O'REILLY®

Getting Started with Enterprise Blockchain

A Guide to Design and Development

Michael Bradley, David Gorman, Matt Lucas & Matthew Golby-Kirk

Getting Started with Enterprise Blockchain

A Guide to Design and Development

Michael Bradley, David Gorman, Matt Lucas, and Matthew Golby-Kirk

Beijing • Boston • Farnham • Sebastopol • Tokyo

Getting Started with Enterprise Blockchain
by Michael Bradley, David Gorman, Matt Lucas, and Matthew Golby-Kirk

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

Acquisitions Editor: Michelle Smith
Development Editor: Michele Cronin
Production Editor: Christopher Faucher
Copyeditor: James Fraleigh
Proofreader: Matthew Burgoyne
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Rebecca Demarest

April 2019: First Edition

Revision History for the First Edition
2019-04-19: First Release

978-1-492-05268-5

[LSI]

Table of Contents

Foreword

The hype around blockchain (thankfully!) peaked in early 2018, largely driven by financial speculation around Bitcoin and Ethereum crypto-currencies. Despite inauspicious beginnings, and the "Isn't this just a database?" technical naysayers, blockchain technology is a new and powerful set of tools for solving reliable and automated intercompany code and data sharing, using cryptography to enforce access control, privacy, and encryption.

Historically, building a consortium with many member corporations, and putting in place automated sharing of data, has been reserved to a very small number of use cases, and to companies with very deep pockets—typically in banking, for example, the Visa or Swift payment networks.

Open source enterprise blockchain technology, notably Hyperledger Fabric, and managed Blockchain as a Service platforms, from IBM and others, empower business leaders and technologists to cost-effectively build consortia for many more use cases and industries, across a broad set of economic (business) networks.

Enterprises have spent the past 10+ years digitizing their *internal* IT processes to the point that the time is now ripe to look at the lack of digitization and inefficiencies *between* enterprises within a business network. Today, most enterprises exchange data with their business partners via an unholy mix of emails, documents, fax, phone calls, CSV- and Excel-format flat files, database dumps, EDI, and web service calls. We can do so much better!

At Clause, via the Open Source Accord Project, we focus on ensuring that the legal contracts that govern business exchanges can also be digitized, and that logic automated using a blockchain smart contract represents the intent of the parties to a governing legal contract.

Blockchain forces business and technical leaders to confront fundamental (and difficult!) issues around standardization of data formats, trust, privacy, governance, and legal/regulatory oversight. The transformation to blockchain-mediated business networks will not happen overnight, but will play out slowly, but surely, over the next decade or more.

— Dan Selman
CTO, Clause Inc.

Preface

This book introduces you to the value that blockchain brings to the enterprise. We focus on private permissioned blockchains, which we describe in Chapter 1, and why they are suitable for use in an enterprise environment. While we make comparisons to public unpermissioned networks in Chapter 1, we don't discuss public networks beyond this. This book looks at how to choose the best scenario for blockchain, and at considerations for designing a blockchain network, before finally looking at how to develop your blockchain application.

Throughout this book we focus on a real-world scenario for *commercial papers* with Chapter 5 introducing a developer tutorial and how to run and extend a *commercial paper* network on blockchain.

Who Is This Book For?

While this book is focused on those people designing and developing blockchain networks, the early chapters have a broader scope. There's no prerequisite reading required, although if you would like to extend the commercial paper scenario in Chapter 5, then knowledger of Node.js would be advantageous.

How Is This Book Organized?

This book contains six chapters, each chapter building on the previous.

- Chapter 1: Introduces blockchain for the enterprise, and describes the main concepts of a private permissioned blockchain.
- Chapter 2: Looks at how to identify when to use blockchain, and includes a set of detailed steps on how best to choose a good blockchain scenario.
- Chapter 3: Considers many of the options you will face when designing a blockchain network.
- Chapter 4: Describes the artifacts that need to be developed for a blockchain network, namely smart contracts and client-side applications.
- Chapter 5: Explains how to run the commercial paper scenario, and how to extend the scenario with additional functions.
- Chapter 6: Takes a look at the future of blockchain for the enterprise.

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic
: Indicates new terms, URLs, email addresses, filenames, and file extensions.

`Constant width`
: Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

`Constant width bold`
: Shows commands or other text that should be typed literally by the user.

`Constant width italic`
: Shows text that should be replaced with user-supplied values or by values determined by context.

NOTE

This element signifies a general note.

O'Reilly Online Learning

O'REILLY®

For almost 40 years, *O'Reilly Media* has provided technology and business training, knowledge, and insight to help companies succeed.

Our unique network of experts and innovators share their knowledge and expertise through books, articles, conferences, and our online learning platform. O'Reilly's online learning platform gives you on-demand access to live training courses, in-depth learning paths, interactive coding environments, and a vast collection of text and video from O'Reilly and 200+ other publishers. For more information, please visit *http://oreilly.com*.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

To comment or ask technical questions about this book, please send an email to *bookquestions@oreilly.com*.

For more information about our books, courses, conferences, and news, see our website at *http://www.oreilly.com*.

Find us on Facebook: *http://facebook.com/oreilly*

Follow us on Twitter: *http://twitter.com/oreillymedia*

Watch us on YouTube: *http://www.youtube.com/oreillymedia*

Acknowledgments

A special thanks to our colleague Evie Wright, for reviewing our draft versions of this book, providing valuable feedback and being part of the many discussions as we worked on this book. Evie joined our IBM Blockchain Engagement Team for a year out from her undergraduate studies at university.

We also want to thank our following colleagues for reviewing our draft versions, without which this book would not have received the much-needed sponsorship, direction, and support required:

- Anita Chung
- Alejandro Pinto
- R. Colby Murphy

Finally, thanks to our colleague Horea Porutiu for reviewing and providing feedback on the draft version.

CHAPTER 1

Introduction to Blockchain

Blockchain is a shared, distributed ledger that records transactions across business networks with the aim of helping businesses remove inefficiencies from trade. It does this through the use of cryptographic proofs, which help engender trust by ensuring facts are reported consistently to all those with a need to see them.

That's the elevator speech for blockchain. In the rest of this chapter, we'll look in more detail into what blockchain is and why it is so important. To begin with, let's explore the need for business networks.

Business Networks

Wealth is generated in market economies by the flow of goods and services over business networks. Business networks are necessary because of the huge advantages gained from specialization. The economist Adam Smith first wrote about these efficiency gains by referring to the example of pin production, in work that is currently immortalized on the back of an English £20 note. The economist Leonard Read, in his short paper "I, Pencil," subsequently used the example of a pencil, and the fact that no one on earth knows how to create one completely from scratch, to demonstrate the importance of business networks to manufacturing.

Business networks are not only required to make manufacturing supply chains more efficient; they are also used in any type of business-to-business interaction. Business networks can span manufacturers, logistics companies, banks, insurers, consumers—any person or organization who is willing to trade one asset for another, whether that asset is tangible (e.g., houses, cars, cash, land) or intangible (e.g., services, intellectual property, patents, licenses).

Let's use an example to illustrate the importance of the business network and how wealth is generated.

Ted the Businessman

Ted is a businessman who turns up in a small town to speak at a conference the following day. He was booked by the conference at the last minute, so when he arrives at the town he still needs a hotel to spend the night. He finds one, goes up to the owner, and reserves a room, placing $100 on the front desk. "I'll be back shortly," he says to the hotel owner. "I'm going to check out the conference venue first." Ted wanders off.

A few moments later, the hotel owner takes the $100 to a local builder who did some minor repair work at the hotel the week before. "Here's the $100 I owe you," the hotel owner says to the builder. The owner returns to her hotel.

The builder, money in hand, had been interested in a nice vase that had been on display in the local antique shop. He takes the $100 to the antique shop, purchases the vase, and happily takes it back to his house.

The owners of the antique shop were an elderly couple who happened to be reaching their 40th wedding anniversary. To celebrate, they take the $100 they just received from the builder and decide to spend the night in a local hotel—the same hotel in which Ted had reserved a room earlier. The couple arrives at the front desk, pays $100 to the hotel owner, checks into their room, and has a lovely short break.

After checking out the conference venue, Ted returns to the hotel and delivers some bad news to the hotel owner. "I'm sorry," says Ted. "It turns out my conference has been canceled. Please can I have my money back?"

The hotel owner agrees and hands the $100 back to Ted, who then returns home, disappointed to not have spoken at the conference but otherwise not out of pocket.

What happened in that story?

The story starts and ends with precisely the same quantity of assets in the town, but crucially the different participants of this business network have each gained utility due to the flow of capital (the $100) within it: the hotel owner has rented a room, the builder has a vase, and the antique shop owners have their vacation. Ted the businessman simply served as a stimulus to start the flow of transactions, providing the initial capital input and final capital output.

You can see this flow in Figure 1-1.

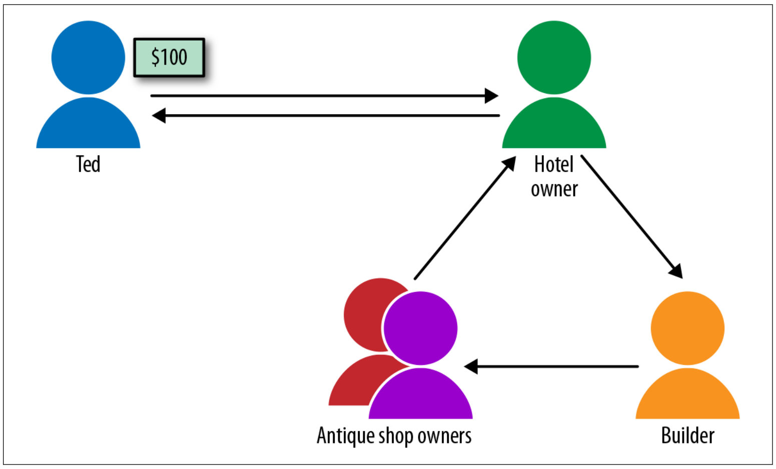

Figure 1-1. The flow of transactions in the town's business network

It's a simplistic scenario, but the lesson here is that what's most important for the generation of utility (and ultimately, wealth) is the flow of assets (goods, services, and cash) around a network. You can manufacture a plethora of assets but if no one is willing to exchange other goods and services (or cash) for them, true economic growth cannot occur. (There are other ways of artificially generating wealth, such as gains made from stock market speculation. It's beyond the scope of this book, but we refer anyone interested to any good economics guide.)

In reality, business networks can be hugely complex. It can take many hundreds of suppliers and manufacturers to source and assemble the components for your car. The value of fund transfers between US financial institutions comes in at trillions of US dollars each day, and this requires a business network system that is highly available, efficient, and resilient.

So what's this got to do with blockchain? Well, a blockchain is a system for *logging the flow of assets* around a business network; it allows the participants associated with transactions to record that they happened. Of course, we already have a system for doing this—it's called the ledger. How is blockchain any different?

The Ledger

Depending on when you start counting, the concept of ledgering has been around for over 500 years. In 1494, a Franciscan monk and mathematician called Luca Pacioli published *Summa de arithmetica, geometria; proportioni et proportionalita* (*Summary of arithmetic, geometry, proportions, and proportionality*), which gave the first written description of the concept of double-entry bookkeeping, giving rise to ledgers as we know them today.

Ledgers are essentially transaction logs: ordered lists of the inputs and outputs of a business. Your bank statement is an example of a personal ledger; it shows every debit and credit to your account and details of the organization with whom you transacted.

Ledgers are particularly important for businesses because they describe what they own, and what the business would be worth if it was to be sold. A company could take its starting balance and for every transaction add up the value of the credits, and subtract the value of the debits, to derive its net worth. The net worth derived from company ledgers is known as a *liquidity position*, which among other things can be used to influence investment decisions, satisfy auditors, and manage risk.

Ledgers have moved on since the leather-bound books of Luca Pacioli's day, and are now usually represented by a database of some kind. But the concepts—and the shortcomings—remain the same. But more on those later.

Transactions and Contracts

If a ledger is a log of transactions, what actually is a transaction? A transaction is usually taken to mean the change in ownership of an asset in exchange for something else. So if Matt buys Helen's car from her, the logged transaction is the exchange of one car to Matt for the agreed price. Matt would record that transaction on his ledger and Helen would record the same transaction on hers (the debit becoming the credit and the credit becoming the debit, of course).

A transaction could be the exchange of two goods, one of which may be cash. More generally, a transaction could be taken to mean any change in state of an asset. If respraying a car from yellow to green increases its resale value, this might then have an effect on the net worth of the owner, so it could be useful to log in a ledger.

Transactions are underpinned by a set of terms and conditions, which are the prerequisites for that transaction to be valid. When buying Helen's car, it might come with a warranty that says that if the car turns out to be faulty, Matt gets his money back. For many transactions, the terms of the transaction are agreed by the participants and represented in a contract.

Transactions and contracts are closely linked; a transaction can be thought of as being the invocation of the rules of a contract, and this is precisely how blockchains tend to execute their implementations of contracts (also known as *smart contracts*). Again, more on this later.

The Problems with Ledgers and Contracts

Both traditional ledgers and traditional contracts have problems. The problem with ledgers is that everyone in the business network has their own version. What if one participant has recorded a transaction on their ledger, but the other participant has recorded it differently, or even has no record of it? What if Matt records the fact that he's bought Helen's car, but Helen does not: whose ledger is correct, and who owns the car? Importantly for businesses, while transactions are in dispute, assets cannot be claimed as part of a company's net worth. It is common for companies to have a number of transactions in dispute at any given time. For example, IBM's Global Financing arm has a great deal of money tied up in disputes, up to $100 million in capital in dispute at a time, which is a significant liability with a dispute resolution process that can take many weeks to settle a transaction.

It's a similar problem for contracts: these are ambiguous by their very nature. Contracts represent an abstract agreement between multiple participants; what is written down on paper or implemented in IT systems may not be the essence of what the participants thought they were signing up to. Contracts can often take teams of lawyers and judges to interpret.

The lack of a mutually agreed meaning for both ledgers and contracts leads to the lengthy and costly process of reconciliation, which ensures that transaction details and contract execution are correctly synchronized and agreed by all the relevant participants. Disputes lead to (often manual) dispute resolution processes, which in turn can lead to even lengthier and costlier legal processes. This may be great for lawyers, but not so much for the participants involved in those disputes.

Enter the Blockchain

Blockchain aims to solve the problems of ledgers and contracts by sharing them in an unambiguous form between the participants of the business network. Blockchain can be referred to as a *shared, distributed ledger* with smart contracts.

Let's pick those terms apart. Firstly, blockchain is a *ledger*: a transaction log that can be used to describe the inputs and outputs of a business. Secondly, it's *shared* between the participants of the network such that everyone sees the same set of facts. Finally, it avoids any single point of failure by ensuring that copies are *distributed* to everyone who needs to see it.

Smart contracts refer to computer code that is shared between participants of the business network, and these implement the business rules associated with each transaction. As the code is shared, it can be executed by all relevant participants and they can agree on the output.

Despite the name, smart contracts are not contracts. Smart contracts can be used to provide the shared execution of contract terms, in the same way that a single participant's IT systems can execute contract terms today. You would not expect a judge to rule over the meaning of a smart contract, as they are not typically software engineers. They might, however, rule over the legal contract from which it derives.

A Car Example

To illustrate how shared ledgers and smart contracts work, imagine the scenario of a blockchain that tracks car ownership. The blockchain itself would be an ordered data structure that contains the details of every ownership change that has been agreed. For example, one transaction could state that "the ownership of the car with identification number CXU7592875 has changed from Helen to Matt." The smart contract associated with this transaction describes the computer logic that makes the transaction happen: for example, "check that the seller is equal to the current owner, and if it is, set the owner to be the buyer, decrement the cash balance of the buyer, and increment the cash balance of the seller."

Each interested participant (e.g., Helen and Matt) would run that smart contract code and agree on the output, and if everything is acceptable, then update the ledger accordingly. The updated ledger might be made visible to Helen and Matt, but also to other interested parties, such as the vehicle regulator or insurers, depending on the agreed rules of the network.

Blockchains are useful because they help engender trust in business networks, which is something traditional business-to-business systems cannot provide. With a shared database for example, there is no guarantee that an administrator has not tampered with the transaction. Blockchain gives the participants of a transaction nonrepudiation, which is evidence that the transaction was agreed to.

Blockchain and Trust

The blockchain can engender trust by providing proof that the transactions were agreed to. To do this, the blockchain implements several related qualities of service including *consensus*, *provenance*, *immutability*, and *finality*.

- *Consensus* is the process by which transactions are agreed upon by participants on the network. This means agreeing which transactions occurred, in what order, and what the result of running each transaction was. Participants who need to provide that agreement might just be those affected by it (in our example, just Matt and Helen), but might also include additional interested participants (such as a payments provider), or in the case of public blockchains like Bitcoin, a majority of the network.
- *Provenance* means that it should be possible to review prior transactions to determine the history associated with assets. For

example, as the current owner of the car, Matt should be able to see its manufacturing, ownership, and servicing history, right up to the point that he sells it. These rules of visibility are decided and governed by the network—more on this later.

- *Immutability* is the fact that the shared transaction history cannot be tampered with. Once a transaction has been agreed to through consensus by the network and stored on the blockchain, it cannot then be edited, deleted, or have new transactions inserted before it. This makes the blockchain an append-only data structure, and is the reason why transaction provenance is possible.
- *Finality* is the property that the transaction cannot be modified once it is agreed upon. Unintended transactions can only be backed out by the addition of a new transaction that reverses the earlier transaction, again with the agreement of the relevant participants.

Bear in mind that proof is not the same as trust. Blockchain provides cryptographic proof of the set of transactions, and it is up to participants to decide whether to trust that proof. In most business scenarios this is not a problem. But imagine, for example, that a blockchain were used to track voting in a national election. Such a system could provide secure once-only voting with a full audit trail, but would crucially need to be trusted by both the losers of the election and the electorate in general. Given many people's distrust in computers, this is not a trivial problem to solve; blockchain can provide the technical component of a solution to a problem that requires more than just technology.

Blockchain and Bitcoin

Blockchain is not the same as Bitcoin, although many people's first experience of blockchain is through Bitcoin. Bitcoin is a payment system that was first described in a 2008 whitepaper presented under the name of Satoshi Nakamoto.[1] This whitepaper doesn't actually mention blockchain by name, but describes a mechanism for securely sending payments between anonymous participants.

1 Satoshi Nakamoto is the name used by the unknown person or group of people who developed Bitcoin.

Bitcoin uses what we now refer to as a blockchain to log the set of confirmed transactions.

Bitcoin introduced a class of asset called a “cryptocurrency,” which is like a normal currency in that it is a scarce resource that can be used (in theory) to pay for goods and services. Since the introduction of Bitcoin, cryptocurrencies have exploded in popularity. At the time of writing, the market capitalization of cryptocurrencies has fallen dramatically, but the number of cryptocurrencies still far outnumber the number of fiat currencies.

The cryptocurrency bitcoin doesn’t really exist outside of the Bitcoin network, in the same way that the balance you hold in a bank account probably doesn’t exist outside a number in a record in a bank database—until you go to an ATM, anyway. The bitcoin currency can be converted into fiat currency using exchanges. Given this ability to cash out and provide real-world capital, bitcoin has been treated like a commodity such as gold. However, unlike other commodities, Bitcoin is largely unregulated and widely misunderstood, which has led, among other things, to huge volatility in its market price.

The Bitcoin Network

The Bitcoin network is pseudonymous and the *blockchain* is fully public; you can view the entirety of the Bitcoin ledger. This pseudonymity means that unless you can perform heavyweight analysis of the network (for example, ChainAnalysis), such as tracking transactions at the exchanges, it is almost impossible to determine the identity of a Bitcoin user. This is why hackers often request payment in bitcoin if your computer has been compromised. Ransom payments cannot be easily traced back to individuals, as the participants associated with individual transactions are meaningless sequences of hexadecimal numbers.

Given this lack of identity on the Bitcoin network, Bitcoin has an innovative but costly technique for ensuring consensus. As an example, imagine you had $100 and you attempt to transfer $50 to three participants at the same time. How does the network agree which two transactions succeed and which one fails? This is known as the double-spending problem, and is an example of the type of processing that the Bitcoin network does.

It's crucial that the network agrees on the order in which transactions are performed, and on the results of those transactions. Otherwise, the system could not reliably function. It uses a complex process of consensus called "Proof of Work" to do this, which relies on a network of collaborating computers.

NOTE

What Is Proof of Work?

Proof of Work works by adding an artificial cost to transaction verification. If you want to convince the network that your set and order of transactions is correct, you have to prove to the network that you have incurred this cost. It does this by forcing the nodes of the network to solve cryptographic puzzles that are difficult to solve (i.e., require brute force techniques), but are trivially easy for other nodes to verify once the correct answer has been found. For each set of transactions (known as a block), the first node on the network to solve the cryptographic puzzle gets rewarded in Bitcoin. This is known as cryptographic mining, and is one of the ways the Bitcoin network is kept secure.

Proof of Work is like giving a room full of students a jumbled-up Rubik's Cube each, and requiring them to solve it before they can ask a question. We know that the cube is difficult to solve, yet easy for the lecturer to verify if it's been solved. This has the effect of removing the incentive to ask bad questions, because we know that a student must be serious about the question if they're willing to go to the effort of solving a Rubik's Cube in order to ask it.

The problem with Proof of Work is that it takes an extraordinary amount of electricity in order to function. Estimates vary, but Bitcoin's implementation of Proof of Work is thought to use the equivalent of the power consumption of a country like Ireland.

Blockchain is not Bitcoin. Moreover, the typical requirements of Blockchain *for business* are totally different from the requirements of the Bitcoin blockchain. Specifically, the use of unregulated assets, anonymity, untraceability, and excessive power requirements lead many businesses to struggle to adopt Bitcoin for business-to-business transactions. This has led to the introduction of different blockchain implementations that more closely fit what businesses require, while still being able to achieve cryptographic proof of a set of transactions.

The Requirements of Blockchain for Business

The requirements of blockchain for business actually differ from Bitcoin in five distinct ways: (1) the assets that are tracked, (2) knowing each transaction's participants, (3) the rules around privacy and confidentiality, (4) how transactions are endorsed, and (5) how the network is governed. We'll discuss each of these in the next section.

The Assets That Are Tracked

Blockchain can be used for a much broader range of assets than just cryptocurrency. Tangible assets such as cars, real estate, and food products, as well as intangible assets such as intellectual property, licenses, and shared information sets, are all fair game, so long as they can be represented digitally. Part of the art of setting up a blockchain in a business environment is deciding what to share.

Knowing Each Transaction's Participants

As we've seen, Bitcoin thrives due to anonymity: anyone can look at the Bitcoin ledger and see every transaction that ever happened, but participant information is untraceable. On the other hand, businesses have requirements such as KYC (know your customer) and AML (anti–money laundering). These are examples of compliance rules that require businesses to know exactly who they are dealing with.

The Rules Around Privacy and Confidentiality

Privacy and confidentiality are key requirements of a blockchain for business. Firstly, in the same way that markets can be public or private, it should be possible for a blockchain to be private, meaning that the network can decide exactly who joins. A public blockchain increases the risk of a business inadvertently making transaction or relationship information public knowledge, whether accidentally or through malicious means (e.g., by exploiting vulnerabilities). Furthermore, individual transactions require confidentiality to avoid giving other members an unfair advantage. We probably don't want one supplier on the network knowing the discount level we're giving another supplier, even if they are on the same blockchain. Of course, a regulator might require total visibility of all transactions.

Characteristics such as these give rise to the need for permissioning of the blockchain network, where different participants can do different things. It's possible to have public and permissioned networks such as Stellar, as well as private permissioned networks such as IBM Food Trust built using Hyperledger Fabric.

Permissioned networks are totally different than Bitcoin, which is both public and unpermissioned. Bitcoin reveals to everybody all the transactions that occurred, but not who is involved with them. Businesses need the complete opposite: knowing who they are dealing with but not necessarily aware of the details of every transaction.

These privacy requirements also mean that it is probably infeasible to have a single blockchain instance that covers everything. Just like there are many ledgers in existence today (and there are many business-to-business networks), there will probably be many blockchains in the foreseeable future, albeit with the ability to transfer assets between instances: a network of networks, in other words.

How Transactions Are Endorsed

Consensus in a blockchain for business is not usually achieved through Proof of Work but often through a process of selective endorsement. This means being able to control exactly who receives and endorses transactions, much in the same way that business happens today. If we transfer money to a third party, then our bank, the recipient's bank, and possibly a payments provider would endorse the transaction. These transactions are then validated by those on the network permissioned to receive them. This is different from Bitcoin, where miners compete to endorse transactions in exchange for a bitcoin reward and a network of users running full nodes (those that fully verify all of Bitcoin's rules) collaborate to verify transactions.

How the Network Is Governed

Blockchain networks can be governed in one of two ways: either using a pre-agreed policy, or through a set of tokens. Policy-based approaches require a set of rules agreed upon up front by key stakeholders, such as a consortium of members, a regulator, or a market maker. The rules can be wide reaching and might, for example, describe how consensus is achieved, how future changes to membership are decided, or who is liable for bugs in smart contracts. Some blockchains use tokens in order to govern behavior. The public Ethereum blockchain is an example where your wealth (in terms of your balance of the Ethereum cryptocurrency known as Ether) is used to determine your smart contract processing capacity.

Token-based policy is considered to be *on chain* governance as the ability to govern is locked into the blockchain. Policy-based governance can be both *on chain* and *off chain* depending on the approach.

Both policy- and token-based governance approaches are reflective of established real-world systems. For example, a country's laws are a policy-based governance system, yet your wealth can determine other things you can do within that framework. While early business blockchains have typically been governed through policy, there is an increasing number of business blockchains that have been augmented with token systems as a means to encourage behavior within the network.

Blockchain Technology

For any business blockchain to be successful, participants need to agree on the approach for sharing transactions and smart contracts. In the same way that HTTP(S) is an agreed approach for sharing information over the internet, there needs to be a common standard for blockchains (at least within each business network) for the network to grow and thrive.

There are lots of decisions that need to be made when setting up a network, including asset and transaction data formats, network topology, governance, and validation rules. One of the most important decisions is the choice of blockchain technology: the software that provides the implementation of the shared ledger and smart contract execution framework. Participants of the network need to adopt the same technology to share information; there is no universal interoperability standard for blockchain technologies yet, although work on standardization continues.

The selected blockchain technology must complement the vendor biases and diverse IT landscapes that are typically present in business networks. This means that openness of the blockchain technology is essential, not only in terms of the source but also such that the entire community has the opportunity to influence the direction of the project (known as *open governance*). It does not typically make sense to adopt a proprietary blockchain technology, as it would require all present and future participants on the business network to adopt the same vendor, which raises the risk of lock-in, cost increases, and a lack of scope for innovation.

The Hyperledger Project

In February 2016, the Linux Foundation® formally announced Hyperledger®, an open source, open governance effort to advance cross-industry blockchain. It serves as a greenhouse for multiple blockchain technologies, including frameworks that implement shared, replicated ledgers, and tools for developing and operating instances of them.

Like other Linux Foundation projects, Hyperledger is built around this spirit of openness, and this has helped contribute to its ongoing success. At the time of writing, more than 260 organizations have joined Hyperledger from a broad range of industries and disciplines, and there is a strong developer community that has been contributing to over 10 individual projects under the Hyperledger umbrella.

One of the most advanced Hyperledger projects is called Hyperledger Fabric™. This provides an implementation of the shared ledger and smart contract execution framework, and is built around the principles of security (to reflect the needs of regulated businesses) and modularity (to allow for innovation). It is developed by a worldwide team representing dozens of unique organizations, and there are numerous instances in production. This book will focus primarily on Hyperledger Fabric.

IBM and Blockchain

IBM has been contributing code, intellectual property, and development resource to Hyperledger since its inception.

The IBM Blockchain Platform was the first commercially available platform to leverage technologies from Hyperledger. Platforms such as this provide a set of tools to assist with the development, governance, and operation of Hyperledger Fabric networks, and have been used to underpin many of the blockchain solutions in production today, including IBM Food Trust and TradeLens.

These networks along with those from other providers are discoverable in a public registry of networks called the Unbounded Network Registry.

Summary

In this section we've looked at what blockchain for business is: a shared, distributed, permissioned ledger with smart contracts. Blockchain is important because it helps engender trust in business networks by providing cryptographic proof over a set of transactions. This can remove friction from business networks, for example, by removing the need for costly dispute resolution processes.

Blockchain is not Bitcoin; the requirements of blockchain for business are completely different, and focus on characteristics such as confidentiality and real-world assets.

We have also looked at Hyperledger, hosted by the Linux Foundation, and how it aims to solve these requirements of blockchain for business. We concluded by looking at IBM's contribution to blockchain and how it is helping customers on the path of this exciting technology.

In the next chapter we will look at what makes a good blockchain solution and evaluate some examples of how blockchain has been used to great effect.

CHAPTER 2

Identifying When to Use Blockchain

In this chapter we look at when to use blockchain as a solution. We saw in Chapter 1 that blockchain is a shared, distributed, and permissioned ledger that records transactions across a business network. There are some specific requirements of a scenario that make it particularly suited to using blockchain, and we will explore those in this chapter.

First, we'll identify the types of issues business networks are facing, before looking at the criteria we use to determine whether blockchain makes a good technology fit to solve those issues.

Identifying Issues in the Business Network

It is important to have a clear idea of the scenario's requirements, and to know which issues you are trying to solve. This might seem obvious, but with any new technology—and particularly a well-hyped one—there is often a temptation to jump to an implementation without a lot of thought about the problems it aims to solve.

Businesses act for several reasons. Most commonly, they are going after a new market (for example, opening trade finance opportunities to small/medium enterprises), or trying to remove cost or inefficiencies from a business process (for example, removing intermediaries). Identifying the issues and expressing them in terms of the expected gain (or return on investment) is the first step to a successful project.

Let's return to IBM's Global Financing blockchain solution introduced in Chapter 1. They identified that within this business network of about *4,000 participants* that $100m of capital was in dispute at any one time. This was a significant liability, and was a direct result of the time taken to resolve some *25,000 disputes* annually from the *2.9 million invoices* issued. Examples of disputes include the wrong number of computer parts being delivered in an order or deliveries going awry. It was also noted that each of the 25,000 disputes would take on average *44 days* to resolve, requiring someone to retrace steps through six or seven applications, including contacting third parties such as banks.

We can summarize the issues as:

1. Disputes arising due to consignment issues
2. No single source of trusted information to help resolve issues
3. Disputed transactions taking a long time to solve

In "The Blockchain Fit" on page 21, we'll review these issues and see if blockchain can help to provide a sensible solution to resolve them. Before that, we'll look at the benefits of a blockchain solution.

What Are the Benefits of a Blockchain-Based Solution?

Blockchain is a solution for business networks. It makes sense to deploy a blockchain-based solution only where there is a network of collaborating participants who are issuing transactions around a set of common assets in the network.

Therefore, our first observation of when blockchain is the right solution is that there must be a *business network of multiple participants*. Our second would be that they require a shared view of assets and their associated transactions.

We then use the following four key blockchain features introduced in Chapter 1 to further define the benefits of a blockchain-based solution. Let's remind ourselves of these benefits:

Consensus
: The process of agreeing on new transactions and distributing them to participants in the network.

Provenance
: A complete history of all transactions related to the assets recorded on the blockchain.

Immutability
: Once a transaction has been stored on the blockchain, it cannot be edited, deleted, or have transactions inserted before it.

Finality
: Once a transaction is committed to the blockchain, it is considered "final" and can no longer be "rolledback" or undone.

There are several other blockchain benefits that underpin these four key benefits, and are worth keeping in mind as you review any potential scenarios:

Identity
: All participants in a permissioned blockchain network have an identity in the form of a digital certificate—the same technology that underpins the security and trust when we use a web browser to access our online bank.

Security
: Every transaction in the permissioned network is *cryptographically signed*, which provides *authenticity* of which participant sent it, *nonrepudiation* (meaning they can't deny sending it), and *integrity* (meaning it hasn't been changed since it was sent).

Contracts
: *Smart contracts* hold the business logic for transactions and are executed across the network by the participants endorsing a transaction.

These benefits help engender trust between the participants in business networks, and we can use them as a litmus test when checking to see if blockchain is a good technology fit. We should note that while it's not necessary for a scenario to require every benefit just listed, the more that are required, the more the case is strengthened for using blockchain.

We should always be wary of thinking that blockchain is a panacea for all solutions. There are many reasons why blockchain wouldn't be a good fit. For example:

- Blockchain is not suitable if there's only a single participant in the business network.
- Although we talk about transactions and world state databases in blockchain, it shouldn't be thought of as a replacement for traditional database or transaction servers.
- Blockchain by design is a distributed peer-to-peer network, and is heavily based on cryptography. With this comes a number of nonfunctional requirement considerations. For example, performance and latency won't match a traditional database or transaction server, but scalability, redundancy, and high availability are built in.

Assets, Participants, and Transactions

When thinking about a potential blockchain solution and the benefits it brings to the network of participants, it is useful to view it in relation to the following concepts:

- Assets
- Participants
- Transactions

We have already introduced some examples of these. They are core concepts in a blockchain network that benefit from the four primary trust benefits introduced in the previous section.

Assets

Either purely digital, or backed by a physical object, an *asset* represents something that is recorded on the blockchain. The asset may be shared across the whole network, or can be kept private depending on the requirements. A smart contract defines the asset.

Participants

Participants occupy different levels in a blockchain network. There are those participants who run parts of the network and endorse transactions. Other members may consume services of the network but may rely on and trust other participants to run the network and endorse transactions. Then there are the end users who are interacting with the blockchain network through a user interface. The end user may not even be aware that a blockchain underpins the system.

Transactions

The transactions are coded inside the smart contracts alongside the assets to which the transactions belong. Think of the transactions as the interaction points between the assets and the participants; a participant can create, delete, and update a given asset, assuming they are authorized to do so. It is these transactions that are stored immutably on the blockchain, which also provides the provenance of any changes to the asset over time.

The Blockchain Fit

In an earlier section, we looked at the issues in the IBM Global Finance example that led to the implementation of a blockchain solution along with the benefits that a blockchain-based system can provide. We will now consider why blockchain technology was the sensible choice.

First and foremost is to check there is a business network in place. The IBM Global Finance system comprises some 4,000 suppliers and partners, as well as IBM within the network. So we have a good business network on which to consider the rest of the blockchain features.

As some of the disputes are related to differences between what was ordered and subsequently received, this can often be the result of different participants in a business network (partners, suppliers, and delivery companies) tracking goods in separate siloed systems.

Therefore, a shared ledger with consensus and finality provided by blockchain across the business network will help to reduce the overall number of disputes as it will give all participants the same information on the assets being tracked.

Furthermore, if changes to the data being tracked either intentionally or unintentionally are part of the root cause of these disputes, then the provenance and immutability features of blockchain could also help.

Last, we consider the amount of time taken to resolve these issues. As there were multiple systems (including third-party systems) that someone needed to check in order to resolve any transactions in dispute, having a single shared ledger that is maintained through consensus will help reduce the time taken to resolve them.

Some further observations about how a blockchain-based solution can benefit this business network:

- Each participant in the business network has an identity and is permissioned in the network. This could help with your processes related to Know Your Customer (KYC) and Anti-Money Laundering (AML).
- Smart contracts could be designed to resolve some of the disputes automatically by maintaining consistency across the business network and therefore further reducing the number of disputes.

Choosing a First Scenario

You may be considering multiple scenarios where blockchain provides a good solution fit. In this case you will need to compare each to determine which is the best scenario to work on first.

We recommend a simple approach for comparing each scenario using a quadrant chart, where each is placed on the chart based on its relative benefit and simplicity.

In Figure 2-1 the x-axis is the simplicity of the scenario (simpler to the right) and the y-axis represents the benefit (more beneficial to the top). Place each scenario on the quadrant chart, considering its expected benefit and simplicity as a blockchain solution. This is best done as a group exercise with appropriate stakeholders who can provide the necessary insight to where each scenario falls in the chart based on level of simplicity and potential benefits.

Once all scenarios have been plotted on the chart, it becomes obvious which are the first scenarios to concentrate on—those that will provide the most benefit and are the simplest.

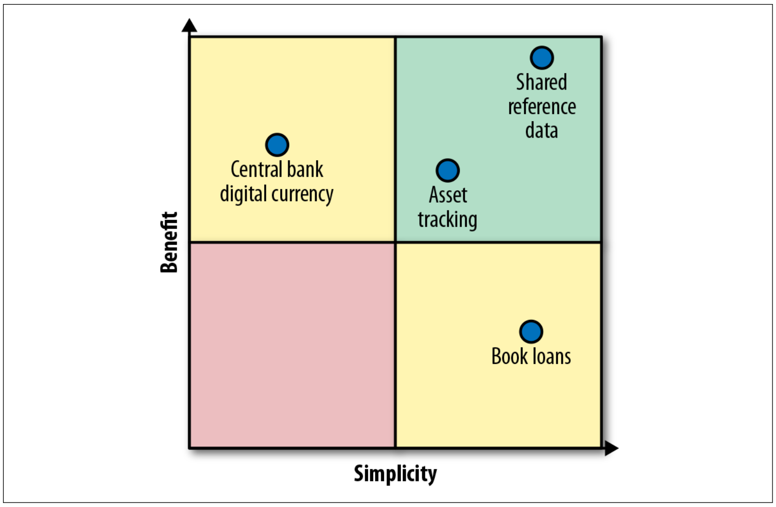

Figure 2-1. Comparing scenarios based on their benefit and simplicity

Transforming the Business Network

Once your first blockchain scenario has been identified, you will want to move to the next phase: building the Minimal Viable Product (MVP). An MVP represents the minimum product that can be built to accomplish a goal of the blockchain scenario.

Starting a MVP with blockchain shouldn't be dissimilar to any other technology, and good software engineering practices, such as using Agile principles, will always be applicable. Following are some observations that will help as you start to transform your business with a new blockchain-based solution:

- Blockchain is a team sport. There will be multiple stakeholders from different organizations in the business network. Some of these organizations may not have traditionally worked directly with one another. Therefore, a clear understanding of the requirements and issues across all participants, and clear lines of communication and agreement, are critical to the success of the project.
- Use design thinking techniques that focus on the goals for the user, to agree on the scope of the MVP.
- Use agile software engineering best practices, such as continuous integration and stakeholder feedback, to iterate throughout

the development of the MVP. Keep stakeholders informed and act on feedback.

- Start with a small network and grow. There will be some challenges ahead, as this may be a paradigm shift for the business network.
- If replacing an existing system, consider running the blockchain-based solution as a *shadow chain* to mitigate risk. By this we mean, during the pilot phase, run the new platform alongside the legacy system. Ideally, you would pass real production data to the new blockchain-based system to test and validate it, while continuing to rely on the legacy system for this phase of the project. Only after thorough testing has been completed and the new system has been proven should you switch from the legacy system to the new.
- Although blockchain is likely to be a core foundational part of the solution, it probably won't be its majority. The blockchain network will still integrate with other external systems, providing additional functions such as off-chain data storage, identity access management, Application Programming Interface (API) management and presentation layers, and so on.

Growing the Business Network

With a blockchain-based solution it is advisable to start with a small network and then grow. What does growing the network mean? It can mean adding any of the following:

- Network participants, i.e. those participants in the network running peers
- Applications interacting with the network
- Users of the blockchain-based solution

It may also mean increasing the following:

- Smart contracts
- Channels or subledgers for privacy between network participants
- Transaction throughput

- Number of assets

There needs to be a strong democratic governance model to manage these types of changes as they are made to the blockchain network. Remember, no single organization is in control.

This is one of the areas that sets a blockchain-based solution apart from other technologies. For example, there are changes within the network that require multiple participants (organizations) within the network to agree before the change can come into effect. The following are two examples of changes that require this type of agreement.

1. An existing network consists of three participants: Dave, Matt, and Luc. John would like to join the network. For John to join the permissioned network, Dave, Matt, and Luc must each cryptographically sign the configuration change before John is allowed to connect to the network. Of course, this is one option of managing this scenario; alternatively, if only one network member is needed to approve such a change, then the network can be configured accordingly.
2. A new smart contract is to be installed on the network, and both Dave and Matt's organizations will be required to endorse transactions for the new smart contract. In this scenario, both Dave and Matt are required to install the smart contract to their peers, and ensure the smart contract is instantiated on a shared channel (or subledger) before a client application.

Ten Questions to Explore the Scenario in More Detail

So far, we have taken a fairly high-level look at whether blockchain makes a sensible choice for the scenario. We can of course explore the scenario in much more detail to help identify the suitability of blockchain, and to start mapping out the type of network that might be involved.

The following questions will help in understanding the scenario from a blockchain perspective:

1. What is the specific business problem or challenge that the scenario will address?
2. What is the current way of solving this business problem?
3. Assuming the business problem is large, what specific aspects of this business problem will be addressed?
4. Who are the business network participants (organizations) involved and what are their roles?
5. Who are the specific people within the organization and what are their job roles?
6. What assets are involved and what is the key information associated with the assets?
7. What are the transactions involved, between whom, and what assets are associated with transactions?
8. What are the main steps in the current workflow, and how are these executed by the business network participants?
9. What is the expected benefit of applying blockchain technology to the business problem for each of the network participants?
10. What legacy systems are involved? What degree of integration with the legacy systems is needed?

Again, working through these questions with the appropriate stakeholders (business, technical, and users) and capturing the results will help enormously before moving the project to the next phase, such as a MVP.

Commercial Paper: An Example Scenario

The Linux Foundation Hyperledger Fabric community has produced an excellent set of materials around the scenario of *Commercial Paper*. For more information on the scenario, you can go here.

We will come back to the commercial paper scenario in later chapters, but we introduce the concepts of a commercial paper business network here, and see why blockchain makes a good technology fit.

Commercial paper is a debt instrument issued by a company that needs to overcome its short-term financing needs. The commercial paper is sold to another company that can redeem the paper at a later date for a higher value than they paid for it. This provides short-term funding to the company issuing the commercial paper, and provides a return on investment for the company that buys the commercial paper. Commercial paper can be resold to other companies during its life cycle.

Earlier in this chapter, we introduced the concepts of assets, participants, and transactions. Let's look at commercial paper through this lens.

Commercial Paper Assets

The main asset in the business network is the commercial paper. This asset will have several attributes, such as:

- The issuing company
- Which company is the current owner
- The issue date
- The maturity date
- The face value
- The current state

Commercial Paper Participants

The main participant in the business network is a company. There will of course be multiple companies in the network, and these companies will play different roles in relation to a commercial paper asset, such as issuers and buyers.

It's clear that within this business network, there are multiple participants. Remember, this is critical for a scenario being a good blockchain fit.

Commercial Paper Transactions

The main transactions associated with the commercial paper asset are:

Issue
    A company issues commercial paper

Buy
: A company buys commercial paper and is therefore the current owner

Redeem
: A company (the current owner) redeems the commercial paper against the original issuer

Lastly, let's look at the four benefits of a blockchain solution and how they might benefit the commercial paper business network.

Consensus

Multiple commercial paper assets will be traded across the network. A single commercial paper asset may move between several different companies, and each company will be trading with multiple other companies. Therefore, it is easy to see how consensus on the state of the asset and any transactions stored in the shared ledger makes sense.

Further benefits could be gained by using a blockchain-based solution in this business network, such as visibility across the network. For example, if a company issues commercial paper for USD $100,000, it is considered okay, but what if they issue 20 of these papers to different companies at the same time? A blockchain-based solution can provide additional information such as how many commercial papers have been issued across the business network, and their total value. This trusted information can then be used to assess the risk of purchasing commercial paper.

Provenance

A company buying existing commercial paper will want to be certain of which company issued it. They will also want to be certain that they are buying the commercial paper from the current owner. For reasons of privacy and confidentiality, the identity of other previous owners of commercial paper may well be hidden from the new purchaser. However, a total number of previous owners of the paper could easily be provided.

Immutability

It's easy to see the importance of this blockchain feature. A company that is buying existing commercial paper from another company wants to ensure it hasn't been modified in any way since it was issued. A company redeeming commercial paper wants to be sure there is no doubt about the validity of the paper.

Finality

Last, finality across the commercial paper network is important. Imagine there's a marketplace or exchange within the business network. A company offers commercial paper to the marketplace. Another company offers to buy the issued commercial paper. All companies involved will want to be certain that the new commercial paper is issued only once across the business network. It wouldn't be good if it were possible for two companies to buy the same commercial paper at the same time.

Summary

In this chapter we have seen the importance of identifying a blockchain scenario's requirements and any issues related to the scenario being considered. We then learned how the four key benefits of blockchain (consensus, provenance, immutability, and finality) help to determine if blockchain is a good technology fit, while ensuring there is always a business network of multiple participants. Using the blockchain concepts (assets, participants, and transactions) will help guide the decision and aid users in understanding how blockchain might be applied to the business network.

CHAPTER 3
Designing a Blockchain Network

In Chapter 2, you've identified that blockchain makes a good technology fit for your scenario. In this chapter, we'll explore some of the aspects of designing a blockchain-based solution.

Governance Model

Broadly speaking there are two models used to initiate a new blockchain network:

Founder led
: A single organization initiates the network. They won't be doing this in isolation, however. It's expected they will have deep industry knowledge and will be working with stakeholders within the respective industry. The founder will define the governance model and policies of the network, and will invite other organizations to join.

Consortium led
: A group of companies initiate the network. For example, a group of banks work together to build a new blockchain-based trade finance platform that they each benefit from using. It's possible this network could remain private to those within the initial consortium, but it's much more likely that once it's built, they will allow other organizations to join the network. It's also possible that ownership and governance will be moved later to a new company. For example, the trade finance platform we.trade

was formed by a consortium of banks and later moved under the ownership of a joint venture.

Regardless of how the network is created, it will require flexible governance policies. The blockchain needs to be adaptable over time, allowing for new types of governance models to be introduced as the network grows. As new organizations join networks, they will of course need to accept the governance models already in place.

Network Members and Consumers

So far, we have been referring to participants or network members in a very broad sense. However, there are different roles and responsibilities for participants in a blockchain-based solution. When designing the network, it's important to identify the role of a participant.

To understand the different types of roles, we need to look at two concepts of a blockchain network:

Network Services
: These are the foundational blockchain network services: the blockchain peer-to-peer network, smart contract execution containers, and security services.

Business Services
: This is the blockchain application built on the underlying network services. They include the smart contract logic, the client-side application logic that connects to the blockchain, and any integration logic with external systems.

With these concepts in mind we can now look at the different roles in the network:

Network Service Provider
: Governs the network and defines network policies.

Network Service Consumer
: A participant on the network running their peers and certificate authorities.

Business Service Provider
: The participant writing the business logic for the platform, including smart contracts, client applications, and integration logic.

Business Service Consumer
: The participant hosting the client-side applications that connect to the blockchain network and the integration logic.

End User
: Connects to the platform through the user interface. They're likely to do this using a mobile, tablet, or web browser, and may be unaware of the blockchain network underpinning it.

It's important to note that an individual participant may have several roles. For example, an organization may be both a network service provider and consumer, as they both define the governance and policies of the network and also actively use the network by running their own peer. Another example is a single technology partner that may write the smart contract and client application logic, as well as provide the IT infrastructure on which the client applications run.

An important characteristic of a good blockchain technology is that it is flexible in its network setup and configuration, and that it can be configured in such a way as to match the requirements from the founders of the network.

Architecture and Design Considerations

This is a very broad topic, and too much to cover in detail in this chapter. However, we will introduce a number of key architectural and design considerations when looking at a blockchain network.

Participant Types

These identify the types of participants in network. For example, in the commercial paper scenario, we identified organizations as being both issuers and buyers.

Network Roles

These are assigned to participants within the network as discussed in the previous section. Using this knowledge, you can start to map out a blockchain network topology.

Assets

These are recorded on the blockchain. For example, in the commercial paper scenario we identified commercial paper as the main asset. This will help identify the smart contracts that will be needed. Assets have three important attributes related to participants in a blockchain network: who issues or creates the asset, who owns the asset, and who destroys the asset. With this information you can add the known assets to the network topology, showing the participants who issue, own, and destroy them.

Transactions

These refer to the assets recorded on the ledger that network participants submit. You can add transactions to the network topology, augmenting the assets added in the previous step.

Endorsements

These are performed by one or more organizations on the network for each transaction submitted. We always recommend a policy of at least two endorsers to maintain trust and prevent any malicious activity. Endorsement policies can be defined both at the smart contract level and for individual states written to the blockchain ledger. For example, in the commercial paper scenario, individual commercial paper assets can define their own endorsement policy (state-based endorsement), or all transactions and state updates can define an endorsement policy applying to all commercial paper (smart contract endorsement). Review the assets and transactions added to the network topology and consider for each which organization should endorse it. You might be surprised what this reveals about the interactions and trust assumptions in the network.

Deployment

There are many ways a blockchain network can be deployed across a heterogeneous environment. For example, a network could be provisioned within a single cloud environment, on premises, or a hybrid of the two. In particular, we're seeing more networks bridging multiple cloud environments. When reviewing deployment, geographical location and any data jurisdiction requirements should also be considered.

Network Access

Will the environments for each of the network components permit access to the URL and ports of the other network components within the network so they can communicate?

Regulations

Regulations such as the European Union's General Data Protection Regulation (GDPR) may influence your design and architecture. For example, the "right to be forgotten" means that you need to avoid storing any personal data on a blockchain if you are later going to be able to comply with a request to delete it. Good design patterns exist for handling personal data on a blockchain; these include, for example, the use of hashes to data stored off the blockchain.

Nonfunctional Requirements

Aspects such as transaction latency, transactions per second, and volume of data should be confirmed as early as possible. These requirements should be reviewed realistically to ensure that blockchain is the correct technology solution.

Security Considerations

Although a permissioned blockchain network such as Hyperledger Fabric is built with security from the ground up, we recommend you start your security planning early. This can easily become the biggest topic in this list.

When reviewing security, these are some of things you will need to consider:

Private Keys
: Where and how private keys will be stored for each of the components in the network. Are Hardware Security Modules a requirement?

Certificate Authorities
: Are there specific certificate authorities that are required to issue certificates for organizations in the network?

Encryption
: What level of Transport Layer Security is required by participants interacting with the blockchain network?

Privacy and Confidentiality
: Is there a requirement for privacy and confidentiality of data, identity, and transactions stored on the blockchain?

Governance, Administration, and Operation Considerations

Earlier in this chapter, we introduced the different roles that participants might have within a blockchain network. Each role defines what the participant can do with regard to administration and operation activities. For example, a network service consumer who is running their own peers and certificate authorities within a Hyperledger Fabric blockchain network will install new smart contracts on their peers. A business service consumer who is running a client application will need to manage their blockchain identities and the application that is connecting to the network.

There are also administrative and operational tasks that need to be coordinated. For example, a smart contract that is deployed by two participants to their peers needs to be designed, coded, and reviewed. After the reviews are complete, each participant installs the smart contract. The design and code are done by the business service provider, and the review and agreement are handled by each of the network service consumers.

To manage these types of changes across the network by different participants, a set of governance policies needs to be defined. These policies could enforce, for example, that all participants digitally sign the smart contract before it can be installed. Thankfully, blockchain networks like Hyperledger Fabric have been designed to allow this. Platforms built on Hyperledger Fabric, such as the IBM Blockchain Platform, can then provide user interfaces to make managing these governance policies simpler.

The advice here is to plan this early. The governance policies will be decided based on the business requirements and possibly the regulatory requirements. Understanding these requirements up front will help when designing the type of blockchain network that will underpin the new system.

Data Considerations

In this section, we look at how data is processed in a blockchain network, such as the location of where it is stored and how it can be made private.

Jurisdiction

A blockchain is a peer-to-peer network. Data in the form of transactions and the actual blockchain structure is shared and replicated across the network to different peers. Because the network can span multiple jurisdictions, it is important to consider what data is stored and where.

Different blockchain technologies allow you to architect different network topologies. For example, the bitcoin network has large number of peers that each ultimately store the same blockchain structure (there are forks in the network that mean at times there might be alternative versions; the consensus algorithm will ultimately resolve this). Other blockchain technologies such as Hyperledger Fabric allow you to architect networks that can manage multiple blockchain structures (known as *channels*) and also specify which peers can store which channel (meaning not all peers have to store everything).

Using Hyperledger Fabric as an example blockchain technology, let's now review the importance of data jurisdiction and the types of choices that might be available.

Hyperledger Fabric has two types of peers based on their role within the network:

Orderer nodes
: These receive transaction proposals, package these transactions into blocks, and deliver them to peers. A group of orderer nodes within the network is referred to as the *ordering service*. There is a single ordering service per network that orders transactions for all channels across the network.

Peers
: These maintain a local copy of a ledger per channel. In Hyperledger Fabric, the ledger comprises both the blockchain structure, which is a list of blocks containing transactions, and also a world state database, which is written to and read from the

smart contract. A peer can join (if permissioned) many channels and thus it could be managing multiple ledgers. Remember that not all peers need to join all channels. An advanced option is for a peer to join channels in different ordering services, creating a network of networks.

We shouldn't forget the client application that submits transactions to the blockchain network, as it will clearly have access to the data as well, either as input to a transaction or from querying content stored on the blockchain.

The physical infrastructure that underpins the collection of orderer nodes and peers can be multisite and span different geographical regions. Organizations that are running orderer nodes and peers can choose cloud providers, each of which will offer a range of locations for deployment. Organizations may also choose to deploy on premises within their own datacenter. As blockchain networks are made up of multiple organizations, each organization can choose the infrastructure on which to run the servers they are responsible for. We call these *multicloud hybrid blockchain networks*.

When considering jurisdictional location of data, we must therefore understand:

- The business requirements of what data is to be submitted to and stored by the network.
- The network topology, consisting of orderer nodes, peers, and channels.
- The physical location of orderer nodes and peers that are accessing the data.

Data Privacy

Let's now see how to make data private and confidential across a blockchain network. Remember that all participants in a permissioned blockchain network such as Hyperledger Fabric have a cryptographic identity in the form of a digital certificate. This includes any orderer nodes and peers. Remember also that transactions are submitted to the network and data is ultimately stored in the channels that contains both the blockchain structure and the world state database.

When considering data privacy, we must first understand which data to make private and to which organizations and participants it should be private.

The types of data that can be shared across the network, and therefore should be considered when looking at privacy, are:

- The digital certificate (identity) of the transaction submitter
- The digital certificate (identity) of the peers that endorse transactions
- The transaction called by the submitting application and any parameters passed to the transaction
- Any data read and written by the smart contract to the world state
- The response to the client application from calling the smart contract

In addition to the preceding, there are a couple of other small pieces of metadata associated with a transaction invocation: for example, the key names of the data stored in the world state, and the names of any private data collections from which the smart contract has read or to which it has written.

Hyperledger Fabric provides several features to manage data, and to make it private from certain participants on the network. We'll summarize each of these next:

Channels

Channels are a way of making a ledger (blockchain and world state) private to only those organizations allowed to join the channel. In addition to the ordering service, only peers and client applications within the permissioned organizations can join and have access to the channel. Any not permissioned to join a channel will never receive blocks from the ordering service and therefore will never have a copy of the ledger for the channel. Channels provide a really nice way of creating subnetworks across the blockchain network. One thing to be aware of is that the ordering service will still see the transactions for all channels, and any organizations permissioned to join the channel will have access to all data held in the channel. This may be ideal for some scenarios but less so for others.

Private Data Collections

Private Data Collections first appeared in Hyperledger Fabric version 1.2. These work within a channel, and allow policies to define that only a subgroup of organizations within the channel can access certain data. Unlike normal transactions, where all the information is sent to the ordering service, for Private Data Collections the data is stored only by the peers specified in the policy (and shared via the gossip protocol), and hashes rather than actual data are sent in the transaction to the ordering service. This means that data can be kept private to a subset of organizations within the channel and never shared with the ordering service.

Encryption

Encryption is the process of converting readable source text into unreadable cipher text. Commonly this is done with a pair of asymmetric keys, where a holder of a public key can encrypt a piece of data knowing that only the holder of the private key can decrypt it. It is a common misconception that data is automatically encrypted in Hyperledger Fabric and some other blockchain technologies. However, Hyperledger Fabric does provide a set of encryption and decryption services to the smart contract, enabling it to decrypt data it receives from either the calling client application or by querying the world state, and also to encrypt any data it writes to the world state or passes back to the calling client application. Encryption can be used along with channels and private data collections as just described. Of course, it's possible for the submitting client application to encrypt data before sending it to the smart contract. One thing to keep in mind is that you will need to manage the keys (both public and private) when using this feature.

Identity

While it's imperative that everyone in a permissioned network have an identity (for the purposes of authentication, integrity, and nonrepudiation), it doesn't mean that there are times when a participant's identity should not be hidden from a larger audience. For example, say there is a channel of 10 organizations who are competitors to each other. Each organization submits transactions to the blockchain network for this channel, but doesn't want to reveal their identity in the transaction stored on the blockchain. Using the Identity Mixer feature added in Hyperledger Fabric version 1.3, it is possible to keep the identity of the submitting client application anonymous, but without losing the ability to authenticate them. Additional roles will be added in the future whose identity can also be made anonymous.

Onboarding

When it comes to onboarding new organizations in a blockchain network such as Hyperledger Fabric, you will need to consider much of what has been discussed in this chapter:

- Know and understand the governance model for the network. Are the policies defined by the founder or the existing consortium suitable?
- Understand the type of role the new organization will have in the network. Does the role meet their requirements, and are there any trust assumptions that need to be reviewed and approved?
- Know whether the new organization will run their own peers, and if they will endorse transactions submitted to the network. Is there a choice of running in a cloud or on premises?
- What policies are in place with regard to changes to the network and new smart contracts?
- What do the policies say for additional organizations being onboarded, and will newly onboarded organizations have a say in any future organizations being onboarded?
- Are there any security requirements that must be adhered to, such as storing private keys in Hardware Security Modules?
- Are there any regulatory requirements such as GDPR or data privacy requirements?

Summary

In this chapter, we have reviewed many of the design considerations when building a new blockchain network. We looked at the types of roles organizations can take in a network and the need for governance. We looked at areas such as security and regulation, as well as reviewing data jurisdiction and privacy requirements.

CHAPTER 4

Developing a Blockchain Network

The previous chapters introduced blockchain concepts and helped you to identify when a scenario would be ideally solved by a blockchain solution. We also reviewed aspects of designing a blockchain network.

In this chapter, we look at the different aspects of developing the blockchain application using Hyperledger Fabric.

Smart Contracts

The first thing we will look at are smart contracts, which are a core part of developing a blockchain application.

Hyperledger Fabric has taken a generic approach to the languages used to write smart contracts. The first supported language was Go, which is also the language that Fabric itself is written in. Node.js and Java have been added in later versions. Hyperledger Burrow™ has also been added to Hyperledger Fabric, which adds languages such as Ethereum's Solidity.

Although the industry uses and understands the generic term *smart contract*, Hyperledger Fabric actually refers to the installable unit as *chaincode*. We can think of smart contracts as being equivalent to chaincode.

In Hyperledger Fabric v1.4 a number of significant API changes were made. These improvements make it simpler to write smart contracts, allowing multiple associated contracts to be written in a single chaincode. This provides a one-to-many relationship between a chaincode and the smart contract(s) inside it, and gives the developer more flexibility to colocate smart contracts that are related. Other API changes were made to the Fabric Software Development Kit (SDK), making it simpler to write the client application that connects to the blockchain network. For the remainder of this chapter we'll just use the term *smart contract* and not refer to chaincode.

Each smart contract written in Hyperledger Fabric defines what state is written to the world state database (we'll see more on this in "World State" on page 46), the transactions associated with the smart contract, and the business logic defined within each transaction.

At the top of each smart contract, an important interface called `fabric-contract-api` must be included that defines the overall structure a smart contract must adhere to.

Once a smart contract has been written, reviewed, and tested, it is then packaged into a chaincode deployment spec file that includes:

- The smart contract (which could consist of multiple files)
- An instantiation policy that describes which organization admins can instantiate the smart contract
- A set of digital signatures from the organizations that own the smart contract

Each organization that needs to run the smart contract to endorse transactions will then install it to each of their peers. After this, the smart contract is instantiated on a specific channel by one of the organizations identified in the instantiation policy. Once the smart contract has been instantiated on the channel, each peer that has joined the channel and has previously installed the smart contract will be able to endorse transactions for the smart contract.

In the next section, after introducing the channel ledger, we will see exactly what smart contract code looks like.

Channel Ledger

As we mentioned in previous chapters, each channel in Hyperledger Fabric has a ledger. The ledger consists of both a blockchain structure and the world state.

Let's review what is recorded in both of these structures, and how this might affect what a developer writes.

Blockchain Structure

The blockchain structure is the cryptographically linked blocks, with each block containing one or more transactions. Hashes are used to link the blocks together. Once a block has been added to the blockchain, it can't be modified or removed.

Every time a transaction is submitted to a Hyperledger Fabric ordering service, it will be written to the blockchain structure for the channel on which it was submitted. As we saw in the previous chapter when looking at data privacy, the transaction includes several pieces of information, which we include again here for information:

- The digital certificate (identity) of the transaction submitter
- The digital certificate (identity) of the peers that endorse transactions
- The transaction called by the submitting application and any parameters passed to the transaction
- Any data read and written by the smart contract to the world state
- The response to the client application from calling the smart contract

Transaction validation actually happens at the last phase of consensus within Hyperledger Fabric. All transactions submitted are added to the blockchain, but only those transactions that are deemed to be valid will also modify the world state. If a transaction is considered to be invalid (for example, it doesn't have sufficient endorsements, or there is a mismatch of the world state between endorsement and validation—also known as an *MVCC conflict*), then it is still written to the blockchain but is flagged as invalid. This is very important, because it means that Hyperledger Fabric is not only recording transactions that affect the world state and the assets recorded there, but is also recording invalid transactions for audit.

World State

The world state is a database in which peers maintain state written by the instantiated smart contracts. The world state can currently be either of the following:

LevelDB
: The default option, in which an instance of LevelDB is embedded inside the peer process. This database allows the smart contract to write simple key-value pairs. Queries can be done on the keys, but there is no support for complex queries on the value.

CouchDB
: An alternative option, in which case each peer will connect to its own external CouchDB server. Data is still recorded as key-value pairs, but CouchDB allows for additional query support of the value when it is modeled as JSON.

Smart contracts decide which data to read and write from the world state using `getState()` and `putState()` functions provided by the Fabric smart contract API.

It's possible for a smart contract to not use the world state at all, although this is very unlikely. It's worth also noting that the world state is derived entirely from the transactions recorded in the blockchain on the ledger. Remember that each transaction contains the read and write set for the world state. This means that if a peer's world state is ever corrupted or lost, then it can be entirely recreated by running through the transactions in the blockchain. In fact, this is exactly what happens when a new peer joins a channel for the first time. It will receive all the blocks in the blockchain from the ordering service (and possibly other peers) and construct its world state based on this.

Here is a very simple `HelloWorld` smart contract that illustrates the Hyperledger Fabric smart contract interface and world state operations.

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

class HelloWorld extends Contract {

  async instantiate(ctx) {
    console.info('Writing to world state');
    await ctx.stub.putState('hw', 'Hello World');
  }

  async query(ctx) {
    console.info('Reading from world state');
    const value = await ctx.stub.getState('hw');
    console.info(value.toString());
  }
}

module.exports = HelloWorld;
```

The smart contract is called HelloWorld. It implements two transactions:

`instantiate(ctx)`
: This transaction is called once when the smart contract is instantiated on the channel.

`query(ctx)`
: This transaction can be called after being instantiated.

The transaction instantiation writes a simple key-value pair (`hw`, `HelloWorld`) to the world state using `putState()`. The query transaction retrieves the value for key *hw* from the world state using `getState()` and writes the value to the console.

In the next chapter, we will explore more complex smart contracts that implement commercial paper.

Client Application

For every smart contract there will be one or many client applications that call transactions implemented by the smart contract. The client application communicates with the Fabric blockchain network using the Fabric SDK. Both Node.js and Java are supported, and additional languages are being developed.

The Fabric SDK allows the application to connect to peers and orderers. Once connected the application can query the blockchain and can also call transactions implemented by smart contracts.

If using the Node.js SDK, the client application should use the Hyperledger Fabric `fabric-network` module to gain access to the API for submitting and querying transactions.

The client application needs two key pieces of information in order to connect to a peer using the API:

Connection profile
: Provides the basic connection details of the peer and ordering service, such as host and port numbers.

Credentials
: A wallet represents the user submitting the transaction. This includes the user's private key and digital certificate.

Let's look at these in more detail.

Connection Profile

A connection profile does not need to include the whole network, only the basic connection details required for the specific client application. The SDK can use Fabric's service discovery feature (if enabled) to discover further information about the network, channels, and smart contracts once a basic connection is made.

This is an excerpt from a connection profile showing the connection details for the orderer, peer, and certificate authority in our network:

```
"orderers": {
  "orderer.example.com": {
  "url": "grpc://localhost:17050"
  }
},
"peers": {
  "peer0.org1.example.com": {
  "url": "grpc://localhost:17051",
  "eventUrl": "grpc://localhost:17053"
  }
},
"certificateAuthorities": {
  "ca.org1.example.com": {
  "url": "http://localhost:17054",
  "caName": "ca.org1.example.com"
  }
}
```

For each component the hostname, as well as the main URL, is provided. Peers have two ports, one for receiving transaction proposals and one to which the client application can connect for registering and receiving event messages. Each certificate authority has a `caName` property.

Credentials

Each component on the network, administrator, and user has a digital identity that is their credential. The digital identity comprises a private key (used to digitally sign transactions) and a digital certificate (public identity).

A blockchain network for business is made up of multiple organizations, and each organization will manage its digital identities for any network components it has (peers and orderers) and any administrators or users. These digital identities are issued from a Certificate Authority associated with the organization. Hyperledger Fabric includes a component called a *Membership Services Provider (MSP)* that provides an abstraction layer for the organization to issue, validate, authenticate, and revoke digital identities.

Hyperledger Fabric has a two-step process for the issuance of new identities: *registration* and *enrollment*, summarized as follows:

Registration
: An administrator registers a new user with a certificate authority within the organization. The result of registering a new user is an *enrollment ID*, and a *secret* (password). No digital identity is issued during registration. The administrator sends the certificate authority connection details along with the enrollment ID and secret to the user.

Enrollment
: The user enrolls with the certificate authority using the *connection details*, enrollment ID, and secret provided by the administrator. Enrollment creates public/private keys locally before calling the certificate authority to issue a digital certificate based on the generated keys. This process uses what is known as a Certificate Signing Request when calling the certificate authority. The result is an X.509 public key certificate. These certificates can then be validated at any point, by checking them against the certificate authority's public certificate, which is also known to the whole network.

This two-step process (register and enrollment) is essential for ensuring the private key is only ever known to the user.

Once the digital identity (private key and digital certificate) are created, these can then be stored in a wallet (directory structure defined by MSP) that is known to Fabric's SDK.

A separate Fabric SDK class called FabricCAClient exists for managing digital identities and connecting to the certificate authority. Following is an example of an application that uses this class to enroll a new administrator's digital identity.

```
// Create a new CA client for interacting with the CA.
const caURL = ccp.certificateAuthorities[caName].url;
const ca = new FabricCAServices(caURL);

// Create a new file system based wallet for managing identities.
const walletPath = path.join(process.cwd(), 'wallet');
const wallet = new FileSystemWallet(walletPath);
console.log(`Wallet path: ${walletPath}`);

// Check to see if we've already enrolled the admin.
const adminExists = await wallet.exists(appAdmin);
if (adminExists) {
    return;
}

// Enroll the admin, and import new identity into wallet.
const enrollment = await ca.enroll({ enrollmentID: appAdmin,
    enrollmentSecret: appAdminSecret });
const identity = X509WalletMixin.createIdentity(orgMSPID,
                                enrollment.certificate,
                                enrollment.key.toBytes());
wallet.import(appAdmin, identity);
```

In the preceding code, we can see how the certificate authority details and the location of the wallet are defined, before enrolling the new administrator by calling `ca.enroll()`.

Full details of this application can be found here. Hyperledger Fabric also includes a command for managing digital identities called `fabric-ca-client`.

SDK

The SDK provides the following three important classes the client application uses to communicate with the blockchain network:

`gateway`
: Provides the connection point for an application to access the Fabric network

`network`
: Represents a channel that is accessible via the gateway

`contract`
  Represents a smart contract instantiated on the network

Following is some sample code showing these three classes in use by the client application. This code connects to our local network and calls the `query` transaction in my `HelloWorld` smart contract seen earlier:

```
// Create a new gateway for connecting to our peer node.
const gateway = new Gateway();
await gateway.connect(ccp, { wallet,
                             identity: 'Admin@org1.example.com',
                             discovery: { enabled: false } });

// Get the network (channel) our contract is deployed to.
const network = await gateway.getNetwork('mychannel');

// Get the contract from the network.
const contract = network.getContract('HelloWorld');

// Evaluate the specified transaction.
const result = await contract.evaluateTransaction('query');
```

In the preceding code, it is the `gateway.connect` statement that does all the work in establishing the connection to the Fabric network using the user's digital identity. Once connected, the application decides which channel and smart contract it needs to reference for sending a transaction.

Code, Debug

As we have seen, Hyperledger Fabric allows you to develop both your smart contracts and client applications using one of several popular languages. It is likely that your IDE (Integrated Development Environment) of choice will support one of these. Both Atom and Visual Studio Code are popular open source IDEs that support plug-ins for these languages.

Plug-ins also exist that specifically help simplify the development and testing of smart contracts for different blockchain networks. For example, there are plug-ins for developing Solidity smart contracts on the Ethereum network and for developing Fabric smart contracts deployed to the IBM Blockchain Platform. This latter plug-in enables the developer to build, test, and debug their Fabric smart contract within the Visual Studio Code environment before then connecting to a remote network to continue testing. The plug-in creates a Fabric test network locally, which makes it very quick to install and test changes to smart contracts.

Smart Contract Features

Hyperledger Fabric is continuing to evolve and mature at a rapid pace. Many additional features have been added to allow the developer to write ever more elaborate smart contracts. In this section, we will cover some of these features at a high level:

- Private Data, which we have described already, allows data to be shared to only those organizations defined within the collection. The smart contract developer will use different APIs to access the world state for any private data (`getPrivateData()` and `putPrivateData()`).
- State-based endorsement, which means that smart contracts can change the endorsement policy for specific entries in the world state, overriding the higher-level smart contract endorsement policy. The developer will use the following APIs to check and set state-based endorsement: `GetStateValidationParameter()` and `SetStateValidationParameter()`.
- An encryption library is available to the smart contract developer that enables them to encrypt and decrypt any data.
- Data is stored in the world state within a namespace associated with the smart contract. This means that one smart contract cannot directly access the world state of another (unless they are written in the same chaincode). Therefore, Fabric includes APIs that enable the first smart contract to invoke another.
- It is possible for a smart contract to emit custom events using `setEvent()`, which is then consumed by the client application if the transaction is validated.
- The clients' digital certificate can be queried for roles and organization affiliations so that rules within the smart contract can be applied.

This list summarizes some of the many options available to the smart contract.

Tutorials and Patterns

There are a number of excellent tutorials and patterns available for both Hyperledger Fabric and the IBM Blockchain Platform. Two very good resources are the Hyperledger Fabric documentation, and the IBM Developer code patterns for blockchain.

The tutorials show how to develop smart contracts and client-side applications, with the patterns showing more complex scenarios and how blockchain can work with other systems within a system architecture.

Summary

In this chapter, we introduced both the smart contract and client application, which together form the basis of a blockchain application. We also looked at the structure of the ledger within Hyperledger Fabric, and the importance of knowing what is recorded in which part of the ledger, the blockchain or the world state. Finally, we reviewed IDEs such as Visual Studio Code, which help the developer to build and test their applications and smart contracts.

CHAPTER 5
A Blockchain Example: Commercial Paper

In Chapter 2, we referenced the Commercial Paper scenario provided as part of the Hyperledger Fabric samples repository. In this chapter, we will examine this scenario in depth as a tutorial to better understand what it is doing. We will then extend the tutorial to create a new transaction in the smart contract and develop a new command line application to invoke it.

What Is Commercial Paper?

The Commercial Paper tutorial simulates a simple commercial paper trading network called PaperNet. Commercial paper is a type of unsecured lending called a *promissory note*. Paper is normally issued by large corporations to raise funds to meet short-term financial obligations at a fixed rate of interest. Once issued at a fixed price for a fixed term, another company or bank will purchase commercial paper at a cost lower than its face value, and when the term is up, it will be redeemed for its face value.

As an example, if paper was issued at a face value of $10 million for a six-month term at 2% interest, then it could be bought for $9.8 million ($10 million – 2%) by another company or bank happy to bear the risk that the issuer will not default. Once the term is up, then the paper could be redeemed or sold back to the issuer for its full face value of $10 million. Between buying and redemption, the paper can be bought or sold multiple times between different parties on a commercial paper market.

Understanding the Commercial Paper Tutorial

Previously, we looked at how we can use the idea of assets, participants, and transactions to analyze blockchain use cases.

From this description, we can see that the main asset here is the commercial paper itself, which will have multiple attributes such as the issuing company, the face value of the paper, the redemption date, and the current state (such as *issued*, *trading*, and *redeemed*) of the paper.

We can also see that there are multiple participants in this scenario: an issuer who will be responsible for creating or issuing commercial paper, and one or more buyers of commercial paper who will own the paper until it is either redeemed or sold on to another party.

Finally, we have the transactions: they are *issue*, *buy* and *redeem*, to issue new commercial paper, trade it, and redeem the face value of the paper with the issuer, respectively.

In the tutorial there are two key participants: MagnetoCorp and DigiBank. The tutorial shows MagnetoCorp initially acting as an issuer, with DigiBank taking on the role of the buyer and redeemer of the commercial paper in the PaperNet network.

Figure 5-1 shows an overview of the network, with Isabella working for MagnetoCorp, the issuer, Balaji working for an example trader (DigiBank), and the two organizations communicating over the PaperNet blockchain:

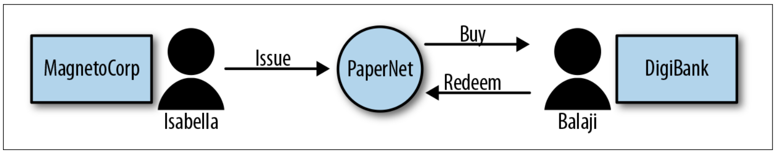

Figure 5-1. Sample network for the Commercial Paper tutorial

Running the Commercial Paper Tutorial

Hyperledger Fabric contains a tutorial for Commercial Paper. In this section, we introduce the purpose of this tutorial and the main tasks involved.

A more detailed description of the Commercial Paper tutorial is available online; if you are a developer looking for hands-on experience of Hyperledger Fabric, then we recommend that you follow this. At the time of writing, the tutorial is available here.

Initially, the tutorial takes you through installing some prerequisite software and downloading the samples. Once this is done you take on the role of Isabella from MagnetoCorp, who will create the network that DigiBank will join. After installing and instantiating the smart contract, she will run a command-line application that will invoke the smart contract's *issue* transaction.

Next, you switch to the role of Balaji from DigiBank, who will use different command-line applications to *buy* the paper that Isabella issued and then *redeem* it with another application.

The tutorial uses a command-line application for each smart contract transaction. Although this makes it very easy to see what is going on from the tutorial's point of view, in a more realistic scenario, one application might be expected to call several smart contract transactions.

At the end of the tutorial, one commercial paper asset has been issued, bought, and redeemed in the network.

Extending the Commercial Paper Tutorial

Although the tutorial shows many aspects of smart contract development and transaction invocation, it currently does not provide a way to query for particular paper. Therefore, in this section we will look at extending the commercial paper smart contract to add a new `getPaper` transaction that will return a string representation of the requested paper to the caller. We will then create a new command-line application based on one of the existing ones in order to invoke it.

This will provide us with a way of looking at the different states commercial paper will go through during its lifecycle. We will then issue new paper and follow it through its lifecycle, querying it at each step along the way.

Writing the New getPaper Transaction

The new `getPaper` transaction follows the same model as the *buy* and *redeem* transactions, but it is much simpler as we do not need to update any properties of the paper in the world state:

```
/**
* Get commercial paper
* @param {Context} ctx the transaction context
* @param {String} issuer commercial paper issuer
* @param {Integer} num paper number for this issuer
*/
async getPaper(ctx, issuer, num) {
 try {
  console.log("getPaper for: " + issuer + " " + num);
  let paperKey = CommercialPaper.makeKey([issuer, num]);
  let paper = await ctx.paperList.getPaper(paperKey);
  return paper.toBuffer();
 } catch (e) {
  throw new Error('Paper does not exist' + issuer + num);
 }
}
```

Add this code to the *commercial-paper/organization/magnetocorp/contract/lib/papercontract.js* file and save it. It needs to sit as a top-level transaction like the other transactions, so place it after the *redeem* transaction almost at the bottom of the file.

We'll now walk through the changes we made.

After the comments, there is the definition of the `getPaper` transaction, which takes three parameters. The first one (`ctx`) is of type `Context` and is the first parameter passed to all transactions. This allows the framework to pass extra information into the transaction function when it is called. For example, it can pass information about the identity of the caller of the contract as well as methods to query the worldstate when the transaction is called. The second and third parameters (`issuer` and `num`) are passed in from the calling application and contain the issuer and number of the paper we wish to retrieve.

After logging the parameters passed in to the console log, the transaction calls the static `CommercialPaper.makeKey` method. This method is defined in the *ledger-api/state.js* file and is a helper method to create a world state key for the paper. Remember from the last chapter that data in the world state is stored as key/value pairs, and for the commercial paper tutorial the key is defined using this method. In this example, the key is simply a concatenation of the issuer and the paper number, separated by a colon; for example, `MagnetoCorp:00001`.

Next, the transaction uses the `paperKey` return value to request a specific paper by calling `ctx.paperList.getPaper` and passing in the key (`paperKey`). As you can see, this is using the `ctx` parameter to access the `paperList` and call its `getPaper` method. The definition of this method is provided in the *lib/paperlist.js* file.

In turn, `paperList.getPaper` simply calls the `getState` method defined in *ledger-api/stateList.js*. It is this method that actually accesses the world state to retrieve the requested commercial paper asset.

Once we have the requested paper, we simply return it to our caller as a buffer object.

If the requested commercial paper does not exist in the world state, an exception will be thrown, informing our caller that the requested paper does not exist.

Writing the New getPaper.js Application

The `getPaper.js` application is based on the existing `buy.js` application that comes with the Commercial Paper tutorial. We can't show you all the code here as there are over 100 lines of code. However, most of these lines remain the same—we are only going to change about 12 of them.

To get started, make a copy of the *commercial-paper/organization/digibank/application/buy.js* file and call the copy `getPaper.js`. Make sure the copy is in the same folder as `buy.js`.

Open the file in a code editor such as VSCode and look through the file for the `console.log` lines that contain the string "Buy program;" for example:

```
console.log('Buy program complete.');
...
console.log('Buy program exception.');
```

To avoid confusion, change `Buy` to `GetPaper` throughout, so that when we subsequently review these console logs we see the correct application's output.

Next, delete the `buy commercial paper` and `process response` sections in the `main` method, as these are not relevant to the `getPaper` application:

```
    // buy commercial paper
    console.log('Submit commercial paper buy transaction.');

    const buyResponse = await contract.submitTransaction('buy',
      'MagnetoCorp', '00001', 'MagnetoCorp', 'DigiBank',
      '4900000', '2020-05-31');

    // process response
    console.log('Process buy transaction response.');

    let paper = CommercialPaper.fromBuffer(buyResponse);
    console.log(`${paper.issuer} commercial paper :
      ${paper.paperNumber} successfully purchased by
      ${paper.owner}`);
```

In place of this deleted block, just before the `Transaction Complete` log message, add the following code. (Note that you will be adding in a few more lines than you deleted, but that's okay—we are doing a little more work to nicely format the returned paper object. Make sure you save the file when you are done.)

```
// get commercial paper
console.log('Evaluate getPaper transaction.');
const getPaperResponse = await contract.evaluateTransaction(
    'getPaper', 'MagnetoCorp', '00001');
console.log('Process getPaper transaction response.');

let paper = CommercialPaper.fromBuffer(getPaperResponse);
let paperState = "Unknown";
if (paper.isIssued()) {
 paperState = "ISSUED";
} else if (paper.isTrading()) {
 paperState = "TRADING";
} else if (paper.isRedeemed()) {
 paperState = "REDEEMED";
}

console.log(` +--------- Paper Retrieved ---------+ `);
console.log(` | Paper number: "${paper.paperNumber}"`);
console.log(` | Paper is owned by: "${paper.owner}"`);
console.log(` | Paper is currently: "${paperState}"`);
console.log(` | Paper face value: "${paper.faceValue}"`);
console.log(` | Paper is issued by: "${paper.issuer}"`);
console.log(` | Paper issue on: "${paper.issueDateTime}"`);
console.log(` | Paper matures: "${paper.maturityDateTime}"`);
console.log(` +-----------------------------------+ `);
```

Looking at this new code that we inserted, we can see that it starts by logging the call it is about to make on the contract to the console. It then calls the `evaluateTransaction` method of the contract to submit the call to the peer to run the `getPaper` transaction.

Note that in `getPaper` we are using `evaluateTransaction` rather than the `submitTransaction` method that the other transactions use; the difference is that `evaluateTransaction` does not record the transaction on the ledger, and as we are not changing any state when returning the paper—this is okay.

The parameters passed to `evaluateTransaction` indicate that we want to get MagnetoCorp's paper 00001, which was the one issued when you ran the tutorial. Once the paper has been returned, we find out which state the paper is in (`ISSUED`, `TRADING`, or `REDEEMED`) and then print this out along with the other information contained within the paper such as its paper number, issuer, owner, and face value.

Upgrading the Smart Contract

Now that we have edited the smart contract to add the new transaction and created the new `getPaper` command line application, it is time to test them both out. To do this, we first have to install the modified smart contract on to the peer. From the `magnetocorp` console window, which you should have open from running the tutorial, issue this command to install the new version 2 of the contract:

```
(magnetocorp admin)$ docker exec cliMagnetoCorp peer chaincode
install -n papercontract -v 2 -p /opt/gopath/src/github.com/co
ntract -l node
```

When it completes, you should see output like this:

```
2019-02-21 13:32:23.824 UTC [chaincodeCmd] checkChaincodeCmdPa
rams -> INFO 001 Using default escc
2019-02-21 13:32:23.824 UTC [chaincodeCmd] checkChaincodeCmdPa
rams -> INFO 002 Using default vscc
2019-02-21 13:32:23.832 UTC [chaincodeCmd] install -> INFO 003
Installed remotely response:<status:200 payload:"OK" >
```

Next, we have to upgrade the version 2 smart contract on the peer to make it live:

```
(magnetocorp admin)$ docker exec cliMagnetoCorp peer chaincode
upgrade -n papercontract -v 2 -l node -c '{"Args":["org.papern
et.commercialpaper:instantiate"]}' -C mychannel -P "AND ('Org1
MSP.member')"
```

When it completes (which can take a few minutes), you should see output like this:

```
2019-02-21 13:32:35.508 UTC [chaincodeCmd] InitCmdFactory ->
INFO 001 Retrieved channel (mychannel) orderer endpoint:
orderer.example.com:7050
2019-02-21 13:32:35.511 UTC [chaincodeCmd] checkChaincodeCmdPar
ams -> INFO 002 Using default escc
2019-02-21 13:32:35.511 UTC [chaincodeCmd] checkChaincodeCmdPar
ams -> INFO 003 Using default vscc
```

Invoking the New getPaper.js Application

Now we are ready to invoke the `getPaper.js` application and test out the new smart contract transaction. First, check that the console window for Balaji from DigiBank, which you should have open from running the tutorial, is currently in the *commercial-paper/organization/digibank/application* directory. If it is not, just use `cd` to change to this directory. This is where the new `getPaper.js` application is located.

Issue this command to run the application:

```
(balaji)$ node getPaper.js
```

You should expect to see output like this:

```
Connect to Fabric gateway.
Use network channel: mychannel.
Use org.papernet.commercialpaper smart contract.
Submit commercial paper getPaper transaction.
Process getPaper transaction response.

 +--------- Paper Retrieved ---------+
 | Paper number: "00001"
 | Paper is owned by: "MagnetoCorp"
 | Paper is currently: "REDEEMED"
 | Paper face value: "5000000"
 | Paper is issued by: "MagnetoCorp"
 | Paper issue on: "2020-05-31"
 | Paper matures on: "2020-11-20"
 +-----------------------------------+

Transaction complete.
Disconnect from Fabric gateway.
GetPaper program complete.
```

Assuming that you have completed the tutorial successfully earlier, you should now see the final `REDEEMED` state of paper number 00001 that MagnetoCorp issued after it was bought and then redeemed by DigiBank.

Testing with New Paper

Now let's go through this one more time, issuing, buying, and redeeming a second paper (number 00002). But this time, we will look at the output from `getPaper` each step along the way.

To do this we will need to edit all four programs to use the new paper. We'll start with one of the easiest: `getPaper.js`. Open up `getPaper.js` in your editor and find this line of code:

```
const getPaperResponse = await contract.evaluateTransaction(
  'getPaper', 'MargetoCorp', '00001');
```

Change the paper number from *`00001`* to *`00002`* and save the change to the file. Although you could run `getPaper.js` at this point, it would return an error as there is currently no paper with the number 00002. So, let's create one.

Open up `issue.js` from the *organization/magnetocorp/application* directory and find this line:

```
const issueResponse = await contract.submitTransaction(
  'issue', 'MagnetoCorp','00001','2020-05-31',
  '2020-11-20','5000000');
```

Change the paper number from `00001` to `00002`, and so we can see more differences, change the amount from `5000000` to `6000000`. You can change the dates as well as if you wish. Your changed line should look something like this:

```
const issueResponse = await contract.submitTransaction('issue',
  'MagnetoCorp','00002','2019-06-30','2019-12-30','6000000');
```

From the MagnetoCorp console window, issue this command to issue paper 00002:

```
(magnetocorp admin)$ node issue.js
```

You should see output indicating that the paper was issued successfully, with output of the form:

```
MagnetoCorp commercial paper : 00002 successfully issued for
value 6000000
```

Next, from Balaji's console window run the `getPaper` application:

```
(balaji)$ node getPaper.js
```

The output should contain:

```
+--------- Paper Retrieved ---------+
 | Paper number: "00002"
 | Paper is owned by: "MagnetoCorp"
 | Paper is currently: "ISSUED"
 | Paper face value: "6000000"
 | Paper is issued by: "MagnetoCorp"
 | Paper issue on: "2019-06-30"
 | Paper matures on: "2019-12-30"
 +-----------------------------------+
```

Now we can see the paper has now been issued, but it is still owned by MagnetoCorp as it has not yet been sold. Let's now buy this new paper as DigiBank. Open up `buy.js` from the *organization/digibank/application* folder and find this line:

```
const buyResponse = await contract.submitTransaction(
'buy', 'MagnetoCorp', '00001', 'MagnetoCorp', 'DigiBank',
'4900000', '2020-05-31');
```

If we look at the `buy` transaction in the *papercontract.js* smart contract, we can see the parameters representing the transaction to call (`buy`), the issuer, the paper number, the current owner, the new owner, the price, and the purchase date and time. Edit the line to update the parameters for the new paper to look like this:

```
const buyResponse = await contract.submitTransaction(
'buy', 'MagnetoCorp', '00002', 'MagnetoCorp', 'DigiBank',
'5880000', '2019-10-30');
```

When you are done, save the file and from Balaji's console window run the `buy` command:

```
(balaji)$ node buy.js
```

You should see output stating that the transaction was successful:

```
MagnetoCorp commercial paper : 00002 successfully purchased
by DigiBank
```

Next, let's run `getPaper` again:

```
(balaji)$ node getPaper.js
```

The output should include this:

```
+--------- Paper Retrieved ---------+
 | Paper number: "00002"
 | Paper is owned by: "DigiBank"
 | Paper is currently: "TRADING"
 | Paper face value: "6000000"
 | Paper is issued by: "MagnetoCorp"
 | Paper issue on: "2019-06-31"
 | Paper matures on: "2019-12-30"
 +-----------------------------------+
```

As we can see, the paper is now owned by DigiBank and is now in the `TRADING` state.

Finally, let's change `redeem.js`, which is alongside `buy.js` in the DigiBank application folder. Open the file in your editor and find this line:

```
const redeemResponse = await contract.submitTransaction(
  'redeem', 'MagnetoCorp', '00001', 'DigiBank', '2020-11-30');
```

Here we need to change the paper number and the redeeming date for completeness, so edit the line to look like this:

```
const redeemResponse = await contract.submitTransaction(
  'redeem', 'MagnetoCorp', '00002', 'DigiBank', '2019-11-30');
```

When you are done, save the file and from Balaji's console window run the redeem command:

```
(balaji)$ node redeem.js
```

You should see output stating that the transaction was successful:

```
MagnetoCorp commercial paper : 00002 successfully redeemed with
MagnetoCorp
```

Next, let's run `getPaper` one last time:

```
(balaji)$ node getPaper.js
```

The output should look like this:

```
+--------- Paper Retrieved ---------+
 | Paper number: "00002"
 | Paper is owned by: "MagnetoCorp"
 | Paper is currently: "REDEEMED"
 | Paper face value: "6000000"
 | Paper is issued by: "MagnetoCorp"
 | Paper issue on: "2019-06-30"
 | Paper matures on: "2019-12-30"
 +-----------------------------------+
```

Here we can see the paper is now redeemed and the cycle is complete. At this point, feel free to experiment on your own, maybe by extending the smart contract further to improve error checking or by writing a new command-line application to find all `TRADING` or `REDEEMED` papers. Or, you could try and make the applications easier to use by taking arguments from the command line for the parameters such as `paperNumber` to save having to edit the files manually to work with a new asset.

Summary

We have taken a whirlwind tour of the Commercial Paper tutorial, and have updated its smart contract to include a new `getPaper` transaction. We have installed and upgraded the PaperNet network to the new version of the contract, and we have even written a new command-line application to execute the new `getPaper` transaction. Finally, we issued a new commercial paper asset and followed it on its lifecycle, looking at the state changes made to it on its journey.

If you are a developer of smart contracts, there is a free-to-use tool available to make your smart contract development much easier. The IBM Blockchain Platform Extension for VSCode helps Hyperledger Fabric developers to develop and test smart contracts and client applications on their local machines, as well as package their projects for deployment into IBM Blockchain Platform runtimes.

In the final chapter, we will take a look at the shape of things to come and where the future lies for blockchain technology.

CHAPTER 6

What's Next in Blockchain

In this book, we have looked in detail at blockchain for business. We started by defining relevant business concepts such as ledgers and contracts, and how the key principle behind blockchain is to share these artifacts between participants of a business network, making the data irrefutable and thereby helping to engender trust.

We looked at common applications for blockchain and how to identify good ideas. We then dived into the technology and looked at how to design and develop for it, using commercial paper as an example.

In this last chapter, we will look at where we currently are in terms of the development and uses of blockchain, and what the future might hold for this exciting innovation. We will do this by looking at blockchain through the lenses of the technology and the applications that use it.

Blockchain Technology

When a technology reaches a certain point of maturity, the core set of requirements has been largely implemented and an important period of optimization begins. We're now seeing this throughout the blockchain for business community, as there are significant efforts toward features that deliver quality-of-life improvements such as standardization, stability, and simplification. In addition, we're seeing (of course) the next generation of innovation. We'll now look at these different areas.

Standardization

Every successful technology goes through three distinct phases: *innovation*, *standardization*, and *commoditization*. Innovation is the trigger that sparks the initial interest in the technology. Standardization is the process of market forces that forces an industry into accepting a common vocabulary—whether that's a technical protocol, specification, or some other layer. Commoditization is the process by which technology becomes cheaper and easier to adopt.

For blockchain, the innovation trigger was arguably Bitcoin's creation in 2008, although you can point to other technologies such as peer-to-peer networks, hashchains, or even Luca Pacioli's documentation of double-entry bookkeeping as stepping stones that have led us to blockchain. Since 2008, however, after several years of blockchain education and experimentation, effort is now being invested on standardization. This phase is important because standards will allow for assets to flow more easily across diverse business networks, and for networks to grow regardless of ledgering technology. As we discussed in Chapter 1, the future lies in the network of networks.

Standardization can occur at many different layers; the best standards are descriptive rather than prescriptive, as they allow for further innovation and for new business models to form. Bodies such as ISO, IEEE, and W3C are all looking at different aspects of blockchain standards. Additionally, the Enterprise Ethereum Alliance has documented a specification for blockchain clients that is starting to gain traction, and the Inter-Ledgering Protocol is an effort at the protocol layer for blockchains to communicate, although this is currently very payments specific.

Over time, standards will emerge and aspects of blockchain technology will become commoditized, which is a good thing as it provides the motivation for further innovation. Vendors will look for additional value on top of the blockchain, whether that's through platforms, analytics, Internet of Things, or some other innovation trigger, and the cycle can begin anew.

Stability

While new features are being added to blockchain technologies such as Hyperledger Fabric, we can expect to see a stabilization period as business networks move their pilots into production. A good example of this is Hyperledger Fabric's v1.4 Long Term Service release, which aims to be a known baseline that businesses can comfortably move into production and upon which fixes can be applied.

Simplification

There is a strong demand for simplification, which is again a sign of the growing maturity of the technology as blockchain moves from being a research community effort into the mainstream. The increase in programming language coverage, simplified smart contracts, and client APIs in Hyperledger Fabric will make it even easier to develop blockchain solutions.

In the future, we can expect to see a further reduction in the barriers to entry for blockchain, including catering for nontechnical constituencies such as business leaders and lawyers, who might not need technical interfaces to the ledger but do have vested interests in the nature of active smart contracts.

The Next Generation of Blockchain Features

A huge amount of innovation continues to be applied to blockchain, typically through the application of related research. Together these help to address concerns and challenges of blockchain adoption within business networks. For example:

Cryptoanchors
These provide a way of encoding digital fingerprints in real-world objects. For blockchain this will provide an interesting way of identifying and verifying assets on a blockchain that cannot be easily encoded (aspirin or oil, for example), which will help to solve the problem of counterfeit goods.

Token management services
These allow permissioned blockchains to create fungible tokens that map to assets, which allows multiples of them to be easily traded. While the UTXO model that describes an efficient mechanism for the issuance, storage, and transfer of tokens has long been present in cryptocurrency implementations such as Bitcoin, we're now seeing it appear in policy-based blockchains such as Hyperledger Fabric. Tokens will help bring together previously separate conceptual models of assets and thereby allow these blockchains to work with a much wider set of asset types.

Zero-knowledge proofs

These allow you to prove to a third-party that you know a fact without revealing the fact itself. This will allow blockchains to be used as a store of proof rather than a store of data, allowing businesses to avoid the risk of sharing confidential information with unauthorized people. This is particularly important in asset-trading scenarios, in which the details of the participants of a trade might remain confidential, yet still providing transparency to regulatory bodies.

Artificial intelligence

This is a broad area that covers machine learning, analytics, and other technologies. Most commonly in blockchain, AI can be used to provide insight into the data stored in the shared system of record, giving insights into the lifecycle of assets and providing the data needed for further process optimization.

Blockchain Applications

Of course, all this technology is useless if it struggles to find a compelling application that solves a real-world need.

With hundreds of companies investing in blockchain, there is no shortage of press releases and other articles describing potential and actual uses of the technology. If the precedent of other innovative technologies over the years—from the printing press and steam engine to the internet—is anything to go by, it will be many years until we discover the full range of applications for which blockchain is suited.

Blockchain applications can be categorized into two main types: those that have a predominant benefit for business and those that have a stronger benefit for society as a whole. We'll now look at several examples.

Blockchains for Business

The early adopters of blockchain for business were predominantly in the finance industry. There are several possible reasons for this, including a strong fintech community, the association of blockchain (at least initially) with cryptocurrencies, and the fact that trust lies at the heart of the banking business (i.e., we trust banks to take care of our money rather than keeping it under our mattresses). There are several well-advanced finance blockchains, for scenarios including international payments, trade finance, and letters of credit.

One of the most popular areas for today's business blockchains is for the tracking of assets in supply chains. This is because of the lack of trust and transparency that many supply chains exhibit, the problem of counterfeiting that results from this, and the ability for blockchains to irrefutably track asset provenance. Some examples of supply chain blockchains include food, freight, diamonds, wine, and drugs, but the same template could be applied to any high-value asset. We're now seeing the emergence of templates to make the development of supply chain blockchains easier (e.g., Hyperledger Grid™).

Looking ahead, what businesses will start to find is that the industries in which blockchains are deployed are no longer relevant. The interrelated nature of business networks means that as blockchains expand their membership and complexity, defining a blockchain as belonging to a single industry or geography no longer makes sense. Supply chains are typically global and can span manufacturing, distribution, retail, finance, and other sectors, too. At what point does a drug blockchain cease to be a drug blockchain when a dispensing transaction triggers a stock-replenishment business process that then triggers a financing operation, manufacturing, and parts replacement?

In a network of networks, the cascade effect will be significant and businesses will become truly interconnected. These efficiency gains will reduce cost, allow new markets to be reached, and bring about a new era of transactional applications. Different governments have different priorities when it comes to protectionism versus free market access, but over time we know that competitive markets will remove barriers to trade and gravitate toward the lowest-cost, most efficient solutions. Blockchain can help make that happen.

Blockchains for the Good of Society

There is also a compelling set of blockchain applications that will (or already) exist predominantly for the good of society, yet also provide complementary benefit to businesses. Here are some of the more interesting ones:

- A 2009 study by the United Nations Office on Drugs and Crime showed that Afghan citizens paid $2.5 billion in bribes (about 23% of Afghanistan's GDP). Smart contracts enforce the terms for any given transaction and the blockchain itself engenders transparency. These characteristics, when applied to charitable donations and aid, will help reduce bribery and the effects of corruption, particularly in developing countries.
- The world is seeing an increase in extreme weather, and the ability to insure homes in affected areas is often impossible because a single insurer cannot adequately cover the risk. In California, for example, just 12% of homeowners are covered by earthquake insurance. Pooling the risk using catastrophe bonds and blockchain to automate the claims process across the insurer network will help families rebuild their lives more quickly after extreme weather events.
- Dynamic blockchain-based marketplaces will lead to more efficient use of natural resources in an increasingly complex energy network that includes diverse producers and consumers, such as electric cars and solar panels. For example, TenneT and Vandebron have a pilot project that offers car owners access to the electricity market. These marketplaces will also help clean up the plastic in our oceans by creating incentives for collection and recycling of the plastic waste (for example, PlasticBank).
- Decentralized identity systems (such as the Sovrin network and platforms built using Hyperledger Indy) put users in control of their own personal data and also provides a mechanism for verifiable claims (facts) against those users. This will allow organizations such as educational institutions and employers to attest qualifications and work history, which will help to prevent fraud. The same idea could also help content producers attribute media to their sources, thus helping to combat plagiarism and "fake news."

Summary

We really are just scratching the surface when it comes to blockchain applications. The technology comes with a very simple set of aims: to engender trust and remove friction from transactions. Imagine being able to reliably exchange assets with anyone, anywhere in the world without borders, as easily as you can use the internet to exchange data today. With blockchain, we are living in an age of innovation, and it's exciting to see where it will lead us.

About the Authors

Michael Bradley is the global Program Director for Blockchain Enablement and Education for IBM. He is responsible for the enabling of IBMers and clients to advocate, sell, and deliver IBM's blockchain offerings. He has had a variety of roles with IBM and over 20 years of industry experience. Michael's leadership roles in IBM have spanned the Development Labs (Messaging, IoT, Emerging Technologies) and IBM Services (Consultancy & Strategic Outsourcing), giving him a deep and broad business understanding.

David Gorman is part of IBM's global blockchain team, and is based at IBM's development lab in Hursley, England. The team is the center of blockchain client engagement worldwide for IBM. In this role, Dave works with customers across a broad spectrum of industries and use cases, including the financial services sector, enabling them in their understanding of blockchain and how they can best make use of the technology. Dave has worked in the IT industry for over 28 years, including several years in enterprise middleware integration technologies and performance analysis.

Matt Lucas is part of IBM's global blockchain team. His role is to help clients understand and apply blockchain technologies, and he works closely with emerging blockchain fabrics such as Hyperledger Fabric and Ethereum. Through collaboration with various universities worldwide, he also helps the next generation of computing professionals understand technology and its application in business. He is based in IBM's development laboratory in Hursley, England, and has worked with IBM for over 20 years on a variety of integration middleware technologies. Most recently he spent several years working on IBM Integration Bus in the product architecture and offering management disciplines. You can contact Matt on Twitter using *@mqmatt*, or via email at *lucas@uk.ibm.com*.

Matthew Golby-Kirk is part of IBM's global blockchain engagement team and has delivered over 100 client events, briefings, and hands-on labs on IBM's Blockchain Platform and Hyperledger Fabric around the world. He is based at IBM's development laboratory in Hursley, England, and has worked with IBM for almost 20 years on a variety of integration and middleware technologies. Prior to his time with the blockchain team, he spent several years working on IBM Integration Bus in the product development team, responsible for many areas of product functionality.