UNIVERSIDADE FEDERAL DE MATO GROSSO DO SUL
FACULDADE DE COMPUTAÇÃO
MESTRADO PROFISSIONAL EM COMPUTAÇÃO APLICADA

Lucas Alves dos Santos

Infraestrutura de Blockchain para Aplicações da Plataforma Pecuária de Baixo Carbono da Embrapa

Campo Grande - MS
2026

Lucas Alves dos Santos

Infraestrutura de Blockchain para Aplicações da Plataforma Pecuária de Baixo Carbono da Embrapa

Dissertação apresentada à Faculdade de Computação da Universidade Federal de Mato Grosso do Sul como parte dos requisitos necessários à obtenção do título de Mestre em Computação Aplicada.
Orientador: Prof. Dr. Dionisio Machado Leite Filho.

Campo Grande - MS
2026

AGRADECIMENTOS

Agradeço primeiramente a Deus por me proporcionar saúde, capacidade e uma família que me apoiou e suportou todas as tensões e dificuldades enfrentadas ao longo deste mestrado.

Agradeço à minha esposa, Hosana, que esteve comigo desde a graduação, enfrentando diversas batalhas, fracassos e vitórias até aqui. Entre essas batalhas, a maior de todas foi a realização deste mestrado, cujo desenvolvimento ocorreu em paralelo com trabalho e responsabilidades familiares.

Agradeço ao meu filho mais novo, Herman, que foi minha inspiração para continuar minha trajetória acadêmica após a especialização, embora minha presença tenha sido mais limitada do que eu gostaria em seus primeiros três anos de vida.

Agradeço ao meu filho mais velho, Horlian, que muito contribuiu ajudando sua mãe e oferecendo atenção ao irmão durante esses anos, preenchendo, de certa forma, a lacuna deixada por minha dedicação a esta tarefa.

Agradeço à minha mãe, Malu, e à minha irmã, Esther, que muitas vezes me ouviram, tanto em momentos de euforia quanto em momentos de desabafo e lamentações, ajudando-me a estabilizar minha mente.

Agradeço ao meu pai, Aduilson, que, junto à minha mãe, desde muito cedo apoiou e impulsionou minha trajetória escolar. Não posso deixar de recordar os esforços deles, mesmo diante das dificuldades, viajando mensalmente para me levar às aulas do Programa de Iniciação Científica Jr. da OBMEP, além de todo o apoio para possibilitar meu ingresso na faculdade.

Agradeço aos meus antigos professores, atualmente colegas de trabalho, e amigos que me incentivaram e, por vezes, contribuíram com sugestões no desenvolvimento desta pesquisa.

Agradeço ao pesquisador da Embrapa, Camilo, que trouxe a temática tratada neste trabalho, oferecendo suporte e ajudando no direcionamento da pesquisa.

Agradeço ao meu orientador, Dionisio, que, por meio das aulas e reuniões de orientação, me ensinou grande parte do que sei sobre pesquisa científica e abriu minha mente para o universo de blockchain e computação distribuída.

Por fim, agradeço à Faculdade de Computação (FACOM) da Universidade Federal de Mato Grosso do Sul (UFMS), pela existência do Programa de Mestrado Profissional em Computação Aplicada, que me proporcionou a oportunidade de aperfeiçoar meus conhecimentos e avançar na carreira acadêmica.

RESUMO

A pecuária é uma das principais atividades econômicas do Brasil. Diante das crescentes preocupações globais com as mudanças climáticas, a Embrapa tem lançado marcas-conceito que funcionam como selos de qualidade para incentivar a adoção de práticas mais sustentáveis no setor. Este trabalho teve como objetivo desenvolver e avaliar uma infraestrutura de blockchain, como prova de conceito, para aplicações descentralizadas voltadas à certificação de produtos das marcas-conceito da Plataforma de Pecuária de Baixo Carbono (PBC) da Embrapa. A infraestrutura foi implementada com o *framework* Hyperledger Fabric em um conjunto de máquinas virtuais hospedadas na Amazon Web Services (AWS). Sua performance foi avaliada por meio da Hyperledger Caliper, considerando cenários com diferentes volumes e tamanhos de transações, além de variações no mecanismo de ordenação da rede. Os resultados indicaram que a infraestrutura suporta de forma estável uma taxa contínua de escrita de 2250 KB/s. Por outro lado, taxas médias de 4500 KB/s excedem a capacidade da rede, resultando em acúmulo de transações pendentes. A granularidade das transações não apresentou impacto significativo no desempenho, enquanto o aumento no número de nós de ordenação reduziu de forma expressiva a performance. Conclui-se que a infraestrutura proposta é tecnicamente viável para sustentar aplicações de certificação vinculadas às marcas da PBC, oferecendo potencial para contribuir com a rastreabilidade e a confiabilidade de dados na cadeia produtiva da pecuária sustentável.

Palavras-chave: Carne Carbono Neutro; Certificação; Blockchain; Hyperledger Fabric; Contratos Inteligentes; Hyperledger Caliper

ABSTRACT

Livestock farming is one of Brazil's main economic activities. Given growing global concerns about climate change, Embrapa has launched concept brands that function as quality seals to encourage the adoption of more sustainable practices in the sector. This work aimed to develop and evaluate a blockchain infrastructure, as a proof of concept, for decentralized applications aimed at certifying products under the concept brands of Embrapa's Low Carbon Livestock Platform (PBC). The infrastructure was implemented with the Hyperledger Fabric framework on a set of virtual machines hosted on Amazon Web Services (AWS). Its performance was evaluated using Hyperledger Caliper, considering scenarios with different transaction volumes and sizes, as well as variations in the network ordering mechanism. The results indicated that the infrastructure stably supports a continuous write rate of 2250 KB/s. Conversely, average rates of 4500 KB/s exceed the network's capacity, resulting in a backlog of pending transactions. Transaction granularity did not significantly impact performance, while increasing the number of ordering nodes significantly reduced performance. We conclude that the proposed infrastructure is technically feasible to support certification applications linked to PBC brands, offering the potential to contribute to traceability and data reliability in the sustainable livestock production chain.

Keywords: Carne Carbono Neutro (Carbon Neutral Beef — Embrapa's quality label); Certification; Blockchain; Hyperledger Fabric; Smart Contracts; Hyperledger Caliper

LISTA DE FIGURAS

Figura 1 – Logotipos de marcas da Plataforma Pecuária de Baixo Carbono 12
Figura 2 – Encadeamento dos blocos em uma *blockchain* 14
Figura 3 – Modelo de avaliação de algoritmos de consenso 15
Figura 4 – Consórcio x Canais de Aplicação 24
Figura 5 – Interação com uma Rede Hyperledger Fabric 25
Figura 6 – Criação de um consórcio Hyperledger Fabric 30
Figura 7 – Arquitetura do consórcio no modo de ordenação Raft com 3 orderers . 32
Figura 8 – Arquitetura do consórcio no modo de ordenação Raft com 7 orderers . 33
Figura 9 – Arquitetura do consórcio no modo de ordenação Solo 33
Figura 10 – Arquivo crypto-config.yaml para geração dos materiais criptográficos . 34
Figura 11 – Organizações no configtx.yaml 36
Figura 12 – Seção Orderer no configtx.yaml 37
Figura 13 – Perfis para criação de consórcios no configtx.yaml 38
Figura 14 – Perfil para criação de canal no configtx.yaml 38
Figura 15 – Especificação de um Orderer no docker-compose.yaml 39
Figura 16 – Especificação de um Peer no docker-compose.yaml 41
Figura 17 – Especificação do Client no docker-compose.yaml 43
Figura 18 – Estados possíveis para um certificado 47
Figura 19 – Regras de entrada empregadas no grupo de segurança 48
Figura 20 – Restrição de alocação do serviço no Docker Swarm 49
Figura 21 – Estrutura de uma rodada de teste do Caliper 50
Figura 22 – Organização dos benchmarks 51
Figura 23 – Processo de execução dos benchmarks por modo de ordenação 51
Figura 24 – Metodologia de consolidação dos resultados obtidos 53
Figura 25 – Variação de latência média entre os cenários de teste 56
Figura 26 – Latência média e erro padrão nos cenários de teste 57
Figura 27 – Throughput da rede em cada cenário 59
Figura 28 – Throughput da rede nos cenários com baixo TPS 59
Figura 29 – Throughput da rede nos cenários com alto TPS 60
Figura 30 – Throughput da rede em Kilobytes 61
Figura 31 – Formação de Filas com Certificados Pequenos e Baixo TPS 62
Figura 32 – Formação de Filas com Certificados Pequenos e Alto TPS 62
Figura 33 – Formação de Filas com Certificados Médios e Baixo TPS 62
Figura 34 – Formação de Filas com Certificados Médios e Alto TPS 63
Figura 35 – Formação de Filas com Certificados Grandes e Baixo TPS 63
Figura 36 – Formação de Filas com Certificados Grandes e Alto TPS 63
Figura 37 – Formação de Filas com 1 Orderer em modo Solo 64

Figura 38 – Formação de Filas com 3 Orderers em modo RAFT 64
Figura 39 – Formação de Filas com 7 Orderers em modo RAFT 65
Figura 40 – Monitoramento de rede - Modo Solo 66
Figura 41 – Monitoramento de rede - Modo RAFT com 3 orderers 66
Figura 42 – Monitoramento de rede - Modo RAFT com 7 orderers 67

LISTA DE TABELAS

Tabela 1 – Síntese comparativa das plataformas *blockchain* analisadas 21
Tabela 2 – Síntese comparativa dos trabalhos relacionados 28
Tabela 3 – Dados do certificado 46
Tabela 4 – Taxa de envio de dados por segundo em cada cenário de teste 54
Tabela 5 – Estatísticas de Latência - 1 orderer Solo 54
Tabela 6 – Estatísticas de Latência - 3 orderers RAFT 55
Tabela 7 – Estatísticas de Latência - 7 orderers RAFT 55
Tabela 8 – Estatísticas de Throughput - 1 orderer Solo 58
Tabela 9 – Estatísticas de Throughput - 3 orderers RAFT 58
Tabela 10 – Estatísticas de Throughput - 7 orderers RAFT 58

LISTA DE ABREVIATURAS E SIGLAS

AMI	Imagem de Máquina da Amazon
API	Application Programming Interface
AWS	Amazon Web Services
BCN	Bezerro Carbono Neutro
BFT	Byzantine Fault Tolerance
CA	Certificate Authority
CBC	Carne Baixo Carbono
CCN	Carne Carbono Neutro
CH4	Gás Metano
CN	Carne Carbono Nativo
CNA	Confederação da Agricultura e Pecuária do Brasil
DApp	Descentralized Application
DLT	Distributed Ledger Technology
EC2	Amazon Elastic Compute Cloud
EVM	Ethereum Virtual Machine
GEE	Gás do Efeito Estufa
IBFT	Istanbul Byzantine Fault Tolerant
ILPF	Integração Lavoura-Pecuária-Floresta
IoT	Internet of Things
IPF	Integração Pecuária-Floresta
IPFS	InterPlanetary File System
KYC	Know your Costumer
LTS	Long-Term Support
MSP	Membership Service Provider
PBC	Plataforma Pecuária de Baixo Carbono
PBFT	Practical Byzantine Fault Tolerance
PKI	Infraestrutura de Chave Pública
PoA	Proof-of-Authority
PoS	Proof-of-Stake
PoW	Proof-of-Work
QBFT	Quorum Byzantine Fault Tolerance
TLS	Transport Layer Security
TPS	Transações por Segundo

SUMÁRIO

1 INTRODUÇÃO 10
2 FUNDAMENTAÇÃO TEÓRICA 12
2.1 PLATAFORMA PECUÁRIA DE BAIXO CARBONO 12
2.2 BLOCKCHAIN 13
2.2.1 Estrutura de uma Blockchain 14
2.2.2 Algoritmos de consenso 15
2.2.3 Contratos Inteligentes 16
2.2.4 Permissionadas e Não permissionadas 16
2.3 FRAMEWORKS PARA CRIAÇÃO DE BLOCKCHAINS 17
2.3.1 Hyperledger Fabric 18
2.3.2 Ethereum 19
2.3.3 Quorum 20
2.3.4 R3 Corda 20
2.3.5 Síntese Comparativa 21
2.4 ARQUITETURA DA HYPERLEDGER FABRIC 22
3 TRABALHOS RELACIONADOS 26
4 METODOLOGIA 30
4.1 ARQUITETURA DO CONSÓRCIO 31
4.2 CRIAÇÃO DA REDE HYPERLEDGER FABRIC 34
4.2.1 Materiais Criptográficos 34
4.2.2 Criação do Bloco Gênesis 35
4.2.3 Configuração do Serviço de Ordenação 39
4.2.4 Configuração dos Peers 40
4.2.5 Serviço Cliente 42
4.2.6 Criação de um Canal de Aplicação 43
4.2.7 Definição dos Anchor Peers 44
4.3 CRIAÇÃO E IMPLANTAÇÃO DO CONTRATO INTELIGENTE . . 45
4.4 IMPLANTAÇÃO DO CONSÓRCIO EM UM CLUSTER AWS 47
4.5 BENCHMARK COM HYPERLEDGER CALIPER 49
5 RESULTADOS 53
5.1 LATÊNCIA 54
5.2 THROUGHPUT 57
5.3 FORMAÇÃO DE FILAS 61
5.4 MONITORAMENTO DE RECURSOS NA AWS 65
5.5 DISCUSSÃO DOS RESULTADOS 67
6 CONSIDERAÇÕES FINAIS 69
REFERÊNCIAS 71

1 INTRODUÇÃO

Com um vasto território, clima e hidrografia favoráveis à agropecuária, de acordo com OECD e FAO (2024), o Brasil é um dos maiores fornecedores globais de alimentos e produtos agrícolas, liderando por exemplo, nas exportações de açúcar e soja. Além de ser um importante fornecedor de alimentos em geral, o Brasil é o segundo maior produtor e o maior exportador de carne bovina do mundo, tendo sido responsável em 2023 por 13,8% da produção global, atingindo um faturamento de 10,55 bilhões de dólares em receita, conforme ABIEC (2024). Além disso, considerando as crescentes preocupações com as mudanças climáticas, a Embrapa, por meio da pesquisa científica e da inovação, e o governo federal, guiado pelos planos ABC e ABC+, vêm buscando meios de otimizar os processos produtivos do setor como um todo, com ênfase para a redução do impacto ambiental da atividade agropecuária no Brasil (Brasil, 2012, 2023; Pellegrino et al., 2022).

A redução da emissão de carbono e de Gases do Efeito Estufa (GEEs) na atmosfera é uma questão que vem sendo trabalhada em diversos setores da economia, onde cada setor apresenta seus próprios desafios e soluções. No campo da pecuária, o desafio em destaque é o de reduzir ou compensar a emissão do Gás Metano (CH4) produzido pelos animais, onde, segundo Alves et al. (2019), o setor brasileiro já vem voluntariamente adotando iniciativas a fim de contribuir para a redução das emissões de GEEs.

A Embrapa, com o apoio de Alves et al. (2019), tem tomado um conjunto de medidas importantes, lançando, por exemplo, marcas-conceito que possibilitam que produtores contribuam formalmente com a redução das emissões de GEEs, adotando medidas estabelecidas pelas respectivas marcas, enquanto agregam valor ao seu produto no mercado nacional e internacional. As marcas-conceito da Embrapa, como a **Carne Carbono Neutro** (CCN), consistem em protocolos formais que determinam regras com relação aos projetos de produção pecuária que utilizem o sistema de integração lavoura-pecuária-floresta ou o pecuária-floresta a fim de formar um ciclo onde o componente arbóreo capture carbono da atmosfera compensando as emissões de gás metano pelos animais. Cumprir as regras determinadas no protocolo dá direito ao uso da marca à qual tal protocolo está vinculado (Alves et al., 2019).

As regras determinadas por tais protocolos exigem que os ciclos de produção sejam acompanhados e que diversas informações sejam continuamente registradas de forma confiável. Nesse contexto, a tecnologia de *blockchain* pode se mostrar como uma ferramenta viável para construção de um sistema que receba e armazene tais dados de forma transparente e auditável, trazendo consigo técnicas que garantem a imutabilidade dos dados, que possibilitam que diversas entidades colaborem na manutenção e verificação dos dados, e que torne a cadeia produtiva rastreável.

Desde a popularização do conceito de *blockchain* após 2008, têm surgido inúmeros trabalhos como os apresentados por Ghani et al. (2022), Vayadande et al. (2022) e Menezes,

Araújo e Nishijima (2023), que propõem modelos e aplicações baseados em *blockchain* e em contratos inteligentes visando criar sistemas de certificação para diferentes finalidades, que sejam confiáveis e transparentes mesmo em ambientes multiagentes. Há também muitos exemplos de aplicações desse gênero já em execução em ambientes corporativos como o apresentado por Kamath (2018).

Dado o contexto apresentado, este trabalho tem como objetivo geral **desenvolver uma infraestrutura de *blockchain* que sirva de base para a criação de aplicações descentralizadas voltadas para a certificação de produtos das marcas-conceito da Plataforma Pecuária de Baixo Carbono da Embrapa (PBC)**. Como objetivos específicos, destacam-se:

- Projetar e implementar uma rede *blockchain* para a PBC como prova de conceito.
- Explorar e apresentar diferentes formas de configuração, implantação e organização da rede.
- Criar, para fins de validação da rede, um contrato inteligente simplificado.
- Avaliar a capacidade da rede por meio de testes de desempenho em diferentes cenários.

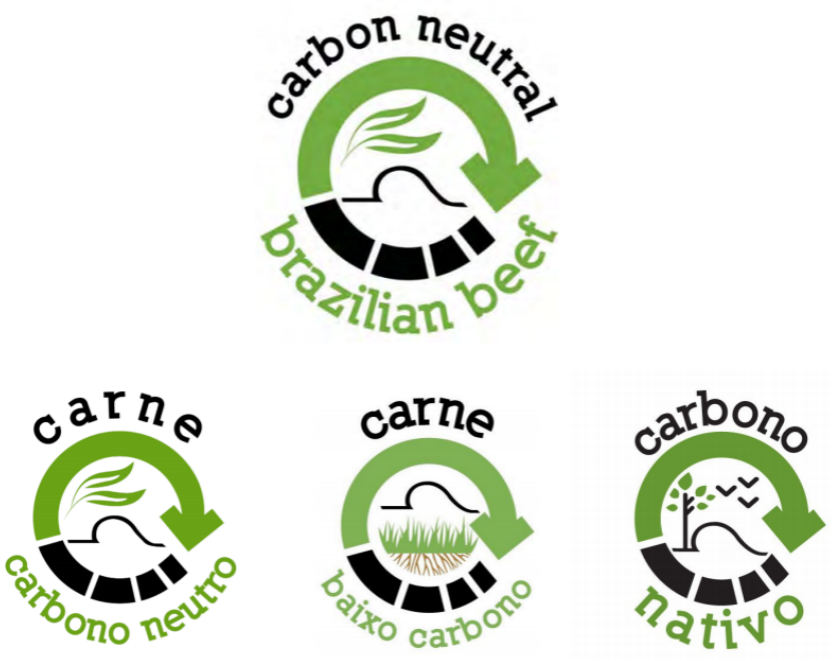
Contextualizada a temática da pesquisa e especificados os objetivos, o restante do trabalho está dividido em: **Capítulo 2**, onde é apresentada uma fundamentação teórica trazendo informações e conceitos base sobre a plataforma PBC da Embrapa e sobre tecnologia *blockchain*; **Capítulo 3**, onde são apresentados uma série de trabalhos relacionados que apresentam o estado da arte em *blockchain* e certificação; **Capítulo 4**, onde é explicada a metodologia adotada no desenvolvimento do trabalho; **Capítulo 5**, onde são apresentados e discutidos os resultados obtidos; e **Capítulo 6**, onde são realizadas as considerações finais.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 PLATAFORMA PECUÁRIA DE BAIXO CARBONO

A Embrapa, que já atua e promove os sistemas de Integração Lavoura-Pecuária-Floresta (ILPF) e Integração Pecuária-Floresta (IPF) há décadas, considerando o potencial que as árvores presentes nesses sistemas têm para fixar carbono, lançou em 2013 a marca-conceito **Carne Carbono Neutro** (CCN) e, posteriormente, as marcas **Carne Baixo Carbono** (CBC), **Carne Carbono Nativo** (CN), **Bezerro Carbono Neutro** (BCN) e **Couro Carbono Neutro**, conforme Alves et al. (2019). Tais marcas consistem em protocolos que, se seguidos pelo pecuarista, mediante adesão formal e acompanhamento junto da Embrapa, permitem que os produtos finais gerados dos rebanhos criados nos sistemas indicados pelos respectivos protocolos sejam comercializados no mercado nacional e internacional com selos que atestam que os mesmos foram produzidos de forma sustentável.

Figura 1 – Logotipos de marcas da Plataforma Pecuária de Baixo Carbono



Fonte: (Alves et al., 2019).

A Figura 1 traz os logotipos de algumas das marcas-conceito da plataforma PBC, apresentando o logotipo que identifica a marca CCN, bem como sua versão em inglês, grafada como *Neutral Carbon Brazilian Beef*, além das marcas Carne Baixo Carbono e Carbono Nativo. Cada marca possui um conjunto de regras e características que precisam ser satisfeitas, mas todas têm como finalidade promover a descarbonização voluntária da atividade pecuária de corte.

Os protocolos baseiam-se em informações técnicas que possibilitam a aplicação de

metodologias científicas a fim de se quantificar a emissão e o sequestro de carbono dentro dos projetos de produção que aderiram ao programa. Para a viabilidade da quantificação e das auditorias, é necessário o registro de uma série de informações. No caso específico da Carne Carbono Neutro, são exigidos os dados de georreferenciamento da área em que o sistema será implantado, informações de identificação dos animais para que os mesmos sejam rastreáveis, as datas em que os animais são inseridos e removidos da área, bem como seu peso vivo nas respectivas ocasiões, dados anuais de análises de solo, informações qualitativas e quantitativas do componente arbóreo a ser empregado, bem como sua destinação, entre outros. (Alves; Almeida; Laura, 2015).

Após o levantamento dos dados, os mesmos precisam ser registrados e precisam ser disponibilizados para consulta. Uma tecnologia que pode ser utilizada para realizar essa atividade de maneira transparente e com garantias de auditabilidade é a **blockchain**.

2.2 BLOCKCHAIN

A *blockchain* é definida como uma tecnologia de livro-razão distribuído, onde as partes mantêm cópias completas do registro compartilhado e as atualizações são realizadas, propagadas e incorporadas pelos participantes através de um algoritmo de consenso executado pelos mesmos (Polge; Robert; Le Traon, 2021). Ela foi proposta inicialmente por Haber e Stornetta (1991), que empregaram o termo *chain of time-stamps*[1], mas passou a ser conhecida como *blockchain* algum tempo após o surgimento do Bitcoin proposto por Nakamoto (2008).

Juntamente do conceito de *blockchain*, vem se tornando popular também a sigla DLT (Distributed Ledger Technology, ou em português, "Tecnologia de livro-razão distribuído"), que muitas vezes são tidos como sinônimos. Nesse cenário, Panwar e Bhatnagar (2020) explicam que DLTs são variadas formas de bancos de dados digitais que são atualizados e mantidos por múltiplos membros independentes sem uma autoridade central.

A *blockchain* pode ser considerada como um subtipo de DLT contendo adicionalmente algumas características mais específicas, onde destacam-se o mecanismo de sequência e encadeamento criptográfico dos dados, em forma de blocos, e a necessidade de algoritmos de consenso para a criação de novos blocos (Panwar; Bhatnagar, 2020). Diferentemente de outras arquiteturas de DLT, que podem adotar modelos de governança, mecanismos de consenso e formatos de registro variados, a *blockchain* utiliza o encadeamento criptográfico dos dados como elemento central. Essa arquitetura garante que, uma vez registrados, os dados não possam ser removidos ou alterados, permitindo apenas inserções sucessivas. Por esse motivo, a *blockchain* é frequentemente descrita como uma estrutura *append-only*, cuja imutabilidade tem se mostrado útil em diversos setores.

[1] Em português, "corrente de carimbos de data e hora".

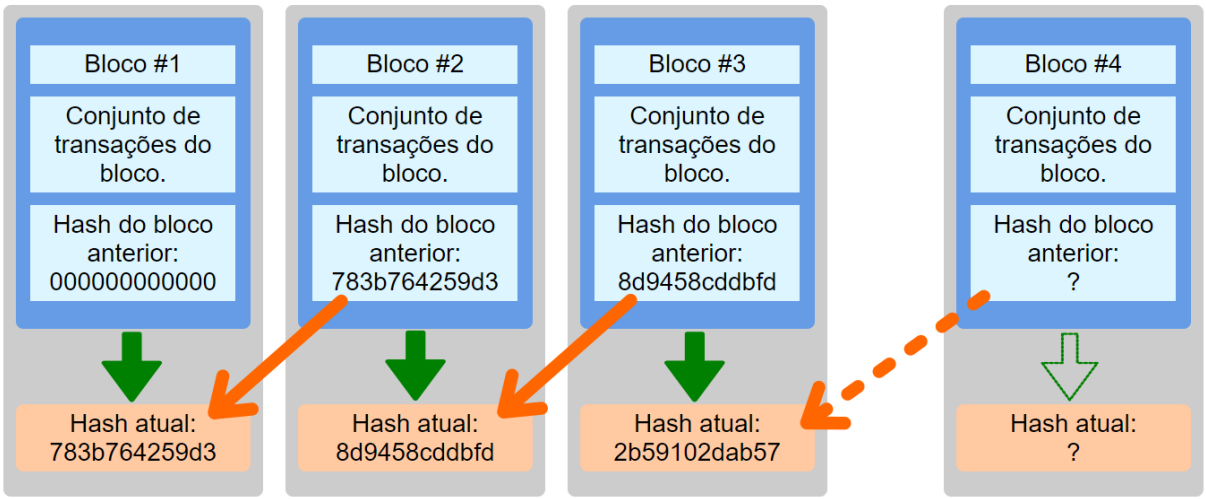
2.2.1 Estrutura de uma Blockchain

Conforme Antonopoulos (2017), uma estrutura de dados em *blockchain* é uma lista ordenada de blocos contendo transações, onde cada bloco é criptograficamente ligado ao bloco anterior. Em essência, cada bloco é identificado pela *hash* calculada a partir de seu próprio conteúdo e cada bloco armazena em seu conteúdo, dentre outros dados, a *hash* do bloco anterior. Desta forma, a sequência de *hashes* que liga cada bloco ao bloco imediatamente anterior cria uma corrente de blocos que leva até o primeiro bloco da história de uma *blockchain*.

A Figura 2 ilustra a lógica de funcionamento deste encadeamento de blocos, onde podemos observar três blocos confirmados na *blockchain* e há um quarto bloco ainda pendente, os passos seguintes para a adição desse bloco, respeitado o protocolo de consenso empregado, são:

1. Adicionar a *hash* do bloco 3 no campo `hash do bloco anterior` do bloco 4;
2. Calcular a *hash* do bloco 4 utilizando os dados do respectivo bloco como entrada, incluindo o valor que foi inserido no campo `hash do bloco anterior`;
3. Homologar o bloco na *blockchain* e propagá-lo para os demais nós da rede.

Figura 2 – Encadeamento dos blocos em uma *blockchain*



Fonte: O autor.

De acordo com a Figura 2, o campo `hash do bloco anterior` dentro de um bloco afeta sua *hash* resultante, uma alteração qualquer no conteúdo do bloco anterior modificaria sua *hash* resultante, ou seja, sua identidade, o que demandaria que o campo `hash do bloco anterior` fosse atualizado no bloco atual, o que por sua vez modificaria sua própria *hash*, demandando que o bloco seguinte também fosse atualizado, e assim por diante.

O efeito em cascata faz com que quaisquer modificações em blocos da *blockchain* torne uma cópia da *blockchain* inválida perante as cópias que estão armazenadas em outros

nós da rede. E, à medida que um bloco se torna mais antigo na rede, ou seja, à medida que ele é precedido por mais e mais blocos, mais difícil se torna corrompê-lo ou invalidá-lo (Antonopoulos, 2017).

Conforme Shinde et al. (2019), uma *blockchain* é uma série de registros de dados imutáveis com carimbo de data e hora que é gerenciada por um *cluster* de computadores que não pertence a nenhuma entidade única.

Para que essa atividade seja realizada de forma coordenada e sem um concentrador ou um coordenador, é necessário que haja mecanismos de consenso distribuído para garantir a integridade e o funcionamento da rede (Karamachoski; Marina; Taskov, 2020).

2.2.2 Algoritmos de consenso

Além da imutabilidade de dados garantida pelo mecanismo de encadeamento de blocos, considerando que uma blockchain é uma base de dados distribuída onde cada novo bloco de dados deve estar conectado ao bloco anterior, é necessário que haja um mecanismo que coordene o processo de adição de novos blocos, tais mecanismos são denominados como "algoritmos de consenso" (Xiao et al., 2020).

Karamachoski, Marina e Taskov (2020) explicam que o mecanismo de consenso provê procedimentos para sincronização descentralizada dos dados, viabilizando a construção de uma rede colaborativa sem a necessidade de haver confiança entre os membros. Os autores afirmam ainda que cada algoritmo de consenso tem suas vantagens e desvantagens, sendo que os dentre os mais populares, o PoW (Proof-of-Work) apresenta propriedades mais confiáveis para o uso em redes públicas, mas por outro lado é ineficiente no uso de energia elétrica, o PoS (Proof-of-Stake), se comparado ao PoW, é menos confiável, mas apresenta um consumo de energia elétrica moderado, e o PBFT (Practical Byzantine Fault Tolerance) tem um baixo consumo de energia mas é inviável para *blockchains* públicas, tendo sido projeto para redes fechadas.

Figura 3 – Modelo de avaliação de algoritmos de consenso

Fonte: Adaptado de Shi et al. (2021).

Conforme ilustradado na Figura 3, os três principais critérios para se avaliar um algoritmo de consenso são eficiência, segurança e descentralização, onde cada algoritmo tende a demonstrar competência em um ou dois destes critérios em detrimento do(s) outro(s). Há diversos trabalhos publicados recentemente como os de Biswas et al. (2019), Bitansky (2020), Chen et al. (2022) e Li et al. (2020) propondo algoritmos de consenso que atinjam melhores resultados dentro destas métricas ou que ofereçam um balanceamento distinto para emprego em contextos específicos.

Outro mecanismo que se destaca dentro do ambiente de *blockchain* são os acordos que podem ser realizados ou a presença de contratos inteligentes, também conhecidos como *smart contracts* (Szabo, 1997).

2.2.3 Contratos Inteligentes

Os conceitos iniciais sobre contratos inteligentes partem de uma proposta de automação de contratos publicada por Szabo (1997), antes mesmo da ascensão da tecnologia *blockchain*. Apesar de contratos inteligentes serem comumente associados à tecnologias de DLTs e *blockchain*, os mesmos podem ser implementados em plataformas centralizadas. O termo "contrato inteligente" tem um significado de baixo nível em plataformas DLTs, "onde é usado para descrever um código replicado que é executado de forma síncrona em múltiplos nós de um livro-razão distribuído" (Clack; McGonagle, 2019).

Shinde et al. (2019) complementam que os contratos inteligentes possibilitam a realização de transações confiáveis sem a necessidade de intermediários, de forma que podem ser estabelecidas regras que definem qual ação deve ser executada quando determinado evento ocorrer. É possível, por exemplo, programar um contrato inteligente que intermedie uma transação de compra e venda *online*, recebendo e armazenando o valor referente a uma compra, de forma que o valor seja enviado ao vendedor somente após a confirmação de entrega pela empresa de logística, ou que o valor seja devolvido ao comprador nas hipóteses de a empresa de logística informar que não pôde efetivar a entrega ou de o prazo limite de entrega expirar.

Técnicas como a de encadeamento de blocos e emprego de protocolos de consenso estão presentes em quaisquer plataformas de *blockchain*. Os contratos inteligentes, por sua vez, estão presentes na maioria das *blockchains*, não em todas, e a forma como tais técnicas são implementadas varia consideravelmente de uma plataforma para outra, especialmente quando comparamos plataformas públicas com plataformas corporativas.

2.2.4 Permissionadas e Não permissionadas

Desde o surgimento do Bitcoin em 2009, sendo a primeira aplicação prática baseada em *blockchain*, dois tipos de *blockchains* se estabeleceram - as públicas, também chamadas de "não permissionadas", e as privadas, também chamadas de "permissionadas" (Swan, 2015).

Segundo Swan (2015), uma *blockchain* não permissionada pode ser descrita como uma *blockchain* que possibilita que cada registro seja compartilhado por todos os usuários da rede, atualizado pelos mineradores, monitorado por todos, mas que pertença e seja controlado apenas por um usuário.

Liu, Wu e Xu (2019) acrescenta que numa rede não permissionada qualquer indivíduo ou organização pode se juntar à rede por meio de um computador ou dispositivo móvel, tendo como benefício a forte descentralização e o já comprovado sucesso de diversas criptomoedas, que se incluem nesta categoria de *blockchain*. Por outro lado, há desvantagens como a velocidade limitada no processamento de grandes volumes de transações comparando-se a sistemas de pagamento como Visa e Mastercard, além das restrições impostas por estas redes no que diz respeito à privacidade.

Uma *blockchain* permissionada, por sua vez, possibilita a gestão de seus membros, limitando quem pode se juntar à rede, bem como controlando os procedimentos e dados que cada membro pode executar ou acessar. Devido ao fato de diferentes membros terem diferentes níveis de autorização, esta categoria de *blockchain* é muitas vezes considerada como parcialmente descentralizada. Por um lado, com uma configuração apropriada de controle de acesso, uma *blockchain* permissionada pode atender aos requisitos de privacidade e governança de negócios, mas por outro lado, caso uma entidade na rede tenha privilégios excessivos, de sobrescrita, por exemplo, a credibilidade da *blockchain* pode ser minada. (Liu; Wu; Xu, 2019).

Helliar et al. (2020) afirmam que as *blockchains* não permissionadas são utilizadas principalmente no contexto de criptomoedas, enquanto que *blockchains* permissionadas vêm sendo adotadas no contexto corporativo, para negócios e práticas institucionais.

2.3 FRAMEWORKS PARA CRIAÇÃO DE BLOCKCHAINS

Com a rápida popularização da tecnologia *blockchain* e do conceito de contratos inteligentes, diversas possibilidades passaram a ser vislumbradas também no contexto empresarial e, com isso, diversas soluções de *frameworks*/plataformas de *blockchain* permissionadas emergiram. Dentre elas, algumas se tornaram mais populares, sendo adotadas na indústria e debatidas no universo acadêmico.

Saraf e Sabadra (2018) realizam uma análise comparativa entre as plataformas Hyperledger[2], R3 Corda[3] e Ethereum[4]. Kuo, Rojas e Ohno-Machado (2019) realizam uma revisão sistemática buscando identificar e apresentar as dez plataformas de blockchain mais populares, dentre as quais três das identificadas são permissionadas: Ethereum (versão privada), MultiChain[5] e um grupo de plataformas da Hyperledger Foundation.

[2] https://www.hyperledger.org/projects
[3] https://r3.com/products/corda/
[4] https://ethereum.org/en/developers/docs/
[5] https://www.multichain.com/

Pongnumkul, Siripanpornchana e Thajchayapong (2017) faz uma análise comparativa de desempenho entre Hyperledger Fabric e Ethereum. Capocasale, Gotta e Perboli (2023) comparam as plataformas Hyperledger Fabric, Hyperledger Sawtooth[6] e ConsenSys Quorum[7]. Oliveira et al. (2019) avaliam a performance das plataformas Parity[8] e MultiChain. Monrat, Schelén e Andersson (2020) comparam as plataformas Hyperledger Fabric, Ethereum, Quorum e R3 Corda. Polge, Robert e Le Traon (2021) comparam as mesmas plataformas, mas incluindo a MultiChain.

Considerando a popularidade das plataformas frequentemente adotadas pela indústria, bem como algumas características que são interessantes de se encontrar na plataforma *blockchain* a ser adotada para o desenvolvimento do presente trabalho, serão apresentados a seguir as características gerais das plataformas Hyperledger Fabric, Ethereum, Quorum e R3 Corda.

2.3.1 Hyperledger Fabric

A Hyperledger Fabric é um *framework open source* criado e mantido pela Hyperledger Foundation, que por sua vez é hospedada pela Linux Foundation. Conforme Androulaki et al. (2018), trata-se de um sistema para criação de *blockchains* permissionadas capazes de executar aplicações descentralizadas escritas em linguagens de propósito geral como Go, Java ou Javascript.

Hyperleger Foundation (2023) afirma que a arquitetura modular da Hyperledger Fabric acomoda a diversidade de casos de uso corporativos por meio de componentes conectáveis, como algoritmos de consenso, privacidade e serviços de associação.

Segundo Hyperledger Foundation (2025), numa rede Hyperledger Fabric há (i) *Certificate Authorities* (CAs), responsáveis por administrar a rede, além de fornecer identidades e permitir o ingresso de novos *nodes*; (ii) *Peer nodes*, responsáveis por armazenar cópias do livro-razão, também chamado de *ledger*, e também por aprovar ou "endossar" transações, conforme terminologia usada pela plataforma; (iii) *Orderer nodes*, responsáveis por ordenar transações aprovadas e enviá-las para os *peers* as armazenarem em suas *ledgers*; (iv) *Clients*, responsáveis por criar, assinar e submeter transações para a rede. Nesta rede, os *peers* não participam do consenso, questão que é tratada exclusivamente pelos *Orderer nodes*.

Além deste *framework* de propósito geral, a Hyperledger Foundation conta com *frameworks* de *blockchain* de propósitos específicos como Iroha, Indy, Bevel, Besu e Sawtooth.

[6] https://www.hyperledger.org/hyperledger-sawtooth-1-0
[7] https://docs.goquorum.consensys.io/
[8] https://www.parity.io/

2.3.2 Ethereum

Uma rede Ethereum é uma rede *blockchain* que, conforme Ethereum (2025), possui um "computador embutido", sendo a fundação para construção de aplicações e organizações descentralizadas, não permissionadas e resistentes à censura. Tal "computador" faz referência a EVM (Ethereum Virtual Machine) que simula o comportamento de um computador executando aplicações, sendo que, neste caso, as aplicações são os *smart contracts* que são enviados para a rede e o computador abstrato parte do conceito de que, apesar de cada nó da rede ser na prática um computador físico diferente, esses nós operam sob um protocolo de consenso que os levam a concordar continuamente a respeito do estado da rede. Cada nó mantém uma cópia do estado da rede e este estado diz respeito aos dados que foram e estão sendo armazenados numa estrutura de *blockchain*.

Segundo Ethereum (2025), um nó na rede Ethereum pode atuar como:

- um ***Full node***: um nó completo no que se refere aos procedimentos da rede, realizando a validação integral de cada bloco da *blockchain*, participando do mecanismo de consenso e atuando como provedor de serviços para a rede, respondendo a requisições de submissão de transações e de consulta a dados. Um *full node* não armazena todo o histórico de transações da rede, mas pode recuperá-los instantaneamente, caso necessário, a partir de um *archive node*.
- um ***Archive node***, que herda funcionalidades do *full node*, mas assume o compromisso de manter todo o histórico de transações da rede, ou seja, manter armazenados todos os blocos desde o gênese. Esse tipo de nó pode mas não é obrigado a participar no consenso da rede.
- um ***Light node***: nós leves, voltados para execução em dispositivos de baixo desempenho. Esses nós não verificam o conteúdo dos blocos, mas apenas a sequência de hashes de seus cabeçalhos. As implementações existentes atualmente dessa categoria de nós não estão habilitadas a participarem do consenso da rede.

Um nó Ethereum deve executar dois clientes, um de execução e outro de consenso. Há várias implementações desenvolvidas por diferentes organizações em variadas linguagens tanto para os clientes de execução, destacando-se Go Ethereum (Geth), Nethermind, Hyperledger Besu e Erigon, quanto para os clientes de consenso, onde destacam-se Lighthouse, Lodestar, Nimbus, Prysm e Teku. Todas as implementações são desenvolvidas com base nas especificações oficiais da rede Ethereum e tal variedade é importante para que a rede não fique totalmente exposta no caso de existência de vulnerabilidade em uma hipotética implementação única ou dominante. (Ethereum, 2025).

O Ethereum é voltado predominantemente para o contexto de *blockchain* não permissionada, mas redes privadas (permissionadas) podem ser configuradas com a customização de alguns poucos parâmetros. Segundo Go-Ethereum (2024), a rede pública

do Ethereum, apelidada como “Mainnet”, é identificada por um ID de rede igual à "1", para criar uma rede permissionada com o Ethereum, é necessário executar um ou mais nós usando algum outro ID de rede que esteja disponível, configurar um algoritmo de consenso e adicionar algum mecanismo de controle de acesso. A Mainnet do Ethereum utiliza atualmente o algoritmo de consenso PoS, mas a maioria das implementações do Ethereum vem acompanhada do PoA (Proof-of-Authority) e do PoW como opções para criação de redes privadas.

2.3.3 Quorum

O Quorum, também chamado de GoQuorum, é uma plataforma de *blockchain* baseada no Ethereum, sendo desenvolvida como um *fork* do cliente Go-Ethereum, onde foram adicionados mecanismos de privacidade e novos algoritmos de consenso. Conforme ConsenSys (2020), a plataforma foi desenvolvida pela líder global de serviços financeiros J.P. Morgan e lançada em 2016, tendo sido adquirida pela companhia especializada em *blockchain* Consensys no ano de 2020.

Conforme ConsenSys (2025), além do mecanismo de consenso PoA padrão do Go-Ethereum, o Quorum oferece os algoritmos QBFT (Quorum Byzantine Fault Tolerance), IBFT (Istanbul Byzantine Fault Tolerant) e Raft. A plataforma traz recursos prontos para permissionamento de nós, possibilitando o controle de ingresso de novos nós na rede, plugins para gerenciamento de contas e suporte a transações e contratos privados. Apesar de seu desenvolvimento ter sido voltado para a indústria de serviços financeiros, suas características gerais não demonstram limitações quanto a sua aplicação em outros cenários.

2.3.4 R3 Corda

Corda é uma plataforma DLT *open source* para aplicações descentralizadas, suportando *smart contracts* escritos em quaisquer linguagens compatíveis com a Java Virtual Machine. O Corda foi desenvolvido de forma independente, com enfoque em atender às necessidades de instituições financeiras fortemente regulamentadas, onde se faz necessário algum nível de KYC (Know your Costumer), ajuste de privacidade, compatibilidade com os processos comumente adotados por bancos e instituições financeiras e alta escalabilidade (R3, 2025b).

Diferentemente de outras plataformas DLTs, o Corda não cria uma cadeia de blocos, mas sim uma cadeia de transações e, uma vez que as transações são registradas de maneira independente, o Corda permite configurações de privacidade mais apuradas, possibilitando que os dados das transações somente sejam visíveis para os envolvidos, enquanto os demais nós registram apenas os cabeçalhos das transações a fim de manter a cadeia de transações integra e conectada. Para que seja possível a validação das transações mantendo a privacidade das mesmas, em uma rede Corda existem *clusters* notariais que

previnem gastos duplos, operam como autoridades de registro de data e hora e podem também, opcionalmente, validar transações, sendo que cada *cluster* notarial pode estar executando um algoritmo de consenso diferente. Ao propor uma transação que movimente fundos, o proponente deverá selecionar um cluster notarial para processar tal transação (R3, 2025a)

Apesar da sua forte afinidade com a indústria de serviços financeiros, R3 (2025a) afirma que os recursos da plataforma também são aplicáveis em um conjunto mais amplo de mercados e indústrias, onde há necessidade de fluxos de trabalho multipartidários gerenciando transações, dados e processos para estabelecer uma visão compartilhada de um conjunto distribuído de dados, incluindo indústrias como as de seguros, de cadeia de suprimentos e de saúde.

2.3.5 Síntese Comparativa

A Tabela 1 sintetiza as principais características das plataformas analisadas, classificando-as quanto à categoria de permissionamento, linguagens suportadas para o desenvolvimento de contratos inteligentes, algoritmos de consenso tipicamente empregados, nível de privacidade das transações e enfoque predominante de aplicação.

Tabela 1 – Síntese comparativa das plataformas *blockchain* analisadas

	Categoria	**Linguagens**	**Consenso**	**Enfoque**	**Privacidade**
Hyperledger Fabric	Permissionada	Go, Java, JS	Raft	Corporativo	Nativa e configurável
Ethereum – Mainnet	Não permissionada	Solidity, Vyper	PoS	Aplicações públicas	Pública
Ethereum – Redes privadas	Tipicamente permissionada	Solidity, Vyper	PoA, PoW	Corporativo	Configurável
Quorum	Permissionada	Solidity, Vyper	PoA, QBFT, IBFT, Raft	Corporativo	Transações privadas
R3 Corda	Permissionada	JVM (Java, Kotlin)	Raft ou BFT-Smart	Financeiro	Granular por transação

Fonte: O autor.

A rede principal do Ethereum, por se tratar de uma infraestrutura pública, pode apresentar custos de transação elevados e variáveis no longo prazo. O Corda, por sua vez, é fortemente orientado ao setor financeiro, de modo que sua adoção em outros contextos pode demandar adaptações arquiteturais relevantes.

A implementação de uma rede Ethereum privada com as características requeridas pela PBC envolve a seleção e integração de componentes adicionais, incluindo mecanismos de consenso e soluções de permissionamento oferecidas por terceiros, o que amplia a complexidade de configuração e validação.

Entre as plataformas voltadas ao ambiente corporativo, a Hyperledger Fabric e a Quorum, observou-se, no período de realização desta pesquisa, maior grau de maturidade documental, volume de materiais técnicos e recorrência em estudos acadêmicos no caso da Hyperledger Fabric. Esses fatores, aliados à sua arquitetura modular e nativamente permissionada, contribuíram para a sua escolha como base para a infraestrutura proposta.

2.4 ARQUITETURA DA HYPERLEDGER FABRIC

A Hyperledger Fabric é uma plataforma de *blockchain* permissionada desenvolvida com foco em ambientes corporativos, oferecendo uma arquitetura modular e altamente customizável. Esse desenho modular é essencial para garantir a interoperabilidade entre diferentes organizações participantes de um consórcio, mesmo que estejam operando com sistemas operacionais distintos, tecnologias de hospedagem variadas ou diferentes configurações de rede. Para viabilizar a execução dos serviços da rede, a Hyperledger Foundation disponibiliza tanto binários pré-compilados (para Linux e MacOS) quanto imagens de contêineres compatíveis com Docker e Podman.

A execução da rede baseada em contêineres pode ser organizada de três formas distintas:

- de forma local, com todos os *peers* e *orderers* em uma única máquina;
- de forma remota, com múltiplas máquinas físicas ou virtuais distribuindo os serviços;
- ou utilizando um *cluster* gerenciado por ferramentas de orquestração como Docker Swarm ou Kubernetes.

Embora a execução local seja útil para testes simples, ela se mostra inadequada para avaliações de desempenho, pois todos os serviços competem pelos mesmos recursos. Já a execução remota com múltiplos nós independentes reflete melhor o comportamento de uma rede real, porém sua configuração e manutenção são mais complexas. O uso de orquestradores, por sua vez, representa um meio termo entre fidelidade e operacionalidade, facilitando a gestão e automação da rede.

Na Hyperledger Fabric, diferentes tipos de atores (*peers*, *orderers*, usuários e aplicações cliente) necessitam de uma identidade digital. Conforme especificado pela Hyperledger Foundation (2025), essas identidades são implementadas por meio de certificados digitais X.509. Esses certificados fazem parte da Infraestrutura de Chave Pública (PKI) e contêm, entre outros campos, uma chave pública, o nome da autoridade certificadora, um número de série e uma assinatura digital (ITU, 2024).

A Fabric utiliza certificados X.509 em dois contextos principais: (i) para identificação lógica dos atores do consórcio, por meio do MSP (Membership Service Provider), e (ii) para segurança da comunicação via protocolo TLS (Transport Layer Security). No primeiro caso, os certificados organizacionais são responsáveis por definir quais usuários e

serviços têm permissão para interagir com a rede. Esses certificados não contêm informações de rede como IP ou domínio, garantindo flexibilidade na movimentação dos serviços. Já os certificados TLS exigem o uso de informações de rede para garantir a segurança das conexões TCP/IP. Essa separação permite que mudanças de endereço IP não afetem a identidade organizacional de um ator.

A emissão dos certificados pode ser feita via Fabric CA ou pelo utilitário cryptogen. A Fabric CA, indicada para ambientes de produção, permite o gerenciamento completo de identidades, com suporte a registro, renovação e revogação. Já o cryptogen, mais apropriado para testes, gera os certificados de forma estática e não permite alterações dinâmicas na rede. A estrutura gerada inclui os certificados públicos, as configurações do MSP e, no caso de cada ator, suas respectivas chaves privadas, que devem ser mantidas isoladamente por questões de segurança. Os serviços da rede mantêm uma estrutura de diretórios semelhante àquela gerada pelo cryptogen. Mesmo em redes que utilizam a Fabric CA, os diretórios precisam seguir o mesmo padrão para garantir compatibilidade entre os *peers* e *orderers*. Essa estrutura possibilita a interoperabilidade entre organizações e a verificação mútua das identidades.

A configuração inicial de uma rede Hyperledger Fabric é feita a partir do arquivo `configtx.yaml`, cujo modelo[9] é fornecido no repositório `fabric-samples` e seu conteúdo é dividido em seis seções:

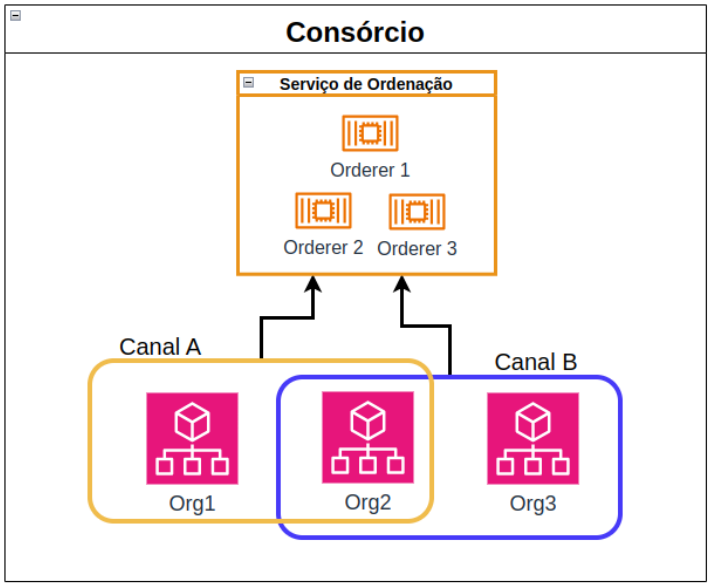
- **Organizations:** Define as identidades organizacionais do consórcio;
- **Capabilities:** Define os recursos que a rede suportará por meio da especificação de versões requeridas para participação de *orderers* e *peers* no consórcio.
- **Application:** Define configurações específicas para os canais de aplicação, que são usados pelas organizações para processar transações e executar *chaincodes*, como são referidos os contratos inteligentes na Fabric.
- **Orderer:** Define as configurações relacionadas ao processo de ordenação, incluindo o protocolo de consenso, os dados dos *orderers* participantes, tempo e tamanho do bloco, entre outras.
- **Channel:** Define as políticas de gestão do canal, especificando, por exemplo, quem pode atualizar o canal ou adicionar novas organizações.
- **Profiles:** Nesta seção, as definições inseridas nas seções anteriores são empregadas na criação de perfis de configuração para a geração do bloco gênesis do consórcio e dos blocos gênesis dos canais de aplicação.

Um dos principais diferenciais da Hyperledger Fabric é a criação de canais de aplicação. Esses canais, conforme ilustrado pela Figura 4, permitem a formação de subgrupos

[9] Disponível em https://github.com/hyperledger/fabric-samples/blob/main/test-network/configtx/configtx.yaml

dentro do consórcio, onde transações e dados são visíveis apenas às organizações participantes do canal, promovendo privacidade e confidencialidade. Cada canal possui sua própria *ledger*, independente da *ledger* do consórcio, e é mantido por *orderers* e pelos *peers* das organizações envolvidas. A *ledger* do consórcio, por sua vez, armazena informações administrativas e de configuração. A Fabric também permite definir, por canal, quais entidades têm permissão de leitura, escrita, administração e endosso de transações, sendo este último o processo de validação e simulação que precede a submissão das transações ao serviço de ordenação.

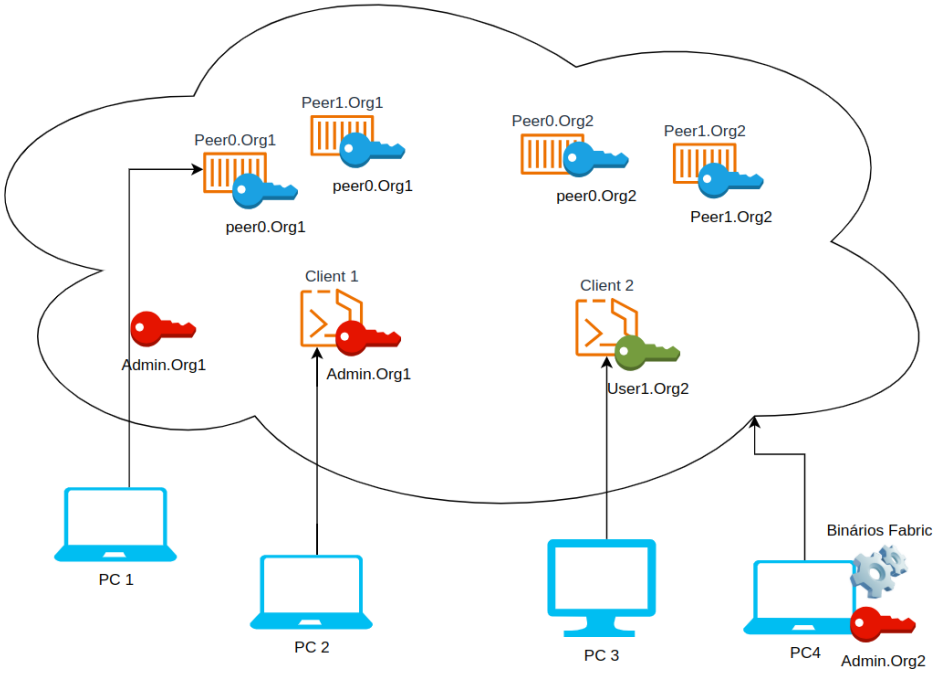
Figura 4 – Consórcio x Canais de Aplicação



Fonte: O autor.

Uma vez instanciada uma rede Hyperledger Fabric, as interações com a mesma exigem uma identidade digital e podem ocorrer de diferentes formas. É possível interagir diretamente a partir de um computador que tenha a identidade digital de um usuário do consórcio, conforme ilustrado na Figura 5 pelo PC 4. No contexto do Docker, é possível acessar o terminal do contêiner de um *peer* e executar ações usando a identidade do próprio *peer*, ou uma outra identidade, sendo que neste último caso, essa outra identidade deve ser copiada para o contêiner do *peer*, conforme ilustrado na Figura 5 pelo PC 1.

Figura 5 – Interação com uma Rede Hyperledger Fabric



Fonte: O autor.

Executar transações diretamente no terminal de um *peer* não é recomendado pois viola o isolamento do serviço, podendo colocar sua segurança e consistência em risco. Por outro lado, executar transações a partir de um computador pessoal demanda da configuração dos binários da Fabric e da gestão manual da identidade de usuário empregada. Desta forma, na Figura 5 é ilustrado pelos PCs 2 e 3, o emprego de contêineres Docker baseados na imagem `hyperledger/fabric-tools`, como ponto de partida para a gestão da rede, onde tais PCs acessam os contêineres `Client 1` e `Client 2` e, partir deles, interagem e administram a rede.

Esse conjunto de práticas e estruturas torna a Hyperledger Fabric uma plataforma robusta e adaptável, capaz de atender às exigências técnicas e operacionais de diferentes cenários corporativos e acadêmicos de uso de *blockchain*.

No próximo capítulo, serão apresentados trabalhos relacionados que abordam a aplicação de tecnologias de blockchain na construção de soluções voltadas à validação, certificação e rastreabilidade de informações. Esses estudos ajudam a contextualizar esta pesquisa e a mostrar como a proposta desenvolvida se insere no cenário atual.

3 TRABALHOS RELACIONADOS

Embora a tecnologia *blockchain* tenha suas origens no setor financeiro, ela já é amplamente empregada em diversas outras áreas, tanto em aplicações reais quanto em estudos de caso. Em virtude da vasta pesquisa nessa área, este capítulo apresenta estudos recentes que propuseram e/ou avaliaram sistemas baseados em *blockchain* para a validação e certificação de dados por meio de contratos inteligentes.

Sunny et al. (2022), com a realização de uma ampla revisão sistemática da literatura, identificam e organizam hierarquicamente inúmeras áreas de estudo onde aplicações baseadas em blockchain vêm sendo propostas. Tais áreas foram categorizadas em gestão financeira, segurança e privacidade, IoT (Internet of Things), saúde, logística, administração pública e educação, cada qual com diversas subcategorias.

Ghani et al. (2022) desenvolveram uma aplicação baseada em *blockchain* para emissão, armazenamento e compartilhamento de certificados de graduação. A aplicação consiste numa *blockchain* associada a um *web service* onde poderiam ser emitidos e armazenados eletronicamente certificados de todas as instituições de ensino superior do Iraque. Nessa aplicação, Ghani et al. (2022) utilizaram o *framework* Hyperledger Fabric devido a características como (i) ser uma rede permissionada; (ii) suportar transações confidenciais; (iii) ser capaz de funcionar sem um criptomoeda; e (iv) ser programável. Os resultados dos testes de desempenho foram considerados satisfatórios, com um *throughput* de 1000 transações a cada 2 segundos, porém, observou-se que o tempo de processamento das transações cresce exponencialmente com o aumento do volume de solicitações.

Dimitriou (2020) apresenta um sistema de votação baseado em *blockchain* que busca ser seguro e satisfazer propriedades básicas esperadas de um processo eleitoral sólido e confiável. Esse trabalho consistiu na construção de um protocolo de votação e na implementação de um protótipo onde utilizou-se a Hyperledger Fabric baseado nos resultados dos testes de desempenho realizados por Androulaki et al. (2018) e Gorenflo et al. (2019). Conclui-se que, embora a característica *append-only* possa ser implementada por outros mecanismos já presentes em sistemas de votação robustos, a *blockchain* se destaca para sistemas de votação distribuídos por permitir que múltiplas entidades mantenham a consistência da rede de forma conjunta e confiável.

Com sua crescente população, a Índia vem enfrentando dificuldades com o gerenciamento de registros de imóveis, nesta temática Vayadande et al. (2022) propõem um modelo para registro de imóveis baseado em *blockchain*. Nesse modelo, os dados básicos dos registros de posse de imóveis são armazenados em uma *blockchain*, enquanto que toda a documentação relacionada à transação de compra e registro, que demanda de maior espaço de armazenamento, é mantida em bancos de dados tradicionais a fim de não onerar a rede. A prova de conceito para o modelo proposto foi desenvolvida utilizando-se a plataforma Ethereum e, além da *blockchain*, criou-se um portal *web* para disponibilizar

as funcionalidades da aplicação aos usuários, intermediando o acesso à *blockchain*. Shinde et al. (2019), propôs uma solução parecida, mas que armazena os documentos oficiais relacionados às operações de compra e registro no IPFS (InterPlanetary File System) e grava sua *hash* na *blockchain* junto dos demais dados de posse do imóvel.

No contexto de serviços cartorários, Menezes, Araújo e Nishijima (2023) propõem uma arquitetura baseada em *blockchain* e contratos inteligentes para digitalizar serviços de cartório respeitando a legislação brasileira vigente. Nesta proposta, os autores empregaram a rede Ethereum, o sistema IPFS e o serviço de certificados digitais do ICP Brasil para desenvolver uma aplicação que possibilita a inserção, a coleta de assinaturas e a homologação de documentos de forma digitalizada. Nesta aplicação, um portal *web* é disponibilizado para que os usuários tenham acesso às suas funcionalidades, enquanto contratos inteligentes na *blockchain* ficam responsáveis por avaliar a validade das assinaturas, registrar data e hora e homologar um documento quando todas as assinaturas digitais necessárias são fornecidas ao mesmo.

Elmay et al. (2022), visando tornar o processo logístico de contêineres de transporte marítimo mais eficiente, reduzindo o tempo de desembaraço e potencialmente reduzindo os prejuízos com roubo e perda de mercadorias, propõem um modelo para gestão e rastreamento de contêineres. Nessa proposta, os autores adotam a rede principal da Ethereum para construção da solução, ressaltando a viabilidade de se empregar alternativamente uma rede privada. Além disso, considerando o grande volume de dados que precisa ser armazenado descrevendo o contrato de transporte do contêiner e as mercadorias que são carregadas nele, é empregado o IPFS para suprir essa demanda por armazenamento.

Segundo Kamath (2018), o Walmart em parceria com a IBM desenvolveu dois projetos pilotos envolvendo a aplicação da tecnologia *blockchain*, por meio do *framework* Hyperledger Fabric, na rastreabilidade na cadeia produtiva de alimentos, um voltado para rastreabilidade da produção de carne suína na China e outro para a rastreabilidade da produção de mangas na América Central e na América do Sul que são importadas para os Estados Unidos. Kamath (2018) expõe uma série de episódios onde, após a ocorrência de intoxicação alimentar com um consumidor, identifica-se que determinado produto está contaminado, sendo necessária a suspensão de sua venda, bem como o rastreamento do lote contaminado desde a origem para prevenir a venda do produto contaminado por outras filiais ou outras redes de supermercado. Com o sistema baseado em *blockchain*, o Walmart tornou-se capaz de rastrear tais produtos em poucos segundos, o que antes levavam-se vários dias, trazendo eficiência, agilidade, precisão, economia e maior segurança ao consumidor.

Du et al. (2020) propõem uma arquitetura de aplicação baseada em *blockchain* para rastreabilidade na cadeia de produção de produtos de origem animal. Considerando os altos custos de se gravar dados em *blockchains* públicas, mas não perdendo de vista também a sua confiabilidade, os autores sugerem um modelo onde as informações fornecidas pelos

entes envolvidos, como produtores, frigoríficos, distribuidores e varejistas, a respeito dos produtos, sejam registradas por meio de transações organizadas em blocos dentro de uma *blockchain* privada, e que as *hashes* destes blocos, por sua vez, sejam registradas em uma *blockchain* pública, fortalecendo o aspecto da imutabilidade dos dados, mesmo que em determinado momento o número de nós na rede fique muito baixo ou que ocorra algum problema com a rede em execução. Uma aplicação de exemplo é desenvolvida utilizando o Hyperledger Fabric e testes de escalabilidade são realizados utilizando-se três diferentes algoritmos de consenso.

A Tabela 2 sintetiza os principais dados dos trabalhos relacionados, apresentando o domínio de cada aplicação desenvolvida, a plataforma de *blockchain* empregada, a existência ou não de armazenamento externo, o tipo de rede adotado e, de forma particularmente relevante, qual aspecto a tecnologia *blockchain* busca aprimorar no contexto abordado.

Tabela 2 – Síntese comparativa dos trabalhos relacionados

Autor	**Aplicação**	**Plataforma**	**Armazenamento externo**	**Rede**	**Finalidade da solução**
Ghani et al. (2022)	Certificados acadêmicos	Hyperledger Fabric	Não	Privada	Validação e emissão
Dimitriou (2020)	Sistema de votação	Hyperledger Fabric	Não	Privada	Integridade eleitoral
Vayadande et al. (2022)	Registro de imóveis	Ethereum	Banco de dados tradicional	Pública	Registro documental
Shinde et al. (2019)	Registro de imóveis	Ethereum	IPFS	Pública	Registro documental
Menezes, Araújo e Nishijima (2023)	Serviços cartorários	Ethereum	IPFS	Pública	Digitalização de serviços
Elmay et al. (2022)	Rastreamento logístico	Ethereum	IPFS	Pública	Contêineres marítimos
Kamath (2018)	Logística de produtos alimentícios	Hyperledger Fabric	Não	Privada	Rastreabilidade de cadeia de produção
Du et al. (2020)	Logística de produtos alimentícios	Hyperledger Fabric	Não	Híbrida	Rastreabilidade de cadeia de produção

Fonte: O autor.

Considerando o desenvolvimento de uma solução de *blockchain* para a gestão e certificação do programa Carne Carbono Neutro, foram buscados trabalhos que propuseram soluções baseadas em *blockchain* para gestão de informações e emissão de certificados ou registros em ambientes multiagentes. Desta análise de trabalhos relacionados, destacam-se os seguintes pontos:

- o emprego de *blockchain* sempre tem enfoque na imutabilidade dos dados e na

consequente transparência e confiabilidade que pode ser alcançada desde que tal tecnologia seja aplicada corretamente;

- a significativa versatilidade dos contratos inteligentes possibilita a execução de tarefas mais simples como permitir ou não o acesso a uma informação, bem como tarefas mais complexas como consultar APIs externas e executar transações quando determinado evento ocorrer;
- as possibilidades de se integrar diferentes tecnologias a depender da situação e dos requisitos, onde foram vistas soluções que integram *blockchain* pública com *blockchain* privada, *blockchain* com banco de dados centralizado, *blockchain* com IPFS, havendo sempre vantagens e desvantagens que precisam ser ponderadas;
- a divisão dos autores entre adotar a Hyperledger Fabric e a rede Ethereum, onde a Ethereum foi adotada mais frequentemente em cenários onde foi considerado viável o pagamento de taxas de transação e onde buscou-se fornecer recursos que seriam acessados pelo público geral, onde já há um ecossistema de aplicações disponíveis para usuários finais. A Hyperledger Fabric por sua vez demanda da disponibilidade de infraestrutura própria ou alugada para funcionar, já que trata-se um *framework* para construção de *blockchains* privadas ou em consórcio, podendo custar mais caro num primeiro momento, mas com tendência de custo mais acessível se comparado a redes públicas a medida que o volume de transações na rede cresce.

Com base nessa análise, a **Hyperledger Fabric** foi adotada para o desenvolvimento da estrutura *blockchain* neste trabalho. Essa escolha se justifica por sua flexibilidade em relação aos algoritmos de consenso, além de suas configurações nativas de controle de permissões e privacidade, que atendem aos requisitos de transparência, confiabilidade e eficiência necessários para a certificação do programa Carne Carbono Neutro.

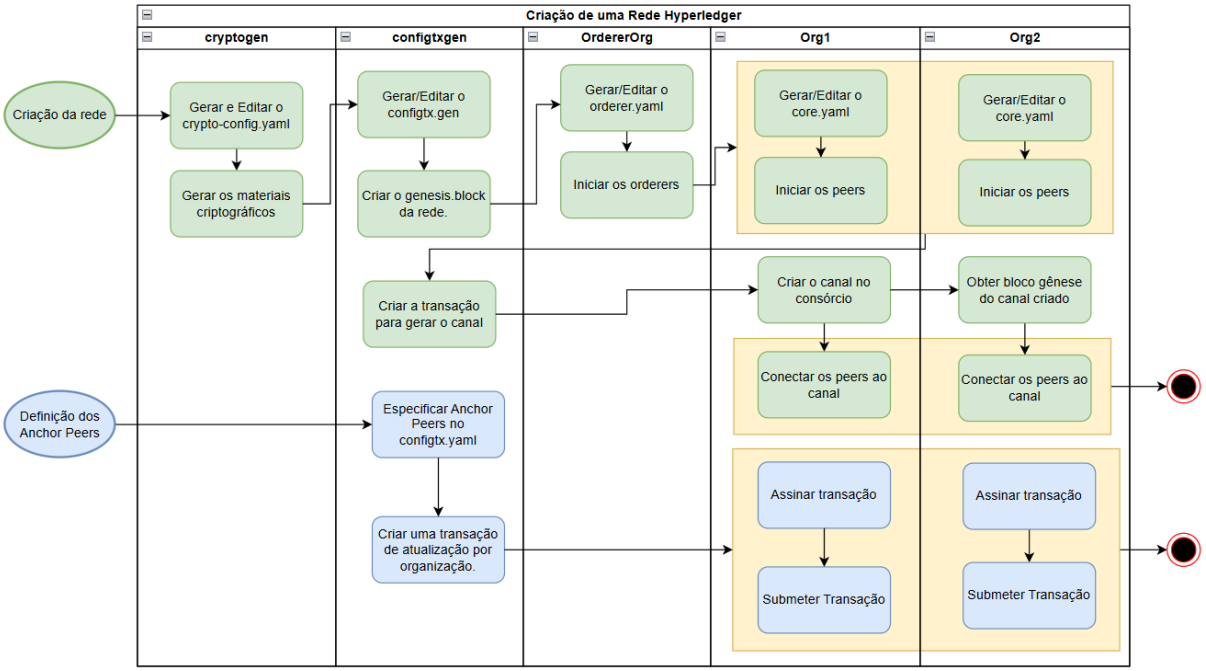
Observa-se que, embora existam diversas propostas de aplicação de *blockchain* para validação e rastreabilidade, ainda há espaço para estudos que detalhem a construção da infraestrutura e sua avaliação de desempenho em um contexto específico, como o da certificação PBC. Em muitos trabalhos, os testes são apresentados, mas a estrutura da rede e suas configurações não são descritas com maior profundidade. Este trabalho contribui para essa lacuna ao realizar a avaliação de desempenho em uma rede *blockchain* especificada de forma detalhada e estruturada de modo a permitir sua replicação.

No próximo capítulo, será apresentada a metodologia adotada, incluindo a arquitetura projetada, o processo de configuração da rede, a construção e o *deploy* do contrato inteligente, a implantação da infraestrutura na AWS e a realização dos testes de desempenho por meio do Hyperledger Caliper.

4 METODOLOGIA

Para a confiabilidade e transparência almejada na Plataforma Pecuária de Baixo Carbono (PBC), a criação de uma aplicação **blockchain** mantida em consórcio é uma alternativa promissora. Propõe-se, então, uma infraestrutura de *blockchain* para a gestão das marcas da PBC. Neste trabalho, optou-se por utilizar a **Hyperledger Fabric** em um modelo baseado em contêineres Docker, de forma a padronizar o ambiente de execução dos serviços envolvidos. O processo de criação de uma rede Hyperledger Fabric inclui várias etapas, onde inúmeras decisões de *design* precisam ser tomadas, essas decisões impactam tanto em aspectos funcionais como em aspectos não funcionais da rede.

Figura 6 – Criação de um consórcio Hyperledger Fabric



Fonte: O autor.

A Figura 6 ilustra essas etapas separando o processo em dois momentos:

1. Criação da rede em si: parte da especificação das organizações participantes e vai até a conexão dos *peers* destas organizações com a rede;
2. Definição de *peers* âncoras: uma vez conectada à rede, cada organização deve submeter uma transação que define quais de seus *peers* estarão habilitados a trocar informações com *peers* de outras organizações.

Neste Capítulo serão discutidas as tecnologias escolhidas, o processo de configuração e a metodologia de avaliação da rede *blockchain* proposta. O subtópico 4.1 irá tratar da arquitetura projetada para este consórcio. O subtópico 4.2 tratará da criação da rede Hyperledger Fabric, contendo as organizações participantes, a configuração do sistema de consenso e a criação de canais na rede, conforme ilustrado na Figura 6. O subtópico 4.3 apresentará o processo de criação de um contrato inteligente e sua instalação na rede. O

subtópico 4.4 tratará do processo de implantação e execução desta rede *blockchain* em um *cluster* de máquinas virtuais da AWS (Amazon Web Services). Por fim, no subtópico 4.5 será trabalhada a configuração e execução da ferramenta de *benchmark* Hyperledger Caliper.

4.1 ARQUITETURA DO CONSÓRCIO

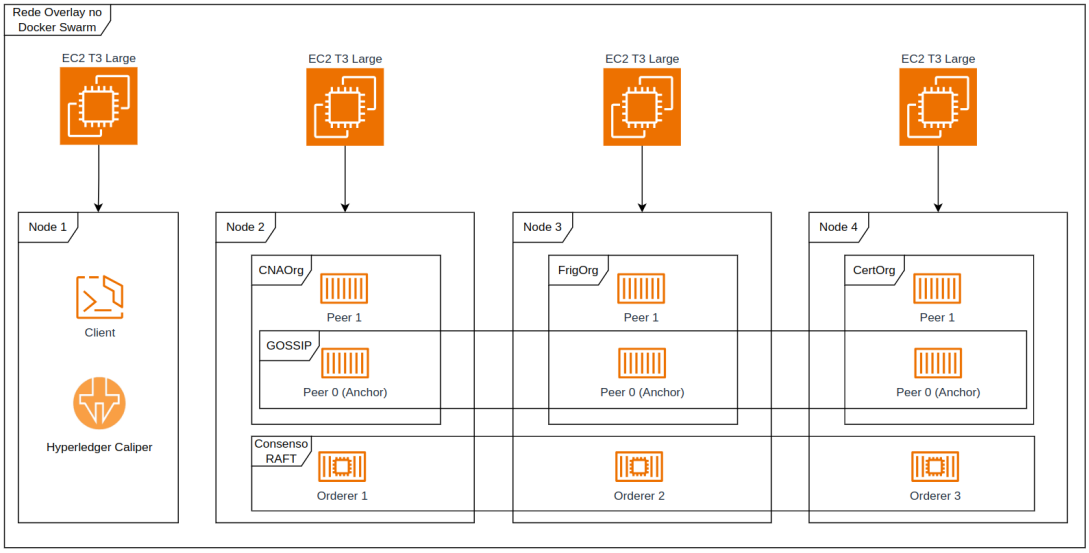
Conforme apontado no Capítulo 2, há diversas formas de configurar, hospedar e executar uma rede Fabric, nesta pesquisa optou-se pela criação de uma rede com nós remotos gerenciados por uma ferramenta de orquestração, no caso, o Docker Swarm[1]. Considerou-se essa opção como a mais equilibrada no sentido de que ela possibilita simular uma rede *blockchain* com serviços sendo executados remotamente entre máquinas diferentes, mas que simplifica o processo de configuração e operação da rede ao passo que possibilita que todo o consórcio seja coordenado a partir de um único nó, denominado como líder.

Embora esta opção simplifique a operação para fins de pesquisa, ela não seria viável em um consórcio real, pois toda a rede estaria sob o controle de uma única entidade. Sendo assim, para uma rede Fabric de um consórcio real, é possível a organização dos serviços em *clusters* orquestrados, mas com cada organização distribuindo seus *peers* e *orderers* em seu próprio *cluster*, sob seu exclusivo domínio.

Conforme ilustrado na Figura 7, foi projetado um consórcio contendo três organizações denominadas CNAOrg, representando de forma ilustrativa a CNA (Confederação da Agricultura e Pecuária do Brasil), FrigOrg, interpretando o papel de um frigorífico, e CertOrg, interpretando o papel de uma empresa certificadora credenciada. Destaca-se que é possível criar consórcios com qualquer número de organizações, optou-se por três organizações a fim de se ilustrar um consórcio inicial para a gestão da CCN.

[1] https://docs.docker.com/engine/swarm/

Figura 7 – Arquitetura do consórcio no modo de ordenação Raft com 3 orderers



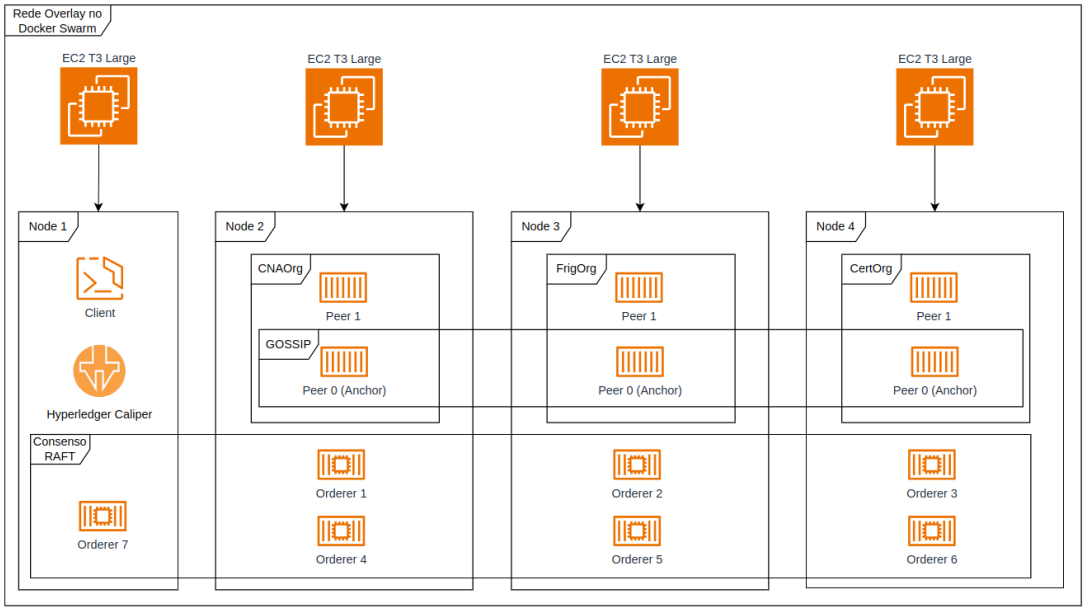
Fonte: O autor.

Para cada organização, foram especificados dois *peers*: um *peer* âncora, capaz de endossar transações e trocar informações com *peers* âncoras de outras organizações, e um *peer* comum capaz de validar e endossar transações, mas que se comunica apenas com *peers* de sua própria organização. Cada organização foi alocada em uma máquina virtual T3 Large, que conta com 8 GB de memória RAM e 2 vCPUs, do serviço EC2 (Amazon Elastic Compute Cloud) da AWS.

Com relação ao serviço de ordenação foram empregadas, para fins de comparação de desempenho, três variantes:

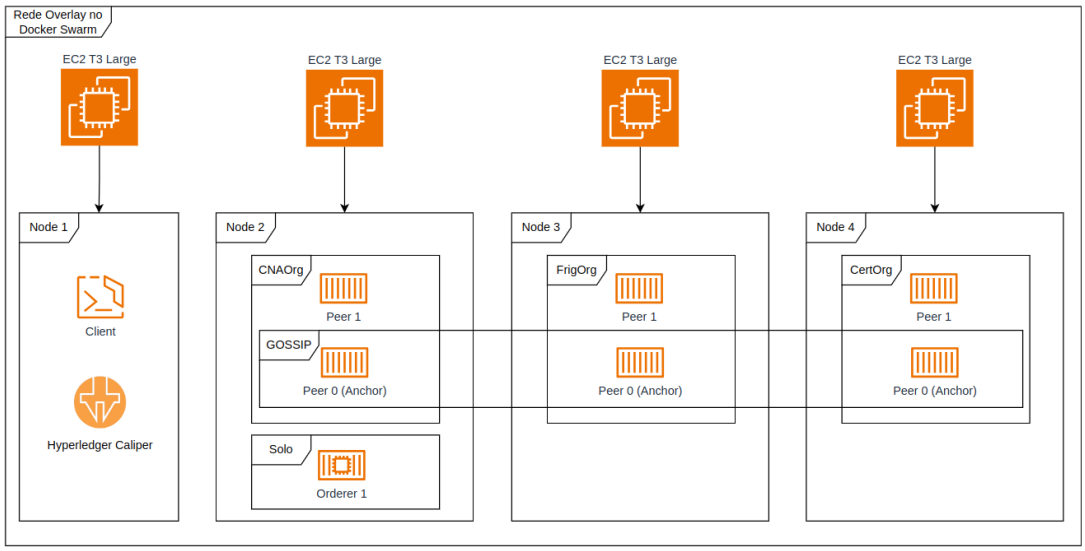
- **3 *orderers* em modo RAFT:** A arquitetura mínima suportada pelo algoritmo de consenso RAFT, com os *orderers* distribuídos nas máquinas virtuais 2, 3 e 4., conforme ilustrado na Figura 7. Este é o quantitativo mínimo de *orderers* suportado pelo algoritmo de consenso RAFT;
- **7 *orderers* em modo RAFT:** Nesta arquitetura distribuíram-se sete *orderers* em modo de consenso RAFT pelas máquinas virtuais 1, 2, 3 e 4, conforme ilustrado na Figura 8. A intenção inicial era empregar seis *orderers*, ou seja, o dobro da variante com três *orderers*, mas como o algoritmo de consenso RAFT demanda de um quórum para a eleição de um líder, é preferível o emprego de quantitativos ímpares de *orderers*. Desta forma, optou-se por empregar sete *orderers*, e não cinco, a fim de se obter uma diferença significativa entre as variantes da arquitetura proposta. Neste caso especificamente, para evitar a sobrecarga excessiva de alguma máquina virtual entre as máquinas 2, 3 e 4, optou-se por alocar um dos *orderers* na máquina virtual 1.
- **1 *orderer* em modo SOLO:** Nesta arquitetura foi alocado apenas um *orderer* no consórcio, que foi hospedado na máquina virtual 2, conforme Figura 9.

Figura 8 – Arquitetura do consórcio no modo de ordenação Raft com 7 orderers



Fonte: O autor.

Figura 9 – Arquitetura do consórcio no modo de ordenação Solo



Fonte: O autor.

Por fim, na máquina virtual 1 foram alocados o serviço *client*, que atua como um ponto de acesso à rede Fabric, mantendo seu isolamento, e o serviço da Hyperledger Caliper, responsável por executar o *benchmark* da rede.

Partindo do processo de especificação desta rede até o seu *deploy* e execução num *cluster* de máquinas virtuais na AWS, há várias etapas essenciais e inúmeras opções diferentes de configuração a serem observadas, que serão tratadas nos próximos subtópicos.

4.2 CRIAÇÃO DA REDE HYPERLEDGER FABRIC

As etapas de especificação da rede Hyperledger Fabric para o consórcio, focadas no ambiente de desenvolvimento, são detalhadas a seguir. Quaisquer adaptações necessárias para a implantação da rede no *cluster* ilustrado nas Figuras 7, 8 e 9 serão apresentadas no Subtópico 4.4.

Mesmo optando-se pelo uso de contêineres Docker, os binários pré-compilados da Fabric são necessários no ambiente de desenvolvimento para as etapas iniciais de criação da rede. Para o *download* destes binários e adição dos mesmos ao `PATH` do ambiente de desenvolvimento foi escrito e utilizado um *script* denominado `downloadFabric.sh`[2].

4.2.1 Materiais Criptográficos

Conforme apontado no Capítulo 2, num consórcio real baseado na Hyperledger Fabric, é recomendado o emprego da Fabric CA para a gestão das identidades criptográficas dos serviços e dos usuários da rede, porém, considerando que o objetivo nesse trabalho é construir um consórcio estático para prova de conceito, e que a Fabric CA é voltada para o registro e gestão dinâmica de identidades criptográficas, optou-se por empregar a ferramenta cryptogen para a geração de tais identidades.

A ferramenta cryptogen permite gerar o material criptográfico de todo o consórcio em dois passos: a edição do arquivo `crypto-config.yaml` (Figura 10), onde são definidas as organizações, seus *orderers*, seus *peers* e seus usuários, e a execução de um comando de terminal.

Figura 10 – Arquivo crypto-config.yaml para geração dos materiais criptográficos

```
1  ##### Independent Orderer Org
2  OrdererOrgs:
3    - Name: OrdererOrg
4      Domain: ordererorg.com
5      EnableNodeOUs: true
6      Specs:
7        - Hostname: orderer1
8        - Hostname: orderer2
9        - Hostname: orderer3
10       - Hostname: orderer4
11       - Hostname: orderer5
12       - Hostname: orderer6
13       - Hostname: orderer7
14
15 #### Peers Orgs
16 PeerOrgs:
17   # Peer configuration for CNA
18   - Name: CNAOrg
19     Domain: cna.com
20     EnableNodeOUs: true
21     Template:
22       Count: 2
23     Users:
24       Count: 2
25
26   # Peer configuration for Frig
27 >  - Name: FrigOrg ···
34
35   # Peer configuration for Cert
36 >  - Name: CertOrg ···
```

Fonte: O autor.

A Figura 10 demonstra como é a especificação do consórcio para a geração dos materiais criptográficos pela ferramenta cryptogen, onde pode-se observar a divisão entre

[2] Disponível em https://github.com/lucas-ads/Plataforma-PBC.HyperledgerFabricSwarm/tree/main/fabric

organizações de *orderers* (OrdererOrgs) e organizações de *peers* (PeerOrgs). Apesar dessa divisão lógica na arquitetura da Hyperledger Fabric, uma mesma organização pode conter *orderers* e *peers*, mas neste caso ela terá que estar especificada nas duas categorias de organizações do `crypto-config.yaml` com o mesmo nome. Na seção representada pela Figura 10, linha 3, define-se uma organização de ordenação, denominada OrdererOrg, com o domínio `ordererorg.com`. Na linha 6, por meio do atributo `Specs`, especificaram-se sete nós de ordenação, identificados como `"orderer1"`, `"orderer2"`, `...`, `"orderer7"`, que serão associados automaticamente, aos subdomínios `"orderer1.ordererorg.com"`, `"orderer2.ordererorg. com"`, `...`, `"orderer7.ordererorg.com"`.

Ainda na Figura 10, na linha 18, é definida a organização de *peers* CNAOrg com o domínio `"cna.com"`. Na linha 21, por meio do atributo `"Template"`, especificou-se a necessidade de dois *peers* para essa organização, que serão gerados com a nomenclatura padrão `"peer0"` e `"peer1"`, sob os domínios `"peer0.cna.com"` e `"peer1.cna.com"`. Por fim, na linha 23, especificou-se a geração de dois usuários do tipo padrão para a organização. A cryptogen já cria, por padrão, um usuário administrador por organização. Essa estrutura se repete nas linhas 27 e 36 para as organizações FrigOrg e CertOrg.

Com o `crypto-config.yaml` editado, basta executar o comando `"cryptogen generate -config=./crypto-config.yaml"` para que a pasta `crypto-config`[3] seja gerada com todo o material criptográfico. É importante garantir que na pasta `cacerts` de cada organização e de cada *peer* estejam presentes os certificados digitais das demais organizações e dos demais *peers* para que a comunicação seja possível. Para automatizar a execução do cryptogen e as cópias dos certificados entre organizações e *peers*, foi escrito um *script* denominado `generateCryptoConfig.sh`[4].

4.2.2 Criação do Bloco Gênesis

Com os materiais criptográficos criados, o próximo passo diz respeito à criação do bloco gênesis do consórcio. Este bloco conterá informações sobre quais organizações fazem parte do consórcio, quais são as políticas de gestão e qual arranjo será adotado para o consenso. A sua criação é realizada com o binário `configtxgen` com base no conteúdo do arquivo de configuração `configtx.yaml`.

No Capítulo 2 foi apresentado o conceito de canal que a Hyperledger Fabric emprega para possibilitar comunicação particular entre subgrupos de organizações dentro de um consórcio. Nessa pesquisa, conforme será detalhado nos próximos subtópicos, optou-se por criar apenas um canal, denominado `"mainchannel"`, englobando as três organizações, onde foi implantado um *chaincode*, com base no qual realizaram-se os *benchmarks* da rede.

[3] A estrutura de arquivos desta pasta pode ser vista em https://github.com/lucas-ads/Plataforma-PBC.HyperledgerFabricSwarm/tree/main/crypto/crypto-config/

[4] Disponível em https://github.com/lucas-ads/Plataforma-PBC.HyperledgerFabricSwarm/tree/main/crypto

Do arquivo `configtx.yaml` fornecido pela Fabric, visando a criação do consórcio e do canal mainchannel, foi necessária a edição das seções Organizations, Orderer e Profiles, enquanto que para as seções Capabilities, Application e Channel foram empregados os valores pré-estabelecidos.

Em Organizations, conforme ilustrado na Figura 11, foram especificadas as organizações que constituem o consórcio. Tanto para a organização de ordenação OrdererOrg, quanto para as organizações de *peers* CNAOrg, FrigOrg e CertOrg, configuraram-se nomes, IDs, caminhos para as respectivas pastas MSP, que contêm as identidades criptográficas das organizações. No que diz respeito às políticas de permissão internas de cada organização, foram mantidos os valores pré-estabelecidos no modelo. Além disso, nas organizações de *peers*, foram especificados os endereços IP e portas dos *peers* âncoras.

Figura 11 – Organizações no configtx.yaml

```
1   Organizations:
2       - &OrdererOrg
3           Name: OrdererOrg
4           ID: OrdererOrgMSP
5           MSPDir: ../crypto/crypto-config/ordererOrganizations/ordererorg.com/msp
6   >       Policies: &OrdererOrgPolicies…
19
20      - &CNAOrg
21          Name: CNAOrg
22          ID: CNAOrgMSP
23          MSPDir: ../crypto/crypto-config/peerOrganizations/cna.com/msp
24  >       Policies: &CNAOrgPolicies…
43
44          AnchorPeers:
45              - Host: peer0.cna.com
46                Port: 7051
47
48      - &FrigOrg
49          # Similar à configuração do CNAOrg
50      - &CertOrg
51          # Similar à configuração do CNAOrg
```

Fonte: O autor.

Em Orderer, conforme Figura 12, definiu-se o tipo de consenso (OrdererType) para etcdraft, indicando que o consórcio empregará, por padrão, o algoritmo de consenso RAFT e foram especificados os endereços dos *orderers*. Por meio do atributo `EtcdRaft`, os *orderers* que estarão envolvidos no processo de consenso foram discriminados junto de seus certificados TLS. Por fim, são apresentadas as configurações relacionadas ao intervalo de tempo e ao tamanho dos blocos a serem gerados.

O atributo `BatchTimeout` especifica o tempo máximo que o serviço de ordenação aguarda, após receber uma transação endossada, para coletar mais transações e formar um novo bloco. Por exemplo, se definido como “2s”, o serviço espera até 2 segundos para incluir o maior número possível de transações no próximo bloco, otimizando o desempenho da

rede. O `BatchSize`, por sua vez, define tamanhos máximos e preferenciais para a emissão dos blocos.

Figura 12 – Seção Orderer no configtx.yaml

```
Orderer: &OrdererDefaults
    OrdererType: etcdraft

    Addresses:
        - orderer1.ordererorg.com:7050
        - orderer2.ordererorg.com:7050
        - orderer3.ordererorg.com:7050

    EtcdRaft:
        Consenters:
            - Host: orderer1.ordererorg.com
              Port: 7050
              ClientTLSCert: ../crypto/crypto-config/ordererOrganizations/ordererorg.com/orderers/orderer1.ordererorg.com/tls/server.crt
              ServerTLSCert: ../crypto/crypto-config/ordererOrganizations/ordererorg.com/orderers/orderer1.ordererorg.com/tls/server.crt
            - Host: orderer2.ordererorg.com
              Port: 7050
              ClientTLSCert: ../crypto/crypto-config/ordererOrganizations/ordererorg.com/orderers/orderer2.ordererorg.com/tls/server.crt
              ServerTLSCert: ../crypto/crypto-config/ordererOrganizations/ordererorg.com/orderers/orderer2.ordererorg.com/tls/server.crt
            - Host: orderer3.ordererorg.com
              Port: 7050
              ClientTLSCert: ../crypto/crypto-config/ordererOrganizations/ordererorg.com/orderers/orderer3.ordererorg.com/tls/server.crt
              ServerTLSCert: ../crypto/crypto-config/ordererOrganizations/ordererorg.com/orderers/orderer3.ordererorg.com/tls/server.crt

    BatchTimeout: 1s

    BatchSize:
        MaxMessageCount: 500
        AbsoluteMaxBytes: 10 MB
        PreferredMaxBytes: 1 MB
```

Fonte: O autor.

A emissão de cada bloco demanda da ordenação das transações pelo *orderer* eleito como líder e da posterior propagação do bloco aos *peers* e aos demais *orderers*. No modelo fornecido, o `BatchTimeout` vem definido em 2 segundos e o `BatchSize.PreferredMaxBytes` em 2 Megabytes. Em ambos os atributos, valores mais altos tendem a gerar uma maior latência na efetivação das transações, não por consumo de recursos, mas por uma escolha de *design*. Por outro lado, valores menores tendem a reduzir a latência em troca de aumentar a sobrecarga da rede. Desta forma, para fins de *benchmark* em ambientes controlados, valores menores para `BatchTimeout` e `BatchSize` são interessantes, como será melhor discutido no Capítulo 5. Neste trabalho, foram definidos valores de 1 segundo para o `BatchTimeout`, de 1 MB para para o `BatchSize.PreferredMaxBytes` e, para o `BatchSize.AbsoluteMaxBytes`, utilizou-se o valor padrão de 10 MB.

Em Profiles, foram definidos três perfis para a geração de blocos gênesis, dois voltados para a criação de consórcios, empregando diferentes modos de ordenação, (Figura 13) e um voltado para a criação de um canal de aplicação (Figura 14).

Figura 13 – Perfis para criação de consórcios no configtx.yaml

```yaml
# Profile para criação de consórcio
# com orderer solo (sem consenso)
PBCConsortiumSolo:
    <<: *ChannelDefaults
    Orderer:
        <<: *OrdererDefaults
        OrdererType: solo
        Organizations:
            - *OrdererOrg
    Consortiums:
        PBCConsortium:
            Organizations:
                - *CNAOrg
                - *FrigOrg
                - *CertOrg
```

```yaml
# Profile para criação de consórcio
# com orderer RAFT (com consenso)
PBCConsortiumRaft:
    <<: *ChannelDefaults
    Orderer:
        <<: *OrdererDefaults
        OrdererType: etcdraft
        Organizations:
            - <<: *OrdererOrg
    Consortiums:
        PBCConsortium:
            Organizations:
                - *CNAOrg
                - *FrigOrg
                - *CertOrg
```

Fonte: O autor.

O perfil **PBCConsortiumSolo** definiu um conjunto de configurações para a criação do bloco gênesis do consórcio usando o modo de ordenação Solo. Referenciando `ChannelDefaults` e `OrdererDefaults`, ele incorpora configurações definidas em seções anteriores do `configtx.yaml` e, por meio dos atributos `OrdererType` e `Organizations` dentro de *Orderer*, sobrescreve o modo de ordenação para solo e define quais organizações participarão do serviço de ordenação. No atributo `Consortiums` é especificada a criação de um consórcio denominado PBCConsortium e as organizações de *peers* que participarão do mesmo.

O perfil **PBCConsortiumRaft** definiu um conjunto de configurações para a criação do bloco gênesis do consórcio usando o modo de ordenação RAFT. Neste perfil, foram empregadas as mesmas configurações utilizadas no PBCConsortiumSolo, porém com o atributo *OrdererType* sendo definido como "etcdraft".

Figura 14 – Perfil para criação de canal no configtx.yaml

```yaml
# Profile para criação de um canal
MainChannel:
    <<: *ChannelDefaults
    Consortium: PBCConsortium
    Application:
        <<: *ApplicationDefaults
        Organizations:
            - <<: *CNAOrg
            - <<: *FrigOrg
            - <<: *CertOrg
```

Fonte: O autor.

O perfil **MainChannel**, por sua vez, define um conjunto de configurações para a criação do bloco gênesis para inicialização de um canal dentro do consórcio PBCConsortium. Neste perfil, são incorporadas novamente as configurações da seção `Channel` especificadas no `configtx.yaml`. Na subseção `Application`, são incorporadas as configurações definidas na seção Application do arquivo, bem como são especificadas as organizações do

consórcio que participarão do canal. Esse perfil é empregado independentemente do modo de ordenação utilizado.

A arquitetura adotada como base nessa pesquisa e que será explanada nos próximos subtópicos é a baseada no protocolo de consenso RAFT. O perfil com o modo Solo será empregado em *benchmarks* posteriormente para fins de comparação com a arquitetura base.

Com o `configtx.yaml` devidamente configurado, é possível gerar o bloco gênesis do consórcio por meio do binário `configtxgen`, esse processo foi incorporado no *script* `init.sh`[5]. Com o bloco gênesis criado, torna-se possível a instanciação da estrutura mínima do consórcio, que tem por base o serviço de ordenação, como será discutido no próximo subtópico.

4.2.3 Configuração do Serviço de Ordenação

Conforme mencionado anteriormente, para essa pesquisa, optou-se por empregar o Docker Swarm para a criação de uma rede baseada em contêineres distribuídos por máquinas virtuais num *cluster* AWS. Desta forma, criou-se um arquivo denominado `docker-compose.yaml` e nele foram especificados os *orderers*, cada um como um serviço aos moldes do apresentado na Figura 15.

Figura 15 – Especificação de um Orderer no docker-compose.yaml

```
3   orderer1_ordererorg:
4     image: hyperledger/fabric-orderer:2.5.12
5     hostname: orderer1.ordererorg.com
6     environment:
7       - ORDERER_GENERAL_LISTENADDRESS=0.0.0.0
8       - ORDERER_GENERAL_LISTENPORT=7050
9       - ORDERER_GENERAL_LOCALMSPID=OrdererOrgMSP
10      - ORDERER_GENERAL_LOCALMSPDIR=/var/hyperledger/orderer/msp
11      - ORDERER_GENERAL_GENESISFILE=/var/hyperledger/orderer/genesis.block
12      - ORDERER_GENERAL_BOOTSTRAPMETHOD=file
13      - ORDERER_GENERAL_TLS_ENABLED=true
14      - ORDERER_GENERAL_TLS_CERTIFICATE=/var/hyperledger/orderer/tls/server.crt
15      - ORDERER_GENERAL_TLS_PRIVATEKEY=/var/hyperledger/orderer/tls/server.key
16      - ORDERER_GENERAL_TLS_ROOTCAS=[/var/hyperledger/orderer/tls/ca.crt]
17    volumes:
18      - /home/ec2-user/infra/crypto/crypto-config/ordererOrganizations/ordererorg.com/
        orderers/orderer1.ordererorg.com/msp:/var/hyperledger/orderer/msp
19      - /home/ec2-user/infra/crypto/crypto-config/ordererOrganizations/ordererorg.com/
        orderers/orderer1.ordererorg.com/tls:/var/hyperledger/orderer/tls
20      - /home/ec2-user/infra/artifacts/pbc.block:/var/hyperledger/orderer/genesis.block
21    networks:
22      fabric-net:
23        aliases:
24          - orderer1.ordererorg.com
```

Fonte: O autor.

A definição do hostname e do aliases como `"orderer1.ordererorg.com"`, apresentada na Figura 15, é essencial para que os serviços consigam se referenciar dentro do *cluster* e que os nomes de domínio coincidam com os definidos nos materiais criptográficos,

[5] Disponível em https://github.com/lucas-ads/Plataforma-PBC.HyperledgerFabricSwarm

de forma que a comunicação TLS possa funcionar. Na seção de variáveis de ambiente, é necessário definir valores para as seguintes variáveis:

- `ORDERER_GENERAL_LISTENADDRESS:` Indica quais IPs podem se comunicar com esse *orderer*, no caso, 0.0.0.0 indica que o mesmo aceita mensagens de qualquer origem;
- `ORDERER_GENERAL_LISTENPORT:` Porta de comunicação do *orderer*;
- `ORDERER_GENERAL_LOCALMSPID:` ID da organização a qual o *orderer* pertence, devendo coincidir com o definido no `configtx.yaml` que deu origem ao bloco gênesis;
- `ORDERER_GENERAL_LOCALMSPDIR:` Caminho para o diretório que contém a identidade criptográfica específica do *orderer*;
- `ORDERER_GENERAL_BOOTSTRAPMETHOD=file:` Indica que o *orderer* deverá carregar o bloco gênesis logo que iniciado;
- `ORDERER_GENERAL_GENESISFILE:` Caminho para o bloco gênesis do consórcio.

Além disso, há um conjunto de variáveis ligadas à chave privada e a certificados TLS. Por fim, na seção de volumes, são especificados três volumes do tipo `bind mount`, que liga uma pasta ou arquivo da máquina *host* ao sistema de armazenamento do contêiner do *orderer*. Os dois primeiros visam prover ao contêiner o material criptográfico necessário para sua execução, enquanto o último visa prover o bloco gênesis do consórcio.

Para que os *orderers* e os demais serviços que serão definidos posteriormente possam se comunicar, especificou-se ao final desse `docker-compose.yaml` uma *network* `overlay` com o apelido "fabric-net" e nome "hyperledger-fabric". Redes *overlay* permitem que contêineres que estejam rodando em diferentes máquinas via Docker Swarm consigam se comunicar como se estivessem numa mesma máquina. Com isso, é possível iniciar os *orderers* usando o comando `"docker stack deploy <nome-stack>"` e então empregar o comando `"docker service ls"` para obter a lista de contêineres em execução.

4.2.4 Configuração dos Peers

No mesmo `docker-compose.yaml` onde foram definidos os serviços dos *orderers*, foram configurados os seis *peers* previstos para esse consórcio, com dois *peers* para cada organização. Para a execução de um *peer*, há inúmeros parâmetros a serem configurados, tais parâmetros podem ser configurados de forma geral num arquivo chamado `core.yaml` e aqueles que demandarem de valores específicos em cada *peer* podem ser sobrescritos na seção de variáveis de ambiente do serviço do *peer* no `docker-compose.yaml`. Para o `core.yaml`, empregou-se o modelo padrão[6] fornecido no repositório `fabric-samples` da Hyperldeger Fabric, sem nenhuma edição.

[6] Disponível em https://github.com/hyperledger/fabric/blob/main/sampleconfig/core.yaml

Figura 16 – Especificação de um Peer no docker-compose.yaml

```yaml
peer0_cna:
  image: hyperledger/fabric-peer:2.5.12
  hostname: peer0.cna.com
  environment:
    # Habilitando logs para o prometheus:
    - FABRIC_LOGGING_SPEC=info
    - CORE_OPERATIONS_LISTENADDRESS=0.0.0.0:9446
    - CORE_METRICS_PROVIDER=prometheus

    - CORE_PEER_ID=peer0.cna.com
    - CORE_PEER_ADDRESS=peer0.cna.com:7051
    - CORE_PEER_LISTENADDRESS=0.0.0.0:7051
    - CORE_PEER_LOCALMSPID=CNAOrgMSP
    - CORE_PEER_MSPCONFIGPATH=/var/hyperledger/peer/msp
    - CORE_PEER_TLS_ENABLED=true
    - CORE_PEER_TLS_CERT_FILE=/var/hyperledger/peer/tls/server.crt
    - CORE_PEER_TLS_KEY_FILE=/var/hyperledger/peer/tls/server.key
    - CORE_PEER_TLS_ROOTCERT_FILE=/var/hyperledger/peer/tls/ca.crt
    # GOSSIP:
    - CORE_PEER_GOSSIP_EXTERNALENDPOINT=peer0.cna.com:7051
    - CORE_PEER_GOSSIP_BOOTSTRAP=peer1.cna.com:7051

    # Necessário para que o container do contrato seja criado na mesma rede do peer:
    - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=hyperledger-fabric

  volumes:
    - /home/ec2-user/infra/crypto/crypto-config/peerOrganizations/cna.com/peers/peer0.
    cna.com/msp:/var/hyperledger/peer/msp
    - /home/ec2-user/infra/crypto/crypto-config/peerOrganizations/cna.com/peers/peer0.
    cna.com/tls:/var/hyperledger/peer/tls
    # Artefatos para criação do channel e junção dos anchor peers
    - /home/ec2-user/infra/artifacts:/var/hyperledger/artifacts

    # Dá ao container atual acesso ao socket docker para que ele possa instanciar
    novos contêineres para os chaincodes.
    - /var/run/docker.sock:/var/run/docker.sock
  networks:
    fabric-net:
      aliases:
        - peer0.cna.com
```

Fonte: O autor.

A Figura 16 apresenta a configuração de um *peer* como serviço Docker. Na seção de variáveis de ambiente do serviço, além das variáveis já mencionadas no subtópico anterior e que se repetem no *peer*, há outras essenciais:

- `CORE_PEER_GOSSIP_EXTERNALENDPOINT`: Define o endereço e a porta pelos quais outros *peers* podem se conectar a este;
- `CORE_PEER_GOSSIP_BOOTSTRAP`: Define o endereço e a porta de um *peer* existente da mesma organização para que este tente se conectar assim for iniciado;
- `CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE`: No modo de funcionamento padrão da Fabric no contexto do Docker, os *chaincodes* são executados como contêineres separados dos *peers* que os gerenciam. Desta forma, esta configuração é imprescindível para que o *deploy* dos *chaincodes* seja feito na mesma rede Docker em que os *peers* estão sendo executados.

Ainda na Figura 16, pode-se observar a especificação de quatro volumes *bind mount*:

- **MSP e TLS:** Esses dois primeiros volumes são necessários para que o *peer* tenha acesso a seu conjunto de materiais criptográficos;
- **Artefatos:** O terceiro volume provê ao *peer* o acesso ao diretório de artefatos, local para onde são exportados arquivos necessários para criação da rede, criação do canal e definição dos *peers* âncoras;
- **Daemon Docker:** O quarto volume é necessário para possibilitar que o contêiner do *peer* interaja diretamente com o *daemon* do Docker no *host* onde ele está sendo executado, para que seja possível a criação e gerenciamento de contêineres para os contratos inteligentes.

Por fim, na rede *overlay* `fabric-net` mencionada no subtópico anterior, é necessário acrescentar o atributo `"attachable: true"`, possibilitando que os *peers* consigam criar novos contêineres para os *chaincodes* na mesma rede Docker em que estão sendo executados.

Neste ponto, já há um consórcio Fabric com um serviço de ordenação em funcionamento e com *peers* em execução, o próximo passo essencial consiste na criação de um canal de aplicação e na conexão dos *peers* das organizações participantes a este canal. Esta etapa demanda da execução de uma série de comandos na rede do consórcio, desta forma, visando prover uma arquitetura mais próxima do ideal, o próximo subtópico abordará a criação de um serviço cliente para intermediar o acesso e gestão da rede entre administrador e *peers*, para então avançar-se à etapa de criação de um canal de aplicação.

4.2.5 Serviço Cliente

Conforme explicado no Capítulo 2, existem várias formas de se conectar a um *peer* para executar comandos de gerenciamento, nesta pesquisa optou-se pela criação e configuração de um serviço cliente para intermediar tal gerenciamento, enviando instruções aos *peers* invés de executar comandos diretamente dentro de seus contêineres.

Na Figura 7 do Subtópico 4.1, que apresenta a distribuição dos componentes do consórcio pelas máquinas virtuais do *cluster* AWS, o *client* alocado na máquina virtual 1 foi configurado para operar com a identidade de usuário administrador da organização CNAOrg e a Figura 17 apresenta sua definição no `docker-compose.yaml`.

Figura 17 – Especificação do Client no docker-compose.yaml

```
client:
  image: hyperledger/fabric-tools:2.5
  hostname: client.com
  tty: true
  stdin_open: true
  working_dir: /infra
  environment:
    - CORE_PEER_LOCALMSPID=CNAOrgMSP
    - CORE_PEER_ADDRESS=peer0.cna.com:7051
    - CORE_PEER_MSPCONFIGPATH=/var/hyperledger/peerOrganizations/cna.com/
    users/Admin@cna.com/msp
    - CORE_PEER_TLS_ENABLED=true
    - CORE_PEER_TLS_ROOTCERT_FILE=/var/hyperledger/peerOrganizations/cna.com/
    tlsca/tlsca.cna.com-cert.pem
    - FABRIC_CFG_PATH=/infra/config
  volumes:
    - /home/ec2-user/infra/crypto/crypto-config:/var/hyperledger/
    - /home/ec2-user/infra:/infra
  command: /bin/bash
  networks:
    fabric-net:
      aliases:
        - client.com
```

Fonte: O autor.

Na Figura 17 é possível observar uma série de variáveis de ambiente onde a maioria já foi mencionada no subtópico que trata da configuração do *peer*, porém, destaca-se aqui a variável `CORE_PEER_MSPCONFIGPATH` que está assumindo a identidade MSP do administrador da organização CNAOrg invés da identidade do *peer* em si, o que torna possível a execução de operações administrativas na rede como o ingresso da organização em um canal e a instalação e atualização de contratos inteligentes.

Na seção de volumes do serviço *client*, foram especificados dois volumes *bind mounts*, o primeiro faz com que o serviço tenha acesso ao diretório que contém todas as identidades criptográficas do consórcio, enquanto o segundo faz com que o serviço tenha acesso a pasta do projeto como um todo. A pasta do projeto contém os *scripts* de automatização, os arquivos de configuração e o `docker-compose.yaml` já mencionados, bem como o código-fonte do contrato inteligente que será empregado e os arquivos necessários para os *benchmarks* com o Hyperledger Caliper.

Com isso, uma vez iniciada a *stack* com todos os serviços que foram apresentados nos últimos subtópicos, o gerenciamento dos *peers* do consórcio e a instalação do contrato inteligente serão realizados por intermédio do *client*.

4.2.6 Criação de um Canal de Aplicação

O consórcio funciona como uma fundação que define quais organizações participam do serviço de ordenação e quais participam com a conexão de *peers*, podendo criar e gerenciar canais, bem como instalar e/ou invocar contratos inteligentes. Porém, a instalação

de contratos e a posterior execução de transações ocorrem no contexto de canais de aplicação. A criação de um canal de aplicação entre organizações em um consórcio envolve três etapas: a geração da transação de criação do canal, a criação do canal em si e, por fim, o ingresso dos *peers* no canal.

A geração da transação de criação do canal, bem como a geração do bloco gênesis do consórcio, apresentado anteriormente, é realizada por meio do binário configtxgen com base no arquivo `configtx.yaml`, porém, nesta etapa utilizou-se o perfil de configuração MainChannel apresentado na Figura 14, resultando num arquivo `.TX` que será empregado na etapa seguinte. A execução desta etapa foi automatizada por meio do *script* `generateArtifacts.sh`[7].

O próximo passo consiste na submissão da transação de criação do canal à rede por meio de um *peer*. Empregou-se para isso o *script* `peer-create-channel.sh`[8], que por meio do binário *peer* presente na pasta `fabric` do projeto, a qual o *client* tem acesso, encaminha a transação gerada na etapa anterior para o serviço de ordenação, que por sua vez, registra a criação do canal no consórcio e retorna ao *client* o bloco gênesis do novo canal.

Com o canal criado no consórcio, a última etapa diz respeito ao ingresso dos *peers* das três organizações ao respectivo canal. Para que um *peer* ingresse num canal é necessária a execução, usando a identidade criptográfica do *peer*, do comando `"peer channel join"` passando como parâmetro o endereço de um *orderer* do consórcio e o bloco gênesis criado na etapa anterior. Considerando que foram empregados dois *peers* em cada uma das três organizações, totalizando seis *peers*, o processo de ingressar os *peers* no canal foi automatizado pelo *script* `peer-join-channel.sh`[9], que foi executado a partir do serviço *client*.

4.2.7 Definição dos Anchor Peers

Com os *peers* conectados ao canal de aplicação, a rede do consórcio já se torna funcional, porém, é importante que cada organização defina um conjunto de *anchor peers* para que *peers* de diferentes organizações possam se descobrir e trocar informações. A definição de *anchor peers* consiste em duas etapas, que cada organização deverá executar: a criação de uma transação de atualização de *anchor peers* e a submissão de tal transação à rede.

Mais uma vez, a criação desta transação foi realizada por meio do configtxgen, com base no arquivo `configtx.yaml`. Na Figura 11, apresentada previamente, foi possível

[7] Disponível em https://github.com/lucas-ads/Plataforma-PBC.HyperledgerFabricSwarm/tree/main/artifacts

[8] Disponível em https://github.com/lucas-ads/Plataforma-PBC.HyperledgerFabricSwarm/tree/main/scripts

[9] Disponível em https://github.com/lucas-ads/Plataforma-PBC.HyperledgerFabricSwarm/tree/main/scripts

observar um exemplo de definição de *anchor peer* para uma organização. Esse processo de geração da transação se repetiu para as três organizações e foi automatizado no *script* `generateArtifacts.sh`, que gera os três arquivos de transação `.TX` quando executado.

O segundo passo na definição dos *anchor peers* foi a submissão à rede, por cada organização, da transação de atualização por meio do comando `"peer channel signconfigtx"`, que empregou como parâmetro o arquivo `.TX` gerado no passo anterior. Esse processo foi automatizado no *script* `peer-join-channel.sh`[10] e executado a partir do serviço *client*.

Com exceção da geração das identidades criptográficas, todas essas etapas foram automatizadas por meio do *script* `init.sh`, citado anteriormente. Tal *script*, quando executado, chamando os demais *scripts* mencionados, gera os artefatos necessários, inicializa a pilha de serviços no Docker com os *peers*, *orderers* e com o *client*, e, por fim, envia comandos ao *client* para que o canal seja criado, os *peers* ingressem no canal e os *anchor peers* sejam definidos. Em contrapartida, para parar e remover toda a rede, com exceção das identidades criptográficas e do bloco gênesis do consórcio, foi criado e empregado o *script* `clean.sh`, disponível junto do `init.sh`.

4.3 CRIAÇÃO E IMPLANTAÇÃO DO CONTRATO INTELIGENTE

O objetivo desta infraestrutura de *blockchain* é servir de base para a construção de aplicações descentralizadas para a PBC. Uma aplicação descentralizada, também chamada de DApp, consiste no conjunto formado por contratos inteligentes rodando em plataformas de *blockchain*, que armazenam e gerenciam dados, e por aplicações cliente, que por sua vez, acessam e enviam requisições para estes contratos. Para esta pesquisa, focada na especificação e validação de uma infraestrutura de *blockchain*, foi criado um contrato inteligente simplificado, que não tem o objetivo de representar um contrato funcional para a PBC, mas que foi essencial para a validação da infraestrutura, uma vez que os *benchmarks* foram realizados por meio do envio e monitoramento de requisições a tal contrato.

O contrato inteligente foi desenvolvido utilizando a linguagem de programação Javascript empregando-se a tecnologia NodeJS. O mesmo consiste em uma classe denominada CertificacaoCCN[11] que contém três métodos:

- **InitLedger:** Registra dois certificados CCN hipotéticos com dados fixos na *ledger* do contrato. Serve para testar o funcionamento do contrato após sua instalação nos *peers*;
- **RegistraCertificacao:** Recebe um conjunto de dados por parâmetro e, com base neles, registra um novo certificado CCN na *ledger* do contrato;

[10] https://github.com/lucas-ads/Plataforma-PBC.HyperledgerFabricSwarm/tree/main/scripts
[11] https://github.com/lucas-ads/Plataforma-PBC.HyperledgerFabricSwarm/tree/main/contratos/carneCarbonoNeutro/ccn

- **GetAllCertificacoes:** Retorna todos os certificados registrados na *ledger* do contrato.

A Tabela 3 apresenta a estrutura do certificado gerido por este contrato.

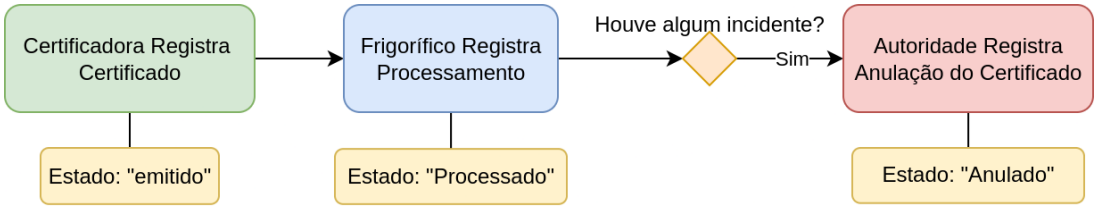
Tabela 3 – Dados do certificado

Atributo	**Descrição**
`id`	ID único do certificado na *ledger* do contrato.
`propriedade`	Nome da fazenda.
`emissao`	Enumera os certificados emitidos pela fazenda.
`certificadora_public_key`	Chave pública da certificadora no consórcio.
`certificadora_nome`	Nome da certificadora.
`proprietario_public_key`	Chave pública do proprietário no consórcio.
`proprietario_nome`	Nome do proprietário da fazenda.
`frigorifico_public_key`	Chave pública do frigorífico responsável pelo processamento.
`frigorifico_nome`	Nome do frigorífico responsável pelo processamento.
`n_animais_rebanho`	Número de cabeças do rebanho certificado.
`codigos_animais`	Lista de identificadores dos animais do rebanho certificado.
`payload_certificadora`	Dados detalhados que poderão ser adicionados pela certificadora a respeito do rebanho certificado.
`payload_frigorifico`	Dados detalhados que poderão ser adicionados pelo frigorífico a respeito do rebanho abatido e dos produtos originados.
`data_emissao`	Data da emissão do certificado.
`data_processamento`	Data do abate pelo frigorífico.
`data_anulacao`	Data da anulação do certificado, se ocorrer.
`motivo_anulacao`	Motivo da anulação do certificado, se ocorrer. (ex.: contaminação identificada)
`estado_atual`	Estágio em que se encontra o certificado, conforme Figura 18. Exemplos: “emitido”, “processado” ou “anulado”.

Fonte: O autor.

A estrutura do certificado foi projetada de forma que cada certificado passe por dois ou três estágios, conforme ilustrado na Figura 18, porém, considerando que o objetivo desta pesquisa reside na organização e validação da infraestrutura de *blockchain*, foi implementada no contrato apenas a função para o primeiro estado do certificado, no caso, a emissão. Essa função já é suficiente para o registro de dados na *blockchain*, e portanto, para a execução de *benchmarks* da infraestrutura, conforme será apresentado no Subtópico 4.5. Além disso, para a criar uma DApp viável em torno deste contrato, seria necessário o registro de frigoríficos, certificadoras e proprietários na *ledger*, bem como a programação de uma lógica de permissões no contrato para restringir quais atores podem executar cada ação.

Figura 18 – Estados possíveis para um certificado



Fonte: O autor.

A instalação do contrato no canal foi realizada a partir do serviço *client*, pelo *script* `install_chaincode.sh`[12], e consistiu em quatro etapas:

- Empacotamento do contrato com o comando `"peer lifecycle chaincode package"`;
- Instalação do contrato em todos os *peers*;
- Aprovação do contrato por todas as organizações, ou seja, em um *peer* de cada a organização;
- *Commit*, ou confirmação final do contrato, por qualquer *peer* de qualquer organização. A partir desta etapa o contrato passa a vigorar de forma irrefutável no canal para todas as organizações participantes.

Para validar o funcionamento das funções do contrato desenvolvido, antes de iniciar a execução dos *benchmarks*, foram escritos três *scripts* que enviam requisições para o contrato, cada um para uma função diferente. Os mesmos estão disponíveis no repositório do projeto, junto do `install_chaincode.sh`.

4.4 IMPLANTAÇÃO DO CONSÓRCIO EM UM CLUSTER AWS

Com todos os serviços do consórcio funcionando corretamente via Docker Swarm na máquina local de desenvolvimento, iniciou-se a etapa de configuração do *cluster* de máquinas virtuais na AWS, por meio do serviço Amazon EC2, conforme ilustrado na Figura 7.

Antes de configurar e iniciar as máquinas em si, foi criado um grupo de segurança com regras de entrada que permitam a comunicação em determinadas portas para possibilitar a conexão remota via SSH com as máquinas virtuais, o uso do *software* IPerf para teste de conexão e de largura de banda entre as máquinas, bem como a conexão entre os nós líder e *workers* do Docker Swarm, conforme Figura 19.

[12] Disponível em: https://github.com/lucas-ads/Plataforma-PBC.HyperledgerFabricSwarm/tree/main/contratos/carneCarbonoNeutro

Figura 19 – Regras de entrada empregadas no grupo de segurança

Versão do IP	Tipo	Protocolo	Intervalo de portas	Origem	Descrição
IPv4	SSH	TCP	22	0.0.0.0/0	SSH
IPv4	TCP personalizado	TCP	2377	0.0.0.0/0	Docker Controle
IPv4	UDP personalizado	UDP	4789	0.0.0.0/0	Docker Overlay network
IPv4	TCP personalizado	TCP	4789	0.0.0.0/0	Docker Overlay network
IPv4	TCP personalizado	TCP	5201	0.0.0.0/0	IPerf
IPv4	UDP personalizado	UDP	5201	0.0.0.0/0	IPerf
IPv4	TCP personalizado	TCP	7946	0.0.0.0/0	Docker Comm Interna
IPv4	UDP personalizado	UDP	7946	0.0.0.0/0	Docker Comm Interna

Fonte: O autor.

Prosseguiu-se então para a criação das máquinas virtuais na Amazon EC2, onde foram criadas quatro máquinas empregando-se a AMI (Imagem de Máquina da Amazon) `Amazon Linux 2023 kernel-6.1`, arquitetura 64 bits (x86), tipo de instância `t3.large` (que conta com 8 GB de memória RAM e 2 vCPU), o grupo de segurança supramencionado e, por fim, 15 GB de armazenamento SSD gp3. Durante a criação destas máquinas, foi gerado um par de chaves RSA e a chave privada foi baixada para a máquina local de desenvolvimento. A fim de facilitar a identificação, as máquinas em execução foram nomeadas como `Node1`, `Node2`, `Node3` e `Node4`. A configuração inicial de cada máquina consistiu em realizar a conexão à máquina via SSH utilizando a chave privada gerada anteriormente, instalar e inicializar o Docker e instalar o NodeJS.

Ao executar os *peers* e *orderers* de forma distribuída, as máquinas sobre as quais tais serviços estão rodando devem dispor dos arquivos e pastas indicadas como volumes *bind mounts* nos mesmos, no caso, trata-se de volumes que dão acesso ao diretório de materiais criptográficos e ao bloco gênesis do consórcio. Sendo assim, na pasta do projeto, ainda na máquina local, foram gerados os materiais criptográficos e o bloco gênesis do consórcio. Com isso, a pasta do projeto foi comprimida num arquivo `.ZIP` que, por meio do protocolo SCP, foi transferido para cada uma das quatro máquinas virtuais. Em seguida, em cada máquina, descompactou-se o arquivo `.ZIP` de forma que o caminho para diretório do projeto ficou igual em todas as máquinas: `"/home/ec2-user/infra"`.

A fim de tornar as quatro máquinas virtuais num cluster gerenciado pelo Docker Swarm, executou-se no `Node1` o comando `"docker swarm init"`, que inicializou um *cluster* Swarm que tem o respectivo nó como líder e forneceu um *token* que possibilita que outras máquinas ingressem no *cluster* Swarm como *workers*. Desta forma, nas máquinas nomeadas como `Node2`, `Node3` e `Node4`, executou-se o comando `"docker swarm join -token <token-informado-pelo-Node1> <IP-Node1>:2377"` definindo-as como *workers* do *cluster* Swarm, capazes de executar contêineres Docker a pedido do `Node1`.

O comportamento padrão do Docker Swarm ao distribuir os serviços pelos contêineres disponíveis é balancear automaticamente a carga de trabalho, e, como consequência, a alocação de serviços nos nós se torna imprevisível. Sendo assim, para alcançar a disposição dos serviços conforme apresentada na Figura 7, adotou-se a estratégia de atribuir

rótulos às máquinas virtuais no contexto do Docker Swarm e configurar restrições nos serviços no que diz respeito à sua alocação. A atribuição de rótulos aos nós do *cluster* Swarm é realizada pelo nó líder por meio do comando `"docker node update -label-add nome-vm=<node1|node2|node3|node4> <node-hostname>"` e a restrição de alocação nos serviços foi realizada adicionando-se a cada serviço do `docker-compose.yaml` a configuração apresentada na Figura 20.

Figura 20 – Restrição de alocação do serviço no Docker Swarm

```
deploy:
  placement:
    constraints:
      - node.labels.nome_vm == node1|node2|node3|node4
```

Fonte: O autor.

Com isso, o *cluster* Swarm está pronto para a execução dos serviços que compõem o consórcio. Executou-se então no nó lider o *script* `downloadFabric.sh`[13] para que os binários da Fabric estejam disponíveis nesta máquina virtual e os *scripts* que foram mencionados anteriormente neste capítulo possam ser executados durante a inicialização da rede *blockchain*. Por fim, ainda no nó líder, executou-se o *script* `init.sh`, responsável por iniciar e configurar a rede Hyperledger Fabric desenvolvida nesta pesquisa e, por meio do terminal do serviço *client*, alocado no próprio nó líder, realizou-se a instalação do contrato CertificacaoCCN na rede.

É importante mencionar que o carregamento da pasta do projeto para as máquinas virtuais, bem como o *deploy* da rede Fabric no *cluster* Swarm foi repetido três vezes, uma vez para a rede empregando a arquitetura baseada no modo de ordenação RAFT com três *orderers* apresentada na Figura 7, outra vez para a arquitetura baseada no modo de ordenação RAFT com sete *orderers* (Figura 8) e, por fim, mais uma vez para a arquitetura baseada no modo de ordenação SOLO com um *orderer* (Figura 9).

4.5 BENCHMARK COM HYPERLEDGER CALIPER

A configuração do Caliper, um *framework* para *benchmark* de performance, envolveu três etapas: (i) o preenchimento de arquivos de configuração que mapeiam os serviços e identidades criptográficas do consórcio, (ii) a criação de um arquivo de configuração do *benchmark* que define quantas rodadas de testes serão executadas e o volume de transações enviadas em cada rodada e, por fim, (iii) a programação de uma classe Javascript contendo um método que submete uma transação de emissão de certificado cada vez que é invocada.

O mapeamento dos serviços e identidades criptográficas foi realizado por meio do arquivo `networkConfig.yaml` e de três arquivos YAML denominados `connection-cna.yaml`,

[13] Disponível em: https://github.com/lucas-ads/Plataforma-PBC.HyperledgerFabricSwarm/tree/main/fabric

`connection-frig.yaml` e `connection-cert.yaml` presentes na pasta `caliper`[14] dentro do repositório do projeto. No `networkConfig.yaml` foram especificados o nome do canal, o nome do contrato inteligente, as organizações de *peers*, as identidades e chaves privadas de seus usuários administradores, bem como os caminhos para os arquivos YAML de conexão. Nos arquivos YAML de conexão, foram especificados os endereços dos *anchor peers* na rede, os caminhos para seus certificados TLS, bem como os endereços e certificados TLS dos *orderers* com os quais os respectivos *peers* podem se comunicar.

A configuração do *benchmark* foi realizada por meio do arquivo `benchmark-config-.yaml`. Neste arquivo foram configuradas seis rodadas de testes, variando o quantitativo total de transações a serem enviadas (`txNumber`), a taxa de envio de Transações por Segundo (TPS) e o tamanho do certificado a ser submetido na transação, por meio do parâmetro customizado `inputSize`, que é enviado à classe `MyWorkload` no arquivo `workload.js`, conforme modelo apresentado na Figura 21. Foi empregado o valor `"fixed-rate"` no atributo `rateControl.type` para que o quantitativo de transações definido no TPS fosse submetido à rede de forma fixa, independentemente da carga de trabalho da rede e da existência de filas de transações pendentes de confirmação. Com isso, foi possível avaliar o nível de estresse da rede em cada situação.

Figura 21 – Estrutura de uma rodada de teste do Caliper

```yaml
- label: Certificado Pequeno - Baixo Volume
  description: Certificado Pequeno - Baixo volume de transações
  txNumber: 225
  rateControl:
    type: fixed-rate
    opts:
      tps: 15
  workload:
    module: ./workload/workload.js
    arguments:
      inputSize: small
```

Fonte: O autor.

Por fim, temos a classe `MyWorkload`[15] onde foi programado o método `submitTransaction`, responsável pela montagem do certificado e envio da transação de registro do mesmo. O certificado foi montado como um vetor de argumentos que coincidem com os parâmetros esperados pelo método `RegistraCertificacao` do contrato inteligente. A fim de variar os tamanhos dos certificados enviados, o último campo do certificado, que diz respeito ao *payload* da certificadora, foi preenchido com quantidades variáveis de *bytes*.

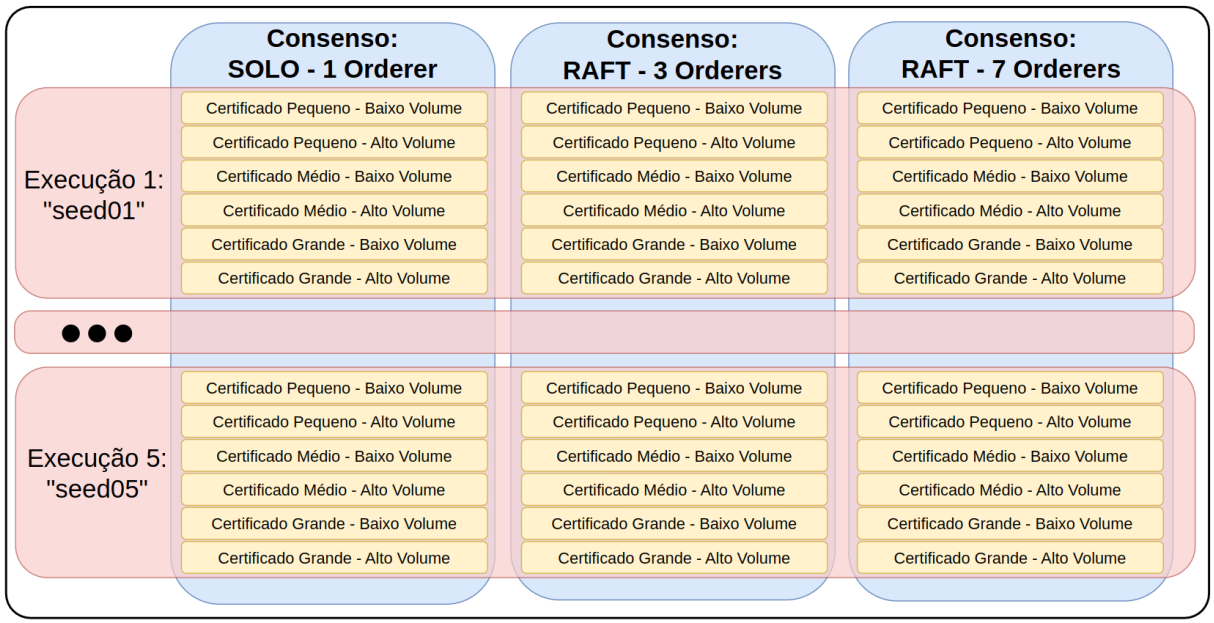
Foram empregadas três faixas de tamanho para os certificados: *"small"*, que varia de 50 a 100 KB, *"medium"*, que varia de 100 a 200 KB e *"large"*, que varia de 200 a 400 KB.

[14] Disponível em: https://github.com/lucas-ads/Plataforma-PBC.HyperledgerFabricSwarm/tree/main/caliper

[15] Disponível em: https://github.com/lucas-ads/Plataforma-PBC.HyperledgerFabricSwarm/blob/main/caliper/workload/workload.js

O tamanho de cada certificado, dentro de tais faixas, de acordo com o `inputSize` definido em cada rodada de teste no `benchmark-config.yaml`, foi definido randomicamente com base em sementes, tornando os *benchmarks* melhor reproduzíveis.

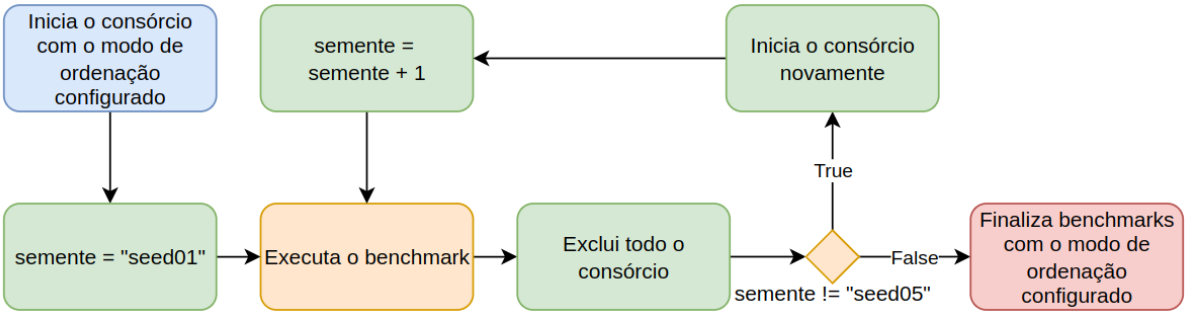
Figura 22 – Organização dos benchmarks



Fonte: O autor.

Conforme apresentado na Figura 22, o conjunto de testes especificados no `benchmark-config.yaml` foi executado cinco vezes, com cinco sementes distintas para a geração de certificados de tamanhos randômicos, em cada um dos três modos de ordenação especificados. A Figura 23 ilustra o processo utilizado na execução dos *benchmarks*, tal processo foi repetido para cada modo de ordenação mencionado.

Figura 23 – Processo de execução dos benchmarks por modo de ordenação



Fonte: O autor.

A execução do Caliper a cada repetição do *benchmark* foi automatizada por meio do *script* `runCaliper.sh`, disponível na pasta `caliper` no repositório do projeto. Já a coleta dos resultados foi realizada de duas formas distintas. Os resultados de latência e *throughput*

são exportados pelo Caliper para um arquivo HTML ao final de cada *benchmark*, esses resultados foram copiados para planilhas Google a fim de facilitar o manuseio.

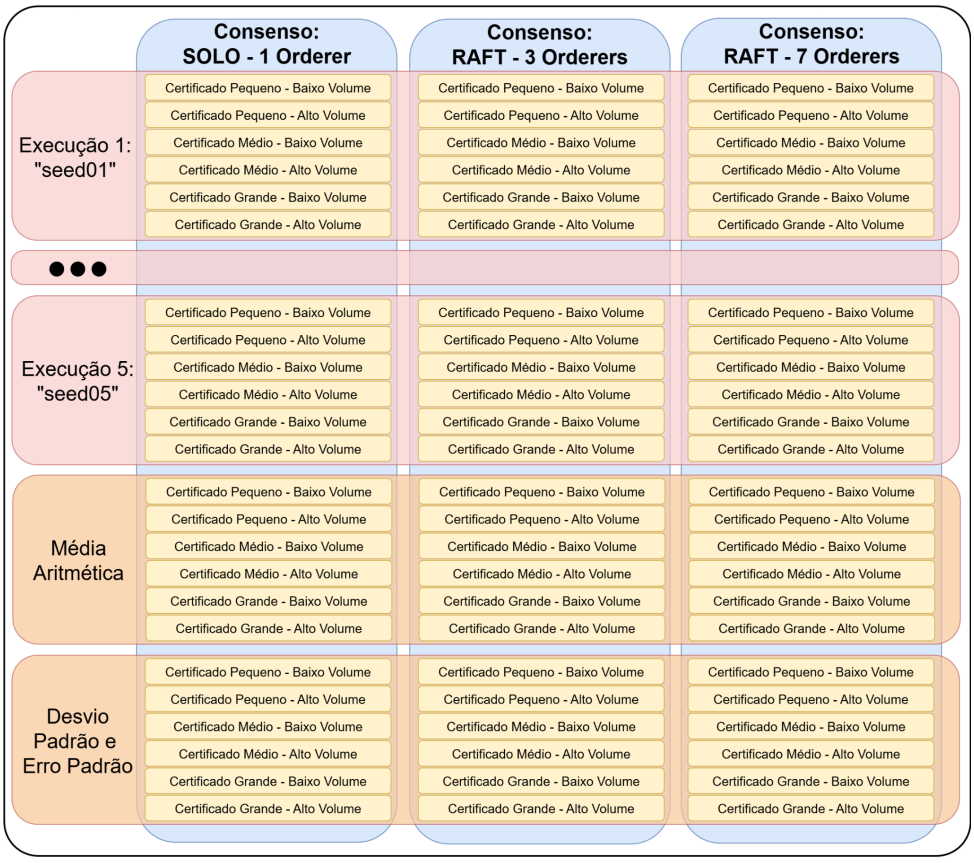
Além disso, durante a execução de cada rodada do *benchmark*, o Caliper emite um *log* a cada cinco segundos informando a quantidade de transações submetidas, bem-sucedidas, mal-sucedidas e não finalizadas. Essa informação foi julgada útil para avaliar a sobrecarga da rede e melhor identificar a carga de trabalho com a qual a rede começa a acumular transações pendentes, indicando de certa forma o seu limite operacional. Apesar de que seria interessante obter este *log* em intervalos menores que cinco segundos, identificou-se que não há configuração disponível no Caliper para alterar tal comportamento. Desta forma, considerando que esses dados não são exportados de outra maneira pela ferramenta, utilizou-se ao final de cada *benchmark* um comando *grep* para filtrar tais *logs*, que por sua vez, foram copiados para uma planilha Google e organizados para consolidação dos resultados.

No próximo capítulo, são apresentados e discutidos os resultados obtidos a partir dos testes de desempenho realizados na infraestrutura descrita neste capítulo, permitindo avaliar sua adequação aos requisitos da certificação PBC.

5 RESULTADOS

Neste capítulo, serão apresentados e discutidos os resultados da pesquisa. Conforme detalhado no Capítulo 4, a prova de conceito da rede **Hyperledger Fabric**, modelada para a Plataforma Baixo Carbono (PBC) da Embrapa, foi hospedada em máquinas virtuais da AWS e testada com a ferramenta **Hyperledger Caliper**. Os *benchmarks* foram realizados empregando-se três diferentes formas de consenso e, para cada forma de consenso, foram criados seis diferentes cenários de teste variando tamanho do certificado e volume de transações (TPS), que, por sua vez, foram repetidos cinco vezes com sementes diferentes a fim de reduzir a influência de fatores externos e garantir resultados consistentes e representativos. A Figura 24 ilustra a metodologia utilizada para consolidação dos resultados obtidos nas cinco repetições de cada cenário de teste, onde foram calculadas as médias aritméticas, os desvios padrões amostrais e os erros padrões de cada valor obtido referente às métricas de latência, *throughput* e formação de filas.

Figura 24 – Metodologia de consolidação dos resultados obtidos



Fonte: O autor.

Os dados obtidos a respeito da latência da rede serão discutidos no subtópico 5.1. O *throughput*, que indica o volume de transações por segundo que a rede é capaz de processar, será abordado no subtópico 5.2. No subtópico 5.3 serão analisadas as dinâmicas de formação de filas de transações pendentes em cada cenário. Uma breve análise do uso de recursos

computacionais, baseado em dados comportamentais fornecidos pela AWS, será realizada no subtópico 5.4. Por fim, no subtópico 5.5, os resultados apresentados serão discutidos buscando resumir as principais observações identificadas ao longo dos experimentos e relacioná-las com os objetivos propostos nesta pesquisa, de forma a avaliar a viabilidade e o desempenho da arquitetura proposta e dos arranjos de consenso empregados.

Para mensurar o volume de dados submetido à rede, a Tabela 4 apresenta as taxas de envio de dados em KB por segundo, calculadas com base no tamanho médio dos certificados e no volume de transações por segundo (TPS) de cada cenário.

Tabela 4 – Taxa de envio de dados por segundo em cada cenário de teste

	Certificado Pequeno	**Certificado Médio**	**Certificado Grande**
Tamanho Médio por Certificado	75 KB	150 KB	300 KB
TPS Baixo (15 certificados por segundo)	1125 KB	2250 KB	4500 KB
TPS Alto (30 certificados por segundo)	2250 KB	4500 KB	9000 KB

Fonte: O autor.

5.1 LATÊNCIA

Em cada execução, para cada cenário de teste, o Hyperledger Caliper forneceu valores para latência mínima, média e máxima. Com isso, os valores para as latências indicadas nas respectivas tabelas são as médias aritméticas dos valores de latência fornecidos pelo Caliper no decorrer das cinco execuções, conforme ilustrado na Figura 24.

Tabela 5 – Estatísticas de Latência - 1 orderer Solo

Consenso: Solo - 1 Orderer						
Certificado	**Volume de Transações**	**Latência Máxima**	**Latência Mínima**	**Latência Média**	**Desvio Padrão**	**Erro Padrão**
Pequeno	Baixo	0,86	0,11	0,37	0,01	0,00
Pequeno	Alto	0,58	0,14	0,28	0,01	0,00
Médio	Baixo	0,72	0,16	0,29	0,00	0,00
Médio	Alto	13,89	0,24	8,51	0,51	0,23
Grande	Baixo	11,60	0,30	7,71	0,79	0,35
Grande	Alto	32,50	0,60	20,50	3,79	1,70

Fonte: O autor.

Tabela 6 – Estatísticas de Latência - 3 orderers RAFT

Consenso: RAFT - 3 Orderers						
Certificado	**Volume de Transações**	**Latência Máxima**	**Latência Mínima**	**Latência Média**	**Desvio Padrão**	**Erro Padrão**
Pequeno	Baixo	0,95	0,12	0,38	0,01	0,00
Pequeno	Alto	0,64	0,15	0,29	0,01	0,00
Médio	Baixo	1,10	0,16	0,31	0,01	0,00
Médio	Alto	12,67	0,23	7,91	0,76	0,34
Grande	Baixo	11,12	0,29	7,22	0,52	0,23
Grande	Alto	32,17	0,52	19,06	2,60	1,16

Fonte: O autor.

Tabela 7 – Estatísticas de Latência - 7 orderers RAFT

Consenso: RAFT - 7 Orderers						
Certificado	**Volume de Transações**	**Latência Máxima**	**Latência Mínima**	**Latência Média**	**Desvio Padrão**	**Erro Padrão**
Pequeno	Baixo	1,04	0,13	0,39	0,00	0,00
Pequeno	Alto	0,72	0,16	0,32	0,01	0,00
Médio	Baixo	1,05	0,18	0,32	0,01	0,00
Médio	Alto	16,75	0,31	10,87	1,03	0,46
Grande	Baixo	13,85	0,31	9,58	1,18	0,53
Grande	Alto	37,71	0,53	26,34	4,23	1,89

Fonte: O autor.

Considerando que os experimentos foram conduzidos empregando-se cinco sementes diferentes numa sequência de cinco execuções, os resultados obtidos representam um conjunto amostral de toda uma população de resultados possíveis existentes. Sendo assim, empregou-se, para cada cenário (tamanho de certificado e volume de transações), com cada modo de consenso, a fórmula do Desvio Padrão Amostral ao conjunto de latências médias fornecidos pelo Caliper nas cinco execuções, cuja fórmula é:

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

Onde:

- x_i são os valores das latências (uma por execução);
- $\bar{x}$ é a média dessas latências;
- n é o número de execuções, neste caso, 5.

Calculados os desvios padrões, foi possível calcular o Erro Padrão em cada cenário, utilizando-se a fórmula:

$$erro = \frac{s}{\sqrt{n}}$$

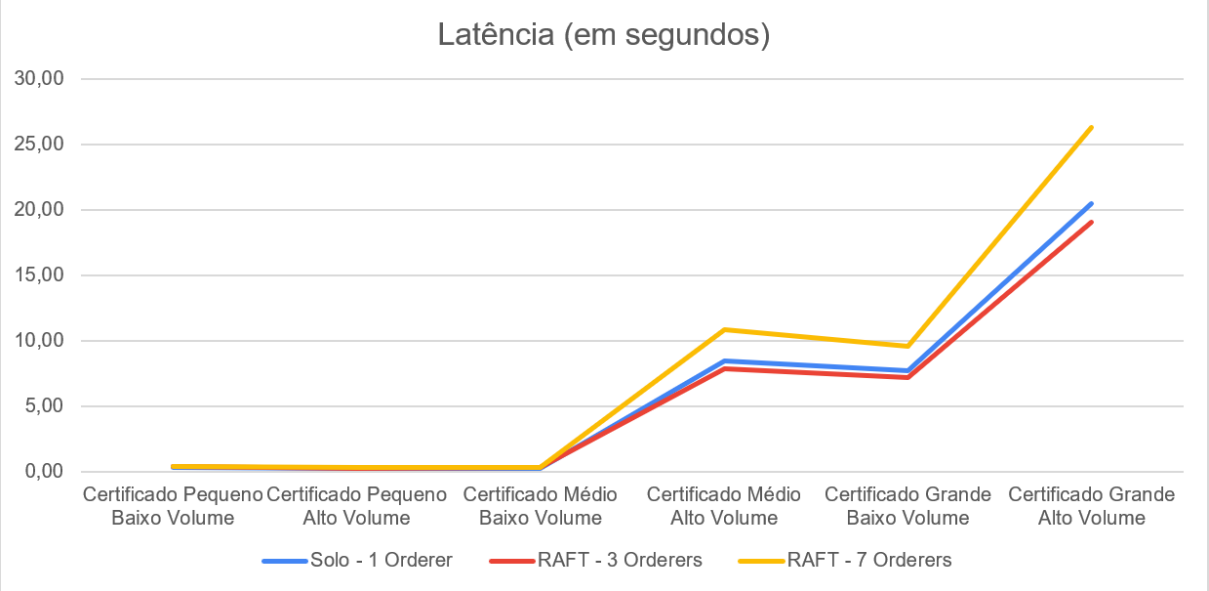
Onde:

- s é o desvio padrão previamente calculado, e;
- n é o número de execuções, neste caso, 5.

As taxas de erros padrões obtidas, sempre abaixo de 10% das latências médias, indicam que tais médias são suficientemente precisas como estimativa da latência real da rede. A Figura 25 ilustra a variação da latência média entre os diferentes cenários de teste. Observa-se que, nas três variações do serviço de ordenação, a latência se mantém próxima de zero nos cenários com certificados pequenos, independentemente do TPS, bem como com certificados de tamanho intermediário e baixo TPS.

Já nos cenários com certificados médios e TPS alto e com certificados grandes e baixo TPS, é possível notar um considerável crescimento da latência da rede nas três variações do modo de ordenação, o que indica que nestes casos a rede está sendo submetida a uma carga de trabalho mais estressante. Por fim, nos cenários com certificados grandes e alto volume de transações, obtêm-se latências ainda mais altas, entre 15 e 30 segundos.

Figura 25 – Variação de latência média entre os cenários de teste



Fonte: O autor.

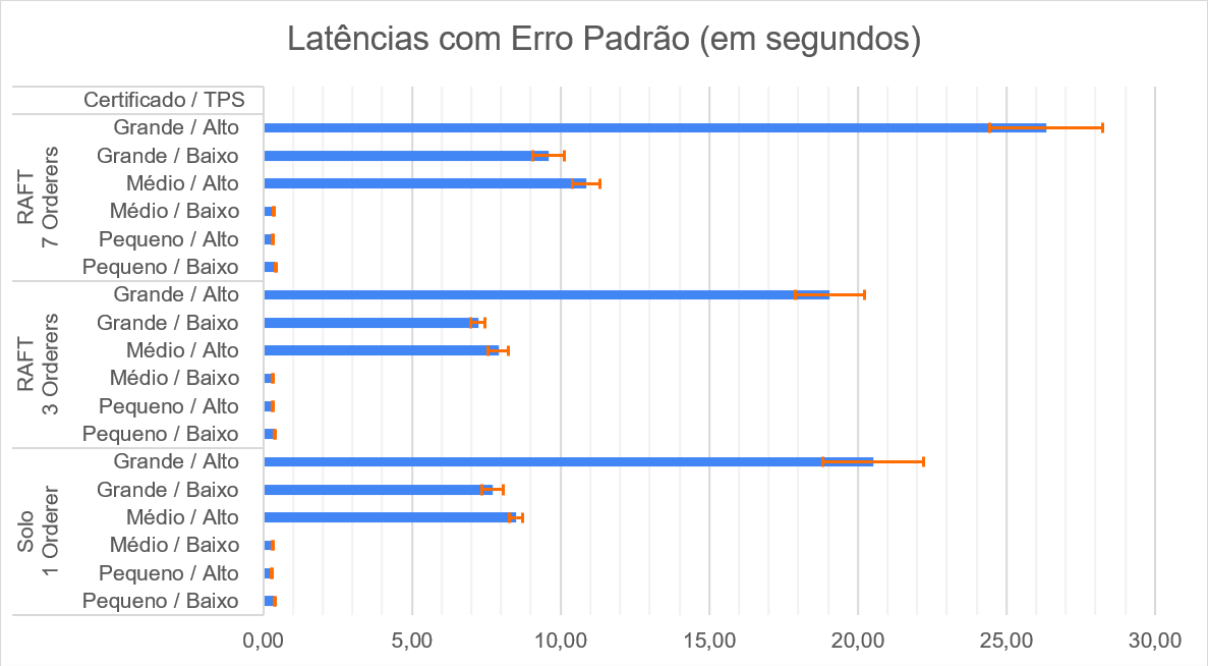
Observando-se o gráfico apresentado na Figura 25 e os dados da Tabela 4, é possível observar que, a partir do cenário com certificado médio e alto TPS, onde a taxa média de envio de dados é de 4500 KB, a latência apresenta um crescimento desproporcional ao crescimento da taxa de envio de dados, o que sugere uma deterioração da capacidade de processamento de transações da rede a partir deste ponto. Tal deterioração será mais observada e discutida nos próximos subtópicos.

Ainda na Figura 25, observam-se desempenhos parecidos entre os cenários onde empregou-se o modo Solo de ordenação e o protocolo RAFT usando 3 nós de ordenação.

Por outro lado, o modo de ordenação baseado no RAFT com 7 ordenadores resultou numa latência média cada vez maior em comparação aos outros modos de ordenação à medida que a taxa de envio de dados cresceu.

A Figura 26 apresenta as latências médias nos mesmos cenários que a Figura 25, porém, com enfoque na apresentação dos erros padrões para análise da consistência e previsibilidade da rede em cada cenário. Novamente observa-se que, para as três diferentes formas de ordenação, o erro padrão cresce à medida que a taxa de envio de dados aumenta. Além disso, reforça-se a percepção de que os resultados dos *benchmarks* nos modos de ordenação Solo e RAFT com três *orderers* foram semelhantes, obtendo empates técnicos em relação aos erros padrões na maioria dos cenários, enquanto que, nos três cenários com maior carga de trabalho, as latências ficaram mais elevadas no modo de ordenação RAFT com sete *orderers*.

Figura 26 – Latência média e erro padrão nos cenários de teste



Fonte: O autor.

5.2 THROUGHPUT

O *throughput* mede a quantidade de transações de escrita de certificados confirmadas por segundo. Essa métrica é crucial para estimar a capacidade da rede de lidar com cargas de trabalho elevadas. As Tabelas 8, 9 e 10 mostram as estatísticas de *throughput*, incluindo o número de transações bem-sucedidas e falhas.

Tabela 8 – Estatísticas de Throughput - 1 orderer Solo

Consenso: Solo - 1 Orderer						
Certificado	**Volume de Transações**	**Sucesso**	**Falha**	**Throughput**	**Desvio Padrão**	**Erro Padrão**
Pequeno	Baixo	225	0	14,56	0,42	0,19
Pequeno	Alto	450	0	29,12	0,13	0,06
Médio	Baixo	225	0	14,44	0,29	0,13
Médio	Alto	450	0	19,62	0,41	0,18
Grande	Baixo	225	0	10,30	0,27	0,12
Grande	Alto	450	0	11,08	0,64	0,29

Fonte: O autor.

Tabela 9 – Estatísticas de Throughput - 3 orderers RAFT

Consenso: RAFT - 3 Orderers						
Certificado	**Volume de Transações**	**Sucesso**	**Falha**	**Throughput**	**Desvio Padrão**	**Erro Padrão**
Pequeno	Baixo	225	0	14,38	0,48	0,21
Pequeno	Alto	450	0	29,06	0,15	0,07
Médio	Baixo	225	0	14,10	0,07	0,03
Médio	Alto	450	0	20,16	0,53	0,24
Grande	Baixo	225	0	10,36	0,22	0,10
Grande	Alto	450	0	11,30	0,37	0,16

Fonte: O autor.

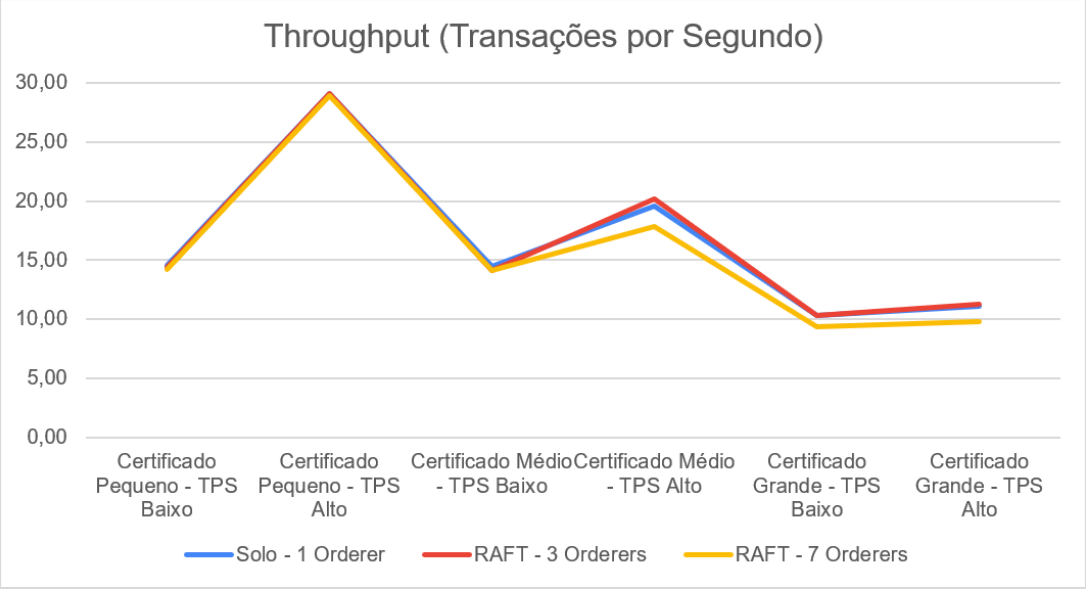
Tabela 10 – Estatísticas de Throughput - 7 orderers RAFT

Consenso: RAFT - 7 Orderers						
Certificado	**Volume de Transações**	**Sucesso**	**Falha**	**Throughput**	**Desvio Padrão**	**Erro Padrão**
Pequeno	Baixo	225	0	14,20	0,39	0,18
Pequeno	Alto	450	0	28,88	0,23	0,10
Médio	Baixo	225	0	14,16	0,26	0,12
Médio	Alto	450	0	17,82	0,72	0,32
Grande	Baixo	225	0	9,38	0,41	0,18
Grande	Alto	450	0	9,82	0,33	0,15

Fonte: O autor.

Considerando que, nos cenários com baixo TPS, foi empregada uma taxa fixa de envio de 15 transações por segundo, e que nos cenários com alto TPS, a taxa adotada foi de 30 transações por segundo, observou-se que, especialmente nos cenários mais leves, um TPS maior resultou em um *throughput* mais elevado. A Figura 27 ilustra esse comportamento.

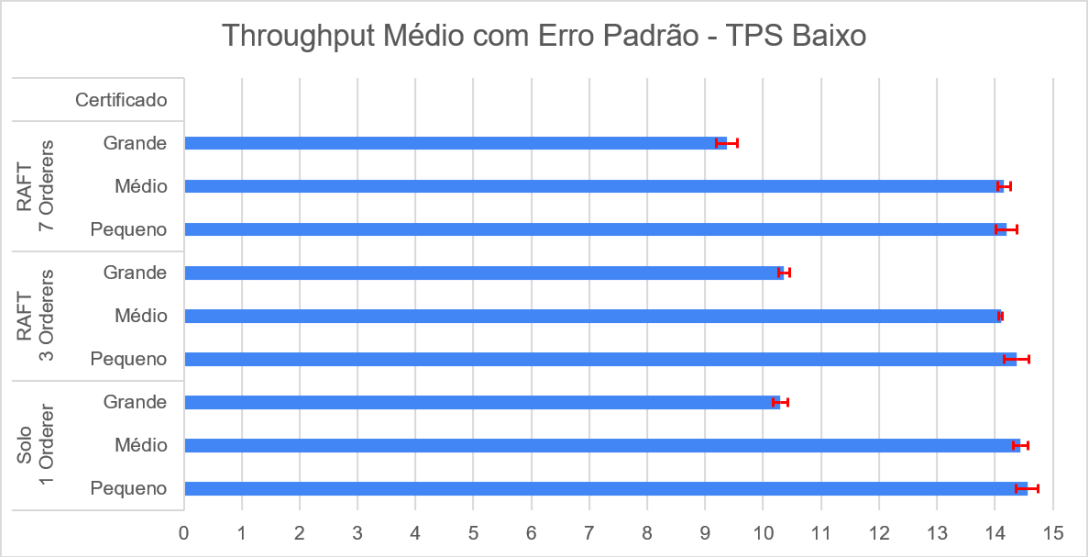
Figura 27 – Throughput da rede em cada cenário



Fonte: O autor.

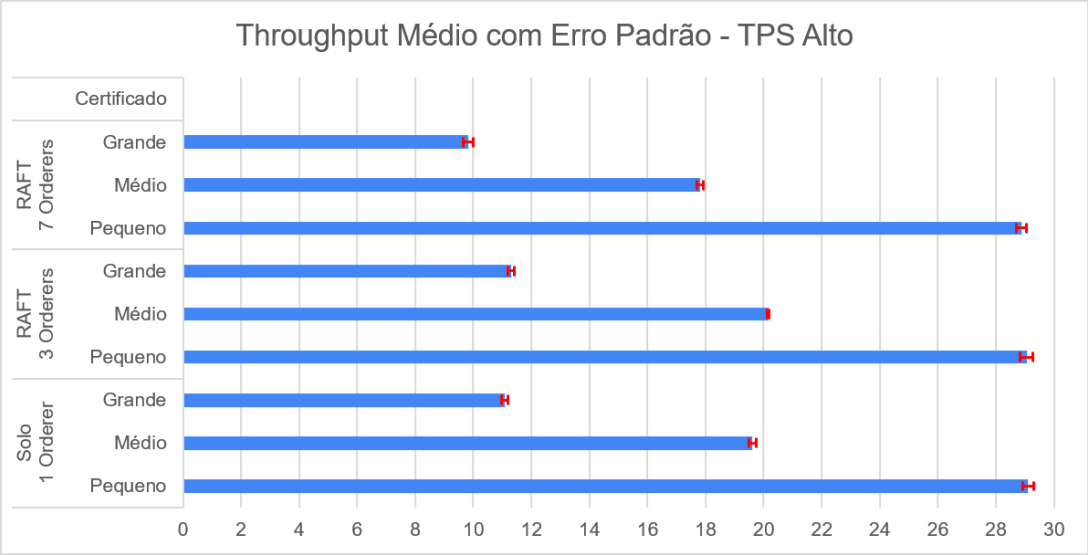
Como o objetivo desta métrica é avaliar a capacidade da rede de processar transações conforme a carga de trabalho aumenta, há o risco de os resultados darem a impressão equivocada de que a rede apresentou maior desempenho apenas por ter recebido mais transações. Por esse motivo, optou-se por apresentar, nas Figuras 28 e 29, os gráficos de *throughput* separadamente para os cenários de baixo e alto TPS, a fim de evitar interpretações incorretas sobre a escalabilidade da rede.

Figura 28 – Throughput da rede nos cenários com baixo TPS



Fonte: O autor.

Figura 29 – Throughput da rede nos cenários com alto TPS



Fonte: O autor.

Observa-se que em ambos os casos, com baixo ou alto volume de envio de transações, a rede conseguiu lidar com facilidade com os certificados pequenos, obtendo um *throughput* muito próximo da taxa de envio de transações. Nos cenários com baixo TPS e certificados de tamanho médio, o *throughput* também manteve-se próximo da taxa de envio de transações.

É importante observar que, entre os cenários com baixo TPS e certificados médios e os cenários com alto TPS e certificados pequenos, o volume de dados em Megabytes transferidos pela rede é similar, conforme demonstrado na Tabela 4. O mesmo ocorre entre os cenários com baixo TPS e certificados grandes e os cenários com alto TPS e certificados médios.

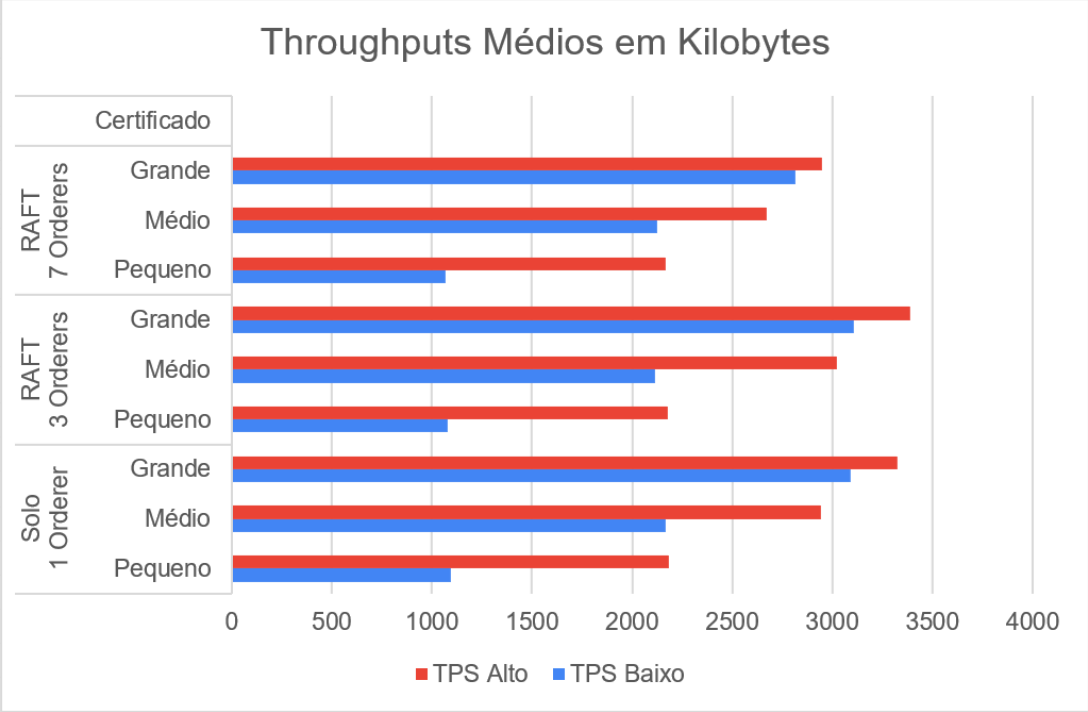
Em todos os cenários envolvendo certificados grandes, bem como nos cenários com TPS alto e certificados de tamanho médio, observou-se uma significativa discrepância entre o volume de transações enviadas e o *throughput* apresentado, indicando que, apesar de conseguir lidar com tal demanda num curto espaço de tempo, já que não houve casos de falhas na execução de transações, a rede está recebendo uma demanda maior do que é capaz de atender de forma contínua.

Observa-se que nos casos com TPS baixo, apresentados pela Figura 28, os erros padrões foram, em geral, mais elevados quando comparados aos erros apresentados na Figura 29, isso pode ser explicado pelas configurações `BatchSize` e `BatchTimeout` da Hyperledger Fabric que fazem com que o *orderer* eleito aguarde juntar um determinado volume de transações em megabytes ou atingir um determinado limite de tempo de espera, o que ocorrer primeiro, para montar um novo bloco e efetivar as transações recebidas. Nos cenários com alto TPS, os limites definidos no `BatchSize` e no `BatchTimeout` são atingidos mais rapidamente, reduzindo o *delay* nas confirmações das transações iniciais, o que pode ter aumentado a consistência dos resultados obtidos.

A Figura 30 apresenta os *throughputs* médios, em kilobytes, obtidos em cada cenário,

com base nos volumes calculados e exibidos na Tabela 4. As semelhanças nos valores de *throughput*, dentro de cada modo de ordenação, entre os pares de cenários "Certificado Pequeno com TPS Alto × Certificado Médio com TPS Baixo" e "Certificado Médio com TPS Alto × Certificado Grande com TPS Baixo", que submetem volumes de dados equivalentes por segundo à rede, sugerem que o fator limitante da capacidade da rede foi o volume de dados transferido entre os nós, e não o poder de processamento para ordenação das transações, como costuma ocorrer em *blockchains* com protocolos de consenso com tolerância a falhas bizantinas.

Figura 30 – Throughput da rede em Kilobytes



Fonte: O autor.

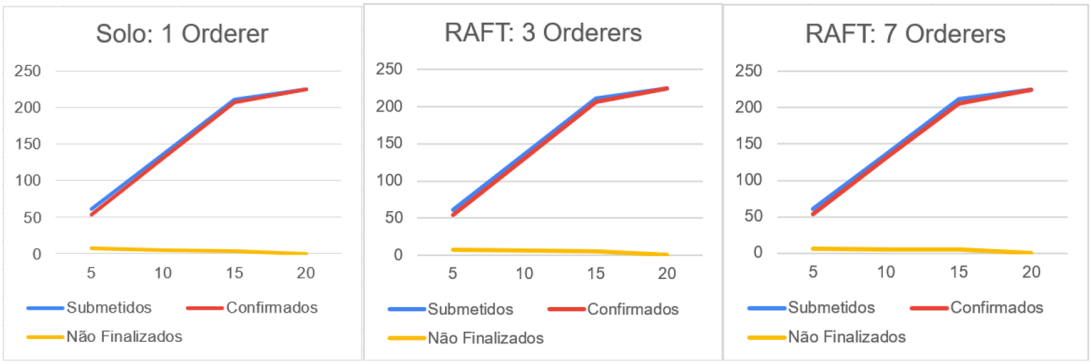
No modo de ordenação Solo, não há algoritmo de consenso, então a ordenação das transações e emissão dos blocos fica a cargo no único *orderer* existente na rede. Já no modo de ordenação baseado no RAFT, o algoritmo de consenso é empregado apenas na eleição de um líder quando a rede é iniciada ou quando o último *orderer* eleito deixa de responder, desta forma, havendo um líder eleito, toda a ordenação de transações e emissão de blocos fica a cargo do mesmo, não havendo na prática sobrecarga da rede no que diz respeito à consenso.

5.3 FORMAÇÃO DE FILAS

A formação de filas diz respeito ao acúmulo de transações pendentes na rede em cada cenário de teste que, como descrito no Capítulo 4, foi extraída de *logs* de execução do Hyperledger Caliper, que informa a cada cinco segundos a quantidade de transações enviadas e não finalizadas.

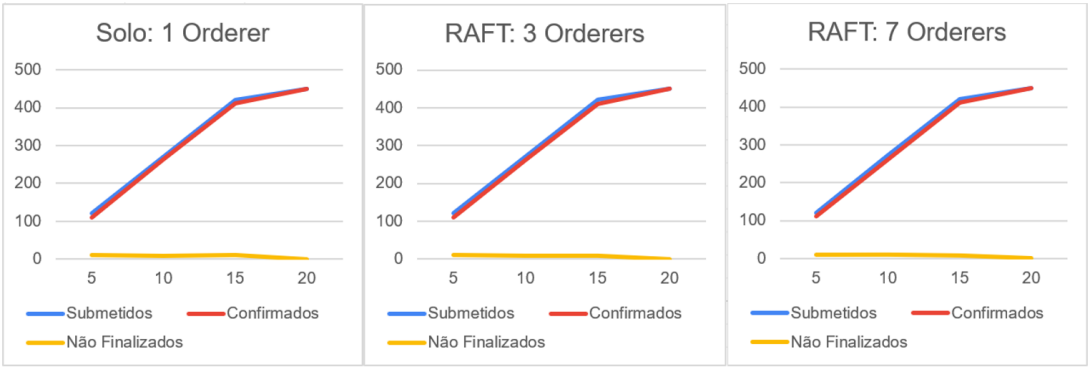
As Figuras 31, 32 e 33, mostram que nos cenários com certificados pequenos, bem como no cenário com certificado médio e baixo TPS, não houve formação significativa de fila, uma vez que a rede foi capaz de processar as transações à medida que elas foram recebidas.

Figura 31 – Formação de Filas com Certificados Pequenos e Baixo TPS



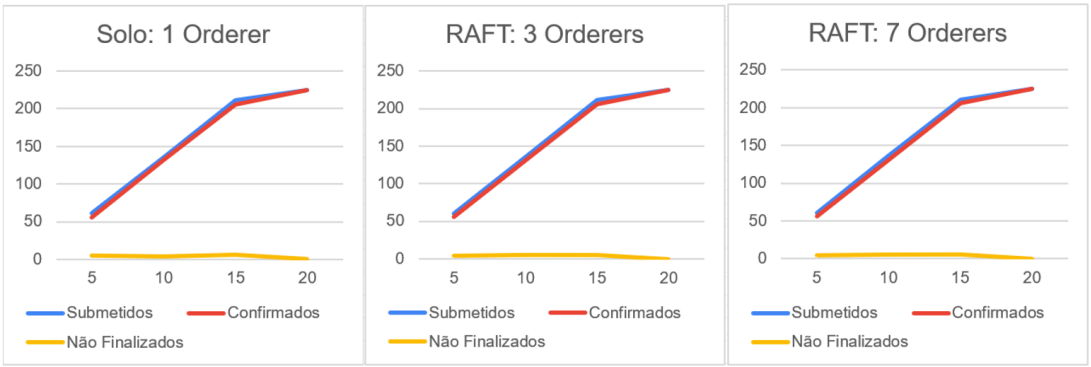
Fonte: O autor.

Figura 32 – Formação de Filas com Certificados Pequenos e Alto TPS



Fonte: O autor.

Figura 33 – Formação de Filas com Certificados Médios e Baixo TPS

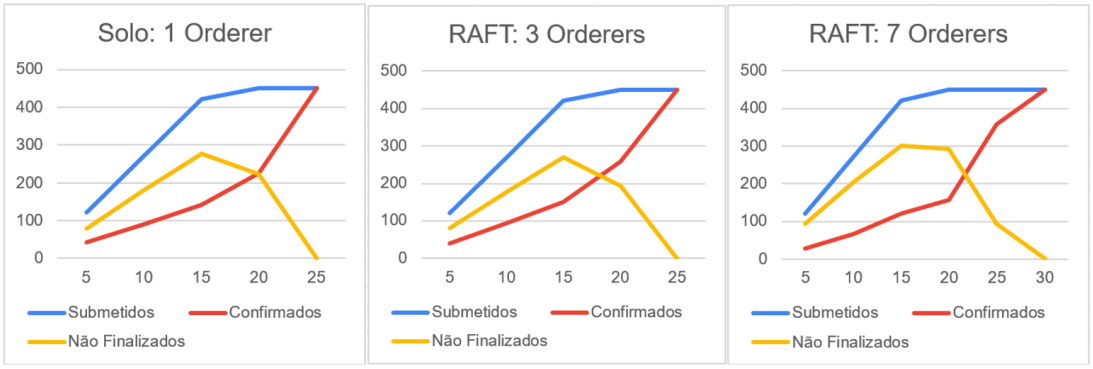


Fonte: O autor.

Já nos cenários com certificados grandes, bem como no cenário com certificado médio e alto TPS, apresentados nas Figuras 34, 35 e 36, observam-se significativos acúmulos

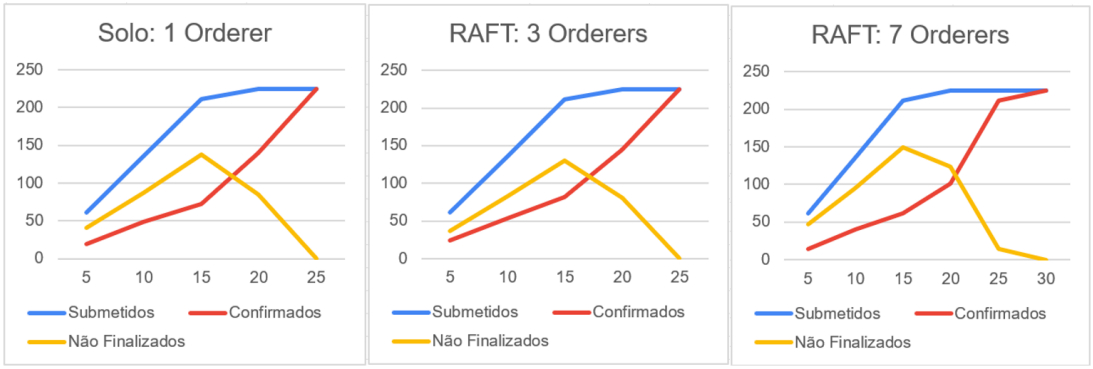
de transações pendentes, destacando a dificuldade da rede, sob estas cargas de trabalho, de dar vazão às transações, em especial enquanto novas requisições estão sendo recebidas.

Figura 34 – Formação de Filas com Certificados Médios e Alto TPS



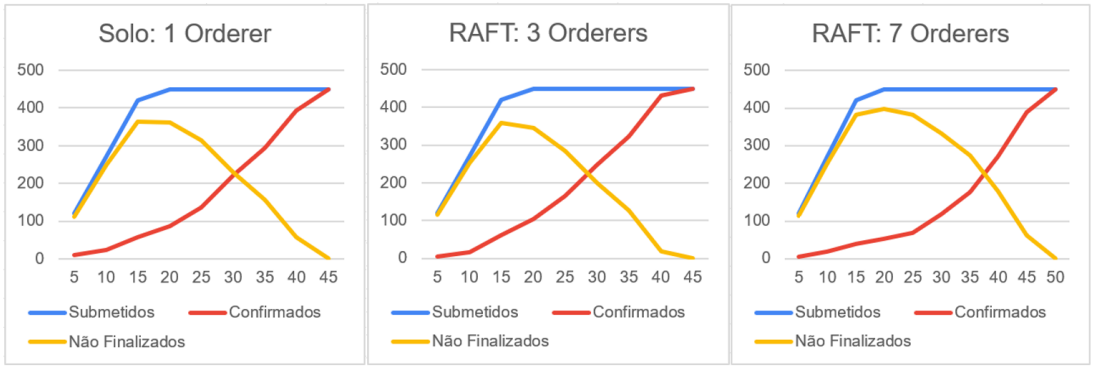
Fonte: O autor.

Figura 35 – Formação de Filas com Certificados Grandes e Baixo TPS



Fonte: O autor.

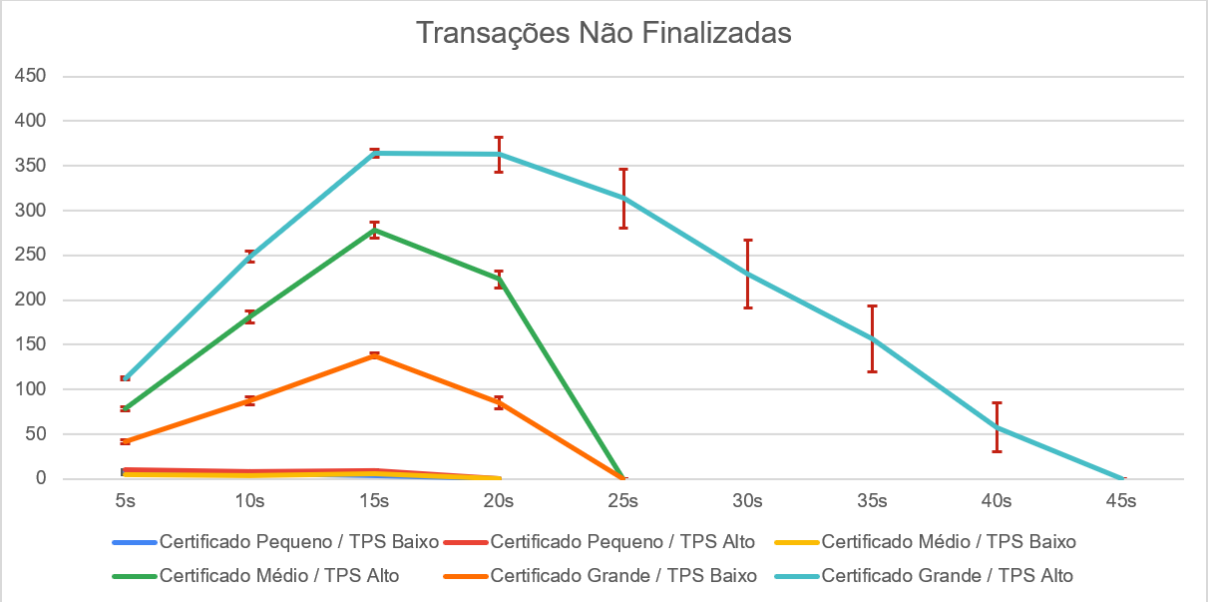
Figura 36 – Formação de Filas com Certificados Grandes e Alto TPS



Fonte: O autor.

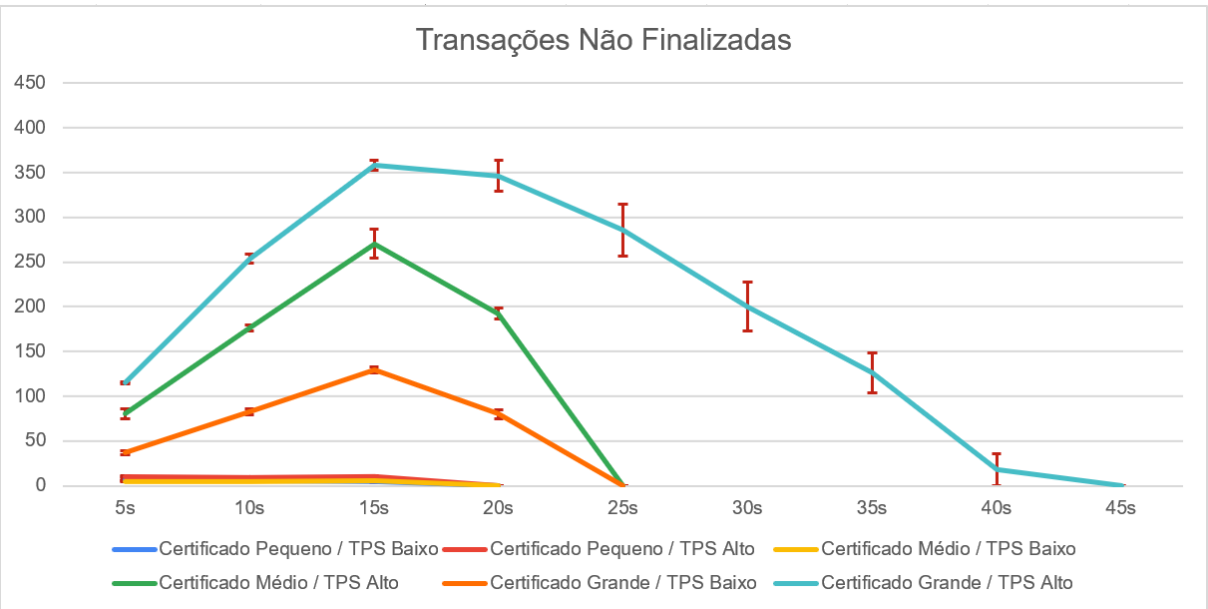
Os gráficos das Figuras 37, 38 e 39, comparam, para cada modo de ordenação, o acúmulo de transações pendentes entre os diferentes cenários de teste, incluindo os níveis de erro padrão.

Figura 37 – Formação de Filas com 1 Orderer em modo Solo



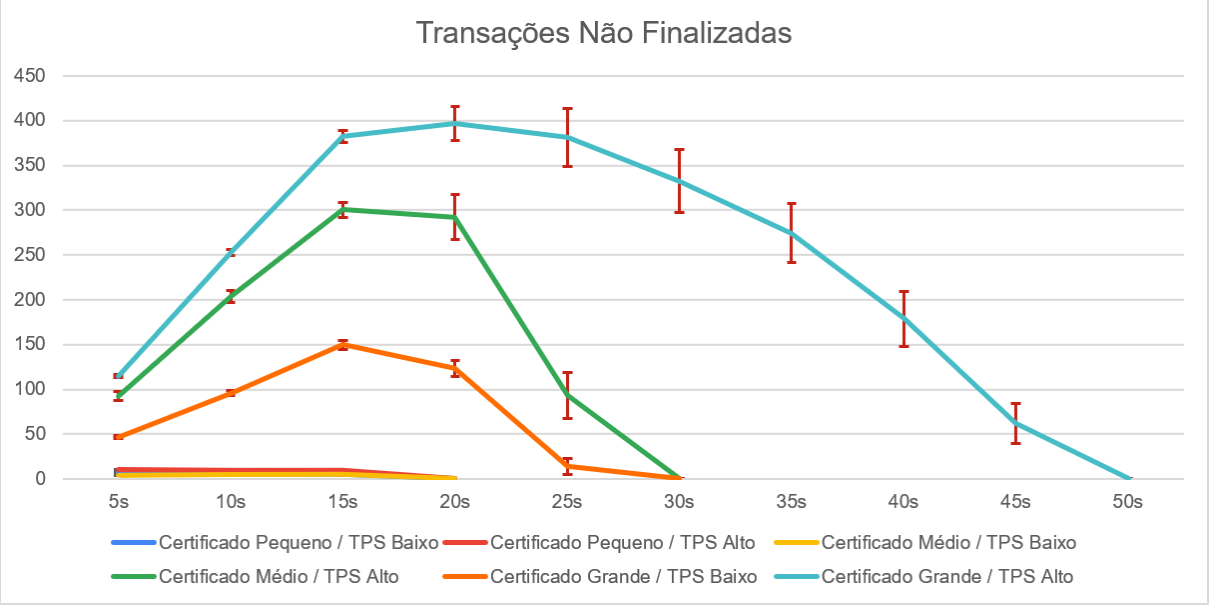
Fonte: O autor.

Figura 38 – Formação de Filas com 3 Orderers em modo RAFT



Fonte: O autor.

Figura 39 – Formação de Filas com 7 Orderers em modo RAFT



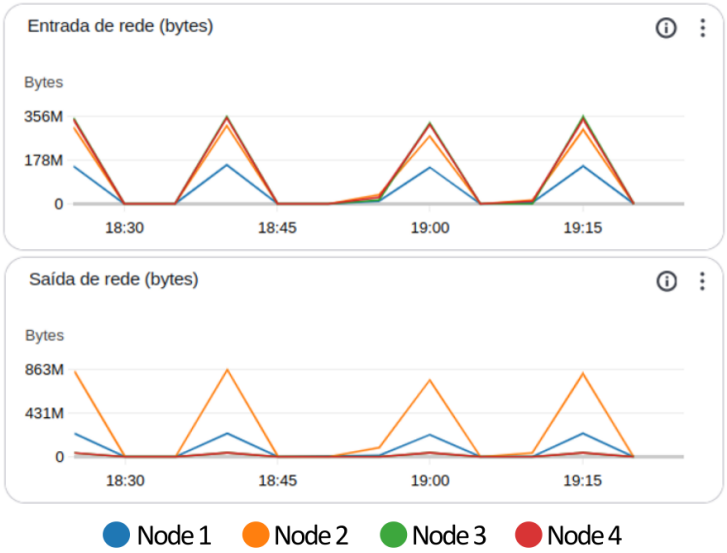
Fonte: O autor.

5.4 MONITORAMENTO DE RECURSOS NA AWS

No *dashboard* da Amazon EC2 é possível monitorar de forma simplificada o uso de CPU e de rede das máquinas virtuais em execução, porém, as estatísticas fornecidas são médias de consumo de intervalos regulares de cinco minutos. Ainda assim, sua análise é interessante para avaliar, de forma geral, o comportamento do *cluster* ao executar os *benchmarks* em cada modo de ordenação testado.

No decorrer de todos os *benchmarks*, as estatísticas de uso de CPU se mantiveram sempre abaixo de 50% em todos os nós, indicando disponibilidade de poder de processamento nas máquinas. Quanto ao uso de rede, observaram-se grandes diferenças entre os nós conforme apresentado pelas Figuras 40, 41 e 42.

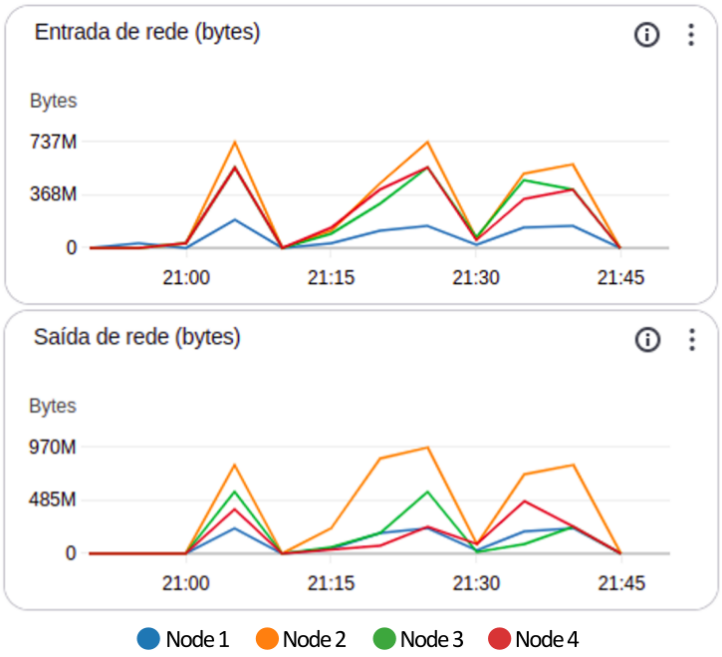
Figura 40 – Monitoramento de rede - Modo Solo



Fonte: O autor.

Os gráficos apresentados em cada uma destas três figuras (40, 41 e 42) abrange o tempo decorrido na execução das cinco repetições do *benchmark* com o modo de ordenação indicado. Isso explica as oscilações nos gráficos que ocorrem em decorrência dos intervalos de tempo entre a execução de cada repetição. Nos três gráficos, a entrada de dados segue um padrão parecido entre os nós 2, 3 e 4 que hospedam os *peers* e *orderers*. Já o nó 1, que hospeda o *client* e o Hyperledger Caliper, teve menor volume de entrada de dados.

Figura 41 – Monitoramento de rede - Modo RAFT com 3 orderers



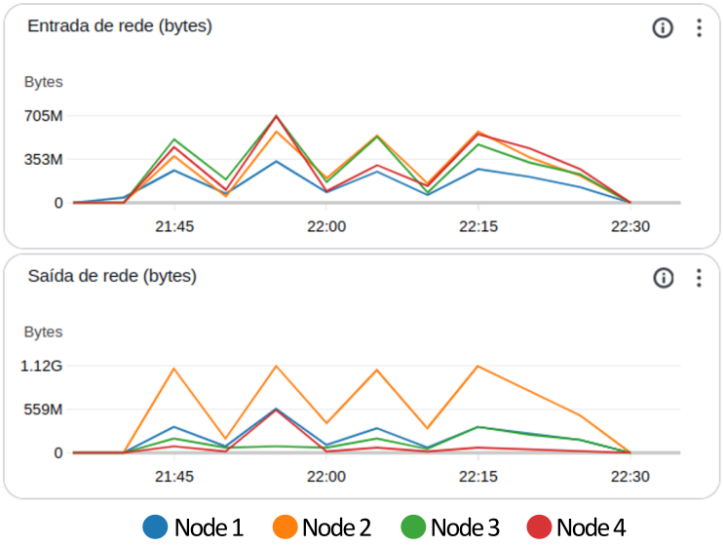
Fonte: O autor.

Quanto à saída de dados, observa-se que em todos os casos, o nó 2 se destacou apresentando uma maior sobrecarga. Nos *benchmarks* com o modo de ordenação Solo,

o nó 2 hospedou o único *orderer* do consórcio, já nos *benchmarks* com os modos de ordenação RAFT, apesar da existência de três ou de sete *orderers*, por estar mais ao topo do `docker-compose.yaml`, o primeiro a ser executado foi o `orderer1`, que é alocado no nó 2, levando-o a ser eleito como líder. Cada bloco de transações gerado pelo *orderer* líder é propagado aos demais *orderers* e a todos os *peers* conectados, gerando a maior saída de dados observada na rede do nó 2.

Na Figura 42, pode-se observar que o volume de saída de dados atingiu níveis mais elevados no modo RAFT com sete orderers do que nos demais modos de ordenação. Isso ocorreu devido ao maior número de *orderers* que precisaram receber cada novo bloco gerado pelo *orderer* líder.

Figura 42 – Monitoramento de rede - Modo RAFT com 7 orderers



Fonte: O autor.

5.5 DISCUSSÃO DOS RESULTADOS

De forma geral, considerando a arquitetura especificada e os mecanismos de consenso adotados, os resultados demonstraram que a infraestrutura foi capaz de processar de forma estável um volume contínuo de aproximadamente 2250 KB por segundo, independentemente da granularidade das transações. No entanto, ao se elevar a taxa de envio para cerca de 4500 KB por segundo, observou-se o acúmulo progressivo de transações pendentes, indicando que a rede não conseguiria sustentar esse volume de forma contínua, delimitando um limite prático de *throughput* associado à capacidade de transmissão de dados dos nós.

Não foram realizados testes intermediários entre 2250 KB/s e 4500 KB/s, o que impede a determinação exata do ponto de saturação da rede. No entanto, os resultados indicam que volumes acima de 2250 KB/s podem ser suportados, mas com um provável aumento no acúmulo de transações pendentes, especialmente em cenários com mais

orderers, devido ao maior volume de dados a serem propagados. Como os cenários com sete *orderers* apresentaram maior acúmulo de transações pendentes, pode-se inferir que o limite máximo de *throughtput* em kilobytes tende a ser menor nestes cenários, em função do maior volume de dados que precisam ser propagados com esse quantitativo de nós ordenadores.

Considerando que as taxas de 15 a 30 transações por segundo, com tamanhos variando entre 50 KB e 400 KB por transação, resultam em volumes expressivos de dados, entende-se que tais valores representam cargas elevadas, sobretudo devido à significativa quantidade de conteúdo textual necessária para atingir esses tamanhos. Sendo assim, os resultados obtidos sugerem que uma rede *blockchain* baseada na Hyperledger Fabric é capaz de suportar volumes compatíveis com cenários de alta demanda de dados, o que reforça a viabilidade da sua adoção na Plataforma PBC, mesmo em aplicações com transações robustas em termos de conteúdo e frequência.

Redes *blockchain*, como as baseadas na Hyperledger Fabric, são bastante versáteis e permitem ajustes na arquitetura para melhorar tanto a latência quanto o *throughput*, conforme a necessidade da aplicação. Por exemplo, é possível alterar os parâmetros `BatchSize` e `BatchTimeout`, que controlam a formação dos blocos: o `BatchSize` define o volume máximo de transações que um bloco pode ter, influenciando o volume de dados por bloco e, consequentemente, o *throughput*; já o `BatchTimeout` define quanto tempo o sistema espera antes de fechar um bloco, mesmo que o número máximo de transações ainda não tenha sido atingido, afetando diretamente a latência. Além disso, para evitar sobrecarga em um único *orderer*, pode-se dividir os *peers* em subconjuntos e conectá-los a diferentes *orderers*, permitindo que mais de um *orderer* participe da distribuição de blocos na rede. Essas possibilidades mostram que é possível adaptar a rede de acordo com os requisitos específicos do sistema.

6 CONSIDERAÇÕES FINAIS

Este trabalho teve como objetivo principal o projeto, a implementação e a avaliação de uma rede *blockchain* como prova de conceito para a Plataforma Pecuária de Baixo Carbono (PBC). Para isso, foi selecionado o *framework* Hyperledger Fabric. Foram exploradas diferentes configurações de rede e foi desenvolvido um contrato inteligente simplificado. A validação do sistema e os testes de desempenho foram conduzidos com o *framework* Hyperledger Caliper.

Os resultados demonstram que a rede é capaz de sustentar um *throughput* contínuo de até 2250 KB/s. No entanto, em cargas de 4500 KB/s ou mais, a rede com seis *peers* e um único *orderer* começa a saturar, acumulando transações pendentes, especialmente nos cenários com maior número de *orderers*. As diferentes configurações testadas reforçaram que a Hyperledger Fabric oferece flexibilidade para ajustes arquiteturais que podem otimizar a latência ou o *throughput*, conforme as necessidades da aplicação.

As estatísticas de latência e de *throughput*, considerando que as transações envolvendo a marca CCN tendem a apresentar baixa granularidade, indicam que a infraestrutura é capaz de suportar uma aplicação voltada à sua gestão. Considerando transações com tamanho médio de 50 KB, a rede pode sustentar aproximadamente 45 transações por segundo (cerca de 162 mil transações por hora). Por outro lado, para transações com tamanho médio de 500 KB, a infraestrutura suporta aproximadamente 4,5 transações por segundo (cerca de 16,2 mil transações por hora).

Caso seja necessário o armazenamento de anexos, como fotografias ou documentos digitalizados, soluções de armazenamento externo podem ser adotadas, a exemplo das abordagens propostas por Vayadande et al. (2022), Shinde et al. (2019), Menezes, Araújo e Nishijima (2023) e Elmay et al. (2022), nas quais apenas as *hashes* ou referências aos arquivos são registradas na *blockchain*.

Ressalta-se ainda que o protocolo Raft demonstrou desempenho consistente durante os testes, de modo que o principal gargalo observado no *throughput* esteve relacionado à largura de banda das conexões de rede das máquinas virtuais utilizadas. Assim, a adoção de instâncias com maior capacidade de transmissão de dados, mantendo-se o mesmo poder de processamento em termos de CPU e memória RAM, tende a produzir resultados superiores.

Este trabalho contribui para a área de computação distribuída ao apresentar uma especificação detalhada da construção e configuração de uma rede baseada em Hyperledger Fabric, acompanhada de estatísticas de desempenho obtidas a partir de testes controlados. Além disso, os resultados podem ser úteis para a Embrapa, na medida em que oferecem subsídios técnicos para a adoção da tecnologia no contexto da certificação da Plataforma de Pecuária de Baixo Carbono. A descrição das principais funcionalidades da plataforma e a análise de seu impacto no desempenho também permitem que a infraestrutura seja

adaptada a futuras necessidades que venham a ser identificadas.

Como trabalho futuro, propõe-se a especificação e modelagem de uma DApp voltada à certificação CCN, utilizando a rede *blockchain* projetada neste estudo como infraestrutura base. A construção dessa DApp possibilitaria não apenas a definição mais precisa dos requisitos funcionais e não funcionais do sistema final, como também permitiria a validação prática do modelo de dados, dos fluxos de transações e dos mecanismos de autenticação e autorização previstos. Além disso, o desenvolvimento de uma aplicação real contribuiria para ampliar a compreensão sobre a usabilidade da solução em um cenário concreto, facilitando a identificação de melhorias ou adaptações necessárias na arquitetura da rede.

Outra possibilidade de continuação deste trabalho envolve a migração da rede para a versão 3.x da Hyperledger Fabric, especialmente quando ocorrer o lançamento de uma versão LTS (Long-Term Support), com o objetivo de testar e comparar o desempenho da mesma rede, porém, utilizando um algoritmo de consenso com tolerância a falhas bizantinas (BFT), que não é suportado nativamente até a versão 2.x da Fabric. A adoção de um consenso BFT poderia elevar a resiliência da rede contra nós maliciosos, relevante em ambientes com menor confiança entre participantes. Tais experimentos permitiriam avaliar o impacto desses algoritmos no *throughput*, na latência e na escalabilidade da rede, contribuindo para decisões mais fundamentadas quanto à configuração ideal para o ambiente da Plataforma PBC.

REFERÊNCIAS

ABIEC. **Beef Report 2024 - Perfil da Pecuária no Brasil**. [S.l.], 2024. Disponível em: https://abiec.com.br/publicacoes/beef-report-2024-perfil-da-pecuaria-no-brasil. Acesso em: 15 abr. 2025.

ALVES, Fabiana Villa; ALMEIDA, Roberto Giolo de; LAURA, Valdemir Antônio. **Carne Carbono Neutro: um novo conceito para carne sustentável produzida nos trópicos**. [S.l.], 2015. Disponível em: https://www.infoteca.cnptia.embrapa.br/infoteca/bitstream/doc/1056155/1/Carnecarbononeutro1.pdf. Acesso em: 7 jan. 2024.

ALVES, Fabiana Villa; ALMEIDA, Roberto Giolo de; LAURA, Valdemir Antônio; COSTA GOMES, Rodrigo da; BUNGENSTAB, Davi José. Marcas-conceito e a proposta de uma Plataforma de Pecuária de Baixo Carbono. In*:* ILPF: Inovação com Integração de Lavoura, Pecuária e Floresta. Brasília, DF: Embrapa, 2019. Disponível em: https://ainfo.cnptia.embrapa.br/digital/bitstream/item/202386/1/ILPF-inovacao-com-integracao-de-lavoura-pecuaria-e-floresta-2019.pdf. p. 169–179.

ANDROULAKI, Elli et al. Hyperledger fabric: a distributed operating system for permissioned blockchains. **EuroSys '18: Proceedings of the Thirteenth EuroSys Conference**, 2018. Disponível em: https://dl.acm.org/doi/10.1145/3190508.3190538.

ANTONOPOULOS, Andreas M. **Mastering Bitcoin: Programming the Open Blockchain**. 2ª Edição. Sebastopol, Califórnia: O'Reilly, 2017.

BISWAS, Sujit; SHARIF, Kashif; LI, Fan; MAHARJAN, Sabita; MOHANTY, Saraju P.; WANG, Yu. PoBT: A Lightweight Consensus Algorithm for Scalable IoT Business Blockchain. **IEEE Internet of Things Journal, Volume 7**, 2019. Disponível em: https://ieeexplore.ieee.org/abstract/document/8926457/.

BITANSKY, Nir. Verifiable Random Functions from Non-interactive Witness-Indistinguishable Proofs. **Journal of Cryptology, Volume 33**, p. 459–493, 2020. Disponível em: https://link.springer.com/article/10.1007/s00145-019-09331-1.

BRASIL. **Plano ABC+ (2020-2030): Programas e Estratégias.** 2023. Disponível em: https://www.gov.br/agricultura/pt-br/assuntos/sustentabilidade/planoabc-abcmais/abc/programas-e-estrategias. Acesso em: 11 ago. 2025.

BRASIL. **Plano Setorial de Mitigação e de Adaptação às Mudanças Climáticas para a Consolidação de uma Economia de Baixa Emissão de Carbono na Agricultura**. [S.l.], 2012. Disponível em: https://www.gov.br/agricultura/pt-br/assuntos/sustentabilidade/planoabc-abcmais/publicacoes/download.pdf/view. Acesso em: 11 ago. 2025.

CAPOCASALE, Vittorio; GOTTA, Danilo; PERBOLI, Guido. Comparative analysis of permissioned blockchain frameworks for industrial applications. **Blockchain: Research**

and Applications, Volume 4-1, 2023. Disponível em: https://www.sciencedirect.com/science/article/pii/S2096720922000549.

CHEN, Yineng et al. An improved algorithm for practical byzantine fault tolerance to large-scale consortium chai. **Information Processing Management, Volume 59-2**, 2022. Disponível em: https://www.sciencedirect.com/science/article/abs/pii/S0306457322000140.

CLACK, Christopher D.; MCGONAGLE, Ciarán. **Smart Derivatives Contracts: the ISDA Master Agreement and the automation of payments and deliveries**. 2019. Disponível em: https://doi.org/10.48550/arXiv.1904.01461. Acesso em: 11 mar. 2024.

CONSENSYS. **A permissioned implementation of Ethereum supporting data privacy**. 2025. Disponível em: https://github.com/ConsenSys/quorum. Acesso em: 17 fev. 2024.

CONSENSYS. **Consensys Acquires Quorum Platform from J.P. Morgan**. 2020. Disponível em: https://consensys.io/blog/consensys-acquires-quorum-platform-from-jp-morgan. Acesso em: 17 fev. 2024.

DIMITRIOU, Tassos. Efficient, Coercion-free and Universally Verifiable Blockchain-based Voting. **Computer Networks, Volume 174**, 2020. Disponível em: https://www.sciencedirect.com/science/article/abs/pii/S1389128619317414.

DU, Zhiguo; WU, Zhihui; WEN, Bin; XIAO, Kehui; SU, Ruijia. Traceability of animal products based on a blockchain consensus mechanism. **IOP Conference Series: Earth and Environmental Science**, IOP Publishing, v. 559, n. 1, p. 012–032, Agosto 2020. Disponível em: https://dx.doi.org/10.1088/1755-1315/559/1/012032.

ELMAY, Feruz Kiflay; SALAH, Khaled; YAQOOB, Ibrar; JAYARAMAN, Raja; BATTAH, Ammar; MALEH, Yassine. Blockchain-Based Traceability for Shipping Containers in Unimodal and Multimodal Logistics. **IEEE Access, Volume 10**, 2022. Disponível em: https://ieeexplore.ieee.org/document/9997538.

ETHEREUM. **Ethereum development documentation**. 2025. Disponível em: https://ethereum.org/en/developers/docs/. Acesso em: 11 ago. 2025.

GO-ETHEREUM. **Go-Ethereum Documentation**. 2024. Disponível em: https://geth.ethereum.org/docs/fundamentals/private-network. Acesso em: 11 ago. 2025.

GHANI, Rana F.; SALMAN, Asia A.; KHUDHAIR, Abdullah B.; ALJOBOURI, Laith. Blockchain-based student certificate management and system sharing using hyperledger fabric platform. **Periodicals of Engineering and Natural Sciences, Volume 10-2**,

2022. Disponível em: https://rke.abertay.ac.uk/en/publications/blockchain-based-student-certificate-management-and-system-sharin.

GORENFLO, Christian; LEE, Stephen; GOLAB, Lukasz; KESHAV, Srinivasan. FastFabric: Scaling Hyperledger Fabric to 20,000 Transactions per Second. In*:* 2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC). [S.l.*:* s.n.], 2019. p. 455–463. Disponível em: https://ieeexplore.ieee.org/document/8751452/.

HABER, Stuart; STORNETTA, W. Scott. How to time-stamp a digital document. **J. Cryptology 3**, p. 99–111, 1991. Disponível em: https://doi.org/10.1007/BF00196791.

HELLIAR, Christine V.; CRAWFORD, Louise; ROCCA, Laura; TEODORI, Claudio; VENEZIANI, Monica. Permissionless and permissioned blockchain diffusion. **International Journal of Information Management**, v. 54, p. 102–136, 2020. Disponível em: https://doi.org/10.1016/j.ijinfomgt.2020.102136. ISSN 0268-4012.

HYPERLEDGER FOUNDATION. **Hyperledger Fabric Docs**. 2025. Disponível em: https://hyperledger-fabric.readthedocs.io/en/release-2.5/. Acesso em: 11 ago. 2025.

ITU. **X.509: Information technology - Open Systems Interconnection - The Directory: Public-key and attribute certificate frameworks. Recommendation X.509 (10/19)**. 2024. Disponível em: https://www.itu.int/rec/T-REC-X.509-201910-I/en. Acesso em: 11 ago. 2025.

KAMATH, Reshma. Food Traceability on Blockchain: Walmart's Pork and Mango Pilots with IBM. **The Journal of The British Blockchain Association**, The British Blockchain Association, v. 1, n. 1, jun. 2018. Disponível em: https://jbba.scholasticahq.com/article/3712-food-traceability-on-blockchain-walmart-s-pork-and-mango-pilots-with-ibm.

KARAMACHOSKI, Jovan; MARINA, Ninoslav; TASKOV, Pavel. Blockchain-Based Application for Certification Management. **Tehnički glasnik**, v. 14, p. 488–492, dez. 2020. Disponível em: https://hrcak.srce.hr/247423.

KUO, Tsung-Ting; ROJAS, Hugo Zavaleta; OHNO-MACHADO, Lucila. Comparison of blockchain platforms: a systematic review and healthcare examples. **Journal of the American Medical Informatics Association**, v. 26, n. 5, p. 462–478, mar. 2019. Disponível em: https://academic.oup.com/jamia/article/26/5/462/5419321. ISSN 1527-974X.

LI, Fengqi; LIU, Kemeng; LIU, Jing; FAN, Yonggang; WANG, Shengfa. DHBFT: Dynamic Hierarchical Byzantine Fault-Tolerant Consensus Mechanism Based on Credit. In*:* WEB and Big Data. Cham: Springer International Publishing, 2020. p. 3–17. Disponível em: https://link.springer.com/chapter/10.1007/978-3-030-60290-1_1.

LIU, Manlu; WU, Kean; XU, Jennifer Jie. How Will Blockchain Technology Impact Auditing and Accounting: Permissionless versus Permissioned Blockchain. **Current Issues in Auditing**, v. 13, n. 2, a19–a29, ago. 2019. Disponível em: https://doi.org/10.2308/ciia-52540. ISSN 1936-1270.

MENEZES, Leonardo Dias; ARAÚJO, Luciano Vieira de; NISHIJIMA, Marislei. Blockchain and smart contract architecture for notaries services under civil law: a Brazilian experience. **International Journal of Information Security**, v. 22, p. 869–880, 2023. Disponível em: https://doi.org/10.1007/s10207-023-00673-3.

MONRAT, Ahmed Afif; SCHELÉN, Olov; ANDERSSON, Karl. Performance Evaluation of Permissioned Blockchain Platforms. In*:* 2020 IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE). [S.l.*:* s.n.], 2020. p. 1–8. Disponível em: https://ieeexplore.ieee.org/document/9411380.

NAKAMOTO, Satoshi. **Bitcoin: A Peer-to-Peer Electronic Cash System**. 2008. Disponível em: https://bitcoin.org/bitcoin.pdf. Acesso em: 11 ago. 2025.

OECD; FAO. **OECD-FAO Agricultural Outlook 2024-2033**. [S.l.], 2024. Disponível em: https://openknowledge.fao.org/items/91c62ae8-b070-4ea8-8281-592376520b03. Acesso em: 7 set. 2024.

OLIVEIRA, Marcela Tuler; CARRARA, Gabriel; FERNANDES, Natalia; ALBUQUERQUE, Célio; CARRANO, Ricardo; MEDEIROS, Dianne; MENEZES, Diogo. Towards a Performance Evaluation of Private Blockchain Frameworks using a Realistic Workload. In*:* 2019 22nd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN). [S.l.*:* s.n.], fev. 2019. p. 180–187. Disponível em: https://ieeexplore.ieee.org/document/8685888.

PANWAR, Arvind; BHATNAGAR, Vishal. Distributed Ledger Technology (DLT): The Beginning of a Technological Revolution for Blockchain. In*:* 2ND International Conference on Data, Engineering and Applications (IDEA). [S.l.*:* s.n.], 2020. p. 1–5. Disponível em: https://ieeexplore.ieee.org/document/9170699.

PELLEGRINO, Giampaolo Queiroz et al. **Portfólio de mudanças climáticas da Embrapa: atuação, oportunidades e desafios para inovação**. [S.l.], 2022. Disponível em: https://www.infoteca.cnptia.embrapa.br/infoteca/bitstream/doc/1139548/1/Doc182-2022.pdf. Acesso em: 7 jul. 2025.

POLGE, Julien; ROBERT, Jérémy; LE TRAON, Yves. Permissioned blockchain frameworks in the industry: A comparison. **ICT Express**, v. 7, n. 2, p. 229–233, 2021. Disponível em: https://www.sciencedirect.com/science/article/pii/S2405959520301909. ISSN 2405-9595.

PONGNUMKUL, Suporn; SIRIPANPORNCHANA, Chaiyaphum; THAJCHAYAPONG, Suttipong. Performance Analysis of Private Blockchain Platforms in Varying Workloads. In*:* 2017 26th International Conference on Computer Communication and Networks (ICCCN). [S.l.*:* s.n.], 2017. p. 1–6. Disponível em: https://ieeexplore.ieee.org/document/8038517.

R3. **Corda 5.1: Key Concepts**. 2025b. Disponível em: https://docs.r3.com/en/platform/corda/5.1/key-concepts.html. Acesso em: 10 ago. 2025.

R3. **Corda Open Source 4.12: Key Concepts**. 2025a. Disponível em: https://docs.r3.com/en/platform/corda/4.12/community/about-corda/corda-key-concepts.html. Acesso em: 10 ago. 2025.

SARAF, Chinmay; SABADRA, Siddharth. Blockchain platforms: A compendium. In*:* 2018 IEEE International Conference on Innovative Research and Development (ICIRD). [S.l.*:* s.n.], 2018. p. 1–6. Disponível em: https://ieeexplore.ieee.org/document/8376323.

SHI, Peichang; WANG, Huaimin; YANG, Shangzhi; CHEN, Chang; YANG, Wentao. Blockchain-based trusted data sharing among trusted stakeholders in IoT. **Software: Practice and Experience**, v. 51, n. 10, p. 2051–2064, 2021. Disponível em: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2739.

SHINDE, Disha; PADEKAR, Snehal; RAUT, Siddharth; WASAY, Abdul; SAMBHARE, S. S. Land Registry Using Blockchain - A Survey of existing systems and proposing a feasible solution. In*:* 2019 5th International Conference On Computing, Communication, Control And Automation (ICCUBEA). [S.l.*:* s.n.], 2019. p. 1–6. Disponível em: https://ieeexplore.ieee.org/abstract/document/9129289.

SUNNY, Farhana Akter; HAJEK, Petr; MUNK, Michal; ABEDIN, Mohammad Zoynul; SATU, Md. Shahriare; EFAT, Md. Iftekharul Alam; ISLAM, Md. Jahidul. A Systematic Review of Blockchain Applications. **IEEE Access**, v. 10, p. 59155–59177, 2022. Disponível em: https://ieeexplore.ieee.org/document/9786734.

SWAN, Melanie. **Blockchain: Blueprint for a New Economy**. [S.l.]: O'Reilly, 2015.

SZABO, Nick. Formalizing and Securing Relationships on Public Networks. **First Monday**, v. 2, n. 9, Setembro 1997. Disponível em: https://firstmonday.org/ojs/index.php/fm/article/view/548.

VAYADANDE, Kuldeep; SHAIKH, Rahebar; ROTHE, Suraj; PATIL, Sangam; BAWARE, Tanuj; NAIK, Sameer. Blockchain-Based Land Record System. **ITM Web Conf.**, v. 50, p. 01006, 2022. Disponível em: https://doi.org/10.1051/itmconf/20225001006.

XIAO, Yang; ZHANG, Ning; LOU, Wenjing; HOU, Y. Thomas. A Survey of Distributed Consensus Protocols for Blockchain Networks. **IEEE Communications Surveys**

Tutorials, v. 22, n. 2, p. 1432–1465, 2020. Disponível em: https://ieeexplore.ieee.org/document/8972381.