namespace, 'MarkPreProcessComplete');
            transaction.batchId = '0';

            // Use the identity for Bank0.
            await useIdentity(bank0);
            await businessNetworkConnection.submitTransaction(transaction);

            // Retrieve and check
            const batchAssetRegistry = await businessNetworkConnection.getAssetRegistry(namespace + '.BatchTransferRequest');
```

```
/Users/nkl/.nvm/versions/node/v8.10.0/bin/node --inspect-brk=16883 packages/fund-clearing-network/node_modules/mocha/bin/_mocha -u td
99 --colors /Users/nkl/git/composer-sample-networks/packages/fund-clearing-network/test/clearing_fv.js
Debugger listening on ws://127.0.0.1:16883/03c61811-aa1b-43fa-bfe0-1f59aec29d7d
Debugger attached.

  Fund Clearing Network
    SubmitTransferRequest Transaction
      ✓ should enable the creation of a TransferRequest from one BankingParticipant to another (261ms)
    CreateBatch Transaction Action
      ✓ should net all TransactionRequests for the invoking Participant and emit an notification event for each BatchTransaferRequest
    MarkPreProcessComplete Transaction
```

Figure 2-2 Debugging by using test suite

2.8 Next steps

The fund clearing business network source code is available for download or cloning as part of the Hyperledger Composer sample networks. At the time of writing, the source is contained in a fork and branch of the main repo. The intention is to merge it at a later date.

To actively participate in the next sections' processes and examples, download a copy of the source code from GitHub before proceeding on to the next section.

Use the following commands to retrieve a copy:

```
git clone https://github.com/hyperledger/composer-sample-networks.git
cd composer-sample-networks/packages/fund-clearing-network
```

3

Basic flow using IBM Composer Playground

After you develop your business network to some degree of satisfaction, you might want to interact with a deployed network in real time, manually creating Participants, Assets, and Transactions. These are the main components of any Hyperledger Composer business network. Any Composer business network designer should understand that Assets are modified by Participants through the submission of Transactions. The Hyperledger Composer Playground is a browser-based user interface that enables the configuration, deployment, and testing of a business network. For more information, see Key Concepts in Hyperledger Composer. IBM Composer Playground enables this capability and more.

The Hyperledger Composer Playground provides a user interface for the configuration, deployment, and testing of a business network. Advanced Playground features allow you to manage the security of the business network, invite participants to business networks, and connect to multiple blockchain business networks. You can get a quick experience of Playground through the online sandbox.

After you experience Playground in the sandbox, this section provides information about how to set up a Fabric network on your workstation, deploy the fund clearing business network, and connect a local instance of Playground.

This chapter includes the following sections:

- Prerequisites
- Adding participants
- Adding assets
- Submitting the CreateBatch transaction
- Submitting the MarkPreProcessComplete transaction
- Submitting the CompleteSettlement transaction
- Submitting the MarkPostProcessComplete transaction

3.1 Prerequisites

You can either use the fund-clearing-network as provided in the online Playground or start Playground locally on your workstation. For the local deployment, you must set up the environment for Composer and start a Fabric local network, create business admin cards, and import those cards into your Compser wallet. To complete these steps, follow the instructions provided in the online Hyperledger tutorial, making modifications as necessary for the fund clearing application.

If you decide to use the online Playground, open Hyperledger Composer in your browser, click **Let's Blockchain**, then click **Deploy a new business network**, as shown in Figure 3-1.

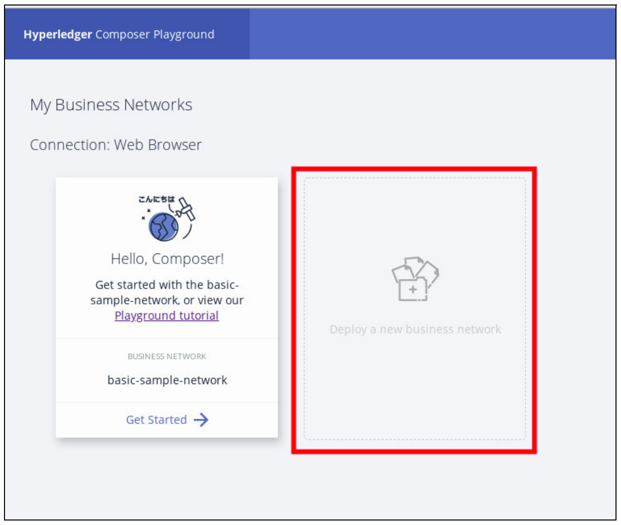

Hyperledger Composer Playground
My Business Networks
Connection: Web Browser
Hello, Composer!
Get started with the basic-sample-network, or view our Playground tutorial
BUSINESS NETWORK
basic-sample-network
Get Started
Deploy a new business network

Figure 3-1 Clicking Deploy a new business network

Click **fund-clearing-network** and complete the on-line instructions to start the network, as shown in Figure 3-2.

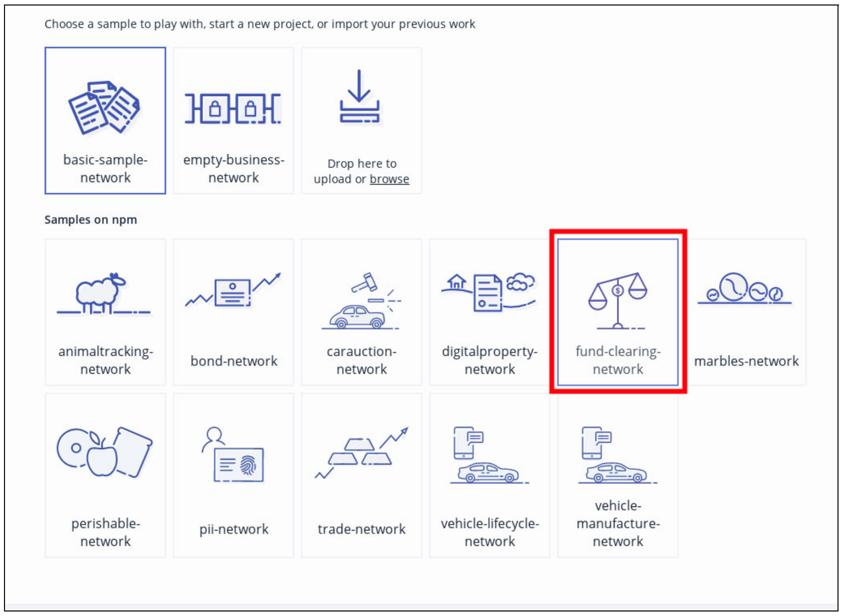


Figure 3-2 Clicking the fund clearing network icon

A Business Network Card provides all of the information that is needed to connect to a blockchain business network. You can only access a blockchain business network through a valid Business Network Card. A Business Network Card contains an Identity for a single Participant within a deployed business network. Business Network Cards are used in the Hyperledger Composer Playground to connect to deployed Business Networks.

After the network is running and the fund clearing Business Network Archive (BNA) is deployed with these instructions, you can use Playground to submit the transactions using a browser interface.

Start Playground by issuing the **`composer-playground`** command from a terminal, which automatically opens a browser and connects to `localhost:8080`.

Figure 3-3 shows the lifecycle of and interactions with a `BatchTransferRequest`.

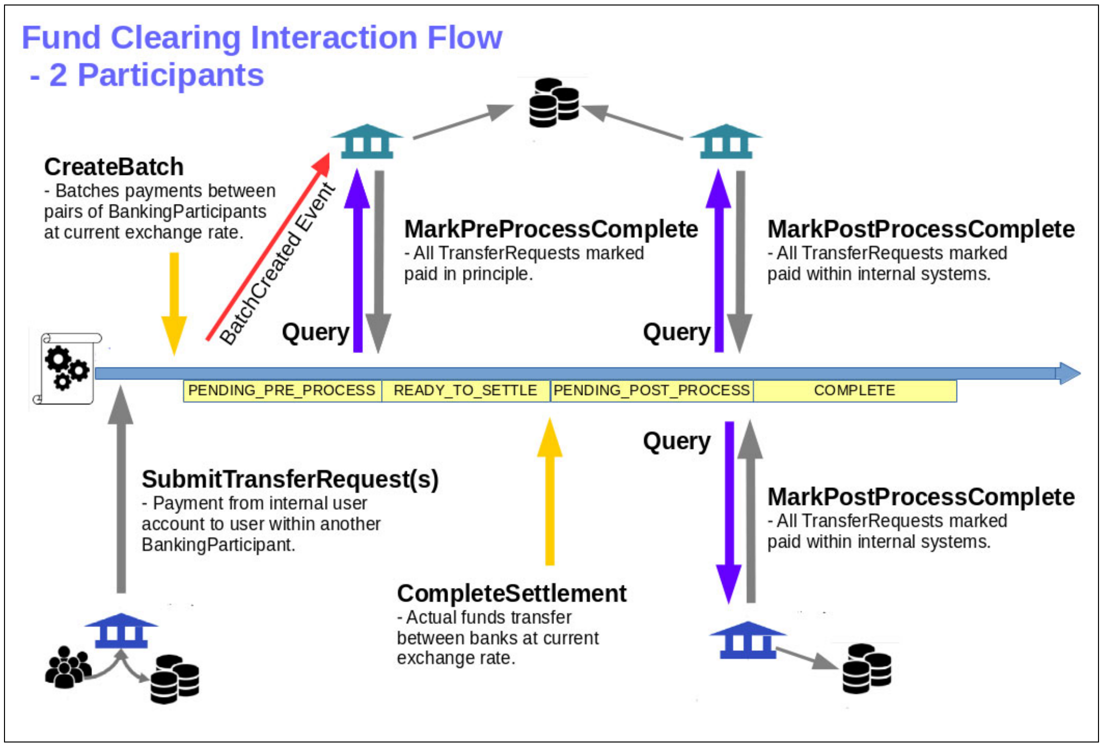

Fund Clearing Interaction Flow
- 2 Participants
CreateBatch
- Batches payments between pairs of BankingParticipants at current exchange rate.
BatchCreated Event
Query
MarkPreProcessComplete
- All TransferRequests marked paid in principle.
Query
MarkPostProcessComplete
- All TransferRequests marked paid within internal systems.
PENDING_PRE_PROCESS
READY_TO_SETTLE
PENDING_POST_PROCESS
COMPLETE
Query
SubmitTransferRequest(s)
- Payment from internal user account to user within another BankingParticipant.
MarkPostProcessComplete
- All TransferRequests marked paid within internal systems.
CompleteSettlement
- Actual funds transfer between banks at current exchange rate.

Figure 3-3 Funding Clearing Interaction Flow diagram

The Fund Clearing Network allows network participants to process inter-participant payments, thus replacing a traditional clearing house. This chapter assumes that participants within the network are banking entities that accumulate and submit one or more transfer requests to other participants. These transfer requests are placed into the business network as assets and periodically accumulated into a series of batches.

All transfer requests between each pair of banking participants are accumulated and represented in a single new asset. Each batch of transfer requests details the required net settlement between each banking participant for the contained array of transfer request assets, based on the currency of the creditor bank.

3.2 Adding participants

A Participant is an actor in a business network, which can, for example, be an individual or an organization. A participant can create assets, and also exchange assets with other participants. A participant works with these assets by submitting transactions.

A participant has a set of identity documents that can be validated to prove the identity of that participant. In Hyperledger Composer, participants are separated from the set of identity documents that they can use to interact with a business network.

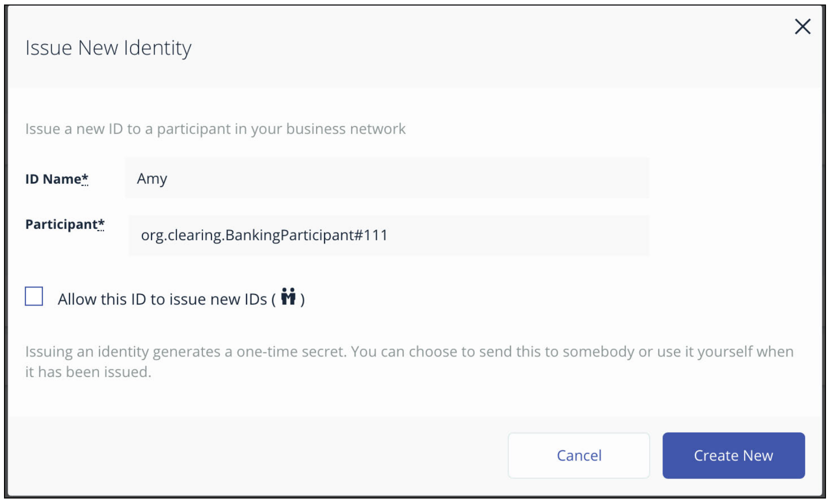
We use the Fabric Certificate Authority (CA) to issue credentials for and authenticate identities, which are then bound to Composer participants. In practice, a single participant can have multiple identities bound to it. In our fund clearing implementation, the banking entity is represented by a participant and employees of the bank are represented by identities stored in the Fabric CA.

To add participants to Playground, complete these steps:

1. Create one participant (`BankingParticipant`) using an identity from one organization (for example, `Org1`).

```
{
  "$class": "org.clearing.BankingParticipant",
  "bankingId": "111",
  "bankingName": "Bank A",
  "workingCurrency": "EURO",
  "fundBalance": 100000
}
```

2. Issue a new identity named `Amy` and bind it with the participant you added in the previous step with a participant ID of `111`, as shown in Figure 3-4.


Issue New Identity

Issue a new ID to a participant in your business network

ID Name*: Amy

Participant*: org.clearing.BankingParticipant#111

☐ Allow this ID to issue new IDs ( )

Issuing an identity generates a one-time secret. You can choose to send this to somebody or use it yourself when it has been issued.

Figure 3-4 Issue New Identity window

3. Create another participant (`BankingParticipant`) using an identity from the other organization (for example, `org2`):

```
{
  "$class": "org.clearing.BankingParticipant",
  "bankingId": "222",
  "bankingName": "Bank B",
  "workingCurrency": "USD",
  "fundBalance": 100000
}
```

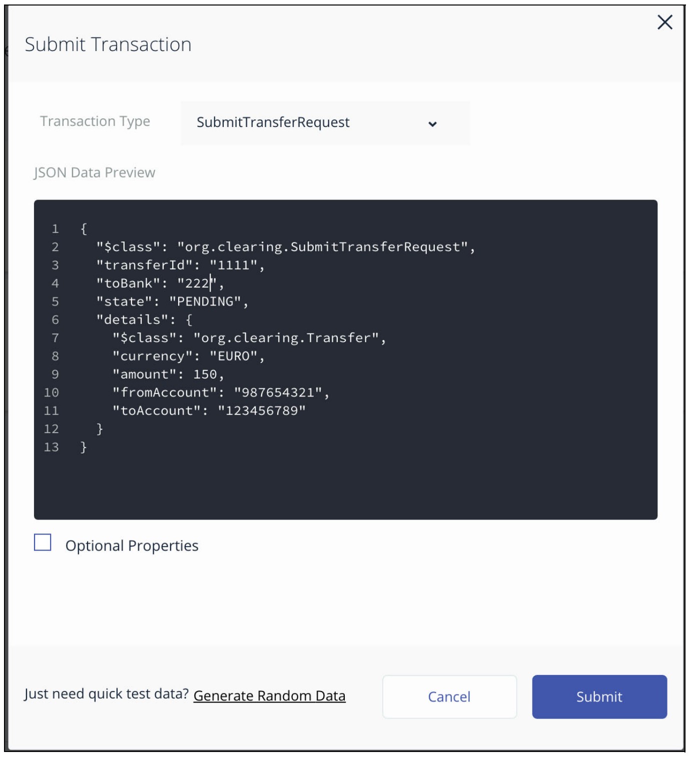
4. Similar to step 2 (Figure 3-4 on page 24), issue a new identity named `Barry` and bind it with the participant you created in step 3 with the participant ID `222`.

3.3 Adding assets

An asset is anything of value (physical or non-physical). A house or car is classified as a physical asset, and a mortgage or loan is classified as a non-physical asset. Assets within Hyperledger Composer can be defined to include either type of asset.

To add assets to Playground, complete these steps:

1. Create a `TransferRequest` transaction, as shown in Figure 3-5, that references the `BankingParticipant` participant in the `TransferRequest` using the identity `111` that you created earlier.



```
{
  "$class": "org.clearing.SubmitTransferRequest",
  "transferId": "1111",
  "toBank": "222",
  "state": "PENDING",
  "details": {
    "$class": "org.clearing.Transfer",
    "currency": "EURO",
    "amount": 150,
    "fromAccount": "987654321",
    "toAccount": "123456789"
  }
}
```

Figure 3-5 Creating TransferRequest transaction

Example 3-1 shows the creation process.

Example 3-1 Creating a TransferRequest transaction

```
{
  "$class": "org.clearing.SubmitTransferRequest",
  "transferId": "1111",
  "state": "PENDING",
  "toBank": "222"
  "details": {
    "$class": "org.clearing.Transfer",
    "currency": "EURO",
    "amount": 150,
    "fromAccount": "987654321",
    "toAccount": "123456789"
  }
}
```

2. Create a second `SubmitTransferRequest` transaction that references the `BankingParticipant` participants in the `TransferRequest` using the identity `222` that you created earlier (Example 3-2).

Example 3-2 Creating second SubmitTransferRequest transaction

```
{
  "$class": "org.clearing.SubmitTransferRequest",
  "transferId": "1111",
  "state": "PENDING",
  "toBank": "111"
  "details": {
    "$class": "org.clearing.Transfer",
    "currency": "USD",
    "ammount": 150,
    "fromAccount": "123456789",
    "toAccount": "987654321"
  }
}
```

3.4 Submitting the CreateBatch transaction

The `CreateBatch` transaction creates a new `BatchTransferRequest` in the Asset Registry for each unique pairing of `BankingParticipant` participants that have pending `TransferRequest` assets. An event is emitted for each `BatchTransferRequest` that is created.

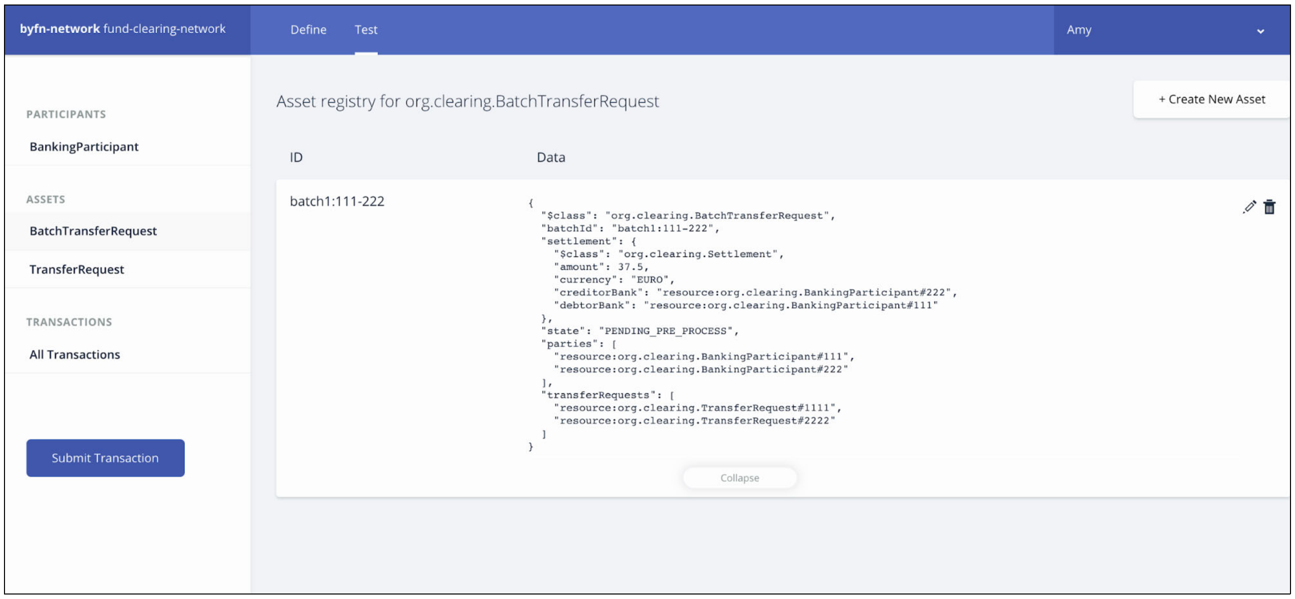
Any participant can submit `CreateBatch` transactions. Using either participant, submit a `CreateBatch` transaction, as shown in Example 3-3.

Example 3-3 Submitting a CreateBatch transaction

```
{
     "$class": "org.clearing.CreateBatch",

     "usdRates": [{
      "$class": "org.clearing.UsdExchangeRate",
      "to": "EURO",
      "rate": 0.75
     }],
     "batchId": "batch1"
     }
```

Running this transaction results in the creation of a `BatchTransferRequest` that contains references to the participating org's `TransferRequests`, as shown in Figure 3-6.



Asset registry for org.clearing.BatchTransferRequest

ID: batch1:111-222

Data:

```
{
  "$class": "org.clearing.BatchTransferRequest",
  "batchId": "batch1:111-222",
  "settlement": {
    "$class": "org.clearing.Settlement",
    "amount": 37.5,
    "currency": "EURO",
    "creditorBank": "resource:org.clearing.BankingParticipant#222",
    "debtorBank": "resource:org.clearing.BankingParticipant#111"
  },
  "state": "PENDING_PRE_PROCESS",
  "parties": [
    "resource:org.clearing.BankingParticipant#111",
    "resource:org.clearing.BankingParticipant#222"
  ],
  "transferRequests": [
    "resource:org.clearing.TransferRequest#1111",
    "resource:org.clearing.TransferRequest#2222"
  ]
}
```

Figure 3-6 BatchTransferRequest

3.5 Submitting the MarkPreProcessComplete transaction

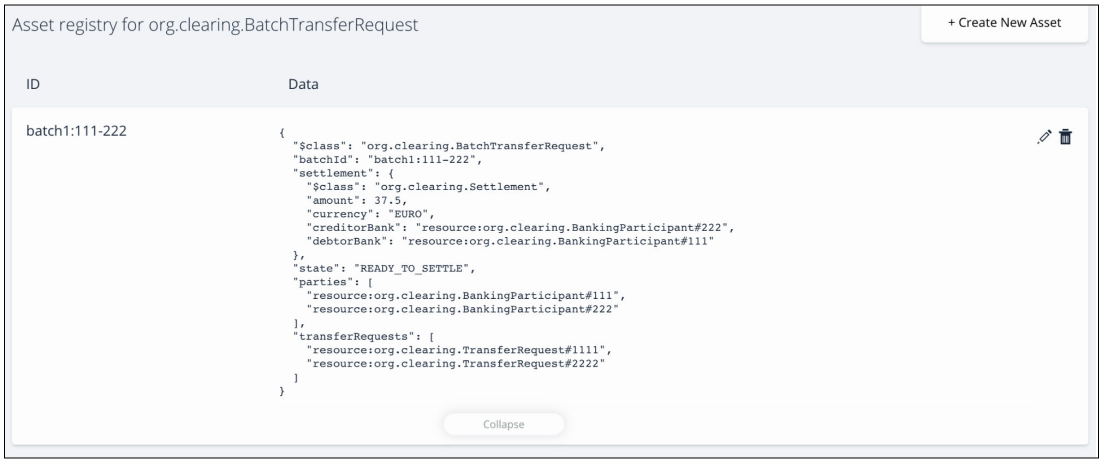
The `MarkPreProcessComplete` transaction submitted by a user with an identity bound to a `BankingParticipant` participant updates all `TransferRequest` assets for the relevant `BankingParticipant` in a `BatchTransferRequest` to be in the PRE_PROCESS_COMPLETE state. If all referenced `TransferRequest` assets are detected to be in the PRE_PROCESS_COMPLETE state, the `BatchTransferRequest` is marked to be in the READY_TO_SETTLE state.

Access the business network under each identity that is bound to a `BankingParticipant` in each organization. Submit `MarkPreProcessComplete` transactions by passing details of the `BatchTransferRequest` identifier from the previous step (as described in 3.4, "Submitting the CreateBatch transaction" on page 27).

Submit one transaction with the identity `Amy` from `org1` and submit another using the identity `Barry` from `org2`, as shown in the following example:

```
{
      "$class": "org.clearing.MarkPreProcessComplete",
       "batchId": "batch1:111-222"
   }
```

Observe how the contents of the `BatchTransactionRequest` and `TransactionRequest` assets change as you run these transactions, as shown in Figure 3-7.


Asset registry for org.clearing.BatchTransferRequest

ID	Data
batch1:111-222	see below

```
{
  "$class": "org.clearing.BatchTransferRequest",
  "batchId": "batch1:111-222",
  "settlement": {
    "$class": "org.clearing.Settlement",
    "amount": 37.5,
    "currency": "EURO",
    "creditorBank": "resource:org.clearing.BankingParticipant#222",
    "debtorBank": "resource:org.clearing.BankingParticipant#111"
  },
  "state": "READY_TO_SETTLE",
  "parties": [
    "resource:org.clearing.BankingParticipant#111",
    "resource:org.clearing.BankingParticipant#222"
  ],
  "transferRequests": [
    "resource:org.clearing.TransferRequest#1111",
    "resource:org.clearing.TransferRequest#2222"
  ]
}
```

Figure 3-7 Changes in contents of BatchTransactionRequest and TransactionRequest assets

3.6 Submitting the CompleteSettlement transaction

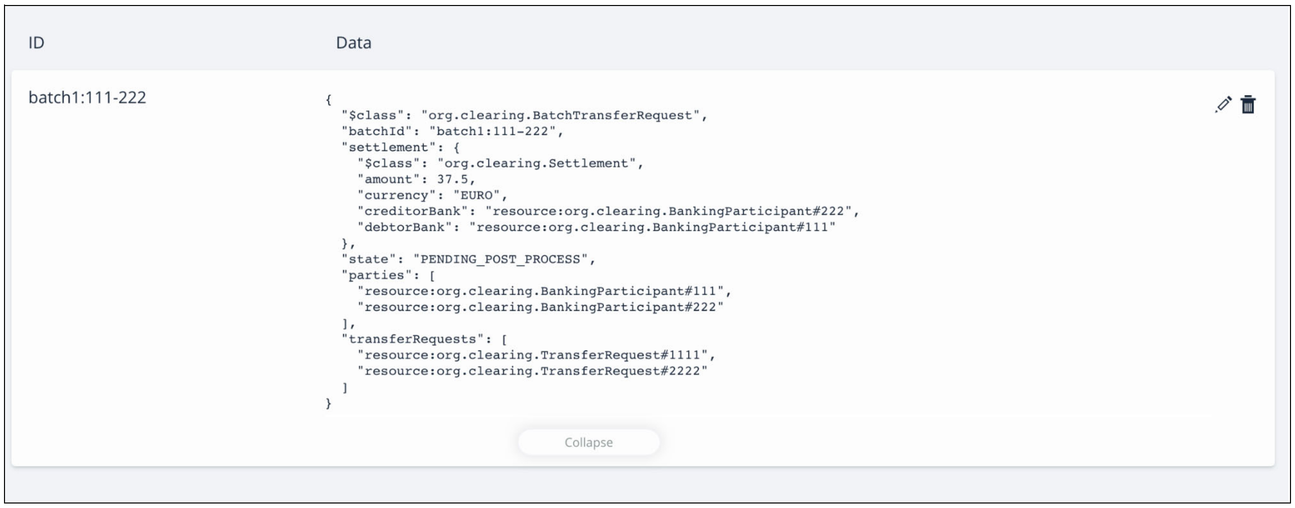
The `CompleteSettlement` transaction submitted by a user with an identity (such as `Amy` or `Barry`) that is bound to a `BankingParticipant` participant completes a fund transfer between the named `BankingParticipant` participants (such as `Bank A` and `Bank B`) from the passed `BatchTransferRequest` identifier, based on the most recent passed `UsdExchangeRates` array. After funds are transferred between `BankingParticipant` participants, the `BatchTransferRequest` is marked in the PENDING_POST_PROCESS state.

Access the business network with any identity that is bound to a `BankingParticipant`. Submit a `CompleteSettlement` transaction by passing details of the `BatchTransferRequest` identifier and an array of current exchange rates to be used for currency adjustment, as shown in Example 3-4.

Example 3-4 Sample CompleteSettlement transaction

```
{
   "$class": "org.clearing.CompleteSettlement",
   "batchId": "batch1:111-222",
      "usdRates": [{"$class":"org.clearing.UsdExchangeRate",
   "to":"EURO","rate":0.75},
   {"$class":"org.clearing.UsdExchangeRate",
   "to":"STERLING","rate":1.75}]
}
```

Notice that the `BatchTransferRequest` will be marked in the PENDING_POST_PROCESS state, as shown in Figure 3-8.



batch1:111-222

```
{
  "$class": "org.clearing.BatchTransferRequest",
  "batchId": "batch1:111-222",
  "settlement": {
    "$class": "org.clearing.Settlement",
    "amount": 37.5,
    "currency": "EURO",
    "creditorBank": "resource:org.clearing.BankingParticipant#222",
    "debtorBank": "resource:org.clearing.BankingParticipant#111"
  },
  "state": "PENDING_POST_PROCESS",
  "parties": [
    "resource:org.clearing.BankingParticipant#111",
    "resource:org.clearing.BankingParticipant#222"
  ],
  "transferRequests": [
    "resource:org.clearing.TransferRequest#1111",
    "resource:org.clearing.TransferRequest#2222"
  ]
}
```

Figure 3-8 Pending BatchTransferRequest

3.7 Submitting the MarkPostProcessComplete transaction

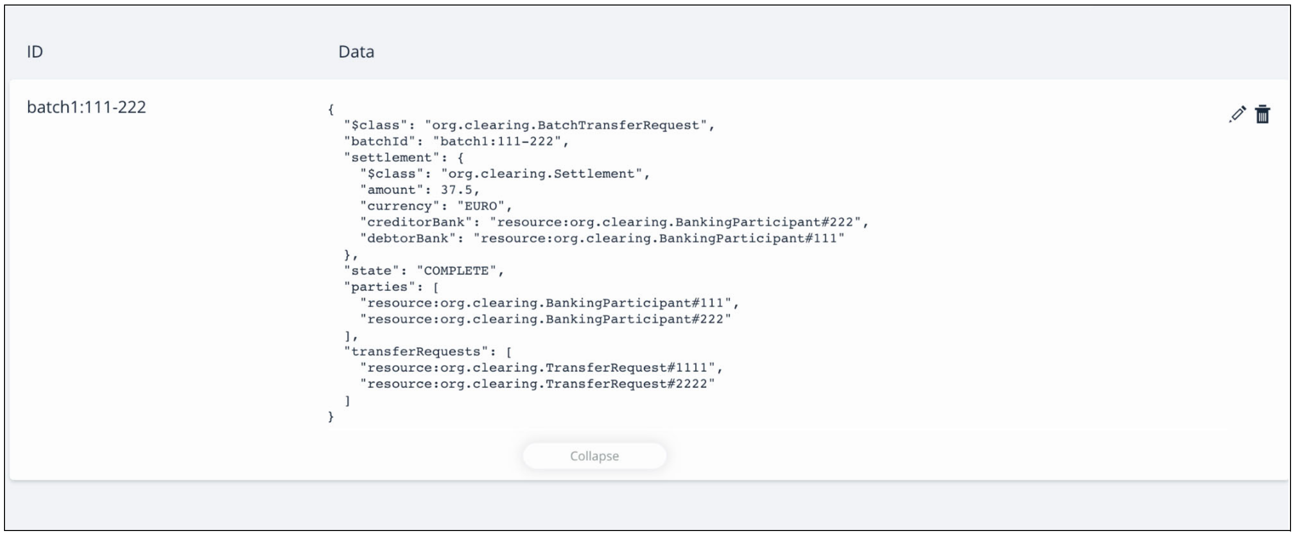
The `MarkPostProcessComplete` transaction submitted by a user (such as `Amy` or `Barry`) with an identity bound to a `BankingParticipant` participant updates all `TransferRequest` assets where the participant is named as a creditor within a `BatchTransferRequest` to be in the COMPLETE state. If all referenced `TransferRequest` assets are detected in the COMPLETE state, the `BatchTransferRequest` is marked in the COMPLETE state.

Submit a `MarkPostProcessComplete` transaction with each user identity (`Amy` or `Barry`) in each organization by passing details of the `BatchTransferRequest` identifier.

Submit a transaction with the identity `Amy` from `org1` and another using other the identity `Barry` from `org2`. See the following example:

```
{
      "$class": "org.clearing.MarkPostProcessComplete",
       "batchId": "batch1:111-222"
   }
```

Notice that `BatchTransferRequest` is now marked in the COMPLETE state, as shown in Figure 3-9.



ID | Data

batch1:111-222

```
{
  "$class": "org.clearing.BatchTransferRequest",
  "batchId": "batch1:111-222",
  "settlement": {
    "$class": "org.clearing.Settlement",
    "amount": 37.5,
    "currency": "EURO",
    "creditorBank": "resource:org.clearing.BankingParticipant#222",
    "debtorBank": "resource:org.clearing.BankingParticipant#111"
  },
  "state": "COMPLETE",
  "parties": [
    "resource:org.clearing.BankingParticipant#111",
    "resource:org.clearing.BankingParticipant#222"
  ],
  "transferRequests": [
    "resource:org.clearing.TransferRequest#1111",
    "resource:org.clearing.TransferRequest#2222"
  ]
}
```

Figure 3-9 Completed BatchTransferRequest

3.8 Summary

In this chapter, you deployed Hyperledger Fabric on your workstation, installed the fund clearing business network, created multiple participants, and used Hyperledger Composer Playground to exercise and explore the end to end processing of transfer requests between banking entities.

This chapter assumes that participants within the network are banking entities that accumulate and submit one or more `TransferRequest` items to other participants. These transfer requests are placed into the business network.

When a `CreateBatch` transaction is invoked, all transfer requests for each `BankingParticipant` are accumulated into a series of net `BatchTransferRequest`. Each `BatchTransferRequest` details the required net settlement between each `BankingParticipant` for the contained array of `TransferRequest` items, based on the currency of the creditor bank.

In the next chapter, we describe how to create a remote Hyperledger Fabric service on IBM Cloud and install the fund clearing business network there.

4 Migrating to a Starter Plan network

Now that you have explored how to develop and run the fund clearing business network locally, you will want to take the first step in putting your blockchain application into production: Deploying the business network to a remote Fabric network. IBP Starter Plan is not intended to be used for a production blockchain business network or for benchmarking your application.

The next step is IBM Blockchain Platform Enterprise Plan. The Enterprise Plan is a production-ready offering for organizations that would like to create or join a blockchain network for real businesses. This plan provides the key infrastructure along with tools and support to easily start a highly secure and production-ready network.

This chapter includes the following sections:

- Creating an IBM Cloud account
- Creating a Starter Plan network
- Acquiring network credentials and connection profiles for the member organizations
- Deploy the fund clearing business network to the IBP Starter Plan network
- Adding the certificates to the Starter Plan instance

4.1 Creating an IBM Cloud account

Before ordering a new IBM Blockchain Platform Starter Plan network, you must create an IBM Cloud customer account first. This account allows the user to have access to the various services offered by IBM Cloud like Kubernetes cluster, IoT, and Blockchain Platform.

To create an account, complete these steps:

1. Go to the IBM Cloud website and click **Create a free account**.
2. Complete the form, providing your email address, name, region, and phone number. Enter a password.
3. Click the link provided by the confirmation email, and follow the instructions to log in to IBM Cloud.
4. Before using IBM Cloud, you need to set up a basic development environment. Because IBP Starter Plan is currently only available in the US-south region while in beta, select this region, and enter a name for your organization. Note that you can change this name or create new organizations later.
5. Name your first development space. Again, this name can be changed at any time. After you have chosen a name, click the **Create** link.
6. Click the **I'm Ready** link to get started with IBM Cloud.

The IBP Starter Plan requires you to upgrade your account to the "Pay-As-You-Go" tier. You can read more on how to perform this upgrade at the Account types web page.

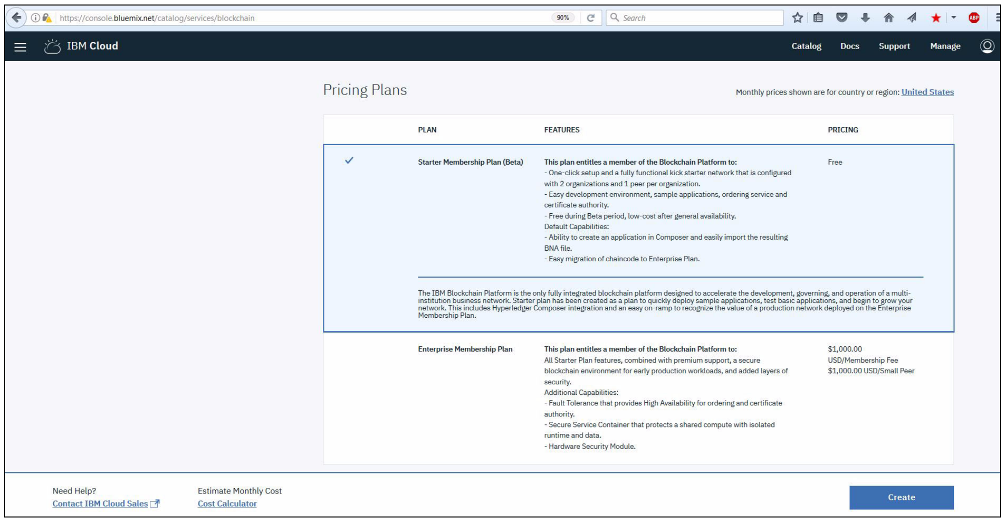
4.2 Creating a Starter Plan network

IBM Blockchain Starter Plan is an entry-level option that enables organizations to simulate multi-organization blockchain networks, quickly develop applications, and work with supplied examples. It also provides a user friendly UI experience. The IBP Starter Plan Fabric network has two organizations, each with its own peer and CA. With this network you can test chaincode, deploy samples, and invite other members to collaborate. We will use it to deploy and test the fund clearing business network.

To create a Starter Plan network, complete these steps:

1. Log in to IBM Cloud with the account you just created and open your dashboard.
2. Verify the region, org, and space in which you want to create your blockchain network, then click **Create Resource** in the upper right of the window.
3. On the catalog window, enter "blockchain" in the **Search** field and search for the blockchain service. Click the **Blockchain service** icon to open the blockchain service window.
4. On the blockchain service window, you can edit the service name or leave it as the default.

5. Scroll down on the page and select **Starter Membership Plan (beta)** from the Pricing Plans section. Click the **Create** button at the bottom of the page, as shown in Figure 4-1.


https://console.bluemix.net/catalog/services/blockchain

IBM Cloud — Catalog Docs Support Manage

Pricing Plans

Monthly prices shown are for country or region: United States

	PLAN	FEATURES	PRICING
✓	Starter Membership Plan (Beta)	**This plan entitles a member of the Blockchain Platform to:** - One-click setup and a fully functional kick starter network that is configured with 2 organizations and 1 peer per organization. - Easy development environment, sample applications, ordering service and certificate authority. - Free during Beta period, low-cost after general availability. Default Capabilities: - Ability to create an application in Composer and easily import the resulting BNA file. - Easy migration of chaincode to Enterprise Plan.	Free
	Enterprise Membership Plan	**This plan entitles a member of the Blockchain Platform to:** All Starter Plan features, combined with premium support, a secure blockchain environment for early production workloads, and added layers of security. Additional Capabilities: - Fault Tolerance that provides High Availability for ordering and certificate authority. - Secure Service Container that protects a shared compute with isolated runtime and data. - Hardware Security Module.	$1,000.00 USD/Membership Fee $1,000.00 USD/Small Peer

The IBM Blockchain Platform is the only fully integrated blockchain platform designed to accelerate the development, governing, and operation of a multi-institution business network. Starter plan has been created as a plan to quickly deploy sample applications, test basic applications, and begin to grow your network. This includes Hyperledger Composer integration and an easy on-ramp to recognize the value of a production network deployed on the Enterprise Membership Plan.

Need Help?
Contact IBM Cloud Sales

Estimate Monthly Cost
Cost Calculator

Create

Figure 4-1 Pricing Plans welcome window

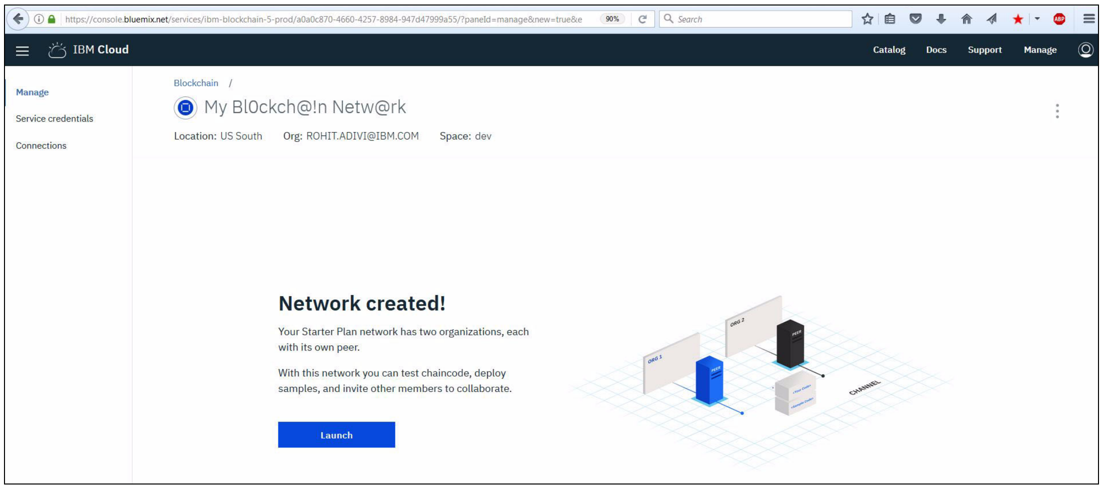
When you see the Welcome page, you have successfully created your first Blockchain Starter Plan network. A Starter Plan network comes configured with these features:

- A SOLO Orderer and two member organizations. Each organization has one peer and one CA.
- Peers of both organizations are joined to a provided channel named `defaultchannel`.

4.3 Acquiring network credentials and connection profiles for the member organizations

To acquire network credentials and connection profiles for the member organizations, complete these steps:

1. On the Welcome window, click **Launch** to enter the Network Monitor for your blockchain network, as shown in Figure 4-2.


https://console.bluemix.net/services/ibm-blockchain-5-prod/a0a0c870-4660-4257-8984-947d47999a55/?paneId=manage&new=true&e

IBM Cloud — Catalog Docs Support Manage

Manage
Service credentials
Connections

Blockchain /

My Bl0ckch@!n Netw@rk

Location: US South   Org: ROHIT.ADIVI@IBM.COM   Space: dev

Network created!

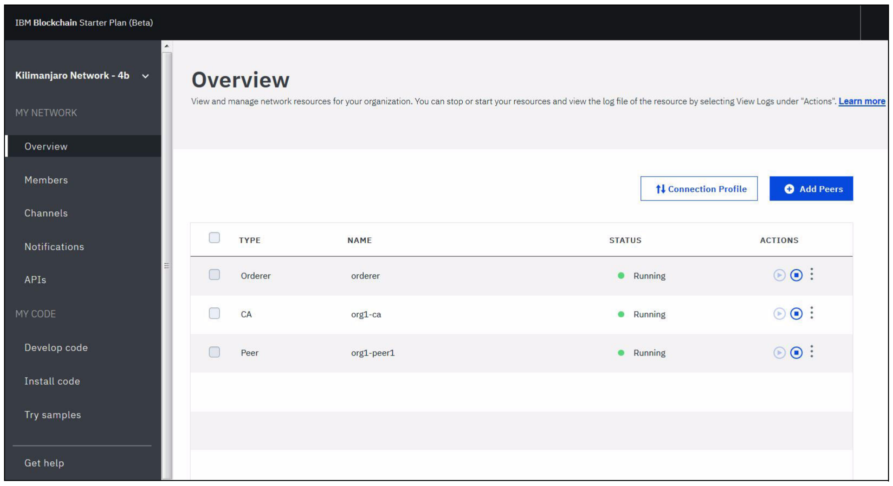
Your Starter Plan network has two organizations, each with its own peer.

With this network you can test chaincode, deploy samples, and invite other members to collaborate.

Launch

Figure 4-2 Network launch window

2. Read the instructions on the “Let’s get started” window to begin your blockchain journey.
3. On the Overview window, click **Connection Profile** to view and download your Connection Profile for `org1`. Save the raw JSON document in a known location for future use, as shown in Figure 4-3.



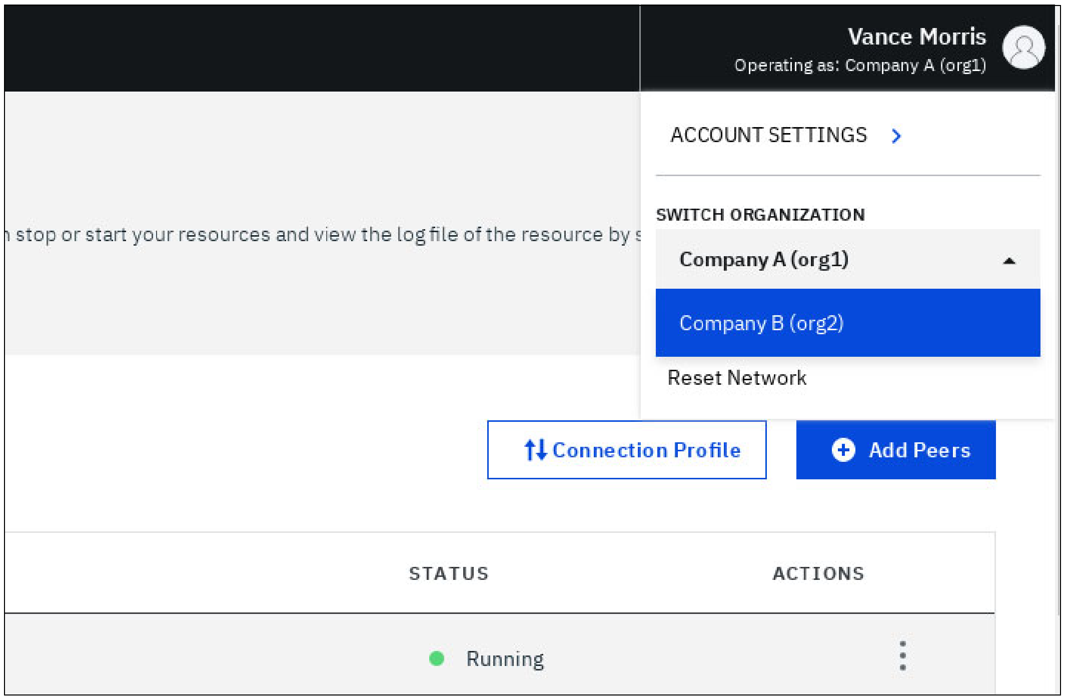
Overview

View and manage network resources for your organization. You can stop or start your resources and view the log file of the resource by selecting View Logs under "Actions". Learn more

TYPE	NAME	STATUS	ACTIONS
Orderer	orderer	● Running	
CA	org1-ca	● Running	
Peer	org1-peer1	● Running	

Figure 4-3 Overview window

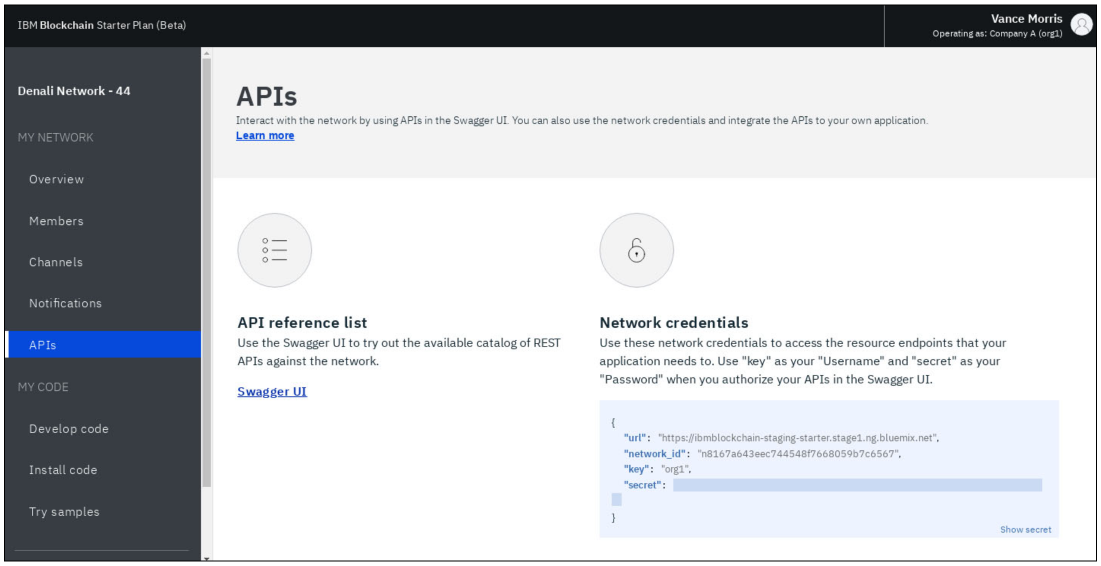
4. Switch to `org2` by clicking the **Operating As** menu in the top right, then download and save the Connection Profile for `org2` as well, as shown in Figure 4-4.



stop or start your resources and view the log file of the resource by

Figure 4-4 Switching to another organization

5. The APIs window contains unique Network Credentials for each org, as shown in Figure 4-5. These credentials can be used to access the REST APIs for interacting with your Fabric network in a programmatic manner.



APIs

Interact with the network by using APIs in the Swagger UI. You can also use the network credentials and integrate the APIs to your own application.
Learn more

API reference list
Use the Swagger UI to try out the available catalog of REST APIs against the network.

Swagger UI

Network credentials
Use these network credentials to access the resource endpoints that your application needs to. Use "key" as your "Username" and "secret" as your "Password" when you authorize your APIs in the Swagger UI.

```
{
  "url": "https://ibmblockchain-staging-starter.stage1.ng.bluemix.net",
  "network_id": "n8167a643eec744548f7668059b7c6567",
  "key": "org1",
  "secret":

}
```

Show secret

Figure 4-5   APIs window

4.4 Deploy the fund clearing business network to the IBP Starter Plan network

Before attempting to deploy to Starter Plan, discover any business network cards that exist in your wallet from previous deployments of the fund clearing application. If left in the Composer wallet, these cards will cause errors in the following steps. The importance of cards is explained in Chapter 3, "Basic flow using IBM Composer Playground" on page 21. Run the following command:

```
composer card list
```

Delete any cards that you find with the following command:

```
composer card delete -c <CARD_NAME>
```

4.4.1 Creating a certificate authority card in the CLI

To interact with the business network, you must create a business network card for each member organization.

Inside each organization's connection profile is a `registrar enrollSecret` property. Each organization has a unique connection profile, so you must switch between them to retrieve both profiles. Retrieve the secret and save a copy of it. Use the following commands to create the certificate authority business network cards for each member organization:

```
composer card create -f org1ca.card  -p org1ConnectionProfile.json -u admin -s
<org1EnrollSecret>
composer card create -f org2ca.card -p org2ConnectionProfile.json -u admin -s
<org2EnrollSecret>
```

Here `org1EnrollSecret` and `org2EnrollSecret` can be obtained from the `admin Enrollsecret` property in the Connection Profile of the respective organizations.

Import these cards from each organization by using the following commands:

```
composer card import -f org1ca.card -n org1ca
composer card import -f org2ca.card -n org2ca
```

Now that the cards are imported, use them to exchange the admin secret for valid certificates from the certificate authority (CA). Run these commands to request certificates from the CA:

```
composer identity request -c org1ca -d org1admin
composer identity request  -c org2ca -d org2admin
```

4.4.2 Adding the certificates to the Starter Plan instance

Now that the admin certificates have been obtained, they must be added to the Starter Plan instance. For convenience, they can be added by using the IBM Blockchain Platform API or IBM Blockchain Platform UI.

For each certificate, complete the following steps:

1. Add/upload the certificate.
2. Restart all the peers in the member organization with the new admin certificate.
3. Synchronize the certificate on the channel.

Depending on whether you prefer to add the certificate by using the UI or the API, select one of the following sections:

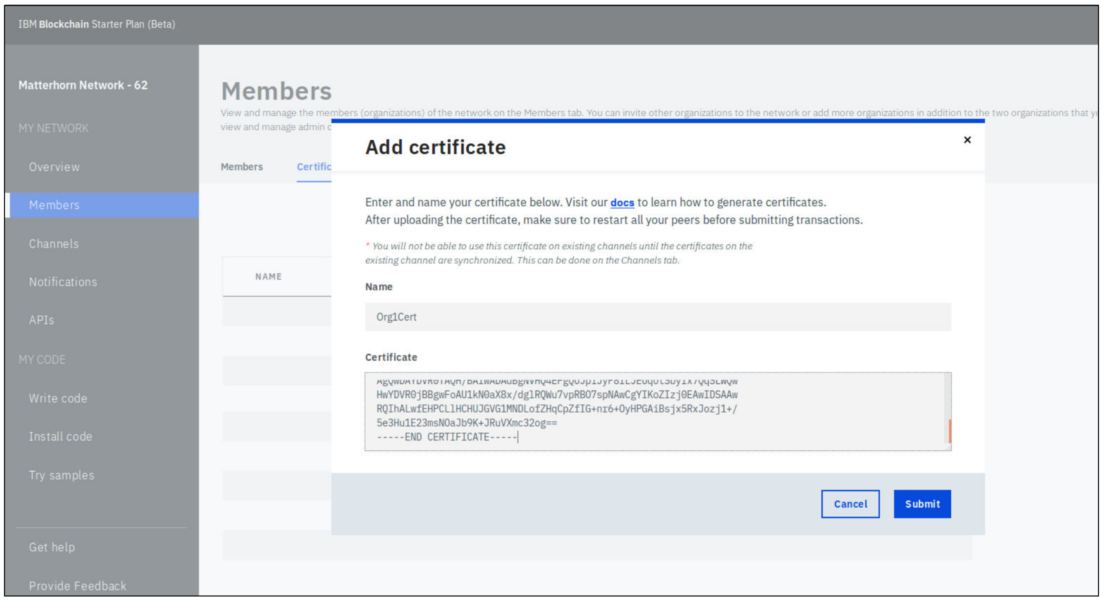
- Adding a certificate with the IBM Blockchain Platform UI
- Adding a certificate with the IBM Blockchain Platform API

Adding a certificate with the IBM Blockchain Platform UI

Note: Adding the certificate and syncing are different steps. First, you add the certificate. Then the peer is stopped and started, after which the certificate sync is completed. Repeat these steps for both `org1` and `org2`.

To add the certificate, complete these steps:

1. Add the certificate by navigating to the Members window, clicking the Certificates tab, and clicking **Add Certificate**.
2. Enter a name for the certificate and paste the certificate that is saved as `org1admin/admin-pub.pem` of member `org1`, then click **Submit**, as shown in Figure 4-6.



View and manage the members (organizations) of the network on the Members tab. You can invite other organizations to the network or add more organizations in addition to the two organizations that y
view and manage admin c

Add certificate

Enter and name your certificate below. Visit our docs to learn how to generate certificates.
After uploading the certificate, make sure to restart all your peers before submitting transactions.

* *You will not be able to use this certificate on existing channels until the certificates on the existing channel are synchronized. This can be done on the Channels tab.*

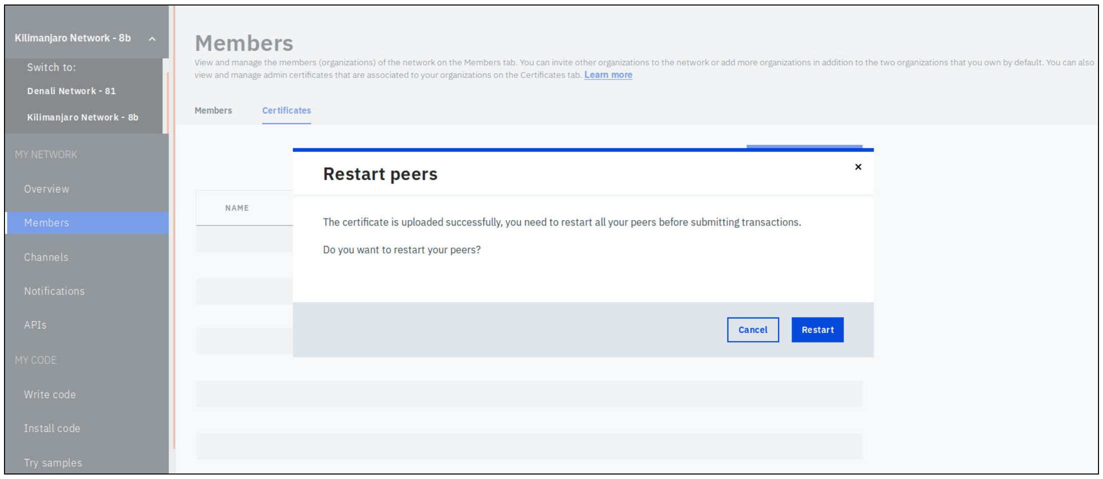
Name: Org1Cert

Certificate:

```
HwYDVR0jBBgwFoAU1kN0aX8x/dg1RQWu7vpRBO7spNAwCgYIKoZIzj0EAwIDSAAw
RQIhALwfEHPCL1HCHUJGVG1MNDLofZHqCpZfIG+nr6+OyHPGAiBsjx5RxJozj1+/
5e3Hu1E23msNOaJb9K+JRuVXmc32og==
-----END CERTIFICATE-----
```

Figure 4-6   Add certificate window

3. After the certificate is added, you will see a prompt to restart the peers. Click **Restart**. This process might take up to two minutes, after which you see a message stating that “All running peers were restarted successfully”, as shown in Figure 4-7.



View and manage the members (organizations) of the network on the Members tab. You can invite other organizations to the network or add more organizations in addition to the two organizations that you own by default. You can also view and manage admin certificates that are associated to your organizations on the Certificates tab. Learn more

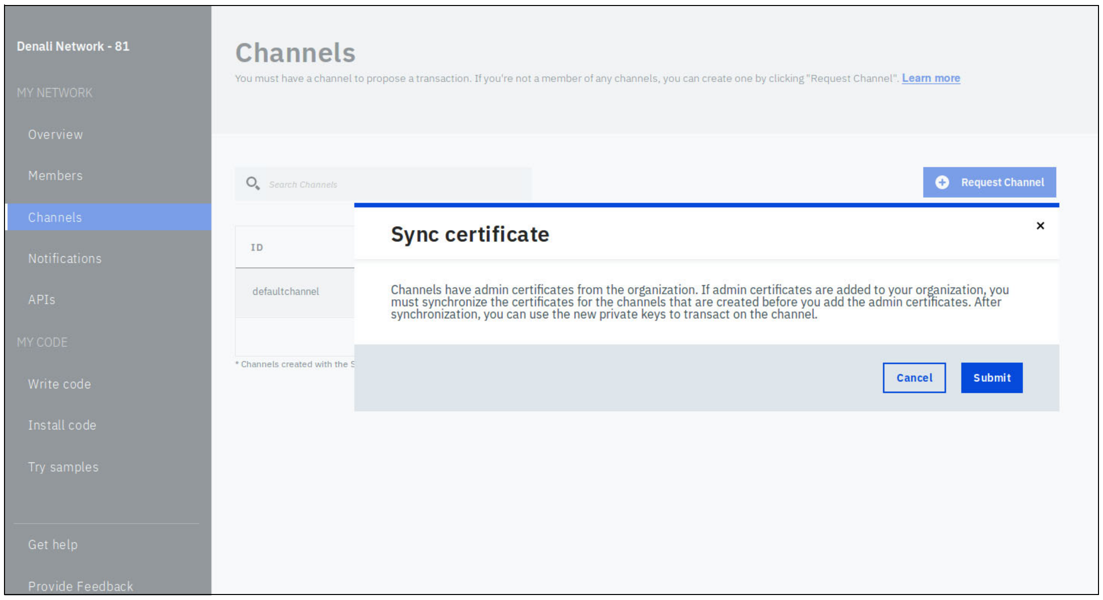
Restart peers

The certificate is uploaded successfully, you need to restart all your peers before submitting transactions.

Do you want to restart your peers?

Figure 4-7   Restart peers window

4. Navigate to the Channels page and sync the certificate on the channels that you created before adding the certificate. This step can be done by clicking **Actions** → **Sync certificate**, then click **Submit**.
5. The confirmation window is shown in Figure 4-8. Click **Submit**. Syncing updates all the org admin certificates in the channel.



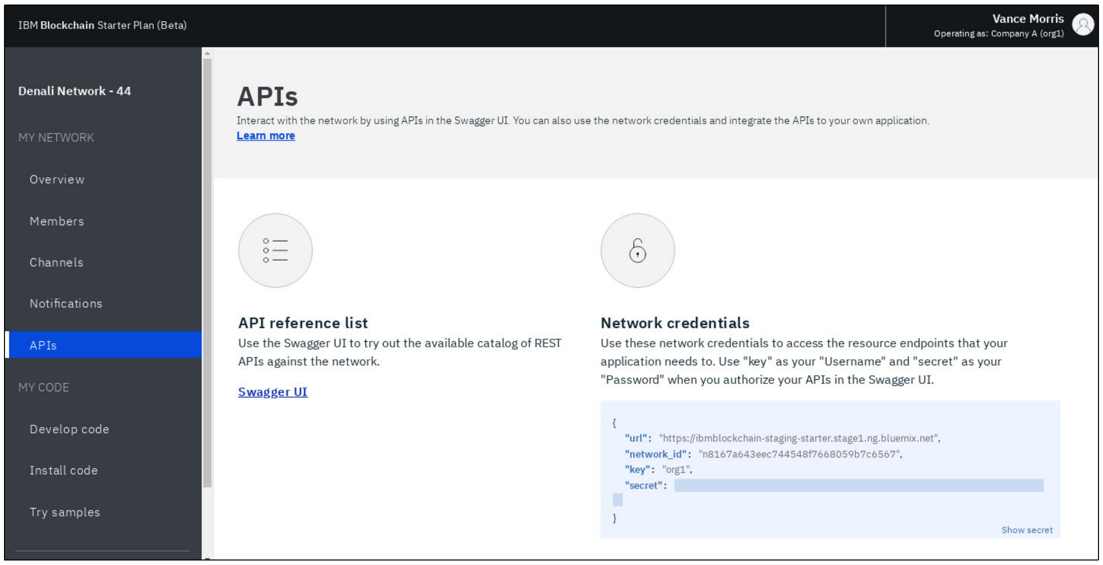
You must have a channel to propose a transaction. If you're not a member of any channels, you can create one by clicking "Request Channel". Learn more

Channels have admin certificates from the organization. If admin certificates are added to your organization, you must synchronize the certificates for the channels that are created before you add the admin certificates. After synchronization, you can use the new private keys to transact on the channel.

Figure 4-8 Sync certificate window

Adding a certificate with the IBM Blockchain Platform API

Note: The values required in the variables that are used in the `curl` commands below are contained in the Connection Profiles and Network Credentials JSON documents for both organizations. The Network Credentials for each org are viewable by clicking the APIs tab, as shown in Figure 4-9.



Interact with the network by using APIs in the Swagger UI. You can also use the network credentials and integrate the APIs to your own application.

Use the Swagger UI to try out the available catalog of REST APIs against the network.

Use these network credentials to access the resource endpoints that your application needs to. Use "key" as your "Username" and "secret" as your "Password" when you authorize your APIs in the Swagger UI.

```
{
  "url": "https://ibmblockchain-staging-starter.stage1.ng.bluemix.net",
  "network_id": "n8167a643eec744548f7668059b7c6567",
  "key": "org1",
  "secret":
}
```

Figure 4-9 Viewing Network Credentials in the APIs window

To add a certificate using the IBM Blockchain Platform API, complete these steps using a CLI window:

1. Add the admin certificate for `org1` using the following API. The ADMIN CERT value can be obtained by executing the following command:

   ```
   awk '{printf "%s\\n", $0}' org1admin/admin-pub.pem
   ```

2. Upload with the following command:

   ```
   curl -s -X POST \
        --header 'Content-Type: application/json' \
        --header 'Accept: application/json' \
        --basic --user org1:<<secret1>> \
        --data '<body1.json content goes here>' \
        <NETWORK_API_URL>/api/v1/networks/<NETWORK_ID>/certificates
   ```

   Where `<secret1>` is the value obtained in the API keys for `org 1`, and the `body1.json` content is as follows:

   ```
    {
     "msp_id": "org1",
     "adminCertName": "PeerOrg1AdminCert",
     "adminCertificate": "<ADMIN CERT>",
     "peer_names": [
       "org1-peer1"
     ],
     "SKIP_CACHE": true
    }
   ```

   Repeat these commands for `org2`.

3. Restart the peers in `org1`.

   Stop peers:

   ```
   curl -s -X POST \
       --header 'Content-Type: application/json' \
       --header 'Accept: application/json' \
       --basic --user org1:<<secret1>> \
       --data-binary '{}' \
       <NETWORK_API_URL>/api/v1/networks/<NETWORK_ID>/nodes/org1-peer1/stop
   ```

   Start peers:

   ```
   curl -s -X POST \
       --header 'Content-Type: application/json' \
       --header 'Accept: application/json' \
       --basic --user org1:<secret1> \
       --data-binary '{}' \
       <NETWORK_API_URL>/api/v1/networks/<NETWORK_ID>/nodes/org1-peer1/start
   ```

   Check the peer status to ensure that the peers are in running state before continuing:

   ```
   curl -s -X GET \
       --header 'Content-Type: application/json' \
       --header 'Accept: application/json' \
       --basic --user org1:<secret1> \
       <NETWORK_API_URL>/api/v1/networks/<NETWORK_ID>/nodes/status
   ```

   Note: Peer start can take up to five minutes.

   Repeat these commands for `org2`.

4. Synchronize all the certificates for `defaultchannel`. This action ensures that all the peers can verify signatures for a specified channel.

   ```
   curl -s -X POST \
         --header 'Content-Type: application/json' \
         --header 'Accept: application/json' \
         --basic --user org1:secret1 \
         --data-binary '{}' \

   <NETWORK_API_URL>/api/v1/networks/<NETWORK_ID>/channels/defaultchannel/sync
   ```

   Repeat these commands for `org2`.

4.4.3 Creating an admin business network card

Now that the admin certificates have been synced with the peers, business network cards can be created that have the permissions to install the Hyperledger Composer run time and start chaincode. To do so, complete these steps using a CLI window:

1. Create an admin card with the channel admin and peer admin roles:

   ```
   composer card create \
       -f org1admin/admin.card \
       -p org1ConnectionProfile.json \
       -u admin \
       -c org1admin/admin-pub.pem \
       -k org1admin/admin-priv.pem \
       --role PeerAdmin --role ChannelAdmin
   ```

   This card is used to deploy a business network to Starter Plan.

2. Import the card created in the previous step:

   ```
   composer card import  -f org1admin/admin.card -n org1admin
   ```

   Repeat these commands for `org2`.

4.4.4 Installing and starting the business network

To install and start the business network, complete these steps using a CLI window:

1. Generate the fund clearing Business Network Archive (BNA) file:

   ```
   composer archive create -t dir -n ./“fund-clearing-network”/
   ```

2. Install the Composer runtime on both `org1` and `org2` peers using the respective organization admins:

   ```
   composer runtime install -c org1admin -n “fund-clearing-network”
   composer runtime install -c org2admin -n “fund-clearing-network”
   ```

3. Start the business network using both organization admin cards and certificates:

   ```
   composer network start -c org1admin -a fund-clearing-network@0.1.0.bna \
       -A org1admin -C org1admin/admin-pub.pem \
       -A org2admin -C org2admin/admin-pub.pem
   ```

   Note: You might need to run this command multiple times.

4.4.5 Creating a card to access the business network as org1 and org2 admins

To create business network cards to access the network, complete these steps:

1. Create business network cards for `org1` and `org2` admins:

   ```
   composer card create -f org1admin/admin.card  -p org1ConnectionProfile.json -u
   admin -c org1admin/admin-pub.pem -k org1admin/admin-priv.pem --role PeerAdmin
   --role ChannelAdmin -n fund-clearing-network

   composer card create -f org2admin/admin.card  -p org2ConnectionProfile.json -u
   admin -c org2admin/admin-pub.pem -k org2admin/admin-priv.pem --role PeerAdmin
   --role ChannelAdmin -n fund-clearing-network
   ```

2. Import the new cards:

   ```
   composer card import -f org1admin/admin.card -n admin1@fund-clearing-network
   composer card import -f org2admin/admin.card -n admin2@fund-clearing-network
   ```

3. At this stage, you can restart the Composer Playground and interact with the remote network.

5

Moving to production

You should now be familiar with multiple blockchain for business use cases and how to get started in defining, developing, and deploying a Hyperledger Composer business network to Hyperledger Fabric, both locally on your workstation and remotely on the IBM Blockchain Platform Starter Plan.

Note: The content in this paper is accurate to a point-in-time. At the time of writing, the IBM Blockchain Platform Starter Plan is in beta. We expect there to be enough changes to require a refresh of this paper when the offering becomes generally available.

This chapter discusses the next steps in producing a real-world bi-lateral funds clearing business network. It includes the following sections:

- Moving to production
- Integration with traditional back-end applications

5.1 Moving to production

This paper details how to develop a bilateral funds clearing business network in Hyperledger Composer, and deploy that network to IBM Blockchain Platform Starter Plan.

It is our intention to continue to develop this story in subsequent papers. Content is already identified for the next paper:

- Integrating two traditional CICS banking applications running in z/OS with the fund clearing network, demonstrating the full end-to-end system with real world examples.
- Using IBM Cloud Toolchains to implement a Continuous Integration and Continuous Deployment flow for the fund clearing business network.
- Migrating the fund clearing business network to IBM Blockchain Platform Enterprise Plan.
- Using Hyperledger Caliper to determine best practices for scalability and capacity planning.

5.2 Integration with traditional back-end applications

IBM CICS is one of the most prolific online transaction processors. Its focus on reliable, lightning-fast transactions makes it ideal for many industries, including the high-volume financial industry. Although CICS supports many programming languages, most legacy CICS applications are written in COBOL.

The back-end banking application that we use to access the bilateral fund clearing business network is indeed a COBOL application that runs in IBM CICS. This application pre-existed blockchain technology by decades and implements a classical transaction-oriented process. Bank employees navigate textual screens to perform balance inquiries, open new accounts, and transfer money from one account to another. Bank customers can request the same operations through secured HTTPS connections provided by IBM CICS Web Services.

To integrate with the fund clearing business network, a new COBOL transaction is added to the banking application that allows transfers to another bank. The following is the end-to-end flow:

1. An inter-bank transfer is requested.
2. The requesting bank performs error checking for conditions within their bank, such as these questions:
   a. Is this a valid customer and account?
   b. Is there a temporary transaction hold on this account?
   c. Has the daily maximum for fund transfers been exceeded?
3. The COBOL application performs a REST call from the CICS environment to a Composer REST server to start the `CreateTransferRequest` transaction, placing a new `TransferRequest` asset on the business network.

4. The remaining steps in resolving the funds transfer between the banks are detailed in the fund clearing `README.md` file, which you can download from GitHub. These integration points about the fund clearing `BatchTransferRequest` must be addressed:
   - Each bank must collect the `BatchCreated` events that are issued by the fund clearing business network.
   - When participating in a `BatchTransferRequest`, each bank must perform pre-processing and post-processing actions to update internal ledgers as required. They also must update the `BatchTransferRequest` by invoking appropriate transactions through the Composer REST server.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this paper.

IBM Redbooks

The following IBM Redbooks publications provide additional information about the topic in this document. Note that some publications referenced in this list might be available in softcopy only.

- *IBM CICS*: Video course series from IBM Redbooks

  `http://www.redbooks.ibm.com/redbooks.nsf/pages/cicsvideo?Open`
- *Practical Migration from x86 to LinuxONE*, SG24-8377
- *Scale up for Linux on IBM Z*, REDP-5461
- *Zero to Blockchain*: An IBM Redbooks course

  `http://www.redbooks.ibm.com/redbooks.nsf/redbookabstracts/crse0401.html?Open`

You can search for, view, download or order these documents and other Redbooks, Redpapers, Web Docs, draft and additional materials, at the following website:

ibm.com/redbooks

Online resources

These websites are also relevant as further information sources:

- Developer tutorial for creating a Hyperledger Composer solution

  `https://hyperledger.github.io/composer/latest/tutorials/developer-tutorial`
- The `composer-vscode-plugin` for Hyperledger Composer

  `https://github.com/hyperledger/composer-vscode-plugin`
- API Class Index for Hyperledger Composer

  `https://hyperledger.github.io/composer/latest/api/allData.html`
- Key Concepts in Hyperledger Composer

  `https://hyperledger.github.io/composer/latest/introduction/key-concepts.html`
- Runtime API for Hyperledger Composer

  `https://hyperledger.github.io/composer/latest/api/allData#runtime-api`
- composer-sample-networks for Hyperledger Composer

  `https://github.com/hyperledger/composer-sample-networks/tree/master/packages/fund-clearing-network`
- Hyperledger Composer Playground

  `https://composer-playground.mybluemix.net/`

- Deploying a Hyperledger Composer blockchain business network to Hyperledger Fabric (multiple organizations)

  https://hyperledger.github.io/composer/latest/tutorials/deploy-to-fabric-multi-org
- Mocha JavaScript test framework

  https://mochajs.org/
- composer-cucumber-steps from NPM

  https://www.npmjs.com/package/composer-cucumber-steps
- IBM Cloud

  http://bluemix.net/
- Account types in IBM Cloud

  https://console.bluemix.net/docs/account/index.html#pricing

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

REDP-5492-00

ISBN 0738456853

Printed in U.S.A.