



**UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH**

**Escola Tècnica Superior d'Enginyeria
de Telecomunicació de Barcelona**

**DEPLOYMENT OF A DECENTRALIZED APPLICATION OVER A
PERMISSIONED BLOCKCHAIN BASED ON BESU**

A Master's Thesis

Submitted to the Faculty of the

**Escola Tècnica d'Enginyeria de Telecomunicació de
Barcelona**

Universitat Politècnica de Catalunya

by

Marc Muro Barbé

In partial fulfilment

of the requirements for the degree of

MASTER IN CYBERSECURITY

Advisor: José Luís Muñoz



UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH



telecos
BCN

Barcelona, September 2021



Title of the thesis: Deployment of a decentralized application over a permissioned blockchain based on Besu

Author: Marc Muro Barbé

Advisor: José Luís Muñoz



Abstract

The transparent nature of for public chains like, allows all the participants to view the transactions, limiting the potential of developing blockchain-based applications. This paper attempts to study and deploy an Hyperledger Besu Network with Orion. Orion is a promising private transaction manager, which as the name indicates, keeps transactions in a Blockchain network private. Furthermore, we will deploy the network using Docker in order to make it easier to replicate the deployment of the network in other systems.



Acknowledgements

I want to thank my advisor, José Luís Muñoz, for all the help and support that I received from him. Moreover, I want to thank every member of the Hyperledger Besu community that has left detailed planned and explanations to make sure that even newer people on the field can understand and solve all their problems.

Revision history and approval record

Revision	Date	Purpose
0	31/08/2021	Document creation
1	10/09/2021	Document revision
2	19/09/2021	Document revision
3	26/01/2021	Document revision

Written by:		Reviewed and approved by:	
Date	31/08/2021	Date	26/09/2021
Name	Marc Muro Barbé	Name	José Luís Muñoz
Position	Project Author	Position	Project Supervisor



Table of contents

DEPLOYMENT OF A DECENTRALIZED APPLICATION OVER A PERMISSIONED BLOCKCHAIN BASED ON BESU	0
Abstract	2
Acknowledgements	3
Revision history and approval record.....	4
Table of contents	5
List of Figures.....	7
List of Tables	8
1. Introduction.....	9
1.1. Statement of purpose	9
1.2. Requirements and specifications	10
1.3. Methods and procedures	10
1.4. Work Plan.....	11
1.5. Gantt Diagram	13
1.6. Derivations	16
2. State of the art of the technology used or applied in this thesis.....	17
2.1. Enterprise Ethereum.....	17
2.2. Hyperledger Besu.....	17
2.2.1 Hyperledger Besu Architecture	18
2.2.2 Privacy	20
2.2.3 Permissioning	22
2.3. Orion	25
2.3.1. Privacy groups.....	26
2.3.2. Processing Private Transactions	28
2.3.3. Multi-Tenancy.....	29
2.4. Docker.....	30
3. Methodology / project development	31
3.1. Introduction to the project	31
3.2. First test: Run Besu from a Docker Image	31
3.3. Second Test: Create a permissioned network	36
3.4. Third Test: Create a permissioned network with Docker	38



3.5. Fourth Test: Create an Hyperledger Besu Network with Docker Compose to send transactions.....	40
3.6. Fifth Test: Create an Hyperledger Besu Network with Docker Compose	43
3.6.1. Fifth Test: Problems and Solutions	46
4. Results	47
4.1. First Network	47
4.2. Fifth Network	51
4.3. Modifying the Network	53
5. Budget.....	54
6. Conclusions and future development.....	55
7. Bibliography.....	56
8. Glossary	59

List of Figures

Figure 1 Private Transaction Manager	10
Figure 2 Hyperledger Besu's Architecture	19
Figure 3 Alice sends a transaction to Bob using Orion in an Hyperledger Besu Network	21
Figure 4 Node-level permissions are a useful system of governance to control connections to an individual node.....	22
Figure 5 Using account permissions a consortium blockchain can limit which accounts a node accepts transactions from, and which it rejects.	23
Figure 6 Local and onchain permissioning rules in Hyperledger Besu	24
Figure 7 Orion supports TLS communication	26
Figure 8 Privacy groups in Hyperledger Besu	27
Figure 9 Private transaction processing in Hyperledger Besu	28
Figure 10 Multi-Tenancy in Hyperledger Besu with Orion	29
Figure 11 Docker's containers compared to a virtual machine	30
Figure 12 Block Explorer	32
Figure 13 Block Explorer block searcher.....	32
Figure 14 Grafana's user interface.....	34
Figure 15 Volumes in Docker.....	39
Figure 16 Test 5 network schematic	43
Figure 17 Quorum developer quickstart.....	47
Figure 18 Launching an Hyperledger Besu network using Docker part 1/2	48
Figure 19 Launching an Hyperledger Besu network using Docker part 2/2	48
Figure 20 List of available services in Hyperledger Besu	48
Figure 21 Block Explorer of our Hyperledger Besu network	49
Figure 22 Best Block in our Hyperledger Besu network	49
Figure 23 Grafana's Dashboard of our Hyperledger Besu network	50
Figure 24 Launching our Hyperledger Besu network with Orion	51
Figure 25 Removing the network	52
Figure 26 Network functioning	52
Figure 27 Virtual Machine cost calculator	54



List of Tables

Table 1 Node's addresses and keys45

1. Introduction

1.1. Statement of purpose

This objective of this project is to deploy an Hyperledger Besu Blockchain network in a virtual environment. We will use Docker and Docker compose to launch the network so there are no compatibility problems when someone wants to replicate the launched network somewhere else.

Regarding the Network, it will be a three-node network with privacy (Orion) and monitored with Prometheus and Grafana. The privacy part is one of the main driving parts of this project. We also studied the options of adding or removing nodes by modifying the docker compose file and checked that the network worked without issues.

Why a privacy layer in blockchain?

While blockchain allows users to collaborate with new trust models, businesses do not trust each other to know the details of their transactions. Moreover, blockchain does not remove the regulations for businesses to keep data and transactions private. Most data must be kept private to those with a business need-to-know. [1]

Processing private transactions involves both the private transaction and the privacy maker transaction. The private transaction is distributed to involved participants and the privacy marker transaction is included on the public blockchain.

Private transactions are not submitted in a transaction pool. Private transactions are distributed directly to participants in the transaction while the privacy marker transaction is distributed to the transaction pool.

Private transaction managers such as Orion allow shared business logic in smart contracts. They turn all transactions and their respective states private. The installation of the contract is also recorded on the main blockchain via a hash. Participants execute the private transaction, and update their private data, as the hash is ordered and distributed via the blockchain network.

Figure 1 is an example of how a privacy transaction manager works.

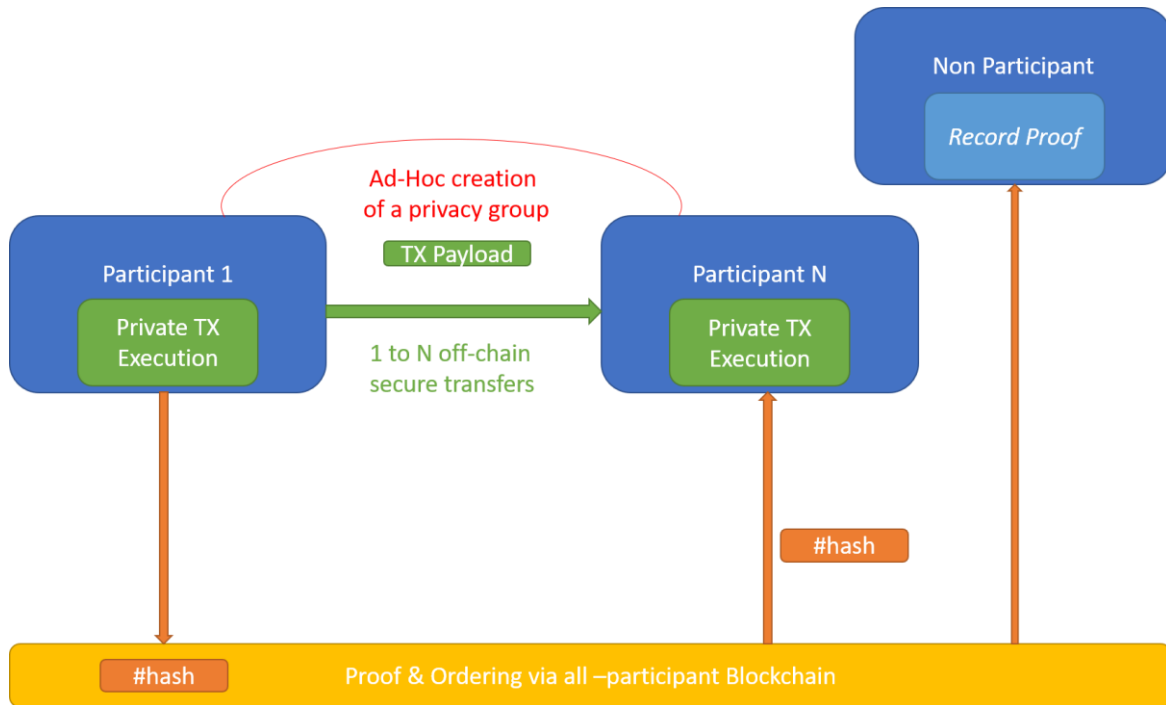


Figure 1 Private Transaction Manager

1.2. Requirements and specifications

During the first month of the project, I had to learn about Hyperledger, Besu and Orion. To do so I used different resources but mostly the Hyperledger Besu Wiki and documentation pages as well as Orion's.

Then on the following weeks I had to learn about Docker and Docker Compose. From why are they used to how.

As for other requirements a basic knowledge of Linux and Latex were needed as well.

1.3. Methods and procedures

This project studied different methods to deploy an Hyperledger Besu Network with Orion as a privacy method. We tested different networks until we found a GitHub with multiple sample networks. One of such samples was a three-node network with Orion. It wasn't as simple as just launching the network, we had to study it first and fix the errors that occurred. Furthermore, we needed to test the network with different smart contracts and studied the possibility of modifying the network to ass or remove nodes.

1.4. Work Plan

Project: Learning about Hyperledger and Orion	WP ref :WP 1
Reading and understanding Hyperledger, Besu and Orion. Learning the main differences between Orion and Tessera.	Start date: 14/02 End Date: 17/03

Project: Learning about Docker and Docker Compose	WP ref :WP 2
Reading and understanding Docker and Docker compose; the main commands and how they work.	Start date: 17/03 End Date: 31/03

Project: Investigating how to launch Hyperledger Besu using Docker	WP ref :WP 3
Using the Hyperledger Besu Documentation to try to launch Hyperledger using docker and then with Orion.	Start date: 31/03 End Date: 07/05

Project: Investigating how to launch Hyperledger Besu with Orion using Docker Compose	WP ref :WP 4
Using different projects on Hyperledger from GitHub to try to launch Hyperledger Besu with Orion using docker compose.	Start date: 07/05 End Date: 28/05



Project: Fixing the problems of the Docker Compose Network	WP ref :WP 5
Solving the problems that arose when we try to launch the Network. These issues included compatibility and connection problems.	Start date: 15/05 End Date: 28/05

Project: Testing and Modifying the Network	WP ref :WP 6
Testing the network and experimenting if the network can be modified to fit different needs that may arise in the future.	Start date: 28/05 End Date: 06/08

1.6. Derivations

Finding a Docker Hyperledger Besu network that worked with Orion took more time than expected and the progress on my end was slower than expected. Moreover, some errors appeared when we tried to test the different networks which took some time to solve.

2. State of the art of the technology used or applied in this thesis

The project consists on launching an Hyperledger Besu Blockchain Network that uses Orion. Hyperledger Besu is an open-source Ethereum client developed with an Apache 2.0 license. Besu implements Proof of Work and Proof of consensus mechanisms.

Orion is an open-source private transaction manager developed with Apache 2.0 license. The main application of Orion is as the private transaction manager for Hyperledger Besu.

Hyperledger Besu is a solution that allows enterprises build scalable, high-performance applications on a private network Furthermore it comes with the support of permissioning and privacy.

2.1. Enterprise Ethereum

Enterprise Ethereum is a private variation of the Ethereum blockchain. Enterprise Ethereum not only works as a normal Ethereum network does but adds permission features along with identity management to offer a permissioned network. Moreover, it increases the privacy level in order to increase scalability and performance. It even introduces different consensus algorithms other than proof of work. [2]

In conclusion, an Enterprise Ethereum can do the following:

- Create an execution environment in Ethereum blockchain for processing transactions
- Persistent data storage including storing transaction execution
- Enable peer-to-peer (P2P) network communication between nodes
- Offers APIs for secure development and blockchain interaction
- JSON RPC APIs for application developers to interact with the blockchain

2.2. Hyperledger Besu

Hyperledger Besu is an open-source Ethereum client that is developed with an Apache 2.0 license. It written in Java and makes use of the Ethereum public network. [3]

As an Ethereum client, Hyperledger Besu can be used the same things an Ethereum network is capable of: [4]

- Decentralized app (dApp) development
- Smart contract development
- Ether mining

As for the consensus method, Hyperledger Besu utilizes Proof of Authority (Clique and IBFT 2.0) and Proof of Work (Ethash).

Hyperledger Besu is the ideal network for enterprises as it was aimed towards building a private and public network for enterprises.

The Notable Features of Hyperledger Besu are: [2] [4]

- Ethereum Virtual Machine (EVM): Enables deployment and execution of smart contracts through transactions within an Ethereum blockchain.
- Consensus protocols: The consensus algorithms of Besu are Proof-of-Authority and Proof-of-work. These consensus algorithms are involved in block validation, transaction validation, and block production.
- Storage: Hyperledger Besu uses a key-value database for persisting chain data locally.
- Monitoring: Node and network performance can be monitored using Besu. Nodes are monitored through Prometheus or Grafana. Network performance is monitored using Block Explorer.
- Privacy: The private transaction manager of Besu, in our case Orion, implements privacy and does not allow other parties to access the transaction content, sending party, and the list of participating parties.
- Permissioning: Besu allows only specific nodes and accounts to participate in the network.

2.2.1 Hyperledger Besu Architecture

We will now discuss some of the significant elements of the Hyperledger Besu architecture.

The key three core components of the Besu include the following: [5]

- Storage
- Ethereum Core
- Networking

Figure 2 showcases Hyperledger Besu’s architecture.

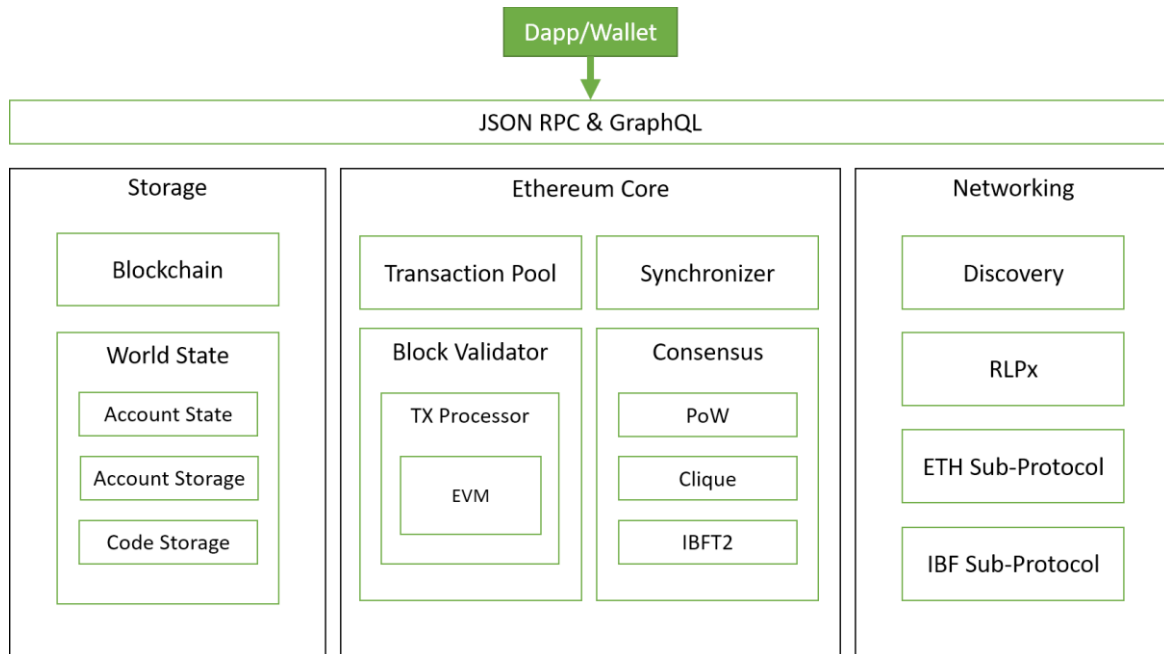


Figure 2 Hyperledger Besu's Architecture

Ethereum Core

The main component of the Ethereum core is the Ethereum Virtual Machine which is responsible for any executing any transaction. The Transaction Processor helps the Ethereum Virtual Machine to function efficiently and effectively, the Transaction Pool stores transaction-related information and the Synchronizer helps to synchronize all the nodes and the network. [2]

Consensus methods are also a part of the Ethereum core. Here we have consensus algorithms including PoW, Clique, and IBFT2. [4]

Regarding PoW, Ethash is a proof-of-work mining algorithm implemented by the Ethereum network. Ethash was developed with protection from ASIC (Application Specific Integrated Circuits) in mind.

Besu implements Clique Proof-of-Authority consensus protocol. In Clique networks, approved accounts, which are also known as signers, validate transactions and blocks. Signers take turns to create the next block. The signers that are already on the network can propose and vote to add or remove signers. [6]

Besu also implements the IBFT 2.0 Proof-of-Authority consensus protocol. In IBFT 2.0 networks, approved accounts, also known as validators, can validate transactions and blocks. Just like signers in clique, validators take turns to create the next block. Before inserting the new block onto the chain; a majority, that has to greater than 66% of validators, must first sign the block. IBFT 2.0 is a robust and stable consensus algorithm suitable for enterprise use cases in a private network. [7]

While the core client architecture is relatively simple, it can be easily expanded with plugins. Users can use existing open-source Besu plugins or build their own plugins. This approach is what enables Hyperledger Besu’s to be modular.

Storage and Networking

In a storage setup, the data is divided into two sub-categories, Blockchain and World State.

The Blockchain is made of block headers, block bodies and transaction receipts. Block headers are used to verify the blockchain stat. Block bodies contain a list of ordered transactions and are included in each block. Transaction receipts contain metadata related to the execution of transactions, including transaction logs. [2] [4]

Every block header references a world state through a hash. World state is a map from addresses to accounts.

On the Networking part, Besu implements Ethereum's devp2p network protocols for inter-client communication and a sub-protocol for IBFT2. The main components include: Discovery, a UDP-based protocol to find users on the network; RLPx, A TCP-based protocol that is used for communication between peers through sub-protocols; ETH sub-protocol; and an IBF Sub-Protocol.

2.2.2 Privacy

In Besu, privacy refers to the ability to keep transactions private between the participants in the network. Other participants will not be able to access the transaction content or list of participants. [8]

Besu uses a private transaction manager, in our case Orion, to implement privacy. Each Besu node that sends or receives private transactions will require an associated Orion node.

In figure 3 we can see how Alice sends a transaction to Bob using Orion in Hyperledger Besu.

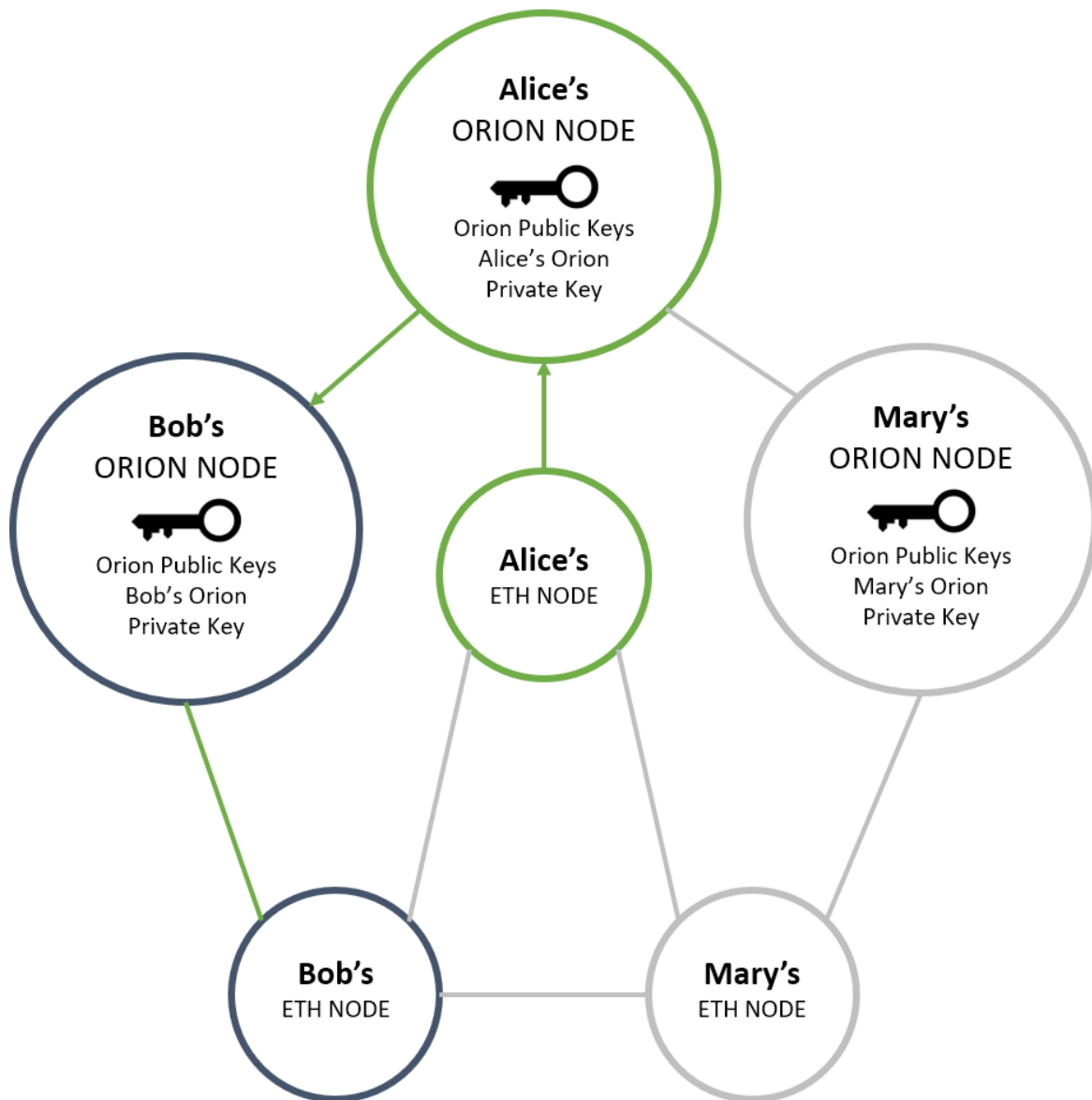


Figure 3 Alice sends a transaction to Bob using Orion in an Hyperledger Besu Network

Private transactions pass from the Besu node to the associated Orion node. The Orion node then proceeds to encrypt and distribute directly the private transaction to the other Orion nodes participating in the transaction.

Each participant in a privacy network uses its own Besu and Orion node. Multi-tenancy allows more than one participant to use the same Besu and Orion node.

Besu and Orion nodes both have public and private key pairs identifying them. A Besu node sends a private transaction to an Orion node so it signs the transaction with the Besu node private key.

We will expand on Privacy when we talk about Orion.

2.2.3 Permissioning

A permissioned network allows only specified nodes and accounts to access the network by enabling node permissioning and account permissioning. [9]

In order to obtain permissioning we need a distributed network of trust across the network where participants must agree to follow the rules. If one bad actor decides not to follow the rules, the other nodes can take action to prevent the bad actor adding to the chain but they cannot prevent the bad actor from allowing access to the chain.

Node permissioning

Node permissioning is used to restrict access to known participants in case a bad actor appears. [9]

Node-level permissions are a useful system of governance to control connections to an individual node.

We can see how node-level permissioning works in Hyperledger Besu in figure 4.

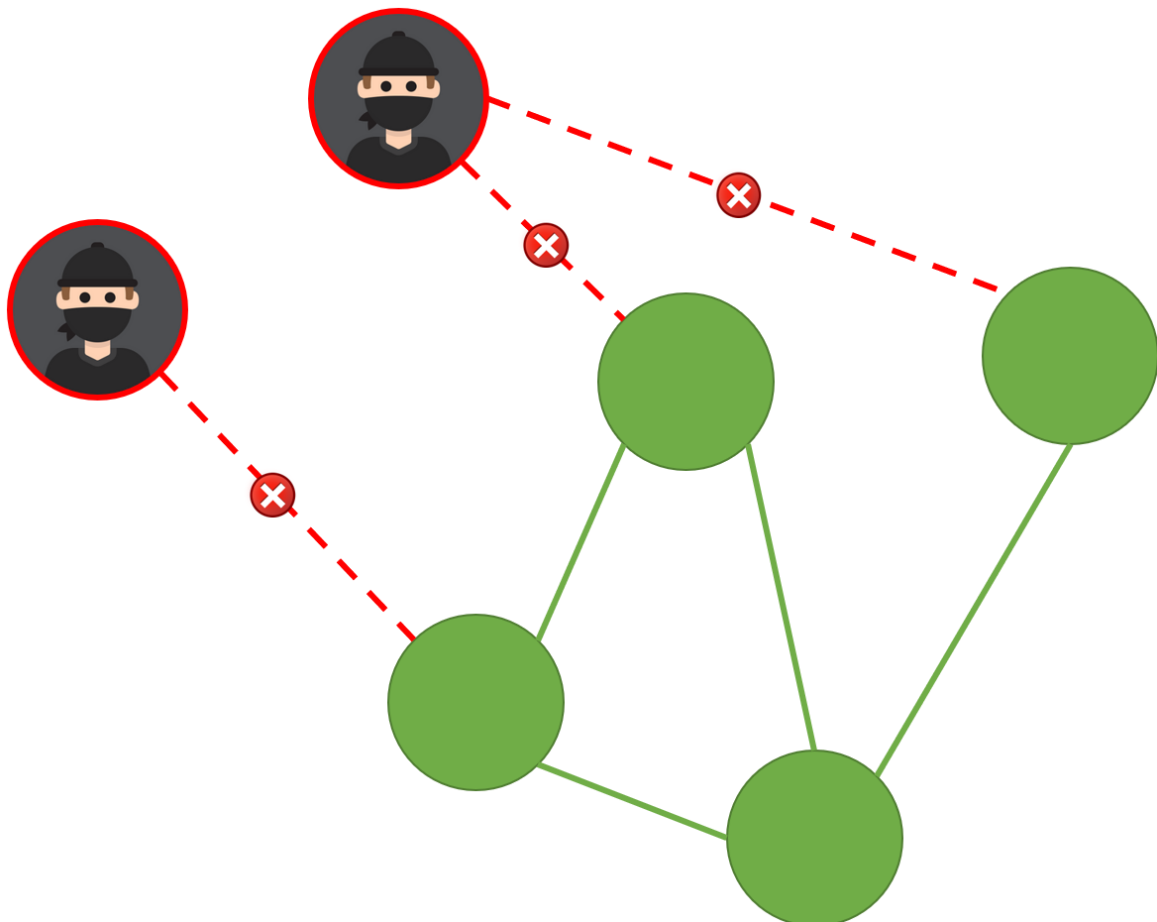


Figure 4 Node-level permissions are a useful system of governance to control connections to an individual node.

Account permissioning

Account permissioning is used to enforce onboarding or identity requirements, suspend accounts, exclude broken contracts using a deny-list and restrict the actions an account can perform. [9]

Using account permissions in Hyperledger Besu, a consortium blockchain can limit which accounts a node accepts transactions from, and which it rejects.

In figure 5, we can observe account permissioning in Hyperledger Besu.

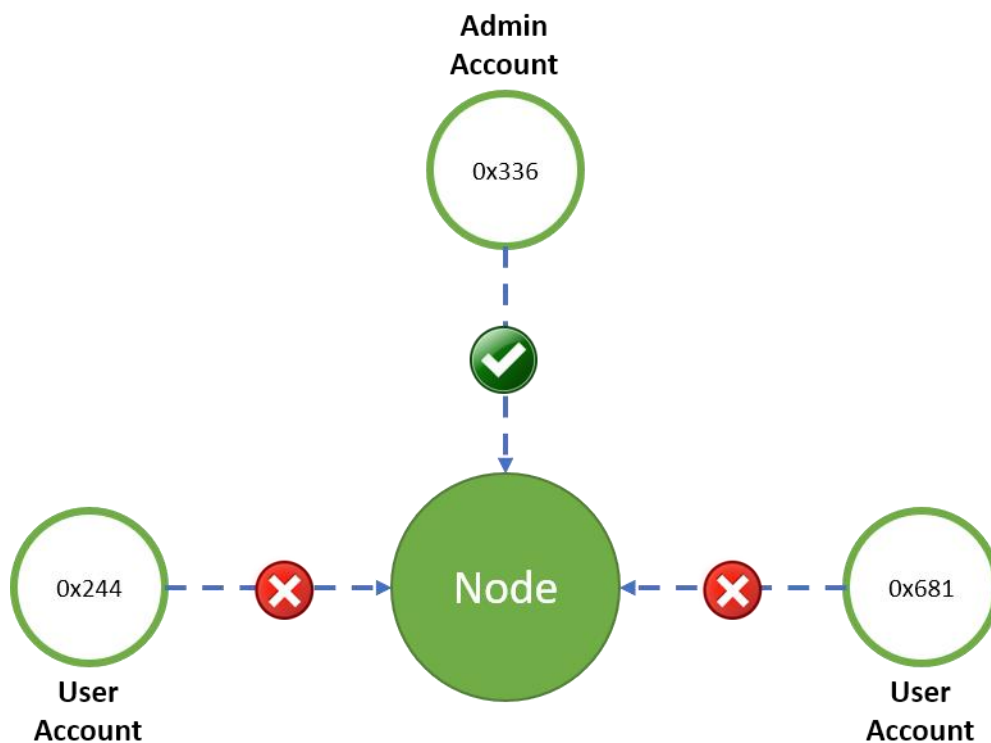


Figure 5 Using account permissions a consortium blockchain can limit which accounts a node accepts transactions from, and which it rejects.

Specifying permissioning

You can specify permissioning locally or onchain. [9]

The following diagram, figure 6, illustrates applying local and onchain permissioning rules.

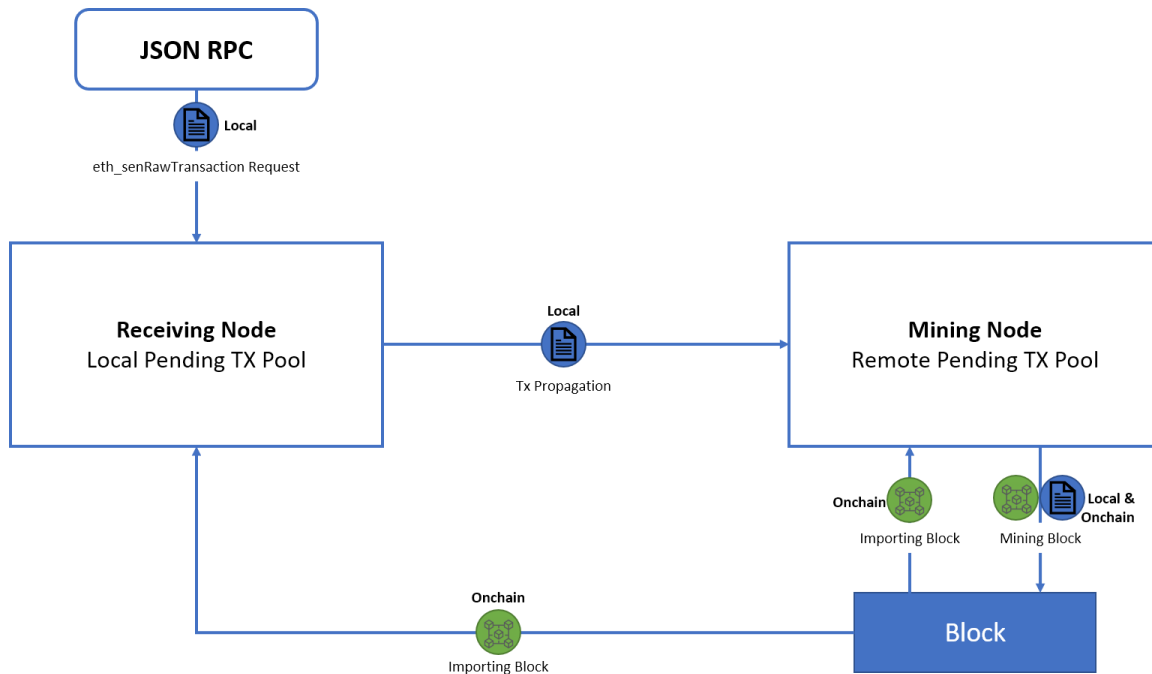


Figure 6 Local and onchain permissioning rules in Hyperledger Besu

Local permissioning works at the node level to do so, each node in the network needs to have a permissions configuration file.

Local permissioning affects a node but not the rest of the network. Local permissioning is used to restrict use of a node. For example, customers able to access your node.

As local permissioning does not require co-ordination with the rest of the network, the user can act immediately to protect their node.

Onchain permissioning works on the network through a smart contract. Using smart contracts, onchain permissioning stores and administers the nodes, accounts, and admin allowlists. Using onchain permissioning enables all nodes to read the allowlists from a single source, the blockchain. [10]

Onchain permissioning enables all nodes to read and update permissioning configuration from one location. To do so, Onchain permissioning requires co-ordination to update the rules.

When you onchain permissioning is updated, the new rules will apply across the network and new blocks automatically abide by the updated rules.

2.3. Orion

Orion is an open-source private transaction manager developed under the Apache 2.0 license and written in Java. [11]

Orion is mainly used as the private transaction manager for Hyperledger Besu. Orion generates and maintains private/public key pairs, stores privacy group details, self manages and discovers all nodes in the Orion network and provides an API for communicating between Orion nodes and an API for communicating with Ethereum clients like Hyperledger Besu.

Besu and Orion use separate keys and nonces for the private part of the transaction. Orion uses different transaction signing keys for the private payload of the transaction and the transaction that is sent to the main blockchain. [1] [12]

Orion uses three different keys:

- The Ethereum key that signs the private transaction
- The Ethereum key that signs the public transaction
- The key that is used to encrypt and transfer the payload to remote participants in the transaction

Besu and Orion use a Smart Contract address on the main chain to track transactions. The transactions that appear on the main blockchain network for Orion will all have the same to address. This address is a special address of a special smart contract that records all of the transactions.

Besu and Orion require explicit management of privacy groups. When submitting transactions to Orion, the target for the private transaction will be a privacy group.

Privacy groups have lifecycles and identities. The existence of privacy groups is broadcast to all participants. Each privacy group is a separate isolated piece of state. [13]

The consequences of this are:

- Multiple privacy groups may have the same sets of participants
- The privacy group must be stated when submitting transactions as well as when reading state

In order to enable disaster recovery, Orion uses a relational database and backup. Orion supports using relational databases PostgreSQL and Oracle. Were the case to happen where the Orion database is deleted or corrupted, all private transaction payloads for the node will be lost. And without a backup users will not be able to recover a lost database. [14]

Furthermore, Orion supports TLS to secure communication to a client and to other Orion nodes. The client, in our case Hyperledger Besu, must be configured to send and accept TLS communication between itself and Orion. [15]

We can observe an Hyperledger Besu Network with Orion using TLS communication in figure 7.

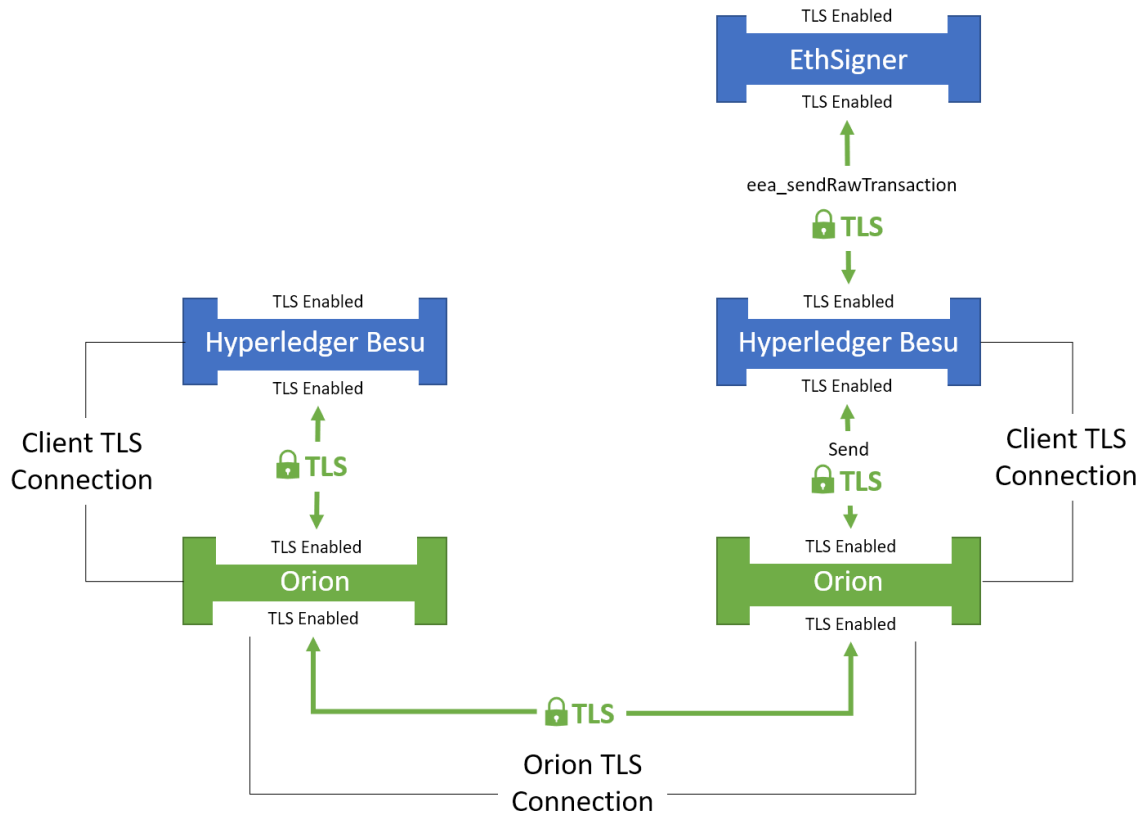


Figure 7 Orion supports TLS communication

Orion also supports multi-tenancy. Usually, every participant in a privacy network uses its own Hyperledger Besu and Orion node. Orion supports multi-tenancy by allowing multiple participants in a privacy network to use the same Besu and Orion node. [16]

2.3.1. Privacy groups

A privacy group is a group of nodes identified by a unique privacy group ID by Orion. Orion stores each private transaction with the privacy group ID. [17]

The Besu nodes maintain the public world state for the blockchain and a private state for each privacy group. The private states contain data that is not shared in the globally replicated world state.

Besu implements two types of privacy: Enterprise Ethereum Alliance (EEA) privacy and Besu-extended privacy. Both privacy types create privacy groups and store private transactions with their privacy group in Orion.

A contract in a privacy group can read or write to a contract in the same privacy group, can read from the public state including public contracts but cannot access contracts from a different privacy group as a public contract cannot access a private contract.

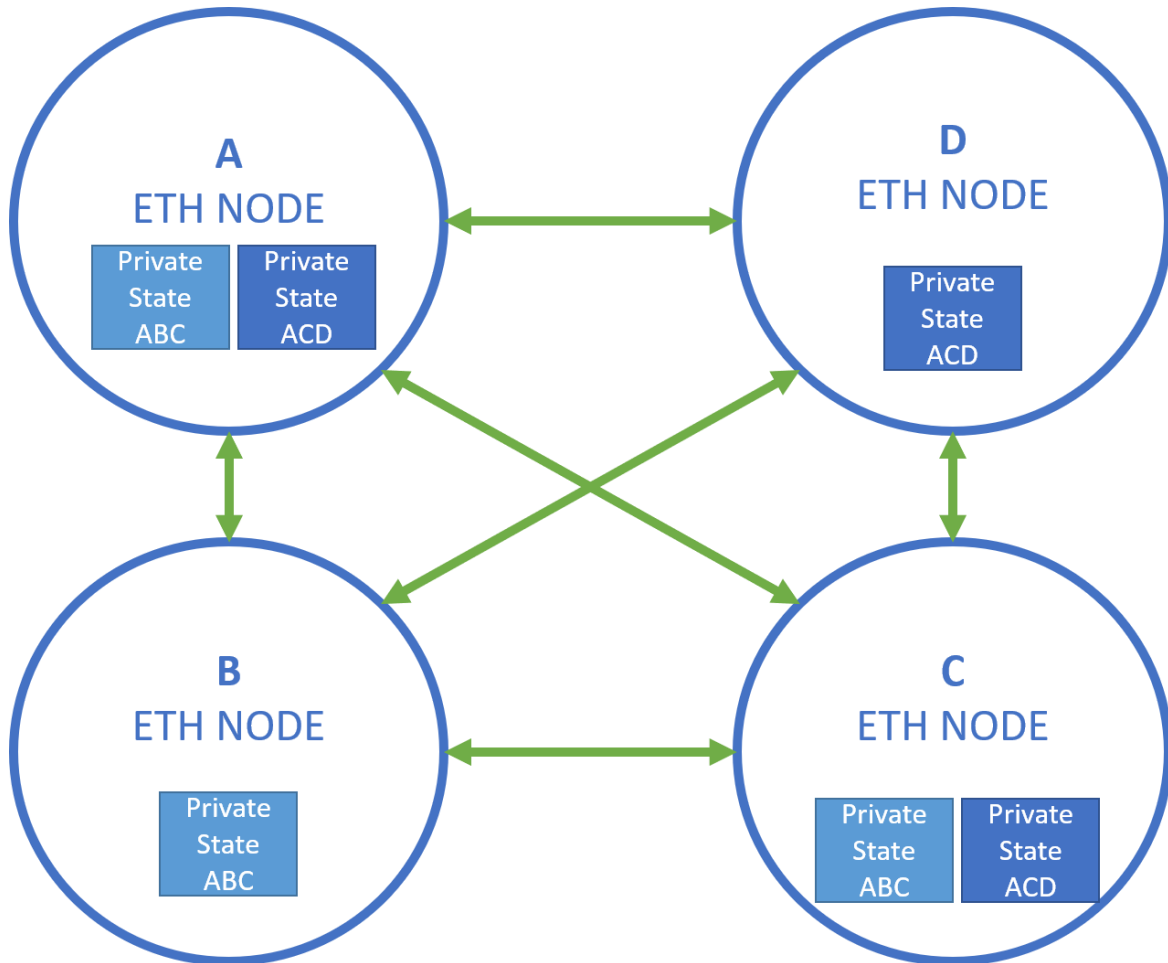


Figure 8 Privacy groups in Hyperledger Besu

Figure 8 illustrates two privacy groups enabling:

- A, B, and C to send transactions that are private from D.
- A, C, and D to send transactions that are private from B.

In the previous diagram, to send private transactions between A, B, and C, A creates a privacy group consisting of A, B, and C. The privacy group ID is specified when sending private transactions and A, B, and C are recipients of all private transactions sent to the privacy group.

To send private transactions between A, C, and D, A creates a privacy group consisting of A, C, and D. The privacy group ID of this group is specified when sending private transactions with A, C, and D as recipients.

2.3.2. Processing Private Transactions

Processing Private transactions involves two things: [18]

- **Precompiled Contract:** A precompiled contract is a smart contract compiled from the source language and stored by an Ethereum node to execute later.
- **Privacy Marker Transaction:** A privacy marker transaction involves a public Ethereum transaction with a payload of the enclave key. The enclave key points to the private transaction in Orion.

Private transaction processing is illustrated in figure 9.

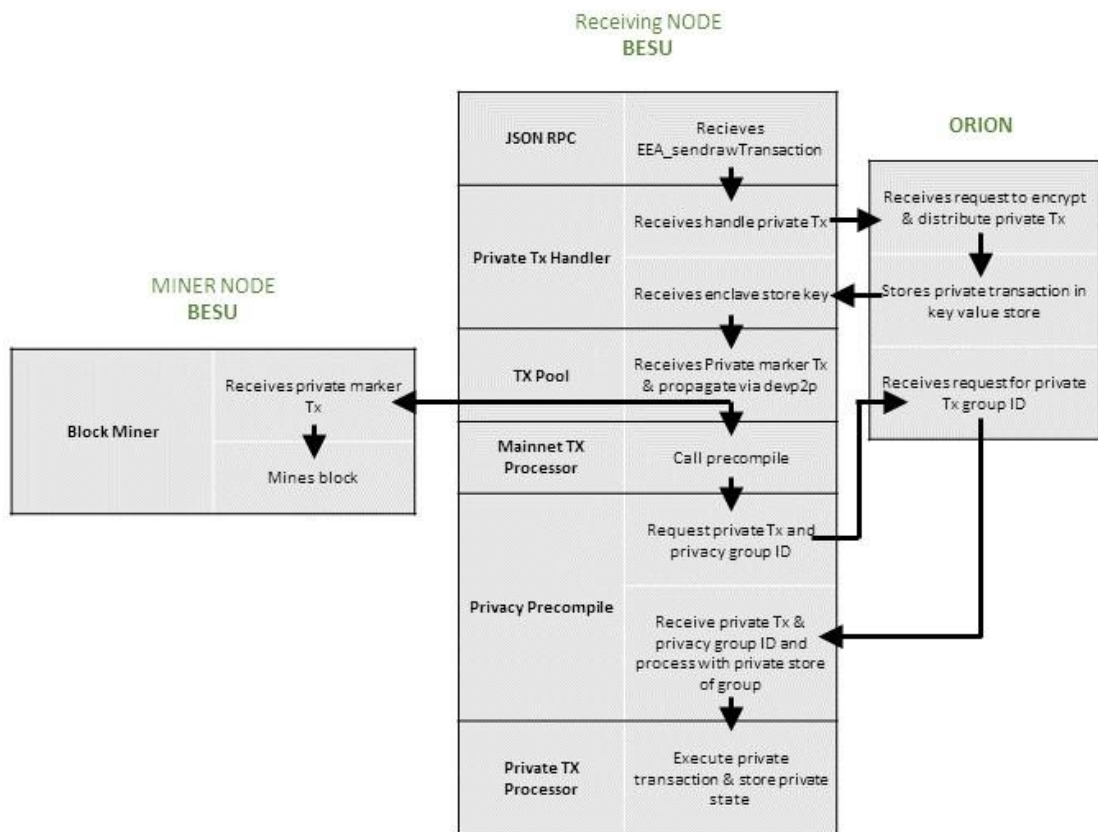


Figure 9 Private transaction processing in Hyperledger Besu

We will explain step by step the diagram:

1. The process starts by submitting a private transaction using `eea_sendRawTransaction`. The signed transaction includes transaction attributes that specify the list of recipients, sender and if the transaction is of type restricted.
2. The private transaction is passed to the Private Transaction Handler.
3. The Private Transaction Handler sends the private transaction to Orion.
4. Orion distributes the private transaction directly to the Orion nodes previously specified or that belonging to a previously specified privacy group. All recipient

Orion nodes store the transaction. Orion associates the stored transaction with the transaction hash and privacy group ID.

5. Orion returns the transaction hash to the Private Transaction Handler.
6. The Private Transaction Handler creates a Privacy Marker Transaction for the private transaction. Just like in an Ethereum transaction, the Private Transaction Handler propagates the Privacy Marker Transaction.
7. Besu mines the Privacy Marker Transaction into a block and the Privacy Marker Transaction is distributed to all Ethereum nodes in the network.
8. The Mainnet Transaction Processor processes the Privacy Marker Transaction in the same way as any other public transaction.
9. Orion is queried by the privacy precompile contract for the private transaction and privacy group ID using the transaction hash.
10. The privacy precompile contract passes the private transaction to the Private Transaction Processor. The privacy group ID specifies the private world state to use.
11. The Private Transaction Processor executes the transaction.

2.3.3. Multi-Tenancy

As previously stated, each participant in a privacy network uses its own Besu and Orion node.

Multi-tenancy allows multiple participants to use the same Besu and Orion node. Each participant is a tenant, and the operator is the owner of the Besu and Orion node. [19]

In figure 10 we can observe

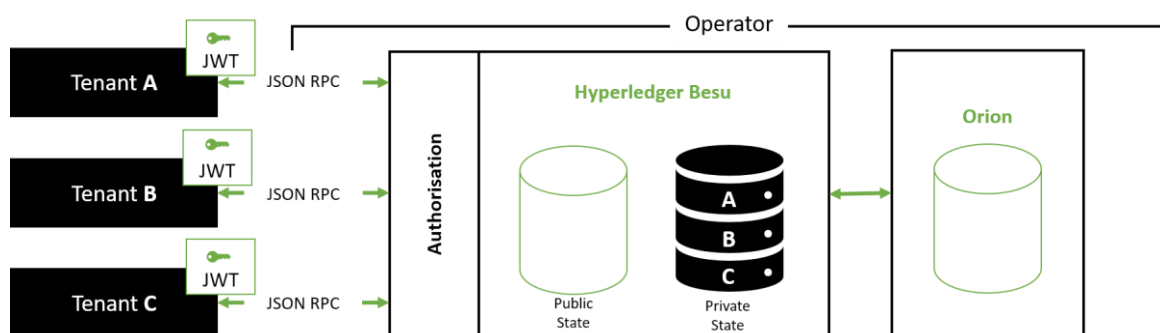


Figure 10 Multi-Tenancy in Hyperledger Besu with Orion

Multi-tenancy validates that tenants have permission to use the specified HTTP or Websocket JSON-RPC requests, and the tenant has access to the requested privacy data.

2.4. Docker

Docker is an open-source project that automates the deployment of software applications inside containers by providing an additional layer of abstraction and automation of OS-level virtualization on Linux. In simpler words, Docker is a tool that allows its users to easily deploy their applications in a sandbox, called containers, to run on the host operating system. The main benefit of Docker is that it allows its users to store an application, including all of its dependencies, into a standardized unit. [20]

By leveraging the low-level mechanics of the host operating system, containers provide most of the isolation of virtual machines at a fraction of the computing power.

In figure 11 we can observe the difference in the structure of a Docker Container with that of a virtual machine

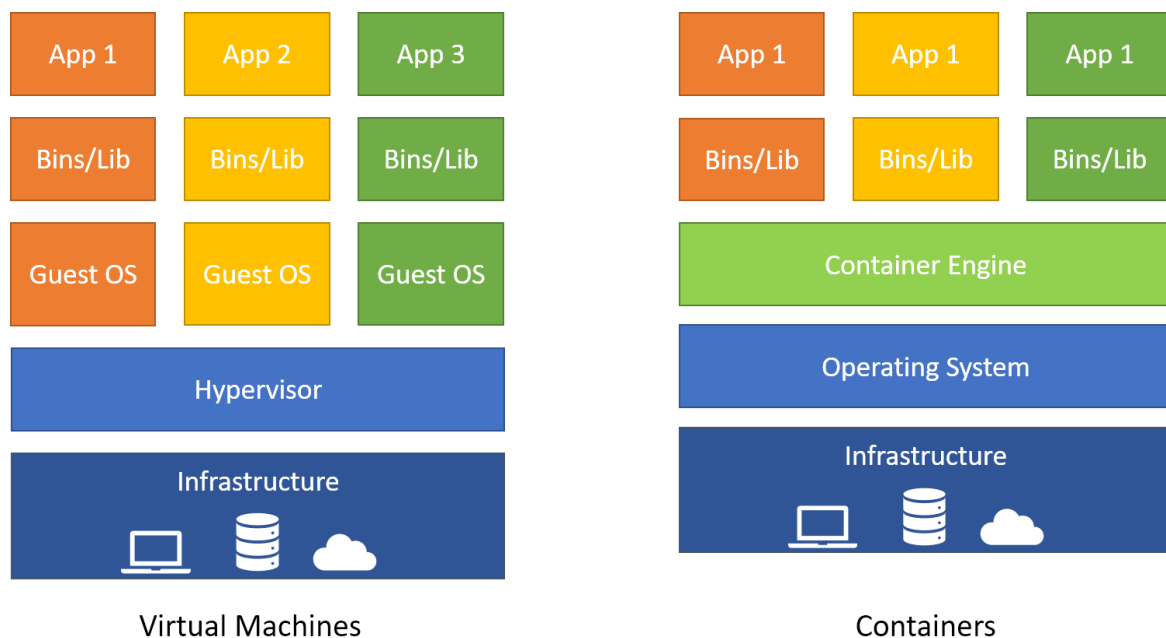


Figure 11 Docker's containers compared to a virtual machine

Docker Compose

There are a bunch of other open-source tools which combine well with Docker. One of said tools is Docker Compose. [21]

Compose is a tool that is used for defining and running multi-container Docker apps in an easy way. It provides a configuration file called docker-compose.yml that can be used to bring up an application and the suite of services it depends on with just one command. Compose works in all environments: production, staging, development, testing, as well as CI workflows, although Compose is ideal for development and testing environments.

3. Methodology / project development

3.1. Introduction to the project

My goal was to launch a functional Hyperledger Besu Network on a Virtual Machine with use Orion as a privacy transaction manager. The network had to be launched using Docker in order to avoid complications if the need to launch the network in another environment arises.

3.2. First test: Run Besu from a Docker Image

Besu provides a Docker image to run a Besu node in a Docker container. We used this Docker image to run a single Besu node without installing Besu. [22]

Prerequisites

- One of the following operating systems is required:
 - Linux on x86_64 architecture
 - macOS on an Intel processor (M1 processor not supported yet)
 - Windows 64-bit edition, with:
 - Windows Subsystem for Linux 2.
 - Docker desktop configured to use the WSL2-based engine.
- Docker and Docker-compose.

Generate the tutorial blockchain configuration files

To create the tutorial docker-compose files and artifacts we run:

- `npx quorum-dev-quickstart`

Start the network

To start the services and the network we launch a terminal in the installation directory and run:

- `./run.sh`

The script builds the Docker images, and runs the Docker containers. Four Besu IBFT 2.0 validator nodes and a non-validator node are created to simulate a base network.

When execution is successfully finished, the process lists the available services:

- Use the JSON-RPC HTTP service endpoint to access the RPC node service from your dapp or from cryptocurrency wallets such as MetaMask.
- Use the JSON-RPC WebSocket service endpoint to access the Web socket node service from your dapp.
- Use the Web block explorer address to display the block explorer Web application.

- Use the Prometheus address to access the Prometheus dashboard. Read more about metrics.
- Use the Grafana address to access the Grafana dashboard. Read more about metrics.
- Use the Kibana logs address to access the logs in Kibana. Read more about log management.

To display the list of endpoints again, we run:

- `./list.sh`

Block Explorer

The block explorer displays a summary of the private network, indicating four peers.

We can obtain the block details by clicking the block number to the right of Best Block. We can see an example of block explorer in image 12.



Figure 12 Block Explorer

We can explore blocks by clicking on the blocks under **Bk** on the left-hand side.

We can search for a specific block, transaction hash, or address by searching in the top left-hand corner. As seen in image 13.

Enter a block number, transaction hash or address.

Figure 13 Block Explorer block searcher



Monitor nodes with Prometheus and Grafana

The sample network also includes Prometheus and Grafana monitoring tools to let you visualize node health and usage. We can directly access these tools from your browser at the addresses displayed in the endpoint list.

We can observe Grafana's user interface in figure 14.

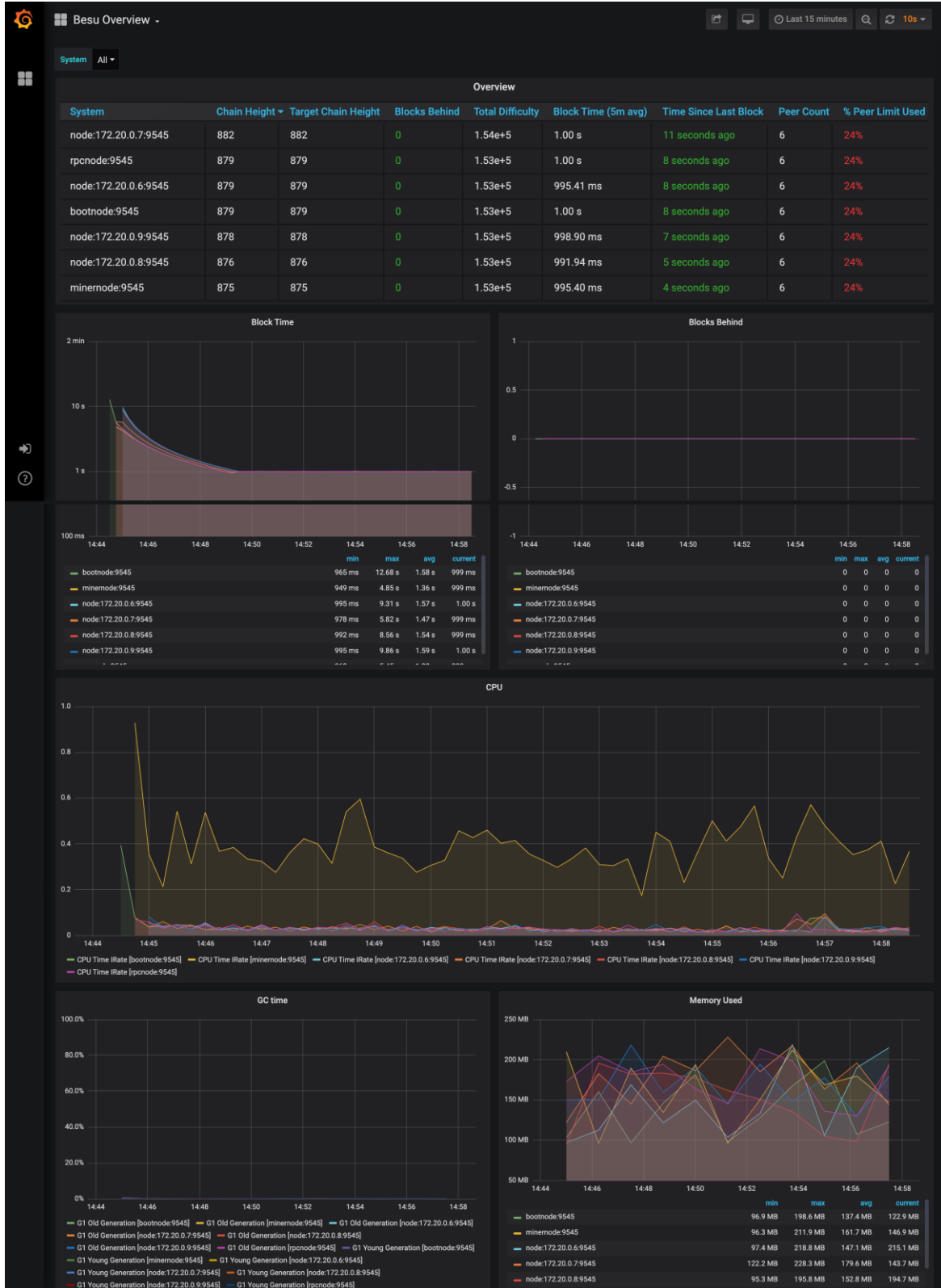


Figure 14 Grafana's user interface



Stop and restart the private network without removing containers

To shut down the private network without deleting the containers we run:

- `./stop.sh`

This command stops the containers related to the services specified in the docker-compose.yml file. And to restart the private network we run:

- `./resume.sh`

Stop the private network and remove containers

To shut down the private network and delete all containers and images created from running the sample network

- `./remove.sh`

3.3. Second Test: Create a permissioned network

The second test consisted on trying to make a permissioned network using docker and the permissioned network example in the Hyperledger Besu documentation page. [23]

Prerequisites

- Hyperledger Besu
- Curl

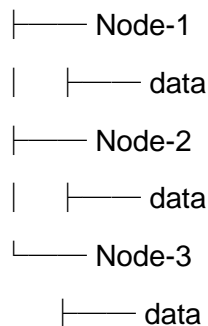
Configure Orion in our Network

Each node requires a data directory for the blockchain data. When the node starts, Besu saves the node key in this directory.

We create directories for each of the three nodes of the permissioned network and a data directory for each node.

In the following schematic we can observe the folder directory for the nodes of an Hyperledger Besu Network.

Permissioned-Network/



In networks that use Clique, we must include the address of at least one initial signer in the genesis file. We will use Node-1 as the initial signer. This requires obtaining the address for Node-1.

- `besu --data-path=data public-key export-address --to=data/nodeAddress1`

The genesis file defines the genesis block of the blockchain, the start of the blockchain. The Clique genesis file includes the address of Node-1 as the initial signer in the extraData field. All nodes in a network must use the same genesis file.

Then we create the permissions configuration file that defines the nodes and accounts allowlists. We create a file called `permissions_config.toml` and save a copy in the data folder of each node. The permissions configuration file includes the first two accounts from the genesis file:

```
accounts-allowlist=["0xfe3b557e8fb62b89f4916b721be55ceb828dbd73",  
"0x627306090abaB3A6e1400e9345bC60c78a8BEf57"]  
  
nodes-allowlist=[]
```

Finally, we start the first node:

- `besu --data-path=data --genesis-file=../cliqueGenesis.json --permissions-nodes-config-file-enabled --permissions-accounts-config-file-enabled --rpc-http-enabled --rpc-http-api=ADMIN,ETH,NET,PERM,CLIQUE --host-allowlist="*" --rpc-http-cors-origins="*"`

The command line allows us to enable nodes and accounts, the JSON-RPC API, the ADMIN, ETH, NET, PERM, and CLIQUE APIs, all-host access to the HTTP JSON-RPC API and all-domain access to the node through the HTTP JSON-RPC.

We proceed by starting the second node:

- `besu --data-path=data --genesis-file=../cliqueGenesis.json --permissions-nodes-config-file-enabled --permissions-accounts-config-file-enabled --rpc-http-enabled --rpc-http-api=ADMIN,ETH,NET,PERM,CLIQUE --host-allowlist="*" --rpc-http-cors-origins="*" --p2p-port=30304 --rpc-http-port=8546`

And third node:

- `besu --data-path=data --genesis-file=../cliqueGenesis.json --permissions-nodes-config-file-enabled --permissions-accounts-config-file-enabled --rpc-http-enabled --rpc-http-api=ADMIN,ETH,NET,PERM,CLIQUE --host-allowlist="*" --rpc-http-cors-origins="*" --p2p-port=30305 --rpc-http-port=8547`

3.4. Third Test: Create a permissioned network with Docker

The third test consisted on doing the previous test while using Docker. The initial thought process was to do exactly the same as the previous test but adding the appropriate docker commands before the command lines seen on test 2. [24]

While initially it seemed to work, we couldn't figure out why we were getting errors when we tried to start the first node claiming it was unable to find the necessary files

After some investigation we found out the cause. The way docker works is that when we use an image everything, from the code, runtime, systemtools, and even libraries and folders are stored in a package called volumes. Thus, when we were trying to launch the code using docker it would search its own folders to find the files and not the folders we created. [25]

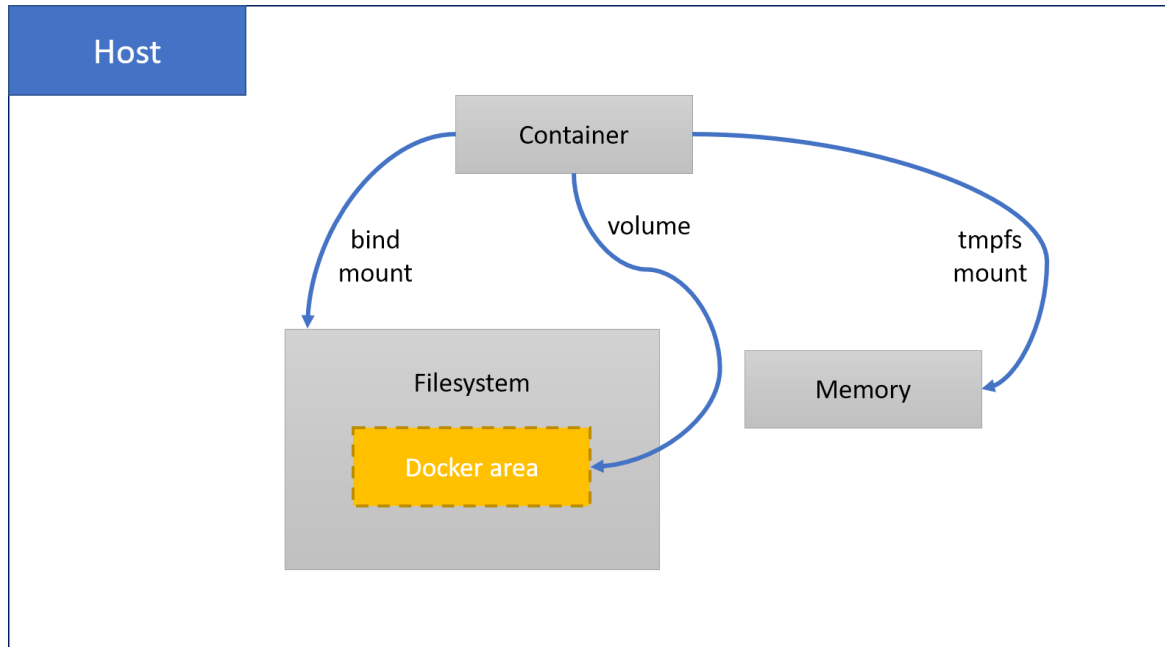


Figure 15 Volumes in Docker

While it is possible to link the Docker Image's folders and files with those of the Virtual Machine. This process is called volume mounting. We can see how volumes and the filesystem interact with each other in figure 13.

However, we decided against it as it is quite complex.

3.5. Fourth Test: Create an Hyperledger Besu Network with Docker Compose to send transactions

After seeing the complications of making a more complex network with Hyperledger Besu using Docker we thought of the idea of searching working networks that were established using Docker Compose instead of Docker.

It is an example network with three Besu nodes ready to use private transactions. It contains Orion, Besu and EthSigner. [26]

Prerequisites

Docker

Docker Compose

Network Mapping

- Besu 1 - <http://localhost:8515>
- Orion 1 - <http://localhost:8881>
- Besu 2 - <http://localhost:8525>
- Orion 2 - <http://localhost:8882>
- Besu 3 - <http://localhost:8535>
- Orion 3 - <http://localhost:8883>
- EthSigner - <http://localhost:9545>

Keys Used

- Orion 1 Public Key: GGilEkXLaQ9yhhtbpBT03Me9iYa7U/mWXrJhnb11XY=
- Orion 2 Public Key: KkOjNlMCl6r+mICrC6l+XuEDjFEzQllaMQMpWLI4y1s=
- Orion 3 Public Key: eLb69r4K8/9WviwlfDiZ4jf97P9czyS3DkKu0QYGLjg=

How to use

First, we execute `docker-compose up` to start the services and run `./send-private-tx.sh` to send a private transaction. This command will output the hash of the private tx.

If we run `./get-private-tx.sh <HASH>` we can obtain the details of the the private transaction.

Finally, we run `./purgedb.sh` to purge the database and start your network from scratch.

Example

```
$ ./send-private-tx.sh
>>
{
  "jsonrpc" : "2.0",
  "id" : 1,
  "result" : "0x1b628d80b8f3db873adea6a2f7a775a62c83d4381ad4f19d7ba87fb05472630c"
}

$ ./get-private-tx.sh
0x1b628d80b8f3db873adea6a2f7a775a62c83d4381ad4f19d7ba87fb05472630c
>>
{
  "jsonrpc" : "2.0",
  "id" : 1,
  "result" : {
    "from" : "0xfe3b557e8fb62b89f4916b721be55ceb828dbd73",
    "gas" : "0x15f90",
    "gasPrice" : "0x0",
    "hash" : "0xbab7f61c4a14942ed8eed3543a4a197b946f6787faba6e409985ff764b709c7",
    "input" : "0x608060405234801561001057600080fd5b5060dc8061001f6000396000f300608060405260
0436106049576000357c01000000000000000000000000000000000000000000000000000000000000000000
0900463ffffffff1680633fa4f24514604e57806355241077146076575b600080fd5b34801560
5957600080fd5b50606060a0565b6040518082815260200191505060405180910390f35b34801
5608157600080fd5b50609e60048036038101908080359060200190929190505060a6565b00
5b60005481565b80600081905550505600a165627a7a723058202bdbba2e694dba8fff33d9d09
76df580f57bffa40e25a46c398f8063b4c00360029",
    "nonce" : "0x5",
    "to" : null,
    "value" : "0x0",
    "v" : "0xfe8",
    "r" : "0xc2a1eb89d18c59c7a1b6ee9da8c70ef0d74f1da550786dccff3240b7ae7f78a6",
    "s" : "0x4f96f9609899b5c3f0ff17a5b167c8769a509a345811e5fb38259e959a764d70",
    "privateFrom" : "GGilEkXLaQ9yhhtbpBT03Me9iYa7U/mwXxrJhnb11XY=",
    "privateFor" : [ "GGilEkXLaQ9yhhtbpBT03Me9iYa7U/mwXxrJhnb11XY=" ],
    "restriction" : "restricted"
  }
}

$ ./find-privacy-group.sh
>>
{
  "jsonrpc" : "2.0",
  "id" : 1,
  "result" : [ {
    "privacyGroupId" : "OGD/4dkDZWb4VqgDfEllovjYMDAcSiRUiB6fLtFRmugU=",
    "name" : "legacy",
    "description" : "Privacy groups to support the creation of groups by
privateFor and privateFrom",
    "type" : "LEGACY",
    "members" : [ "KkJNlMCI6r+mICrC6l+XuEDjFEzQ1laMQMpwLl4y1s=",
"GGilEkXLaQ9yhhtbpBT03Me9iYa7U/mwXxrJhnb11XY=" ]
  } ]
}
```

```
} ]  
}  
$ ./get-transaction-count.sh 0xfe3b557e8fb62b89f4916b721be55ceb828dbd73  
OGD/4dkDZWb4VqgDfElvjYMDAcSiRUiB6fLtFRmugU=  
>>  
{  
  "jsonrpc" : "2.0",  
  "id" : 1,  
  "result" : "0x2"  
}  
  
$ ./get-eea-transaction-count.sh  
>>  
{  
  "jsonrpc" : "2.0",  
  "id" : 1,  
  "result" : "0x2"  
}
```

Conclusion

While this network was what we wanted, an Hyperledger Besu Network with Orion set up with Docker, it was very limiting in what it allowed the user to do. We decided to search a bit more to see if we could find something more promising. In case we couldn't we could work around this configuration to make our own network.

3.6. Fifth Test: Create an Hyperledger Besu Network with Docker Compose

After digging up a bit we found what would make our fifth and final test. From a repository created by Lucas Aldhana. [27]

The repository includes multiple sample networks, each of them has a POW and POA example. We can choose the default setup which is made by a 4-node network with Block Explorer and Prometheus and Grafana dashboard to track the progress of the chain.

The network we will make can be viewed in figure 16.

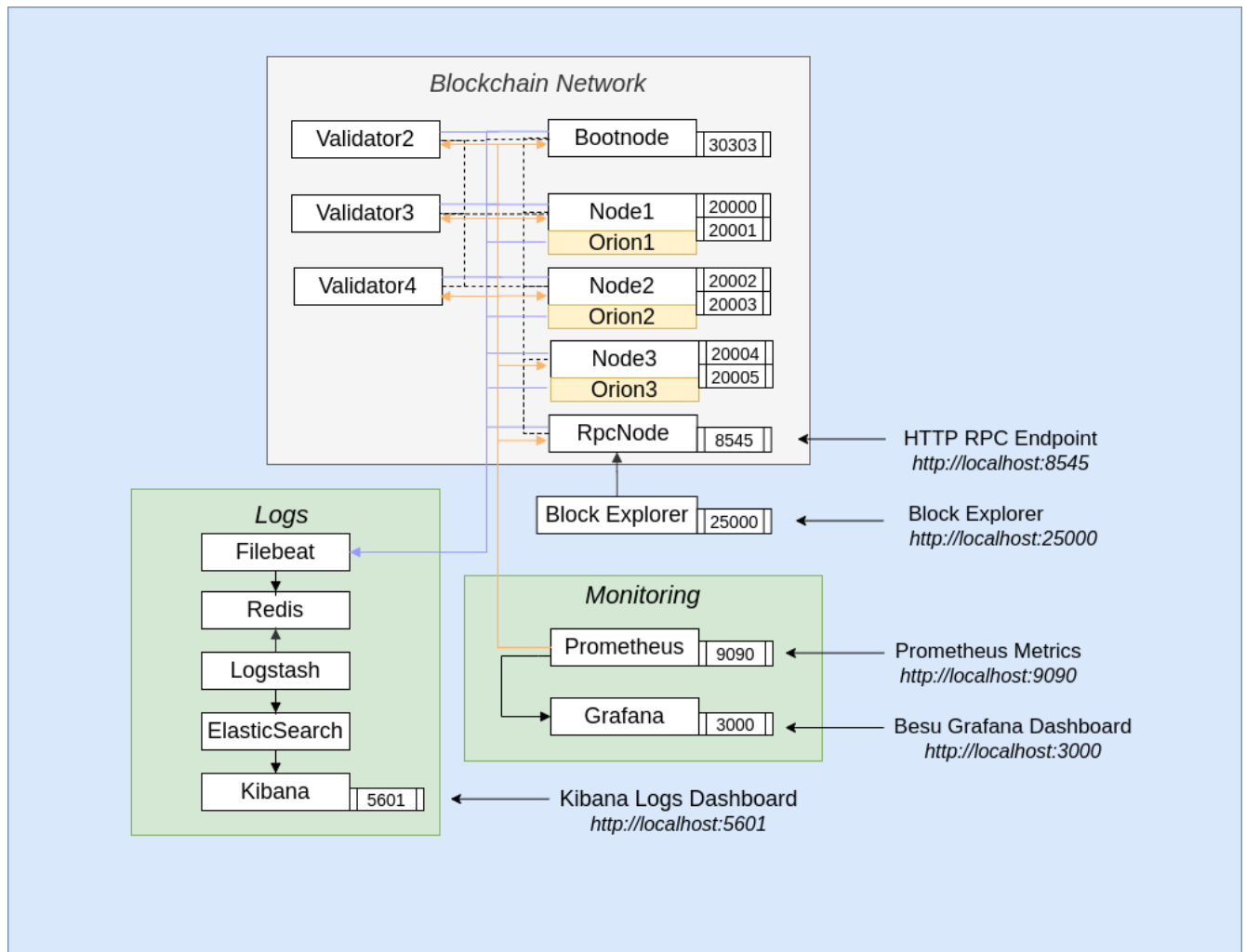


Figure 16 Test 5 network schematic

The network has four validators, 3 called as such seen in the picture above and the Bootnode. The RPCNode is used to make queries and the Block Explorer shows us the progress of the network. Furthermore, we have three Besu nodes, each one of them with an Orion node.

When we send a contract or transaction between two nodes (Node1 and Node2) and check all three nodes information we should see that only Node1 and Node2 have this information



because they were involved in the transaction and that Orion3 responds with a 0x value for reads at those addresses.

The node details can be seen in table 1.

Table 1 Node's addresses and keys

Name	Besu Node Address	Orion Node Key	Host
Node1	0x866b0df7138daf807300ed9204de733c1eb6d600	9QHwUJ6uK+FuQMzFSXlo7wOLCGFZa0PiF771OLX5c1o=	http://localhost:20000
Node2	0xa46f0935de4176ffeccdeecaf3c6e3ca03e31b22	qVDsbJh2UluZOePxbXAL49g0S0s2gGIJ3ftQceMlchU	http://localhost:20002
Node3	0x998c8bc11c28b667e4b1930c3fe3c9ab1cde3c52	T1ltOQxwgY1pTW6YXb2EbKXYkK4saBEys3CfJ2OIKHs	http://localhost:20004

Prerequisites

Docker

Docker Compose

Nodejs

Starting the Services and Network

In order to start eh network we have to input the following command

- `./run-privacy.sh` to start all the docker containers in POW mode

Or

- `./run-privacy.sh -c ibft2` to start all the docker containers in POA mode using the IBFT2 Consensus algorithm

Additionally, we can add the `-e` parameter which will provide centralised logging functionality via ELK.

In order to start the image, we need to run:

- `docker-compose -f docker-compose_privacy_poa.yml up`

To stop the services, we use then `./stop.sh`, which stops all the docker containers created and `./remove.sh` which stops and removes all the containers and volumes.

3.6.1. Fifth Test: Problems and Solutions

The first problem was related to permissions, the Docker compose code launches an update command that installs a dos 2 unix EOL converter (supporting either alpine or ubuntu images). In Linux an update normally needs root permission every time its launched and this time it was no exception. However, the root permission can't be given just from running the Docker command with root permissions.

What we did to solve the problem was change a bit the Dockerfile and give root access just before it tries to update.

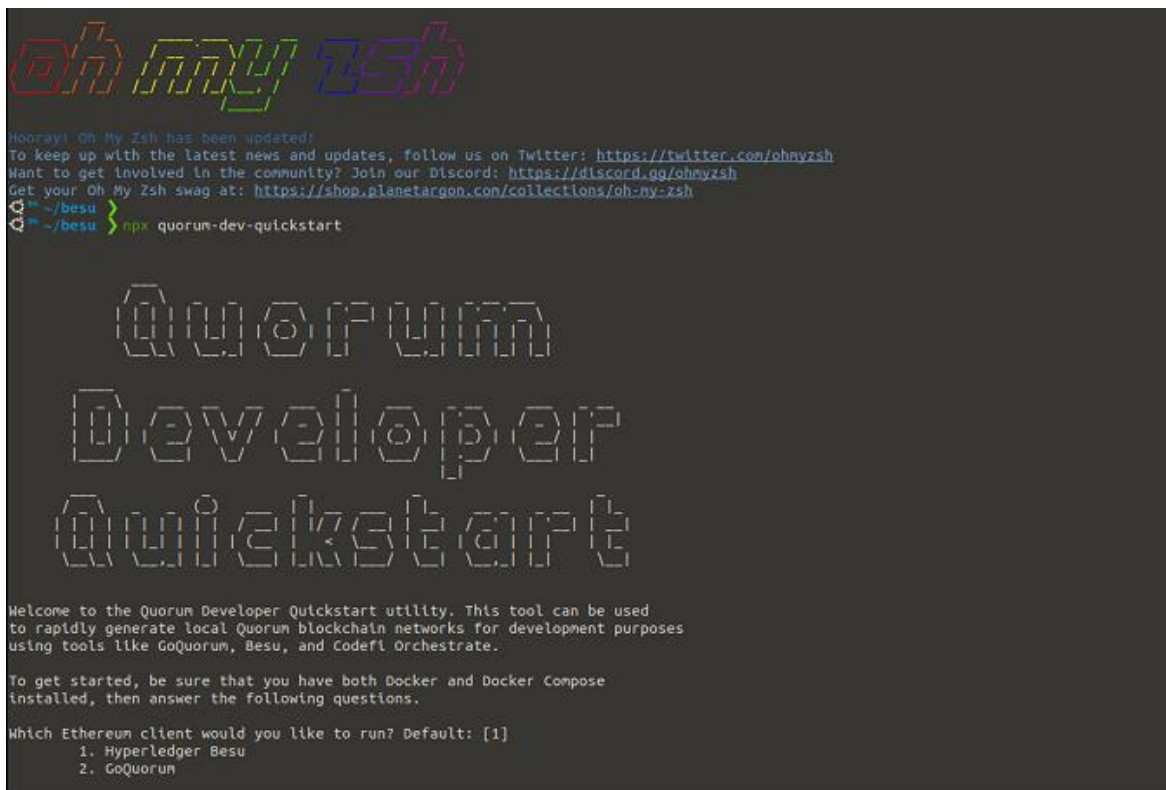
Next, we had so timeouts problems. In order to solve this problem, we decided to make sure that the ports for all nodes were open, from port 2000 to 2005. Also we decided to just in case stop all processes and restart the terminal as well.

4. Results

The first part of the results section will be dedicated into showcasing two of the tested networks, the first and the fifth, working. The second part will be a theoretical one on how should an user modify the documents in the given repository in order to add or delete nodes in the network.

4.1. First Network

Before starting the network, we will run again `npx quorum-dev-quickstart`, so that it installs any updates. In figure 18, we can observe the quickstart developer quickstart.



```
oh my zsh
Hooray! Oh My Zsh has been updated!
To keep up with the latest news and updates, follow us on Twitter: https://twitter.com/ohmyzsh
Want to get involved in the community? Join our Discord: https://discord.gg/ohmyzsh
Get your Oh My Zsh swag at: https://shop.planetargon.com/collections/oh-my-zsh
~/besu > npx quorum-dev-quickstart

Quorum
Developer
Quickstart

Welcome to the Quorum Developer Quickstart utility. This tool can be used
to rapidly generate local Quorum blockchain networks for development purposes
using tools like GoQuorum, Besu, and Codefi Orchestrate.

To get started, be sure that you have both Docker and Docker Compose
installed, then answer the following questions.

Which Ethereum client would you like to run? Default: [1]
1. Hyperledger Besu
2. GoQuorum
```

Figure 17 Quorum developer quickstart

To launch the network then we run `./run.sh`, which will build the Docker images and containers. Which we can see in figures 18 and 19.

```

alice@tfm-01:~/besu
File Edit View Search Terminal Help
~/besu > ./run.sh
*****
Quorum Dev Quickstart
*****
Start network
-----
Starting network...
validator1 uses an image, skipping
validator2 uses an image, skipping
validator3 uses an image, skipping
validator4 uses an image, skipping
rpcnode uses an image, skipping
ethsignerProxy uses an image, skipping
member1orion uses an image, skipping
member1besu uses an image, skipping
member2orion uses an image, skipping
member2besu uses an image, skipping
member3orion uses an image, skipping
member3besu uses an image, skipping
prometheus uses an image, skipping
grafana uses an image, skipping
Building explorer
Step 1/4 : FROM nginx:alpine
alpine: Pulling from library/nginx
a0d0ad46f8b: Pull complete
4dd4ef99939: Pull complete
c1368e94e1ec: Pull complete
3e72c40d0ff4: Pull complete
969825a5ca61: Pull complete
61074acc7dd2: Pull complete
Digest: sha256:686aac279fd6e7bab67663fd38750c135b72d993d0bb0a942ab02ef647fc9c3
Status: Downloaded newer image for nginx:alpine
--> 513f9a9d8748
Step 2/4 : COPY dist /usr/share/nginx/html
--> da640efd0151
Step 3/4 : RUN sed -l 's@NODE_ENV:"production",BASE_URL:"/@NODE_URL:"/rpcnode/jsonrpc",CONNECTION_TYPE:"json_rpc"@g' /usr/share/nginx/html/js/*.js
--> Running in 07141b5fc927
Removing intermediate container 07141b5fc927
--> 10b38034e45b
Step 4/4 : COPY default.nginx /etc/nginx/conf.d/default.conf
--> e77149e1c803

```

Figure 18 Launching an Hyperledger Besu network using Docker part 1/2

```

Successfully built e77149e1c803
Creating network "besu_quorum-dev-quickstart" with driver "bridge"
Creating volume "besu_public-keys" with default driver
Creating volume "besu_prometheus" with default driver
Creating volume "besu_grafana" with default driver
Creating volume "besu_cakeshop" with default driver
Creating besu_member3orion_1 ... done
Creating besu_member2orion_1 ... done
Creating besu_grafana_1 ... done
Creating besu_member1orion_1 ... done
Creating besu_prometheus_1 ... done
Creating besu_validator1_1 ... done
Creating besu_validator3_1 ... done
Creating besu_rpcnode_1 ... done
Creating besu_member1besu_1 ... done
Creating besu_validator4_1 ... done
Creating besu_member2besu_1 ... done
Creating besu_validator2_1 ... done
Creating besu_member3besu_1 ... done
Creating besu_explorer_1 ... done
Creating besu_ethsignerProxy_1 ... done

```

Figure 19 Launching an Hyperledger Besu network using Docker part 2/2

When execution is successfully finished, the process lists the available services. Which we can see on figure 20.

```

*****
Quorum Dev Quickstart
*****
-----
List endpoints and services
-----
JSON-RPC HTTP service endpoint : http://localhost:8545
JSON-RPC WebSocket service endpoint : ws://localhost:8546
Web block explorer address : http://localhost:25000/
Prometheus address : http://localhost:9090/graph
Grafana address : http://localhost:3000/d/XE4V8NGZz/besu-overview?orgId=1&refresh=10s&from=now-30m&to=now&var-system=All
-----
For more information on the endpoints and services, refer to README.md in the installation directory.

```

Figure 20 List of available services in Hyperledger Besu

From these services we will check Block explorer and Grafana:

The block explorer displays a summary of the private network, indicating seven peers. It can be observed on figure 21

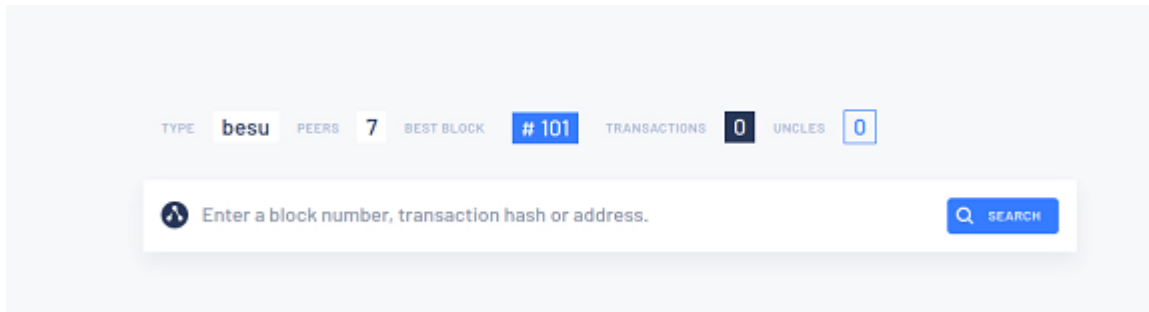


Figure 21 Block Explorer of our Hyperledger Besu network

If we click on the number to the right of Best Block to display the block details as seen in figure 22.

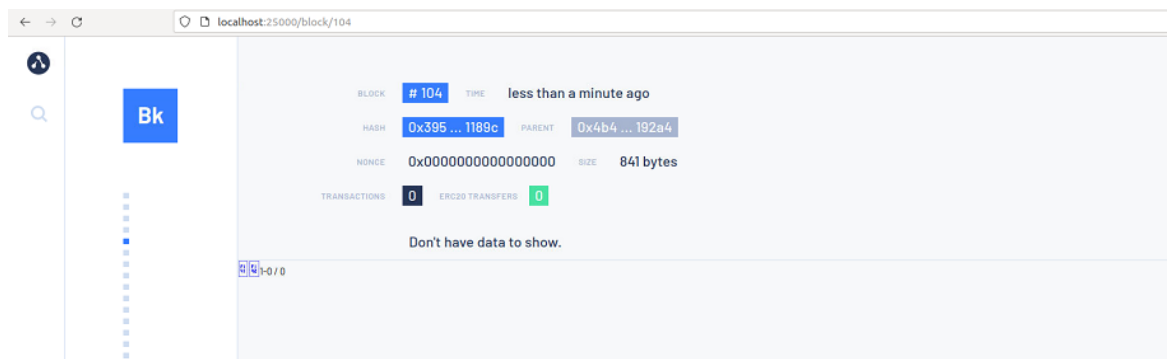


Figure 22 Best Block in our Hyperledger Besu network

The services of the tested network the Grafana Dashboard monitoring tool. Grafana lets you visualize node health and usage. We can see Grafana's dashboard in figure 23.

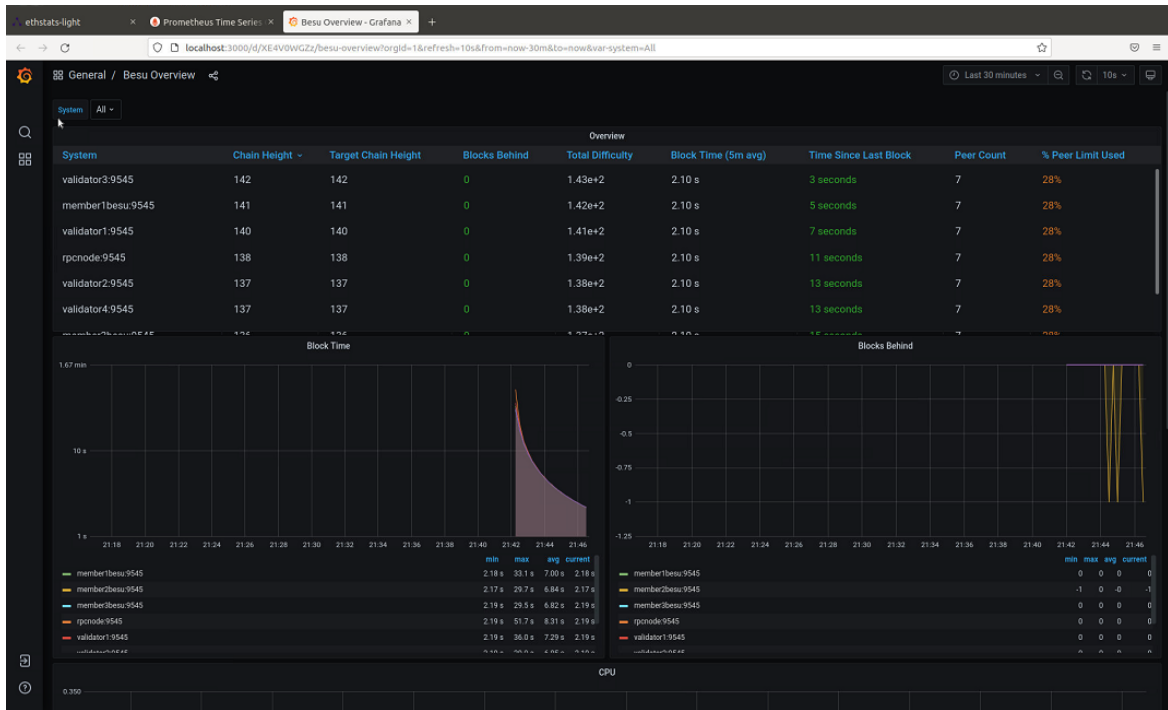


Figure 23 Grafana's Dashboard of our Hyperledger Besu network

4.2. Fifth Network

Once everything is set up, we just need to open the terminal on the folder we have all the scripts for the sample networks and run `sudo ./run-privacy.sh -c ibft2`.

```

File Edit View Search Terminal Help
alice@tfm-01:~/sample-networks/besu-sample-networks-master
$ sudo ./run-privacy.sh -c ibft2
Converting file /opt/besu/bootnode_start.sh to unix format...
Converting file /opt/besu/node_start.sh to unix format...
Removing intermediate container 607def7e4341
--> 2a7059f8c0fb
Step 7/7 : ENTRYPOINT ["/opt/besu/node_start.sh"]
--> Running in 9b8a639ec09a
Removing intermediate container 9b8a639ec09a
--> 35cf14699083
Successfully built 35cf14699083
Successfully tagged sample-network/besu:latest
WARNING: Image for service explorer was built because it did not already exist. To rebuild this image you must use 'docker-compose build' or 'docker-compose up --build'.
Building explorer
Step 1/4 : FROM nginx:alpine
--> 729b4337b0d5
Step 2/4 : COPY dist /usr/share/nginx/html
--> 0a90244e8593
Step 3/4 : RUN sed -i 's@NODE_ENV@"production",BASE_URL:"/"@NODE_URL:"/rpcnode/jsonrpc",CONNECTION_TYPE:"json_rpc"@ /usr/share/nginx/html/js/*.js
--> Running in 03981ff3ad7f
Removing intermediate container 03981ff3ad7f
--> 94b3f68ffc2b
Step 4/4 : COPY default.nginx /etc/nginx/conf.d/default.conf
--> 8a44c2b04d22
Successfully built 8a44c2b04d22
Successfully tagged sample-network/block-explorer-light:latest
WARNING: Image for service explorer was built because it did not already exist. To rebuild this image you must use 'docker-compose build' or 'docker-compose up --build'.
Creating besu-sample-networks-master/prometheus_1 ... done
Creating besu-sample-networks-master/orion1_1 ... done
Creating besu-sample-networks-master/orion2_1 ... done
Creating besu-sample-networks-master/bootnode_1 ... done
Creating besu-sample-networks-master/validator2_1 ... done
Creating besu-sample-networks-master/rproneus_1 ... done
Creating besu-sample-networks-master/validator4_1 ... done
Creating besu-sample-networks-master/validator3_1 ... done
Creating besu-sample-networks-master/explorer_1 ... done
Creating besu-sample-networks-master/node2_1 ... done
Creating besu-sample-networks-master/node1_1 ... done
Creating besu-sample-networks-master/node3_1 ... done
Sample Network for Besu at Latest
-----
List endpoints and services
-----
Name Command State Ports
-----
besu-sample-networks-master/bootnode_1 /opt/besu/bootnode_start.s ... Up (healthy) 30303/tcp, 8545/tcp, 8546/tcp, 8547/tcp
besu-sample-networks-master/explorer_1 /docker-entrypoint.sh ngin ... Up 0.0.0.0:25000->80/tcp, :::25000->80/tcp
besu-sample-networks-master/grafana_1 /run.sh Up 0.0.0.0:3000->3000/tcp, :::3000->3000/tcp
besu-sample-networks-master/prometheus_1 /bin/prometheus --config.f ... Up 0.0.0.0:9090->9090/tcp, :::9090->9090/tcp
besu-sample-networks-master/rproneus_1 /opt/besu/node_start.sh -- ... Up (health: starting) 30303/tcp, 0.0.0.0:8545->8545/tcp, :::8545->8545/tcp, 8546/tcp, 8547/tcp
-----
JSON-RPC HTTP service endpoint : http://localhost:8545
JSON-RPC WebSocket service endpoint : ws://localhost:8540
GraphQL HTTP service endpoint : http://localhost:8547
Web block explorer address : http://localhost:25000/
Prometheus address : http://localhost:9090/graph
Grafana address : http://localhost:3000/d/RE4W0GZz/besu-overview?orgId=1&refresh=10s&from=now-30m&to=now&var-system=All
-----

```

Figure 24 Launching our Hyperledger Besu network with Orion

The first thing to note, as we can observe in figure 24, is the different nodes that appear as done, including the Orion ones. We can also observe how the same services as the first network open like block explorer and Grafana.

In order to stop the network and services, we run `./stop.sh`.

When we are done, we run `./remove.sh` to remove all containers and volumes. Which can be observed in figure 2.

```

root@besu-sample-networks-master:~/# ./remove.sh
Sample Network for Besu at latest
=====
Stop and remove network...
Stopping besu-sample-networks-master_node3_1 ... done
Stopping besu-sample-networks-master_node1_1 ... done
Stopping besu-sample-networks-master_node2_1 ... done
Stopping besu-sample-networks-master_explorer_1 ... done
Stopping besu-sample-networks-master_validator4_1 ... done
Stopping besu-sample-networks-master_validator3_1 ... done
Stopping besu-sample-networks-master_rpcnode_1 ... done
Stopping besu-sample-networks-master_validator2_1 ... done
Stopping besu-sample-networks-master_bootnode_1 ... done
Stopping besu-sample-networks-master_orlon3_1 ... done
Stopping besu-sample-networks-master_grafana_1 ... done
Stopping besu-sample-networks-master_prometheus_1 ... done
Stopping besu-sample-networks-master_orlon1_1 ... done
Stopping besu-sample-networks-master_orlon2_1 ... done
Removing besu-sample-networks-master_node3_1 ... done
Removing besu-sample-networks-master_node1_1 ... done
Removing besu-sample-networks-master_node2_1 ... done
Removing besu-sample-networks-master_explorer_1 ... done
Removing besu-sample-networks-master_validator4_1 ... done
Removing besu-sample-networks-master_validator3_1 ... done
Removing besu-sample-networks-master_rpcnode_1 ... done
Removing besu-sample-networks-master_validator2_1 ... done
Removing besu-sample-networks-master_bootnode_1 ... done
Removing besu-sample-networks-master_orlon1_1 ... done
Removing besu-sample-networks-master_grafana_1 ... done
Removing besu-sample-networks-master_prometheus_1 ... done
Removing besu-sample-networks-master_orlon2_1 ... done
Removing network besu-sample-networks-master_default
Removing volume besu-sample-networks-master_public-keys
Removing volume besu-sample-networks-master_prometheus
Removing volume besu-sample-networks-master_grafana
No stopped containers
Intagged: sample-network/besu:latest
Deleted: sha256:135c72a0e0f9b9f4fe0fda5afed16d799e10d3da520dc
Deleted: sha256:2a7059f8c9fb5d5809146e6fc9c434d017fcb0b568e4a807b01335974
Deleted: sha256:2e711a808d1c4f8f8bcb914eb6445817fcb0b768fc1580978dbccab85f
Intagged: sample-network/prometheus-light:latest
Deleted: sha256:8a4c2b04d22c4e685d284e98b58994f0c5c36284bca73b747ec380
Deleted: sha256:4b452795d9b81c0ad4915d58c1352cc4d629b1d388481d6af9fd4ad2498
Deleted: sha256:1403f68f72bb3b38f20999966bc1c29e63522267292e3b9e97424
Deleted: sha256:eaa31237297282c2e5fa3232966d39e3529d84c4fb59b07ef16e4cca
Deleted: sha256:8e9d24e4593071e3e5a7e4b67df0b0a5192d413987d6151bb2a59da
Deleted: sha256:9db04ba2807110440a3f7c9776076024129e7715450504053c00d437
rm remove write protected regular file .sampleNetworks.lock *
lock file .sampleNetworks.lock removed
root@besu-sample-networks-master:~/#

```

Figure 25 Removing the network

Once the image is set up, we will run, as seen in image 26, docker-compose -f docker-compose_privacy_poa.yml up. In the image we can see the network running.

```

File Edit View Search Terminal Help
docker-compose-1-docker-compose_privacy_poa.yml up
node1_1 2021-09-26 15:44:49.725+00:00 | main | INFO | DefaultP2PNetwork | Node address 6x27306099aabb3ae140099345b0cc78a8bef57
node1_1 2021-09-26 15:44:49.729+00:00 | main | INFO | DefaultSynchronizer | Starting synchronizer.
node1_1 2021-09-26 15:44:49.731+00:00 | main | INFO | FullSyncLoader | Starting full sync.
node1_1 2021-09-26 15:44:49.733+00:00 | main | INFO | Runner | Ethereum main loop is up.
node1_1 2021-09-26 15:44:49.733+00:00 | vert.x-eventloop-thread-14 | INFO | MetricsHTTPEndpoint | Metrics service started and listening on 9545:9545
node1_1 2021-09-26 15:44:49.744+00:00 | main | INFO | FullSyncTargetManager | No sync target, waiting for peers: 0
node1_1 2021-09-26 15:44:49.749+00:00 | main | INFO | LogBlockCacheMetadata | Lookup cache metadata file in data directory: /opt/besu/data/caches
node1_1 2021-09-26 15:44:49.769+00:00 | nioEventLoopGroup-3-9 | INFO | FullSyncTargetManager | No sync target, waiting for peers: 6
node1_1 2021-09-26 15:44:49.799+00:00 | nioEventLoopGroup-3-1 | INFO | FullSyncTargetManager | No sync target, waiting for peers: 2
node1_1 2021-09-26 15:44:49.816+00:00 | vert.x-eventloop-thread-8 | INFO | JsnonRpcHTTPEndpoint | JSON-RPC service started and listening on 0.0.0.0:8545
node1_1 2021-09-26 15:44:49.825+00:00 | vert.x-eventloop-thread-10 | INFO | GraphQLHTTPEndpoint | GraphQL HTTP service started and listening on 0.0.0.0:8547
node1_1 2021-09-26 15:44:49.826+00:00 | main | INFO | WebSocketService | Starting WebSocket service on 0.0.0.0:8546
node1_1 2021-09-26 15:44:49.835+00:00 | vert.x-eventloop-thread-12 | INFO | WebSocketService | WebSocket service started and listening on 0.0.0.0:8546
node1_1 2021-09-26 15:44:49.836+00:00 | main | INFO | MetricsHTTPEndpoint | Metrics service started and listening on 0.0.0.0:9545
node1_1 2021-09-26 15:44:49.844+00:00 | vert.x-eventloop-thread-14 | INFO | MetricsHTTPEndpoint | Metrics service started and listening on 9545:9545
node1_1 2021-09-26 15:44:49.847+00:00 | main | INFO | Runner | Ethereum main loop is up.
node1_1 2021-09-26 15:44:49.847+00:00 | nioEventLoopGroup-3-7 | INFO | FullSyncTargetManager | No sync target, waiting for peers: 4
node1_1 2021-09-26 15:44:49.847+00:00 | nioEventLoopGroup-3-8 | INFO | FullSyncTargetManager | No sync target, waiting for peers: 4
node1_1 2021-09-26 15:44:49.847+00:00 | nioEventLoopGroup-3-10 | INFO | FullSyncTargetManager | No sync target, waiting for peers: 7
node1_1 2021-09-26 15:44:49.848+00:00 | nioEventLoopGroup-3-4 | INFO | FullSyncTargetManager | No sync target, waiting for peers: 2
node1_1 2021-09-26 15:44:49.849+00:00 | nioEventLoopGroup-3-4 | INFO | FullSyncTargetManager | No sync target, waiting for peers: 2
node1_1 2021-09-26 15:44:49.876+00:00 | nioEventLoopGroup-3-5 | INFO | FullSyncTargetManager | No sync target, waiting for peers: 5
node1_1 2021-09-26 15:44:49.904+00:00 | nioEventLoopGroup-3-1 | INFO | FullSyncTargetManager | No sync target, waiting for peers: 7
node1_1 2021-09-26 15:44:49.909+00:00 | nioEventLoopGroup-3-2 | INFO | FullSyncTargetManager | No sync target, waiting for peers: 1
node1_1 2021-09-26 15:44:49.918+00:00 | main | INFO | LogBlockCacheMetadata | Lookup cache metadata file in data directory: /opt/besu/data/caches
node1_1 2021-09-26 15:44:49.918+00:00 | nioEventLoopGroup-3-2 | INFO | FullSyncTargetManager | No sync target, waiting for peers: 6
node1_1 2021-09-26 15:44:50.149+00:00 | nioEventLoopGroup-3-9 | INFO | FullSyncTargetManager | No sync target, waiting for peers: 5
node1_1 2021-09-26 15:44:50.142+00:00 | nioEventLoopGroup-3-4 | INFO | FullSyncTargetManager | No sync target, waiting for peers: 8
node1_1 2021-09-26 15:44:50.149+00:00 | nioEventLoopGroup-3-1 | INFO | FullSyncTargetManager | No sync target, waiting for peers: 3
node1_1 2021-09-26 15:44:50.149+00:00 | nioEventLoopGroup-3-5 | INFO | FullSyncTargetManager | No sync target, waiting for peers: 2
node1_1 2021-09-26 15:44:50.157+00:00 | nioEventLoopGroup-3-1 | INFO | FullSyncTargetManager | No sync target, waiting for peers: 4
node1_1 2021-09-26 15:44:50.176+00:00 | nioEventLoopGroup-3-4 | INFO | FullSyncTargetManager | No sync target, waiting for peers: 5
node1_1 2021-09-26 15:44:50.176+00:00 | nioEventLoopGroup-3-9 | INFO | FullSyncTargetManager | No sync target, waiting for peers: 5
node1_1 2021-09-26 15:44:50.188+00:00 | nioEventLoopGroup-3-3 | INFO | FullSyncTargetManager | No sync target, waiting for peers: 7
node1_1 2021-09-26 15:44:50.189+00:00 | nioEventLoopGroup-3-9 | INFO | FullSyncTargetManager | No sync target, waiting for peers: 5
node1_1 2021-09-26 15:44:50.189+00:00 | nioEventLoopGroup-3-1 | INFO | FullSyncTargetManager | No sync target, waiting for peers: 7
node1_1 2021-09-26 15:44:50.189+00:00 | nioEventLoopGroup-3-2 | INFO | FullSyncTargetManager | No sync target, waiting for peers: 6
node1_1 2021-09-26 15:44:50.189+00:00 | nioEventLoopGroup-3-2 | INFO | FullSyncTargetManager | No sync target, waiting for peers: 6
node1_1 2021-09-26 15:44:50.192+00:00 | nioEventLoopGroup-3-2 | INFO | FullSyncTargetManager | No sync target, waiting for peers: 6
node1_1 2021-09-26 15:44:51.023+00:00 | nioEventLoopGroup-3-2 | INFO | FullSyncTargetManager | No sync target, waiting for peers: 6
node1_1 2021-09-26 15:44:51.023+00:00 | nioEventLoopGroup-3-2 | INFO | FullSyncTargetManager | No sync target, waiting for peers: 6
node1_1 2021-09-26 15:44:51.023+00:00 | nioEventLoopGroup-3-4 | INFO | FullSyncTargetManager | No sync target, waiting for peers: 7
node1_1 2021-09-26 15:44:51.023+00:00 | nioEventLoopGroup-3-4 | INFO | FullSyncTargetManager | No sync target, waiting for peers: 7
node1_1 2021-09-26 15:44:51.023+00:00 | nioEventLoopGroup-3-3 | INFO | FullSyncTargetManager | No sync target, waiting for peers: 7
node1_1 2021-09-26 15:44:51.023+00:00 | nioEventLoopGroup-3-4 | INFO | FullSyncTargetManager | No sync target, waiting for peers: 7
node1_1 2021-09-26 15:44:51.023+00:00 | nioEventLoopGroup-3-3 | INFO | FullSyncTargetManager | No sync target, waiting for peers: 7
node1_1 2021-09-26 15:44:51.023+00:00 | nioEventLoopGroup-3-4 | INFO | FullSyncTargetManager | No sync target, waiting for peers: 7
node1_1 2021-09-26 15:44:51.023+00:00 | nioEventLoopGroup-3-4 | INFO | FullSyncTargetManager | No sync target, waiting for peers: 7

```

Figure 26 Network functioning

4.3. Modifying the Network

While we could not do this as we were testing until the last day and worried touching the network might have adverse effect, we studied the different parts of the network to find out how to add or remove more nodes in the case it is needed in the future.

In order to do so, two files need to be changed.

First of all, if we want to add nodes, we should check the `config/orion/networkFiles` folder, there we can observe that there are only three folders one for each possible Orion node in the sample network.

If we want to add more nodes then what we need would be to create a new folder for the node adding there the 3 files we find inside: the public and private key of the node as well as a configuration file for orion.

In the case of the configuration file, it is almost the same for each orion node, the only difference would be `nodeurl` field where it should be `orion4` in this case.

As for the keys they can be obtained the same way we obtained them in our second test.

Finally, we need to change the yml file of whatever Docker-compose sample network we want to modify. In this case we will call it `node3` as it starts with `bootnode (node0)`.

```
Node3
:
  image: sample-network/besu:${BESU_VERSION}
  environment:
    - BESU_PUBLIC_KEY_DIRECTORY=${BESU_PUBLIC_KEY_DIRECTORY}
    - LOG4J_CONFIGURATION_FILE=/config/log-config.xml
  command: *base_options
  volumes:
    - public-keys:${BESU_PUBLIC_KEY_DIRECTORY}
    - ./config/besu/config.toml:/config/config.toml
    - ./logs/besu:/var/log/
    - ./config/besu/log-config.xml:/config/log-config.xml

  - ./config/besu/ibft2GenesisPermissioning.json:/config/genesis.js
  on
    - ./config/besu/networkFiles/node2/keys:/opt/besu/keys
    - node4:/opt/besu/data
  depends_on:
    - bootnode
  networks:
    sample:
      ipv4_address: 172.24.2.13
```


5. Budget

The realization of this project was fairly simple when counting costs:

- Acer Aspire 3 a315-41-r8zc - 500€
- Virtual Machine Server – 245€
- Junior engineer - 3750€

The total rounds up to 4500 €.

For the Laptop, the virtual machine can be accessed by cheaper computers.

The salary for junior engineers is 21.948 € per year which would translate to approximately the amount specified on the costs, taking into account it is a salary for nine daily working hours.

For the Virtual Machine we calculated cumulative costs of seven months for using an online virtual machine. As seen in figure 26. [28]

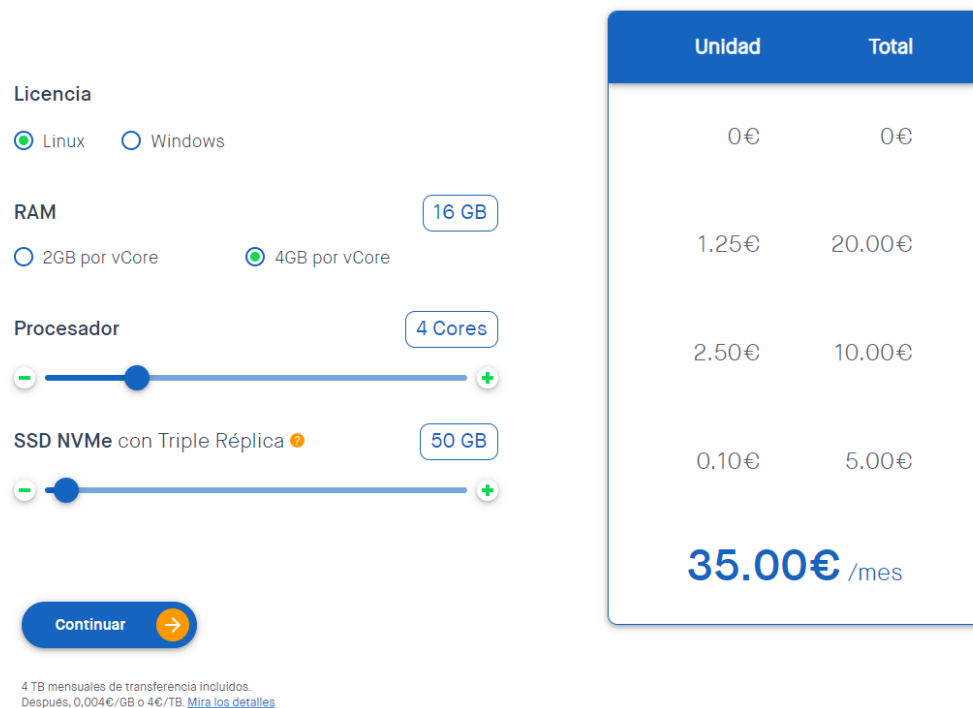


Figure 27 Virtual Machine cost calculator

While the cost may seem initially high, this project will allow to future testers of the network to save time when launching the network as with Docker Compose it can be launched in any machine without worrying about compatibility issues.

6. Conclusions and future development

Hyperledger Besu with the help of private transaction managers offers a great potential to the development of blockchain-based applications. The objective of this project was to find a way to easily set up an Hyperledger Besu with Orion. Thanks to Docker and Docker Compose, the interested user will not have to worry about dependencies or the Operating System they are on, making it easier for them to study the network or develop applications for it.

The continuation of this project could be the study of different smart contracts or applications; including banking, healthcare, financial services or supply chain management, being deployed using a docker based blockchain network showcasing the strengths of Hyperledger Besu and specially of Orion. [29] [30]

Another possible take would be updating the network, short before the project ended, Orion was absorbed into Tessera. Tessera supports the same endpoints and functionality, nothing needs to be changed in the Besu deployment to migrate. However, as Orion and Tessera cannot communicate, it should go into the user's interest to update to Tessera.

A utility is included in Tessera which enables migration of an Orion configuration file and database to a Tessera configuration file and database.

Moreover, since we tested it, Quorum developer quickstart has been updated to include privacy.

7. Bibliography

- [1] P. Broadhurst, "Kaleido," 19 December 2019. [Online]. Available: <https://www.kaleido.io/blockchain-blog/private-transactions-on-blockchain-with-hyperledger-besu-and-orion>.
- [2] "101 Blockchains," 10 October 2019. [Online]. Available: <https://101blockchains.com/hyperledger-besu/>.
- [3] R. Jones, "Hyperledger Besu Wiki," 02 June 2021. [Online]. Available: <https://wiki.hyperledger.org/display/BESU/Hyperledger+Besu>.
- [4] T. K. Sharma, "Blockchain Council," 14 October 2019. [Online]. Available: <https://www.blockchain-council.org/blockchain/hyperledger-besu-an-exhaustive-guide/>.
- [5] "Hyperledger Besu Architecture Overview," 18 March 2021. [Online]. Available: <https://besu.hyperledger.org/en/stable/Concepts/ArchitectureOverview/>.
- [6] "Hyperledger Besu Clique," 7 September 2021. [Online]. Available: <https://besu.hyperledger.org/en/stable/HowTo/Configure/Consensus-Protocols/Clique/>.
- [7] "Hyperledger Besu IBFT 2.0," 7 September 2021. [Online]. Available: <https://besu.hyperledger.org/en/stable/HowTo/Configure/Consensus-Protocols/IBFT/>.
- [8] "Hyperledger Besu Privacy Overview," 20 January 2021. [Online]. Available: <https://besu.hyperledger.org/en/21.1.5/Concepts/Privacy/Privacy-Overview/>.
- [9] "Hyperledger Besu Permissioning Overview," 14 December 2020. [Online]. Available: <https://besu.hyperledger.org/en/stable/Concepts/Permissioning/Permissioning-Overview/>.
- [10] "Hyperledger Besu Onchain Permissioning," 8 September 2021. [Online]. Available: <https://besu.hyperledger.org/en/stable/Concepts/Permissioning/Onchain-Permissioning/>.
- [11] "Orion Documentation," 1 November 2020. [Online]. Available: <https://docs.orion.consensys.net/en/stable/>.
- [12] "Hyperledger Besu Private Transactions," 14 December 202. [Online]. Available: <https://besu.hyperledger.org/en/21.1.5/Concepts/Privacy/Private-Transactions/>.

- [13] “Orion Privacy Groups,” 23 September 2020. [Online]. Available: <https://docs.orion.consensys.net/en/stable/Concepts/Privacy-Groups/>.
- [14] “Orion Disaster Recovery,” 23 September 2020. [Online]. Available: <https://docs.orion.consensys.net/en/stable/Concepts/Disaster-Recovery/>.
- [15] “Orion TLS Communication,” 23 September 2020. [Online]. Available: <https://docs.orion.consensys.net/en/stable/Concepts/TLS-Communication/>.
- [16] “Orion Multi-Tenancy,” 23 September 2020. [Online]. Available: <https://docs.orion.consensys.net/en/stable/Concepts/Multi-Tenancy/>.
- [17] “Hyperledger Besu Privacy Groups,” 29 September 2020. [Online]. Available: <https://besu.hyperledger.org/en/21.1.5/Concepts/Privacy/Privacy-Groups/>.
- [18] “Hyperledger Besu Processing Private Transactions,” 14 December 2020. [Online]. Available: <https://besu.hyperledger.org/en/21.1.5/Concepts/Privacy/Private-Transaction-Processing/>.
- [19] “Hyperledger Besu Multi-Tenancy,” 20 January 2021. [Online]. Available: <https://besu.hyperledger.org/en/21.1.5/Concepts/Privacy/Multi-Tenancy/>.
- [20] P. Srivastav, “Docker for beginners,” 22 March 2015. [Online]. Available: <https://docker-curriculum.com/#introduction>.
- [21] “Docker Docs Compose,” [Online]. Available: <https://docs.docker.com/compose/>. [Accessed 04 2021].
- [22] “Hyperledger Besu Developer Quickstart,” 30 October 2020. [Online]. Available: <https://besu.hyperledger.org/en/21.1.5/Tutorials/Developer-Quickstart/>.
- [23] “Hyperledger Besu Create a permissioned network,” 14 December 2020. [Online]. Available: <https://besu.hyperledger.org/en/21.1.5/Tutorials/Permissioning/Create-Permissioned-Network/>.
- [24] B. S, “PolarSPARC Hyperledger Besu Private Network using Docker,” 16 May 2020. [Online]. Available: <https://www.polarsparc.com/xhtmll/BesuPrivateCliqueDocker.html>.
- [25] “Docker Docs Use Bind Mounts,” [Online]. Available: <https://docs.docker.com/storage/bind-mounts/>.
- [26] L. Saldanha, “Github Besu Three Nodes Example,” 10 January 2020. [Online]. Available: <https://github.com/lucassaldanha/besu-three-nodes-example>.
- [27] L. Saldanha, “Github Besu Sample Networks,” 17 February 2020. [Online]. Available: <https://github.com/lucassaldanha/besu-sample-networks>.



- [28] “Clouding.io,” [Online]. Available: <https://clouding.io/>. [Accessed 20 September 2021].
- [29] “Hyperledger,” [Online]. Available: <https://www.hyperledger.org/use>. [Accessed September 2021].
- [30] L. P. & R. D. Purathani Praitheeshan, “Private and Trustworthy Distributed Lending Model Using Hyperledger Besu,” *SN Computer Science*, vol. 2, no. 115, 2021.

8. Glossary

TX - Transaction

API - Application Programming Interface

JSON - JavaScript Object Notation

RPC - Remote Procedure Call

IBFT - Istanbul Byzantine Fault Tolerant

EVM - Ethereum Virtual Machine

POW - Proof of Work

POA - Proof of Authority

RLPx - The name is not an acronym and has no particular meaning.

ETH - Ethereum

UDP - User Datagram Protocol

TCP - Transmission Control Protocol

ID - Identity

EEA - Enterprise Ethereum Alliance

JWT - JSON Web Token

HTTP - Hypertext Transfer Protocol

OS - Operating System