PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
ESCOLA POLITÉCNICA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

ANÁLISE DE CUSTOS DE BLOCKCHAINS EM COMPUTAÇÃO EM NUVEM

GILBERTO LUIS KOERBES JUNIOR

Trabalho de Conclusão II apresentado como requisito parcial à obtenção do grau de Bacharel em Ciência da Computação na Pontifícia Universidade Católica do Rio Grande do Sul.

Orientador: Prof. Tiago Ferreto

Porto Alegre
2024

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

ANÁLISE DE CUSTOS DE BLOCKCHAINS EM COMPUTAÇÃO EM NUVEM

RESUMO

A utilização da tecnologia *blockchain* tem crescido de forma significativa, especialmente no ambiente empresarial, como uma medida para aumentar a segurança nas transações entre entidades. Em organizações empresariais são preferíveis as *blockchains* privadas, principalmente no setor financeiro, sendo o *Hyperlerdger Fabric* uma alternativa considerada. Da mesma forma, a adoção da computação em nuvem por empresas proporciona uma alternativa para executar *blockchains* com capacidade sob demanda. Entretanto, para esses conjuntos de soluções, devemos considerar os custos associados à implementação dessas tecnologias, incluindo o provisionamento de aplicações em nuvem e a estimativa dos custos relacionados ao crescimento da rede. Neste trabalho, é apresentada uma introdução aos conceitos-chave para entendimentos iniciais, uma revisão de trabalhos relacionados, uma modelagem e implementação com JupyterNotebook para análise de recursos necessários e os custos desses recursos em nuvem. Desse modelo apresentar-se-ão cenários e resultados obtidos, e, por fim, a conclusão com objetivos atingidos e trabalhos futuros.

Como objetivo principal, destaca-se o modelo proposto que analisa a partir da estrutura da *blockchain Hyperledger Fabric* as transações, blocos, o conjunto de blocos encadeados e, por meio de parâmetros de entrada, aplicar-se-ão de acordo com uma projeção do crescimento do processamento, custo de armazenamento e sustentação de componentes da rede em computação em nuvem.

Palavras-Chave: *Blockchain*, Computação em Nuvem, *Hyperledger Fabric*, Custos, Modelo Preditivo.

ABSTRACT

Blockchain technology has seen a significant increase in its use, especially in the corporate environment, as a measure to reinforce the security of transactions between entities. In business organizations, private blockchains, particularly in the financial sector, are preferable, with Hyperlerdger Fabric being a considered alternative. Likewise, the adoption of cloud computing by companies has provided an alternative to running blockchains with on-demand capacity. However, for these solution sets, we must consider the costs associated with implementing these technologies, including provisioning cloud applications and estimating costs related to network growth. In this work, we present an introduction to key concepts for initial understanding, a review of related work, the proposal of a predictive model for costs, and details about future activities. As a main objective, we highlight our proposed model, which analyzes transactions, blocks, the set of blockchain blocks from the Hyperledger Fabric blockchain structure and, considering input parameters, we apply them to a projection of network growth, storage cost and support of network components in cloud computing.

Keywords: Blockchain, Cloud Computing, Hyperledger Fabric, Costs, Predictive Model.

1. INTRODUÇÃO

Blockchain é uma tecnologia disruptiva com impactos nas relações entre pessoas, consumo e produção de bens e serviços [18]. O uso de *blockchains* tem conquistado espaço nos meios corporativos e públicos com o intuito de tornar mais seguras as transações entre entidades. Os modelos mais comuns de *blockchains* são públicos ou privados, sendo os privados mais adequados quando as corporações buscam introduzir *blockchains* em seu negócio [15], como no mercado financeiro. Assim como a adoção de *blockchains*, as corporações têm utilizado cada vez mais a computação em nuvem para executar suas aplicações, visto que é uma alternativa para *blockchains* com capacidade sob demanda.
Porém, para adoção dessas tecnologias, deve-se considerar o custo monetário para provisionamento das aplicações em nuvem e estimar os custos do crescimento da rede na sua totalidade. Nesse contexto, verifica-se haver uma lacuna na abordagem desses pontos e, que se não estimados, geram imprevisibilidade e podem inviabilizar a adoção dessas tecnologias.

Este trabalho propõe uma ferramenta para projetar a alocação dos recursos necessários de CPU, rede e armazenamento de uma *blockchain Hyperledger Fabric*, ou uma *blockchain* com comportamento semelhante, com base na demanda de transações por segundo e valores de configuração. A partir dos recursos preditos, são estimados os custos para hospedar-se-os em uma provedora de computação em nuvem, com valores reais dos recursos informados pelo utilizador. Com os custos estimados, diversos *insights* podem ser obtidos, como, por exemplo, o crescimento contínuo do registro distribuído (*ledger*) impacta nos custos.
O objetivo da ferramenta é auxiliar organizações e partes interessadas na tomada de decisão ao adotar uma *blockchain* e garantir a manutenibilidade a curto, médio e longo prazo.

Neste trabalho, são descritos no Capítulo 2 os principais conceitos para a compreensão dos elementos citados neste documento, como a definição de *blockchain*, alguns modelos empregados, uma breve introdução sobre moedas digitais, componentes e transações do *Hyperledger Fabric*, e definição, características, modelos e serviços de computação em nuvem. No Capítulo 3 são apresentados trabalhos relacionados, com contribuições semelhantes, experimentos para referências e lacunas que a ferramenta cobre de maneira complementar. Na sequência, no Capítulo 4, são descritos os objetivos gerais e específicos deste projeto, explicados os parâmetros de entradas utilizados pela ferramenta e modelagem do problema com base no comportamento de cada recurso para *blockchain* escolhida. No Capítulo 5 apresentamos cenários de uso e resultados obtidos com a ferramenta. Por fim, a conclusão com os objetivos atingidos, contribuições e trabalhos futuros.

2. FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta os principais conceitos sobre *blockchain*, *Central Banking Digital Currency* (CBDC), *Hyperledger Fabric* e Computação em Nuvem que serão utilizados neste trabalho. É apresentada a motivação de *blockchain*, sua arquitetura principal, tipos de redes, vantagens e desvantagens, e exemplos de uso. Na sequência são apresentados breves conceitos sobre CBDCs, assim como a *Hyperledger Fabric*, seus componentes e estruturas para transações em *blockchain*. Por fim, em relação à Computação em Nuvem, são abrangidos os principais modelos, níveis de serviço, características e elementos que agregam no custo e alguns dos serviços que podem ser utilizados.

2.1 Blockchain

A *blockchain* é uma tecnologia disruptiva com impactos nas relações entre pessoas, consumo e produção de bens e serviços [18]. que tem por característica ser o registro distribuído (*Distributed Ledger Technology* — DLT) o qual permite a criação de um registro imutável e seguro de transações. Comumente, as redes *blockchain* funcionam através da criação de blocos de transações adicionados à cadeia de blocos de forma cronológica e descentralizada, ou seja, sem a necessidade de um intermediário centralizado. Cada bloco contém um registro de transações validadas e criptografadas, verificadas por uma rede de nós participantes. Há características e variações das redes para serem utilizadas de diferentes formas e em diferentes contextos.

De modo geral, aplicações *blockchains* em ambientes empresariais possuem alguns requisitos, como [6]:

- Os participantes devem ser identificados/identificáveis;
- As redes precisam ser permissionadas, isso é, privadas;
- Alto desempenho na taxa de transferência de transações;
- Baixa latência de confirmação de transações;
- Privacidade e confidencialidade das transações e dos dados referentes às transações comerciais.

2.1.1 Redes permissionadas e não permissionadas

As redes *blockchains* possuem variações na forma como os participantes integram-se à rede, e podem ser classificadas como permissionadas ou não permissionadas. As diferenças entre as redes permissionadas e não permissionadas incluem o modelo de governança, o algoritmo de consenso, a privacidade e a escalabilidade [18].

As redes permissionadas são caracterizadas por restringirem o acesso a um grupo de participantes autorizados, são mais adequadas para casos de uso empresariais, pois permitem maior controle e privacidade, além de serem mais escaláveis.

As redes não permissionadas são as que qualquer pessoa pode participar e contribuir para a validação das transações, possuem maior transparência, uma vez que todas as transações são registradas publicamente e podem ser verificadas por qualquer participante. Além disso, grandes redes não permissionadas são mais resistentes a ataques, pois exigem que a maioria dos participantes concorde com as transações para que sejam validadas.

2.1.2 Protocolos de consenso

Os protocolos de consenso são algoritmos que permitem que os participantes de uma rede *blockchain* cheguem a um acordo sobre o estado atual da rede. Esses protocolos são essenciais para garantir a segurança e a integridade da rede, pois impedem que um participante mal-intencionado altere o registro de transações. Pode-se destacar como principais a Prova de Trabalho (*Proof of Work* - PoW), Prova de Aposta (*Proof of Stake* - PoS) e Prova de Autoridade(*Proof of Authority* - PoA)

Conforme [18], em PoW, um nó participante só pode adicionar um novo bloco na *blockchain* se provar seu esforço computacional. Para tantos, o participante deve encontrar um valor (comumente chamado de *Nonce*) que satisfaça a condição de esforço mínimo exigido pela rede. Quando esse valor é encontrado, pode ser facilmente verificado pelos demais participantes.

Ainda segundo [18], no PoS, o nó participante que pode adicionar um novo bloco na próxima rodada é determinado pelo tamanho de sua participação — ou aposta — na *blockchain*. Em moedas digitais, isso significa que quanto mais moeda digital um nó dispõe para executar uma transação, maior será a probabilidade de ser selecionado para adicionar o próximo bloco ao *blockchain*. Comparado ao PoW, o PoS é mais econômico porque utiliza menos poder computacional na mineração.

Já na Prova de Autoridade (*Proof of Authority* - PoA), um grupo de nós de confiança, conhecidos como validadores, é responsável pela validação das transações e pela

adição de novos blocos à cadeia de blocos. Os validadores são normalmente selecionados com base na sua reputação, experiência ou hierarquia de participação na rede. Pode-se citar, como exemplo QBFT, IBFT 2.0 e Clique, que são protocolos de consenso que se baseiam em PoA. Os protocolos de consenso de tolerância a falhas bizantinas (*Byzantine Fault Tolerance* - BFT) IBFT, QBFT e Clique foram desenvolvidos para redes de *blockchain* privadas e de consórcio, baseados em PoA e são alternativas mais adequadas para redes onde a confiança entre os participantes é alta.

2.1.3 Escalabilidade

A escalabilidade é fator determinante na escolha e uso das *Blockchains*, sendo permissionadas ou não permissionadas. Segundo [18], as redes permissionadas e privadas podem ser mais escaláveis do que as redes públicas, pois têm menos restrições em termos de tamanho de bloco e taxa de transação. Além disso, as redes permissionadas e privadas podem usar algoritmos de consenso mais eficientes, como o *Proof of Authority* (PoA), que podem melhorar a escalabilidade.

2.2 CDBCs

As CBDCs (*Central Bank Digital Currency*), ou moeda digital do Banco Central, são uma forma de dinheiro digital na mesma unidade corrente nacional e é uma responsabilidade direta do Banco Central do país que a detém[7]. Ao contrário das criptomoedas, as CBDCs são emitidas e apoiadas por um Banco Central, o que as torna uma forma de moeda fiduciária digital. As CBDCs podem ser concebidas para utilização apenas entre intermediários financeiros (como troca entre bancos e instituições financeiras) ou pela economia em geral (uso direto pelos consumidores).

O uso de CBDC depende das necessidades específicas do Banco Central, e em meio digital, a adoção de *Blockchain* pode auxiliar em questões como segurança, interoperabilidade, eficiência e inovação [7].

2.3 Hyperledger Fabric

Conforme [2] o *Hyperledger Fabric* é um projeto de código aberto sob o escopo do *Hyperledger* que possibilita a criação de *blockchains* permissionadas. O *Fabric* permite a implantação de contratos inteligentes com finalidade mais generalista, chamados de *chain-*

code, pois utilizam várias linguagens de programação, incluindo *JavaScript, Java* e *Go*, o que traz mais flexibilidade no desenvolvimento destes contratos.

Como é uma aplicação *blockchain* de registro distribuído (DLT), é importante ressaltar que, apesar de outras *blockchains* também implementarem um Livro-Razão distribuído, o formato da *blockchain* (blocos, transações e componentes) possui implementação diferente de outras. Por exemplo, difere de *Bitcoin* e *Ethereum*, e também de outra aplicação *Hyperledger*, o *Hyperledger Besu* — que possui sua implementação baseada em *Ethereum*. O *Fabric* tem sido uma alternativa considerada por organizações que buscam baixa latência e alta vazão de ponta a ponta.

2.3.1 Componentes de rede e nomenclaturas

O *Hyperledger Fabric* possui vários componentes para formar a rede de participantes, implementar consenso e registrar as transações de forma imutável na *blockchain*. Sendo assim, a abordagem será limitada nos próximos tópicos a elementos que participam de forma explícita para uma transação, bem como não serão aprofundadas as configurações e componentes auxiliares, como *Membership Service Provider* (MSP), infraestrutura de chaves públicas (*Public Key Infrastructure* — PKI), Autoridades Certificadoras (*Certificate Authorities* — CA) e também sobre outros componentes opcionais que não fazem parte do núcleo de funcionamento (como *Apache Kafka* e aplicações para cache de informações).

Endosso

O endosso no *Hyperledger Fabric* refere-se ao processo de aprovação pelo qual os pares validam e assinam transações. Quando um cliente envia uma proposta de transação, a mesma é feita aos pares endossantes, já que existem políticas que podem ser definidas para o quórum de pares endossantes que simulam a execução da transação proposta e, se bem-sucedida, dão seu endosso assinando os resultados, isso é, inserem na mensagem sua assinatura com uso de chaves autorizadas. Vários endossos são coletados para aumentar a resiliência e a confiabilidade da transação. Somente as transações com o número necessário de endossos válidos passam para a próxima fase. O endosso garante que as transações propostas sejam válidas e acordadas pelas partes necessárias antes de serem inseridas com o *blockchain* a fim de melhorar a integridade e a confiabilidade do *Ledger*(Razão) distribuído.

Ledger

O *Ledger* (Livro-razão em livre tradução) no *Hyperledger Fabric*, assim como em outras, é o ponto central de uma *blockchain* e refere-se ao registro e histórico das transações ocorridas, mantido por todos os nós participantes (*Peers*) da rede.

No *Fabric* existem dois tipos de *Ledgers*, o *Log Transaction* (Registo de Transações) e o *World State* (Estado Global):

- *Log Transaction*: o Log de Transações armazena detalhes de todas as transações em blocos, desde o primeiro bloco (bloco *genesis*) até o estado atual. Esse seria o que é denominado *blockchain*, pois os blocos são encadeados uns aos outros através dos seus cabeçalhos *Block Header* que apontam o *Hash* do bloco anterior, conforme ilustra a Figura 2.1. Nesses blocos são inseridas as transações e os metadados do bloco.

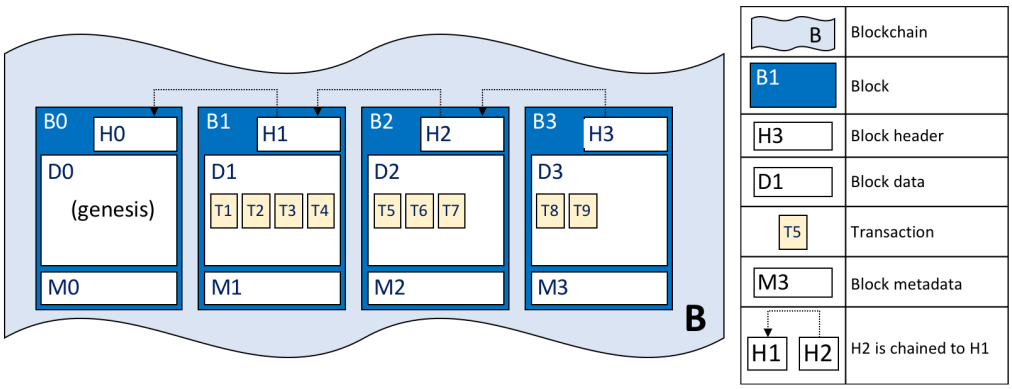


Figura 2.1: *Fabric Blockchain* (Fonte: [6])

- *World State*: representa o estado atual dos ativos, isto é, o último estado válido registrado pela *blockchain*, como mostrado na Figura 2.2. O *World State* é uma implementação que visa simplificar as consultas de transações e otimizar o desempenho da rede. Ao receber uma nova transação, não é necessário buscar e recalcular na *blockchain* informações do registro já que o *World State* terá essas informações do último estado.

Canais

No *Hyperledger Fabric*, os canais (*Channels*) definem uma visão lógica para conectar os participantes da rede, criam sub-redes privadas dentro da rede *blockchain* principal e estabelecem quais participantes podem visualizar os registros de transações (*blockchains*). Em uma rede *Fabric*, pode-se fixar mais de um canal e isolar participantes de determinadas conexões, pois cada canal possui seu *Ledger* separado. É importante ressaltar

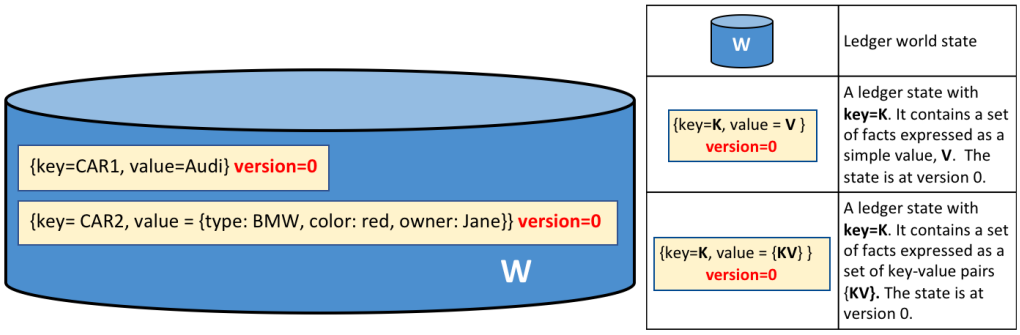


Figura 2.2: *Fabric World State* (Fonte: [6])

que os canais são uma visão e delimitação lógica, pois assumem que a camada subjacente de infraestrutura e rede (tradicional rede de computadores) já tenha comunicação entre os pares.

Pares

No *Hyperledger Fabric*, pares (*peers*) são nós que mantêm uma cópia do *Ledger*, executam *chaincodes* e possuem duas funções principais: endossar e confirmar transações. Para endossar a transação, o endosso é, como já mencionado anteriormente, o momento em que o *peer* simula a transação e inclui sua assinatura na mesma. A confirmação ocorre posteriormente, quando o *peer* recebe a transação do Serviço de Ordenação (*Ordering Service*) reconfirma que sua execução é válida e então grava no *Ledger*.

Em uma rede que utiliza CFT (*Crash Fault Tolerance*), como pode ser configurado o *Fabric*, caso um *peer* falhe ou esteja indisponível, o *Ledger* é atualizado através do protocolo *Gossip* (fofoca). Assim, o canal de participantes depende apenas que um esteja íntegro para salvar o histórico do *Ledger*. Nesse caso, ainda que o histórico de transações (*blockchain*) seja mantido por um único par, a falta dos demais pares pode impactar a execução de novas transações, já que as mesmas podem não ter endosso suficiente, consequentemente, ficarão sem aprovação, e impactarão no funcionamento da rede.

Serviço de ordenação

O Serviço de Ordenação (*Ordering Service*) no Hyperledger Fabric é responsável por organizar as transações em blocos e garantir sua ordem sequencial. Isso o torna tão importante quanto outros componentes. O serviço recebe transações enviadas pelos clientes, previamente já endossadas pelos pares, ordena-as e entrega os blocos a todos os nós da rede. O serviço de pedidos pode usar diferentes algoritmos de consenso e tolerância a falhas, sendo o principal uma implementação própria, o *Raft*.

Por ser uma parte crítica da aplicação, o serviço de ordenação necessita de alta disponibilidade feita pelo *Raft*, o qual simula um *cluster* onde é eleito um líder pelo consenso dos nós *Raft* em execução. Isso significa que, se houver três nós em um canal, poderá suportar a perda de um nó — e restar dois nós. Se tiver cinco nós em um canal, poderá perder dois nós e restam três.

Chaincode, Smartcontracts e SDK

Os Contratos Inteligentes (*SmartContracts*) são códigos que modificam o *World State* dos ativos na *blockchain*, oferecendo métodos para adição, modificação, recuperação e exclusão de estados. Esses contratos definem regras executáveis entre diferentes organizações, isto é, podem condicionar determinadas ações dentro de uma transação, como por exemplo, uma organização A incrementa X em seus ativos e, de forma mútua, uma organização B decrementa X de seus ativos. É importante ressaltar que as transações entre as organizações A e B só ocorrem, pois ambas estão em comum acordo das execuções realizadas.

No *Hyperledger Fabric*, uma *Chaincode* pode ser assimilada aos *SmartContracts*. Assim, um *SmartContract* pode pertencer a uma *ChainCode*, mas uma *ChainCode* pode agrupar mais de um *SmartContract* simultaneamente. As *ChainCodes* também são a definição do que é implementado, logo, *ChainCodes* são de fato quais *SmartContracts* estão implementados à rede. Para uma *ChainCode* poder ser implementada na rede, é preciso que os pares da organização a aceitem. Isso garante que somente serão executados contratos conhecidos e validados pelos participantes que podem ser desenvolvidos em linguagens de programação como *Go*, *Java* e *JavaScript*. Os SDK (*Software Development Kits*) fornecem ferramentas e bibliotecas para desenvolvimento desses *SmartContracts* que interagem com a *blockchain*.

Clientes e Gateway Fabric

Os clientes do *Hyperledger Fabric* são entidades que interagem com a rede e enviam propostas de transações utilizando os *SmartContracts*. O *Gateway Fabric* é uma camada de abstração que simplifica a interação entre aplicações e a rede *blockchain*, e facilita a conexão e a transação. Os clientes não possuem comunicação direta com os pares e acesso ao *Ledger*. Ao invés disso, enviam transações para o *Gateway* que encaminha para todos os pares de endosso.

Transações

As transações incorporadas nos blocos são formadas pelos seguintes campos (Figura 2.3):

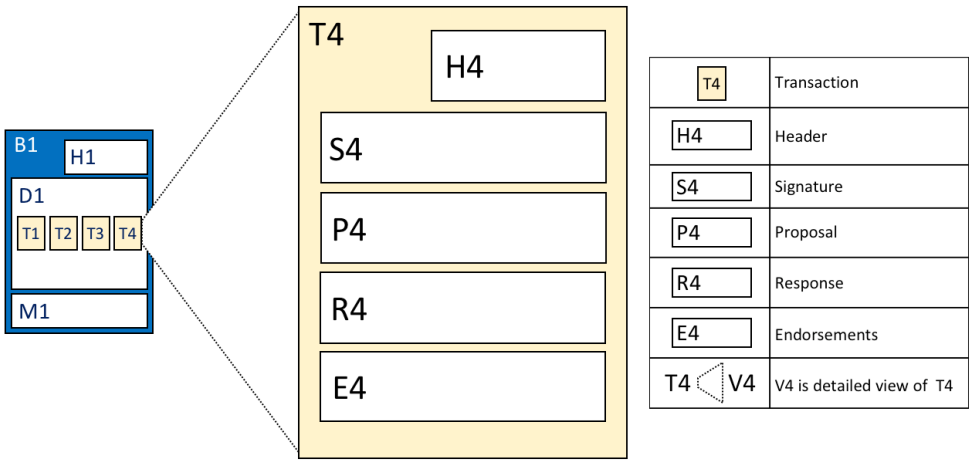


Figura 2.3: Estrutura de uma transação no *ledger* (Fonte: [6])

- Cabeçalho (*Header* - H4): contém metadados essenciais sobre a transação, como o nome e a versão relevantes do *Chaincode*.
- Assinatura (*Signature* - S4): inclui uma assinatura criptográfica gerada pela aplicação do cliente que usa sua chave privada. Essa assinatura é crucial para verificar a integridade dos detalhes da transação e garantir que não foram adulterados.
- Proposta (*Proposal* - P4): codifica parâmetros de entrada de um aplicativo para um contrato inteligente ao definir qual atualização ocorrerá no *Ledger*. O contrato inteligente, quando executado, utiliza esses parâmetros juntamente com o *World State* atual para determinar o novo *World State*.
- Resposta (*Response* - R4): captura valores antes e depois do *World State* como um conjunto de leitura e gravação assim como representa a saída de um contrato inteligente.
- Endossos (*Endorsements* - E4): consiste em uma lista de respostas de transações assinadas de cada organização necessária (*peers*), atende aos critérios da política de vários endossos incluídos, cada um codificando a resposta específica da transação de sua respectiva organização.

2.3.2 Funcionamento do registro de transações

Conforme [14], o fluxo de transação (apresentado na Figura 2.4) é realizado pelas seguintes etapas:

1. O aplicativo cliente envia uma proposta de transação a pares de diferentes organizações para endosso;

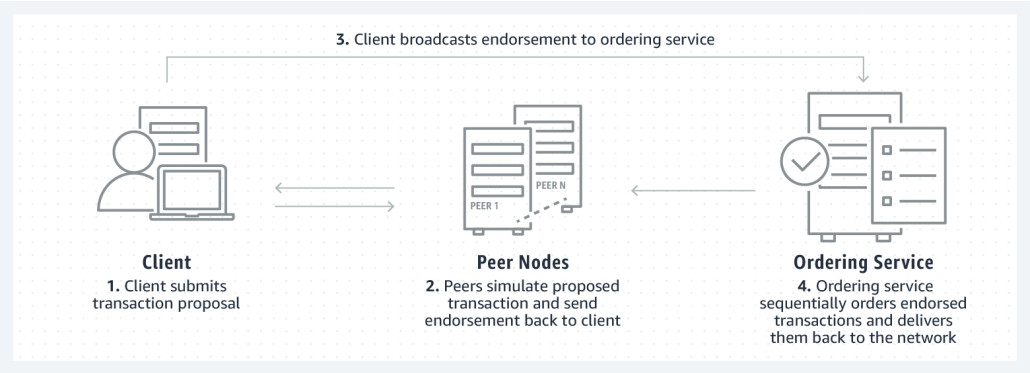


Figura 2.4: Fluxo de Transação no *Fabric* (Fonte: [14])

2. Os pares verificam a identidade e a autoridade do cliente, simulam a transação proposta e retornam uma assinatura de endosso ao cliente após uma correspondência bem-sucedida.

3. Posteriormente, o cliente coleta a quantidade necessária de endossos de acordo com a política de endosso e então encaminha a transação para o Serviço de Ordenação (*Ordering Service*).

4. O Serviço de Ordenação, por sua vez, valida os endossos em relação à política, organiza as transações aprovadas cronologicamente em blocos e distribui esses blocos para os nós pares dentro de cada organização que recebem os novos blocos, realizam uma validação final das transações e atualizam o estado do *ledger blockchain*.

O Serviço de Ordenação garante que cada transação dentro de um bloco tenha os endossos necessários antes de empacotá-la no livro-razão. Assim que o processo de validação for concluído, o novo bloco será integrado ao *ledger* e as transações confirmadas serão refletidas no *World State*. Se uma transação não tiver endossos suficientes, será rejeitada e não atualizará o *World State*, todavia é salva no *ledger blockchain* com a marcação de inválida.

2.3.3 Parâmetros

Apesar de não aprofundar as configurações, há alguns parâmetros relevantes para este trabalho, pois influenciam no tamanho da rede e blocos:

- *BatchTimeout*: define o tempo a uma taxa fixa que um bloco será inserido no *ledger blockchain* após a primeira transação.
- *BatchSize - MaxMessageCount*: designa a quantidade máxima de transações incluídas em um bloco.

- *BatchSize - AbsoluteMaxBytes*: estabelece o tamanho máximo que um bloco pode conter, somados os tamanhos das transações inseridas.

2.4 Computação em Nuvem

A computação em nuvem, em sua essência, proporciona um modelo flexível e altamente eficiente para a entrega de serviços de computação e recursos. Baseia-se no princípio de disponibilizar uma vasta gama de recursos de TI, como servidores, armazenamento, redes, aplicativos e serviços, de maneira acessível, sob demanda e por meio de uma rede onipresente [13].

Esse paradigma inovador permite que empresas, organizações e até mesmo indivíduos acessem esses recursos de forma conveniente, com agilidade e escalabilidade, sem a necessidade de investimentos pesados em infraestrutura física. Em vez disso, os recursos são fornecidos por provedores de serviços em nuvem e podem ser rapidamente provisionados ou liberados com um esforço mínimo de gerenciamento, ou interação com o provedor.

Segundo [8], a computação em nuvem possui um conjunto de características essenciais, e também pode ser classificada de acordo com modelos de serviços e implantação.

2.4.1 Características essenciais

Segundo o NIST (*National Institute of Standards and Technology*) [8], a computação em nuvem possui cinco características essenciais que são voltadas à experiência do uso e o estabelecimento do que um operador de computação em nuvem pode esperar dessa interação. São elas:

- Autoatendimento sob demanda: os usuários podem provisionar recursos de computação sem interação humana com o provedor de serviços.
- Amplo acesso à rede: os recursos são acessíveis pela rede através de mecanismos padrão, atendendo a diversas plataformas de clientes.
- Agrupamento de recursos: os provedores usam um modelo multilocatário (*multitenant*), atribuem dinamicamente recursos como armazenamento e processamento para atender a demanda do consumidor.
- Elasticidade: os recursos podem ser aumentados ou reduzidos rapidamente em resposta à demanda, apesar que muitas vezes pareçam ilimitados para os usuários.

- Serviço medido: os sistemas em nuvem monitoram e otimizam o uso de recursos, oferecem transparência por meio de medição para diversos serviços, como armazenamento e processamento.

2.4.2 Modelos de serviço

Os modelos de serviço referem-se ao nível gerencial que o utilizador da computação em nuvem pode ter acesso, sendo eles:

- *Software* como serviço (SaaS): os usuários podem acessar os aplicativos do provedor por meio da nuvem sem gerenciar a infraestrutura. Esses aplicativos são acessíveis por meio de vários dispositivos de clientes e os usuários normalmente definem configurações limitadas.
- Plataforma como serviço (PaaS): os consumidores podem implantar seus próprios aplicativos na nuvem usando linguagens de programação, bibliotecas e ferramentas suportadas. Embora não controlem a infraestrutura subjacente, eles têm controle sobre seus aplicativos implantados e algumas definições de configuração.
- Infraestrutura como serviço (IaaS): os usuários podem gerenciar e definir a quantidade de recursos desejados em computação, como processamento, armazenamento e redes, como também implantar e executar *software*, incluir sistemas operacionais e aplicativos, com controle sobre esses elementos, mas não sobre a infraestrutura de nuvem subjacente.

2.4.3 Modelos de implantação

Os modelos de implantação referem-se a quem adquire, implanta e gerencia o hardware da camada subjacente. Os modelos definidos pelo NIST são:

- Nuvem Privada: uma infraestrutura em nuvem provisionada exclusivamente para uma única organização que pode consistir em vários consumidores ou unidades de negócios. Pode pertencer, ser gerenciado e operado pela própria organização, por um terceiro ou por uma combinação de ambos, e pode estar localizado dentro ou fora das instalações da organização.
- Nuvem Pública: é disponibilizada para uso do público em geral. Pode pertencer, ser gerenciado e operado por uma empresa, instituição educacional, organização governamental ou uma combinação destes, e normalmente existe nas instalações do provedor de nuvem.

- Nuvem Híbrida: combinação de duas ou mais infraestruturas de nuvem distintas, que podem ser nuvens privadas ou públicas. Estas nuvens individuais permanecem entidades separadas, mas estão interligadas através de tecnologia padronizada ou proprietária.
- Nuvem Comunitária: neste modelo, a infraestrutura de nuvem é provisionada exclusivamente para uso de uma comunidade específica de consumidores de organizações que compartilham preocupações comuns. Essa infraestrutura pode ser de propriedade, gerenciamento e operação de uma ou mais das organizações da comunidade, de terceiros, ou uma combinação de ambos.

2.4.4 Amazon Web Services

A *Amazon Web Services* (AWS) é atualmente a plataforma de nuvem pública mais adotada globalmente e oferece mais de 200 serviços completos de datacenters em todo o mundo. A AWS é caracterizada por ser uma nuvem pública e disponibiliza um modelo de definição de preço com pagamento conforme o uso (*pay-as-you-go*) [13].

Conforme a análise de [3], dentro de sua avaliação dos serviços IaaS, PaaS, SaaS e entre outros, a AWS é Líder no Quadrante Mágico de tecnologia do setor, como mostrado na Figura 2.5.

Os principais serviços fornecidos pela AWS, e compõem a sua arquitetura básica (Figura 2.6) são: EC2, EBS e VPC.

EC2

O *Amazon Elastic Compute Cloud* (Amazon EC2) é um serviço IaaS de computação elástica sob demanda. As instâncias do EC2, dadas suas proporções, assimilam-se à uma máquina virtual. O *hardware* subjacente não é gerenciado pelo cliente, mas pela própria AWS.

EBS

O *Amazon Elastic Block Store* (Amazon EBS) é um serviço IaaS da AWS que fornece volumes de armazenamento em bloco para serem utilizados com instâncias do EC2, que se comportam como dispositivos de bloco não formatados e podem ser montados nas instâncias EC2.

Figura 2.5: Quadrante Mágico para infraestrutura de nuvem e serviços de plataforma (Fonte: [3])

VPC

A *Amazon Virtual Private Cloud* (Amazon VPC) permite agrupar as conexões de recursos da AWS em uma rede virtual personalizada, a qual se assemelha a uma rede convencional em um datacenter. Desta forma, é uma rede oferecida como IaaS.

Managed Blockchain

O *Amazon Managed Blockchain* (AMB) permite a criação de nodos conectados a *blockchains* públicas (como *Bitcoin* e *Ethereum*) e também a criação de *blockchains* privadas com *Hyperledger Fabric*. O AMB *Fabric* permite construir uma rede *blockchain* gerenciada sem servidor, isto é, fornecida em um modelo PaaS, em que o usuário final não precisa passar por tarefas de configuração complexas, mas possui um certo grau de geren-

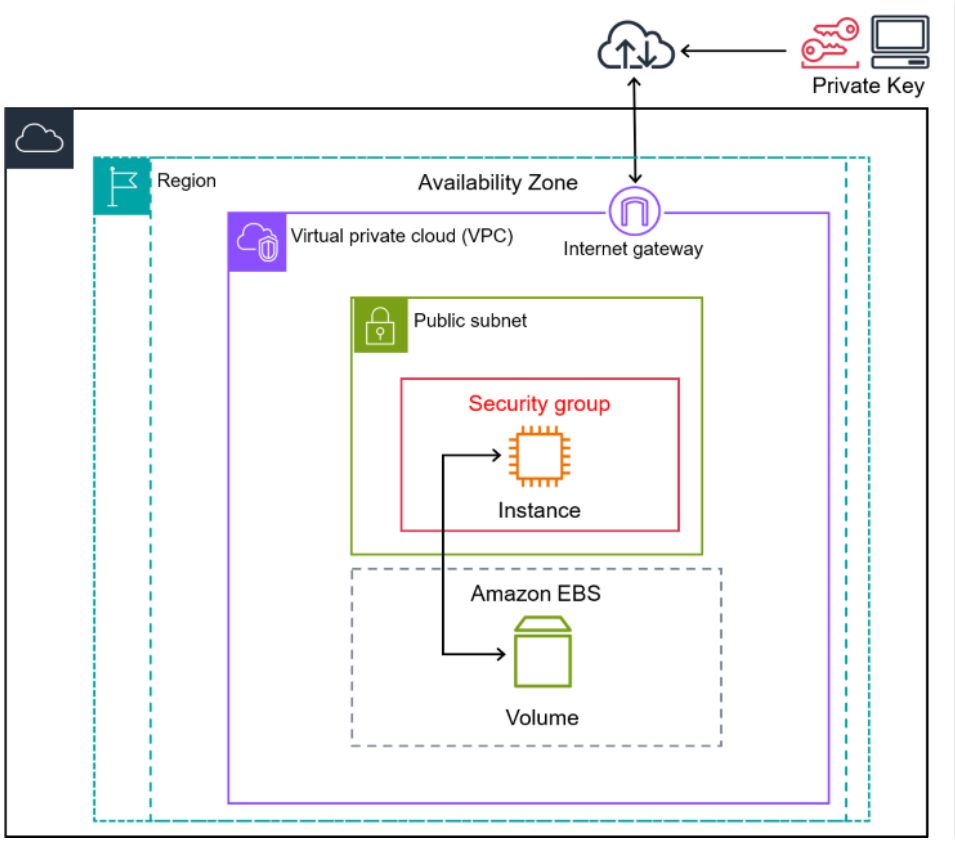


Figura 2.6: Arquitetura básica (Fonte: [1])

ciamento. Todavia, o AMB depende de recursos auxiliares, conforme mostra a Figura 2.7, como EC2 — para conexão dos clientes — e VPC — para conexão de redes.

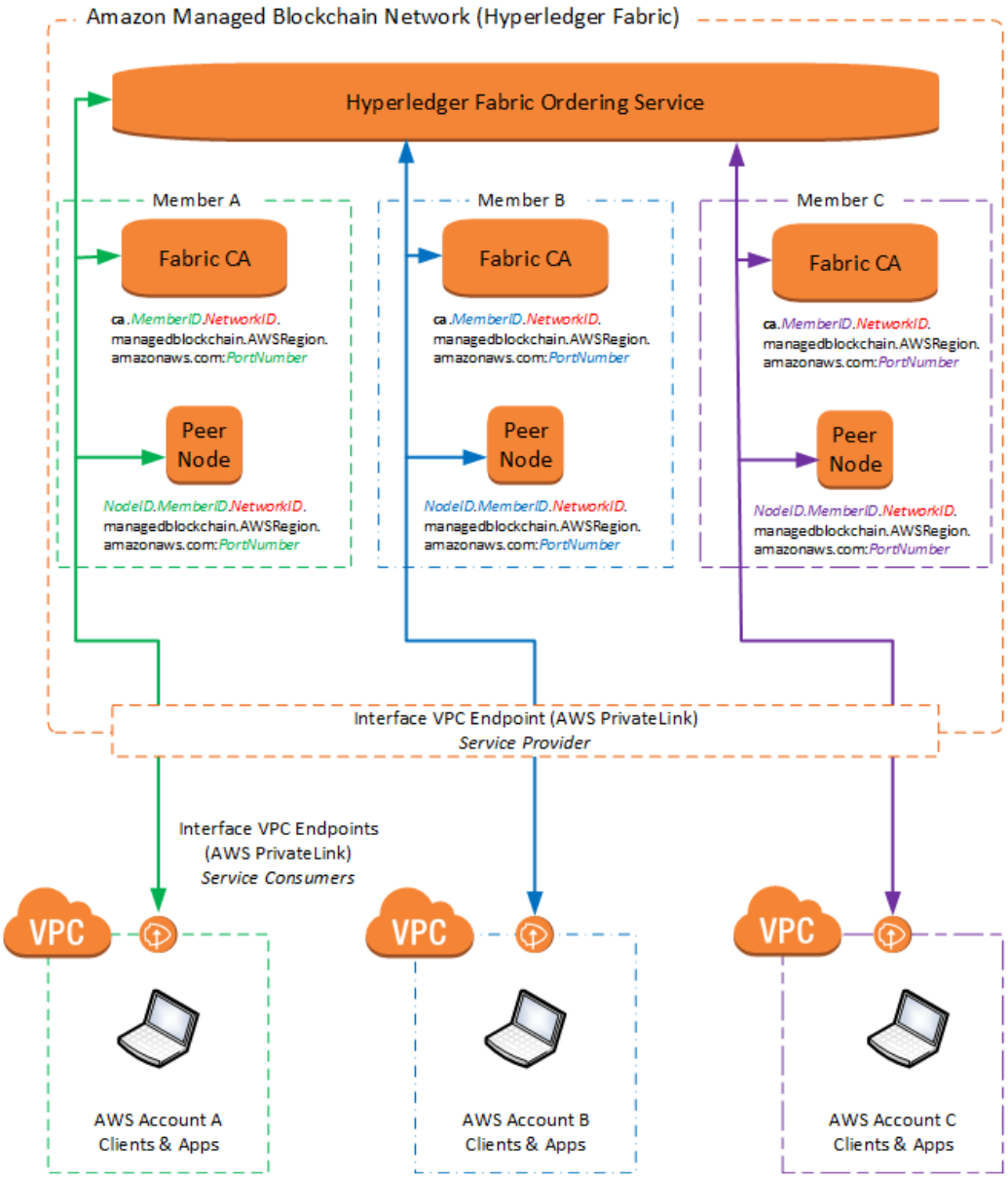


Figura 2.7: Arquitetura *Amazon Managed Blockchain — Hyperledger Fabric* — (Fonte: [14])

3. TRABALHOS RELACIONADOS

Este capítulo apresenta alguns trabalhos relacionados que propõem formas de contabilizar o custo para implantação de blockchains.

Em [5] os autores abordam a utilização de *blockchain* na cadeia de suprimentos marítimos. O artigo apresenta um modelo de custo monetário efetivo para indicar o ponto ótimo para substituição das transações físicas pelo modelo de DLT (*Distributed Ledger Technology*) e ajuda as partes interessadas a avaliar a implementação de uma aplicação de *blockchain*. Os autores desenvolvem um modelo de custo efetivo flexível para quantificar a implementação de uma aplicação *blockchain* e também focar nos benefícios potenciais trazidos pelas soluções baseadas em *blockchain*. O artigo descreve equações para mensurar o esforço operacional por operadores ao preencher informações em formulários, realizar autenticação e coleta de assinaturas de superiores e realizar o envio dos documentos e formulários. Contudo, o foco do modelo não faz uma projeção de custos conforme a evolução da rede *blockchain*, tampouco descreve a infraestrutura subjacente tratando os custos de TIC (Tecnologia da Informação e Comunicação) como um campo da equação que deve ser determinado como uma das entradas pelo utilizador e assume outros fatores como determinantes para sua análise.

Em [9] são apresentados modelos para análise da disponibilidade, a capacidade e custo de provisionamento de contêineres em infraestruturas de computação em nuvem públicas e privadas, sendo as públicas Amazon, Microsoft e Google. O artigo apresenta a execução de aplicativos distribuídos de *blockchain* baseados na plataforma *blockchain Ethereum* e um caso de estudo que compara as despesas necessárias em infraestruturas de nuvem públicas e privadas. O artigo realiza a modelagem para verificar o número de contêineres necessários, isto é, a capacidade para garantir o alvo de disponibilidade da rede em caso de falhas. Porém, nos ensaios e informações apresentadas, as estimativas são para apenas 5 anos, e consideram cenários sem estimativas de transações suportadas e assumem uso constante da aplicação. Além disso, o artigo está voltado para *blockchain Ethereum* que possui um comportamento diferente da *Hyperledger Fabric.*

O estudo realizado em [10] apresenta uma comparação para os custos de execução de processos de negócios em *blockchain Ethereum* e serviços em nuvem, comparando a execução de contratos inteligentes com os processos de fluxos do Amazon SWF. O estudo verifica as capacidades de vazão de cada abordagem, porém no caso da *blockchain* gera uma carga maior de processamento necessário em relação ao processo do *Workflow*. Os resultados apontam que há uma diferença de cerca de 2x mais custos da *blockchain Ethereum* executada em nuvem em comparação com o serviço de nuvem *Amazon Simple Workflow Service* (SWF).

Outros trabalhos que possuem relevância e agregam na compreensão dos comportamentos das *blockchains*, são publicações relacionadas a aspectos de desempenho e escalabilidade. Podem ser citados as contribuições de [4], [16] e [17] que realizaram análises de desempenho da plataforma *Hyperledger Fabric*.

No trabalho [4], os autores utilizaram o *Hyperledger Caliper* e realizaram diversos experimentos em diferentes cenários para avaliar latência e taxa de transferência em várias configurações. Alguns dos experimentos foram para estimar o gargalo e vazão para diferentes cargas em uma instância do Amazon EC2. Também, indicaram como o tamanho do payload de uma transação afeta na vazão da rede. O trabalho verifcou-se como que as políticas de endosso geram enfileiramento e carga nos peers de validação. Os testes foram executados na versão 1.1 do *Fabric* e não mencionam os custos para executar os experimentos.

Em [16], os autores exploraram a otimização do cache e as políticas de endosso, forneceram diretrizes de configuração da rede e identificaram gargalos de desempenho. O trabalho verifica que as políticas de endosso, modeladas de forma lógica *AND* e *OR*, a vazão em *OR* é significativamente maior, ou seja, com menor carga de CPU. O trabalho também valida como a criação de mais de um Canal (*Channel*) pode melhorar a vazão e diminuir a latência. Há resultados validados para servidores executados localmente com 2 e 4vCPU.

O trabalho apresentado em [17] caracteriza o desempenho de fases do ciclo de vida de transações no *Hyperledger Fabric*, diferentes abordagens de ordenação, experimentos com políticas de endosso e configurações de *batches*. Foram executados testes de validação de desempenho em cada etapa da transação, como endosso, ordenação e confirmação. Verificaram na fase de endosso como a política de endosso impacta no desempenho. Na parte de ordenação, validaram as implementações Solo, Kafka e Raft e concluíram que a diferença não é significativa. Na parte de confirmação, verificaram como os parâmetros de *batches* influenciam na vazão e confirmação dos blocos. Os testes foram executados localmente em processadores com até 8 vCPU.

Por fim, o trabalho de [11] propõe um *framework* para melhorar o uso da *blockchains* em dispositivos IoT. O *framework* denominado *FabMAN* propõe em sua abordagem alterar configurações de *batches*, otimizando a confirmação de blocos conforme taxa de transações. Em relação ao armazenamento, é proposto uma limpeza da blockchain devido ao seu constante crescimento no dispositivo IoT, confirmando os dados nos demais *peers* com todas as transações e gerando um hash do último bloco e o inserindo esse valor como um novo bloco gênese, iniciando uma nova *blockchain* sem os dados anteriores.

Através da análise dos trabalhos relacionados, verifica-se que nenhum apresenta uma solução para projetar o crescimento dos recursos conforme a demanda e parâmetros da *blockchain*. Além disso, apesar de alguns projetos utilizar a nuvem, nem todos possuem modularidade na projeção dos custos da *blockchain* conforme o seu crescimento.

Artigo	Contribuição	Analise de recursos	Predição de crescimento	Análise de custos
[5]	- Análise de custo/beneficio de substituição operacional por registro distribuído	-	-	analisa o ponto ótimo de custos para implementação de uma *blockchain*.
[9]	- Modelagem para verificar a quantidade de contêineres para garantir a disponibilidade em rede *Ethereum*; - Predição de custos em múltiplas nuvens públicas e privada durante 5 anos;	Containers	Considera o crescimento constante	possui análise dos custos para alocação de contêineres em 3 provedores de nuvem publica distintos
[10]	Testes de vazão e latência de execução de *blockchain* em nuvem em contexto de execução de processos em etapas;	EC2 e SWF	-	Verificou um alto consumo de créditos de CPU e custos maiores com a *blockchain Ethereum*.
[4]	- Testes experimentais de desempenho em instância do Amazon; - Validação de políticas de endosso; - Testes de vazão e latência;	EC2	Somente crescimento da vazão sem alterar o modelo de servidor;	-
[16]	- Testes experimentais de desempenho com diferentes políticas de endosso e diferentes quantidades de canais; - Validação da vazão conforme configuração;	Local	Somente crescimento da vazão alterando a quantidade de canais e endosso; Não é alterado o modelo de servidor;	-
[17]	- Testes experimentais de desempenho com diferentes políticas de endosso; - Testes em difernetes clusters de ordenação (*Solo, Kafka e Raft*); - Testes de diferentes configurações de *batches*	Local	Somente crescimento da vazão	-
[11]	- *Framework* para otimização da configuração de desempenho em tempo de execução; -Reorganização do armazenamento;	armazenamento e IoT	Crescimento do armazenamento; otimização de desempenho;	-

Tabela 3.1: Tabela de contribuições dos artigos analisados

4. OBJETIVOS DO PROJETO E DESENVOLVIMENTO

A introdução do uso de *blockchains* está ganhando presença nos mais variados segmentos [5]. Além disso, o uso de computação em nuvem pode auxiliar na escalabilidade de novos nodos da rede, mas, em contrapartida, o custo monetário é ligeiramente maior ao usar serviços de nuvem [10].

Em todos os casos, verifica-se que há uma lacuna quando se refere a *blockchain* e computação em nuvem em conjunto, em especial, no que tange a projeção de crescimento e do custo monetário para manutenibilidade — manter a infraestrutura, armazenamento e rede existente — e também para a operação — suportar novas transações e crescimento.

4.1 Formulação do problema

A adoção do uso de *blockchain* em ambiente produtivo de organizações requer planejamento prévio para sua implantação. A partir da sua implantação, a organização assume o compromisso com os dados inseridos na *blockchain* e também o funcionamento da mesma para continuidade do negócio com a troca de informação entre os participantes. Com isso, torna-se relevante uma análise para médio e longo prazo dos recursos necessários para executar e manter a *blockchain*. Com as estimativas de recursos, a *blockchain* pode ser executada em nuvem, ao trocar despesas de capital (CAPEX), maior investimento inicial, por despesas operacionais (OPEX) que crescem sob demanda.

4.2 Objetivos gerais e específicos

O objetivo geral deste projeto é realizar a projeção do crescimento dos recursos necessários ao longo de um período determinado pelo utilizador. A partir da estimativa de recursos, são aplicados os cálculos de custos para implantação de uma aplicação *blockchain* em nuvem, a partir de um conjunto de informações de entrada.

Como objetivos específicos, a implementação baseia-se no comportamento da *Blockchain Hyperledger Fabric* que solicita ao utilizador a partir de configurações reais da rede e de dados que devem ser estimados com base nos *chaincodes* executados. A partir das configurações, são executados cálculos que estimam a quantidade de recursos e posteriormente, são calculados os custos em nuvem, com base em preços praticados pelos provedores, como Amazon Web Services e concorrentes. A formulação dos códigos e

processamento de informações foi realizada em um Jupyter Notebook [1], que permite incluir textos e orientações juntamente com o código aberto.

4.3 Metodologia

A metodologia deste trabalho utilizou-se de artigos científicos, casos de uso que contém informações já validadas como experimentos de desempenho e documentações oficiais das ferramentas. A construção em Jupyter Notebook permite a visualização do código aberto e validação. As saídas são estimativas de possíveis cenários de implantação e diferentes cenários podem ser facilmente construídos.

Dessa forma, a predição não está atrelada necessariamente à co-existência da *blockchain* real em execução. Contudo, requer ajustes de entrada compatíveis com a rede desejada. Assim, a implementação torna viável a predição sem alocar recursos reais para as estimativas.

4.4 Modelo proposto

Em uma *Blockchain Hyperledger Fabric*, diversas topologias e arquiteturas podem ser implementadas. Para este modelo, considera-se que a organização seja responsável por 1 *peer* e 1 nó do *cluster* de ordenação. Conforme a Figura 4.1, a estimativa é predita para o contexto da organização A e do ponto de vista de um canal (*Channel*). Os recursos analisados são: armazenamento, computação (*Peer* e *OrderingService*) e link de dados (*PrivateLink*):

A modelagem do código em Jupyter Notebook permite criar um arquivo com múltiplas seções de textos e códigos, para facilitar as entradas do usuário, o processamento dos dados, visualização de dados e estimativas geradas. A apresentação é de forma interativa, assim, os resultados são gerados e mostrados ao longo do processamento e não somente no final.

O projeto é dividido em duas grandes seções: predição dos recursos e custos dos recursos provisionados. No entanto, para fins de organização dos próximos tópicos, serão abordados inicialmente questões de parametrização, e em seguida tópicos agrupados por recurso, demanda e custo, como: armazenamento, CPU e rede.

[1] https://jupyter.org/

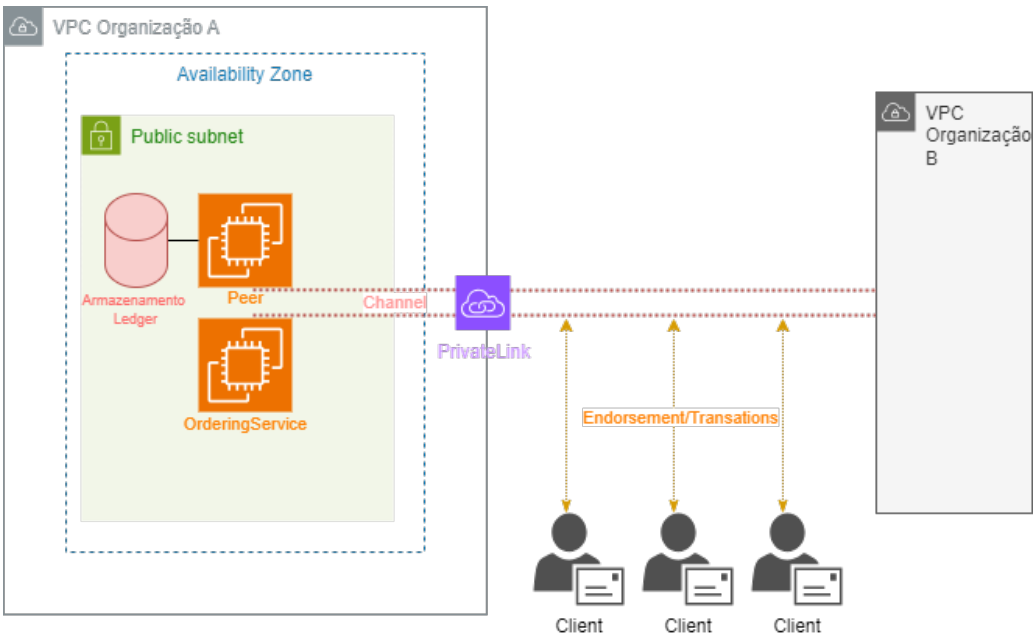


Figura 4.1: Arquitetura referência da análise proposta (Fonte: do autor)

4.4.1 Parametrizações

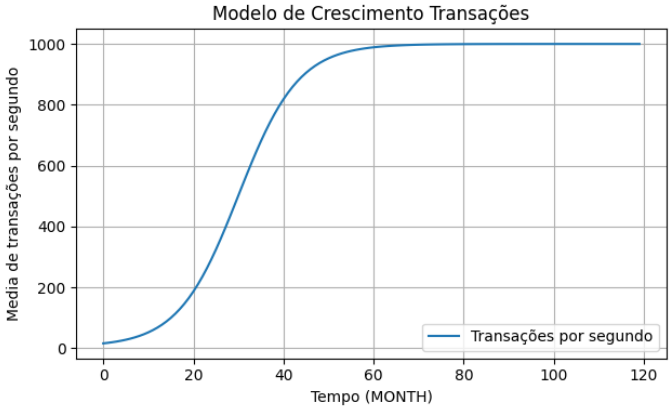
Esta seção trata a modelagem do período desejado, os preços e os custos para cada unidade de nuvem e as configurações da rede.

Definição do período analisado e taxa de transações

A primeira entrada para as estimativas requer que seja definido um período “A” e um vetor com “A” posições, com cada posição indicando a taxa de transações por segundo (TPS) estimada para o ponto analisado. O vetor é uma representação para $f(x) = y$, para cada momento x esperara-se uma taxa y de transações. O período x pode ser em dias, semanas, meses e anos, ou seja, um período pode ser definido como 120 meses, ou 12 anos, no entanto, quanto maior a granularidade de espaço, menor o refinamento das predições. O crescimento da taxa de TPS é uma métrica que deve ser avaliada no contexto do utilizador, com possíveis cenários de carga esperada.

Como exemplo, será apresentado um cenário com uma curva logística, mas outros cenários, curvas e entradas podem ser definidas pelo usuário no vetor de entrada. Essa curva, em forma de "S", é frequentemente utilizada para modelar diversos processos de crescimento que apresentam um crescimento inicial lento, seguido por uma aceleração e, por fim, uma estabilização.

A Figura 4.2 descreve o total de transações projetado pela curva logística com os seguintes parâmetros:

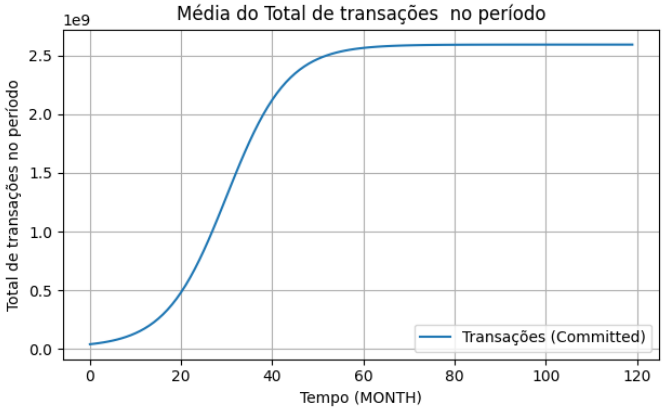


	Tempo (MONTH)	Transações por segundo
10	10	52.19
20	20	186.51
30	30	502.50
40	40	818.49
50	50	952.81
60	60	989.07
70	70	997.54
80	80	999.45
90	90	999.88
100	100	999.97
110	110	999.99

Figura 4.2: Transações por segundo com tabela (Fonte: do autor)

- Período: A = 120
- Transações iniciais: P0 = 5
- Valor máximo transações: K = 1000
- Taxa de crescimento: r = 0.15
- Ponto de inflexão: t0 = 30
- Pontos $f(x)$: t = [0...A-1]

Com esses valores de TPS, pode-se analisar a soma das transações que ocorrem em um período dado o tipo de período escolhido (dia, semana, mês ou ano). Por exemplo, se há um valor de transações por segundo igual a 5 e o período escolhido é um mês (considerando 30 dias e 2592000 segundos), a estimativa aproximada do total cumulativo de transações desse ponto x é de 5 * 2592000 = 12.960.000 (total de 12 milhões, quinhentas e noventa e duas mil transações). O gráfico da Figura 4.3 agrega as informações conforme o TPS anterior da Figura 4.2 (transações por segundo com tabela).



	Tempo (MONTH)	Transações (Committed)
10	10	135,273,223.96
20	20	483,442,722.92
30	30	1,302,480,000.00
40	40	2,121,517,277.08
50	50	2,469,686,776.04
60	60	2,563,664,235.48
70	70	2,585,623,005.97
80	80	2,590,574,361.78
90	90	2,591,681,760.45
100	100	2,591,928,984.35
110	110	2,591,984,153.93

Figura 4.3: Agregado de transações no período com tabela (Fonte: do autor)

Parâmetros de preços da nuvem

Os parâmetros de preços da nuvem são variáveis utilizadas para estimar o custo total dos recursos e que podem ser ajustadas conforme o valor atualizado praticado pelas provedoras de nuvem, permitindo também o uso da ferramenta para diferentes provedoras de nuvem que praticam a cobrança com métricas de consumo semelhantemente.

```
CLOUD_PRICING = { #USD
    # sa-east-1 #AWS São Paulo
    "Storage": {
      "Pricing": 0.19
    },
    "NetworkThroughput": {
      "PricingDataTransfer": 0.01,
      "PricingEndpointHour": 0.01
    },
    "VirtualMachine": {
       "FilePricing": None #None requires insert manual values.
       "InstancesTypes": [
      # E.g:
        {"Name": " c6a.large","vCPUs": 2,  "Pricing": 0.1179},
        {"Name": " c6a.xlarge","vCPUs": 4,  "Pricing": 0.2358},
        {"Name": " c6a.2xlarge","vCPUs": 8,  "Pricing": 0.4716},
        {"Name": " c6a.4xlarge","vCPUs": 16,  "Pricing": 0.9432},
        {"Name": " c6a.8xlarge","vCPUs": 32,  "Pricing": 1.8864},
       ]

    },
    "Additional": 0 # E.g: taxes or discounts
```

Os paramêtros lidos em *CLOUD_PRICING* são específicos de cada recurso descrito para as seguintes unidades de consumo:

- Storage - Pricing: define o preço cobrado por *gigabyte* armazenado;
- NetworkThroughput - PricingDataTransfer: define o preço cobrado por *gigabyte* trafegado;
- NetworkThroughput - PricingEndpointHour: define o preço cobrado por hora pela existência do recurso de conectividade;

Para os preços de instâncias "*VirtualMachine*", é possível inserir modelos de duas formas:

- FilePricing: um arquivo do tipo *csv* com metadados de colunas com os campos *Name*, *vCPU* e *Pricing* para facilitar a entrada de modelos de instâncias.
- InstancesTypes: uma lista de objetos com chave-valor conforme demonstrado no exemplo.

As entradas para recursos de *VirtualMachines* são disjuntivas, ou seja, a entrada de uma descarta os valores da outra, assim, deve-se utilizar somente um dos casos.

Parâmetros de configuração

Os parâmetros de configuração são variáveis que podem ser manipuladas pelo utilizador e que se referem tanto às configurações da *blockchain* (conforme configuração real), quanto aquelas resultantes dos *Smart Contracts* como tamanhos de transação. Essas configurações são lidas a partir da seguinte estrutura de dados JSON:

```
CONF = {
  "BlockchainParameters": {
        "GenesisBlockSize": 0,
        "BatchSize": {
          "AbsoluteMaxBytes": 10485760,   # Bytes
          "MaxMessageCount": 100000
        },
        "BatchTimeout": 600, #Seconds
      "EndorsementPolicy": 1/2,
  },

  "BlockHeaders": { # Bytes
       "MetadataSize": 64,
       "HeaderSize": 68
  },

  "TransactionFields": {#Bytes
       "R4": 512,
       "E4": 1024,
       "H4": 256,
       "P4": 256,
       "S4": 256
  },
}
```

Os valores que constam em *BlockchainParameters* referem-se às configurações da *blockchain* e influenciam no comportamento da rede, principalmente na questão de vazão. São eles:

- BatchSize - AbsoluteMaxBytes: define o máximo de *bytes* que devem podem ser colocados no bloco.
- BatchSize - MaxMessageCount: apresenta o máximo de transações que podem ser inseridas em um bloco.
- BatchTimeout: designa o tempo limite que o bloco pode aguardar para ser inserido na cadeia.
- EndorsementPolicy: define a política de endossso, ou seja, quantos *peers* sobre o total de *peers* da rede devem assinar a transação.

Os demais parâmetros referem-se ao espaço ocupado pelos campos de um bloco e da transação. Alguns valores podem ser aproximados pela natureza do tipo, como o *HeaderSize*, composto pelo SHA256 (32 *bytes*) do bloco anterior, o SHA256 (32 *bytes*) do bloco atual e um inteiro (4 *bytes*) que identifica o bloco. O somatório resulta em 68 *bytes*. No entanto, os Metadados do bloco (*MetadataSize*) podem ter tamanho variado e devem ser analisados conforme o *chaincode* executado.

Da mesma forma, os campos de *TransactionFields* são variados conforme o *chaincode* executado, mas conforme [6], em média, uma transação não deve ser menor que 1KB e cada endosso *E4* gera um aumento de 1KB. Assim, os valores de R4, H4, P4, S4 foram escolhidos para atender a documentação, porém são parametrizáveis pelo usuário conforme seu *chaincode*.

4.4.2 Armazenamento

Nesta seção, será descrita a predição do crescimento do armazenamento requerido com base nos parâmetros de entrada de configuração e transações e, desses resultados, calculam-se os custos dados os valores de entrada apresentados anteriormente em parâmetros de preços.

Estimativas de geração de blocos

A estimativa de blocos é gerada através do total de transações por período, descritos na Figura 4.3, a taxa de transações por segundos, Figura 4.2, e as configurações de

batches. A expressão utilizada verifica, dada a configuração *MaxMessageCount*, *AbsoluteMaxBytes* e *BatchTimeout*, o tempo provável para um bloco ser encadeado pela quantidade de mensagens a partir da taxa de transações por segundo no instante *i*.

A Equação 4.1 realiza o cálculo para verificar o provável tempo de encadeamento do bloco com base na taxa de transações por segundo (TPS) a partir da configuração de máximo de mensagens (*MaxMessageCount*):

$$time\_to\_batch\_messages_i = \frac{MaxMessageCount}{avg\_tps_i} \tag{4.1}$$

Para exemplificar, dado o *MaxMessageCount* com valor de 100000 e supondo que a taxa TPS seja igual a 500, dividindo o valor de 100.000 sobre 500, o valor resultante será 200, ou seja, tempo de 200 segundos para preenchimento do bloco caso atingir a condição *MaxMessageCount*.

As Equações 4.2 e 4.4 realizam os cálculos para verificar o provável tempo de encadeamento do bloco com base na taxa de transações por segundo (TPS) a partir da configuração de máximo de *bytes* no bloco (*AbsoluteMaxBytes*). Na Equação 4.2 é calculado o tamanho máximo de *bytes* aceito dividindo pelo tamanho da transação, onde *TRANSACTION_SIZE* é a soma de todos os campos da configuração *TransactionFields*. Isso dará a estimativa de quantas transações podem ser colocadas no bloco a partir do seu tamanho.

$$aggregate\_total\_transactions\_if\_batch\_max\_bytes_i = \frac{AbsoluteMaxBytes}{TRANSACTION\_SIZE} \tag{4.2}$$

Em seguida, na Equação 4.4, calcula-se o tempo que a taxa de TPS levaria para atingir o fechamento do bloco, com base no cálculo da quantidade de transações anterior.

$$time\_to\_batch\_bytes_i = \frac{aggregate\_total\_transactions\_if\_batch\_max\_bytes_i}{avg\_tps_i} \tag{4.3}$$

Para exemplificar os cálculos, considere *AbsoluteMaxBytes* com valor de 10.485.760 bytes, e a soma dos campos da transação *TRANSACTION_SIZE* com total de 2.304 *bytes*, e assim o total de transações em um bloco com essas configurações será de aproximadamente 4.551 transações. Ao considerar uma taxa de transações por segundo no instante *i* de 500, a divisão de 4.551 sobre 500 resulta em cerca de 9 segundos para preenchimento do bloco.

Calculados esses valores, verifica-se qual condição é atendida para preenchimento do bloco, isto é, quem atinge o gatilho do menor tempo. Se os resultados dos segundos calculados ultrapassarem o *BatchTimeout*, esse será o gatilho para encadeamento do bloco.

Verificado o tempo estimado para os *batches*, a Equação 4.4 calcula o total de blocos para o instante *i* com base agregado de transações no período, descrito na Figura 4.3.

$$total\_blocks = \frac{total\_transacoes\_por\_periodo}{time\_to\_batch} \tag{4.4}$$

No exemplo descrito, com *BatchTimeout* de 600 segundos, a condição AbsoluteMaxBytes de 9 segundos seria o gatilho do lote por ser a menor. Ao considerar a estimativa de meses, um total de transações no período *i* de 1.314.900.000 — resultado de 500 TPS vezes 2.629.800 segundos — são gerados aproximadamente 146.100.000 blocos.

Estimativas de armazenamento

Com as estimativas de quantidades de blocos, o armazenamento é calculado agregando as informações dos blocos e transações. A Equação 4.5 representa o total de armazenamento em *bytes*, onde *t_transactions_period* é o total de transações, *t_blocks* é o total de blocos e *t_storage* é o volume do armazenamento acrescido, todos referentes ao instante *i* período:

$$t\_storage_i = (block\_headers\_size \times t\_blocks_i) + (transaction\_size \times t\_transactions\_period_i) \tag{4.5}$$

Nessa abordagem, é calculado o armazenamento ocupado em um período *i* de forma agregada e não por blocos individuais. No entanto, esse valor é, na verdade, o armazenamento parcial de cada instante *i*, representado pelo vetor de demanda de armazenamento $t\_storage = [t\_storage_1, t\_storage_2, ..., t\_storage_i]$. Para ter o total de armazenamento ocupado, é necessário de realizar uma soma cumulativa. O armazenamento total até o instante *i* é a soma cumulativa, definida como um vetor ***storage_cum***:

$$storage\_cum = [storage\_cum_1, storage\_cum_2, ..., storage\_cum_i] \tag{4.6}$$

Onde cada elemento $storage\_cum_j$ é dado por:

$$storage\_cum_j = \sum_{k=1}^{j} t\_storage_k \tag{4.7}$$

O valor máximo de armazenamento, então, é a soma de todo o armazenamento cumulativo, logo, estará na última posição resultante de *storage_cum*. A abordagem de soma cumulativa de $storage\_cum_j$ é uma forma de verificar e acompanhar o crescimento e não apenas o valor de armazenamento final.

Custos de Armazenamento

Com o levantamento do total de crescimento do armazenamento, pode-se estimar o custo para o armazenamento crescente com base nas informações fornecidas anteriormente na entrada de *['Storage']['Pricing']* que deve fornecer o valor praticado pelo provedor de nuvem, geralmente em custo de *gigabytes* por mês. Mas, além disso, é necessário atentar-se à forma de cobrança. A provedora AWS explica em sua calculadora oficial [12] que o tempo considerado para meses é de 730 horas.

Esse valor em horas diferencia-se do comum de calcular 24 horas vezes 30 dias que resulta em 720 horas. Essa diferença é justificada para amenizar a diferença do total de dias durante diferentes meses. Com isso, para estimativas diferente de meses é necessária uma normalização dos valores de entrada, especialmente para dias e semanas.

O provedor considera que o total de semanas é de 4,34 semanas por mês. Então, o custo proporcional para uma semana em relação a 730 horas é, na verdade, o custo de uma semana em relação ao total de semanas (4,34). Por exemplo, se o custo mensal para cada gigabyte de armazenamento for de $0,45, o custo por semana proporcional é de aproximadamente $0,10.

4.4.3 Computação

Nesta seção é descrita a predição da demanda de CPU necessária com base nos parâmetros de entrada de desempenho e transações por segundo. Dos resultados será calculado o custo dado dos valores de entrada descritos anteriormente em parâmetros de preços.

Estimativas de recursos de computação (CPU)

A estimativa de computação é realizada por meio de valores de desempenho (informados pelo utilizador) para calcular o desempenho proporcional dado a taxa de transações por segundo. As entradas são computadas para o desempenho do *Peer* e do nodo de *OrderingService*:

```
1 vCPU_BASE_PEER     =  2
2 vCPU_BASE_PEER_TPS = 200
3 vCPU_BASE_OSN      =  1
4 vCPU_BASE_TPS_OSN = 150
```

As configurações de *vCPU_BASE_PEER* e *vCPU_BASE_OSN* referem-se à quantidade de CPU capaz de processar o total de transações por segundos dadas por, respec-

tivamente, *vCPU_BASE_PEER_TPS* e *vCPU_BASE_TPS_OSN*. A quantidade de transações que o *peer* processa e o *OrderingService* são diferentes e do *peer*, além do TPS, deve ser considerada a política de endosso.

O *OrderingService* processa a quantidade de TPS que estão ocorrendo no canal. O *peer* deve processar todas as transações, como também o endosso que ocorre separadamente da validação (*commit*) da transação. Por exemplo, dada uma taxa de 100 TPS e a política do endosso é dada por $OR(A, B)$, a modelagem de entrada traduz isso como $1/2$ de endossos, onde 2 é o total de *peers* e 1 o endosso necessário. Assim, o *peer* processa o total de transações da rede confirmadas (100) mais as transações com endosso (50), processando 150 transações.

A modelagem para o crescimento de CPU, tanto do *peer*, quanto do nodo de ordenação (*OrderingService*), tem seu cálculo separado conforme a sua taxa de transações a serem processadas que é dada pelo trecho de pseudocódigo apresentado no Algoritmo 4.1.

Algoritmo 4.1: Cálculo de vCPU para taxa de transações

Data: Taxa de transações `transacoes_processadas`, `constante vCPU_BASE`, `constante vCPU_BASE_TPS`
Result: Lista agregada de CPU necessária
Entrada: Lista de taxa de transações `transacoes_processadas`
Saída : Lista `cpu_requerida`

```
for cada taxa em transacoes_processadas do
    cpu_requerida ← taxa / vCPU_BASE_TPS
    if cpu_requerida > 1 then
        cpu_requerida ← vCPU_BASE + (vCPU_BASE × int(cpu_requerida))
    end
end
```

Assim, a quantidade de CPU é acrescida conforme a capacidade de processamento. O ponto ótimo de processamento é igual a 1. Quando a CPU requerida passa do ponto ótimo, indica que é necessário acrescer a quantidade de CPU para suportar a taxa. A capacidade de CPU sempre deve ser maior que a taxa TPS para evitar gargalos.

Por exemplo, dada uma taxa de 500 TPS em um instante *i* do período, configuração de 2vCPU com capacidade de processamento total de 200 TPS, há $500/200 = 2.5$, então, do valor 2.5 obtém-se o valor inteiro 2, seguindo o algoritmo, $cpu\_requerida = 2vCPU + (2vCPU * 2)$, então $cpu\_requerida = 6$ possui uma capacidade de processamento de 600 transações por segundo, ligeiramente maior que 500, evitando falta de recursos. O crescimento cresce ao passo das unidades de total de CPU e desempenho base estimados na entrada do utilizador. Pode-se visualizar este exemplo na Figura 4.4, onde a quantidade de TPS é próxima a 500 no instante 30 do período e a capacidade de TPS da CPU que se projeta é igual ou maior do que a demanda.

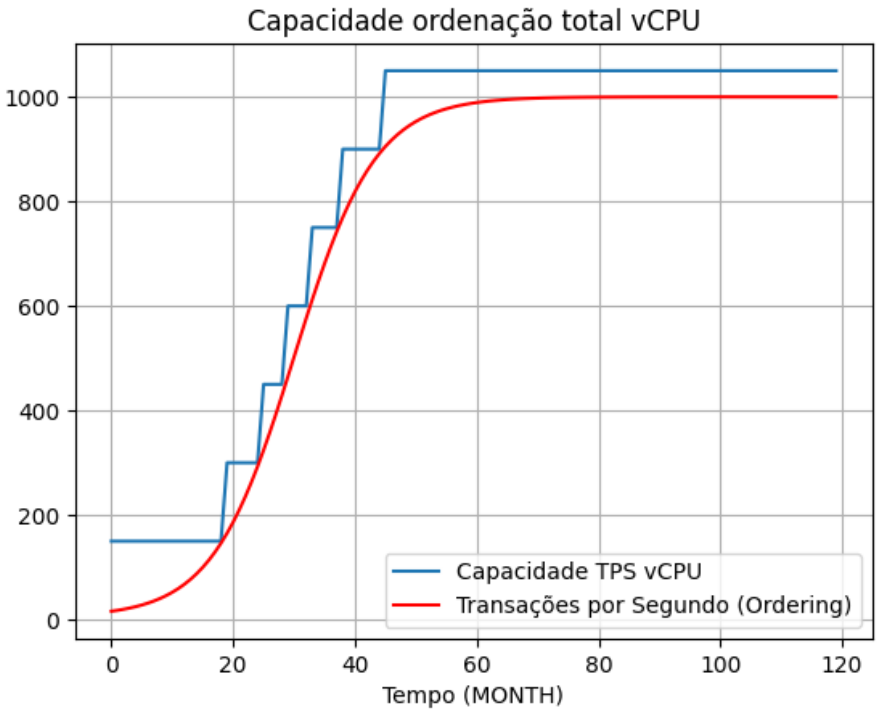


Figura 4.4: Taxa TPS do Canal *versus* CPU projetada (Fonte: do autor)

Alocação dos recursos em nuvem

Com a quantidade de CPU requerida conforme a taxa de crescimento, é necessário verificar nos modelos de computação qual tipo de instância possui o número igual ou maior de CPUs. Isso porque, dentre os modelos, eventualmente, os modelos disponíveis não sejam exatamente com a quantidade de CPU desejada. Para isso, verifica-se a quantidade de CPU requerida da etapa anterior e é realizada uma busca dentre os modelos com CPU igual ou maior informados pelo utilizador no campo de precificação e modelos descritos anteriormente na Seção 4.4.1 em "Parâmetros de preços da nuvem".

O pseudocódigo apresentado no Algoritmo 4.2 descreve a busca do modelo adequado.

Algoritmo 4.2: Provisionamento de Tipos de Instância

Data:
Lista de CPU requerida: `cpu_requerida`
Dicionário com os tipos de instância e suas vCPUs: `instancias`
Result:
Lista de tipos de instância provisionados: `provisioned_vcpu`

```
modelos_provisionados ← array vazio
for cada vcpu em cpu_requerida do
    if vcpu existe como chave em instancias then
        inst ← valor correspondente à chave vcpu em instancias
        modelos_provisionados ← adicionar inst a modelos_provisionados
    end
    else
        maior_inst ← valor correspondente à maior chave em instancias
        modelos_provisionados ← adicionar maior_inst a modelos_provisionados
    end
end
return modelos_provisionados
```

Assim, o algoritmo de busca retorna um vetor com informações, pois contém para cada instante *i* do período, o modelo que deve ser utilizado. Esses cálculos aplicam-se de forma igual para o provisionamento de instância tanto para o *peer* quanto para o nodo de ordenação.

Os provedores de nuvem fornecem, geralmente, tipos e modelos de instância em crescimento exponencial de base dois, com limite superior para cada categoria [1]. Assim sendo, o provisionamento adequado deve ser igual ou maior que a capacidade demandada pelo Canal, o que em alguns momentos gera uma alocação significativamente maior que o necessário, todavia, não há modelos intermediários e sem alternativas. A Figura 4.5 permite visualizar esta ocorrência.

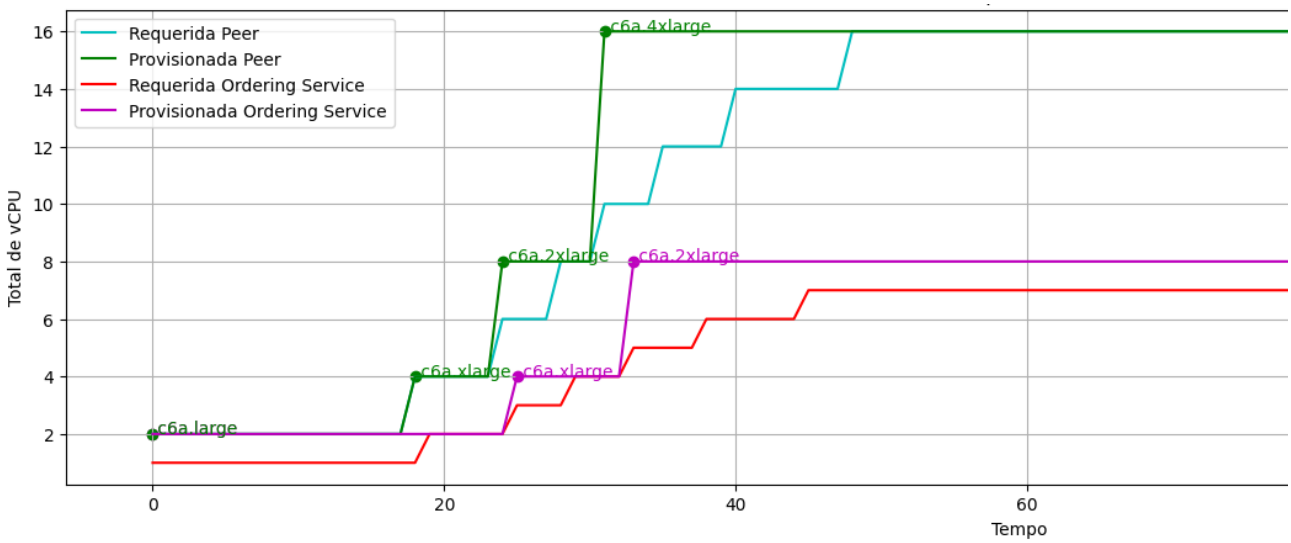


Figura 4.5: Instâncias provisionadas e demanda de CPU requerida (Fonte: do autor)

Custos de instâncias para computação

Dado os modelos adequados para cada instante *i* do período, o tipo de instância possui um custo monetário proporcional ao seu tamanho. No entanto, o custo deve ser confirmado nas entradas do utilizador para cada modelo de instância inserido. Assim como calculamos anteriormente em “custos de armazenamento”, os custos de computação possuem particularidades que influenciam nos cálculos. A cobrança pela computação utilizada é medida em horas, logo, para um dia, a precificação é de 24 horas. Todavia, as estimativas para semanas, meses ou anos devem ser ajustadas aos valores de referência de 730 horas/mês [12]. O valor de horas para cada tipo de período é dado pela Tabela 4.1.

Formato	Conversão	Total de Horas
Dia	-	24
Semana	730 / 4.3452	168
Mês	730	730
Anos	730 * 12	8760

Tabela 4.1: Conversão dos períodos em horas

Assim, o custo para cada instante *i* do período é realizado a partir do custo/hora multiplicado pelo total de horas conforme o formato período analisado.

4.4.4 Rede

Nesta seção, será descrito como os valores do custo de rede são agregados, conforme as suposições da calculadora fornecidas pelas provedoras de nuvem [12].

Além dos recursos de computação e armazenamento, o tamanho das transações gerará custos tráfego de rede. Os recursos de rede empregado, normalmente, são do tipo de *PrivateLink* (link privado) que fornecem uma conectividade entre organizações que utilizam a mesma provedora de nuvem, sem trafegar dados pela Internet. Esse recurso possuí uma dupla cobraça: fixa e variável. O custo fixo é baseado em horas da existência do recurso, utilizado ou não, assim como vimos para computação na Seção 4.4.3, é utilizada a mesma conversão de horas para semanas, meses e anos, conforme a Tabela 4.1 anteriormente descrita. Então o custo fixo será calculado por hora, multiplifcando o preço de hora pelo total de horas do tipo período escolhido.
O custo variável se referente a quantidade de *gigabytes* trafegados pelos recursos de rede, logo, este recurso está vinculado proporcionalmente a demanda.
Por exemplo, se 1 GB trafegar em um segundo ou uma hora, esse 1 GB terá o mesmo custo. Então, o valor total para um determinado período é o total do custo variável mais o total custo fixo de hora pré-fixado.

Para este modelo, foi considerado que a arquitetura possui o *peer* e *OrderingService* (OS) na mesma sub-rede. Dessa forma, o cliente para chegar no *peer* e OS deve trafegar em entrada e saída. No contexto descrito, considerando a Figura 2.4 anterior (Fluxo de Transação no Fabric), os pacotes do fluxo cliente para o *peer*, *peer* para cliente e cliente para *OrderingService* são considerados no tráfego. Os pacotes para confirmção no *ledger* do *OrderingService* para o *peer* não são considerados por estarem na mesma rede.
Para o contexto do projeto, o total de dados trafegados é considerado o tamanho da transação, sendo assim, pacotes com pouca carga de dados, como *acknowledged*, são desconsiderados.

4.4.5 Projeto Jupyter

O Jupyter Notebook é uma ferramenta interativa que permite a criação e o compartilhamento de documentos e contém código executável, equações, visualizações e texto explicativo em blocos denominados células. As células de código permitem a execução de *scripts Python*, enquanto as células de *Markdown* são utilizadas para adicionar descrições, explicações e formatação ao documento. No contexto do projeto, o Jupyter Notebook é utilizado para adicionar contexto aos códigos com *Markdown*, inclusive, com elementos visuais HTML dentro do código. Além disso, foram utilizadas diversas bibliotecas do ecossistema *Python* simplificando a execução de alguns cálculos, como:

- NumPy: biblioteca fundamental para computação numérica em *Python* fornecendo suporte para arrays de alto desempenho e uma vasta gama de funções matemáticas.
- Pandas: biblioteca essencial para manipulação e análise de dados, oferecendo estruturas de dados flexíveis e eficientes como DataFrames que facilitam a limpeza, transformação e análise dos dados.
- Matplotlib: biblioteca amplamente utilizada para criação de gráficos estáticos, animados e interativos em *Python* que permite a visualização de dados de maneira clara e informativa.

Todos os algoritmos, equações e parâmetros das seções anteriores estão inseridos no Jupyter Notebook adaptados à linguagem de programação *Python*. Por exemplo, quando é verificado o total de armazenamento requerido através da soma cumulativa ao longo do período, é necessário somente o seguinte trecho de código simplificado:

```
1 import numpy as np
2 cumulative_size  = np.cumsum( np.array([array_blocks_size]))
3 cumulative_siz_GB = cumulative_size / pow(1024 , 3)
```

Na linha 1, o *numpy* é importado e atribuído a um *alias* para *np* e em seguida, na linha 2 soma-se cumulativamente todos os valores do array. Na linha 3 são divididos todos os valores inteiros, em uma operação vetorial dividindo todos os valores pelo resultado da exponenciação de 1024^3.

Células de texto, código e parâmetros

Para melhorar a experiência do utilizador, foram inseridos textos em *Markdown* para que além deste documento, exista contexto e explicações auxiliares. Na Figura 4.6, pode-se verificar o exemplo de instruções iniciais do projeto.

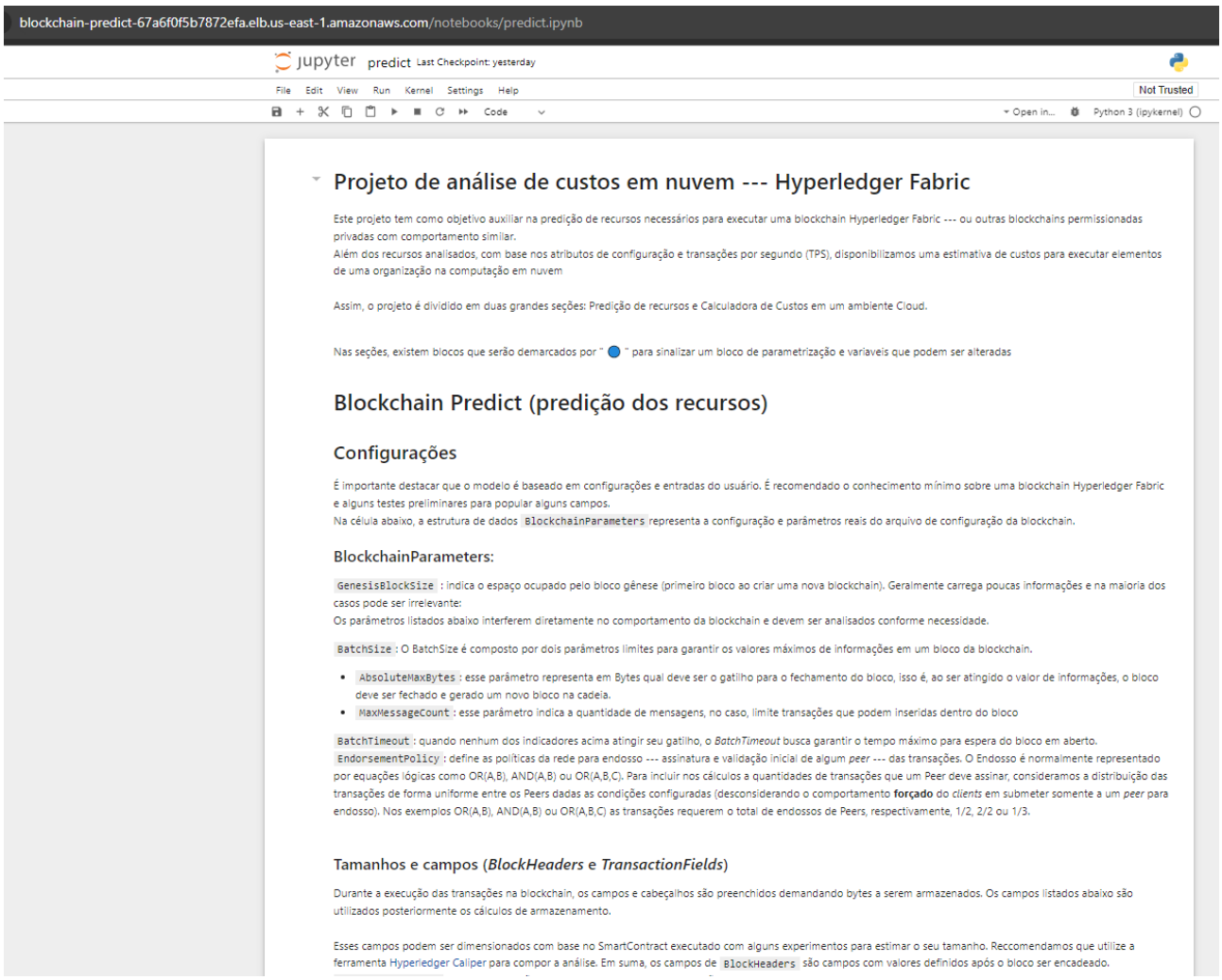
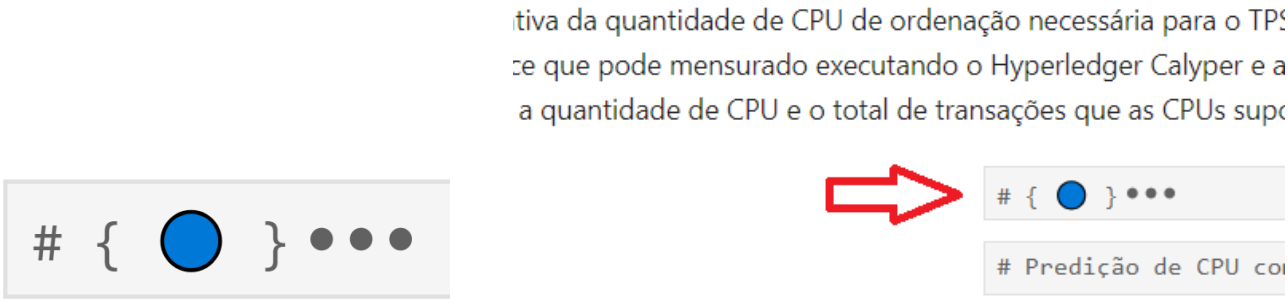


Figura 4.6: Descrições iniciais (Fonte: do autor)

As seções com parâmetros, em geral, são células de código que expõem variáveis que serão utilizadas ao longo do processamento. Um ícone demarcador foi utilizado nas seções com parâmetros, e as células com esse item podem ser modificadas e abertas ao serem clicadas.



(a) Ícone demarcador (b) Célula com ícone demarcador

Figura 4.7: Demarcador e células de parâmetros

As parametrizações são a implementação do que foi anteriormente descrito na Seção “Parametrizações” (4.4.1). Nas figuras seguintes, elencamos as parametrizações

possíveis: configurações da *blockchain* (Figura 4.8), valores de recursos em nuvem (Figura 4.9), modelagem do período e taxa TPS (Figura 4.10) e, por fim, desempenho (Figura 4.11).

```
# { 🔵 }

CONF = {
  "BlockchainParameters": {
    "GenesisBlockSize": 0,
    "BatchSize": {
      "AbsoluteMaxBytes": 10485760,  # Bytes
      "MaxMessageCount": 100000
    },
    "BatchTimeout": 600, #Seconds
  "EndorsementPolicy": 1/2,
  },

  "BlockHeaders": { # Bytes
    "MetadataSize": 64,
    "HeaderSize": 68
  },

  "TransactionFields": {#Bytes
    "R4": 128,
    "E4": 1024,
    "H4": 64,
    "P4": 256,
    "S4": 256
  },

}
```

Figura 4.8: Célula de configurações da *blockchain* (Fonte: do autor)

```
# { 🔵 }
CLOUD_PRICING = { #USD
    # sa-east-1 #AWS São Paulo
    "Storage": {
      "Pricing": 0.19
    },
    "NetworkThroughput": {
      "PricingDataTransfer": 0.01,
      "PricingEndpointHour": 0.01
    },
    "VirtualMachine": {
      "FilePricing": None,#"./ec2_instances_pricing.csv", #None requires insert manual values InstancesTypes.
       "InstancesTypes": [
      # E.g:
        {"Name": " c6a.large","vCPUs": 2,  "Pricing": 0.1179},
        {"Name": " c6a.xlarge","vCPUs": 4,  "Pricing": 0.2358},
        {"Name": " c6a.2xlarge","vCPUs": 8,  "Pricing": 0.4716},
        {"Name": " c6a.4xlarge","vCPUs": 16,  "Pricing": 0.9432},
        {"Name": " c6a.8xlarge","vCPUs": 32,  "Pricing": 1.8864},
       ]

    },
    "Additional": 0 # E.g: taxes or discounts
  }
```

Figura 4.9: Célula de preços dos recursos em nuvem (Fonte: do autor)

A parametrização do tipo de período (dia, semana, mês e ano) é realizada conforme a Figura 4.10a. As Figuras 4.10b e 4.10c se referem a taxa de TPS e são disjuntivas, ou seja, as entradas no vetor da Figura 4.10b anulam a modelagem da curva logística da Figura (4.10c). Em geral, o vetor com valores fixos é indicado para inserir valores estipulados, enquanto a curva logística pode facilitar a modelagem de diferentes crescimentos pela função logística.

```
# 🔵 Required PREDICT_TYPE

global PREDICT_TYPE
PREDICT_TYPE = 'MONTH' # 🔵 'DAY':'WEEK':'MONTH':'YEAR'
```

(a) Tipo do período

```
# 🔵 Caso array [vazio]
global AVG_RATE_TRANSACTIONS

ARRAY = []# 🔵 E.g [50,100,150,200,250,320,370,550,750,1000]
AVG_RATE_TRANSACTIONS = np.array(ARRAY)
```

(b) Entrada fixas em vetor

```
# 🔵 Caso Função Logística (default)

A = 120         # 🔵 Numero de períodos. E.g: para 18 meses => PREDICT_TYPE = 'MONTH'

t = np.linspace(0, A - 1, A)

# Parâmetros da curva logística
P0 = 5          # 🔵 Transações inicial
K = 1000        # 🔵 Valor máximo transações estimado
r =  0.09       # 🔵 Taxa de crescimento
t0 = 12     # 🔵 Ponto de inflexão -> Meio

# Calcula a população usando a função da curva logística

if len(ARRAY) == 0: # Ativar função logística
    AVG_RATE_TRANSACTIONS = logistic_curve(t, P0, K, r, t0)
```

(c) Parâmetros de modelagem da curva

Figura 4.10: Parametrizações de período e taxa TPS (Fonte: do autor)

Ainda dentro do projeto, há seções referentes às entradas de desempenho base para estimar a demanda de CPU, conforme Figuras 4.11a e 4.11b.

```
# { 🔵 }
vCPU_BASE_OSN      =  1
vCPU_BASE_TPS_OSN = 150
```

(a) Tipo do período

```
# { 🔵 }
vCPU_BASE_PEER      =  2
vCPU_BASE_PEER_TPS = 200
```

(b) Tipo do período

Figura 4.11: Desempenho base de CPU (Fonte: do autor)

Código e ambiente de execução

O Notebook desenvolvido com todos os códigos e marcações está disponível em repositório público do *Github* através do link:

https://github.com/gilbertokoerbes/TCCII_Blockchain_Predict

A parametrização e a execução do *Notebook* pode ser realizada das seguintes formas:

- Localmente via *Docker*: realizado o download do repositório, é possível executar localmente com contêineres, sem prejuízo das interatividades com o projeto. No entanto, é necessário que estejam instaladas as ferramentas *Docker* e *Docker Compose*.

É possível iniciar localmente um contêiner JupyterNotebook do projeto ao executar o *shell script* "*start.sh*" que está disponível no repositório. Esse *script* possui todos os comandos para fácil inicialização. Ao executar o "*./start.sh*", o log do terminal exibirá o link de acesso em *localhost*, conforme Figura 4.12.

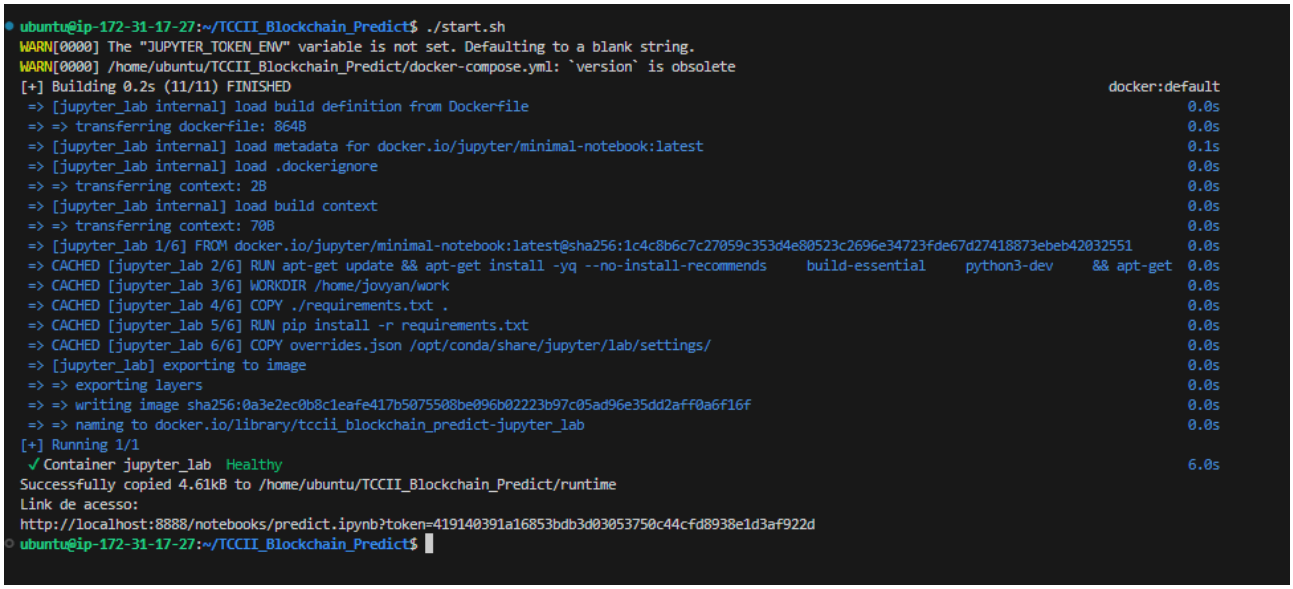


Figura 4.12: *Script* de execução local (Fonte: do autor)

- Localmente via IDE: é possível também executar a ferramenta na IDE Visual Studio Code. No entanto, é necessário possuir instalada a linguagem Python 3.11 ou superior e as extensões JupyterNotebook, conforme ilustrado na Figura 4.13. Como ponto negativo, algumas interações HTML não são possíveis.

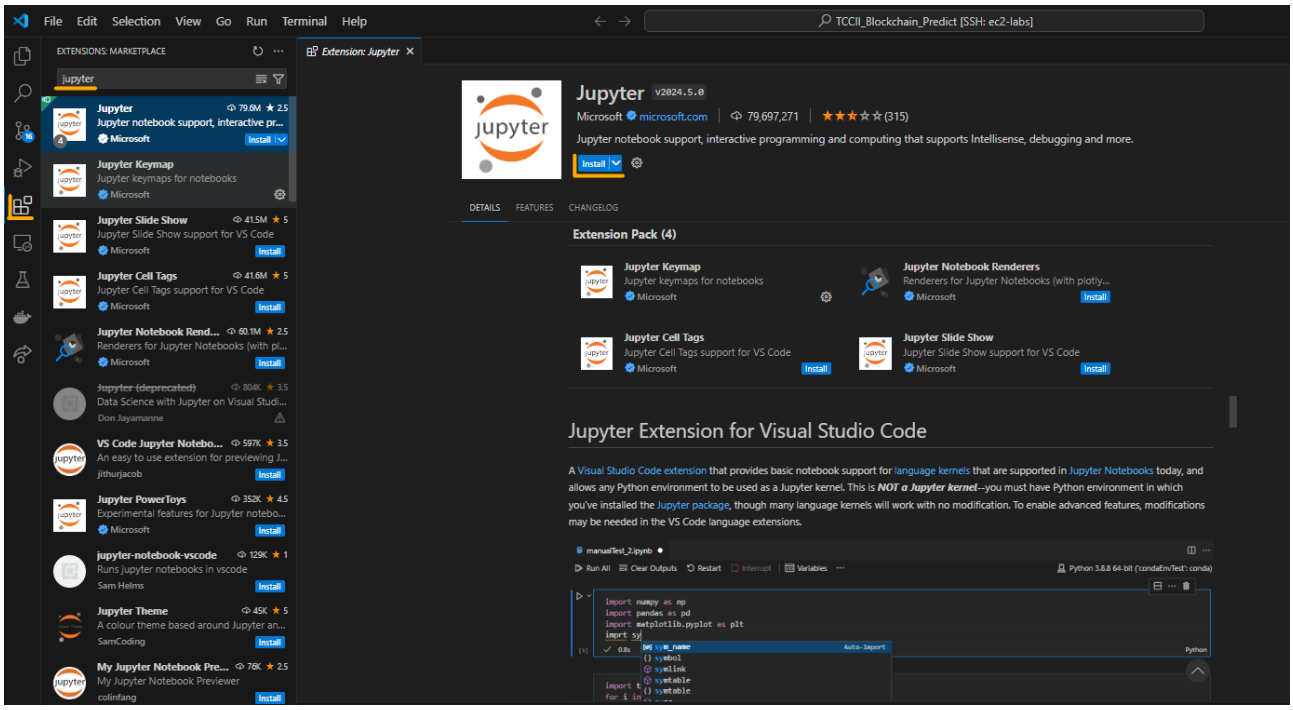
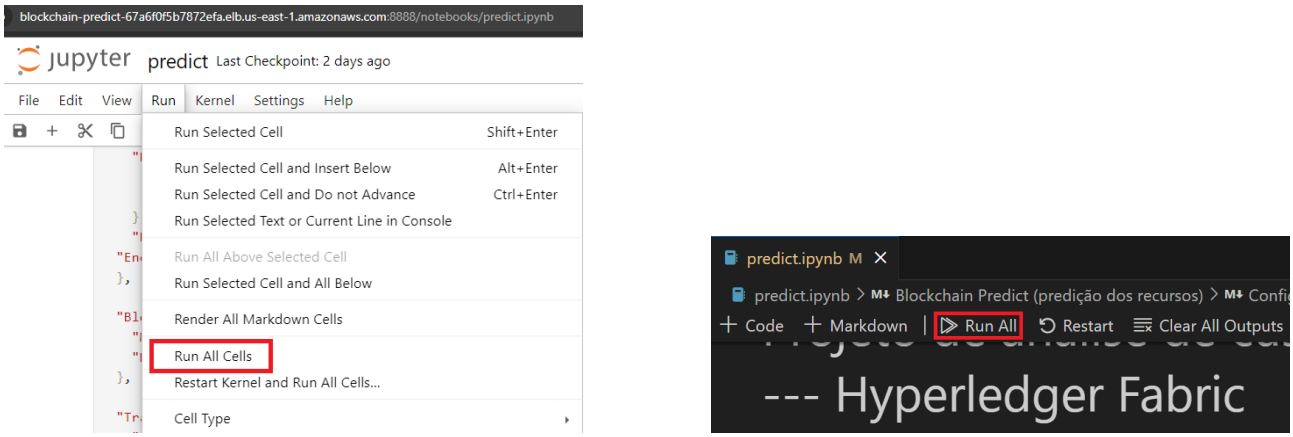


Figura 4.13: Extensão para execução local em IDE (Fonte: do autor)

Em todos os casos, ao alterar parâmetros é necessário reexecutar todas as células do Notebook. Na versão *Docker*, conforme Figura 4.14a, no Menu "*Run*", na opção "*Run All Cells*". Na IDE local, utilizar *Run All*, conforme mostrado na Figura 4.14b:



(a) Execução em Jupyter Notebook

(b) Execução em IDE

Figura 4.14: Orientações de execução (Fonte: do autor)

5. CENÁRIOS AVALIADOS

Com o modelo proposto, são agregadas informações do utilizador com o comportamento da *Blockchain Hyperledger Fabric*. São geradas estimativas e, dessas, são computados valores estimados do custo para provisionar uma infraestrutura em nuvem equivalente à demanda requerida. O projeto apresenta saídas em figuras, gráficos e tabelas. Neste capítulo, serão descritos e comparados os seguintes cenários da utilização da ferramenta para exemplificar o auxílio a tomada de decisão:

1. Comparação de diferentes taxas de adoção e crescimento;
2. Comparação de custos em diferentes provedores de nuvem;
3. Comparação de diferentes políticas de endosso e configurações de *batches*.

5.1 Comparação de diferentes taxas de adoção e crescimento

Para esta comparação, o período analisado, os parâmetros de configuração e custos de nuvem são os mesmos utilizados em ambos os cenários.

```
# { }

CONF = {
  "BlockchainParameters": {
    "GenesisBlockSize": 0,
    "BatchSize": {
      "AbsoluteMaxBytes": 10485760,  # Bytes
      "MaxMessageCount": 100000
    },
    "BatchTimeout": 600, #Seconds
  "EndorsementPolicy": 1/2,
  },

  "BlockHeaders": { # Bytes
    "MetadataSize": 64,
    "HeaderSize": 68
  },

  "TransactionFields": {#Bytes
    "R4": 128,
    "E4": 1024,
    "H4": 64,
    "P4": 256,
    "S4": 256
  },

}
```

(a) *Blockchain* e campos de transações

```
# { }
CLOUD_PRICING = { #USD
    # sa-east-1 #AWS São Paulo
    "Storage": {
      "Pricing": 0.19
    },
    "NetworkThroughput": {
      "PricingDataTransfer": 0.01,
      "PricingEndpointHour": 0.01
    },
    "VirtualMachine": {
      "FilePricing": None,#"./ec2_instances_pricing.csv", #None requires insert manual values InstancesTypes.
       "InstancesTypes": [
      # E.g:
        {"Name": " c6a.large","vCPUs": 2,  "Pricing": 0.1179},
        {"Name": " c6a.xlarge","vCPUs": 4,  "Pricing": 0.2358},
        {"Name": " c6a.2xlarge","vCPUs": 8,  "Pricing": 0.4716},
        {"Name": " c6a.4xlarge","vCPUs": 16,  "Pricing": 0.9432},
        {"Name": " c6a.8xlarge","vCPUs": 32,  "Pricing": 1.8864},
       ]

    },
    "Additional": 0 # E.g: taxes or discounts
  }
```

(b) Custos dos recursos de nuvem

Figura 5.1: Parâmetros e configurações

As comparações são realizadas a partir de diferentes taxas de transações por segundo (TPS) para o mesmo período, determinados em 120 *MONTHS* (meses). Os pa-

râmetros utilizados para modelagem das curvas foram escolhidos de forma arbitrária, mas como objetivo de visualmente ser perceptível a diferença, destacada na Figura 5.2, a curva “A”, com tendência de crescimento mais ao começo e a curva “B” mais ao final. Os parâmetros para modelagem das curvas constam na Tabela 5.1, mas mais especificamente, o parâmetro *t0* (ponto de inflexão) foi alterado para gerar o comportamento desejado.

Parâmetros	Cenário A	Cenário B
P0	10	10
K	1000	1000
r	0.09	0.09
t0	12	96

Tabela 5.1: Parâmetros para modelagem da curva TPS

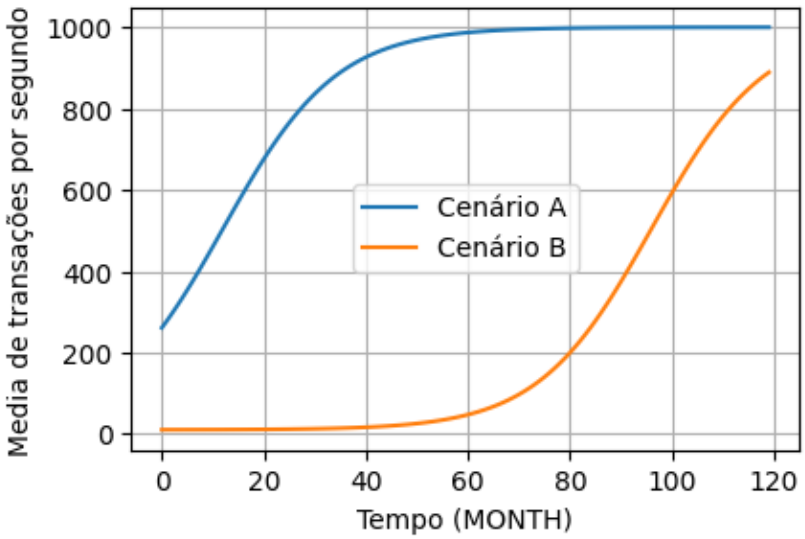
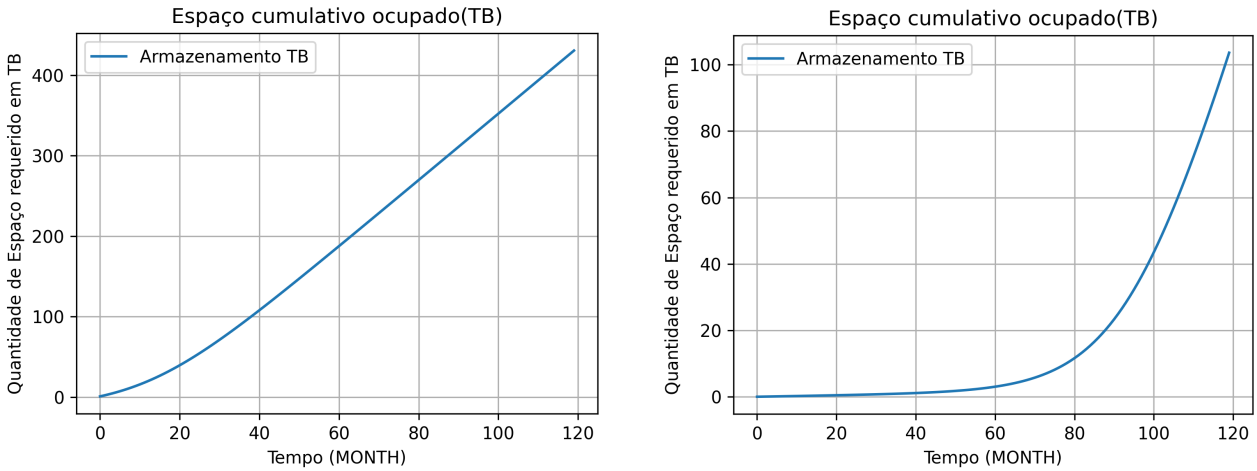


Figura 5.2: Parâmetros e configurações (Fonte: do autor)

Com essas entradas, ao executar o Notebook completo individualmente para cada um dos cenários A e B, há as estimativas geradas que serão analisados à seguir.

Inicialmente, verificamos a demanda de armazenamento (Figura 5.3). Como no cenário A, a taxa de transações por segundo atinge na maior parte do período mais de oitocentas transações por segundo, o crescimento é quase constante, e assim o ponto máximo de armazenamento total estimado (Figura 5.3a) é de aproximadamente 430 *terabytes*. Para o cenário B (Figura 5.3b), próximo ao ponto de inflexão, período 96, começa a ter um crescimento substancial, porém, mesmo com esse aumento, o armazenamento máximo requerido é consideravelmente menor, visto que chega a aproximadamente 106 *terabytes*.

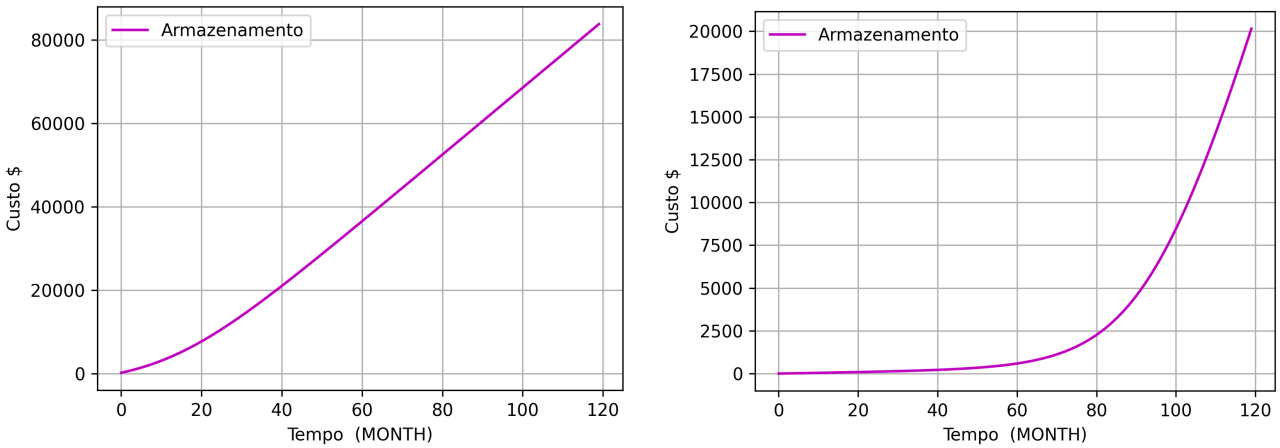


(a) Demanda armazenamento cenário A (b) Demanda armazenamento cenário B

Figura 5.3: Crescimento de armazenamento ocupado (Fonte: do autor)

Ao comparar os cenários, pelo fato do cenário A possuir a maior taxa de TPS na maior parte do período, além das transações, a quantidade de blocos e cabeçalhos impactam para este maior crescimento e, assim, a diferença quase de quatro vezes mais de armazenamento necessário.

Esses valores refletem proporcionalmente ao custo de armazenamento, estimado no ponto máximo em cerca 83.745,95 e 20.152.94 dólares respectivamente aos cenários A e B, conforme Figura 5.4.



(a) Custo de armazenamento – cenário A (b) Custo de armazenamento – cenário B

Figura 5.4: Comparação do custo de armazenamento (Fonte: do autor)

A CPU requerida para o *peer* e para o nodo do *cluster* de ordenação (*Ordering-Service*) são preditas conforme a seguinte configuração da Tabela 5.2. Os valores foram arbitrariamente escolhidos, mas com base e referência aproximada no trabalho de [17].

Parâmetro	Valor
vCPU_BASE_OSN	1
vCPU_BASE_TPS_OSN	150
vCPU_BASE_PEER	2
vCPU_BASE_PEER_TPS	200

Tabela 5.2: Parâmetros de desempenho

Conforme a taxa TPS e os números de desempenho de entrada da Tabela 5.2, as Figuras 5.5 e 5.6, respectivamente aos cenários A e B, apresentam em cada uma a CPU requerida e o modelo possível a ser provisionado para o *peer* e *OrderingService*.

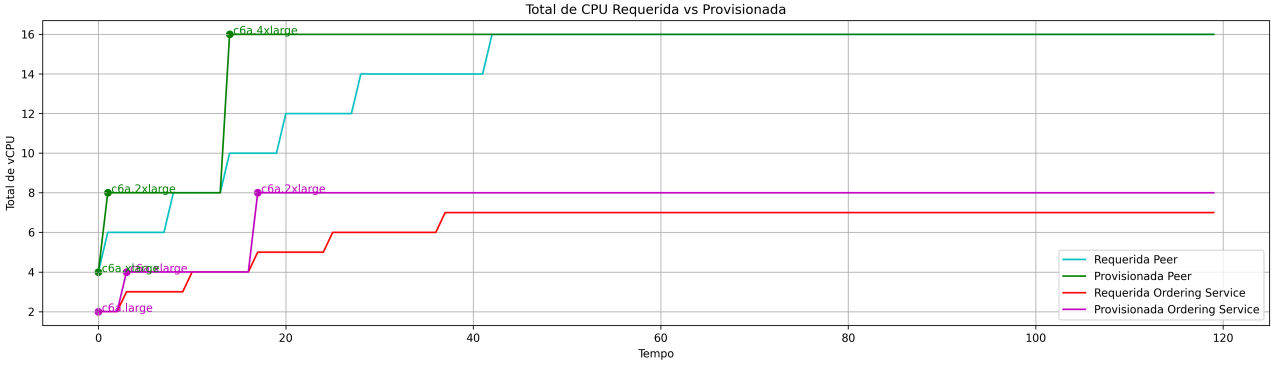


Figura 5.5: Instâncias provisionadas e demanda de CPU requerida cenário A

(Fonte: do autor)

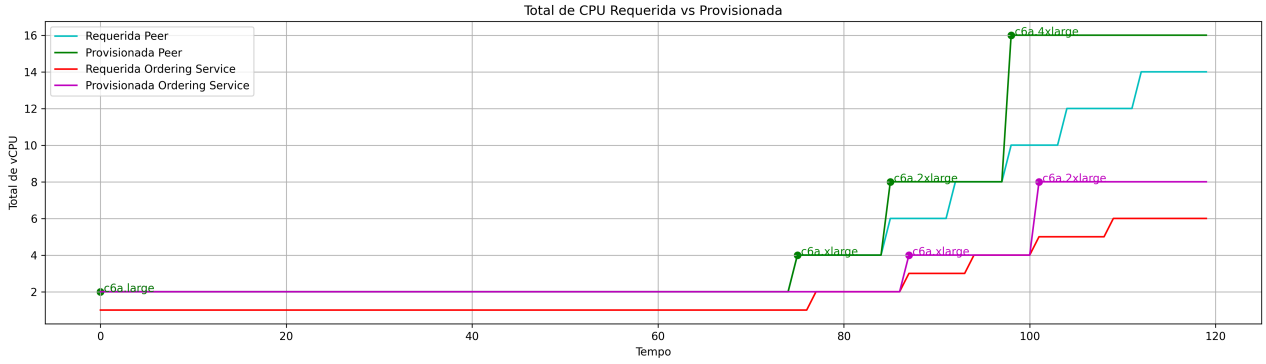


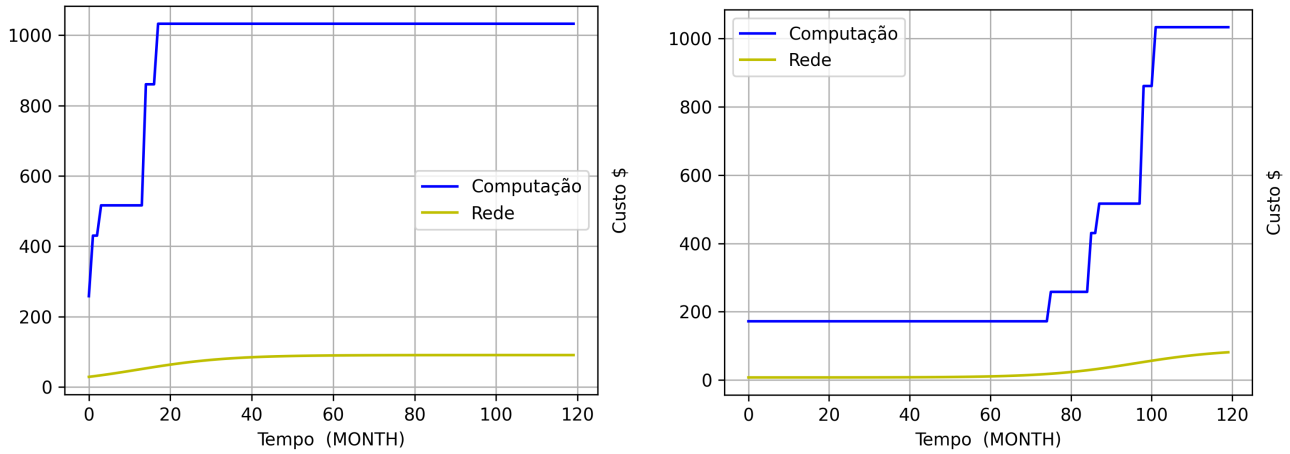
Figura 5.6: Instâncias provisionadas e demanda de CPU requerida no cenário B

(Fonte: do autor)

Verifica-se que, apesar das diferentes cargas em cada cenário, os modelos possíveis provisionados em A e B serão os mesmos a partir dos instantes 18 e 101, respectivamente.

A CPU requerida e rede são custos ligados também à taxa de transações por segundo, em maior parte, aos custos de computação. Devido à natureza dos recursos,

estes não possuem um crescimento cumulativo por serem proporcionais e dependentes a demanda no instante *i*. Assim, o custo de provisionamento final nas Figuras 5.7a e 5.7b indicam valores similares.



(a) Custos cenário A

(b) Custos cenário B

Figura 5.7: Comparação do custo de computação e rede (Fonte: do autor)

Dadas as análises acima, a ferramenta agrega todos os valores e exibe as estimativas do custo total mensal que deve ser considerado para a manutenibilidade e a continuidade da *blockchain*. As Figuras 5.8 e 5.9 exibem a evolução dos custos de todos os recursos analisados (armazenamento, instância para o *peer*, instância para *OrderingService* e rede).

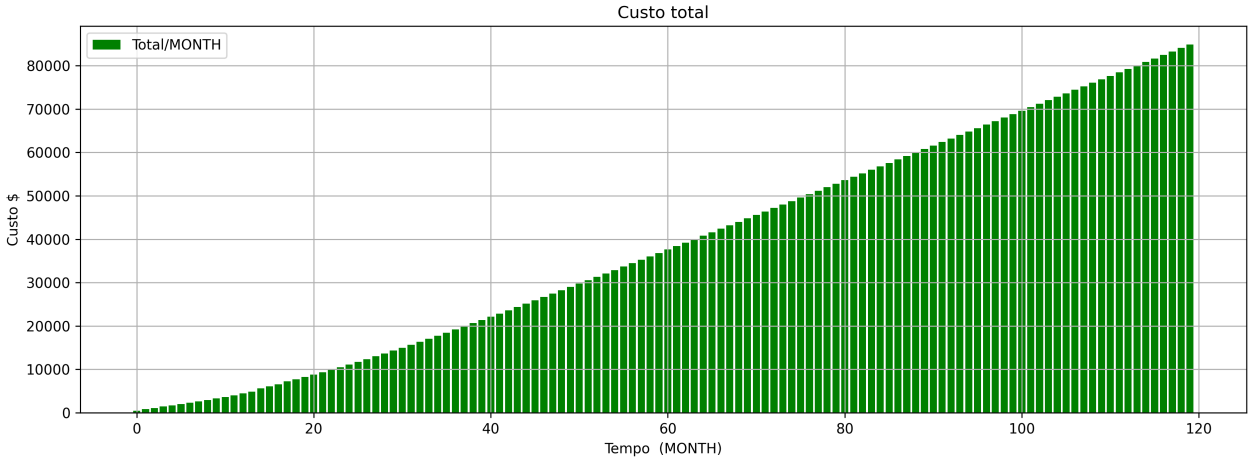


Figura 5.8: Custo total mensal ao cenário A (Fonte: do autor)

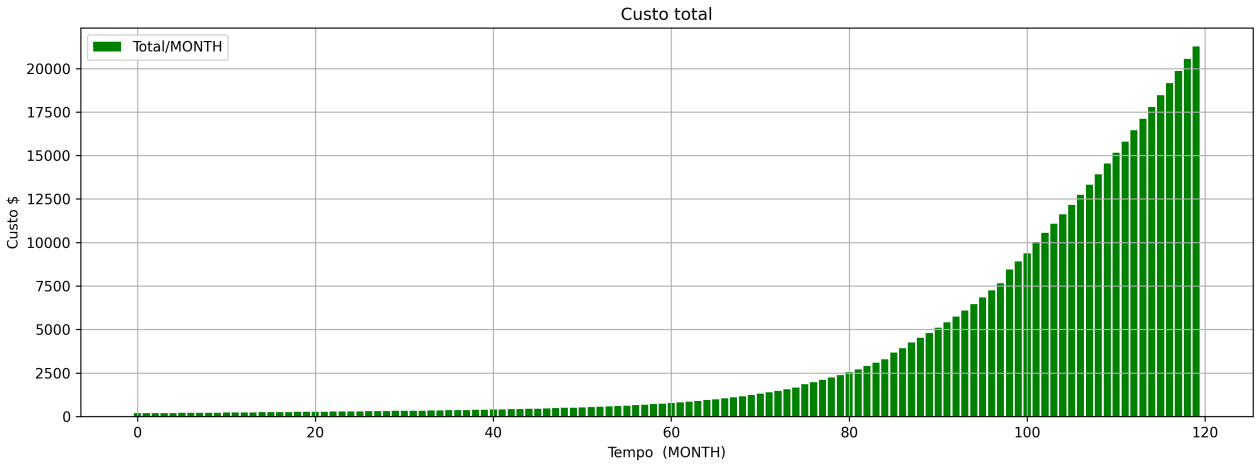


Figura 5.9: Custo total mensal cenário B (Fonte: do autor)

Os valores máximos do custo mensal em cada figura são de, respectivamente, 84,869.47 e 21,267.10 doláres, sendo, que aproximadamente 95% do custo de armazenamento, mais de 4% em computação e o restante menor que 1% em rede.

Ao analisar o custo mensal somado em todo o período, há para o cenário “A” o total de 4.634.965,68 de dólares e para o cenário “B”, 454.073,81, ou seja, uma diferença 10 vezes maior no custo entre os cenários.

Como considerações adicionais, os valores exibidos nesta seção representam apenas uma parte das configurações existentes na ferramenta e aqui foram expostos somente aqueles que são considerados relevantes para a análise do cenário específico.

5.2 Comparação de custos em diferentes provedores de nuvem

Com objetivo principal de comparar os custos de duas soluções de nuvem sob demanda, a partir da lista do provedor AWS estão inseridos todos os modelos de instância otimizados para computação da família *c6a* (Figura 5.10a). A outra provedora de nuvem a ser comparada é a Microsoft Azure, descrita anteriormente no “Quadrante Mágico para infraestrutura de nuvem e serviços de plataforma” (Figura 2.5) como segunda colocada. Os preços de Microsoft Azure são descritos na Figura 5.10b, retirados das documentações oficiais na data deste documento:

```
# { }
CLOUD_PRICING = { #USD
    # sa-east-1 #AWS São Paulo
    "Storage": {
      "Pricing": 0.19
    },
    "NetworkThroughput": {
      "PricingDataTransfer": 0.01,
      "PricingEndpointHour": 0.01
    },
    "VirtualMachine": {
      "FilePricing": None,#"./ec2_instances_pricing.csv", #None requir
       "InstancesTypes": [
      # E.g:
        {"Name": "c6a.large",    "vCPUs":  2,   "Pricing": 0.1179},
        {"Name": "c6a.xlarge",   "vCPUs":  4,   "Pricing": 0.2358},
        {"Name": "c6a.2xlarge",  "vCPUs":  8,   "Pricing": 0.4716},
        {"Name": "c6a.4xlarge",  "vCPUs":  16,  "Pricing": 0.9432},
        {"Name": "c6a.8xlarge",  "vCPUs":  32,  "Pricing": 1.8864},
        {"Name": "c6a.12xlarge", "vCPUs":  48,  "Pricing": 2.8296},
        {"Name": "c6a.16xlarge", "vCPUs":  64,  "Pricing": 3.7728},
        {"Name": "c6a.24xlarge", "vCPUs":  96,  "Pricing": 5.6592 },
        {"Name": "c6a.32xlarge", "vCPUs":  128, "Pricing": 7.5456 },
        {"Name": "C6a.48xlarge", "vCPUs":  192, "Pricing": 11.3184},
```

(a) Preços de recursos - provedor AWS

```
# { }
CLOUD_PRICING = { #USD
    # sa-east-1 #Azure São Paulo
    "Storage": {
      "Pricing": 0.30
    },
    "NetworkThroughput": {
      "PricingDataTransfer": 0.01,
      "PricingEndpointHour": 0.01
    },
    "VirtualMachine": {
      "FilePricing": None,#"./ec2_instances_pricing.csv", #Non
       "InstancesTypes": [
      # E.g:
        {"Name": "F2s  v2", "vCPUs": 2,  "Pricing": 0.1310},
        {"Name": "F4s  v2", "vCPUs": 4,  "Pricing": 0.2620},
        {"Name": "F8s  v2", "vCPUs": 8,  "Pricing": 0.5240},
        {"Name": "F16s v2", "vCPUs": 16, "Pricing": 1.0480},
        {"Name": "F32s v2", "vCPUs": 32, "Pricing": 2.0960},
        {"Name": "F48s v2", "vCPUs": 48, "Pricing": 3.1440},
        {"Name": "F64s v2", "vCPUs": 64, "Pricing": 4.1920},
        {"Name": "F72s v2", "vCPUs": 72, "Pricing": 4.7160}

       ]
```

(b) Preços de recursos - provedor Azure

Figura 5.10: Comparação de preços e recursos provedores de nuvem

(Fonte: do autor)

Para esta comparação, entre os cenários criados foram utilizados os mesmos valores referente ao período analisado, taxa de TPS, as configurações gerais da *blockchain* (*BlockchainParameters, EndorsementPolicy, BlockHeaders e TransactionFields*) e parâmetros de desempenho da CPU (*peer* e *OS*).

A taxa de transações por segundo utilizada nos cenários, Figura 5.11, foi inserida no vetor de parâmetros com os valores equivalentes a $f(x) = 6x$, ignorando a curva logística. O tipo do período escolhido foi "*DAY*" (dia) no período de 365 dias.

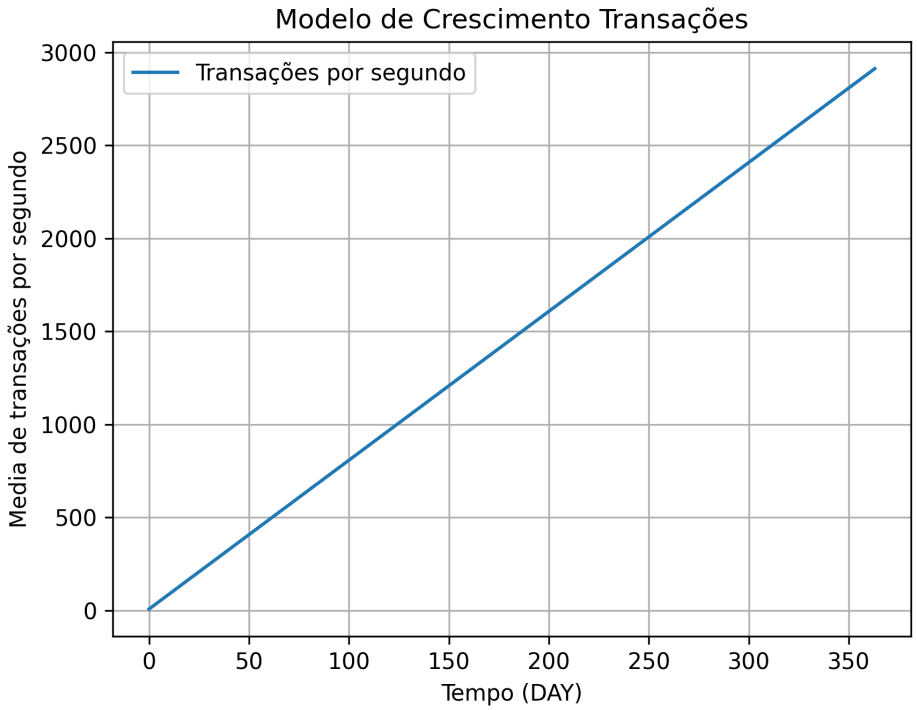


Figura 5.11: Curva TPS utilizada nos cenários (Fonte: do autor)

Os parâmetros de *blockchain* utilizados foram conforme Figura 5.12, os quais estão destacados a política de endosso (*EndorsementPolicy*), utilizado $2/2$ equivalente à $AND(A, B)$, e o campo E4 com 2048 *bytes* (1024 *bytes* por endosso). Isso torna a carga de processamento maior nos *peers*:

```
# { }

CONF = {
  "BlockchainParameters": {
    "GenesisBlockSize": 0,
    "BatchSize": {
      "AbsoluteMaxBytes": 10485760,  # Bytes
      "MaxMessageCount": 1000000
    },
    "BatchTimeout": 600, #Seconds
  "EndorsementPolicy": 2/2,
  },

  "BlockHeaders": { # Bytes
    "MetadataSize": 64,
    "HeaderSize": 68
  },

  "TransactionFields": {#Bytes
    "R4": 128,
    "E4": 2048,
    "H4": 64,
    "P4": 256,
    "S4": 256
  },

}
```

Figura 5.12: Parâmetros de configuração dos cenários (Fonte: do autor)

Com isso, dado o período e parâmetros de configurações, o valor requerido de armazenamento foi de aproximadamente 117,74 *terabytes*, conforme evolução demonstrada na Figura 5.13.

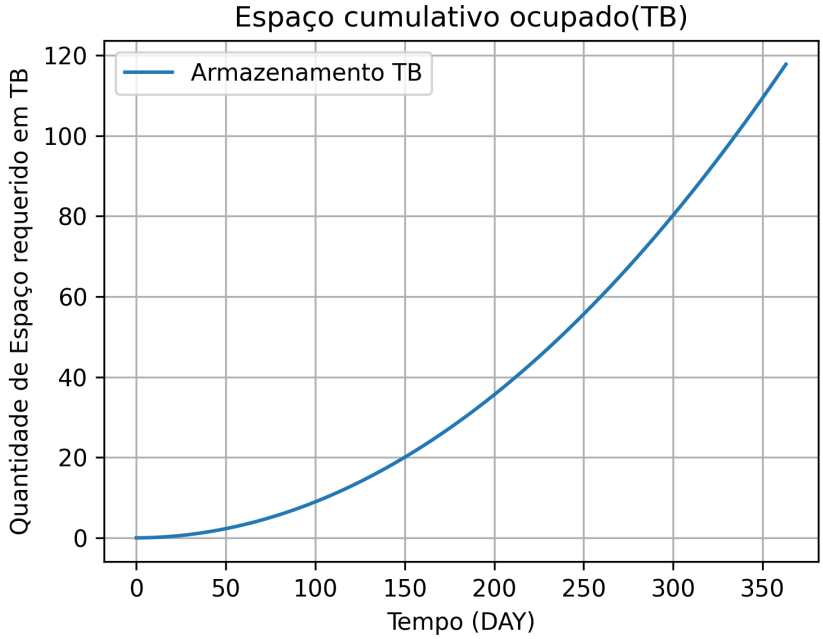
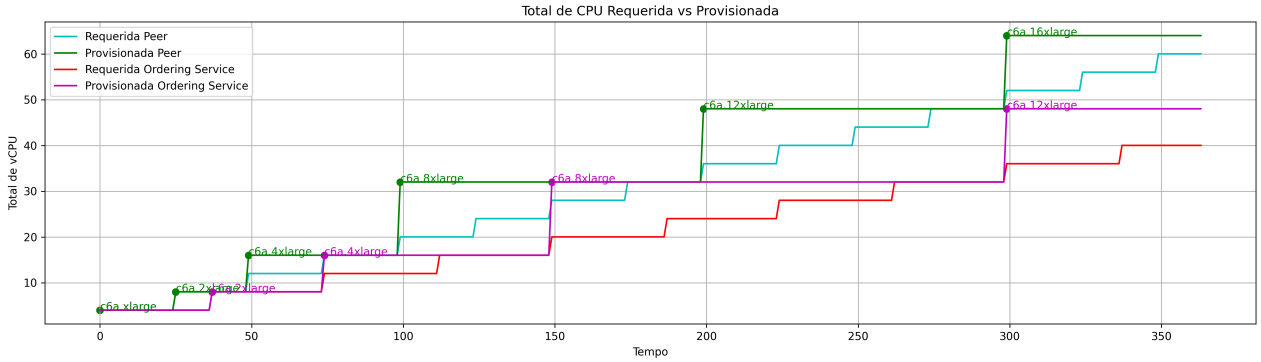


Figura 5.13: Demanda de armazenamento (Fonte: do autor)

Com a taxa de TPS utilizada em ambos os cenários gera-se a mesma demanda de CPU requerida para o cenário “a” (AWS) e “b” (Azure). Em cada cenário, verifica-se

haver alocação de modelos equivalentes, sendo os maiores de 64 CPUs. Para essa carga, os maiores modelos provisionados em AWS e Azure foram, respectivamente, *c6a.16xlarge* e *F64sv2*, esse o segundo maior disponível pelo provedor.

(a) Provedor AWS



(b) Provedor Microsoft Azure

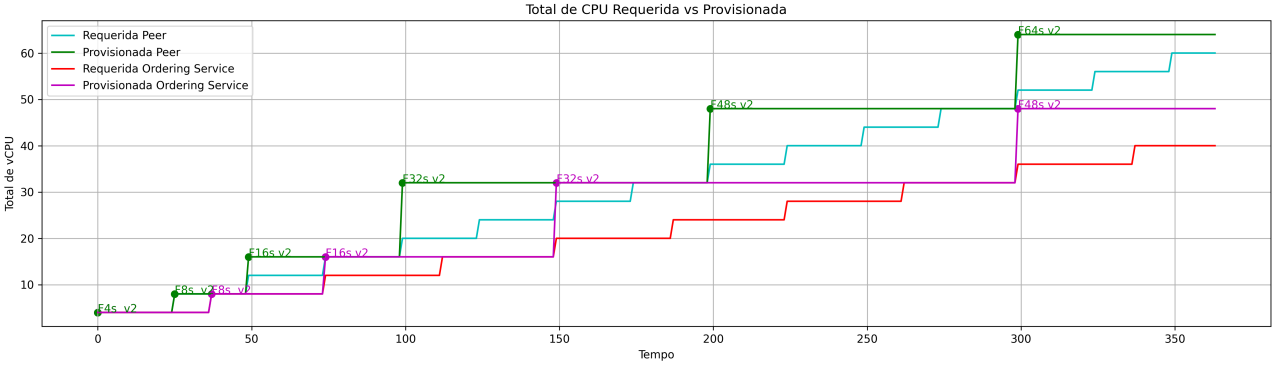
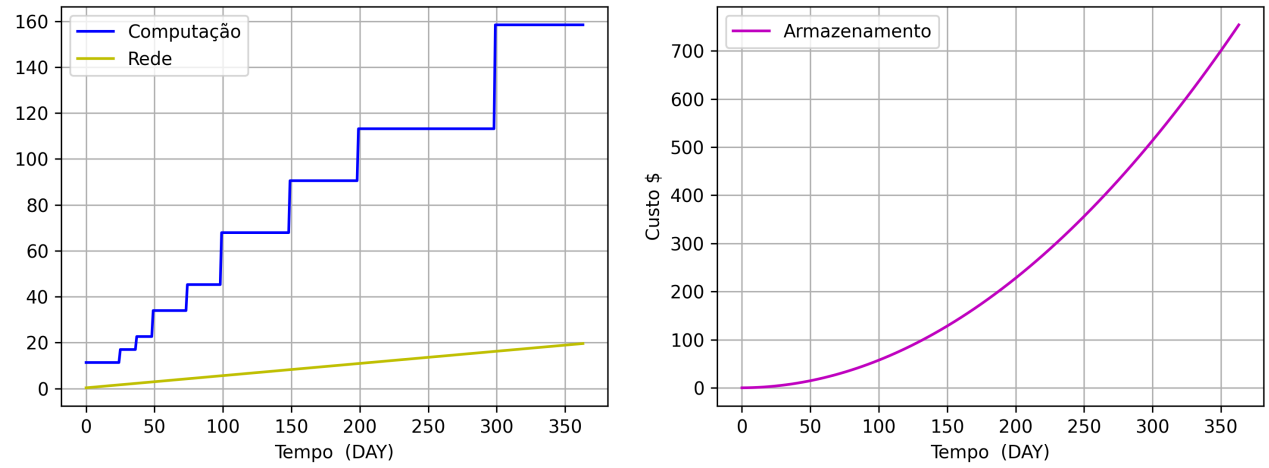


Figura 5.14: Modelos de instâncias alocadas em cada provedor e CPU requerida

(Fonte: do autor)

Assim, os custos de recursos para cada nuvem conforme a demanda são expressos nas Figuras 5.15a e 5.15b:

(a) Provedor AWS



(b) Provedor Microsoft Azure

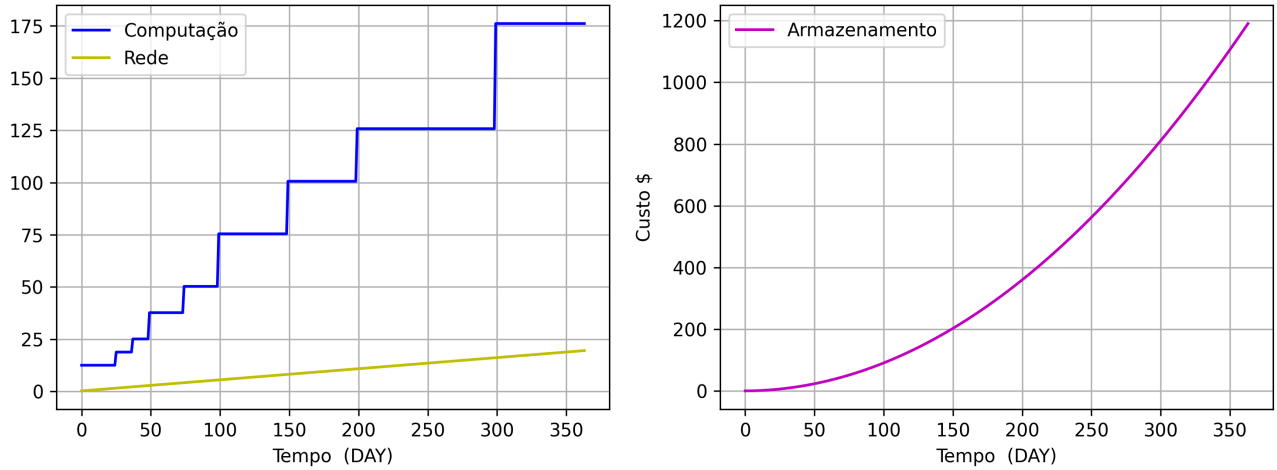


Figura 5.15: Custos individuais dos recursos em cada provedor

(Fonte: do autor)

Os custos do provedor Azure são maiores em computação e armazenamento que da AWS. Para rede, os custos são equivalentes. Para computação, é estimado o valor máximo de 176 e 158 dólares por dia para Azure e AWS, respectivamente, uma diferença de cerca de 11% para computação em cada provedor. Para o armazenamento, os valores máximos no último instante do período foram de 741,51 dólares para AWS e 1.170,81 dólares para Azure, sendo aproximadamente 36% diferença.

Cumulativamente, o custo total do período analisado, isto é, a soma de todos os recursos em todo gráfico, é de 127.341,21 dólares para AWS e 183.868,72 dólares para Azure. O montante tem uma diferença 56.527,51 dólares, cerca de 30,74% do custo menor ao hospedar-se os recursos em AWS.

5.3 Comparação de diferentes políticas de endosso e configurações de batches

O objetivo nesta seção é comparar como as políticas de endosso e configurações de *batches* impactam no armazenamento requerido. Tendo em vista os trabalhos relacionados apresentados, verifica-se a menção da política de endosso e configuração dos *batches* na latência e vazão de rede. Desses mesmos parâmetros, será analisado como podem interferir no armazenamento.
Para esta comparação, o período analisado e taxa de transações por segundo (Figura 5.16) é constante nos cenários. Os valores de desempenho de CPU e custos de precificação são irrelevantes para o objetivo a ser comparado.

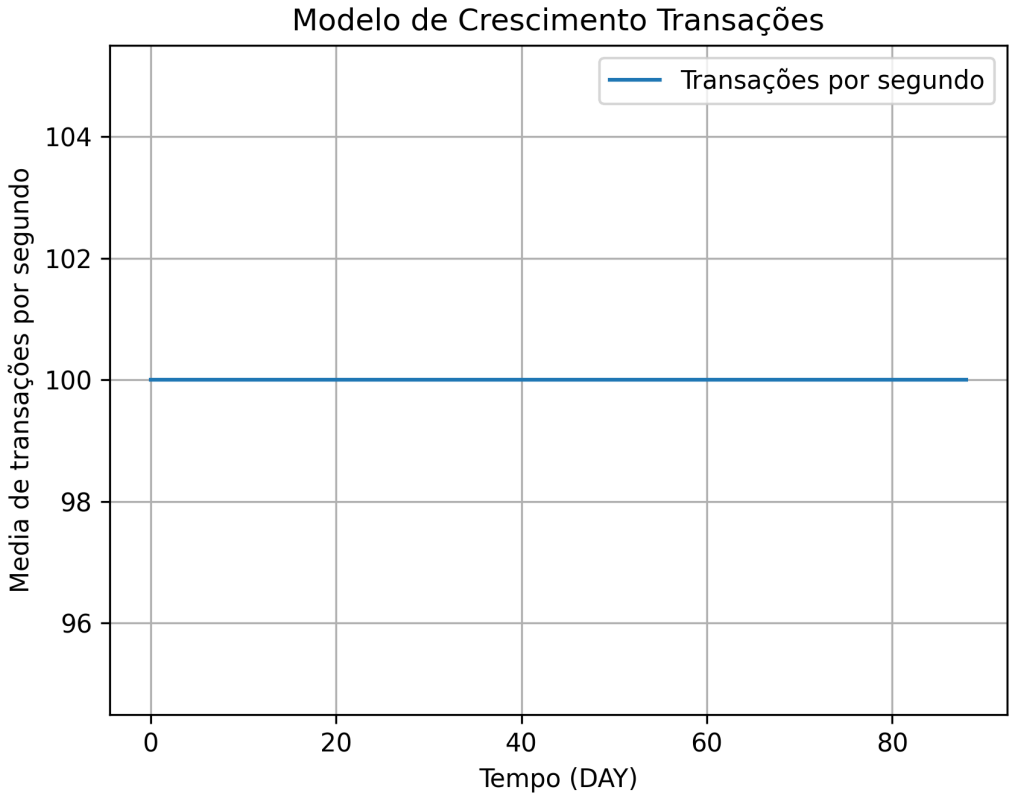


Figura 5.16: Período e TPS (Fonte: do autor)

As variações aplicadas nos arquivos de configuração foram somente em alguns parâmetros conforme a Tabela 5.3.

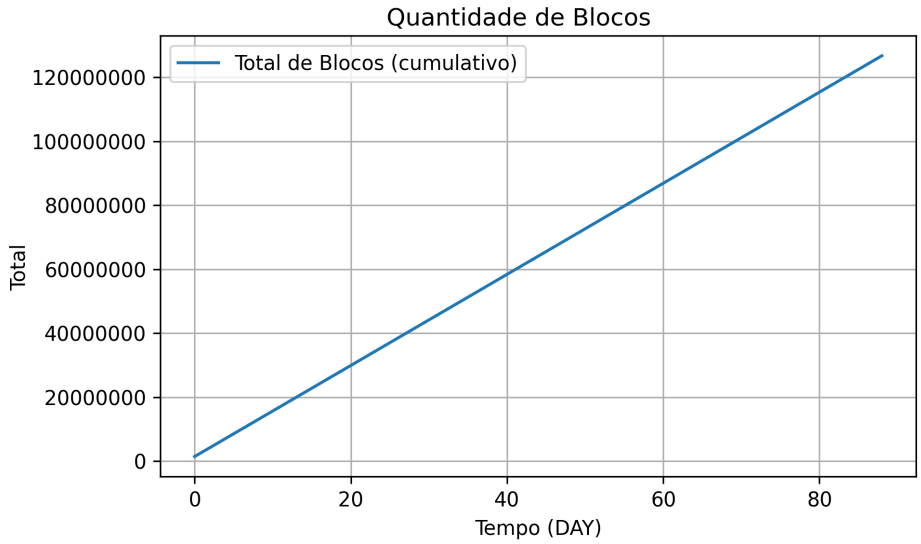
Parâmetro	Cenário A	Cenário B	Cenário C
AbsoluteMaxBytes	10 MBytes	10 MBytes	100 MBytes
MaxMessageCount	1 milhão	1 milhão	10 milhões
BatchTimeout	600	600	900
E4	1024	4096	4096
EndorsementPolicy	1/4	4/4	4/4

Tabela 5.3: Parâmetros de configurações variados entre cenários

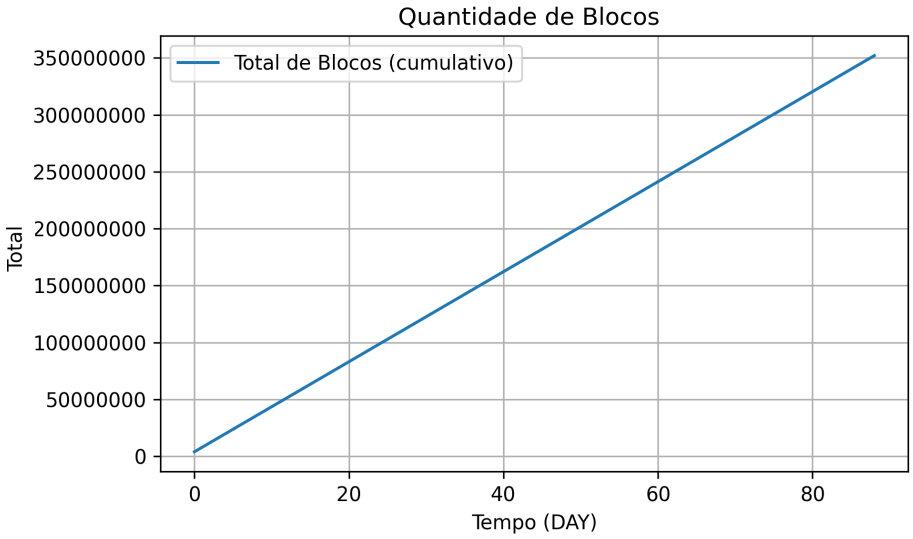
Para o cenário “A” e “B” as configurações de *batches* foram as mesmas, ou seja, as métricas para confirmação do bloco são as mesmas, mas a política de endosso foi alterada para que no cenário B mais *peers* tivessem que realizar o endosso. Sabe-se que o endosso gera mais carga de CPU, mas no caso avaliado o campo E4 se tornará maior para a maior quantidade de endossos necessários.

Para comparação dos cenários “B” e “C”, as políticas de endosso foram as mesmas e com o mesmo tamanho do campo E4, mas as configurações de confirmação do bloco foram incrementadas. O *BatchTimeout* foi acrescido para que o Canal aguarde um tempo maior para confirmação a fim de induzir o bloco a aceitar mais *bytes* (*AbsoluteMaxBytes*) e mais transações (*MaxMessageCount*), ambos parâmetros também foram incrementados.

(a) Provedor AWS



(b)



(c)

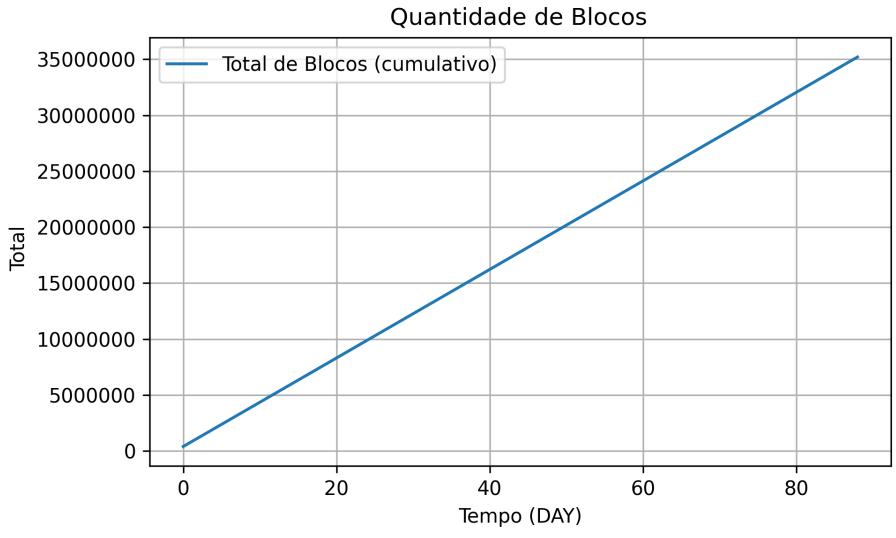


Figura 5.17: Comparação da quantidade de blocos confirmados no período

(Fonte: do autor)

Os resultados na Figura 5.17 confirmam que houve diferença no total de blocos gerados em virtude das configurações. Em comparação dos cenários “A” (Figura 5.17a) e “B” (Figura 5.17b), os valores máximo foram de, respectivamente, 12.672.070,31 e 352.001.953,12 de blocos, uma diferença expressiva no total em virtude de uma política de endosso com mais *peers* incluindo assinaturas. O total de blocos, dadas as mesmas configurações de tamanho dos *batches*, gerou no ponto máximo de armazenamento requerido cerca de 1,10 *terabytes* para o cenário A e 3,09 *terabytes* no cenário B.
Em comparação dos cenários “B” e “C” (Figura 5.17c), a quantidade de blocos no cenário “C” foi de 35.200.195,31, cerca de quase 10 vezes menor que o comparado, no entanto, a diferença de armazenamento foi pouca, cerca de 3,06 *terabytes,* ou seja, 0,03 *terabytes* menor em relação ao cenário “B”. Essa diferença representa o total do armazenamento ocupado pelos cabeçalhos do bloco, visto que as transações possuem as mesmas cargas de informações (mesma política de endosso e campo *E4*).

5.4 Considerações gerais

Nos cenários abordados, a utilização de diferentes parâmetros na *blockchain* e diferentes taxas de transações por segundo gera uma alocação de recursos proporcional a demanda, porém sempre haverá crescimento do armazenamento.

Nos casos avaliados, é possível perceber que o armazenamento possui o custo mais relevante ao ser comparado com o custo total. Isso se deve ao comportamento da *blockchain* que não só atualiza o valor do objeto da transação, bem como registra toda a transação.

Há também de considerar-se que a escolha da nuvem pode influenciar consideravelmente o preço total e determinar a capacidade dos recursos necessários disponíveis. No cenário de “comparação de custos em diferentes provedores de nuvem” (Seção 5.2) caso a taxa TPS fosse maior, o provedor Azure não teria, dentre os modelos, capacidade para alocar maiores instâncias para atender a demanda.

Por fim, pode-se concluir que, em relação ao armazenamento, a política de endosso possui maior relevância que os tamanhos dos *batches* de confirmação dos blocos. Ao utilização uma política de endosso com mais assinaturas, a alteração dos *batches* pode atenuar o armazenamento requerido, mas o ganho de espaço pode ser inferior a 1%, podendo não compensar a medida que *batches* maiores tornam o tempo de confirmação do bloco mais demorado, o que pode ser ruim da visão dos clientes do Canal.

6. CONCLUSÃO

A adoção de *blockchains* privadas tem ganhado espaço em diferentes organizações, inclusive em banco centrais. O *Hyperledger Fabric* tem sido uma escolha alternativa as demandas que necessitam de uma rede *blockchain* permissionada, com características que atendam aos requisitos de identidade, segurança, alta vazão e baixa latência. No entanto, nem todas as organizações estão preparadas para suportar uma rede de configurações complexas ou não queiram ter custos com CapEx (despesas de capitais).

As plataformas de nuvem pública tornam essa adoção mais fácil, seja através de PaaS (como o AMB Fabric) ou IaaS (como AWS EC2, VPC e EBS). Mas em um cenário de computação em nuvem, onde as despesas comumente se dão através do modelo de pagamentos *pay-as-you-go* (pago conforme o uso), é importante se atentar além do custo de computação para operação da *blockchain* com novas transações. Muitos trabalhos relacionados abordam, em maioria, questões de desempenho e escalabilidade em nuvem. Todavia, o custo para manter o histórico das transações, independente de novas transações, relacionado aos custos de armazenamento, é um fator que deve ser considerado nos cálculos a longo prazo para sustentação de uma *blockchain* em nuvem.

Com a ferramenta desenvolvida, é possível modularmente especificar diferentes períodos e variações de configurações através de entradas do utilizador. A ferramenta em Notebook disponibilizada em diversos meios permite que sejam estimados os valores com elementos visuais que auxiliam na tomada de decisão, como na viabilidade de implantação da *blockchain*.

As contribuições do projeto reforçam-se ao analisar os cenários avaliados e resultados obtidos, como com a predição de recursos é possível verificar a quantidade de recursos demandados (não somente em nuvem). Ao analisar os cenários propostos, percebe-se um crescimento disparado de armazenamento sobre os outros recursos de computação e rede. O custo de computação e rede está diretamente ligado ao tempo de execução, isto é, é proporcional a computação utilizada, logo, se o uso da *blockchain* for descontinuado, este custo não existirá. Por outro lado, o custo armazenamento é crescente e cumulativo. Isso pode ser decisivo nas tomadas de decisões a longo prazo. O principal aspecto é que mesmo que uma organização deixe de utilizar a *blockchain*, a depender da legislação aplicada, deverá manter estes dados por período determinado, mantendo o custo de armazenamento. Neste sentido, este projeto contribui para que organizações e partes interessadas tenham previsibilidade do custo de nuvem se essa for utilizada, mas também dos recursos necessário, como armazenamento, computação e rede, independente do ambiente a ser executado.

Dito isso, as organizações devem buscar a eficiência da rede *blockchain*, utilizando os *smartcontracts* de maneira assertiva, evitando, por exemplo, transações inválidas que também são gravadas no *ledger*.

Em conclusão, o desenvolvimento deste trabalho foi bem-sucedido na criação de um modelo e implementação como ferramenta aberta, através do estudo das documentações oficiais, artigos e experimentações, este projeto detém conhecimento sobre o funcionando além do superficial sobre *blockchains* privadas, especialmente *Hyperledger Fabric* e ao provisionamento de recursos em nuvem. Como trabalhos futuros, destaca-se a possibilidade de criar estimativas e cenários com a ferramenta e hospedar-se uma *blockchain* real em nuvem, acompanhando pelo período estipulado, a validação dos resultados preditos com a execução e crescimento.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Amazon Web Services, I. “O que é o amazon ec2?” Capturado em: https://docs.aws.amazon.com/pt_br/AWSEC2/latest/WindowsGuide/concepts.html, 2023.

[2] Androulaki, E.; Barger, A.; Bortnikov, V.; Cachin, C.; Christidis, K.; De Caro, A.; Enyeart, D.; Ferris, C.; Laventman, G.; Manevich, Y.; et al.. “Hyperledger fabric: a distributed operating system for permissioned blockchains”. In: Proceedings of the thirteenth EuroSys conference, 2018, pp. 1–15.

[3] Bala, R.; Smith, D.; Ji, K.; Wright, D.; Borrega, M. A. “Quadrante mágico para infraestrutura em nuvem e serviços de plataforma”. Publicado em 19 de outubro de 2022. ID G00756608, Capturado em: https://www.gartner.com/technology/media-products/reprints/AWS/1-2AOZQARL-PTB.html, 2022.

[4] Baliga, A.; Solanki, N.; Verekar, S.; Pednekar, A.; Kamat, P.; Chatterjee, S. “Performance characterization of hyperledger fabric”. In: 2018 Crypto Valley conference on blockchain technology (CVCBT), 2018, pp. 65–74.

[5] Carlan, V.; Sys, C.; Vanelslander, T. “Cost-effectiveness and gain-sharing scenarios for purchasing a blockchain-based application in the maritime supply chain”, *European Transport Research Review*, vol. 14–1, 2022, pp. 1–19.

[6] Fabric, H. “A blockchain platform for the enterprise”. Acesso em 07 de novembro de 2023, Capturado em: https://hyperledger-fabric.readthedocs.io/en/release-2.5/index.html, 2020-2023.

[7] for International Settlements (BIS), B. “The Future Monetary System”. 2022.

[8] Mell, P.; Grance, T.; et al.. “The nist definition of cloud computing”, 2011.

[9] Melo, C.; Dantas, J.; Pereira, P.; Maciel, P. “Distributed application provisioning over ethereum-based private and permissioned blockchain: availability modeling, capacity, and costs planning”, *The Journal of Supercomputing*, 2021, pp. 1–27.

[10] Rimba, P.; Tran, A. B.; Weber, I.; Staples, M.; Ponomarev, A.; Xu, X. “Quantifying the cost of distrust: Comparing blockchain and cloud services for business process execution”, *Information Systems Frontiers*, vol. 22, 2020, pp. 489–507.

[11] Roy, U.; Ghosh, N. “Fabman: A framework for ledger storage and size management for hyperledger fabric-based iot applications”, *IEEE Transactions on Network and Service Management*, 2024, pp. 1–1.

[12] Services, A. W. “Suposições da calculadora”. Capturado em: https://aws.amazon.com/pt/calculator/calculator-assumptions/,, 2023.

[13] Services, A. W. “What is cloud computing?” Capturado em: https://aws.amazon.com/pt/what-is-cloud-computing/, 2023.

[14] Services, A. W. “What is hyperledger fabric?” Capturado em: https://aws.amazon.com/pt/blockchain/what-is-hyperledger-fabric/, 2023.

[15] Tao Zhang, Z. “Blockchain and central bank digital currency”. Capturado em: https://www.sciencedirect.com/science/article/pii/S2405959521001399, Jun 2022.

[16] Thakkar, P.; Nathan, S.; Viswanathan, B. “Performance benchmarking and optimizing hyperledger fabric blockchain platform”. In: 2018 IEEE 26th international symposium on modeling, analysis, and simulation of computer and telecommunication systems (MASCOTS), 2018, pp. 264–276.

[17] Wang, C.; Chu, X. “Performance characterization and bottleneck analysis of hyperledger fabric”. In: 2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS), 2020, pp. 1281–1286.

[18] Xu, X.; Weber, I.; Staples, M. “Architecture for blockchain applications”. Springer, 2019.