UNIVERSIDADE FEDERAL DO ABC

TRABALHO DE GRADUAÇÃO EM

ENGENHARIA DE INFORMAÇÃO

Consistência e Qualidade de Big Data em Sistemas Distribuídos

Marcos Romero

Santo André, SP

2020

Marcos Romero

Consistência e Qualidade de Big Data em Sistemas Distribuídos

Monografia apresentada ao curso de Engenharia de Informação da Universidade Federal do ABC como parte dos requisitos para a obtenção do grau de Engenheiro.

UNIVERSIDADE FEDERAL DO ABC

Orientador:
Prof. Dr. Ricardo Suyama

Santo André, SP

2020

Dedico este trabalho a minha amada Verônica, que nunca deixou de me motivar a alcançar meus sonhos.

Sumário

Lista de Figuras

Resumo

As relações sociais e econômicas são fortemente dependentes da troca de dados entre sistemas. Bases de dados distribuídas podem estar eventualmente inconsistentes por inúmeros fatores. Dados inconsistentes geram informações incorretas e podem levar a falhas operacionais, perdas financeiras e até mesmo fatalidades. Portanto, se torna necessário a apreciação da qualidade dos dados e da informação para evitar prejuízos. A forma de se tratar a qualidade, especificamente a questão da consistência dos dados, em sistemas Big Data tem sido computacionalmente ineficaz, baseando-se no cruzamento e batimento de grandes conjuntos de dados entre si, ou seja, comparando cada linha e coluna entre dois ou mais conjuntos para encontrar eventuais inconsistências. Novas arquiteturas de sistemas Big Data são propostas para melhorar os modelos anteriores no ponto de garantia da consistência de dados distribuídos. Observar não só o último valor do dado, mas também o registro de eventos das trocas dos valores durante seu ciclo de vida pode trazer mais informações e ajudar no tratamento de falhas. Logo, é proposto neste trabalho uma prova de conceito de uma nova arquitetura de sistemas Big Data, comparando um sistema distribuído tradicional, como exemplo, de uma empresa de telecomunicações, evoluindo para um sistema com um intermediador baseado em registros capaz de tratar inconsistências sempre que um evento ocorre, e um último sistema com uma arquitetura redesenhada, orientada a eventos, para minimizar a possibilidade de inconsistências, aceitando a maior probabilidade da falha.

CAPÍTULO 1

Introdução e Motivação

A popularização das redes de Internet tornou o mundo hiperconectado e, com isso, um volume de dados recorde está sendo coletado a todo momento. Em 2013, previa-se que mais dados seriam coletados em um ano do que todos registros já produzidos desde o início da humanidade [1]. Foi necessário desenvolver novas ferramentas para tratar o denominado Big Data, grandes volumes de dados, com alta velocidade de resposta e enorme variedade que não podem ser tratados somente com abordagens tradicionais de forma eficiente em tempo e recursos. O processamento de Big Data com técnicas avançadas para análise de dados tornou-se ponto chave para manter a competitividade das empresas no mercado [2].

Dados isolados não representam informação: para isso é necessário agregá-los e analisá-los, trazendo assim valor para as organizações. Um grande aparato computacional é necessário para que o Big Data seja processado eficientemente e assim sejam produzidas informações úteis e com significado prático ao negócio e ao propósito das organizações. As fontes de dados no mundo real não são ideais, ou seja, não necessariamente atendem as premissas de transações **ACID**, Atômicas, Consistentes, Isoladas e Duráveis, pelo contrário, são comumente heterogêneas, de diferentes formatos e esquemas de dados, sem controle de transações, distribuídas e não estruturadas. Analisar e manter a qualidade desses dados é fundamental para muitos modelos atuais. O conceito qualidade de dados é definido pela norma ISO/IEC 25012:2008 como "o grau em que as características dos dados satisfazem necessidades declaradas e implícitas quando usados sob condições específicas". A mesma norma define os atributos que podem ser avaliados no âmbito da qualidade: correção, completude, consistência, credibilidade, atualidade, acessibilidade, conformidade, confidencialidade, eficiência, precisão, rastreabilidade, inteligibilidade, disponibilidade, portabilidade e recuperabilidade

[3].

Em Sistemas Distribuídos, podem ser utilizados diferentes bancos de dados e, cada vez mais frequente, com volume, variedade e velocidade de Big Data. A integração e consistência dessas grandes bases distribuídas é vital para o funcionamento dos sistemas. A integração dos dados permite que os sistemas se comuniquem entre si de forma adequada e a consistência dos parâmetros permite que as operações envolvendo esses dados não gerem informações incorretas ou desatualizadas.

Durante a integração de dados de bases distribuídas são encontrados diversos desafios devido às diferentes tecnologias, esquemas variados de banco ou problemas externos, como redes deficientes ou estruturas de dados diversas. Essas características afetam principalmente a questão da qualidade dos dados e da informação [4].

Problemas com a qualidade dos dados estão presentes diariamente, porém nem sempre de forma evidente. Se dados isolados não trazem informação, dados incorretos podem levar a informações incorretas, falhas em sistemas e até mesmo situações fatais. Por exemplo, falhas de chamada em uma rede celular, às vezes simplesmente atribuídas a interferência na rede de acesso ou falha de hardware, na verdade, podem ser devidas às inconsistências dos dados dos assinantes entre bases internas da operadora. Por exemplo, uma falha que deixe inconsistente as chaves de autenticação de um *simcard* na base de dados da operadora pode fazer aparentar que o chip está inutilizado, sem apresentar registro ou sinal no aparelho celular. E, de fato, a primeira suspeita para o cliente ou colaboradores de atendimento nas lojas é que o chip necessita ser substituído, trazendo o custo e o impacto ambiental de um novo cartão plástico com o chip. O problema poderia ser evitado corrigindo a inconsistência da chave de autenticação na base de dados. Em 2005, baseado em entrevistas com especialistas da indústria, foi estimado que a perda financeira com dados de má qualidade era de mais de 600 bilhões de dólares ao ano, somente no ambiente de negócios dos EUA [5].

Para manter os dados e informações com uma qualidade aceitável, sem falhas críticas e impactos aos usuários, diversas empresas disponibilizam soluções de gerência de qualidade de dados (*Data Quality Managemnt Tools*). Dentre empresas que oferecem os softwares mais bem avaliados pela consultoria Gartner estão:

- Informatica - *Data Quality Advanced Edition*;
- SAP - *SAP Data Services*;
- Microsoft - *Microsoft Data Quality Services*;
- Oracle - *Oracle Enterprise Data Quality*;
- SAS - *SAS Data Management*

Portanto, as grandes empresas de tecnologia da atualidade estão competindo diretamente pelo mercado de qualidade de dados e informações. O que é mais um indicador da importância do assunto e da necessidade de ser um ponto considerado desde a concepção do projeto.

No setor de Telecomunicações, os temas relacionados à qualidade de dados que mais concentram esforços são aqueles que trazem prejuízo financeiro direto, relacionados à garantia de receita. Podem ser citadas algumas empresas especializadas, que propõem realizar o controle de fraudes, garantia de receita e garantia de qualidade de serviços nas operadoras. Alguns exemplos:

- Roscom - *Telecom Assurance*;
- WeDo Technologies - *RAID - Revenue Assurance Integrity Driller*;
- Visent - *Revenue Assurance*

Durante muitos anos a estratégia utilizada para se tratar as inconsistências de bases de dados distribuídas foi ineficiente. Era necessário interromper completamente as atualizações de novos dados, extrair as bases de dados em horário sincronizado, como tirar uma fotografia de cada base do sistema. Carregar as bases em algum servidor comum e realizar a comparação dos dados entre as bases. Após esse processo um relatório seria gerado com as inconsistências encontradas. Com base no relatório, deveria se definir, para cada parâmetro, qual dos dados seria considerado o "verdadeiro"e então partir para a equalização das bases. Dependendo do tamanho das bases e da complexidade do sistema, este processo consumiria horas ou até dias. Caso não fosse possível manter o sistema totalmente parado até a equalização das bases, o resultado a ser equalizado já provavelmente estaria desatualizado no momento da correção.

Assim o processo se torna caro e demorado, até que se consiga corrigir todas as inconsistências e todas as causas de originação dessas falhas. Porém, para muitos cenários complexos, com sistemas com dezenas de bases diferentes, não se encontram todas as causas de problemas que venham a gerar falhas. Logo, um processo de tratamento de forma *offline* das inconsistências de base se torna um processo contínuo, mobilizando equipes inteiras, com alto custo e pouco retorno à longo prazo [6].

Este é o caso de uma das ferramentas utilizadas por operadoras de telecomunicações, RAID da empresa Mobileum. De acordo com seu prospecto, a coleta de dados é um processo necessário para a auditoria [7]. Extrair os dados, analisá-los e processá-los, como dito, tende a ser ineficiente para volumes de Big Data e o prazo para correção se torna inviável.

Pesquisas e produtos recentemente estão propondo uma nova abordagem de arquitetura de sistema que privilegia a análise dos registros (*logs*) e não somente o último valor dos dados, bem como a orientação a eventos. Considerando que os eventos e registros possuem mais

informações do que somente o dado do estado atual, analisá-los traz uma vantagem muito maior na garantia de consistência e correção. Como disse Pat Helland no seu artigo *Immutability Changes Everything*, a verdade é o *log*, o banco de dados é apenas um subconjunto temporário dos *logs* [**8**]. Enquanto o valor representado no banco de dados só informa o estado atual, o *log* traz todo o ciclo de vida do dado, do nascimento ao estado atual, incluindo eventuais erros durante o ciclo.

Este trabalho tem por objetivo apresentar uma prova de conceito em tamanho reduzido de uma nova arquitetura orientada a eventos para mitigar e tratar inconsistências em transações distribuídas, baseada em mensagens e tópicos de publicação e assinatura (*publish/subscribe)* com Apache Kafka. Além da apresentação dessa arquitetura, objetiva-se a implementação de uma solução para o problema da qualidade de dados em arquiteturas legadas, que não foram projetadas inicialmente para trabalhar com análise de eventos. A solução funcionaria como um intermediário, capaz de ter acesso a todas as transações distribuídas de alteração de dados e de verificar e corrigir as inconsistências geradas utilizando Apache NiFi assim que há a disponibilidade para a correção, o chamado quase tempo real, *near real-time*.

As duas propostas serão comparadas e com uma abordagem tradicional, contendo um elemento principal com a base de dados original e dois elementos secundários que receberão os comandos do elemento principal e devem se manter consistentes com a base de dados original. Algumas falhas com perda de comunicação serão simuladas para evidenciar cenários de possíveis novas inconsistências no ciclo de vida dos dados. Após as falhas, será realizado o processo de batimento, uma exportação das bases no mesmo instante e comparação dos dados separados do ambiente ativo para posterior correção dos dados se necessário.

CAPÍTULO 2

Estado da Arte

Em Sistemas de Informação de grandes empresas é muito comum que seja necessário realizar a integração de grandes volumes de dados. Bases de diferentes fornecedores e diferentes formatos devem se relacionar de acordo com as regras de negócio estabelecidas em projeto. O ciclo de vida dos dados é tão importante quanto o ciclo de vida do cliente, um erro nos dados pode ser determinante para a perda de um cliente.

O maior desafio encontrado é quando as bases de dados não foram concebidas individualmente para ter esse tipo de relação e não possuem mecanismos de garantia da qualidade dos dados entre bases distintas, ao contrário do que se encontra em bancos de dados monolíticos que atendem somente a um tipo de sistema. No caso dos bancos monolíticos, é uma premissa que as transações garantam as propriedades **ACID**, acrônimo para Atomicidade, Consistência, Isolamento e Durabilidade [9].

Um exemplo de necessidade de integrações de diferentes bancos de dados, mantendo a consistência das informações entre eles, é o sistema de uma operadora de telecomunicações. O sistema de atendimento ao cliente, voltado para canais digitais, *call center*, lojas e vendas, deve compartilhar certas informações com os sistemas da Rede que cuidam do registro, autenticação, mobilidade, controle de saldo e cota. Por exemplo, a recarga de créditos um assinante pré-pago pela Internet, deve refletir em alguns segundos na liberação para os serviços de voz e dados do cliente, ou seja, a informação recebida via canal digital, deve ser propagada das bases de dados de informação de recargas para as bases de dados de Rede, como HLR - *Home Location Register* e VLR - *Visitor Location Register*.

O nome dado em operadoras de telecomunicações para os sistemas que suportam os dados de clientes e tarifação é BSS - *Business System Support* que atua em conjunto com os

sistemas de OSS - *Operating System Support* que são responsáveis pela mediação e aprovisionamento entre os sistemas de BSS e os elementos da Rede de Telecomunicações. Normalmente, essa arquitetura segue alguns padrões internacionais como o estabelecido pelos órgãos TM Forum e ETSI - *European Telecommunications Standards Institute*. Na norma 3GPP TR 32.901 é descrito como o sistema BSS utiliza elementos chamados *Provisioning Gateways* para realizar a replicação dos dados de clientes dos sistemas de tarifação para as bases de dados de assinantes nos elementos de Rede [**10**].

Alguns exemplos de dados que precisam ser refletidos de um ambiente de tarifação e controle para o ambiente de rede são número do assinante, número do *simcard*, código do plano, data de renovação de franquia, características de tarifação para chamadas de voz, dados, mensagens de texto. Todos são parâmetros que, caso estejam inconsistentes, geram uma percepção de falha ou desserviço para o usuário final, até mesmo prejuízos financeiros para a operadora ou cliente. Estudos anteriores apontam que as falhas como essas foram determinantes para 44% dos clientes que cancelaram seu plano e mudaram de operadora, tiveram essas falhas em serviços centrais como um dos gatilhos para o cancelamento de seu vínculo [**11**].

O processo executado hoje por algumas empresas baseia-se em um fluxo ETL - *Extract, Transform, Load* para um grande conjunto de banco de dados como um *Data Warehouse* e então são analisadas as inconsistências, possíveis falhas ou fraudes, conforme demonstrado na Figura 2.1. Uma lista de inconsistências é gerada, para então programar a correção. Por mais que esse processo seja automatizado, muitas vezes são dezenas de tipos de bases diferentes que possuem valores de centenas de *Gigabytes* ou dezenas de *Terabytes*. O processo de ETL a cada minuto ou hora se torna inviável, sendo normalmente agendado um dia específico para a extração de todas as bases. Do momento da extração, até a geração da lista de inconsistências, existe um delta que pode ser maior do que a validade daquela informação. Ou seja, a lista de inconsistência pode se tornar obsoleta em questão de segundos [**7**, **12**].

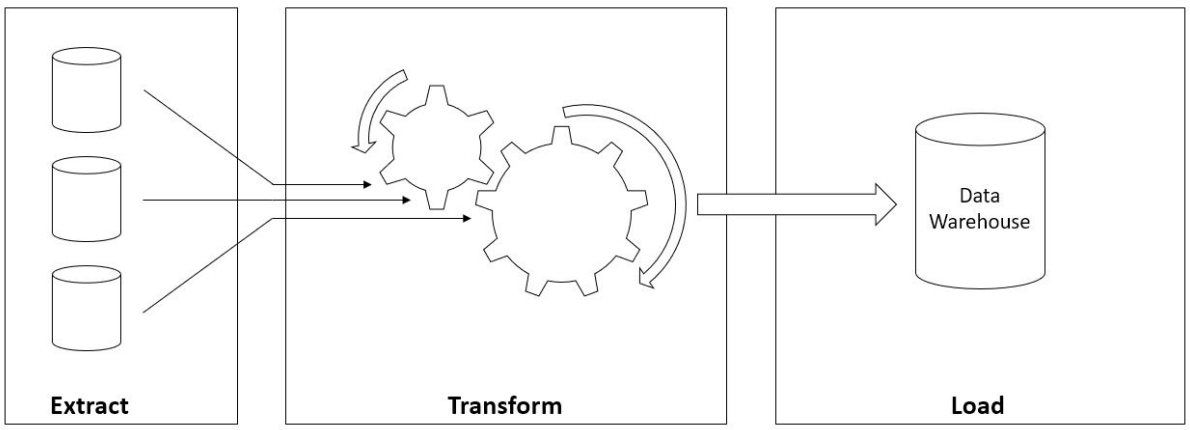

Figura 2.1: Diagrama do Fluxo ETL.

Esse é um dos principais motivos de ser um processo com alto custo e ineficiente, por mais que se corrija uma determinada causa raiz do problema que gera as inconsistências, outros problemas podem surgir e a correção estará sempre um passo atrás da falha. O problema encontrado então leva a formulação da nova proposta neste trabalho, correção das inconsistências em quase tempo real por meio de uma nova arquitetura que dispense o processo de auditoria com ETL e *Data Warehouse* com grandes atrasos e grande quantidade de servidores paralelos [12].

De fato, a determinação de quais alterações nas bases geram inconsistências, ou não, deve se basear em regras de negócios e validações cruzadas. Algumas regras podem não ser absolutas e quanto mais bases existirem, mais complexo é o problema.

Por exemplo, supondo que um parâmetro é compartilhado por três bases distintas, como código do *simcard* do cliente. As três bases envolvidas são uma base de tarifação em BSS, uma base HLR para redes 2G/3G e uma base HSS - *Home Subscriber Service* para rede 4G. O cliente se dirigiu a uma loja para substituir um *simcard* por roubo e a base de tarifação em BSS recebeu uma ordem da loja para troca. Supondo que ocorra uma falha de atualização em uma das bases de rede, no caso um HLR, por queda de comunicação. Então, o sistema, recebendo o código de erro da transação, realiza um retorno para código de *simcard* anterior, finalizando com BSS e HLR com *simcard* antigo e HSS com *simcard* atualizado. O assinante deixa a loja e já troca o *simcard* para o novo código, aguardando a liberação do serviço. A base de rede HSS consegue perceber que as tentativas de registro estão sendo realizadas via *simcard* novo, e atualiza o registro em sua base com o horário das tentativas. Pela melhor experiência do usuário, para não gerar a necessidade um retorno à loja, o ideal seria considerar como "verdade"o dado com última atualização, no caso o HSS. Assim o sistema poderia tentar novamente a atualização da base de tarifação e HLR, quando a comunicação retornasse.

Esse último exemplo seria um caso em que a regra de negócio pura e simples, tendo no BSS a base principal, não atenderia. Seria necessário realizar outras validações. No caso, poderia ser utilizado o horário da última tentativa de registro, e atribuir à base com o dado mais recente a função de base principal frente às demais bases.

Em outros casos, pode ser necessário a atribuição de pesos para definir qual base será escolhida como principal. Cada parâmetro combinado pode gerar um valor ponderado que determinará um fator de qualidade do dado que representaria probabilidade de certo dado ser o dado real ou que trará menor impacto ao negócio. Também podem ser aplicadas estratégias de Inteligência Artificial e Aprendizado de Máquina cruzando informações de reclamações, KQIs - *Key Quality Indicators* e KPIs - *Key Performance Indicators* para definir quais bases devem ser as principais e quais dados devem ser corrigidos, definindo um fator de qualidade global [4]. Situações mais complexas de definição de regras de negócio e ações dinâmicas fogem do escopo deste trabalho e poderão ser aprofundadas em pesquisas futuras.

Algumas pesquisas sobre tratamento em tempo real de bases de telecomunicações já foram realizadas, porém sempre focando na garantia de receita ou monetização de clientes. É o caso da pesquisa realizada na TCS Innovation Labs, da empresa indiana de telecomunicações Tata. O foco é no tratamento em tempo real de registros de chamadas (CDR) entre as centrais de comutação de voz (MSC) e a área de tarifação. Utilizam para esse tratamento um produto da empresa IBM, Infosphere Streams, que dispões de uma interface de programação para definição de *data flows* e ferramentas de visualização [**13**].

No caso do estudo realizado em 2013, a análise também se baseou em CDRs, porém visando a análise comportamental dos clientes, selecionando potenciais clientes para novas promoções. Apesar de ser um tratamento em tempo real, concentrou-se somente em um tipo de base, a base de CDRs [**14**].

Em outro estudo realizado em 2005, o foco foi em integrações de sistemas de TI e a integridade dos sistemas e dados. Citando a integridade de dados como um pilar da Segurança de Informação. Frisando que, se as informações e dados são consumidos em tempo real pelos sistemas, então, também é necessário uma auditoria e controle em tempo real para garantia dos serviços da empresa como um todo [**15**].

A nova arquitetura é proposta orientada a eventos, fundamentada em serviços de mensagens com tratamento de registros (*logs*). Na arquitetura orientada a eventos, algo notável ocorre no ambiente de negócios, gerando um evento, que dispara processos dentro e fora do sistema, o evento então é avaliado pelas partes interessadas que tomam a decisão de seguir com uma próxima ação [**16**]. Enquanto as informações existentes nos bancos de dados representam somente o último estado do dado, com a análise dos *logs* e eventos é possível analisar e visualizar todo o ciclo de vida dos dados e determinar quando uma ação de criação, alteração ou deleção de dados é correta ou gerará uma inconsistência.

O principal estudo que se aproxima da proposta deste trabalho, porém fora da área de aplicação de telecomunicações, traz o conceito de OLEP - *Online Event Processing* em contraste ao OLTP - *Online Transaction Processing* e OLAP - *Online Analytics Processing*. A arquitetura baseada em OLEP permite que os sistemas alcancem consistência, alta performance, tolerância a falhas e escalabilidade. O OLEP privilegia os *logs* frente às transações como modelo de gerenciamento de dados. Essa arquitetura só foi idealizada graças à popularização dos modelos de sistemas *publish/subscribe* como o Apache Kafka [**9**].

CAPÍTULO 3

Sistemas Distribuídos

Para melhor compreensão do problema abordado, se faz necessário aprofundar no assunto de Sistemas Distribuídos. Os Sistemas de Informação em geral podem ser classificados de diferentes formas, foi utilizada a classificação quanto a distribuição, heterogeneidade e autonomia [17]. A distribuição é a capacidade de se ter dados e processamento na mesma máquina ou em máquinas distintas. A heterogeneidade considera todas as tecnologias diversas que podem ser utilizadas dentro dos sistemas, como softwares de gerenciamento, linguagens de programação, sistemas operacionais, etc. A autonomia diz sobre o grau de coordenação e regulação dos elementos no sistema. A Figura 3.1 mostra possíveis classificações de sistemas.

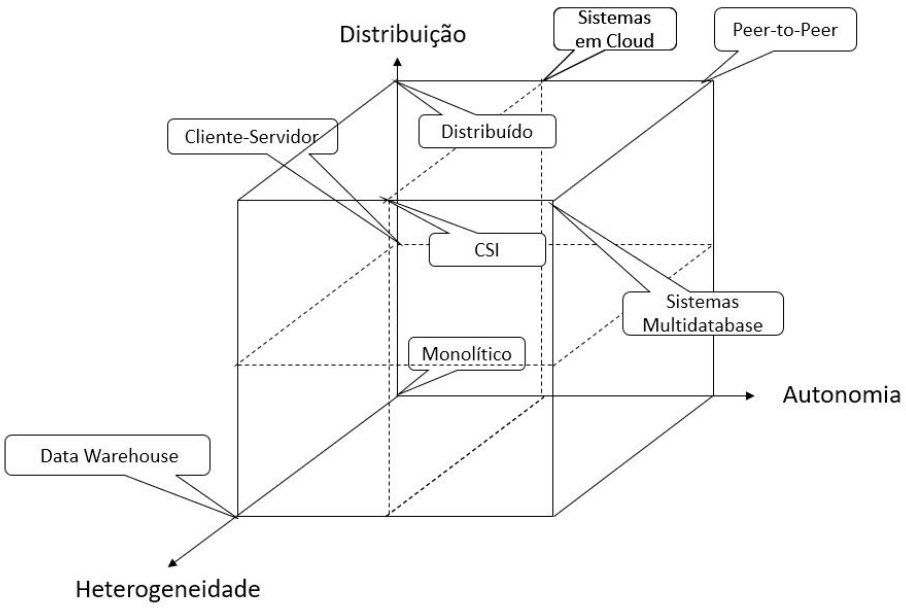

Figura 3.1: Classificação dos sistemas de informação.

Nos sistemas monolíticos, todos os dados e processamento estão localizados no mesmo componente computacional. Com todos os dados no mesmo ambiente, há uma facilidade maior em se ter o controle de qualidade e consistência. Os formatos e esquemas de dados, gerenciadores de banco de dados e fornecedores são os mesmos. Portanto, os custos de operação e complexidade costumam ser menores do que em outros sistemas.

Um exemplo de sistemas heterogêneos, porém não distribuídos, são os *Data Warehouses*. São repositórios em comum para armazenagem de bases heterogêneas extraídas de vários sistemas e são utilizados para análises de negócio e controle de qualidade de dados em muitas empresas. Os problemas mais comuns em sistemas heterogêneos residem na integração de diferentes bases e limpeza de dados para armazenagem somente dos dados que serão efetivamente utilizados.

Os sistemas distribuídos removem as restrições locais dos sistemas monolíticos, o que permite que os recursos e dados das aplicações sejam distribuídos geograficamente pela rede para outras máquinas. Eventualmente podem também apresentar certa autonomia ou heterogeneidade. Um sistema intermediário entre monolítico e distribuído é o cliente-servidor, que apresenta uma separação de funções que permite o desacoplamento dos elementos.

O CIS - *Cooperative Information System* é um exemplo de um sistema heterogêneo, distribuído e de certa forma autônomo. O CIS pode ser definido como um sistema de informação de larga escala que interconecta vários sistemas de diferentes organizações autônomas compartilhando algum objetivo em comum. Em alguns casos, são utilizadas integrações de dados virtuais, em que um único esquema de dados virtual é elaborado para integrar os dados virtualmente e apresentar as informações de forma unificada. Essa técnica é afetada quando há inconsistências de dados em diferentes pontos do sistema, pois podem impactar diretamente na apresentação da informação integrada.

Outros exemplos de sistemas distribuídos são os sistemas em nuvem (*cloud*) que são um grupo de servidores remotos organizados para permitir um acesso centralizado a armazenamento e elementos computacionais; os sistemas ponto-a-ponto (*peer-to-peer*) que são totalmente distribuídos e autônomos, não necessitam de uma coordenação centralizada; até os sistemas com múltiplas bases, presentes na maior parte das grandes empresas, com bases e sistemas heterogêneos, distribuídos e autônomos tendo que se relacionar de acordo com as regras de negócio em prol de um mesmo objetivo [4].

Como os sistemas distribuídos removem restrições que existiam antes nos sistemas monolíticos, novos problemas passam a ocorrer. Algumas funções podem ser necessárias para o correto funcionamento do sistema, como por exemplo um serviço de trava (*lock service*) para permitir o sincronismo de atividades entre os servidores, compartilhamento das suas configurações e estados básicos, bem como a eleição de um líder no sistema e apresentação de um *name space*. Uma ferramenta pioneira foi desenvolvida pela Google denominada

Chubby para atender as demandas de um *lock service* com alta disponibilidade e resiliência [18].

3.1 Apache Zookeeper

Posteriormente, a empresa Yahoo! desenvolveu a ferramenta Zookeper, inicialmente, para atender a demanda de escrever continuamente eventos dos processos de seus conjuntos Big Data em arquivos de *log*. O Zookeeper nasceu dentro do projeto Hadoop, focado em processamento distribuído de Big Data, porém depois de algum tempo se tornou um projeto de código aberto independente dentro da estrutura da fundação Apache.

Zookeeper é muito mais que apenas um *lock service* distribuído. É um serviço distribuído de alta disponibilidade, escalável, de configuração, consensual, de membros de grupo, de eleição de líder, de nomeação e de coordenação.

O Zookeeper trabalha com um protocolo próprio para comunicação entre servidores. Cada cliente no sistema se conecta a um servidor Zookeeper e os servidores se encarregam de se comunicar via protocolo ZAB - *Zookeeper Atomic Broadcast* e manter as informações confiáveis e resilientes. Caso um servidor fique indisponível, os clientes conseguem estabelecer novas conexões com outro servidor do conjunto e manter a transação que estavam executando. O sistema o tempo todo está se comunicando para informar qualquer alteração e manter a ordem de execução de tarefas [19]. A Figura 3.2 mostra um esboço do funcionamento do Zookeeper.

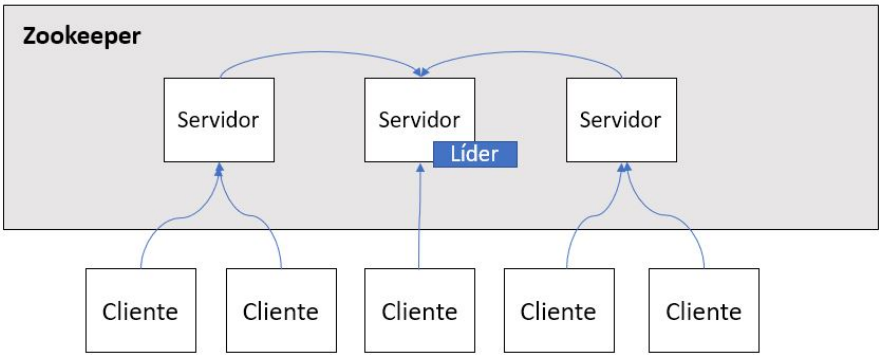

Figura 3.2: Funcionamento do Zookeper.

Atua principalmente na resolução de conflitos em situações de concorrência entre aplicações, disponibilizando diversas funcionalidades para sistemas distribuídos dispensando cada aplicação da necessidade desta implementação. Permite que os processos se coordenem entre si a partir de um espaço de nomes hierárquico de registros de dados compartilhado, chamados *znodes*, como um sistema de arquivos. Porém, ao contrário dos sistemas de arquivos

tradicionais, o Zookeeper provê alta taxa de transferência, alta disponibilidade, baixa latência e acesso estrito aos *znodes*. Estas características são possíveis devido ao armazenamento em memória da árvore de dados compartilhados entre todas os servidores. Todos as alterações devem seguir a ordem exata, recebendo um identificador para cada atualização, chamado *zxid*. Toda leitura realizada é respondida com o marcador do último *zxid* do sistema àquele momento.

Atualmente, o Zookeeper é uma ferramenta amplamente adotada no mercado, empresas como Facebook, Google, Amazon, Huawei, IBM, Microsoft e Red Hat suportam diretamente o desenvolvimento do projeto e a comunidade envolvida. Diversos outros projetos usam a ferramenta como base para desenvolvimento de suas funcionalidades:

- Apache Hadoop;
- Apache HBase;
- Apache Hive;
- Apache Kafka;
- Apache NiFi.

Dos projetos de código aberto citados, o Apache Kafka e Apache NiFi, se destacam para o cenário proposto, pois são ferramentas muito utilizadas para tratar de fluxos de dados.

3.2 Apache Kafka

O Apache Kafka foi originalmente desenvolvido no Linkedin para resolver o problema de processamento de *logs* em tempo real e depois foi transformado em um projeto de código aberto. Como já abordado, os *logs* possuem maior informação agregada do que o último estado do dado no banco de dados, portanto são um habilitador para tratamento em tempo de real de transações em bases distribuídas.

O Apache Kafka é uma plataforma de fluxo (*streaming*) de eventos. Permite que publicar (escrever) e assinar (ler) fluxos de eventos em tópicos, armazenar os fluxos de forma confiável e durável pelo tempo que se desejar, processar fluxos de eventos conforme ocorrem ou retroativamente. Todas as funcionalidades são providas por um serviço distribuído, altamente escalável, elástico, tolerante a falhas e seguro, se utilizando da robustez do Apache Zookeeper. Frente a outras aplicações que realizam serviço semelhante como RabbitMQ e ActiveMQ, o Kafka demonstrou desempenho extremamente mais eficiente [**20**].

Dentre as empresas que utilizam o Apache Kafka em sua estrutura estão: Uber, Firefox, Itaú, Oracle, PayPal, Twitter, Spotify, Netflix, AirBnB e Cisco. O Kafka utiliza o serviço Zookeeper, esquematizado na Figura 3.3, como base para seu serviço.

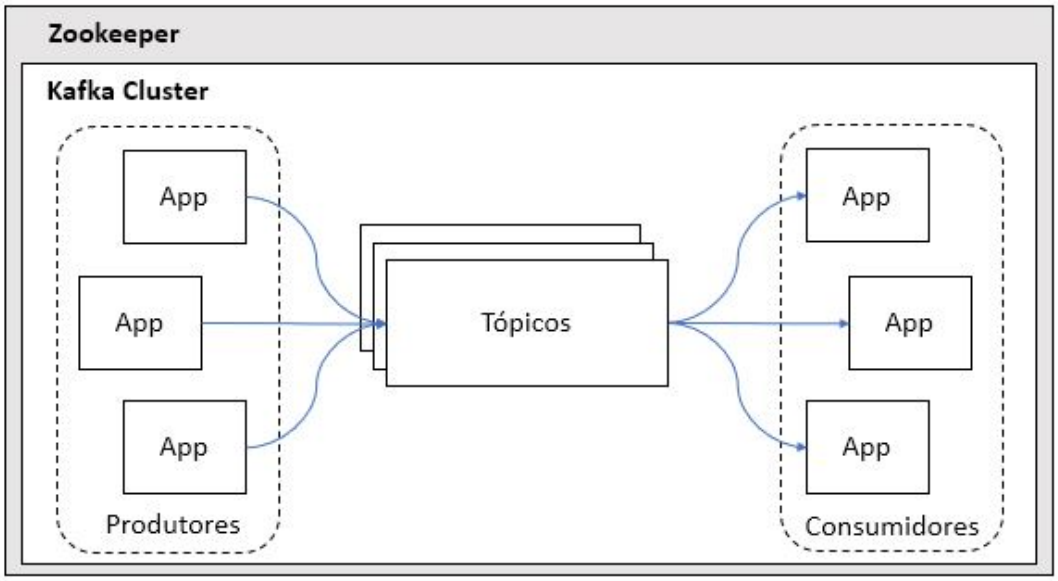

Figura 3.3: Exemplo da estrutura Apache Kafka.

3.3 Apache NiFi

O Apache NiFi é um projeto de código aberto para automatizar e gerenciar o fluxo de dados entre sistemas com suporte gráfico. Foi baseado inicialmente no software *Niagara Falls* da Agência de Segurança Nacional dos Estados Unidos (NSA) e foi transformado em código aberto em 2014.

Dos desafios presentes em sistemas distribuídos o Apache NiFi pode auxiliar nas falhas de sistemas, limite excedido na capacidade de acesso aos dados, diferentes taxas de transmissão entre sistemas, dados de diferentes formatos e tamanhos, segurança e melhoria contínua. Os benefícios do Apache NiFi são:

- Criação visual e gerenciamento gráfico dos processadores;
- Assincronismo intrínseco que permite altas taxas de vazão e amortização mesmo quando taxas do fluxo variam;
- Dispõe de um modelo com alta capacidade de concorrência sem que o desenvolvedor tenha que se preocupar com complexidades da concorrência;
- Promove o desenvolvimento de componentes coesos e levemente acoplados que podem ser reutilizados em outros contextos e em testes unitários;
- Tratamento de erros se torna muito mais simples;
- Os pontos de entrada e saída de dados tornam-se bem conhecidos e como o fluxo se comporta é facilmente verificado.

Possui normalmente um arquivo de entrada chamado *FlowFile* que então é repassado continuamente, conforme novos dados são populados, aos processadores e conectores. Cada trecho também possui uma fila configurada para aguardar as ações do processador, possibilitando assim a execução do fluxo desejado de forma contínua em tempo real. NiFi é executado na máquina virtual Java (JVM) em um sistema operacional hospedeiro, no caso o Linux. Os principais componentes do NiFi são executados via JVM, conforme Figura 4.4. O Web Server disponibiliza a interface gráfica com o usuário e os repositórios se comunicam com o sistema operacional.

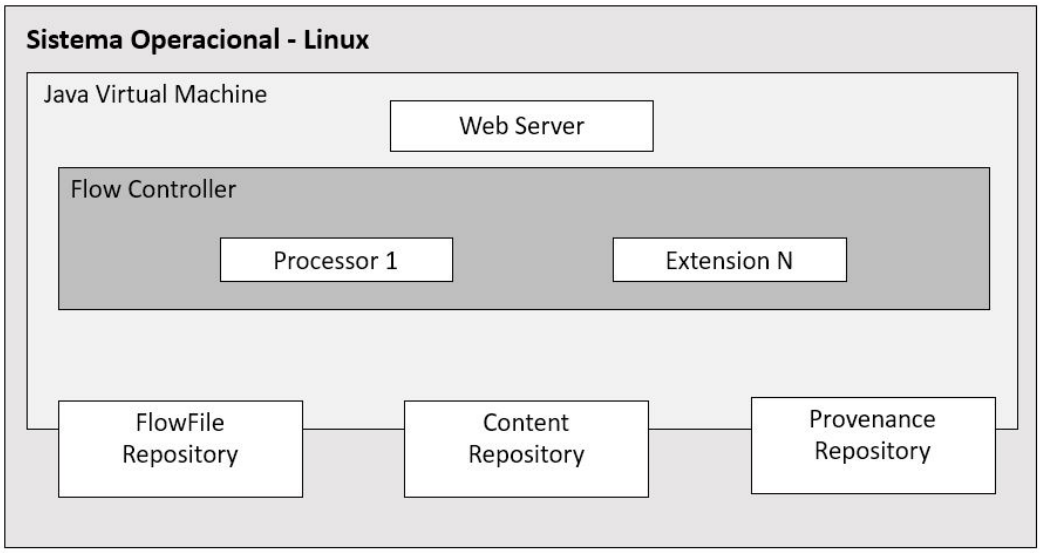

Figura 3.4: Exemplo da estrutura Apache NiFi.

É possível também executar o Apache NiFi em *cluster* e, neste caso, o Nifi executa as funções com o apoio do Apache Zookeeper, porém neste trabalho não foi utilizado tal formato.

CAPÍTULO 4

Metodologia

O problema proposto foi modelado como um sistema distribuído não-heterogêneo, mantendo o mesmo formato de esquema de dados nas máquinas envolvidas. Para a comparação dos diferentes meios de se realizar uma garantia de qualidade e consistência de dados, foi realizada uma prova de conceito em menor escala com três nós computacionais. Um nó representando as informações principais e os outros dois nós recebendo a replicação das alterações feitas no nó principal.

As ferramentas utilizadas para simulação foram o ambiente de *Cloud Computing* da Amazon Web Services, com máquinas virtuais Linux Red Hat, aplicações GGenerator, Apache Zookeeper, Apache Kafka e Apache NiFi. Foram escolhidas as ferramentas Apache pelo critério de ampla adoção no mercado, por possuírem código aberto e forte comunidade de suporte e desenvolvimento.

O *Cloud Computing* foi escolhido por ter alta disponibilidade e ser acessível de qualquer local do mundo com acesso a internet. Cada nó foi configurado em uma máquina virtual localizada na nuvem pública da AWS na região *sa-east-1* (São Paulo), utilizando os recursos de camada gratuita e créditos da parceria para estudantes do GitHub Student Developer Pack (https://education.github.com/pack). As máquinas virtuais foram configuradas no serviço EC2 - *Elastic Cloud Compute* e possuíam configuração idêntica do tipo *t2.medium* - 2 vCPUs, 4 GBs de memória RAM e 20GB de armazenamento. O sistema operacional escolhido foi o Red Hat Enterprise Linux 8 (HVM) de 64 bits. As instâncias podem ser visualizadas na Figura 4.1.

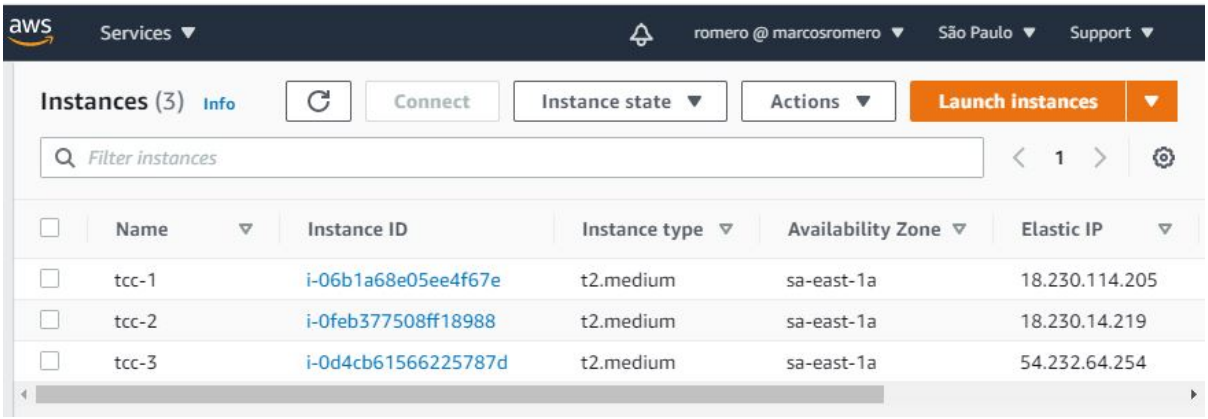

Figura 4.1: Painel de controle das instâncias na AWS.

Foram estudados três modelos distintos para garantir a qualidade dos dados alterando a arquitetura dos sistemas, descritos a seguir.

4.1 Primeira Arquitetura - Modelo Tradicional

A primeira arquitetura proposta, demonstrada na Figura 4.2, tentou simular o aprovisionamento padrão realizado entre sistemas BSS e elementos de Rede. Uma alteração em uma certo dado gera a replicação do dado nas demais plataformas de forma síncrona e sem mecanismo alternativo para falhas, ou seja, em caso de erro durante o processo, o sistema simplesmente armazena um *log* de erro e não executa o comando e nem faz outra tentativa. Com os erros, simulação de queda de link, inconsistências foram geradas. As inconsistências foram tratadas de forma *offline*, exportando as três bases envolvidas e realizando comparações com comandos Linux *comm*. Uma lista de inconsistências foi gerada e encaminhada para correção.

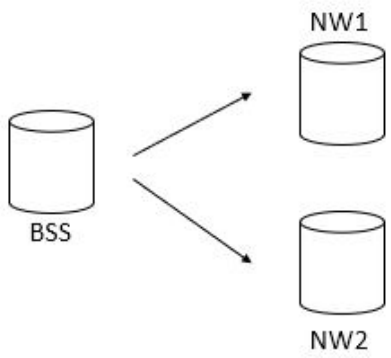

Figura 4.2: Arquitetura 1: proposta com sistema tradicional.

Os principais programas foram escritos em Shell para melhor interação com o sistema

operacional em mais baixo nível e estão no Anexo I, bem como os arquivos de configuração. Para geração dos dados aleatórios utilizados nos experimentos foi utilizada uma ferramenta de código aberto chamada GGenerator desenvolvida pela empresa Datenworks. Para a primeira e segunda arquiteturas, o algoritmo principal utilizado na máquina 1 - BSS é demonstrado no pseudocódigo Algoritmo 1. É gerada uma linha aleatório com o GGnerator para substituir a em uma posição também aleatória, primeiro se tenta alterar a base local na máquina 1, caso tenha sucesso tenta efetuar a troca nas demais máquinas, tanto em caso de sucesso ou em caso de falha, o resultado é registrado.

Algoritmo 1 Provisionador

```
Result: Realiza a troca de uma linha aleatória na máquina local e tenta realizar a troca nas 2
        máquinas remotas
gera nova linha aleatoriamente via ggenerator
tenta substituir linha no arquivo local if linha foi substituída then
    registra resultado OK máquina 1
    tenta substituir linha no arquivo da máquina 2 if linha foi substituída then
        registra resultado OK máquina 2
    else
        registra resultado FAIL máquina 2
    end
    tenta substituir linha no arquivo da máquina 3 if linha foi substituída then
        registra resultado OK máquina 3
    else
        registra resultado FAIL máquina 3
    end
else
    registra resultado FAIL máquina 1
end
```

Para a geração das falhas nas máquinas remotas foi utilizada a lógica descrita no Algoritmo 2, um número aleatório entre 1 e 10 é gerado, caso o número seja igual a 1 (probabilide de 10%) a máquina entra em estado de falha, na próxima execução do programa, outro número aleatório é gerado, se for 1 continua em falha, se for diferente de 1 retorna para o estado ativo. Ambos algoritmos são iniciados via agendamento no Linux *crontab* a cada minuto.

Algoritmo 2 Falha

```
Result: Simula a falha de acesso ao arquivo de aprovisionamento
Input: numero
numero ← rand(1..10)
if numero = 1 then
    altera arquivo de aprovisionamento para off
else
    altera o arquivo de aprovisionamento para on
end
```

4.2 Segunda Arquitetura - Proposta de Modelo Intermediário

A segunda arquitetura, demonstrada na Figura 4.3, simula um sistema legado, com funcionamento similar à primeira, porém insere um componente de inteligência baseado no software para fluxo de dados Apache NiFi. Esse componente realiza a leitura dos *logs* e sempre que encontra um registro de falha tenta realizar a correção da inconsistência assim que possível, no momento em que o sistema se recuperar da falha que causou o erro inicial.

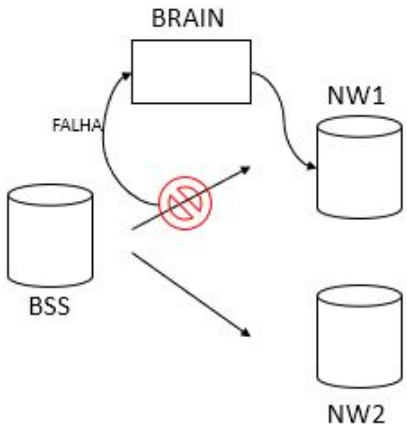

Figura 4.3: Arquitetura 2: proposta com sistema inteligente.

O fluxo de dados utilizado na Arquitetura 2, apresentado na Figura 4.4, realiza a leitura do arquivo de *logs* com o Apache NiFi e separa as linhas com falha entre a máquina 2 e 3, acionando em sequência um provisionador individual, descrito no Algoritmo 3, para tratar exclusivamente das falhas registradas. Enquanto a primeira falha de determinada máquina não é resolvida, qualquer nova falha desta máquina é enfileirada para tratamento quando possível. Como na Arquitetura 2, dois processos, o Provisionador e o Provisionador Individual, podem estar ativos ao mesmo tempo, eventualmente o acesso ao arquivo pode concorrer. Para tal, foi criado um arquivo de trava para garantir que somente um processo faça escrita e leitura por vez.

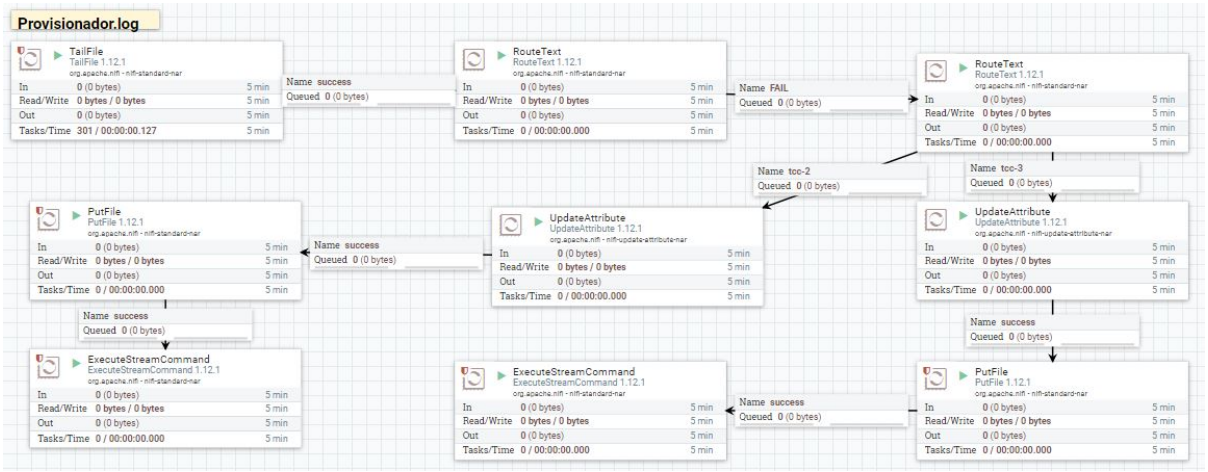

Figura 4.4: Fluxo de Dados construído no Apache NiFi.

Algoritmo 3 Provisionador Individual

Result: Recebe uma linha de falha e tenta realizar a troca as máquina indicada
Input: $linha$, $hostname$
if $hostname = maquina2$ **then**
    while *linha não foi substituída* **do**
        tenta substituir $linha$ no arquivo da máquina 2
        if *linha foi substituída* **then**
            registra resultado OK máquina 2
            continue
        else
            registra resultado FAIL máquina 2
        end
        espera 5 segundos
    end
end
if $hostname = maquina3$ **then**
    while *linha não foi substituída* **do**
        tenta substituir $linha$ no arquivo da máquina 3
        if *linha foi substituída* **then**
            registra resultado OK máquina 3
            continue
        else
            registra resultado FAIL máquina 3
        end
        espera 5 segundos
    end
end

4.3 Terceira Arquitetura - Proposta de Modelo Orientado a Eventos

A terceira arquitetura, apresentada na Figura 4.5, redesenhou o sistema todo de aprovisionamento, com a instalação dos componentes Apache Zookeeper e Apache Kafka nos três nós. O nó principal inseriu todos os comandos em um tópico Kafka único e os nós secundários se inscreveram no tópico para consumir os comandos na ordem exata, porém cada um em um grupo consumidor diferente. Assim, mesmo com uma falha de um dos nós, a fila do tópico ainda estaria disponível para consumo quando a falha do elemento cessasse.

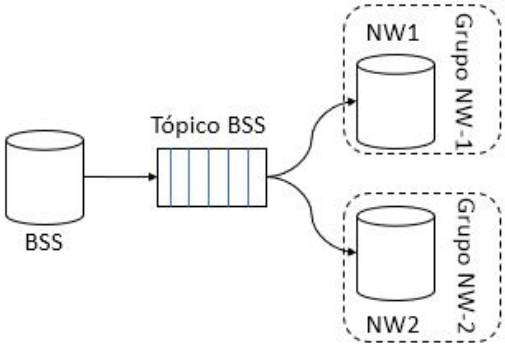

Figura 4.5: Arquitetura 3: proposta com Apache Kafka.

Para os cenários de simulação foi utilizado um arquivo texto para emular a base de dados. O arquivo continha em cada linha três parâmetros: *ID*, *MSISDN* (número do telefone) e Plano. Cada linha representava uma entrada de um cliente. Foram realizadas duas rodadas de simulação em cada arquitetura, ambas com duração de duas horas cada. Na primeira rodada foi utilizado um arquivo de dez milhões de linhas e na segunda rodada um arquivo com mil linhas. Os arquivos eram sempre copiados no início da simulação para os nós secundários, para que no instante inicial não houvesse nenhuma inconsistência entre os três nós. Todo comando de troca era registrado em um arquivo de *log* com o resultado *OK* ou *FAIL*. A cada minuto os nós secundários rodavam o programa gerador de falha com probabilidade de 10% para entrar em um estado de falha e 90% para se recuperar de um estado de falha.

Após as duas horas de simulação, os arquivos foram todos transferidos para um mesmo diretório na máquina 1 e então foi realizada análise e verificação dos arquivos com comandos de comparação entre os três arquivos para identificar eventuais inconsistências e seguir com a correção. A replicação não foi paralisada para simular possíveis conflitos no momento de correção com a replicação em curso.

Foi realizada a comparação dos três sistemas quanto a dois parâmetros objetivos, o tempo entre a falha de inconsistência e o tempo de correção e a acurácia da correção aplicada.

CAPÍTULO 5

Resultados e Discussão

Após as simulações, foram elaborados alguns gráficos para melhor visualização de como as três arquiteturas se comportaram durante as duas horas de atuação em cada cenário. A Figura 5.1 mostra um gráfico contendo o tempo de indisponibilidade em minutos das máquinas durante as seis simulações, duas em cada arquitetura, sendo a primeira com arquivo de mil linhas e a segunda com dez milhões de linhas. Como o tempo de indisponibilidade está relacionado com a chance de falha, configurada para 10%, os valores se mantêm semelhantes, em média 11 minutos.

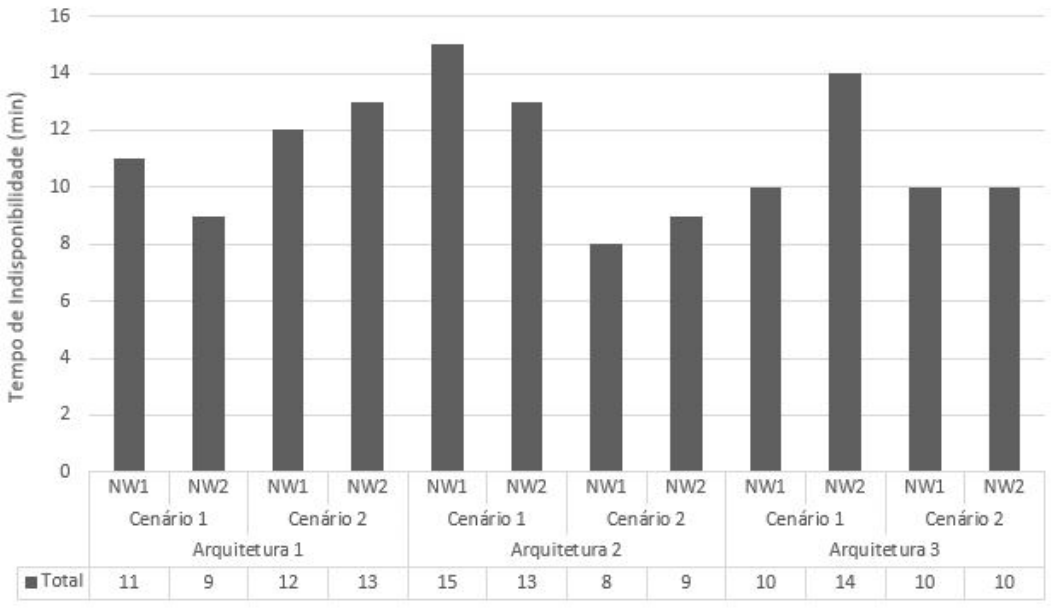

Figura 5.1: Tempo de Indisponibilidade em minutos para cada máquina e arquitetura.

Na Figura 5.2 é possível observar as inconsistências encontradas ao final da simulação após a análise e comparação para cada arquitetura e cenário. Pode-se observar que, como não há inconsistências ao final da simulação, as arquiteturas 2 e 3 resolvem o problema de inconsistências durante a simulação e eliminam a necessidade de batimentos esporádicos como ocorre na arquitetura 1. Como as inconsistências são geradas durante a indisponibilidade das máquinas e cada máquina recebe um comando por minuto, é de se esperar que as inconsistências, se não tratadas durante a simulação, sejam iguais aos tempo em minutos de indisponibilidade.

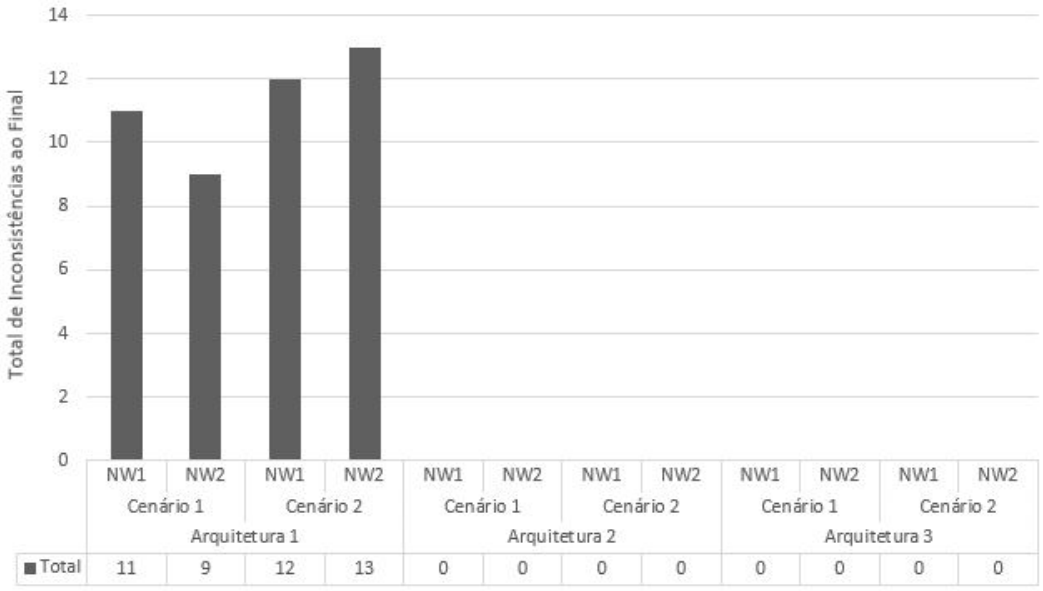

Figura 5.2: Inconsistências ao final para cada máquina e arquitetura.

No gráfico apresentado na Figura 5.3 é possível observar que os tempos de correção para as falhas da primeira arquitetura são bem expressivos, pois dependem de uma ação externa de batimento para início da correção. No pior caso, o tempo de correção foi superior a 100 minutos, visto que a simulação foi de 120 minutos. Ou seja, se o ciclo de batimentos for realizado a cada 120 minutos, as falhas logo após um batimento podem ter um tempo de impacto próximo de 120 minutos. Para solucionar o problema do tempo de impacto para cada falha, poderia ser proposto um batimento em tempo menor, porém vale ressaltar que em cenários maiores, o batimento é um trabalho que demanda paralisação de alguns ambientes e grande processamento computacional, eventualmente inviabilizando ciclos com tempos menores.

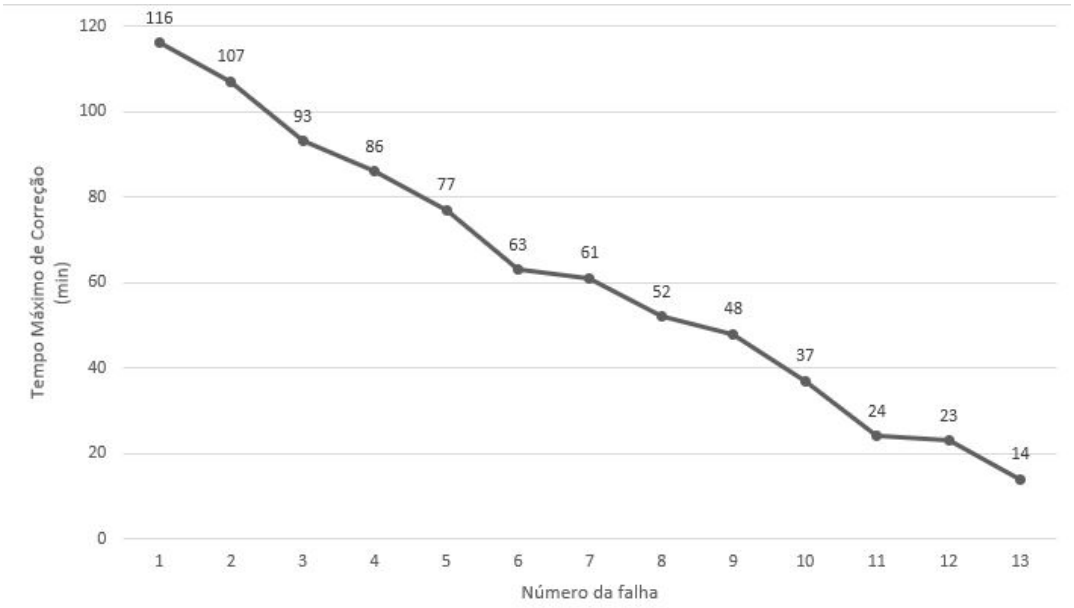

Figura 5.3: Arquitetura 1: Tempo de correção em minutos de cada falha.

Nas Figuras 5.4 e 5.5 é possível verificar que o tempo de correção é praticamente em tempo real, assim que o sistema se torna disponível novamente, as correções já são executadas. Ou seja, o tempo de correção depende somente do tempo de indisponibilidade do nó. Como na simulação foi o evento de falha ou disponibilidade só alterava a cada minuto, então os tempos se mantiveram em torno de 1 a 2 minutos para correção. A exceção dos primeiros eventos da arquitetura 3 na máquina 3 acabaram levando os números para cima, houve uma demora na inicialização do *cluster* Apache Kafka. Mas para as demais falhas que ocorreram, o sistema se recuperou em 1 ou 2 minutos também.

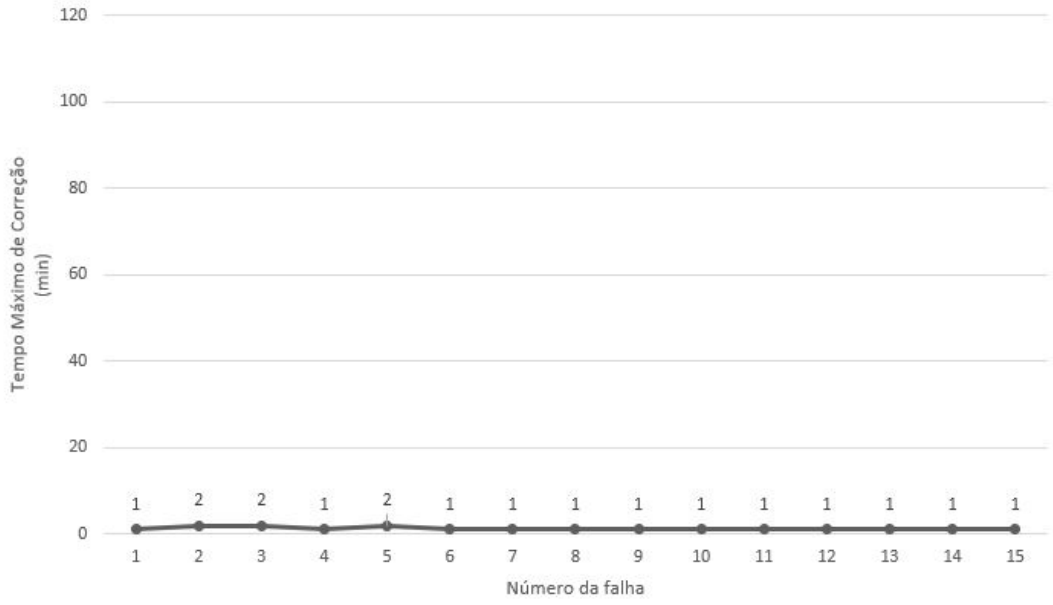

Figura 5.4: Arquitetura 2: Tempo de correção em minutos de cada falha.

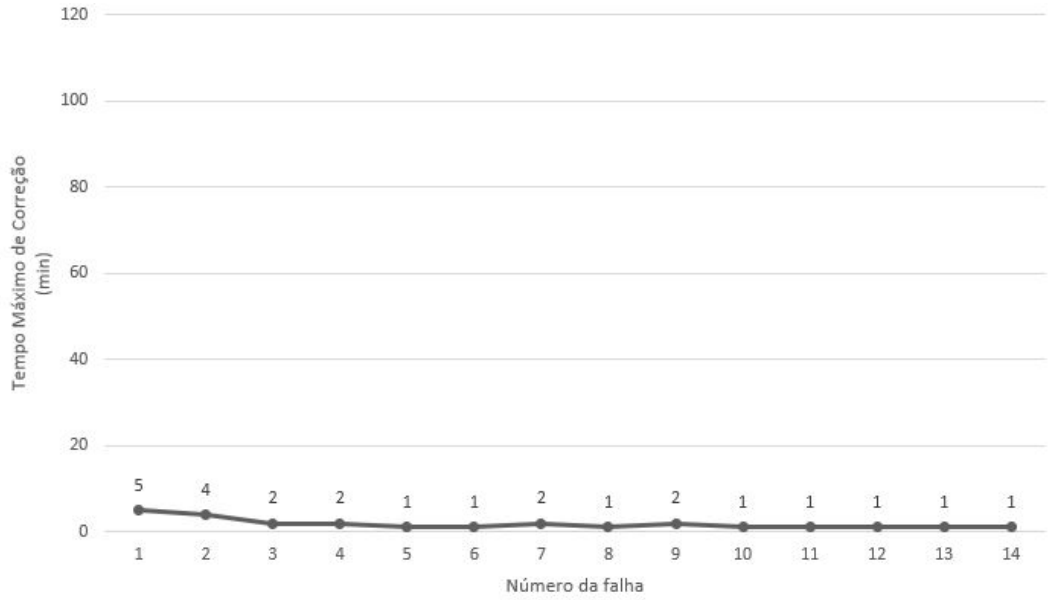

Figura 5.5: Arquitetura 3: Tempo de correção em minutos de cada falha.

Quanto à acurácia da correção, ou seja se o valor corrigido foi o que realmente deveria ser aplicado, foi observado que em todos as arquiteturas foi possível realizar a correção. Porém cabem algumas observações se forem comparados o que foi simulado com sistemas mais próximos da realidade. Em alguns sistemas, existe a dependência de um valor anterior para se permitir a troca dos parâmetros, ou seja, caso um parâmetro "A"seja alterado para "B"e depois para "C"na máquina principal, e durante estas trocas não haja a replicação para a máquina secundária por falha, a correção possivelmente indicada seria "C"em um eventual batimento, porém a troca de "A"para "C"pode ser proibida pela regra de negócio, gerando uma impossibilidade de correção.

Outro ponto importante que pode afetar a acurácia e correção é a janela de tempo entre o batimento e disponibilização da lista com as correções até a correção efetiva. Quando há a necessidade de validar a lista de correções antes da aplicação efetiva nas bases, pode ser demandado um tempo longo, e com o sistema ainda em funcionamento, a correção a ser aplicada pode estar defasada do que seria o correto para o momento. A característica da volatilidade dos dados passa a ser um considerável ponto de falha. Quanto mais se demorar para realizar o batimento das informações, mais podem ter falhas com grande tempo de correção. E quanto mais se demorar para realizar a correção após o batimento, maior a probabilidade de se ter erros gerados pela correção. Em grandes sistemas esses pontos podem se tornar cada vez mais problemáticos e chegar a um ponto em que as bases nunca alcançam a consistência total.

Tabela 5.1: Vantagens e desvantagens de cada arquitetura.

Arquitetura	**Vantagens**	**Desvantagens**
Arquitetura 1 - Tradicional	Simples, menor tempo de projeto e implementação	Maior tempo para correção, necessidade de batimento, suscetível à volatilidade
Arquitetura 2 - NiFi	Integrável com o sistema tradicional, correção em tempo real, fácil entendimento do fluxo de dados de forma gráfica	Concorrência da aplicação de correção com a aplicação online de provisionamento
Arquitetura 3 - Kafka	Correções sempre que o sistema se torna disponível, alta disponibilidade	Arquitetura nova, tempo de projeto maior e mais complexo, maior consumo de processamento

A arquitetura 1 possui a vantagem de ser a arquitetura mais simples, portanto a que demandaria menor tempo de projeto e implementação. Porém as desvantagens quanto à qualidade e consistência dos dados são praticamente impeditivas para sistemas de operação crítica, que seja extremamente sensível a falhas, como sistemas médicos, de telecomunicações ou controles aeroespaciais.

A arquitetura 2 foi projetada para atender uma demanda de um sistema que já esteja na arquitetura 1, tradicional, porém que possua pelo menos registros de falhas e necessite de uma melhoria quanto ao tratamento das falhas em tempo real. Nos sistemas distribuídos e heterogêneos, com diferentes tipos de estruturas de dados e fornecedores diferentes, pode ser inviável reescrever a arquitetura interna do sistema, instalar aplicações e alterar o funcionamento. Por isso, a arquitetura 2 simula um agente externo que trabalha somente com o que a arquitetura 1 já oferece, que são os *logs*. Um agente externo é capaz de fazer a leitura, identificar as falhas e acionar um processo externo de correção, assim que os nós com falha se recuperam. Com o agente externo realizando o tratamento de falhas em tempo real, o sistema aumentou profundamente a qualidade e consistência dos dados já na arquitetura 2.

A desvantagem observada na arquitetura 2, até mesmo durante a execução dos testes iniciais das simulações, é a existência de concorrência do processo de replicação principal e o processo externo de correção, sendo necessária a inclusão de uma trava para permitir que somente um processo atue no arquivo por vez. Como já é extensamente discutido na literatura de Sistemas Distribuídos e Computação Paralela, o problema de concorrência pode gerar casos de *Starvation* ou *Dead Lock*. No caso da simulação executada, como se tratava apenas de uma prova de conceito, não ocorreu nenhum dos problemas com o uso da trava,

porém para sistemas mais complexos, com mais componentes, com maior probabilidade de falha ou maior duração da indisponibilidade, essas situações passam a ser mais prováveis.

A arquitetura 3 também foi capaz de aumentar significativamente a consistência e tempo de correção em relação a arquitetura 1, porém envolve uma alteração mais profunda em todo o sistema. Como necessita da instalação e configuração do Apache Zookeeper e Apache Kafka, para sistemas já em produção pode se tornar inviável, sendo recomendado, portanto, que seja elaborado ainda em tempo de projeto para que o proveito da nova arquitetura seja maior. Em relação a arquitetura 2, houve uma melhoria, eliminando o problema de concorrência, já que o processo principal e o processo após uma falha são únicos. Durante a falha, ocorre o enfileiramento no próprio tópico da máquina principal, bastante que o tópico volte a ser consumido assim que o nó secundário se recupere da falha. Um ponto de atenção para arquitetura 3 é que a troca de dados e consumo de processamento tende a ser maior e para grandes sistemas deve ser um ponto a ser considerado.

Evidentemente, os problemas em casos reais são mais complexos que os demonstrados neste trabalho. Nas aplicações de telecomunicações, as quatro maiores empresas brasileiras têm em média 60 a 70 milhões de clientes, se cada cliente representar uma linha na base de dados, se teria um tamanho 6 a 7 vezes maior do que o simulado. Além do número maior de linhas, as bases possuem muito mais parâmetros, algumas chegando a centenas de parâmetros, o que torna o volume das bases muito maior, em vários *Gigabytes* a *Terabytes*. O tamanho da base vai impactar diretamente no tempo extração e transferência e, posteriormente, no tempo de batimento e correção. Outro ponto que extrapola o caso apresentado é a quantidade de bases, em uma aplicação real, podem ser dezenas de bases a serem cruzadas. O tempo total de batimento e correção pode chegar a dias inteiros, mostrando mais uma vez a necessidade de uma arquitetura redesenhada para atender a garantia de consistência e qualidade.

As arquiteturas 2 e 3 podem facilmente ser escaladas, uma vez que as ferramentas utilizadas, tanto Apache NiFi da arquitetura 2, quanto Apache Zookeeper e Kafka da arquitetura 3, são projetadas para um estresse maior em números de mensagens tratadas por segundo e quantidade de servidores. O próprio artigo inicial do Kafka cita a possibilidade de processar 50 mil mensagens por segundo com lotes de uma mensagem e 400 mil mensagens por segundo com lotes de 50 mensagens [**20**]. O NiFi pode ser também utilizado em *clusters* associado com o Zookeeper, distribuindo o processamento de dados em mais servidores. O amplo uso no mercado também demonstra a robustez das soluções para maiores volumes de dados e transações.

CAPÍTULO 6

Conclusão

O projeto previa demonstrar a necessidade de se manter a consistência e qualidade de dados em sistemas distribuídos em bases de Big Data, focando nos registros com erros, ao invés de se tratar sempre a base inteira, método este que se apresentou ineficiente. A simulação numa escala reduzida apresentou as dificuldades que podem ocorrer durante os processos de integração e replicação dos dados com falhas de comunicação e serviço a qualquer momento e como diferentes arquiteturas podem tratar do mesmo problema com eficiências distintas. Tais dificuldades, se extrapoladas para sistemas mais complexos, podem se tornar pontos impeditivos para determinadas operações. Então, foram apresentadas duas opções de como resolver o problema de uma arquitetura tradicional dispondo de ferramentas de código aberto já existentes no mercado que não foram necessariamente desenvolvidas para esta aplicação, mas que foram adaptadas e inseridas no contexto com bastante sucesso.

Como foi possível observar a partir dos resultados obtidos, a arquitetura 1 não é a melhor abordagem para tratar de grandes sistemas com grandes volumes de dados e alta complexidade com soluções baseadas em arquivos de lotes e correções *offline* como tradicionalmente tem sido feito. As soluções que atualmente têm atendido as premissas de resiliência, qualidade e consistência de Big Data são aquelas que tratam da transmissão de eventos em tempo real. As arquiteturas baseadas em eventos e *logs*, como as arquiteturas 2 e 3, apresentam maior robustez contra falhas, trazendo maiores chances de sucesso para o negócio envolvido.

A consistência e qualidade dos dados devem ser preocupações contínuas, desde a definição da arquitetura até a operação do sistema. É fundamental que o erro e a indisponibilidade dos sistemas sejam tratados como algo natural, provável e já previsto em fase de projeto, como foi proposto na arquitetura 3. Porém, mesmo que não sejam características defini-

das em tempo de projeto, ainda há métodos intermediários para aprimorar as arquiteturas tradicionais, como exposto na arquitetura 2.

Para trabalhos futuros, são sugeridos alterações nas estruturas de dados de arquivos textos para bancos de dados, tanto relacionais quanto não-relacionais para observação de performance e das características de concorrências. Também são sugeridas sistemas mais próximos da realidade com mais bases de dados e mais parâmetros, até mesmo parâmetros que não tenham uma relação explícita, mas sim uma relação baseada em regras de negócio e bases que tenham outra fonte de erro, além das falhas de comunicação, como, por exemplo, o erro humano. Outra sugestão é explorar mais aspectos da qualidade de dados além da correção e consistência, como por exemplo: completude, credibilidade, conformidade, atualidade, precisão, rastreabilidade, inteligibilidade, recuperabilidade, entre outros. Ainda estudos mais avançados podem propor mecanismos de Inteligência Artificial para definição de qual dado deve ser corrigido ou correlacionado naquele sistema quando há conflito entre dados de uma ou mais bases. Diversas áreas do conhecimento também podem propor aplicações para arquiteturas baseadas em eventos e apresentar as particularidades para cada área.

Se os dados são os maiores ativos das corporações, o cuidado com a qualidade e integridade deles deve ser o objetivo de qualquer setor da empresa, não só o setor tecnológico, mas também os setores que gerenciam e fazem uso dos dados. O foco não deve ser apenas em dados que trazem prejuízos financeiros diretos, mas também naqueles que trazem prejuízos de qualidade e experiência dos usuários, pois, futuramente, se tornam também prejuízos financeiros e prejudicam a imagem da organização, já que as empresas agora também precisam se preocupar com questões de proteção dos dados de seus clientes em atendimento à legislação nacional e aos acordos internacionais.

Bibliografia

[**1**] S. Kaisler, F. Armour, J. A. Espinosa, e W. Money, "Big data: Issues and challenges moving forward," pags. 995–1004, Jan 2013.

[**2**] R. Heisterberg e A. Verma, *Creating business agility: How convergence of cloud, social, mobile, video, and big data enables competitive advantage*. John Wiley & Sons, 2014.

[**3**] ISO-25012, "Software engineerging - software product quality requirements and evaluation (SQuaRE) - data quality model," ISO/IEC, Tech. Rep. 25012:2008, 2008.

[**4**] C. Batini, M. Scannapieco *et al.*, *Data and information quality*. Cham, Switzerland: Springer International Publishing. Google Scholar: Springer, 2016.

[**5**] T. Anderson, "The penalties of poor data," *Whitepaper published by GoImmedia. com and the Data Warehousing Institute dw-institute.com*, 2005.

[**6**] R. Mattison, *The telco revenue assurance handbook*. Oakwood Hills, Illinois: XiT Press, 2005.

[**7**] Mobileum-Wedo, "Raid brochure," 2017. [Online]. Available: https://www.mobileum.com/media/1855/raid_brochure_may2017.pdf

[**8**] P. Helland, "Immutability changes everything," *7th Biennial Conference on Innovative Data Systems Research*, 2015.

[**9**] M. Kleppmann, A. R. Beresford, e B. Svingen, "Online event processing," *Commun. ACM*, vol. 62, nº 5, pag. 43–49, 2019. [Online]. Available: https://doi.org/10.1145/3312527

[**10**] 3GPP-TR-32.901, "Study on User Data Convergence (UDC) information model handling and provisioning: Example use cases," ETSI 3GPP, 650 Route des Lucioles F-06921

Sophia Antipolis Cedex - France, Tech. Rep. TR 32.901, October 2014. [Online]. Available: https://www.etsi.org/deliver/etsi_tr/132900_132999/132901/12.00.00_60/tr_132901v120000p.pdf

[**11**] J.-H. Ahn, S.-P. Han, e Y.-S. Lee, "Customer churn analysis: Churn determinants and mediation effects of partial defection in the korean mobile telecommunications service industry," *Telecommunications Policy*, vol. 30, nº 10, pags. 552 – 568, 2006. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0308596106000760

[**12**] K. Baamann, "Data quality aspects of revenue assurance." em *ICIQ*, 2007, pags. 19–28.

[**13**] D. Kar, P. Misra, P. Bhattacharjee, e A. Mukherjee, "Real-time telecom revenue assurance," em *Proceedings of the Seventh International Conference on Digital Telecommunications*. Citeseer, 2012, pags. 130–135.

[**14**] P. Jayawardhana, D. Perera, A. Kumara, e A. Paranawithana, "Kanthaka: Big data caller detail record (cdr) analyzer for near real time telecom promotions," em *2013 4th International Conference on Intelligent Systems, Modelling and Simulation*, 2013, pags. 534–538.

[**15**] S. Flowerday e R. von Solms, "Real-time information integrity=system integrity+data integrity+continuous assurances," *Computers Security*, vol. 24, nº 8, pags. 604 – 613, 2005. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167404805001458

[**16**] B. M. Michelson, "Event-driven architecture overview," *Patricia Seybold Group*, vol. 2, nº 12, pags. 10–1571, 2006.

[**17**] M. T. Özsu e P. Valduriez, *Principles of distributed database systems*. Springer, 1999, vol. 2.

[**18**] M. Burrows, "The chubby lock service for loosely-coupled distributed systems," em *Proceedings of the 7th symposium on Operating systems design and implementation*, 2006, pags. 335–350.

[**19**] P. Hunt, M. Konar, F. P. Junqueira, e B. Reed, "Zookeeper: Wait-free coordination for internet-scale systems." em *USENIX annual technical conference*, vol. 8, nº 9, 2010.

[**20**] J. Kreps, N. Narkhede, J. Rao *et al.*, "Kafka: A distributed messaging system for log processing," em *Proceedings of the NetDB*, vol. 11, 2011, pags. 1–7.

Anexo I

Crontab Máquina 1

```
PATH=/home/ec2-user/.local/bin:/home/ec2-user/bin:/usr/local/
   ↪ bin:/usr/bin:/usr/local/sbin:/usr/sbin:/opt/kafka_2
   ↪ .12-2.4.1/bin
* * * * * /bin/bash /data/bin/provisionador /data/dataset/
   ↪ msisdn_plano >> /data/log/provisionador.log 2>&1
```

msisdn.json

```
{
   "datasets": {
      "sample": {
         "size": 1000,
         "locale": "pt_BR",
         "fields": [
            {
               "name": "id",
               "type": "integer:sequence",
               "generator": {
                   "start_at": 1
               }
            },
            {
               "name": "MSISDN",
               "type": "msisdn",
               "generator": {
```

```
                }
            },
            {
                "name": "Plano",
                "type": "month",
                "generator": {
                }
            }
        ],
        "format": {
            "type": "csv",
             "options":{
               "header": false,
               "sep": " "
             }

        },
        "serializers": {
          "to": [
             {
               "type": "file",
               "uri": "/data/dataset/msisdn_plano"
             }
            ]
        }
    }
}
```

inicia

```
#!/bin/bash

time ggenerator generate --spec /data/config/msisdn.json
scp /data/dataset/msisdn_plano ec2-user@18.230.14.219:/data/
   ↪ dataset/msisdn_plano
scp /data/dataset/msisdn_plano ec2-user@54.232.64.254:/data/
   ↪ dataset/msisdn_plano
```

provisionador

```
#!/bin/bash
#
# Autor: Marcos Romero
# Data: 20/09/2020
# Recebe uma lista de numeros e aprovisiona uma linha aleatoria
# na outra maquina
############################################################

data=`date +%d-%m-%Y"_"%H:%M:%S.%N`

echo "$data [INFO] Iniciando Provisionador..."
arquivo=$1
rand=`shuf -i 1-1000 -n 1`
linhas=`cat $arquivo | wc -l`

antigo=`sed -n "$rand"p $arquivo`
ggenerator generate --spec /data/config/msisdn_unico.json > /
   ↪ dev/null 2>&1
novo=`cat /data/dataset/msisdn_unico`
numero=`cat /data/dataset/msisdn_unico | awk -F"\"" '{print
   ↪ "\""$2"\""}'`
plano=`cat /data/dataset/msisdn_unico | awk -F"\"" '{print "\""
   ↪ $4"\""}'`
lock=/data/dataset/.lock

data=`date +%d-%m-%Y"_"%H:%M:%S.%N`
echo "$data [INFO] Substituir $antigo por $novo"
sed -i "$rand c\\$rand $novo" $arquivo > /dev/null 2>&1
if [ $? -eq 0 ]; then
  data=`date +%d-%m-%Y"_"%H:%M:%S.%N`
  resultado=`sed -n "$rand"p $arquivo`
  echo "$data [OK] Substituido" $resultado" hostname: tcc-1"
else
  data=`date +%d-%m-%Y"_"%H:%M:%S.%N`
  echo "$data [FAIL] Falha ao substituir $antigo por $novo
     ↪ hostname: tcc-1"
```

```
  exit 1
fi



ssh -o ConnectTimeout=2 ec2-user@18.230.14.219 "flock $lock sed
   ↪  -i '$rand c\\$rand $numero $plano' $arquivo" > /dev/null
   ↪  2>&1
if [ $? -eq 0 ]; then
  data=`date +%d-%m-%Y"_"%H:%M:%S.%N`
  resultado=`ssh ec2-user@18.230.14.219 "sed -n \"$rand\"p
     ↪ $arquivo"`
  echo "$data [OK] Substituido" $resultado" hostname: tcc-2"
else
  data=`date +%d-%m-%Y"_"%H:%M:%S.%N`
  echo "$data [FAIL] Falha ao substituir $antigo por $novo
     ↪ hostname: tcc-2"
fi

ssh -o ConnectTimeout=2 ec2-user@54.232.64.254 "flock $lock sed
   ↪  -i '$rand c\\$rand $numero $plano' $arquivo" > /dev/null
   ↪  2>&1
if [ $? -eq 0 ]; then
  data=`date +%d-%m-%Y"_"%H:%M:%S.%N`
  resultado=`ssh ec2-user@54.232.64.254 "sed -n \"$rand\"p
     ↪ $arquivo"`
  echo "$data [OK] Substituido" $resultado" hostname: tcc-3"
else
  data=`date +%d-%m-%Y"_"%H:%M:%S.%N`
  echo "$data [FAIL] Falha ao substituir $antigo por $novo
     ↪ hostname: tcc-3"
fi

exit 0
```

Crontab Máquina 2 e 3

```
PATH=/home/ec2-user/.local/bin:/home/ec2-user/bin:/usr/local/
   ↪ bin:/usr/bin:/usr/local/sbin:/usr/sbin:/opt/kafka_2
```

```
   ↪ .12-2.4.1/bin
#* * * * * /bin/bash /data/bin/falha >> /data/log/falha.log
   ↪ 2>&1
```

falha

```
#!/bin/bash

rand=`shuf -i 1-10 -n 1`
if [ $rand -eq 1 ]; then
      mv /data/dataset/msisdn_plano /data/dataset/
         ↪ msisdn_plano_off > /dev/null 2>&1
      if [ $? -eq 0 ]; then
            data=`date +%d-%m-%Y"_"%H:%M:%S.%N`
            echo "$data [ERROR] Servidor tcc-2 Down"
            exit 0
      fi
else
      mv /data/dataset/msisdn_plano_off /data/dataset/
         ↪ msisdn_plano > /dev/null 2>&1
      if [ $? -eq 0 ]; then
            data=`date +%d-%m-%Y"_"%H:%M:%S.%N`
            echo "$data [INFO] Servidor tcc-2 Up"
            exit 0
      fi
fi
data=`date +%d-%m-%Y"_"%H:%M:%S.%N`
echo "$data [INFO] Servidor tcc-2 Nada a fazer"
exit 0
```

provisionador_individual

```

#!/bin/bash
#
# Autor: Marcos Romero
# Data: 20/09/2020
# Recebe um de numero e aprovisiona uma linha
# na outra maquina
############################################################
```

```
data=`date +%d-%m-%Y"_"%H:%M:%S.%N`
arquivo=$1
host=$2
log=/data/log/provisionador.log
echo "$data [INFO] Iniciando Provisionador Individual... $host"
   ↪  >> $log
destino=/data/dataset/msisdn_plano
lock=/data/dataset/.lock

if [ -f "$arquivo" ]; then
antigo=`cat $arquivo | awk '{print $6,$7,$8}'`
novo=`cat $arquivo | awk '{print $10,$11}'`
rand=`cat $arquivo | awk '{print $6}'`
numero=`cat $arquivo | awk -F"\"" '{print "\""$6"\""}'`
plano=`cat $arquivo | awk -F"\"" '{print "\""$8"\""}'`
echo "$rand"
echo "$numero"
echo "$plano"
data=`date +%d-%m-%Y"_"%H:%M:%S.%N`
echo "$data [INFO] Substituir $antigo por $novo"

result=1
if [[ $host == "tcc-2" ]]; then
      while [ $result -eq 1 ]
      do
      ssh -o ConnectTimeout=2 ec2-user@18.230.14.219 "flock
         ↪ $lock sed -i '$rand c\\$rand $numero $plano'
         ↪ $destino" > /dev/null 2>&1
      if [ $? -eq 0 ]; then
        data=`date +%d-%m-%Y"_"%H:%M:%S.%N`
        resultado=`ssh ec2-user@18.230.14.219 "sed -n \"$rand
           ↪ \"p $destino"`
        echo "$data [OK] Substituido" $resultado" hostname:
           ↪ tcc-2 (Provisionador Individual)" >> $log
        rm $arquivo
        exit 0
```

```
      else
        data=`date +%d-%m-%Y"_"%H:%M:%S.%N`
        echo "$data [FAIL] Falha ao substituir $antigo por
          ↪ $novo hostname: tcc-2"
        result=1
      fi
      sleep 5
      done
fi

if [[ $host == "tcc-3" ]]; then
      while [ $result -eq 1 ]
      do
      ssh -o ConnectTimeout=2 ec2-user@54.232.64.254 "flock
        ↪ $lock sed -i '$rand c\\$rand $numero $plano'
        ↪ $destino" > /dev/null 2>&1
      if [ $? -eq 0 ]; then
        data=`date +%d-%m-%Y"_"%H:%M:%S.%N`
        resultado=`ssh ec2-user@54.232.64.254 "sed -n \"$rand
          ↪ \"p $destino"`
        echo "$data [OK] Substituido" $resultado" hostname:
          ↪ tcc-3 (Provisionador Individual)" >> $log
        rm $arquivo
        exit 0
      else
        data=`date +%d-%m-%Y"_"%H:%M:%S.%N`
        echo "$data [FAIL] Falha ao substituir $antigo por
          ↪ $novo hostname: tcc-3"
        result=1
      fi
      sleep 5
      done
fi
exit 0

else
```

```
echo "$data [FAIL] Arquivo de falha nao existe!"
exit 1
fi
```

provisionador_kafka - Rodando na máquina 1

```
#!/bin/bash
#
# Autor: Marcos Romero
# Data: 20/09/2020
# Recebe uma lista de numeros e aprovisiona uma linha aleatoria
# no topico
#############################################################

data=`date +%d-%m-%Y"_"%H:%M:%S.%N`

echo "$data [INFO] Iniciando Provisionador..."
arquivo=$1
rand=`shuf -i 1-1000 -n 1`


antigo=`sed -n "$rand"p $arquivo`
ggenerator generate --spec /data/config/msisdn_unico.json > /
   ↪ dev/null 2>&1
novo=`cat /data/dataset/msisdn_unico`
numero=`cat /data/dataset/msisdn_unico | awk -F"\"" '{print
   ↪ "\""$2"\""}'`
plano=`cat /data/dataset/msisdn_unico | awk -F"\"" '{print "\""
   ↪ $4"\""}'`

data=`date +%d-%m-%Y"_"%H:%M:%S.%N`
echo "$data [INFO] Substituir $antigo por $novo"
sed -i "$rand c\\$rand $novo" $arquivo > /dev/null 2>&1
if [ $? -eq 0 ]; then
  data=`date +%d-%m-%Y"_"%H:%M:%S.%N`
  resultado=`sed -n "$rand"p $arquivo`
  echo "$data [OK] Substituido" $resultado" hostname: tcc-1"
  kafka-console-producer.sh --broker-list 18.230.114.205:9092
```

```
    ↪ --topic bss <<< $resultado
else
  data=`date +%d-%m-%Y"_"%H:%M:%S.%N`
  echo "$data [FAIL] Falha ao substituir $antigo por $novo
    ↪ hostname: tcc-1"
  exit 1
fi


exit 0
```

provisionador_kafka - Rodando nas máquinas 2 e 3

```
#!/bin/bash
#
# Autor: Marcos Romero
# Data: 20/09/2020
# Recebe uma linha no topico kafka e tenta atualizar arquivo
#
############################################################

data=`date +%d-%m-%Y"_"%H:%M:%S.%N`
arquivo=/data/dataset/msisdn_plano
log=/data/log/provisionador.log
echo "$data [INFO] Iniciando Provisionador..." >> $log

while true
do
comando=`kafka-console-consumer.sh --bootstrap-server
  ↪ 18.230.114.205:9092 --topic bss --group tcc-2 --max-
  ↪ messages 1 --timeout-ms 2000` > /dev/null 2>&1
if [ -z "$comando" ];
then
  sleep 10
else
  rand=`echo "$comando" | awk '{print $1}'`
  novo=`echo "$comando" | awk '{print $2,$3}'`
  antigo=`sed -n "$rand"p $arquivo`
```

```
  data=`date +%d-%m-%Y"_"%H:%M:%S.%N`
  echo "$data [INFO] Substituir $antigo por $novo" >> $log
  sed -i "$rand c\\$rand $novo" $arquivo > /dev/null 2>&1
  if [ $? -eq 0 ]; then
      data=`date +%d-%m-%Y"_"%H:%M:%S.%N`
      resultado=`sed -n "$rand"p $arquivo`
      echo "$data [OK] Substituido" $resultado" hostname: tcc
         ↪ -2" >> $log
  else
      data=`date +%d-%m-%Y"_"%H:%M:%S.%N`
      echo "$data [FAIL] Falha ao substituir $antigo por $novo
         ↪ hostname: tcc-2" >> $log
  fi
fi
done

exit 0
```

falha_kafka

```
#!/bin/bash
log=/data/log/provisionador.log
rand=`shuf -i 1-10 -n 1`
if [ $rand -eq 1 ]; then
      pkill -f "provisionador_kafka"
      if [ $? -eq 0 ]; then
            data=`date +%d-%m-%Y"_"%H:%M:%S.%N`
            echo "$data [ERROR] Servidor tcc-2 Down" >> $log
               ↪ 2>&1
            exit 0
      fi
else
      pgrep -f provisionador_kafka
      if [ $? -ne 0 ]; then
         /bin/bash /data/bin/provisionador_kafka & > /dev/null
            ↪ 2>&1
         if [ $? -eq 0 ]; then
               data=`date +%d-%m-%Y"_"%H:%M:%S.%N`
```

```
                echo "$data [INFO] Servidor tcc-2 Up" >> $log
                    ↪ 2>&1
                exit 0
            fi
        fi
fi
data=`date +%d-%m-%Y"_"%H:%M:%S.%N`
echo "$data [INFO] Servidor tcc-2 Nada a fazer"
exit 0
```

zookeeper.properties (máquina 1)

```
# Licensed to the Apache Software Foundation (ASF) under one or
   ↪  more
# contributor license agreements. See the NOTICE file
   ↪ distributed with
# this work for additional information regarding copyright
   ↪ ownership.
# The ASF licenses this file to You under the Apache License,
   ↪ Version 2.0
# (the "License"); you may not use this file except in
   ↪ compliance with
# the License. You may obtain a copy of the License at
#
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing,
   ↪ software
# distributed under the License is distributed on an "AS IS"
   ↪ BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express
   ↪ or implied.
# See the License for the specific language governing
   ↪ permissions and
# limitations under the License.
# the directory where the snapshot is stored.
dataDir=/opt/kafka_2.12-2.4.1/data/zookeeper
# the port at which the clients will connect
```

```
clientPort=2181
# disable the per-ip limit on the number of connections since
   ↪ this is a non-production config
maxClientCnxns=60
# Disable the adminserver by default to avoid port conflicts.
# Set the port to something non-conflicting if choosing to
   ↪ enable this
admin.enableServer=false
# admin.serverPort=8080
tickTime=2000
initLimit=10
syncLimit=5
server.1=0.0.0.0:2888:3888
server.2=18.230.14.219:2888:3888
server.3=54.232.64.254:2888:3888
```

server.propertis (configuração Kafka máquina 1)

```
# Licensed to the Apache Software Foundation (ASF) under one or
   ↪  more
# contributor license agreements. See the NOTICE file
   ↪ distributed with
# this work for additional information regarding copyright
   ↪ ownership.
# The ASF licenses this file to You under the Apache License,
   ↪ Version 2.0
# (the "License"); you may not use this file except in
   ↪ compliance with
# the License. You may obtain a copy of the License at
#
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing,
   ↪ software
# distributed under the License is distributed on an "AS IS"
   ↪ BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express
   ↪ or implied.
```

```
# See the License for the specific language governing
   ↪ permissions and
# limitations under the License.

# see kafka.server.KafkaConfig for additional details and
   ↪ defaults

############################# Server Basics
   ↪ #############################

# The id of the broker. This must be set to a unique integer
   ↪ for each broker.
broker.id=0
host.name=172.31.15.245
advertised.host.name=18.230.114.205


############################# Socket Server Settings
   ↪ #############################

# The address the socket server listens on. It will get the
   ↪ value returned from
# java.net.InetAddress.getCanonicalHostName() if not configured
   ↪ .
# FORMAT:
# listeners = listener_name://host_name:port
# EXAMPLE:
# listeners = PLAINTEXT://your.host.name:9092
#listeners=PLAINTEXT://:9092
#listeners=PLAINTEXT://localhost:9092

# Hostname and port the broker will advertise to producers and
   ↪ consumers. If not set,
# it uses the value for "listeners" if configured. Otherwise,
   ↪ it will use the value
# returned from java.net.InetAddress.getCanonicalHostName().
#advertised.listeners=PLAINTEXT://your.host.name:9092
```

```
advertised.listeners=PLAINTEXT://18.230.114.205:9092

# Maps listener names to security protocols, the default is for
   ↪  them to be the same. See the config documentation for
   ↪ more details
#listener.security.protocol.map=PLAINTEXT:PLAINTEXT,SSL:SSL,
   ↪ SASL_PLAINTEXT:SASL_PLAINTEXT,SASL_SSL:SASL_SSL

# The number of threads that the server uses for receiving
   ↪ requests from the network and sending responses to the
   ↪ network
num.network.threads=3

# The number of threads that the server uses for processing
   ↪ requests, which may include disk I/O
num.io.threads=8

# The send buffer (SO_SNDBUF) used by the socket server
socket.send.buffer.bytes=102400

# The receive buffer (SO_RCVBUF) used by the socket server
socket.receive.buffer.bytes=102400

# The maximum size of a request that the socket server will
   ↪ accept (protection against OOM)
socket.request.max.bytes=104857600



############################# Log Basics
   ↪ #############################

# A comma separated list of directories under which to store
   ↪ log files
log.dirs=/opt/kafka_2.12-2.4.1/data/kafka

# The default number of log partitions per topic. More
```

```
   ↪ partitions allow greater
# parallelism for consumption, but this will also result in
   ↪ more files across
# the brokers.
num.partitions=4

# The number of threads per data directory to be used for log
   ↪ recovery at startup and flushing at shutdown.
# This value is recommended to be increased for installations
   ↪ with data dirs located in RAID array.
num.recovery.threads.per.data.dir=1

############################## Internal Topic Settings
   ↪ #############################
# The replication factor for the group metadata internal topics
   ↪  "__consumer_offsets" and "__transaction_state"
# For anything other than development testing, a value greater
   ↪ than 1 is recommended to ensure availability such as 3.
offsets.topic.replication.factor=1
transaction.state.log.replication.factor=1
transaction.state.log.min.isr=1

############################## Log Flush Policy
   ↪ #############################

# Messages are immediately written to the filesystem but by
   ↪ default we only fsync() to sync
# the OS cache lazily. The following configurations control the
   ↪  flush of data to disk.
# There are a few important trade-offs here:
# 1. Durability: Unflushed data may be lost if you are not
   ↪ using replication.
# 2. Latency: Very large flush intervals may lead to latency
   ↪ spikes when the flush does occur as there will be a lot
   ↪ of data to flush.
# 3. Throughput: The flush is generally the most expensive
   ↪ operation, and a small flush interval may lead to
```

```
   ↪ excessive seeks.
# The settings below allow one to configure the flush policy to
   ↪  flush data after a period of time or
# every N messages (or both). This can be done globally and
   ↪ overridden on a per-topic basis.

# The number of messages to accept before forcing a flush of
   ↪ data to disk
#log.flush.interval.messages=10000

# The maximum amount of time a message can sit in a log before
   ↪ we force a flush
#log.flush.interval.ms=1000

############################# Log Retention Policy
   ↪ #############################

# The following configurations control the disposal of log
   ↪ segments. The policy can
# be set to delete segments after a period of time, or after a
   ↪ given size has accumulated.
# A segment will be deleted whenever *either* of these criteria
   ↪  are met. Deletion always happens
# from the end of the log.

# The minimum age of a log file to be eligible for deletion due
   ↪  to age
log.retention.hours=168

# A size-based retention policy for logs. Segments are pruned
   ↪ from the log unless the remaining
# segments drop below log.retention.bytes. Functions
   ↪ independently of log.retention.hours.
#log.retention.bytes=1073741824

# The maximum size of a log segment file. When this size is
   ↪ reached a new log segment will be created.
```

```
log.segment.bytes=1073741824

# The interval at which log segments are checked to see if they
   ↪  can be deleted according
# to the retention policies
log.retention.check.interval.ms=300000

############################# Zookeeper
   ↪ #############################

# Zookeeper connection string (see zookeeper docs for details).
# This is a comma separated host:port pairs, each corresponding
   ↪  to a zk
# server. e.g. "127.0.0.1:3000,127.0.0.1:3001,127.0.0.1:3002".
# You can also append an optional chroot string to the urls to
   ↪ specify the
# root directory for all kafka znodes.
zookeeper.connect=18.230.14.219:2181,localhost
   ↪ :2181,54.232.64.254:2181

# Timeout in ms for connecting to zookeeper
zookeeper.connection.timeout.ms=6000


############################# Group Coordinator Settings
   ↪ #############################

# The following configuration specifies the time, in
   ↪ milliseconds, that the GroupCoordinator will delay the
   ↪ initial consumer rebalance.
# The rebalance will be further delayed by the value of group.
   ↪ initial.rebalance.delay.ms as new members join the group,
   ↪  up to a maximum of max.poll.interval.ms.
# The default value for this is 3 seconds.
# We override this to 0 here as it makes for a better out-of-
   ↪ the-box experience for development and testing.
# However, in production environments the default value of 3
```

```
   ↪ seconds is more suitable as this will help to avoid
   ↪ unnecessary, and potentially expensive, rebalances during
   ↪  application startup.
group.initial.rebalance.delay.ms=0
```