UNICESUMAR – UNIVERSIDADE CESUMAR
CENTRO DE CIÊNCIAS EXATAS TECNOLÓGICAS E AGRÁRIAS
CURSO DE GRADUAÇÃO EM ENGENHARIA DE SOFTWARE

UMA ANÁLISE SOBRE O TERMO *DATAOPS*

ENDERSON MENEZES CÂNDIDO

MARINGÁ – PR
2022

Enderson Menezes Cândido

UMA ANÁLISE SOBRE O TERMO *DATAOPS*

Artigo apresentado ao Curso de Graduação em Engenharia de Software da UNICESUMAR – Universidade Cesumar como requisito parcial para a obtenção do título de Bacharel(a) em Engenharia de Software, sob a orientação do Prof. Msc. Aparecido Vilela Júnior.

MARINGÁ – PR

2022

ENDERSON MENEZES CÂNDIDO

UMA ANÁLISE SOBRE O TERMO *DATAOPS*

Artigo apresentado ao Curso de Graduação em Engenharia de Software da UNICESUMAR – Universidade Cesumar como requisito parcial para a obtenção do título de Bacharel(a) em Engenharia de Software sob a orientação do Prof. Msc. Aparecido Vilela Júnior.

Aprovado em: 11 de julho de 2022.

BANCA EXAMINADORA

_____

Prof. Msc. Marcello Erick Bonfim - UniCesumar

_____

Prof. Msc. Aparecido Vilela Júnior - UniCesumar

UMA ANÁLISE SOBRE O TERMO *DATAOPS*

Enderson Menezes Cândido

RESUMO

O armazenamento de dados deixa de se tornar o maior problema, a tecnologia se torna capaz de armazenar grandes quantidades de dados com fácil implementação e baixo custo. O processamento de dados se torna alvo dos novos questionamentos, em poucos anos se observa *BigData* evoluir para *DataOps*. Este trabalho reúne estudos ao longo de dois anos de atuação em Engenharia de Dados e apresenta uma arquitetura para o manuseio de dados genérica. Baseado em uma análise de artigos internacionais e casos de uso de grandes empresas que se tornaram ferramentas de mercado. Buscando incentivar artigos científicos de produção nacional que abordem esse tema, quebrando barreiras linguísticas e tornando-se uma base para informatização de empresas e formação de pessoas.

Palavras-chave: Dados. Arquitetura de Software. Ferramentas para Dados.

AN ANALYSIS ABOUT *DATAOPS*

ABSTRACT

Data storage is no longer the biggest problem, technology becomes capable of storing large amounts of data with easy implementation and low cost. Data processing becomes the target of new questions, in a few years BigData can be seen evolving into DataOps. This work brings together studies over two years of experience in Data Engineering and presents an architecture for handling generic data. Based on an analysis of international articles and use cases from large companies that have become market tools. Seeking to encourage scientific articles of national production that address this topic, breaking language barriers and becoming a basis for computerization of companies and training of people.

Keywords: Data. Data Architecture. Data Tools.

1 INTRODUÇÃO

Dados são produzidos durante todo o tempo, independente do meio tecnológico, seja na Internet, sistemas com sensores, dispositivos móveis e inúmeros outros equipamentos. Uma pesquisa conduzida pelo entro Regional de Estudos para o Desenvolvimento da Sociedade da Informação (Cetic.br) estimou que em 2019 cerca de 134 milhões de brasileiros possuíam acesso à internet (CGI.br/NIC.br, Pesquisa sobre o Uso das Tecnologias de Informação e Comunicação nos Domicílios Brasileiros – TIC Domicílios 2019). As grandes companhias tecnológicas como Google e Facebook recebem e indexam conteúdo diariamente que seriam humanamente impossíveis de serem consumidos por uma única só pessoa no mesmo período (MARR, 2015).

Dado essa realidade, a comunidade de TI (Tecnologia da Informação) começou a implementar e melhorar técnicas de manipulação e gerenciamento de dados (*Big Data*). Por essa razão, no ambiente de inovação e tecnologia, comumente fala-se que os dados são o novo petróleo, haja vista a sua grande possibilidade de valorização e exploração.

Dhananjay Mehta, em seu trabalho intitulado "*Building a scalable distributed data platform using lambda architecture*" (p. 2, 2017) preleciona que no contexto técnico geral, o mundo está passando por uma transformação tecnológica muito grande, uma geração de banco de dados legados, agora tendo que ser atualizada e/ou integrar/migrar para novas tecnologias e novas formas de gerenciamentos.

A comunidade de dados também acompanha as tendências de metodologias ágeis ao desenvolvimento, que surgem com o propósito de organizar, observar, tornar cíclico o processo de desenvolvimento. Na área de dados essas metodologias são aplicadas para responder as seguintes questões: "O que fazer com esses dados? Como armazenar os dados? Como gerenciar os dados? Como tornar o processo de análise gerenciável e cíclico?".

Com o surgimento do Manifesto Ágil[1], surgiram posteriormente diversas adaptações, como o movimento cultural iniciado por Patrick Debois para operações de desenvolvimento (DevOps), explicado no New Relic Portal[2], o qual serve de base para os termos: Entrega Contínua e Integração Contínua. Os profissionais de tecnologia com ênfase em dados se

1 MANIFESTO, 2001. Agile. Disponível em< http://agilemanifesto. org>. Acesso em 18 de abril, 2022.

2 NEW RELIC. What is a DevOps?. Disponível em: https://newrelic.com/devops/what-is-devops. Acesso em: 25 de maio, 2021.

apropriam e começam a definir processos de entrega contínua de dados e integração contínua de sistemas analíticos.

As teorias mais aceitas de *Big Data* já identificaram que muito da infraestrutura legada se mostra incapaz de evoluir, em razão de dois principais pontos: devido à sua capacidade de processamento de dados ou, quando possível o processamento, devido ao custo financeiro impraticável em relação às ferramentas modernas (MEHTA, 2017).

Dado aos fatos supracitados e observados da modernização da sociedade com a adoção das novas metodologias e a implementação de novas tecnologias, surgem alguns problemas, tais como: arquitetar várias fontes de dados, integrar micro serviços existentes e diferentes formatos de arquivos, realizar transformações de dados, trabalhar com um ou diversos bancos de dados e *data warehouses* (armazém de dados) e centro com tecnologias diferentes. Com isso, passar-se-ia a lidar com a produção de processos fragmentados, arquiteturas e sub arquiteturas que não possuem integração, e podendo gerar ainda um dos maiores problemas: o alto gasto em manutenção (METHA, 2017).

O objetivo desse trabalho é analisar o surgimento do termo *DataOps* e os conceitos que estão associados a ele. Para tanto, propõem-se os seguintes objetivos específicos:

- Definir e estudar uma base de referência teórica;
- Traçar uma história analítica dos estudos de dados com foco em tecnologia;
- Apresentar um modelo conceitual de *pipeline* para dados;
- Apresentar um modelo genérico para ingestão de dados e troca de dados;

2 DESENVOLVIMENTO

2.1 Base de referência teórica

Esse estudo abordará o universo do *DataOps* e, para compreendê-lo, faz-se necessária a introdução breve dos seguintes termos:

- cultura *devops*
- *cloud* pública
- *cloud* privada
- *pipeline*
- arquitetura *serverless*
- arquitetura *lambda*

A **Cultura *DevOps*** combina diversas filosofias culturais, práticas e ferramentas que aumentam a capacidade de um desenvolvedor e/ou uma equipe de desenvolvimento para distribuir as atividades que lhe foram conferidas em alta velocidade, otimizando e aperfeiçoando produtos em um ritmo mais rápido do que o das empresas que usam processos tradicionais de desenvolvimento de software e gerenciamento de infraestrutura (NEW RELIC, 2021).

Essa velocidade torna a comunicação entre ator do modelo de negócio e desenvolvedor, passando por diversas etapas com menos fricção que os processos convencionais.

Quanto à **cloud pública** e **cloud privada**, pode-se definir uma série de serviços de computação oferecidos e catalogados, os quais são disponibilizados de forma a otimizar a utilização de recursos. Esta *cloud* será **pública** quando os serviços estiverem disponíveis a qualquer pessoa, de modo pago ou gratuito, e **privada** quando for oferecida internamente pela ou para uma companhia, visando a sua utilização interna. A *cloud* **privada** pode ainda ser alugada e somente administrada pelo seu corpo tecnológico ou equipada inteiramente dentro da companhia (RED HAT, 2018).

Uma ***pipeline*** é um conceito comumente usado na vida cotidiana para definir e processar um conjunto de instruções em lote ou em paralelo, dependendo do problema que se deseja resolver.

Uma **arquitetura *serverless***, é um modelo de execução onde o provedor de cloud pública ou privada será o responsável por executar pedaços de código com recursos que irão ser alocados dinamicamente e cobrando apenas pelos recursos usados para executar aquele código em específico. Nesse modelo, o desenvolvedor é responsável apenas pelo código enviado e o provedor pelos seus recursos. O código é geralmente escrito em forma de funções. Em decorrência, pode-se verificar a arquitetura *serverless* ser referenciada como "*functions as a service*" (funções como serviço) ou "FaaS" (RED HAT, 2017).

A **arquitetura *lambda*** define cadeia de fornecimento de dados define princípios para construir e gerenciar um *pipeline* de dados. *Lambdas* implementam todos os componentes de uma cadeia de fornecimento de dados seguindo em três camadas:

- **camada de lote** com foco na ingestão, armazenamento e processamento de dados em lote;
- **camada de velocidade:** com foco na ingestão, armazenamento e processamento de dados em tempo real;
- **camada de fluxo:** com foco em servir uma recuperação de visualizações carregadas em lote ou em tempo real.

2.2 UMA HISTÓRIA SOBRE *DATAOPS*

Uma história pode ser contada de diversas maneiras a depender da realidade do observador. Pode-se levar em conta fatos geopolíticos, históricos, geográficos e diversos outros fatores. Neste trabalho, apresenta-se fatos tecnológicos.

Historicamente, pode-se observar um padrão evolutivo na análise de dados. O artigo "***From Ad-Hoc Data Analytics to DataOps"*** [3]ou em tradução livre "De uma análise de dados manual ao DataOps" descreve em seu desenvolvimento três grandes marcos históricos que teve participação na evolução da análise de dados, nos próximos parágrafos deste artigo estará disponível uma síntese com complemento do autor.

O processo de evolução para uma análise de dados se tornar um *DataOps* começa na **análise *ad-hoc***, no qual relatórios e insights são criados sob demanda e podem ser altamente personalizados. Normalmente, a análise *ad-hoc* é realizada para responder a uma questão muito específica (MUNAPPY et al., 2022).

Esse processo começa a se tornar **semiautomático** com as *pipelines* de dados para coletar e processar dados, tornando-se uma maneira mais eficiente e automatizada de implementar a análise de dados.

Ao adotar as metodologias ágeis, a **Ciência de Dados** consegue ter múltiplas equipes trabalhando em um mesmo repositório de dados e, se adaptar requisitos do cliente, mudam com tempo e a fim de lidar com os requisitos em evolução, é necessário seguir a metodologia ágil em que os *insights* são entregues em *sprints* curtos.

Continuidade, testes e monitoramento tornam-se um elemento essencial ao lidar com dados em tempo real, ajudando a detectar os problemas imediatamente antes de serem transportados para as próximas fases. Sem um mecanismo de monitoramento, quando os dados recebidos no final do *pipeline* não são o esperado, torna-se difícil identificar o motivo da saída inesperada. (MUNAPPY et al., 2022).

Dada a evolução descrita e as características supracitadas, somando-se a um conjunto de instruções, pode-se definir ***DataOps*** adotando-se princípios *DevOps* para gerenciar efetivamente os artefatos como dados, metadados e código. Uma grande diferença entre o

---

3 MUNAPPY, Aiswarya Raj et al. From ad-hoc data analytics to dataops. In: Proceedings of the International Conference on Software and System Processes. 2020. p. 165-174.New Relic.

DataOps e *DevOps* é que o primeiro tem que gerenciar dados e código, enquanto o segundo se preocupa apenas com o código em evolução.

2.3 BOAS PRÁTICAS E ARQUITETURA REUTILIZÁVEL

A definição de *pipeline* já abordada neste artigo pode ser resumida como um conjunto de instruções. Assim, para definir uma *data pipeline* genérica, é preciso definir um conjunto de instruções genérico para trabalhar com dados. Faz-se a analogia com uma esteira de produção, na qual seja possível colocar uma peça no início dos trabalhos; e essa peça, no final da esteira, tenha passado por uma manutenção e uma pintura; o resultado seria uma boa arquitetura de instruções que será formada pelo conjunto:

- Observabilidade
- Segurança
- Continuidade

Com **Observabilidade** é esperado que seu conjunto de instruções consiga mostrar "o que" são meus dados; "onde" estão sendo salvos/processados; "quantos" dados terão em nossa "esteira".

Com **Segurança** sua pipeline estará preocupada com duas visões a de "integridade" dos seus dados e a de armazenamento dos seus dados. Nesse sentido, pode-se indagar sobre a segurança: poder-se-ia afirmar que as informações enviadas são as informações requeridas?; poder-se-ia afirmar que os dados processados foram realmente salvos?

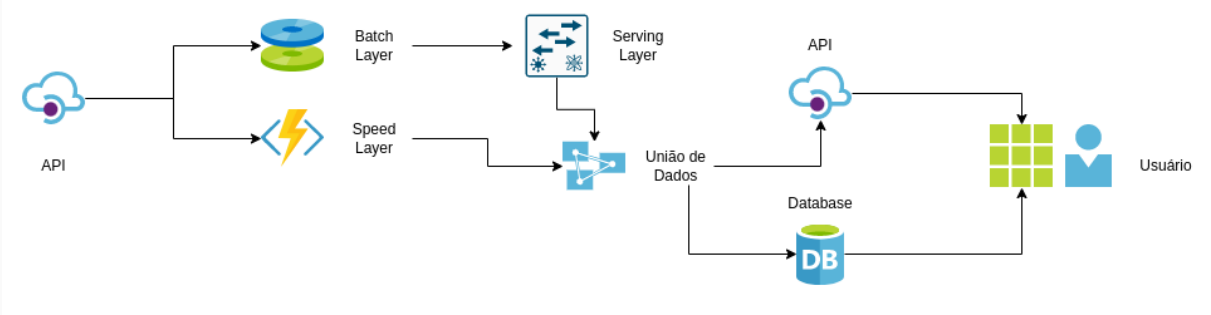
Com **Continuidade** todo o seu processo precisa conseguir afirmar que **tudo** que entrou em sua esteira, saiu dela. Elementos desconhecidos não podem "crescer" em sua *pipeline;* eles podem nascer, ser identificados pelo seu processo de observabilidade e então ser tratados pelo seu processo de segurança, garantindo sua continuidade plena.

2.4 ARQUITETURA GENÉRICA PARA INGESTÃO DE DADOS

A abordagem de Arquitetura *Lambda* é uma boa metodologia para análise de dados em diversos casos, mistura recursos de processamento de dados em lote (*batch*) e tempo real (*streaming*) e pode ser dividida em três camadas de processamento: camada *batch* (lote), camada servidora (*serving*) e camada em tempo real (*speed layer*) (PERRON, 2020).

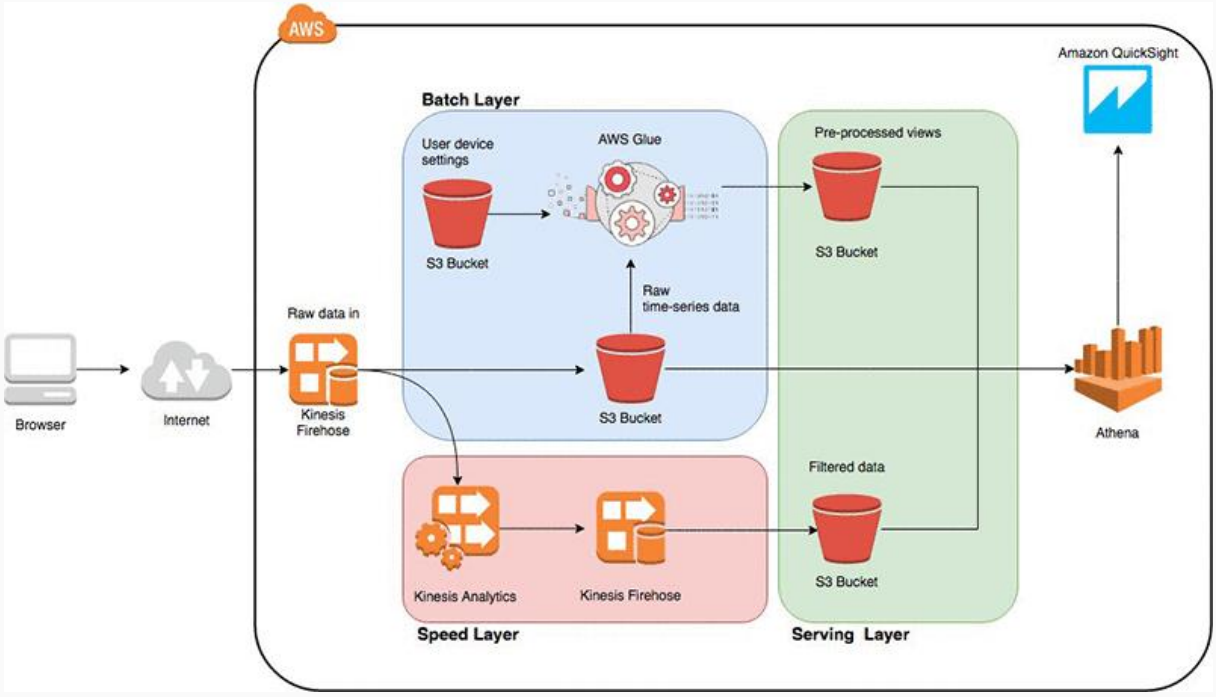
Todas as camadas de processamento se unem em um ponto, buscando entregar valor a um cliente, seja ele o Analista de Dados ou o "Cliente Final" de um Dashboard. conforme imagem a seguir:

Figura 1 – Modelo de arquitetura lambda com destino final.



Fonte: Desenhado pelo autor com Draw.io utilizando biblioteca de ícones gratuitos.

Figura 2 – Modelo de arquitetura lambda utilizando serviços públicos da Amazon.



Fonte: Amazon (2017).

A arquitetura ilustrada na figura 2, descrita no artigo "*Unite Real-Time and Batch Analytics Using the Big Data Lambda Architecture, Without Servers!*", propõe uma utilização da teórica da Arquitetura *Lambda* com seus serviços públicos. Essas ferramentas possuem opções de diversas maneiras de gestão dos dados e todas em conjunto fornecerão uma arquitetura robusta e escalável para análise de dados.

2.5 ARQUITETURA GENÉRICA PARA TRANSFERÊNCIA DE DADOS

A arquitetura proposta a seguir exemplifica atores que, em determinado momento no negócio, necessitam trocar dados/arquivos entre si ou com terceiros. A solução proposta visa poder ser escalável, independentemente de linguagem de programação escolhida e ser tolerante a falhas, independente do cliente a ser utilizado para envio/download dos dados.

O cenário que inicialmente parece simples pode ser expandido em muitos sentidos, como por exemplo o de autenticação, que não será abordado aqui, e substitui muitos casos de utilização de FTP que depende de um protocolo no qual o cliente é responsável por garantir seu envio/entrega, e não o protocolo.

A seguir, será demonstrada uma aplicação que utiliza da metodologia de uma arquitetura *lambda* num caso genérico, no qual os atores precisam sequencialmente enviar dados, baixar dados e realizar transferências automáticas de dados.

2.5.1 ENVIO DE DADOS

Nesse cenário para envio e armazenamento de arquivo, a proposta que pode ser adaptada e o caso de uso base será:

1. o usuário informa ao sistema o tamanho do arquivo a ser enviado, e o tamanho dos conjuntos desejados.
2. sistema estou enviando 1MB de arquivo e desejo enviar partes de 100KB.
3. o sistema devolve ao usuário de acordo com as informações enviadas.
   host.com.br/api/envio/<hash>/parte1
   ...
   host.com.br/api/envio/<hash>/parte10
4. o sistema devolve ao usuário um fragmento de código (*snippet*) para envio do arquivo via *Shellscript*, ou alguma outra linguagem de programação desejada.
5. o usuário executa o *snippet* que deverá:
   1. Criar uma frase secreta de criptografia (*secret phrase)* para o arquivo.
   2. Dividir o arquivo em memória ou em arquivos conforme o solicitado pela API (*Application Programming Interface* ou Interface de Programação de Aplicações).
   3. Enviar cada parte com sua determinada *phrase hash* para a API.
6. O sistema recebe as partes do arquivo, mantém em um sistema de arquivos criptografados com as partes desejadas.

7. O sistema fornece ao usuário a possibilidade de compartilhar seus arquivos enviados com quem desejar apenas fornecendo uma API, e o cliente que desejar baixar fornecendo a *hash* para descriptografar.

2.5.2 DOWNLOAD DE DADOS

1. O usuário informa ao sistema uma *hash* e deseja baixar o arquivo.
2. O sistema fornece ao usuário diversas APIs para baixar o arquivo, sendo que na primeira rota ele poderá consultar informações do arquivo.
   host.com.br/api/download/hash/info
   host.com.br/api/download/hash/parte1
   …
   host.com.br/api/download/hash/parte10
3. O sistema fornece ao usuário um *snippet* de código para baixar o arquivo.

2.5.3 TRANSFERÊNCIA SERVIDOR PARA SERVIDOR

O sistema necessita realizar trocas de arquivos periódicas entre a Empresa A e a Empresa B; no caso, a Empresa A seria conduzida pelo usuário A e a Empresa B seria conduzida pelo usuário B. A empresa A está enviando arquivos para empresa B:

1. O Usuário A cria uma solicitação de troca periódicas para o sistema e cadastra a Empresa B como destinatária, cadastrando o Usuário B como responsável.
2. O sistema pede para o Usuário B um *webhook*.
3. O sistema notificará esse *webhook* sempre que um novo arquivo estiver disponível para ele.
4. O sistema envia uma notificação para o *webhook* com informações do arquivo.
5. O Usuário A faz a solicitação de envio de um arquivo com uma *hash* previamente cadastrada.
6. O Usuário A executa o fluxo de envio de arquivo.
7. O Usuário B é notificado que existe um arquivo disponível.
8. O Usuário B realiza o processo de download do arquivo.
   host.com.br/api/share/<hash>
9. Quando o Usuário B termina de baixar uma parte do arquivo, o Usuário B sinaliza o servidor.

host.com.br/api/share/<hash>/parte/completed

10. O servidor realiza o controle de partes baixadas/não-baixadas.

    Caso possua falha em alguma das partes, o servidor sinaliza o Usuário B automaticamente, que retorna a baixar uma parte ainda não completada.
11. O Usuário B termina de baixar o arquivo inteiro e sinaliza ao servidor.

    host.com.br/api/share/<hash>/completed
12. O Usuário B só pode baixar o arquivo novamente caso autorizado pelo fornecedor.
13. O arquivo baixado precisa ser descriptografado pelo Usuário B, que, por algum outro meio, de preferência offline, recebe a chave de descriptografia do arquivo, garantindo assim que os arquivos são seguros.

3 CONCLUSÃO

Após a condução de estudos e leitura de artigos e pesquisas na área, é possível observar que o tema de **dados** não é novo, mas também não está alocado no futuro. Ele sempre será **presente**, acompanhará as tendências e ferramentas da época, gerando novos problemas e soluções.

Reunindo todas as informações aqui presentes, pode-se concluir que *DataOps* não é apenas *DevOps* para Análise de Dados, mas vai muito além, significam novos atores, processos e principalmente o foco e a entrega de valor.

Além deste trabalho atingir seu objetivo propondo uma análise sobre o tema para ser discutido nacionalmente e de livre acesso para entusiastas e pequenas empresas, também oferece uma arquitetura possível de ser utilizada em diversos casos.

A partir dessa conclusão, indaga-se acerca do tema proposto: o *DataOps* não será necessário um dia?; a arquitetura abordada neste artigo será um dia não mais necessária?; os problemas existentes nessa área agora seriam com governança e documentação dos dados em foco?

REFERÊNCIAS

AL-SAADOON, Laith, 2017. Unite Real-Time and Batch Analytics Using the Big Data Lambda Architecture, Without Servers! Disponível em: https://aws.amazon.com/pt/blogs/big-data/unite-real-time-and-batch-analytics-using-the-big-data-lambda-architecture-without-servers/. Acesso em: 25 maio, 2021.

CGI.br/NIC.br, Pesquisa sobre o Uso das Tecnologias de Informação e Comunicação nos Domicílios Brasileiros. TIC Domicílios 2019.

MANIFESTO, 2001. Agile. Disponível em< http://agilemanifesto. org>. Acesso em 18 de abril, 2022.

MARR, Bernard. Big data: 20 mind-boggling facts everyone must read, 2015. Disponível em: https://www.forbes.com/sites/bernardmarr/2015/09/30/big-data-20-mind-boggling-facts-everyone-must-read. Acesso em: 18 de abril, 2022.

MEHTA, Dhananjay. Building a scalable distributed data platform using lambda architecture. 2017.

MUNAPPY, Aiswarya Raj et al. From ad-hoc data analytics to dataops. In: Proceedings of the International Conference on Software and System Processes. 2020. p. 165-174.

NEW RELIC. What is a DevOps?. Disponível em: https://newrelic.com/devops/what-is-devops. Acesso em: 25 de maio, 2021.

PERRON, Matthew et al. Starling: A scalable query engine on cloud functions. In: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. 2020. p. 131-141.

RED HAT. What is Serverless?, 2017. Disponível em: https://www.redhat.com/pt-br/topics/cloud-native-apps/what-is-serverless. Acesso em 13 de julho, 2021.

RED HAT. Tipos de cloud computing. 2018. Disponível em https://www.redhat.com/pt-br/topics/cloud-computing/public-cloud-vs-private-cloud-and-hybrid-cloud. Acessado em 25 de Maio, 2021.