Blockchain for Enterprise

Build scalable blockchain applications with privacy, interoperability, and permissioned features

Packt

www.packt.com

Narayan Prusty

Blockchain for Enterprise

Build scalable blockchain applications with privacy, interoperability, and permissioned features

Narayan Prusty

Packt

BIRMINGHAM - MUMBAI

Blockchain for Enterprise

Commissioning Editor: Sunith Shetty
Acquisition Editor: Namrata Patil
Content Development Editor: Chris D'cruz
Technical Editor: Suwarna Patil
Copy Editor: Safis Editing
Project Coordinator: Nidhi Joshi
Proofreader: Safis Editing
Indexer: Tejal Daruwale Soni
Graphics: Tom Scaria
Production Coordinator: Nilesh Mohite

First published: September 2018
Production reference: 1250918

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham
B3 2PB, UK.

ISBN 978-1-78847-974-5

`www.packtpub.com`

Contributors

About the author

Narayan Prusty is the founder and CTO of BlockCluster, world's first blockchain management system. He has five years of experience in blockchain. He specializes in Blockchain, DevOps, Serverless, and JavaScript. His commitment has led him to build scalable products for start-ups, governments, and enterprises across India, Singapore, USA, and UAE. He is enthusiastic about solving real-world problems. His ability to build scalable applications from top to bottom is what makes him special. Currently, he is on a mission to make things easier, faster, and cheaper using blockchain. Also, he is looking at ways to prevent corruption, fraud, and to bring transparency to the world using blockchain.

About the reviewers

Nikhil Bhaskar is the founder and CEO of Ulixir Inc—a newly founded tech company that builds decentralized and traditional software. He completed B9lab's Ethereum Developer Course, and he is now a certified Ethereum developer. Aside from running Ulixir, he spends his time traveling and eating. He is a bit of a digital nomad; this year, he's lived in five countries and plans to live in six more before the year ends.

Ivan Turkovic is a geek, visionary, start-up enthusiast, writer, blogger, mentor, and advisor. He wrote the book *PhoneGap Essentials*. Since 2011 he has had a strong interest in Bitcoin and blockchain. In 2013 he co-founded a social start-up, Babberly, which was among the first to use gamification with the help of blockchain.

He is focused on bringing value to the internet users. He employs the latest technologies to build empowering web products and intuitive user experiences. He's interested in technology, entrepreneurship, education, behavior psychology, product management, and marketing. He runs Blaeg, a company that helps start-ups get off the ground with their blockchain technology.

Anand V. is a technology architect who has more than 20 years of experience in IT. He has worked with Verizon Communications, Cognizant, HP, HCL, and Oracle. Currently, he is the managing partner of Anasup Consulting and works with clients on emerging technologies such as blockchain, IoT, cybersecurity, and AI. He is also a specialist in the DevSecOps area and acts as a mentor to many start-up companies. He is a public speaker and regularly writes articles in journals as well as online channels.

Table of Contents

Preface

Blockchain is growing massively, and is changing the way that business is done. Leading organizations are already exploring the possibilities of blockchain. With this book, you will learn how to build end-to-end, enterprise-level **decentralized applications** (**DApps**) and scale them across your organization to meet your company's needs.

This book will help you understand what DApps are and the workings of the blockchain ecosystem with some real-world examples. This is an extensive end-to-end book covering every aspect of blockchain, such as its applications for businesses and developers. It will help you be aware of the process flows so you can incorporate them into your own enterprise. You will learn how to use J.P. Morgan's Quorum to build blockchain-based applications. You will also be introduced to how to write applications that can help communicate in enterprise blockchain solutions. You will learn how to write smart contracts that run without censorship and third-party interference.

Once you have a good grip on what blockchain is and have learned all about Quorum, you will jump into building real-world practical blockchain applications for sectors such as payment and money transfer, healthcare, cloud computing, supply chain management, and much more.

Who this book is for

This book is for innovators, digital transformers, and blockchain developers who want to build end-to-end DApps using blockchain technology. If you want to scale your existing blockchain system across the enterprise, you will find this book useful too. It gives you the practical approach needed for solving real problems in an enterprise using a blend of theory- and practice-based approaches.

What this book covers

`Chapter 1`, *What are Decentralized Applications?*, will explain what DApps are and provide an overview of blockchain-based DApps.

`Chapter 2`, *Building Blockchain Using Quorum*, introduces the basics of Ethereum blockchain and the features of Quorum. This chapter also teaches you how to set up a Raft network using Quorum and various third-party tools and libraries.

`Chapter 3`, *Writing Smart Contracts*, shows how to write smart contracts and use geth's interactive console to deploy and broadcast transactions using web3.js.

`Chapter 4`, *Getting Started with web3.js*, introduces web3.js and how to import and connect to geth, and explains how to use it in Node.js or client-side JavaScript.

`Chapter 5`, *Building Interoperable Blockchains*, explores what interoperable blockchains can achieve, the various technologies and patterns for achieving blockchain interoperability, and building interoperable blockchain networks to represent FedCoins.

`Chapter 6`, *Building Quorum as a Service Platform* , will teach you the basics of cloud computing and containerization by examples. You'll learn how to install minikube, deploy containers on Kubernetes, and develop a Quorum-as-a-service using QNM.

`Chapter 7`, *Building a DApps for Digitizing Medical Records* , gets into how to use proxy re-encryption to enable encrypted data sharing in blockchain. Besides proxy re-encryption, you'll also learn about a lot of JavaScript and Python libraries, such as `etherumjs-wallet`, `ethereumjs-tx`, `ethereumjs-util`, and `npre`. Also, you'll learn about signing transactions using keys stored outside a geth node.

`Chapter 8`, *Building a Payment Solution for Banks*, looks at how to implement network permissioning in Quorum and how to build a solution to transfer money using a mobile number.

To get the most out of this book

You must have experience with the JavaScript and Python programming languages.

You must have developed distributed web applications before.

You must understand the basic cryptography concepts, such as signing, encryption, and hashing.

Download the example code files

You can download the example code files for this book from your account at `www.packt.com`. If you purchased this book elsewhere, you can visit `www.packt.com/support and register to have the files emailed directly to you.`

You can download the code files by following these steps:

1. Log in or register at `www.packt.com`.
2. Select the **SUPPORT** tab.
3. Click on **Code Downloads & Errata**.
4. Enter the name of the book in the **Search** box and follow the onscreen instructions.

Once the file is downloaded, please make sure that you unzip or extract the folder using the latest version of:

- WinRAR/7-Zip for Windows
- Zipeg/iZip/UnRarX for Mac
- 7-Zip/PeaZip for Linux

The code bundle for the book is also hosted on GitHub at `https://github.com/PacktPublishing/Blockchain-for-Enterprise`. In case there's an update to the code, it will be updated on the existing GitHub repository.

We also have other code bundles from our rich catalog of books and videos available at `https://github.com/PacktPublishing/`. Check them out!

Conventions used

There are a number of text conventions used throughout this book.

`CodeInText`: Indicates code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles. Here is an example: "This Raft ID will appear while adding a node using `raft.addPeer`."

A block of code is set as follows:

```
url = "http://127.0.0.1:9002/"
port = 9002
storage = "dir:./cnode_data/cnode2/"
socket = "./cnode_data/cnode1/constellation_node2.ipc"
othernodes = ["http://127.0.0.1:9001/"]
publickeys = ["./cnode2.pub"]
privatekeys = ["./cnode2.key"]
tls = "off"
```

Any command-line input or output is written as follows:

```
git clone https://github.com/jpmorganchase/quorum.git
cd quorum
make all
```

Bold: Indicates a new term, an important word, or words that you see onscreen. For example, words in menus or dialog boxes appear in the text like this. Here is an example: "Now select a file, enter the owner's name, and click on **Submit**. ."

Warnings or important notes appear like this.

Tips and tricks appear like this.

Get in touch

Feedback from our readers is always welcome.

Reviews

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions, we at Packt can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about Packt, please visit `packt.com`.

1 What are Decentralized Applications?

Since the beginning of internet, all internet-based applications that have been developed have been based on client-server architecture, where there is a centralized server that forms the backend of the application and controls the complete application. These applications often end up with issues such as having a single point of failure, failure to prevent net censorship, lack of transparency, users not trusting their data, activity and identity privacy, and so on. This centralized architecture even made it impossible to build certain kinds of applications. For example, you cannot build a digital currency using this architecture. Due to these issues, a new kind of architecture emerged called **Decentralized Applications** (**DApps**). In this chapter, we will learn about DApps.

In this chapter, we'll cover the following topics:

- What are DApps?
- What is the difference between decentralized, centralized, and distributed applications?
- What is a blockchain?
- What is the difference between public and permissioned DApps?
- Examples of some of the popular consortium DApps, and how they work
- What are the various popular platforms on which to build enterprise DApps?

What is a DApp?

A DApp is a kind of application whose backend runs on a decentralized peer-to-peer network, and its source code is open source. No single node in the network has complete control of the DApp. Remember that, when we say that an application is decentralized we mean technically it's decentralized but the governance can be distributed, decentralized, or centralized.

The major advantages of DApps are that they don't have a single point of failure, and prevent censorship. DApps do have some disadvantages: it's difficult to fix bugs or add features once deployed as everyone in the network has to update their node software, and it's very complicated to couple different DApps together as they are very difficult to build compared to centralized applications and involve very complex protocols.

To be able to use a DApp, you first need the DApp's node server running so that you can connect to the peer-to-peer network. Then, you need a client respective to the DApp that connects to the node server and exposes a UI or command line interface to use the DApp.

Currently, DApps are not yet as mature as centralized applications in terms of performance and scalability. There is still a lot of research and development on these topics such as performance, scalability, users identity, privacy, communication between DApps, data redundancy, and so on. A use case may fit into a DApp, but whether the use case can be made production-ready with the currently available technology can be a challenge. Popular examples of decentralized applications are Torrent, Bitcoin, Ethereum, Quorum, and so on.

A DApp can be public or permissioned. Public DApps are those which anyone can be part of, in other words, they are permissionless, whereas permissioned DApps are those which are not open for everyone to join, so you will need permission to join. A permissioned DApp is called a **consortium DApp** when the participants of the DApp are enterprises and/or government entities. Similarly, when the participants of a permissioned DApp are only enterprises, then we can call it an enterprise DApp. In this book we will learn everything about permissioned DApps.

As you just got a basic introduction to what decentralized applications are, you must be wondering what the difference between decentralized and distributed applications is. Well, an application is said to be distributed when it's spread across multiple servers. Decentralized applications are by default distributed, whereas centralized applications may or may not be distributed. Centralized applications are usually distributed across multiple servers to prevent downtime, and also to handle huge data and traffic.

What is a blockchain?

Before we get into what a is, we need to understand what a ledger is. A ledger in computer science is software that stores transactions. A database is different from a ledger such that in a database we can add, remove, and modify records, whereas in a ledger we can only append but not delete or modify.

A blockchain is basically a data structure to implement a decentralized ledger. A blockchain is a chain of blocks connected to each other. Every block contains a list of transactions and certain other metadata, such as when it was created, which is it's previous block, the block number, who is the creator of the block, and so on. Every block maintains a hash of the previous block, therefore creating a chain of blocks linked with each other. Every node in the network should hold the complete copy of the blockchain and, when a new node comes in, it will request and download the blockchain from other nodes.

Technologies such as blockchains are called **Distributed Ledger Technology** (**DLT**). A DLT is the process of replicating, sharing, and synchronizing digital transactions geographically stretched across numerous sites, countries, and/or institutions. You can think of a blockchain as a type of DLT. Also, not every DLT system has to be decentralized. In this book, we only learn to build decentralized blockchain-based applications.

The major advantages of using a blockchain is that it enables the facilitation of transactions without a central trusted party; data is secured using cryptography, and data is immutable, as blockchain removes friction and reduces risk so settlements happen in real time, and so on. Basically, it automates auditing, makes the application transparent, and provides a single source of truth.

In the real world, private blockchains are used in trade finance, cross-border payments, digital identity, the clearing and settlement of tokenized and digital assets, provenance of ownership of a product, record keeping for critical data, signing contracts, multi-party aggregation (namely, they can be used as a shared master repository for common industry information, allowing members to query for data), payment-versus-payment or payment-versus-delivery, and so on.

Every blockchain node maintains a database that contains the blockchain's state. The state contains the final result of running all the transactions in the blockchain. For example, in a blockchain, the state represents the final balances of all addresses. So when you query the blockchain node for an addresses balance, it doesn't have to go through all transactions and calculate the final balance of the address; instead, it directly fetches the balance from the state of the blockchain. Bitcoin uses LevelDB to maintain the state of the blockchain. Even if the database gets corrupted, the database can be restored by simply running all the transactions in the blockchain.

Understanding Byzantine Fault Tolerance

Byzantine Fault Tolerance (**BFT**) is a characteristic of a decentralized system that indicates that it can tolerate Byzantine failures. A crash failure is when nodes just stopping to do anything (no messages at all) and Byzantine failure is when nodes just don't do anything or exhibit arbitrary behavior. Basically, Byzantine failures include crash failures.

In any decentralized computing environment where a blockchain data structure is used, there is a risk that one or more rogue or unreliable actors could be a reason for the environment to disband. A server cluster will not work well if a few servers within it lose out on passing data to other servers in a consistent manner. In order to be reliable, the decentralized computing environment has to be designed in a way that it has solutions to these kinds of Byzantine failures.

On blockchain-based decentralised applications, there is, by definition, no central authority, so a special kind of protocol called the **consensus protocol** is used to achieve BFT.

In simple terms, you must be wondering how to ensure that everyone has the same copy of the blockchain, and how to know which blockchain is correct when two nodes publish different blockchains? Also, how do you decide who creates the blocks, as there is nothing such as a master node in decentralized architecture? Well, consensus protocols provide an answer to these questions. A few examples of consensus protocols are **Proof-of-Work** (**PoW**), **Proof-of-Stake** (**PoS**), **Proof-of-Authority** (**PoA**), PBFT, and so on.

A consensus protocol is designed specially for permissioned or public blockchains. A consensus protocol made for a public blockchain is likely to create security and performance issues when implemented in a permissioned blockchain. Every consensus protocol has different performance and scalability vectors. You have to be alert while selecting a consensus protocol for your blockchain-based DApp.

TIP

Consensus protocols such as Raft and Paxos are not BFT; rather, they make the system only crash-tolerant. So, you should also consider this when choosing a consensus protocol.

You might have come across the term PoA. PoA is a categorisation of consensus protocols in which there is a set of authorities—nodes that are explicitly allowed to create new blocks and secure the blockchain. Ripple's iterative process, PBFT, Clique, Aura, and so on, are examples of PoA-based consensus protocols.

Representation of user accounts

In blockchain-based applications, user accounts are identified and authenticated using asymmetric key pairs. The private key is used to sign transactions on behalf of the user. Username and password-based accounts systems will not work in blockchain as it cannot be used to prove which user has sent a transaction. The demerits in using private-public key pair include that they are not user-friendly and if you lose the private key then there is no way to recover it. So, it adds a new responsibility for the users to secure their private key. The address of a user account acts as the account identifier on blockchain. The address of a user account is derived from the public key.

What are UTXOs?

Some blockchain applications use the UTXO model for transactions. Blockchain applications such as Bitcoin and MultiChain use this model. Even DLTs such as R3 Corda also use this model. Let's understand this model by understanding how Bitcoin transactions work.

In Bitcoin, a transactions is a collection of zero or more and outputs. These input and output objects are called **Unspent Transaction Outputs** (**UTXO**). Outputs of transactions are used as inputs of future transactions. A UTXO can be used as input only once. Each UTXO in Bitcoin contains a denomination and an owner (a Bitcoin address). In this model, the balances of addresses in the unconsumed UTXOs are stored. For a transaction to be valid, these requirements should be met:

1. The transaction must contain a valid signature for the owner of each UTXO that it consumes
2. The total denomination of the UTXOs consumed must be equal to or greater than the total denomination of the UTXOs that it produces

A user's balance is computed as the total sum of the denominations of UTXOs that they own. A transaction can consume zero or more UTXOs and produce zero or more UTXOs. For a miner to pay reward to itself, it includes a transaction in the block that consumes zero UTXOs but produces one UTXO with the denomination assigned the amount of Bitcoin it is supposed to award itself.

A UTXO transaction model is suitable when blockchain transactions involve the transfer of asset, but for non-assets transfer transactions such as recording facts, invoking smart contracts, and so on, this model it not suitable.

Popular permissioned blockchain platforms

Now we have a basic idea about what a DApp, blockchain, and DLT is, let's have an overview of what platforms are available to build a permissioned blockchain applications and DApps. We will only go through the ones that are popular on the market, and for which there is a demand.

Ethereum

Ethereum is the most popular DApp after Bitcoin. Ethereum is a decentralized platform that allows us to build other blockchain-based DApps on top of it. In Ethereum, we build DApps using Ethereum smart contracts. Smart contracts are applications that run exactly as programmed without any possibility of downtime, censorship, fraud, or third-party interference. Ethereum can be thought of as a platform to deploy and run smart contracts. Ethereum supports two consensus protocols, PoW and PoA (Clique).

The main public Ethereum network uses PoW for consensus. If you want to deploy your own private Ethereum network, then you have to use PoA. PoW requires a lot of computation power to keep the blockchain secure, therefore it's good for public blockchain use, whereas PoA doesn't have any such computation power requirement; instead it requires a few authority nodes in the network to achieve consensus.

You must be wondering why we need smart contracts to build DApps. Why cannot we simply put formatted messages on blockchain in the form of transactions and interpret them on client? Well, using smart contracts gives you both technical and business benefits.

Quorum

Quorum is a decentralized platform that allows us to build permissioned blockchain-based DApps on top of it. Actually, Quorum is a fork of Ethereum (actually Quorum is a fork of Go Ethereum, which is an implementation of Ethereum using Golang), therefore if you have ever worked on Ethereum then you will find it easy to learn and build permissioned blockchains using Quorum. Many enterprises select Quorum for building blockchains because of Ethereum's large community, which makes it easy to find Ethereum developers. What makes Quorum different from Ethereum is that it supports privacy (it lets parties do transactions privately); peer whitelisting, so you can mention a list of other nodes that are allowed to connect to your node (in Ethereum this needs to be done at network level); many different flavors of consensus protocols suitable for permissioned blockchain, and provides very high performance.

Quorum currently supports three consensus protocols, QuorumChain, IBFT, and Raft. We will skip QuorumChain in this book, as Raft and IBFT fulfil all our requirements.

Microsoft Azure provides BaaS to easily build your own Quorum network on the Cloud. But, in this book, we will learn how to install it manually, and we won't be using BaaS.

Parity

Popular node software for Ethereum include Go Ethereum, Ethereum C++, and Parity. Parity also supports two other consensus protocols, other than Ethereum's PoW, which are specifically designed for permissioned blockchains. These consensus protocols are Aura and Tendermint. Many Ethereum developers use parity compared to Quorum when they don't need the extra features provided by Quorum.

As parity doesn't provide any unique features compared to Quorum, we will be skipping parity in this book. But, once you finish this book, you will find it really easy to grasp parity's concepts and will be able to build something using it too.

MultiChain

MultiChain is a platform to build permissioned blockchain-based DApps. Unique features of MultiChain include permissions management, data streams, and assets. It doesn't support smart contracts. This is an example of a non-smart contract-based platform for building blockchain-based DApps. MultiChain uses round robin validation consensus.

Initially MultiChain was based on the idea of managing ownership and transfer of assets on blockchain. Operations on assets includes issuance, reissuance, transfer, atomic exchange, escrow, and destruction of assets. Later on, data streams were introduced to provide a different flavor of representing data in MultiChain. Any number of streams can be created in a MultiChain, and each stream acts as an independent append-only collection of items. Operations on streams include creating streams, writing, subscribing, indexing, and retrieving. So, basically, a blockchain use case on MultiChain can be built on a foundation of assets or streams. Finally, permission management is used to control who can connect, transact, create assets/streams, mine/validate, and administrate.

MultiChain provides maximal compatibility with the Bitcoin ecosystem, including the peer-to-peer protocol, transaction/block formats, the UTXO model, and Bitcoin Core APIs/runtime parameters. So, before you start learning MultiChain, it's better to learn how Bitcoin works at a high level at least.

Hyperledger Fabric 1.0

Before we get into what Hyperledger Fabric 1.0 is, we need to understand what Hyperledger is specifically. Hyperledger is an umbrella project of open source blockchains and related tools, started in December 2015 by the Linux Foundation. At the time of writing this book, there are four projects under Hyperledger:Fabric, Sawtooth, Iroha, and Burrow.

Hyperledger Fabric is the most popular project under Hyperledger. IBM is the main contributer to the project. IBM's Bluemix also provides BaaS to build your own Fabric network on the Cloud easily.

Hyperledger Fabric 1.0 is a platform to build your own permissioned blockchain-based applications. Currently, at the time of writing this book, Hyperledger Fabric 1.0 supports only distributed architecture, and for the creation of blocks it depends on a central trusted node called the **orderer**. It supports smart contracts, network permissioning, privacy, and other features. In HLF 1.0, there is a special kind of node called as **OSN**, which is hosted by a trusted party. This OSN creates blocks and distributes to peers in networks. As you trust this node, there is no need for consensus. HLD 1.0 currently supports CouchDB and LevelDB to store the state of the blockchain. Peers in the network store the state of the blockchain in the LevelDB database, by default.

HLF 1.0 has a concept of channels to achieve privacy. A channel is a sub-blockchain in the network and allows certain parties to be part of a channel depending on configuration. Actually, every transaction has to belong to a channel and when the HLF 1.0 network is deployed, a default channel is created. OSN can see all the data in all the channels, therefore it should a trusted party. Technically, it's possible to configure the network to have multiple OSNs hosting different channels if you cannot trust a single party for all channels. Even if the traffic is going to be huge or OSN availability is critical, then you can plug Kafka into OSN for better performance and increased stability. We can even have multiple OSNs per channel connected via Kafka if high availability is required.

Fabric 1.0 has a feature called **transaction endorsement**, which provides a mechanism of taking approvals from certain parties before sending a transaction. When we say that a transaction has been endorsed by a member in the network, we mean that the member has verified the transaction. Every chaincode (smart contracts in HLF) has an endorsement policy defined to it at the time of deployment. The policy states which members has to endorse the transactions associated with this chaincode. The default policy states that any one member of the channel has to sign the transaction. But, we can define custom policies containing **AND** and **OR** operators.

Also, peers of the same channel broadcast blocks to each other regardless of the presence or absence of OSN, but in the absence of OSN new blocks cannot be created for the channel. Peers broadcast blocks using a special protocol called as **gossip data dissemination protocol**.

HLF 1.0 has very advanced membership features to control network membership, and that are also internal to a specific organization. In HLF 1.0, you can write chaincodes in Java or Go programming languages. In the future, Fabric 1.0 will come with the **Simple Byzantine Fault Tolerance** (**SBFT**) consensus protocol and some other features that will enable us to build DApps. Similarly, there are various new features that are under development and will be released in future as a sub-version of the product.

TIP

The best way to get started with building your first HLF 1.0 application is the check out examples at `https://github.com/hyperledger/fabric-samples` and modify them according to your application needs. You can find HLF 1.0 detailed docs at `http://hyperledger-fabric.readthedocs.io/en/latest/`.

BigchainDB

BigchainDB is a decentralized database that uses blockchain. BigchainDB is highly scalable and customizable. It uses the Blockchain data structure. It supports features such as rich permissioning, petabytes capacity, advanced querying, linear scaling, and so on. At the time of writing this book, BigchainDB is not production-ready but can be used for building **Proof of Concepts** (**PoCs**). We will learn how it works, and will create a basic PoC using it, in later chapters.

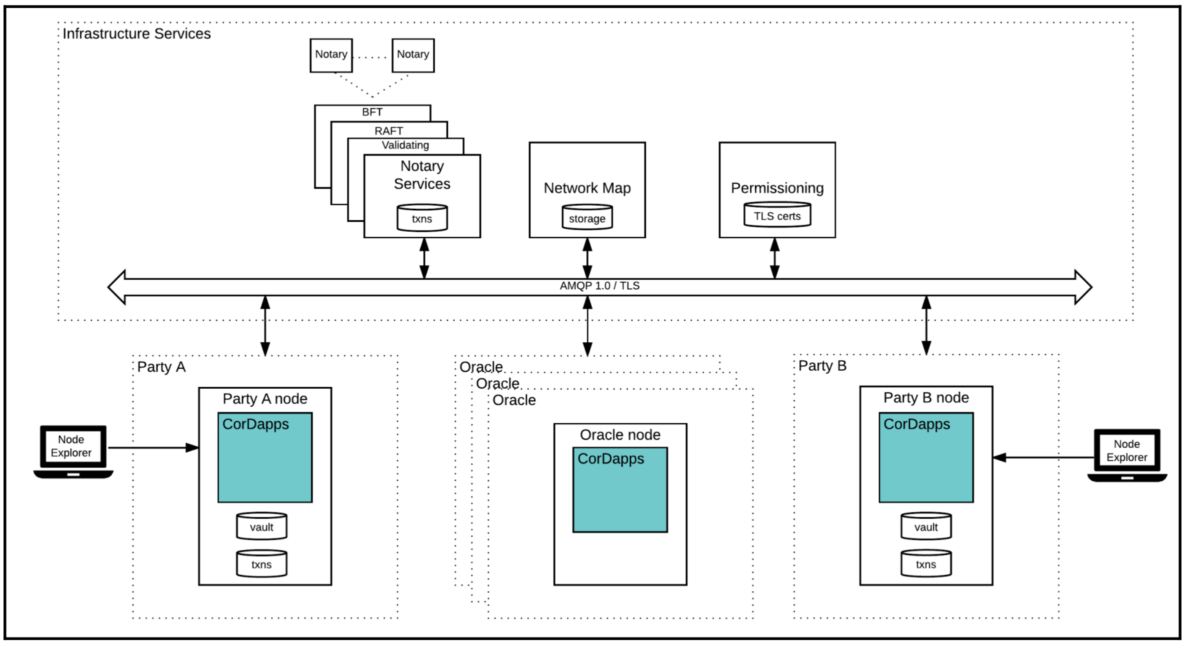
InterPlanetary File System

InterPlanetary File System (**IPFS**) is a decentralized filesystem. IPFS uses **Distributed Hash Table** (**DHT**) and Merkle **Direct Acyclic Graph** (**DAG**) data structures. It uses a protocol similar to Torrent to decide how to move the data around the network. One of the advanced features of IPFS is that it supports file versioning. To achieve file versioning, it uses data structures similar to Git.

Although it's called as a decentralized filesystem, it doesn't adhere to a major property of a filesystem, namely, when we store something in filesystem, it should be there until deleted. But, IPFS doesn't work this way. Each node doesn't store all files, instead it stores only those files it needs. Therefore, if a file is not popular, then many nodes will not have the file therefore there is a huge chance of the file disappearing in the network. Due to this, we can call IPFS a decentralised peer-to-peer file-sharing application. We will learn about about how it works in later chapters.

Corda

Corda is a platform on which to build your own permissioned DLT-based applications. Corda is a product of R3. R3 is an enterprise software firm working with over 100 banks, financial institutions, regulators, trade associations, professional services firms, and technology companies to develop Corda. The latest version of Corda is 1.0, which aims to replace legacy softwares used for financial transactions, and enables organisations to digitalize various business process that were cumbersome using legacy software systems:



The preceding diagram shows the high level architecture of a Corda network. Let's understand Corda's architecture at a high level. The idea of R3's Corda is to provide a shared trusted ledger for financial transactions. R3's Corda is not a blockchain platform, therefore there is no concept of blocks, global broadcasts, and so on. All the transactions are point to point. Corda applications are not decentralized. In Corda, smart contracts are called as **CorDapps** and they are written in either Java or Kotlin.

Infrastructure services form the nodes in the network that should be hosted by the trusted parties. Network Map publishes IP addresses of all the other nodes, so that nodes can reach out to other nodes. Permissioning service gives permission to nodes to join the network ; the node will receive a root-authority-signed TLS certificate from the network's permissioning service if permitted to join the network. Notaries provide transaction ordering and time stamping services (optionally, a notary also acts as the timestamping authority, verifying that a transaction occurred during a specific time-window before notarizing it). A notary service may be a single network node, a cluster of mutually-trusting nodes, or a cluster of mutually-distrusting nodes.

Notaries are expected to be hosted by enterprises that the network doesn't trust, therefore consensus is required between the notaries, due to which Corda provides various pluggable consensus protocols, such as Raft, BFT, and so on.

Sometimes, Corda applications need to depend on external application APIs. For example, a multi-currency bank-to-bank payment application built using Corda will need to fetch the exchange rate. In this scenario, the node initiating the transaction can fetch the exchange rate and put on the transaction, but how can you trust that node? Also, every node cannot simply re-fetch the exchange rate to verify if it's correct because by the time other nodes fetch it the rate might have changed, and also this is not a scalable solution. Therefore, Corda provides oracles to solve this issue. There can be one or more oracles in the network. An Oracle is a service that acts as a bridge for communication between two applications. In Corda, the transaction initiator can fetch the information from outside the Corda network and get the information signed from **Oraclize** to prove its validity. Optionally, Oraclize can also provide the information to the transaction initiator on request. Obviously, the Oraclize should be hosted by trusted parties with respect to what information they provide and sign.

Corda supports any pluggable RDBMS (currently, it is using the H2 database) to store smart contracts data. Data privacy is maintained as to which nodes can see the transactions. Multisignature support is also given by the framework, which enables multiple nodes to sign a transaction. One of the major downsides of Corda is that as there is no global broadcasting, each node has to maintain its own backup and failover redundancy in a traditional way as there is no redundancy built into the network. A node will store transactions and retry sending the messages to the recipient until the recipient has successfully received it. Once the messages are received, the sender has no more responsibility.

Transaction validity

As all transactions are not broadcasted to all parties in the network, to prevent a double spend (a double spend is an attack on DLTs to spend the same money twice, transfer the same asset twice and so on), we use notaries. Notaries contain all the unconsumed UTXOs, and after notarization they mark them as consumed and add the new unconsumed ones to their state. The transaction purposer gets the transaction notarized by a notary before sending to the other parties for commit.

A notary will only be able to sign a transaction if it has earlier signed input states of the transaction. But, this may not always be the case, therefore Corda also lets us change the state's appointed notary. This situation can occur mostly due to the following reasons:

- A transaction consuming states that have different appointed notaries
- A node wishes to use a different notary for achieving privacy or efficiency

Before these transactions can be created, the states must first be repointed to all have the same notary. This is achieved using a special notary-change transaction.

CorDapps are not like smart contracts of other platforms. They don't have a state. Their purpose is to just validate if the outputs produced from the inputs are correct. Every UTXO points to a CorDapp. CorDapps define the format of UTXOs. In a transaction, we can have UTXOs of multiple CorDapps, and in these cases each CorDapp will run only once and validate all the inputs and outputs belonging to it. For a transaction to be valid, it must be contractually valid; the CorDapp should approve it.

Apart from inputs and outputs, transactions might consist of commands too, small data packets that the platform doesn't decipher itself but which help CorDapps to process the inputs and outputs. A command is a piece of data associated with some public keys. Commands are used to provide additional information to the CorDapps that it cannot get via the UTXOs. The platform assures that the transaction is signed by every key listed in the commands before the contracts start to execute. Thus, the CorDapp can trust that all listed keys have signed the transaction, but is responsible for verifying that the intended parties have signed it. Public keys may be random or identityless for privacy, or linked to a well-known legal identity.

Oracles provide signed information to the transaction purposer in the form of commands that encapsulate a specific fact, and list the oracle as a required signer.

Also, transactions can contain a hash of attachments. Attachments are ZIP/JAR files. Attachments are useful when there's a large fragment of data that can be reused across several different transactions.

It is possible that while verifying a proposed transaction, the node may not have all the transactions of the transaction chain that it needs to verify. Therefore, Corda lets the node request the missing transactions from the proposer(s). It's always true that the transaction proposer will have all the transactions of the required transaction chain, as they would have requested it when verifying the transaction and created the purposed transaction's input states.

Finally, once the transaction is committed, you can query the Vault (which keeps track of both unconsumed and consumed states).

To learn more about Corda and build your first Corda app, visit `https://docs.corda.net/`, which contains detailed documentation. There are several example apps that you can download and experiment with.

Hyperledger Sawtooth

Sawtooth is a decentralized platform to build your own permissioned DApps. The main contributer to Sawtooth is Intel. What makes Sawtooth special is that it uses a **Trusted Execution Environment** (**TEE**) (it currently supports Intel's SGX only) for consensus, which makes the network very safe and trustworthy and increases trust in the final result of the consensus.

The TEE is a secure area of the main processor. It guarantees that the code and data loaded inside is protected with respect to confidentiality and integrity. The TEE as an isolated execution environment that provides security features such as isolated execution, integrity of Trusted Applications, along with confidentiality of their assets.

Proof of Elapsed Time (**PoET**) is the name of the consensus protocol that Sawtooth uses. In PoET, there are special types of nodes called as validators. Validators must run their node on an SGX-supported CPU. This is how PoET works.

Every validator requests a wait time from an enclave (a trusted function). The validator with the shortest wait time for a particular transaction block is elected the leader. One function, say `CreateTimer`, creates a timer for a transaction block that is guaranteed to have been created by the enclave. Another function, say `CheckTimer`, verifies that the timer was created by the enclave and, if it has expired, creates an attestation that can be used to verify that the validator did, in fact, wait the allotted time before claiming the leadership role. PoET randomly distributes leadership election across the entire population of validators. The probability of election is proportional to the resources contributed (in this case, resources are general-purpose processors with a TEE). An attestation of execution provides information for verifying that the certificate was created within the enclave (and that the validator waited the allotted time). Further, the low cost of participation increases the likelihood that the population of validators will be large, increasing the robustness of the consensus algorithm.

Sawtooth also supports smart contracts (specifically, Ethereum smart contracts can be executed on Sawtooth). Performance-wise, Sawtooth scales well in terms of large numbers of transactions and nodes.

Popular blockchain use cases

Let's see some of the popular use cases for permissioned blockchains. It will help us to understand what enterprises can use permissioned blockchains for and what use cases are valid for permissioned blockchains.

Everledger

Everledger is a digital registry for diamonds powered by blockchain. It's an example of supply chain management on blockchain. Blockchain was used because, in blockchain, records are immutable. Everledger uses more than 40 features, including color and clarity, to create a diamond's ID. When this information is placed on blockchain, this information becomes a certificate chronicling the jewel's ownership, from mine to ring. Everledger has digitized more than a million diamonds and partnered with firms including Barclays. Participants in the blockchain network, such as merchants, banks, and insurers, can verify if a diamond is legitimate. Everledger is built on the Hyperledger Fabric platform. In the future, they are also planning to add other precious goods to their blockchain.

Let's take an example scenerio and see how blockchain helps in this use case. Alice purchases a diamond, insures it, and registers it on the Everledger blockchain. Next, she loses the diamond and reports it as stolen. The insurance company then compensates her for the loss. Finally, Bob the thief attempts to sell the stolen diamond to Eve the jeweller. She requests verification from Everledger and finds out that it's a stolen diamond. The insurance company is notified about the stolen diamond and they take possession of it.

Walmart's food tracking

Walmart's food tracking use case is a combination of blockchain and IoT to make a food product's history transparent and traceable to it's origin. It's an example of supply chain management on blockchain. Walmart's food tracking supply chain management is built on top of the Hyperledger Fabric platform.

A lot of people die every year due to food poisoning. As soon as someone falls sick or dies due to food poisoning the authorities try to track the source of the food and make sure that all the food items from the source that are distributed is suspended from selling and is called back. This saves lives of a lot of people. But the issue is that as every participation in the supply chain have their own ways and processes storing and retrieving information therefore it takes weeks for the authorities to track the source prevent everyone in the chain from selling the food items. Blockchain in combination with the IoT might just be able to solve this problem.

With every party in the supply chain storing and retrieving information, blockchain can fasten the process of finding the source of a food item. The following list shows additional benefits blockchain can add:

- Consumers can see exactly where a food product was harvested.
- Due to panic over food poisoning, people tend to throw away clean food, which increases the amount of food wasted. Blockchain can pinpoint the tainted food, therefore preventing food waste.
- Each step in the supply chain is visible to everybody. Fraudulent food entering the market can be avoided.
- Blockchain can act as evidence that a tainted food items was shipped from a particular producer. Due to this, producers will take care and adhere to safe practices because, if they don't, they will be caught with the evidence.
- Finally every food item gets a story associated with it. This enables users to learn about the food item's history.

IoT technology, such as sensors and RFID tags, enables real-time data to be written on the blockchain as food products pass along the supply chain.

Let's see an example of what the blockchain records in this case, and who the participants are. The participants are farms from where the food originates, factories where they are packed and processed, cargo companies who ship the food, Walmart stores, and so on. The data recorded on the blockchain is farm origin data, the batch number, factory and processing data, expiration dates, storage temperatures, and shipping details.

Ghana's land registry

BenBen is a team of research and development engineers dedicated to building innovative products to improve government technology in Ghana. They developed a digital land registry solution using blockchain for Ghana citizens.

In Ghana, banks don't accept land as a collateral when giving loans. That's because in Ghana, a paper registry system is unenforceable in court. This is preventing millions of people from getting loans.

BenBen provides a top-of-stack land registry and verification platform for financial institutions. This platform captures transactions and verifies the data. BenBen works with financial institutions to update current registries, enable smart transactions, and distribute private keys for clients, to allow automated and trusted property transactions between all parties.

Dubai's housing rental

Dubai's housing rental use case is a blockchain application that let's individual expats lease an apartment or renew their housing tenancy contract online within minutes. In Dubai, if an individual wants to take an apartment for rent, then they have to provide KYC documents, cheques as a contract-term guarantee, and create an Ejari (a government contract to legalize the otherwise unpleasant relationship between landlords and tenants in Dubai). In Dubai, most real estate companies rent apartments only if you want to stay for a longer period of time (for instance, at least a year) and to make sure you obey the contract, they ask you to provide postdated cheques as a guarantee, as in Dubai, a cheque bounce is considered a criminal offence. As the process of renting an apartment and renewing the tenancy contract is a cumbersome process for both tenants and real estate companies, **Dubai Smart Government** (**DSG**) (a technology arm of Smart Dubai, a city-wide initiative to transform Dubai technologically) launched a mission to make this whole process easier and quicker using blockchain.

This housing rental application was built using Hyperledger Fabric 1.0, and initially seven entities participated in the network. DSG, **General Directorate of Residency and Foreigners Affairs Dubai** (**DNRD**), wasl, **Dubai Land Department**, The **Dubai Electricity and Water Authority** (**DEWA**), Emirates **National Bank of Dubai** (**NBD**), and **Emirates Islamic** (**EI**) bank were the entities who shared their data on blockchain to make the tenancy contract creation and renewing easier.

Earlier, DSG and the **Emirates Identity Authority** (**EIDA**) launched DubaiID, which allowed Dubai residents a unified access to all eServices provided by government agencies through one login, and interaction with them via the internet. In this blockchain use case, the tenant had to log in to the real estate's portal using DubaiID; in this case, wasl's tenant must have a DubaiID to login. Once logged in, SDG will write the Emirates ID number into blockchain, and DNRD shares visa and passport information on the blockchain for that tenant. Then, wasl's portal redirects users to submit digital cheques using an Emirates NBD or EI bank account. Once digital cheques are submitted, a request is made to DLD via blockchain to renew or create an Ejari. Finally, once Ejari processing is done, DEWA is notified to activate the water and electricity supply. So basically, the first pilot was for individuals who wanted to lease or renew a wasl apartment and had a bank account with Emirates NBD or EI. Soon, more banks and real estate companies will be added to the network to provide this service for more people in Dubai. In this process, it was ensured that a piece of information can only be seen by the concerned parties.

This use case fits in well as a blockchain use case because a signed immutable ledger was required to store KYC, cheques, and Ejaris and the latter can be proved if the customer or any entity tries to commit fraud. For example, when Emirates NBD issues cheques, if they do it without blockchain and simply make point-to-point API calls, then there is a very good possibility of intentional and unintentional disagreement between ENBD, the tenant, and wasl regarding the existence of a digital cheque or its current status. Therefore, blockchain can be the final tool for reference if any dispute occurs.

Project Ubin

Project Ubin is a digital cash-on-ledger project run in partnership between **Monetary Authority of Singapore** (**MAS**) and R3, with the participation of **Bank of America** (**BOA**) **Merrill Lynch**, **Credit Suisse**, DBS bank, **The Hongkong and Shanghai Banking Corporation Limited**, J.P. Morgan, **Mitsubishi UFJ Financial Groupb** (**MUFG**), OCBC bank, **Singapore Exchange** (**SGX**), and **United Overseas Bank** (**UOB**), as well as **BCS Information Systems** as a technology provider.

The aim of Project Ubin is building a digitalized form of the SGD (Singapore's national currency) on a distributed ledger to bring many benefits to Singapore's financial ecosystem. The benefits would be the same as that of any other cryptocurrencies.

Currently, this application is built using Quorum, but in future it may move to Corda as R3 is one of the partners.

MAS is Singapore's central bank and financial regulatory authority. MAS acts as a settlement agent, operator, and overseer of payment, clearing, and settlement systems in Singapore that focus on safety and efficiency.

Summary

In this chapter, we learned what DApps are and got an overview of blockchain-based DApps. We saw what a blockchain is, what its benefits are, and saw various platforms that we can use to build our own blockchain-based DApps. Finally, we saw some use cases and how blockchain can bring change to the financial and non-financial industry. In the next chapter, we will get into Ethereum and Quorum, and build a basic example DApp.