UNIVERSIDADE FEDERAL DE SANTA CATARINA

CENTRO TECNOLÓGICO

DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

CURSO DE GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO

Análise da estrutura e formatação de dados para comunicação eficiente entre ambientes de computação em nuvem

Rafael Begnini de Castilhos

Florianópolis, Santa Catarina

2024

UNIVERSIDADE FEDERAL DE SANTA CATARINA

CENTRO TECNOLÓGICO

DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

CURSO DE GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO

Análise da estrutura e formatação de dados para comunicação eficiente entre ambientes de computação em nuvem

Rafael Begnini de Castilhos

Trabalho de Conclusão de Curso do Curso de Graduação em Ciências da Computação do Centro Tecnológico da Universidade Federal de Santa Catarina apresentado como parte dos requisitos para obtenção do título de Bacharel em Ciências da Computação.

Orientador: Prof. Carlos Becker Westphall, Dr.

Florianópolis, Santa Catarina

2024

Ficha catalográfica para trabalhos acadêmicos
[Elemento obrigatório.]

[Insira neste espaço a ficha catalográfica para trabalhos acadêmicos.]

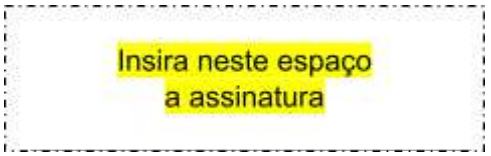
[A ficha é elaborada pelo(a) autor(a) no seguinte link: http://portalbu.ufsc.br/ficha]

Rafael Begnini de Castilhos

Análise da estrutura e formatação de dados para comunicação eficiente entre ambientes de computação em nuvem

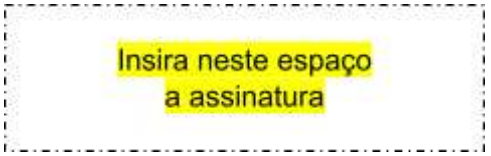
Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do título de Bacharel em Ciências da Computação e aprovado em sua forma final pelo Curso de Ciências da Computação.

Local [inserir local da defesa], [dia] de [mês] de [ano].



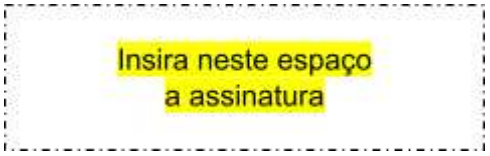
Coordenação do Curso

Banca examinadora



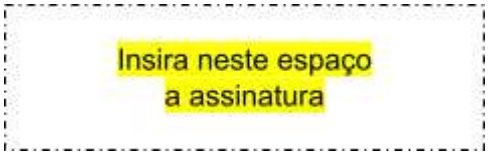
Prof.(a) Carlos Becker Westphall, Dr.(a)

Orientador(a)



Prof. Douglas Dyllon Jeronimo de Macedo, Dr.(a)

Universidade Federal de Santa Catarina



Prof.(a) Carla Merkle Westphall, Dr.(a)

Universidade Federal de Santa Catarina

Florianópolis, 2024.

DEDICATÓRIA

Dedico este trabalho aos meus pais, que sempre souberam que o melhor investimento para os filhos é a educação.

AGRADECIMENTOS

Primeiramente, gostaria de agradecer meu orientador Professor Dr. Carlos Becker Westphall por todo conhecimento transmitido, pelo apoio e instruções na elaboração deste trabalho.

Também agradeço minha família, por todo incentivo e inspiração que me proporcionaram nessa jornada. Em especial à minha mãe Marli, meu pai Nilson, meu irmão Rodrigo e minha namorada Maria Eduarda.

Agradeço a Universidade Federal de Santa Catarina pela oportunidade de me permitir evoluir intelectualmente, profissionalmente e pessoalmente. Juntamente com todos os servidores e profissionais da comunidade, que são responsáveis pela alimentação, limpeza e organização.

Sem a participação de cada uma dessas pessoas, essa conquista não seria possível.

“Lembre-se que as pessoas
podem tirar tudo de você,
menos o seu conhecimento.”
(Albert Einstein)

RESUMO

A computação em nuvem teve um crescimento exponencial, principalmente para aplicativos da web comerciais. De acordo com o Instituto Nacional de Padrões e Tecnologia (NIST), “a computação em nuvem é um modelo para permitir acesso de rede onipresente, conveniente e sob demanda a um conjunto compartilhado de recursos de computação configuráveis (por exemplo, redes, servidores, armazenamento, aplicativos e serviços) que podem ser rapidamente provisionados e liberados com o mínimo de esforço de gerenciamento e/ou interação com o provedor de serviços”. Por essas razões, os aplicativos cada vez mais frequentemente realizam trocas de informações entre serviços com finalidade de compartilhar, replicar, armazenar e entre outras funcionalidades características de um sistema distribuído, e, com isso, se faz necessário definição da estrutura e formatação por parte de desenvolvedores, arquitetos e engenheiros. A escolha deste formato e protocolo implica diretamente no tamanho, na latência e eficiência da comunicação. Será realizado o desenvolvimento de uma aplicação que atuará como emitente que enviará dados fictícios de diferentes tamanhos de um comércio eletrônico para um destinatário. Posteriormente, será efetuado monitoramento dos dados trafegados, sendo possível extrair métricas e avaliar os resultados obtidos de maneira comparativa e exploratória. Logo, esse trabalho visa analisar, comparar e encontrar as opções de formato de dados e protocolos já existentes que são satisfatórias para este cenário, potencializando operações escaláveis e utilizando menos recursos.

Palavras-chave: Computação em nuvem. Serialização. Desserialização. Sistemas distribuídos. Redes de computadores.

ABSTRACT

Cloud computing has seen exponential growth, particularly for commercial web applications. According to the National Institute of Standards and Technology (NIST), “Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications and services) that can be rapidly provisioned and released with minimal management effort and/or service provider interaction”. For these reasons, applications increasingly exchange information between services in order to share, replicate, store and among other features characteristic of a distributed system, and, therefore, it is necessary to define the structure and formatting on the part of developers, architects and engineers. The choice of this format and protocol directly affects the size, latency and efficiency of the communication. Preambularly, an application will be developed that will act as an issuer that will send fictitious data of different sizes from an electronic commerce to a recipient. Subsequently, the traffic data will be monitored, making it possible to extract metrics and evaluate the results obtained in a comparative and exploratory manner. Therefore, this work aims to analyze, compare and find existing data format and protocol options that are satisfactory for this scenario, enhancing scalable operations and using fewer resources.

Keywords: Cloud computing. Serialization. Deserialization. Distributed systems. Computer network.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AWS	Amazon Web Services
VMs	Virtual machines
SLA	Service level agreement
SOA	Service oriented architecture
SaaS	Software as a service
PaaS	Plataform as a service
IaaS	Infrastructure as a service
FaaS	Function as a service
EC2	Elastic Compute Cloud
ECS	Elastic Container Service
TI	Information technology
HTTP	Hypertext Transfer Protocol
API	Application Programming Interface
XML	Extensible Markup Language
JSON	Javascript Object Notation
AMI	Amazon Machines Images

SUMÁRIO

1 INTRODUÇÃO

A computação em nuvem é amplamente adotada na indústria como uma tecnologia que permite acesso fácil e barato ao processamento e armazenamento de dados. Essa tecnologia é sustentada por servidores físicos que hospedam máquinas virtuais (VMs), as quais são disponibilizadas aos usuários (JINDAL, et al., 2019). Além disso, a computação em nuvem permite provisionar serviços através da Internet de maneira escalável e sob demanda (EMEAKAROHA, et al., 2016).

Nas últimas décadas, muitas aplicações de software, que antes operavam de maneira standalone, como sistemas de reserva, sistemas bancários e comércio eletrônico, passaram a ser distribuídas. Segundo Abdullah, et al. (2022), um sistema distribuído é uma coleção de sistemas autônomos, estações de trabalho e servidores conectados que fornecem serviços ao cliente ou usuário virtualmente, utilizando um protocolo chamado middleware. O objetivo de um sistema distribuído é compartilhar recursos, aumentar a disponibilidade, o rendimento e a eficiência, além de superar falhas.

Um aspecto crucial para o funcionamento eficiente desses sistemas é a comunicação entre os serviços, que ocorre através de mensagens geradas por um serviço emitente, trafegadas pela rede e recebidas por um serviço destinatário. Para que essa comunicação seja efetiva, é necessário que o emitente serialize o dado a ser transacionado, estruturando-o de acordo com o protocolo utilizado, e que o destinatário deserialize o dado recebido. Serialização é o processo de converter um objeto ou estrutura de dados em um formato que possa ser facilmente armazenado ou transmitido, enquanto desserialização é o processo inverso, de reconstruir o objeto ou estrutura original a partir do formato serializado.

Diferentes mecanismos e protocolos podem ser utilizados para realizar a serialização e desserialização. Existem muitas aplicações cuja funcionalidade principal é o envio e recebimento de dados pela rede. O aumento do volume de dados intercambiados pela Internet tornou a seleção de um formato de serialização adequado cada vez mais importante (SUMARAY; MAKKI, 2012). A serialização e a desserialização de objetos é essencial para a comunicação eficiente entre nós de computação distribuída, especialmente em ambientes de execução potencialmente não uniformes (JANG, et al., 2020). Portanto, a relevância desse tema é destacada

pela enorme quantidade de troca de informações diárias em ambientes distribuídos em nuvem.

Este trabalho propõe desenvolver uma aplicação responsável por emitir e receber informações utilizando diferentes abordagens de serialização e desserialização. O objetivo é obter um conjunto de dados para extrair métricas, analisar e comparar os variados mecanismos utilizados. Isso permitirá identificar os métodos mais eficazes, objetivando aumentar a eficiência, reduzir custos e latência, além de identificar abordagens menos adequadas. Assim, este trabalho visa concluir quais são as melhores práticas para estruturar e formatar dados na comunicação entre dispositivos, potencializando operações escaláveis e utilizando menos recursos.

1.1 MOTIVAÇÃO

A computação em nuvem está crescendo rápido e constantemente (COSTELLO, 2019), e, com esta, o uso de microsserviços. Por isso, deseja-se saber o impacto na eficiência de uma solução quando escolhido um formato para as mensagens comunicadas entre os serviços.

Durante o desenvolvimento de sua atividade profissional o autor desenvolve aplicações distribuídas, utilizando microsserviços localizados em diferentes pontos do globo que comunicam-se entre si. Para estas aplicações, diferentes formatos para as mensagens, e diferentes modelos de computação foram considerados, mas uma análise profunda se mostra necessária, sendo assim, o propósito deste trabalho.

1.2 OBJETIVOS

Esta seção apresenta o objetivo geral e objetivos específicos deste trabalho.

1.2.1 OBJETIVO GERAL

O objetivo geral deste trabalho é desenvolver uma aplicação para trafegar mensagens, visando eficiência entre máquinas em ambiente de nuvem. Para isso, diferentes métodos de serialização, desserialização e modelos de computação serão considerados e comparados entre si.

1.2.2 OBJETIVOS ESPECÍFICOS

Os objetivos específicos deste trabalho são:

- Examinar a literatura referente a comunicação eficiente em ambiente de nuvem.
- Analisar estrutura e formatação na serialização e desserialização de objetos.
- Pesquisar diferentes técnicas e bibliotecas para serializar e desserializar objetos, a fim de enriquecer análises e comparações.
- Analisar modelos de computação em nuvem para comunicação entre máquinas.
- Comparar os dados obtidos nas análises, a fim de expor resultados satisfatórios e insatisfatórios.

1.3 ORGANIZAÇÃO DO TEXTO

O texto é organizado nos seguintes capítulos:

- Capítulo 1 - Introdução: apresenta o trabalho, sua motivação, justificativa, objetivos e organização do texto;
- Capítulo 2 - Fundamentação Teórica: apresenta os conceitos e fundamentação teórica necessários para a compreensão deste trabalho. Esta seção também apresenta o problema e uma solução ao mesmo tendo como base as referências bibliográficas;
- Capítulo 3 - Trabalhos Relacionados: apresenta o estado da literatura referente a formatos de serialização e comparação de serviços da computação em nuvem;
- Capítulo 4 - Desenvolvimento e Análise: descreve como foi realizado o processo para implementar o algoritmo proposto e são apresentados os resultados e análises. Esta seção também descreve os algoritmos para implementar a proposta;
- Capítulo 5 - Conclusão: apresenta as conclusões finais e trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 COMPUTAÇÃO EM NUVEM

A computação em nuvem é um modelo ubíquo, conveniente e de acesso compartilhado a recursos computacionais como servidores, armazenamento, aplicações e redes (MELL; GRANCE, 2011). Estes recursos compartilhados podem ser rapidamente provisionados e liberados com um esforço mínimo de gestão ou interação com o provedor de serviços (MELL; GRANCE, 2011).

Segundo Jennings e Stadler (2015), emerge, então, o paradigma de Computação em Nuvem (Cloud Computing), no qual um conjunto de recursos computacionais é compartilhado entre aplicativos que o acessam pela Internet. Este conceito pode também se referir a hardware e software de sistema que reside nos data centers que hospedam tais aplicativos. Os objetivos dos provedores de nuvem estão centrados no uso eficiente dos recursos, dentro de limites estabelecidos por um Acordo de Nível de Serviço, ou Service Level Agreement (SLA), que é um contrato formal entre o provedor e o usuário, cujo objetivo é definir os aspectos funcionais e não funcionais do serviço em termos quantitativos (JENNINGS; STADLER, 2015). De acordo com a sua infraestrutura e o modelo de provisionamento de recursos, podemos classificar a nuvem em quatro principais categorias (BUYYA, *et al*., 2009):

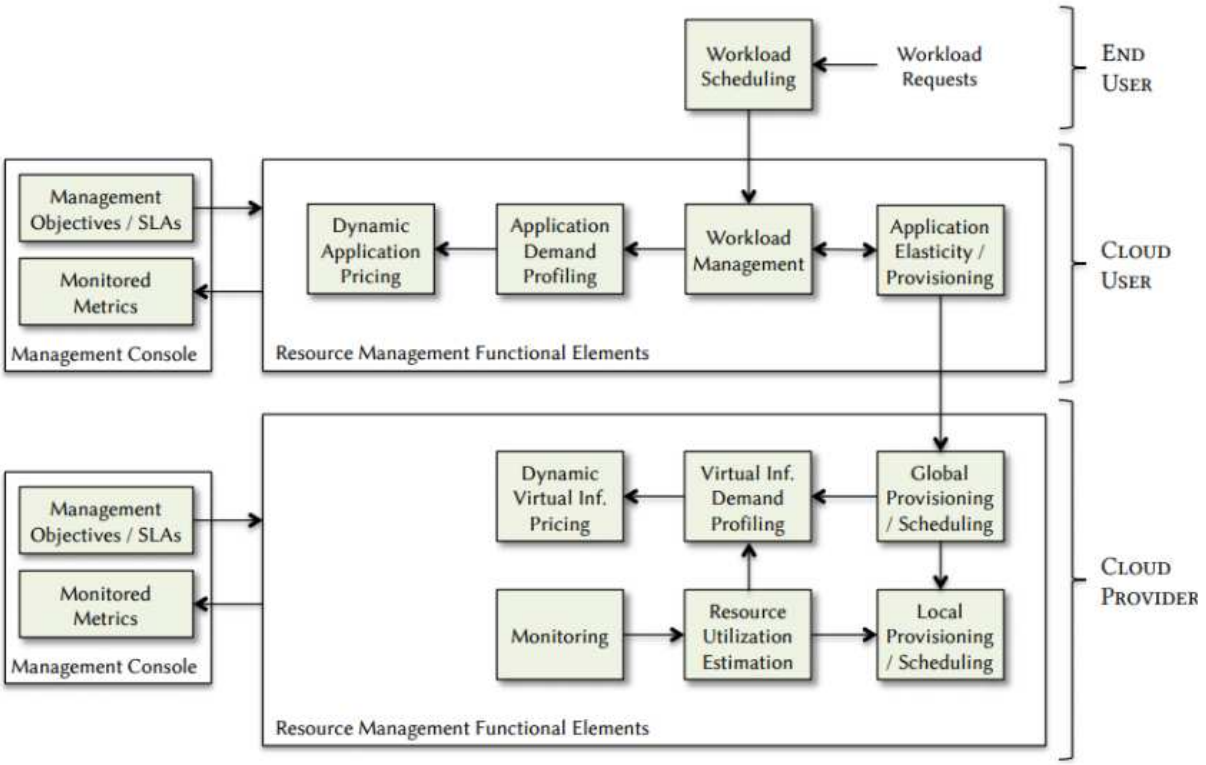
- Nuvem pública: a nuvem abrange o provisionamento de recursos a terceiros alugando-os com base no uso.
- Nuvem privada: compreende as infraestruturas privadas mantidas e utilizadas por indivíduos e organizações.
- Nuvem híbrida: pode ser definida como o cenário em que uma organização estende a sua nuvem privada ao alugar parte dos seus recursos de uma nuvem pública.
- Nuvens comunitárias: nesta categoria, os recursos são contribuídos por vários indivíduos e/ou organizações de forma descentralizada.

Este trabalho considera apenas a nuvem pública, cujas propriedades são definidas na seção 2.1.1, seguidas da definição de modelos de serviço na seção 2.1.2.

De acordo com Jennings e Stadler (2015), de forma geral, podemos destacar três papéis principais num ambiente de nuvem, conforme demonstrado na figura 1:

- Provedor da nuvem: Gerencia os data centers, provendo estes recursos de forma abstrata para os usuários da nuvem ou usuários finais e é responsável por cumprir os SLAs.
- Usuário da nuvem: utiliza serviços da nuvem para oferecer aplicativos para os usuários finais, procurando minimizar custos, maximizar lucros e manter em sintonia a quantidade de recursos requerida pelos seus clientes com a quantidade alugada do provedor da nuvem.
- Usuário final: Num ambiente comercial, é o cliente final. Apesar de geralmente não atuar diretamente no gerenciamento da nuvem, pode influenciar as decisões tomadas pelo usuário da nuvem e provedor da nuvem.

Figura 1: Papéis principais em um ambiente de nuvem.



Fonte: (JENNINGS; STADLER, 2015).

2.1.1 Propriedades

Em (MELL; GRANCE, 2011) são definidas como características básicas de um ambiente de computação em nuvem:

1. Acesso aos recursos sob demanda: O usuário pode escolher quais serviços/componentes ele irá usar dentro do que é disponibilizado pela nuvem sem a necessidade de interação com administradores;

2. Disponibilidade: Ambientes de computação em nuvem devem estar sempre disponíveis para o usuário pela internet independente da plataforma de acesso (tablet, smartphone, desktop e outros);

3. Recursos compartilhados: Os recursos físicos da nuvem devem poder atender diferentes usuários simultaneamente;

4. Elasticidade: Conforme um usuário solicita mais recursos, a nuvem deve alocar mais recursos para este usuário, evitando a degradação dos serviços prestados, assim como se um usuário deixar de usar recursos, a nuvem pode remover recursos desnecessários;

5. Monitoramento de serviços: A nuvem deve ser capaz de monitorar o uso de recursos por cada serviço, proporcionando transparência aos fornecedores e usuários do serviço.

2.1.2 Modelos de serviços

Mell e Grance (2011) descrevem os modelos de serviço disponibilizados pela nuvem conforme o nível de abstração que é oferecido, conforme demonstrado na figura 2, como sendo:

1. Software as a Service (SaaS): o provedor de nuvem fornece uma aplicação final via rede para usuários. O usuário não tem controle algum sobre a infraestrutura que hospeda a aplicação ou da plataforma que a aplicação usa, ele apenas consegue usar a aplicação;

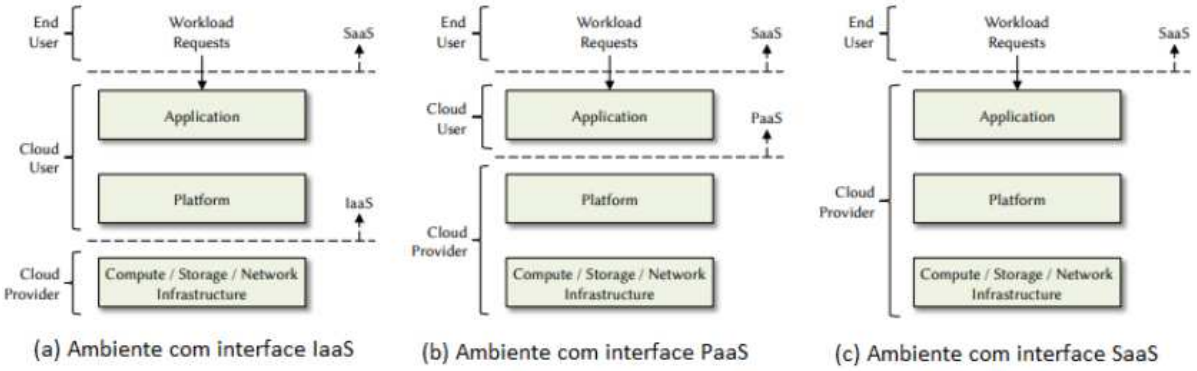
2. Platform as a Service (PaaS): são disponibilizadas diferentes plataformas para o usuário desenvolver e hospedar suas próprias aplicações. Neste nível o usuário tem controle total das aplicações, e pode configurar por completo a plataforma. O usuário não possui acesso a recursos do sistema operacional que hospeda a plataforma;

3. Infrastructure as a Service (IaaS): a nuvem fornece toda a infraestrutura para o usuário, como processador, disco, memória e rede. Neste nível o usuário tem uma máquina virtual pronta para uso, ele define quais plataformas ele irá usar e quais aplicações ele irá executar em cada plataforma, porém o usuário não tem acesso e controle dos componentes físicos do ambiente. Cada modelo de nuvem

descrito visa um público diferente com diferentes propósitos, o software como serviço visa mais usuários de aplicações. O modelo de plataforma como serviço procura atender os desenvolvedores que usarão as plataformas para hospedar suas aplicações. Em contrapartida, o modelo de fornecimento de infraestrutura como serviço tem como foco empresas que desenvolvem serviços e os hospedam na infraestrutura de ambientes de computação em nuvem para minimizar seus custos com infraestrutura.

4. Function as a Service (FaaS): O provedor de nuvem gerencia os recursos, o ciclo de vida e a execução orientada a eventos de funções fornecidas pelo usuário (EYK, *et al*, 2017). A computação sem servidor baseia-se no conceito de função como computação, que decorre de uma longa tradição de abstração de nível cada vez mais alto e especialização da computação em nuvem. Em sistemas modernos, protocolos baseados em eventos permitem que sistemas em um ecossistema se comuniquem sem dependência excessiva dos detalhes de implementação de cada sistema individualmente.

Figura 2: Níveis de abstração na nuvem.



Fonte: (JENNINGS; STADLER, 2015).

2.1.3 Serviços de computação

O Amazon EC2 (Elastic Compute Cloud) é um serviço de computação em nuvem que fornece capacidade de computação escalável na nuvem da Amazon Web Services (AWS). O EC2 permite que os usuários criem e configurem instâncias virtuais, que são basicamente servidores virtuais, com diferentes capacidades de CPU, memória, armazenamento e rede, para atender às necessidades específicas de suas aplicações. O funcionamento do EC2 envolve a criação de instâncias a partir de imagens de máquina da Amazon (AMI), que podem ser personalizadas

para incluir o sistema operacional e o software necessário. As instâncias podem ser dimensionadas horizontalmente ou verticalmente, permitindo que os usuários ajustem a capacidade de computação conforme a demanda, pagando apenas pelo tempo de uso.

O Amazon ECS (Elastic Container Service) é um serviço gerenciado de orquestração de contêineres que permite executar, parar e gerenciar containers Docker em um cluster de instâncias do EC2, que é uma solução de contêineres sem servidor. O ECS facilita a implantação de aplicativos em contêineres, fornecendo uma maneira eficiente de gerenciar a escalabilidade e a disponibilidade dos contêineres, além de integrar-se de forma nativa com outros serviços da AWS. No ECS, os usuários definem tarefas que descrevem um ou mais contêineres que devem ser executados juntos. O ECS lida com o balanceamento de carga, a descoberta de serviços, a integração com redes e a escalabilidade automática, simplificando o gerenciamento da infraestrutura.

O AWS Lambda é um serviço de computação sem servidor que permite executar código em resposta a eventos sem a necessidade de provisionar e gerenciar servidores. Com o Lambda, os usuários podem executar trechos de código (chamados de funções Lambda) que são disparados por eventos específicos, como chamadas HTTP por meio do API Gateway. O funcionamento do Lambda envolve a definição de funções que contêm o código a ser executado, juntamente com os eventos que disparam a execução dessas funções. O serviço gerencia automaticamente todos os recursos de computação necessários para executar as funções, incluindo o provisionamento de servidores, escalabilidade, balanceamento de carga e monitoramento, permitindo que os desenvolvedores se concentrem exclusivamente no código e suas regras de negócio. Os usuários pagam apenas pelo tempo de execução do código, tornando o Lambda uma opção econômica para aplicações orientadas a eventos.

2.2 MICROSSERVIÇOS

De acordo com Dragoni, et. al, um microsserviço é um processo coeso e independente interagindo por meio de mensagens. Como exemplo, considere um serviço destinado a computar cálculos. Para chamá-lo de microsserviço, ele deve fornecer operações aritméticas solicitadas por meio de mensagens, mas não deve

fornecer outras funcionalidades (possivelmente vagamente relacionadas, ou seja, que realizam operações extras subsequentes que não estão sob o mesmo domínio) como plotagem e exibição de funções.

O termo “microsserviço” tem sido amplamente utilizado desde 2012 para se referir a aplicativos desenvolvidos como um conjunto de aplicativos relativamente pequenos, consistentes, isolados e serviços autônomos implantados de forma independente, com um propósito único e claramente definido.

Os microsserviços são o oposto das arquiteturas monolíticas, onde os aplicativos geralmente são implantados como um único pacote em um contêiner da Web, como o Tomcat (Apache Tomcat, http://tomcat.apache.org/) ou JBoss (JBoss, http://www.jboss.org), e podem ser desenvolvidos independentemente por diferentes equipes de desenvolvimento. (TAIBI; LENARDUZZI; PAHL, 2017).

Nesse viés, muitos desenvolvedores e arquitetos de software estão promovendo esse modelo arquitetônico, levando em consideração os custos da migração do sistema monolítico. Entretanto, alguns profissionais ainda hesitam em adotar esse modelo pois ainda não conhecem os benefícios e malefícios. Com isso, torna-se notável que é de suma importância estudar e avaliar as motivações de utilização desse estilo.

Do ponto de vista técnico, os microsserviços devem ser componentes independentes implantados conceitualmente de forma isolada e equipados com persistência de memória dedicada, fazendo uso de bancos de dados e sistemas de arquivos.

2.3 FORMATO DOS DADOS

Em meio a computação contemporânea, a seleção do formato de dados adequado desempenha um papel essencial na eficiência, desempenho e interoperabilidade de sistemas. Diversos formatos tiveram projeção, cada um com características distintas que influenciam diretamente a manipulação e transmissão de informações.

Para escolher um formato de dados, existem alguns aspectos que devem ser considerados, entre eles:

- O formato deve ser compatível com outros sistemas que precisam acessar os dados.

- O formato deve ser eficiente em termos de espaço, memória e desempenho.
- O formato juntamente com a aplicação devem garantir a segurança e confidencialidade dos dados.

A distinção entre diferentes formatos de dados é essencial para compreender como cada um se adapta a distintas necessidades de aplicativos e sistemas. Características específicas moldam a utilidade e eficácia de cada formato em contextos variados. A presença ou ausência de uma especificação formal é um fator determinante. Formatos como XML e JSON possuem especificações que definem a estrutura e semântica dos dados, proporcionando consistência e interoperabilidade.

Outro aspecto crucial é a natureza binária ou textual do formato. Formatos binários, exemplificados por Protobuf, BSON, Avro, MessagePack, Kryo e entre outros, focam em eficiência e compactação, enquanto formatos textuais, como JSON e XML, oferecem legibilidade humana, embora possam ser mais extensos (SUMARAY; MAKKI. 2012).

2.4 SERIALIZAÇÃO E DESSERIALIZAÇÃO

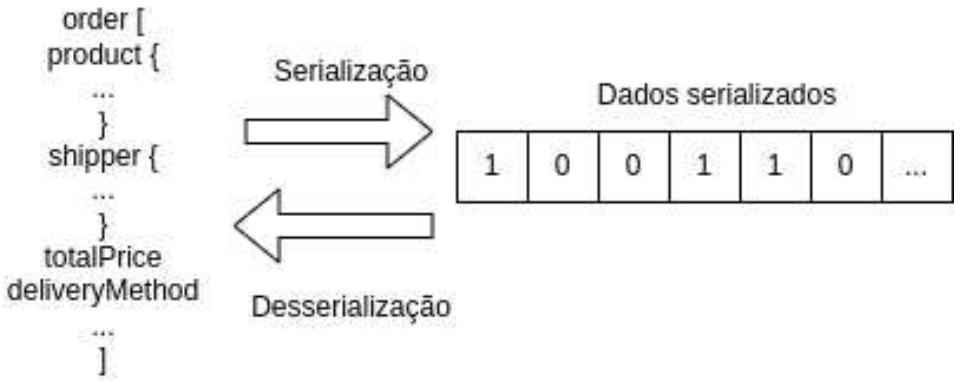
A serialização e a desserialização de objetos e dados são processos essenciais na programação que envolvem a conversão de estruturas de dados complexas, como objetos, em um formato que possa ser facilmente armazenado, transmitido e posteriormente recriado. Esses processos são cruciais em cenários onde é necessário preservar a integridade dos dados ao longo do tempo, transmitir informações entre diferentes sistemas ou compartilhar dados com outras partes.

Serializar é o processo de transformar objetos, ou estruturas de dados, em uma sequência de bytes, para que possam ser salvos em disco ou transportados pela rede (MICROSOFT, 2018). O processo inverso, a desserialização, por sua vez, envolve a conversão dessa sequência de bytes de volta para objetos ou estruturas de dados originais. Como visto na figura 3, existe um objeto fictício em memória que é serializado (da esquerda para direita) em uma sequência de bytes. Já o processo na direção inversa (da direita para esquerda), também torna-se possível, executando a desserialização.

Habitualmente, uma linguagem de programação orientada a objetos, suporta a serialização de seus objetos nativamente, como é o caso de C#, Java e entre outras. Contudo, a serialização binária de forma nativa transfigura um obstáculo para

a interoperabilidade entre sistemas, visto que muitas linguagens de programação antigas ou com outros paradigmas, fica difícil ou impossível realizar o processo de desserialização. Porém, um contraexemplo é a linguagem JavaScript, na qual produz o resultado da serialização em JSON (MOZILLA, 2017), formato o qual é um dos analisados neste trabalho, na seção 4.3.2.

Figura 3: Processo de serialização e desserialização.



Fonte: O Autor.

Ademais, um formato de dados pode ser textual, assim sendo legível para humanos, porém mais restrito quanto a possíveis representações, ou em bytes (não textuais), tendo total liberdade para definir sua representação dos dados em bytes. Compilando os fatores apresentados, serialização engloba:

- Formato dos dados serializados;
- Processo de serializar os dados;
- Processo de desserializar os dados.

O formato de serialização conta com dois fatores de eficiência:

- Eficiência temporal, diretamente relacionada com o tempo de execução do algoritmo;
- Eficiência espacial, que está associada à quantidade de memória utilizada pelo algoritmo durante sua execução .

Além de contar com outras características qualitativas, das quais se destacaram:

- Interoperabilidade (número de linguagens de programação suportadas);
- Binário ou textual.

Sintetizando, a serialização é utilizada em diversas áreas de tecnologia da informação e programação, entre elas, está a persistência de dados e armazenamento em banco de dados, que são utilizadas para salvar e carregar o estado de um aplicativo em um ponto específico ou armazenar objetos complexos, permitindo que sejam recriados quando necessários. Além disso, o compartilhamento de dados e comunicação entre sistemas, na qual este trabalho irá dissertar, com ênfase na troca de informações entre diferentes sistemas localizados na nuvem, mantendo a estrutura e integridade de dados.

2.5 COMUNICAÇÃO VIA PROTOCOLO HTTP

O HTTP é um protocolo cliente-servidor: as requisições são enviadas por uma entidade, o agente-usuário (ou um *proxy* em nome dele). A maior parte do tempo, o agente-usuário é um navegador da Web, mas pode ser qualquer coisa, como por exemplo um robô que varre a Web para preencher e manter um índice de mecanismo de pesquisa e coletar informações (MOZILLA, 2017).

O Hypertext Transfer Protocol (HTTP) é um protocolo de grande sucesso. No entanto, o HTTP/1.1 tem várias características que têm um efeito geral negativo no desempenho. Visando otimizar o protocolo mais utilizado para comunicação na internet, o HTTP/2 fornece um transporte otimizado para semântica HTTP. O HTTP/2 suporta todos os principais recursos do HTTP/1.1, mas pretende ser mais eficiente de várias maneiras (BELSHE, 2017).

Uma conexão é controlada na camada de transporte, e portanto fundamentalmente fora do controle do HTTP. Entretanto o HTTP não requer que o protocolo de transporte utilizado seja baseado em conexões, só requer que seja confiável ou não perca mensagens. Dentre os dois protocolos de transporte mais comuns na internet, o TCP é confiável e o UDP não. Portanto, o HTTP utiliza o padrão TCP, que é baseado em conexão, mesmo que nem sempre seja obrigatório o uso de uma conexão (MOZILLA, 2017).

Em uma requisição, há vários elementos que constituem o contexto, entre as principais:

- Método: É um verbo como GET, POST, DELETE, PUT, etc, na qual define a operação que o cliente quer fazer. Como por exemplo pegar um recurso

(GET), publicar dados (POST), atualizar dados (PUT), remover um recurso (DELETE), entre outros.

- Caminho: O recurso a ser acessado, constituído pelo protocolo (https://), pelo domínio e/ou porta TCP.
- Cabeçalhos: Informações adicionais para os servidores, podendo incluir configurações, autenticação e preferências do cliente.
- Corpo: Em alguns casos, como no POST ou PUT, podem ser enviadas informações.

De maneira análoga à requisição, na resposta os principais elementos são:

- Código de status: Um código indicando se a requisição foi bem sucedida, ou não, e por quê.
- Mensagem de status: Uma pequena descrição informal sobre o código de status.
- Cabeçalhos: Análogos aos da requisição.
- Corpo: Contém dados e informações importantes do recurso requisitado.

2.6 VISÃO GERAL

A computação em nuvem tem ganhado cada vez mais destaque e adesão nos últimos anos devido, principalmente, ao seu baixo custo de alocação e gerenciamento. Conforme o crescimento da demanda por serviços ofertados por ambientes computacionais, a estrutura necessária para hospedar esses ambientes aumenta em tamanho e complexidade, impactando diretamente no gerenciamento destes ambientes (WEINGÄRTNER; BRÄSCHER; WESTPHALL, 2015) e (GERONIMO, *et al*., 2013).

Analisando e estudando os trabalhos relacionados, constatou-se a necessidade de comparar os formatos de serialização e desserialização no cenário de computação em nuvem, abrangendo diferentes modelos de computação e tipos de dados. Como apresentado anteriormente nos trabalhos relacionados, existem diferentes comparações entre formatos, porém nenhum deles leva em consideração os modelos de computação em nuvem.

No âmbito de dois serviços na nuvem se comunicando sobre o protocolo HTTP, seguindo as premissas de SOA, um serviço não tem conhecimento da implementação do outro, mas apenas da sua interface disponibilizada. Para que

dados sejam comunicados através desta interface, um formato interoperável é adotado, como JSON ou XML (BRAY, 2017; W3C, 2016).

Nesta situação, desperta-se interesse em compreender o efeito dos diversos padrões na eficiência da comunicação entre estes dois serviços. Ao selecionar esse padrão em um contexto real, também se considera suas restrições, como variedades que podem ser representadas e se é ou não necessário definir a estrutura dos dados transmitidos durante o processo de compilação. São avaliadas exclusivamente as métricas quantitativas de eficiência dos formatos escolhidos ao operarem como serviços na nuvem, de modo que os resultados sirvam como parâmetro na seleção do formato em situações reais.

De maneira análoga aos estudos apresentados no capítulo 3, são considerados como elementos determinantes o tamanho dos dados serializados e o custo da serialização e desserialização. Enquanto o tamanho permanece imune a influências externas, dependendo somente dos dados a serem serializados, já o custo das operações de serialização e desserialização é afetado pelos recursos de hardware disponíveis e pela biblioteca utilizada para a conversão, que nem sempre é padronizada.

Além disso, a comunicação entre os serviços ocorre por meio da Internet, e o tempo dessa comunicação é influenciado pelo tamanho da mensagem transmitida. Assim, o tempo de comunicação também é um fator de interesse para análise. Logo, é esperado que quanto maior for o tamanho em bytes da mensagem transmitida, maior será o tempo de comunicação. Do mesmo modo, a latência inicial e a capacidade de transferência variam de acordo com a distância, o tráfego e a qualidade do canal entre os dois serviços.

2.7 TIPOS DE DADOS

Objetivando abranger diversos tipos de dados, os experimentos são executados com: booleanos, números inteiros, números em ponto flutuante, caracteres, cadeia de caracteres e objetos. Os exemplos utilizam a sintaxe da linguagem de programação Java.

2.8 FORMATOS ANALISADOS

Os formatos escolhidos originam-se dos mais utilizados entre os trabalhos relacionados, escolhendo dois formatos para o tipo textual e dois formatos para o

tipo binário. Dos formatos textuais, foram escolhidos XML e JSON, na qual o JSON foi considerado ser melhor na comparação segundo os trabalhos relacionados. Já os formatos binários, foram escolhidos KRYO e MSGPACK, na qual o KRYO foi considerado ser melhor na comparação do trabalho relacionado 3.2.

2.8.1 XML

A Extensible Markup Language permite definir e armazenar dados de maneira compartilhada. A XML oferece suporte ao intercâmbio de informações entre sistemas de computador, como sites, bancos de dados e aplicações de terceiros. Regras predefinidas facilitam a transmissão de dados como arquivos XML em qualquer rede, pois o destinatário pode usar essas regras para ler os dados com precisão e eficiência (AMAZON WEB SERVICES, 2022).

Foi projetado para ser legível e compreensível por seres humanos, sendo independente de plataforma e linguagem de programação, permitindo a criação de tags personalizadas, podendo definir a própria estrutura de dados para representar informações específicas de acordo com necessidades do domínio ou caso de uso. É uma tecnologia subjacente em milhares de aplicações, variando de ferramentas comuns de produtividade, como processamento de texto, software de publicação de livros e até mesmo sistemas complexos de configuração de aplicações.

Em minha experiência profissional atuando como desenvolvedor de software com aplicações distribuídas, pude interagir e identificar problemas de performance e tempo de execução ao utilizar XML para trafegar dados entre sistemas legados.

2.8.2 JSON

JSON, ou JavaScript Object Notation, é um formato textual baseado em um subconjunto da linguagem JavaScript. Ademais, é um formato independente de linguagem, de fácil leitura e escrita para humanos, mas também de processamento simples para máquinas (BRAY, 2017). Possui duas estruturas básicas:

1. Vetor: uma coleção ordenada de valores;
2. Objeto: um conjunto não-ordenado de pares nome / valor.

Valores podem ser qualquer um dos tipos básicos: string, número, booleano (true ou false) ou nulo (null). Também podem ser um objeto ou vetor, permitindo aninhamentos. Por padrão, JSON não possui uma definição de schema para as estruturas representadas.

Para serialização e desserialização dos experimentos, foi utilizada a biblioteca Gson de propriedade do Google. O Gson pode trabalhar com objetos Java arbitrários, incluindo objetos pré-existentes dos quais você não possui o código-fonte.

2.8.3 KRYO

O Kryo é frequentemente utilizado em aplicações que precisam armazenar ou transmitir dados de forma compacta, como em jogos, sistemas distribuídos e outras aplicações que requerem alta performance de serialização, isso se torna viável pois é otimizado para minimizar o consumo de memória durante o processo (SWEET, 2023). A biblioteca de serialização e desserialização Kryo foi criada pela EsotericSoftware, uma empresa de desenvolvimento de software fundada por Nathan Sweet, que é o principal desenvolvedor por trás do projeto Kryo, que é uma biblioteca de código aberto em Java usada para serialização e desserialização de objetos, incluindo tipos primitivos, coleções (Listas, Mapas, Enumerações, etc.) de forma rápida e eficiente.

2.8.4 MSGPACK

MessagePack, ou MsgPack, é um formato de serialização binário baseado em JSON, e, por isso, herda muitas de suas propriedades, das quais se destacam:

- Não necessita de um schema para definir seus dados.
- Utilizar os mesmos tipos de JSON, porém dividindo números entre suas representações em inteiros de tamanho variável e ponto flutuante de 64 bits.

O formato também permite “tipos de extensão”, que suportam a definição de um tipo arbitrário. O tipo timestamp, que define uma medida de tempo (i.e., data, hora e fuso-horário), é o único tipo de extensão definido por padrão pelo formato (FURUHASHI,2013).

Oficialmente, são mantidas apenas a definição do formato e bibliotecas para um pequeno conjunto de linguagens, e Java está incluso, possuindo a biblioteca nativamente, e por isso foi utilizada para os experimentos.

3 TRABALHOS RELACIONADOS

3.1 A Comparison of Data Serialization Formats For Optimal Efficiency on a Mobile Platform

No artigo de Audie Sumaray e S. Kami Makki, os autores examinam e comparam diferentes formatos de serialização de dados com o objetivo de identificar a opção mais eficiente em uma plataforma móvel.

A serialização de dados é um processo de conversão de objetos em uma sequência de bytes para facilitar seu armazenamento ou transmissão. Nos dispositivos móveis, onde os recursos como capacidade de armazenamento, largura de banda e processamento são limitados, é essencial escolher um formato de serialização que minimize o consumo desses recursos.

Os autores do artigo investigam quatro formatos de serialização amplamente utilizados: XML, JSON, Protobuf e Thrift. Eles conduzem uma série de experimentos para avaliar o desempenho desses formatos em termos de tamanho do arquivo serializado, velocidade de serialização e desserialização, e consumo de recursos (CPU e memória).

Os resultados da pesquisa mostram que o formato binário supera significativamente o formato textual em termos de tamanho do arquivo serializado, sendo consideravelmente menor. Além disso, o Protobuf e Thrift também se destaca na velocidade de serialização e desserialização, superando tanto o XML quanto o JSON.

Os autores também levam em consideração a complexidade e a facilidade de uso dos diferentes formatos. Embora o XML e o JSON sejam mais fáceis de ler e editar manualmente, o artigo ressalta que o protobuf oferece melhor desempenho em uma plataforma móvel, especialmente quando se lida com grandes volumes de dados.

Em conclusão, o artigo recomenda o uso do formato Thrift e Protobuf para serialização de dados em plataformas móveis devido à sua eficiência em termos de tamanho de arquivo, velocidade de serialização e desserialização. Ademais, postula que o formato XML deve ser evitado, e evidencia que a diferença entre Thrift e Protobuf é negligenciável, porém o Thrift pode ser compilado em uma quantidade maior de linguagens, suportando também dispositivos iOS. No entanto, os autores observam que a escolha do formato de serialização também depende dos requisitos específicos do aplicativo e do equilíbrio entre eficiência e facilidade de uso.

3.2 Smart Grid Serialization Comparison

Petersen et al, propõem uma comparação de diferentes formatos de serialização para aplicação em redes elétricas inteligentes (smart grids). As redes elétricas inteligentes são sistemas complexos que integram tecnologias de comunicação, medição e controle para otimizar a distribuição de energia elétrica. A serialização de dados desempenha um papel crucial nesse contexto, permitindo a troca eficiente e confiável de informações entre os dispositivos e sistemas que compõem uma smart grid.

Os autores analisam e comparam formatos de serialização comumente utilizados em aplicações de smart grid: XML, JSON, Protobuf, Protostuff, Kryo e Avro. Eles exploram características como tamanho do arquivo serializado, velocidade de serialização e desserialização, capacidade de extensibilidade e interoperabilidade dos formatos.

Por intermédio de experimentos e análises, os autores concluíram que o formato binário utilizando Protostuff, Protobuf e Kryo apresenta vantagens significativas em comparação com XML e JSON para aplicações em smart grids. O formato tem uma estrutura binária compacta, resultando em tamanhos de arquivo serializados menores e maior eficiência na transmissão e armazenamento de dados. Além disso, demonstra melhor desempenho em termos de velocidade de serialização e desserialização.

Outra vantagem destacada pelos autores é a capacidade de extensibilidade e a interoperabilidade oferecida pelo formato binário. Ele permite a definição precisa de tipos de dados complexos e oferece suporte a uma variedade de protocolos de comunicação, facilitando a integração de sistemas heterogêneos em uma smart grid.

No entanto, os autores também observam que a escolha do formato de serialização depende dos requisitos específicos da aplicação e do ambiente em que a smart grid está operando. Considerações como recursos de hardware disponíveis, interoperabilidade com sistemas legados e requisitos de segurança devem ser levadas em conta ao selecionar o formato mais adequado.

Em resumo, o artigo destaca que o formato binário é uma opção promissora para a serialização de dados em aplicações de smart grid, oferecendo vantagens significativas em termos de tamanho, desempenho e interoperabilidade. No entanto,

uma análise cuidadosa das necessidades e requisitos específicos é fundamental para tomar a decisão adequada de escolha do formato de serialização.

3.3 Sufficient Comparison Among Cloud Computing Services: IaaS, PaaS, and SaaS: A Review

Neste artigo é apresentado uma revisão abrangente e uma comparação detalhada dos serviços de computação em nuvem: IaaS (Infrastructure as a Service), PaaS (Platform as a Service) e SaaS (Software as a Service).

A computação em nuvem se tornou uma parte essencial das estratégias de TI para muitas organizações, permitindo o acesso a recursos computacionais flexíveis e escaláveis por meio da Internet. No entanto, escolher o modelo de serviço de nuvem adequado para atender às necessidades de uma empresa pode ser um desafio, devido às diferentes características e funcionalidades oferecidas por cada um desses serviços.

Os autores fornecem uma visão geral dos três modelos de serviço de nuvem. O IaaS oferece recursos de infraestrutura, como servidores virtuais, armazenamento e redes, permitindo que os usuários tenham controle total sobre o ambiente em nuvem e implantem suas próprias aplicações. O PaaS oferece uma plataforma completa para desenvolvimento e implantação de aplicativos, fornecendo ferramentas e serviços adicionais para facilitar o desenvolvimento. Já o SaaS oferece aplicativos prontos para uso, hospedados e disponíveis para os usuários acessarem pela Internet.

Os autores comparam esses modelos de serviço de nuvem com base em vários critérios, como escalabilidade, disponibilidade, segurança, personalização, custo e complexidade. Eles examinam as vantagens e desvantagens de cada modelo, bem como os cenários em que cada um se destaca.

No final do artigo, os autores destacam a importância de entender as necessidades e requisitos específicos de uma organização ao selecionar um modelo de serviço de nuvem. Eles enfatizam que não há um modelo de serviço de nuvem que seja adequado para todas as situações e recomendam que as empresas avaliem cuidadosamente suas necessidades antes de fazer uma escolha.

Em suma, o artigo fornece uma revisão completa dos modelos de serviço de nuvem IaaS, PaaS e SaaS, destacando suas características distintas e fornecendo

uma análise comparativa para ajudar as organizações a tomar decisões informadas ao adotar serviços de computação em nuvem.

Tabela 1 - Comparação ativa e direta de trabalhos correlatos.

Tópicos/Trabalhos	A Comparison of Data Serialization Formats For Optimal Efficiency on a Mobile Platform	Smart Grid Serialization Comparison	Sufficient Comparison Among Cloud Computing Services: IaaS, PaaS, and SaaS: A Review	Este trabalho
XML	**X**	**X**		**X**
JSON	**X**	**X**		**X**
Protobuf	**X**	**X**		
Protostuff		**X**		
Thrift	**X**			
Kryo		**X**		**X**
MsgPack				**X**
EC2			**X**	**X**
ECS			**X**	**X**
Lambda			**X**	**X**

Fonte: O Autor.

4 DESENVOLVIMENTO E ANÁLISE

4.1 TAMANHO DOS DADOS

No artigo de Kazuaki Maeda, é realizado um experimento utilizando 10 tamanhos diferentes de objetos a serem trafegados, sendo eles: 0, 100, 200, 300, 400, 500, 600, 700, 800, 900 (Kazuaki Maeda, 2012, p.180).

Focando em abordar diferentes tamanhos dos dados transmitidos, de maneira análoga ao experimento de Maeda, foram utilizados três tamanhos de vetor:

- Pequeno: vetor com tamanho de 1 elemento.
- Médio: vetor com tamanho de 100 elementos.
- Grande: vetor com tamanho de 1000 elementos.

Números maiores de elementos não foram considerados por limitações da infraestrutura, que falha com requisições de 10000 elementos ou mais em função do tempo de cada requisição ou quantidade de memória utilizada.

4.2 ARQUITETURA DA APLICAÇÃO

Como proposta, este trabalho considera desenvolver uma aplicação seguindo a arquitetura cliente x servidor, que permita o cliente realizar requisições utilizando o protocolo HTTP para o servidor, conforme apresentado na figura 4, comparando os formatos XML, JSON, MSGPACK e KRYO, em que o conteúdo dessas mensagens esteja inserido no contexto de um comércio eletrônico. O conteúdo das mensagens serão geradas aleatoriamente utilizando o site 4devs.

Objetivando abranger diversos tipos de dados, o conteúdo das mensagens serão:

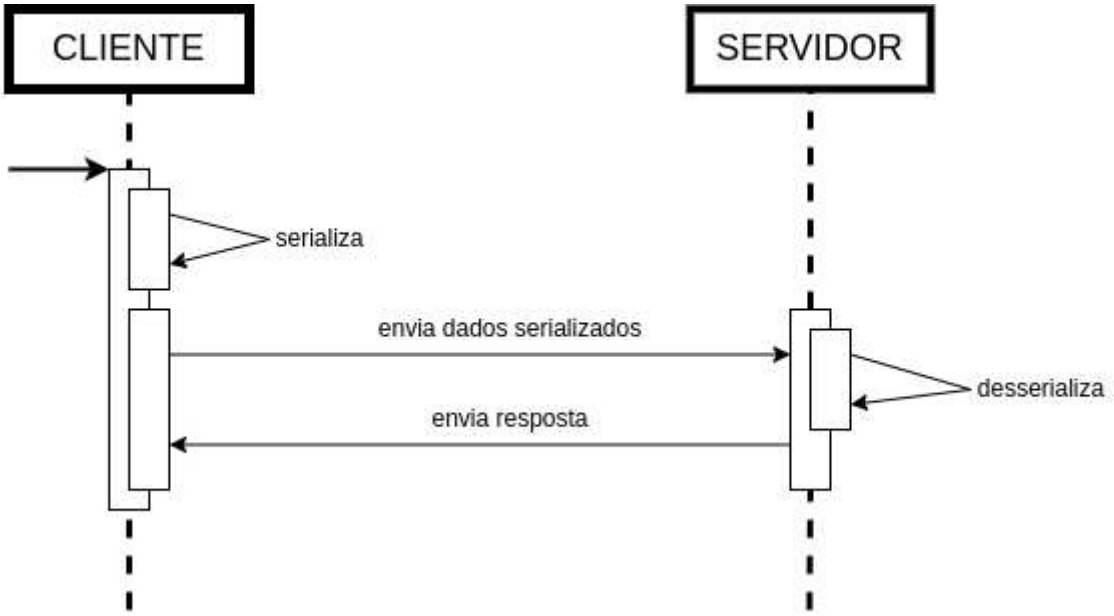
- Vetor de booleanos
- Vetor de caracteres
- Vetor de inteiros
- Vetor de doubles
- Vetor de strings
- Vetor de objetos

Além disso, a aplicação fará uso dos diferentes modelos de computação fornecidos pela AWS, como AWS EC2, AWS ECS e AWS Lambda Functions. O cliente será localizado na região de disponibilidade *us-east-1* (Norte da Virgínia, Estados Unidos), já o servidor, será localizado na região eu-central-1 (Frankfurt, Alemanha), conforme visto na figura 5.

No caso dos modelos AWS EC2 e AWS ECS, será utilizado a linguagem Java, juntamente com o framework Spring Boot, na qual possui como objetivo facilitar o desenvolvimento e integração entre cliente e servidor. Já para o modelo AWS Lambda, não será necessário nenhum framework, pois será utilizado o AWS API Gateway, na qual atua como a "porta da frente" para que os aplicativos acessem dados, lógica de negócios ou funcionalidade de seus serviços de back-end (AMAZON WEB SERVICES, 2023).

Desse modo, serão criadas combinações onde para cada modelo de computação, serão realizadas 100 iterações com cada tipo de dado, e com cada tamanho de vetor.

Figura 4: Diagrama de uma iteração na comunicação entre cliente e servidor.



Fonte: O Autor.

Os critérios de avaliação serão: Quantidade de bytes serializados, o Tempo de serialização, o Tempo de desserialização e o Tempo total da requisição, que serão armazenados no banco de dados AWS DynamoDB, para análise futura.

Inicialmente, para realizar a configuração do ambiente de nuvem, será necessário criar uma conta no console da AWS, criar um usuário no AWS IAM, para restringir o acesso aos serviços, seguindo as boas práticas de privilégio mínimo, e criar chaves para poder acessar os serviços via linha de comando. Os serviços que serão utilizados pelo usuário IAM serão: API Gateway, Elastic Compute Cloud,

Elastic Container Service, Lambda, DynamoDB e CloudFormation. Com auxílio do AWS CLI, é possível definir um perfil do usuário e então interagir com os serviços por meio da linha de comando.

Figura 5: Regiões da infraestrutura na Amazon Web Services.

Fonte: (AMAZON WEB SERVICES, 2023).

Visando a facilidade de reprodução dos experimentos e análise dos resultados, o código desenvolvido foi disponibilizado em: https://github.com/rafaelbcastilhos/thesis/tree/main, e o fluxo da execução está representado na figura 6. Para desenvolver a lógica de cliente e servidor, o autor sugere que seja utilizado o editor de texto IntelliJ IDEA, utilizando a linguagem Java 11. Inicialmente, será criado o módulo configuration, que terá cinco pacotes:

- ***database***: Possui a definição das configurações para conectar a aplicação no banco de dados.
- ***formatter***: Possui a implementação de serialização e desserialização de todos os formatos analisados.
- ***generator***: Possui a implementação da geração dos dados fictícios que devem ser trafegados para todos os tipos de dados analisados.

- ***model***: Possui as classes que formulam os objetos de um comércio eletrônico.
- ***utils***: Possui a implementação das configurações que devem ser trafegadas juntamente com a requisição: Definição de cabeçalhos, qual o tipo de dado utilizado, qual o tamanho do experimento e qual o serviço de computação será utilizado.

Tendo as configurações bem definidas, torna-se possível realizar o desenvolvimento dos clientes e servidores, que irão utilizar as funcionalidades descritas no módulo configuration, para isso serão criados os seguintes módulos:

- ***lambda-client***: Possui a implementação da função AWS Lambda que irá agir como cliente.
- ***lambda-server***: Possui a implementação da função AWS Lambda que irá agir como servidor.
- ***lambda-scripts***: Possui shell scripts com a finalidade de realizar a construção e implantação do lambda-client e lambda-server na AWS.
- ***spring-client-server***: Possui a implementação do cliente e servidor que serão utilizados no AWS EC2 e AWS ECS.
- ***local-client***: Possui a implementação que será responsável por interagir com os experimentos, na qual deve ser executada localmente.

Finalizado o desenvolvimento, iniciaremos a implantação das aplicações nos serviços da AWS. Inicialmente, é necessário realizar a construção dos módulos utilizando o Maven. Em primeiro lugar serão implantadas as funções Lambdas, posteriormente serão implantados os containers, e por fim serão implantadas as máquinas virtuais, conforme segue:

- **Funções Lambda**: É necessário executar o script *deploy.sh*, localizado dentro da pasta *lambda-scripts* e escolher qual região deve ser implantada. Não é necessário realizar nenhuma configuração, pois no arquivo *infra.yml*, já possui todas as definições de memória, segurança e limites das funções.
- **Containers**: Utilizando o Console da AWS, no serviço AWS ECS, é necessário criar um cluster em cada região, e definir o Limite de memória como 1024MiB, e mapear as portas 8080 e 80 com tcp. Com os clusters criados, é necessário associar o repositório docker disponível em: https://hub.docker.com/repository/docker/rafaelbcastilhos/ecs. Após isso, é necessário definir a VPC, que pode ser usada no padrão já existente na sua

conta na região, e também o grupo de segurança para definir as permissões de entrada e saída.

- **Máquinas virtuais**: Utilizando o Console da AWS, no serviço AWS EC2, é necessário criar uma instância em cada região, e definir o tipo da instância como t2.micro. Também é necessário definir a AMI (Imagem de máquina da Amazon) que possui as configurações necessárias para instanciar a máquina virtual, o autor sugere que seja utilizado Ubuntu em sua última versão LTS, ou Amazon Linux na última versão, além disso, é necessário permitir o tráfego de saída da instância cliente e o tráfego de entrada da instância de servidor nos Grupos de Segurança. Após isso, é necessário executar a aplicação de cliente e servidor utilizando o Spring Boot.

Ademais, será criada a tabela nomeada como “item” no banco de dados na região de us-east-1, que possui as seguintes colunas: i*d (Primary Key), bytesSerialize, method, service, size, timeDeserialize, timeRequest, timeSerialize* e *type*. Estas colunas não precisam ser criadas juntamente com a tabela, pois o DynamoDB lida de forma dinâmica com as colunas de um registro, criando e removendo conforme a regra de negócios. Por fim, a capacidade de leitura e escrita será definida como de acordo com a demanda, ou seja, irá escalar conforme o uso. Havendo essa capacidade estabelecida, será necessário criar uma réplica em eu-central-1, que habilite a replicação cruzada de tabelas do DynamoDB entre as duas regiões usando o recurso de Streams e Lambda triggers. Quando os dados são gravados ou atualizados em uma região, um gatilho Lambda pode ser acionado para replicar os dados para a tabela correspondente na outra região.

Após finalizar os experimentos, a réplica em eu-central-1 pode ser removida e a capacidade de leitura e escrita pode ser alterada para 1 unidade de leitura e escrita em modo provisionado. A pilha existente no CloudFormation pode ser deletada, pois isso fará com que as funções lambdas sejam deletadas também, os containers no ECS podem ser removidos, e as máquinas virtuais podem ser terminadas no EC2, evitando gerar custos inesperados.

Os principais objetivos dos experimentos são:

- Identificar o tempo de serialização e desserialização e compará-lo entre os diferentes tamanhos, serviços e formatos.
- Identificar a quantidade de bytes trafegados e compará-los entre os diferentes tamanhos, serviços e formatos.

- Identificar o tempo total de requisição/resposta e compará-lo entre os diferentes tamanhos, serviços e formatos.

Para os casos onde a conta na AWS estiver enquadrada com o período de free-tier, o ambiente necessário para a execução das simulações não resultará em quaisquer custos. Entretanto, em contas sem o nível de gratuidade, ou em casos onde o ambiente utiliza mais recursos computacionais do que os definidos neste trabalho, haverá custo.

Para desenvolver a lógica das análises, o autor sugere que seja utilizado o editor de texto PyCharm, utilizando a linguagem Python 3 e o SDK da AWS Boto3.

4.3 SERIALIZAÇÃO E DESSERIALIZAÇÃO

Durante o processo de avaliação, os períodos de serialização e desserialização foram registrados individualmente. No entanto, dada a interdependência dessas operações e com o objetivo de simplificar a apresentação dos resultados nesta seção, optou-se por somar os dois tempos para cada caso.

É importante ressaltar que os valores apresentados representam a média obtida ao longo de várias iterações para cada conjunto de parâmetros. Além disso, é relevante observar que a serialização ocorre no ambiente do serviço produtor, enquanto a desserialização ocorre no ambiente do serviço consumidor. Ambos os serviços foram configurados com o mesmo hardware, garantindo condições equivalentes. Os resultados para os três diferentes tamanhos de entrada estão disponíveis na tabelas 1, 2 e 3.

Tabela 2 - Tempo médio de serialização e desserialização em milissegundos entre tamanhos e formatos.

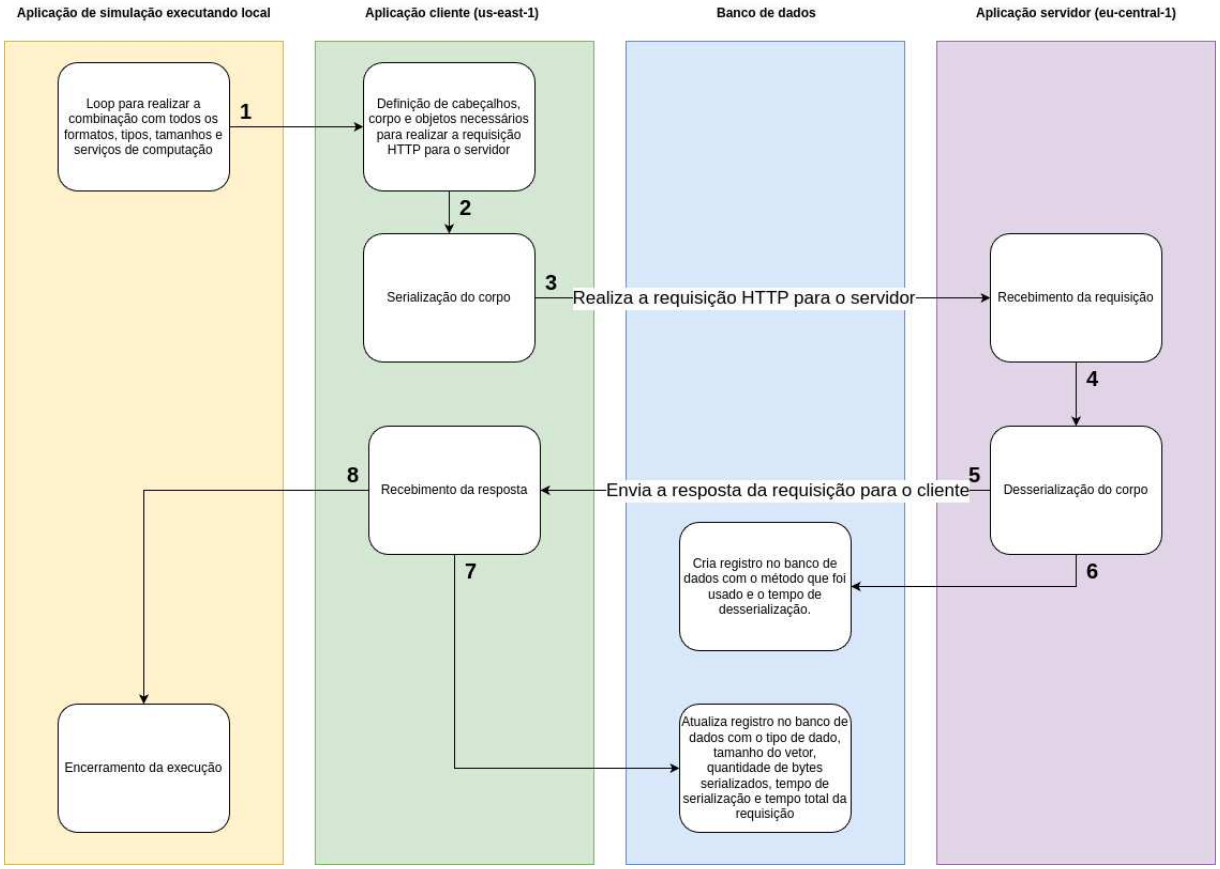
Tam./Formato	**XML**	**JSON**	**MSGPACK**	**KRYO**
Pequeno	4.47	1.14	1.36	139.16
Médio	4.80	1.96	3.21	139.43
Grande	19.01	16.55	25.69	142.02

Fonte: O Autor.

Ao relacionar o tamanho do conjunto de dados com o formato, nota-se ocusto da representação do formato KRYO em comparação aos outros formatos em todos

os tamanhos. O formato que obteve o melhor tempo médio foi o JSON em todos os tamanhos, conforme pode ser observado na figura 7.

Figura 6: Fluxograma de execução do algoritmo.



Fonte: O Autor.

Tabela 3 - Tempo médio de serialização e desserialização em milissegundos entre serviços e formatos.

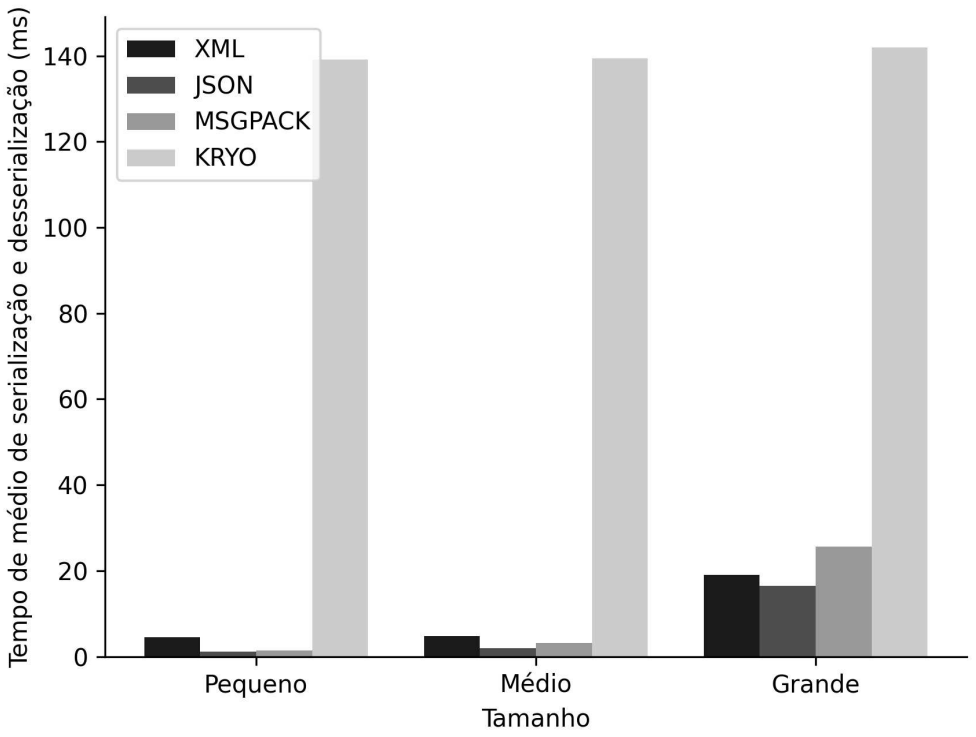
Serv./Formato	**XML**	**JSON**	**MSGPACK**	**KRYO**
EC2	6.70	4.88	7.84	105.79
ECS	9.15	8.06	15.06	203.62
LAMBDA	12.43	3.36	7.36	111.20

Fonte: O Autor.

Destes resultados, o tempo médio mais satisfatório de formatos textuais é o JSON, e de formatos binários é o MSGPACK. É notável o grande tempo levado para o KRYO, isso acontece devido ao processo de serialização, conforme pode ser visto

nas figuras 8 e 9. O tempo médio de serialização e desserialização nos serviços está diretamente relacionado ao tempo médio de serialização e desserialização dos diferentes tipos de dados, conforme apresentado na tabela 3.

Figura 7: Gráfico do tempo médio em milissegundos de serialização e desserialização em diferentes tamanhos de dados.



Fonte: O Autor.

Tabela 4 - Tempo médio de serialização e desserialização em milissegundos entre tipos de dados e formatos.

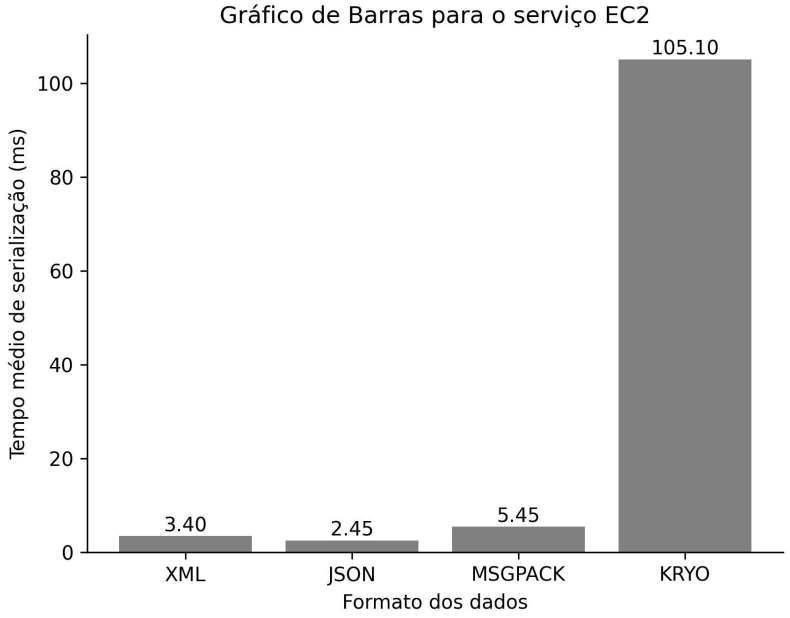
Tipo/Formato	**XML**	**JSON**	**MSGPACK**	**KRYO**
STRUCT	42.86	34.62	53.85	145.99
STRING	2.72	0.97	1.66	139.33
CHAR	2.42	0.78	1.99	139.99
DOUBLE	4.05	1.22	1.23	139.36

INTEGER	2.29	0.92	0.87	137.09
BOOLEAN	2.13	0.79	0.87	139.47

Fonte: O Autor.

Ao analisar sob a ótica de tipos de dados, percebe-se que o formato JSON leva uma pequena vantagem ao comparar com MSGPACK, que por sua vez leva uma pequena vantagem ao comparar com XML. De acordo com Petersen *et al*, em relação à memória, a maioria dos serializadores utiliza pouca memória e, portanto, não deve ser um problema, mas alguns deles usam muito menos memória do que outros, o que em certas situações os torna uma escolha melhor. A compressão de fato reduz o tamanho da mensagem, o que em alguns casos de uso torna válido, mas o preço pago em tempo de processamento não compensa para os serializadores mais eficientes na maioria dos casos.

Figura 8: Gráfico do tempo médio em milissegundos de serialização no serviço EC2.

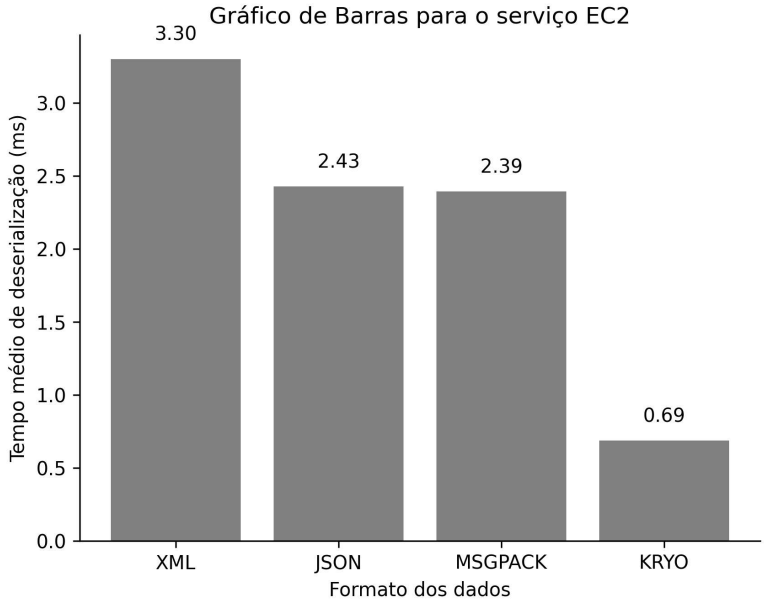


Fonte: O Autor.

Durante a execução dos experimentos, foi possível constatar que no processo de serialização, o KRYO pode priorizar a eficiência espacial em detrimento da velocidade de serialização. Além disso, o desempenho do KRYO também pode ser afetado pela complexidade dos objetos que estão sendo serializados (objetos

complexos com muitas estruturas aninhadas podem levar mais tempo para serem processados).

Figura 9: Gráfico do tempo médio de desserialização em milissegundos no serviço EC2.



Fonte: O Autor.

4.4 QUANTIDADE DE BYTES TRAFEGADOS

Nota-se que os valores, apesar de representarem bytes, possuem casas decimais por serem a média do tamanho resultante para todas as iterações, que estão apresentados nas tabelas 4, 5 e 6.

Tabela 5 - Quantidade de bytes serializados entre tamanhos e formatos.

Tam./Formato	**XML**	**JSON**	**MSGPACK**	**KRYO**
Pequeno	491.84	348.29	295.99	212.53
Médio	45843.09	33449.56	28671.69	1754.83
Grande	458030.72	334367.36	286609.03	15105.83

Fonte: O Autor.

Ao explorar os resultados, percebe-se uma enorme diferença entre os formatos textuais e binários. Sendo que o melhor resultado para todos os formatos é

o KRYO, tendo uma pequena vantagem para o MSGPACK, que por sua vez tem uma grande vantagem para os formatos textuais.

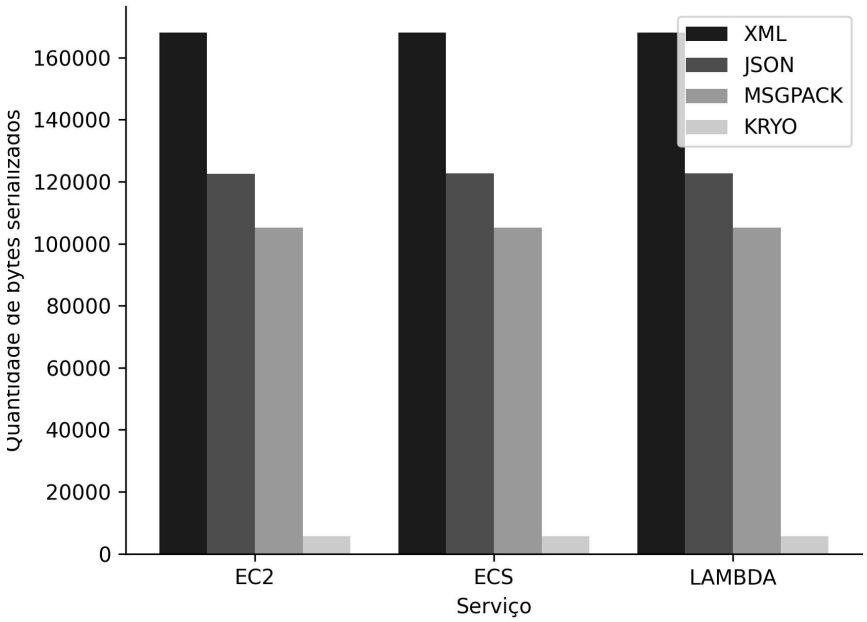
Tabela 6 - Quantidade de bytes serializados entre serviços e formatos.

Serv./Formato	**XML**	**JSON**	**MSGPACK**	**KRYO**
EC2	168133.19	122535.72	105191.36	5691.19
ECS	168111.49	122720.51	105187.53	5690.98
LAMBDA	168120.97	122705.33	105197.81	5691.03

Fonte: O Autor.

Ao avaliar os diferentes serviços, pode-se perceber uma média muito próxima entre os serviços, mas bem espaçosa entre os formatos. O resultado mais satisfatório foi o do KRYO, tendo uma grande diferença para o MSGPACK, que possui pequena diferença para JSON e XML. A disparidade entre formatos pode ser melhor comparada observando a figura 10.

Figura 10: Gráfico de quantidade de bytes serializados em diferentes serviços.



Fonte: O Autor.

Tabela 7 - Quantidade de bytes serializados entre tipos de dados e formatos.

Tipo/Formato	**XML**	**JSON**	**MSGPACK**	**KRYO**
STRUCT	955345.33	708604.45	608129.33	27076.33
STRING	23890.43	18714.73	18211.75	1100.72
CHAR	6640.00	1479.00	744.33	1125.33
DOUBLE	8254.14	2359.94	3313.33	3327.33
INTEGER	6640.00	745.00	377.33	758.33
BOOLEAN	7961.41	2066.44	377.33	758.33

Fonte: O Autor.

Considerando apenas os formatos binários, os valores são muito próximos. MSGPACK se destaca na representação de números inteiros, valores booleanos e caracteres, porém não performou muito bem para os outros tipos. Isso é consequência da otimização presente em todos os formatos para números inteiros, booleanos ou caracteres, que utilizam representações de tamanho variável. Já para outros tipos primitivos, os valores têm tamanho fixo, e o MSGPACK utiliza um byte adicional para informar o tipo do dado, impactando o resultado de STRUCT, já que este engloba outros tipos apresentados, justificando os resultados.

4.5 REQUISIÇÃO E RESPOSTA

Ao considerar o tempo total da requisição somado ao da resposta, é fundamental ter em mente as implicações decorrentes do meio de comunicação empregado: a latência e a taxa de transferência (TANENBAUM, *et al*., 2003). Enquanto a latência se refere a um intervalo fixo de tempo em cada ciclo, a taxa de transferência dimensiona o tempo de comunicação de acordo com a quantidade de dados transmitidos. Portanto, quanto maior a quantidade de dados, maior é o impacto da taxa de transferência.

Tabela 8 - Tempo médio de requisição e resposta em milissegundos entre tamanhos e formatos.

Tam./Formato	**XML**	**JSON**	**MSGPACK**	**KRYO**
Pequeno	339.57	345.06	327.13	328.49

Médio	446.27	398.59	392.13	338.32
Grande	539.26	522.97	497.88	410.13

Fonte: O Autor.

Com esses resultados, para os diferentes tamanhos é possível perceber que os formatos binários têm uma considerável vantagem em comparação aos formatos textuais. KRYO e MSGPACK possuem resultados semelhantes, da mesma maneira que XML e JSON.

Tabela 9- Tempo médio de requisição e resposta em milissegundos entre serviços e formatos.

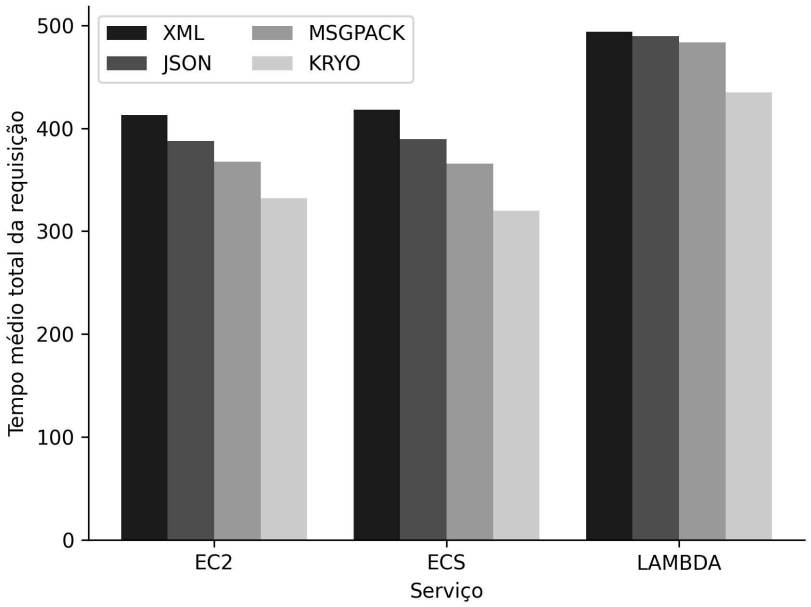
Serv./Formato	**XML**	**JSON**	**MSGPACK**	**KRYO**
EC2	412.94	387.60	367.62	332.14
ECS	418.02	389.40	365.95	319.76
LAMBDA	494.15	489.56	483.56	435.04

Fonte: O Autor.

Ao analisar o tempo médio de requisição e resposta nos diferentes serviços, percebe-se que existe similaridade entre os formatos. Nota-se que o AWS Lambda possui uma média mais elevada ao comparar com os demais serviços, isso ocorre devido ao tempo de inicialização do container, conhecido como *cold start*. De acordo com Manner *et al (2018)*, o início frio é um problema inerente às técnicas de virtualização quando as funções Lambda são executadas em contêineres. A primeira execução de uma função de nuvem enfrenta um início frio, já que o contêiner deve ser iniciado antes da execução. Por questões de desempenho e custo, os provedores de FaaS não desligam os contêineres imediatamente, apenas após um período variável de 10 a 15 minutos. Execuções subsequentes usam contêineres já criados para se beneficiar dos ambientes de execução "quentes", entretanto, execuções realizadas posteriormente ao período variável, irão enfrentar uma leve latência na inicialização. Evitar a ociosidade e dimensionar conforme a demanda são mudanças significativas em comparação com outros modelos de serviço em nuvem, mas implicam em mais inícios frios. O melhor resultado utilizando KRYO e

MSGPACK é utilizando ECS, já o melhor resultado utilizando XML e JSON é utilizando EC2, conforme pode ser visto na figura 11.

Figura 11: Gráfico de tempo médio total da requisição em milissegundos em diferentes serviços.



Fonte: O Autor.

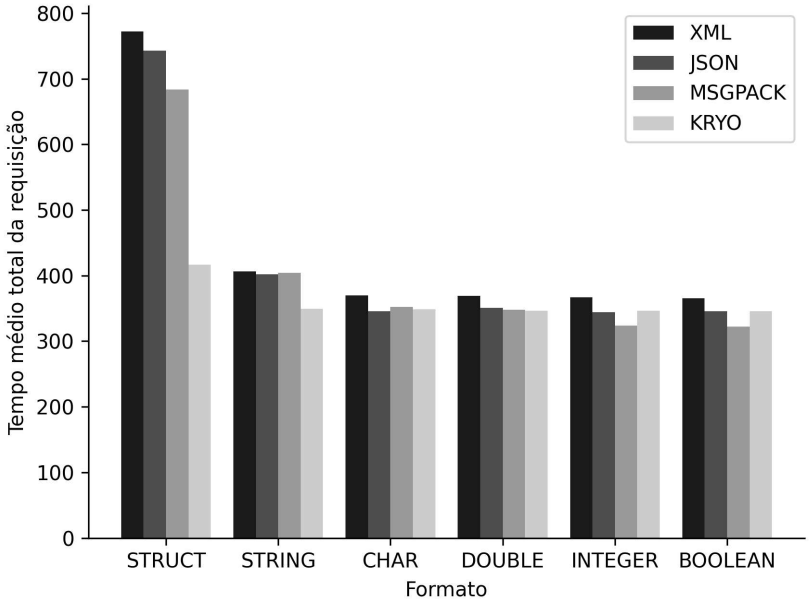
Tabela 10 - Tempo médio de requisição e resposta em milissegundos entre tipos de dados e formatos.

Tipo/Formato	**XML**	**JSON**	**MSGPACK**	**KRYO**
STRUCT	772.44	742.91	683.52	416.72
STRING	406.11	402.01	403.98	349.59
CHAR	370.20	345.96	352.61	348.72
DOUBLE	368.97	350.96	348.22	346.69
INTEGER	367.17	344.29	323.84	346.11
BOOLEAN	365.31	345.79	322.11	346.05

Fonte: O Autor.

Ao expor os tipos de dados, com exceção do tipo CHAR, que obteve melhores resultados com JSON, todos os demais tipos possuem vantagem utilizando formatos binários. Nos tipos STRUCT, STRING e DOUBLE o KRYO obteve o tempo médio mais satisfatório, já nos tipos INTEGER e BOOLEAN o MSGPACK foi o formato mais eficiente. Destaca-se que para todos os tipos, o formato XML foi o que obteve os piores resultados, conforme apresentado na figura 12.

Figura 12: Gráfico de tempo médio total da requisição em milissegundos em diferentes tipos.



Fonte: O Autor.

5 CONCLUSÃO E CONSIDERAÇÕES FINAIS

Neste estudo, foi examinada a eficácia de diferentes métodos de serialização junto às suas respectivas bibliotecas em Java durante a comunicação em ambientes de nuvem empregando diferentes serviços. Ao explorar os conceitos essenciais deste estudo, tornou-se evidente a importância da interoperabilidade na nuvem, o que influenciou a seleção dos métodos de serialização a serem analisados. Além disso, foram destacados os diversos fatores que impactam o desempenho ao variar o método de serialização, incluindo o tamanho dos dados serializados e o tempo necessário para realizar a serialização e desserialização.

Com base na abordagem metodológica dos trabalhos correlatos e na interação entre dois serviços na nuvem por meio de comunicação HTTP, foi elaborada a estrutura experimental para este estudo. O objetivo foi conectar as variáveis já investigadas em pesquisas anteriores com os elementos introduzidos por esta forma específica de comunicação, em especial, a relação entre o tamanho dos dados, o tempo de serialização, desserialização e o tempo de resposta das requisições. Dessa maneira, estabeleceu-se uma correlação entre o tempo de serialização/desserialização e o tempo total de comunicação. Considerando que a eficácia da comunicação é influenciada pela distância física entre os pontos, foi examinado um cenário com diferentes disposições geográficas, realizando a interação entre dois continentes. Após a construção da metodologia, os experimentos foram conduzidos e os resultados foram apresentados.

A compreensão dos fatores que influenciam o desempenho dos métodos de serialização, como o tamanho dos dados e o tempo de processamento, é fundamental para a tomada de decisões informadas na arquitetura e implementação de sistemas na nuvem. A capacidade de selecionar o método de serialização mais adequado para cada contexto específico pode resultar em ganhos significativos em termos de desempenho e eficiência operacional.

Na análise dos resultados obtidos neste estudo, observamos que o padrão geral refletiu o que foi observado nos trabalhos relacionados. No entanto, identificamos uma distinção significativa: a eficácia de um formato de comunicação sobre HTTP torna-se menos relevante quando os dados são pequenos o suficiente. Por outro lado, à medida que o tamanho dos dados aumenta, torna-se evidente a disparidade de desempenho entre os diferentes formatos. Além disso, foi estabelecido critérios para a seleção de um dos formatos mencionados em

ambientes práticos, levando em conta a localização geográfica dos serviços e as características específicas dos dados transmitidos, como tipo e volume. Ademais, pode-se concluir que a eficiência de um formato é crucial para os serviços na nuvem, especialmente em cenários com comunicação de baixa latência e para volumes de dados mais elevados.

Reafirmou-se que, em geral, formatos textuais são ineficientes quando contrapostos a formatos binários. Ainda assim, a escolha deste também pode depender de parâmetros qualitativos, como, por exemplo, o requisito de um formato legível num cenário onde os dados são usualmente acessados diretamente por humanos, ou a necessidade de lidar com dados dinâmicos e flexíveis, que não possuam um schema os definindo. Nos formatos textuais, XML deve ser evitado a menos que seja necessário, já que JSON é uma alternativa superior, uma vez que pode ser escrito e inspecionado em qualquer plataforma. Em suma, o uso de ECS com KRYO é identificado como a melhor opção, seguido pela utilização de EC2, representando os cenários mais favoráveis.

Portanto, este estudo não apenas contribui para o corpo de conhecimento existente sobre comunicação na nuvem, mas também fornece percepções valiosas para profissionais e pesquisadores que buscam otimizar a eficiência e a confiabilidade de sistemas distribuídos em ambientes de nuvem. A análise cuidadosa dos resultados e a aplicação prática das conclusões podem levar a melhorias tangíveis no desempenho e na escalabilidade de aplicações baseadas na nuvem, beneficiando tanto os provedores de serviços quanto os usuários finais.

5.1 TRABALHOS FUTUROS

Como continuação deste trabalho, é possível conduzir os mesmos experimentos empregando uma segunda linguagem de programação, a fim de mitigar eventuais tendências introduzidas pelo uso do Java e suas bibliotecas associadas.

Além disso, há diversos formatos não mencionados neste estudo, que se assemelham aos analisados e poderiam ser submetidos aos mesmos experimentos, como o Thrift, JSONB e entre outros mencionados nos resultados dos trabalhos correlatos.

Embora apenas uma infraestrutura de nuvem tenha sido avaliada, há concorrentes que oferecem serviços similares e poderiam ser considerados para uma avaliação semelhante. A transição entre esses provedores pode influenciar vários aspectos de desempenho, além de possivelmente apresentar diferenças de custo a serem avaliadas.

Enquanto este estudo ofereceu uma análise abrangente na nuvem, há espaço para uma investigação mais específica em cenários particulares, como o processamento de mensagens de Smart Grids. Nesse sentido, seria pertinente empregar a metodologia de experimentação mencionada, adaptando-a conforme necessário para se adequar aos conjuntos de dados específicos desse contexto. Uma abordagem focada em um cenário específico também pode requerer restrições diferentes em relação aos tipos de dados analisados, possivelmente excluindo alguns dos tipos testados anteriormente e introduzindo outros como alternativa.

Por fim, é necessário aperfeiçoar no código fonte os componentes de identificação de tipo, tamanho e método que é utilizado no tráfego de dados. No estado atual, foi usado um método fortemente acoplado para informar ao receptor o tipo de técnica que o remetente está aplicando, a fim de facilitar um processo de desserialização correto. O objetivo é relaxar esse processo para alcançar um nível genérico para que possa ser aplicado em diferentes cenários.

REFERÊNCIAS

M. Hogan and A. Sokol. NIST Cloud Computing Standards Roadmap Version 2. NIST Cloud Computing Standards Roadmap Working Group, NIST Special Publications 500-291, NIST, Gaithersburg, MD, 2013, p.1-113

Schroer, Christoph; KRUSE, Félix; Gómez, Jorge Marx. A Systematic Literature Review on Applying CRISP-DM Process Model. [S.l.: s.n.], 2021. v. 181, p. 526–534. DOI: 10.1016/j.procs.2021.01.199.

Abdullah, Khalid & Alzubaidi, Abdulaziz & Dauda, Muhammed & Al-Yahya, Mohammed & Al-Haddad, Mohammed. (2022). Distributed Systems: Concepts, Principles, Models and Algorithms. Journal of Early Modern Studies. 6. 256-269.

Sumaray, Audie & Makki, S.. (2012). A comparison of data serialization formats for optimal efficiency on a mobile platform. 10.1145/2184751.2184810.

Emeakaroha, Vincent & Healy, Philip & Fatema, Kaniz & Morrison, John. (2013). Analysis of Data Interchange Formats for Interoperable and Efficient Data Communication in Clouds. 10.1109/UCC.2013.79.

Jindal, Anshul & Podolskiy, Vladimir & Gerndt, Michael. (2019). Performance Modeling for Cloud Microservice Applications. 25-32. 10.1145/3297663.3310309.

Emeakaroha, M. Bullman & J. P. Morrison, Towards Automated Cost-Efficient Data Management for Federated Cloud Services. 2016 5th IEEE International Conference on Cloud Networking (Cloudnet), 2016, pp. 158-163, doi: 10.1109/CloudNet.2016.37.

Costello, K. Gartner Forecasts Worldwide Public Cloud Revenue to Grow 17.5 Percent in 2019. 2019. Disponível em: <https://www.gartner.com/en/newsroom/press-releases/2019-04-02-gartner-forecasts-worldwide-public-cloud-revenue-to-g>. Acessado em 2023-07-26.

Weingartner, R.; Brascher, G. B.; Westphall, C. B. Cloud resource management: A survey on forecasting and profiling models. Journal of Network and Computer Applications, Academic Press, v. 47, p. 99–106, 2015.

Geronimo, G. A. et al. Provisioning and resource allocation for green clouds. In: 12th International Conference on Networks (ICN). [S.l.: s.n.], 2013.

Mell, P.; Grance, T. et al. The nist definition of cloud Mell, P.; Grance, T. et al. The nist definition of cloud computing. 2011.computing. 2011.

Bray, T. STD, The JavaScript Object Notation (JSON) Data Interchange Format. [S.l.]: RFC Editor, 2017. <https://datatracker.ietf.org/doc/html/rfc8259>. Acessado em 2023-07-27.

Dragoni, N. et al. (2017). Microservices: Yesterday, Today, and Tomorrow. In: Mazzara, M., Meyer, B. (eds) Present and Ulterior Software Engineering. Springer, Cham. https://doi.org/10.1007/978-3-319-67425-4_12

Taibi, D., Lenarduzzi, V., & Pahl, C. (2017). Processes, Motivations, and Issues for Migrating to Microservices Architectures: An Empirical Investigation. IEEE Cloud Computing, 4(5), 22–32. doi:10.1109/mcc.2017.4250931

E. van Eyk et al., “A visão de pesquisa do grupo de nuvem SPEC sobre FaaS e arquiteturas sem servidor”, Anais do 2º Workshop Internacional sobre Computação Sem Servidor (WoSC 17), 2017, pp. 1–4.

Sweet, N. (2023), kryo. GitHub. https://github.com/EsotericSoftware/kryo. Acessado em 2023-07-29.

Amazon Web Services (2022). O que é XML? https://aws.amazon.com/pt/what-is/xml/. Acessado em 2023-07-29.

MICROSOFT. Serialization (C#). 2018. <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/serialization/>. Acessado em 2023-08-13.

MOZILLA. Serialization - Glossary. 2017. <https://developer.mozilla.org/en-US/docs/Glossary/Serialization>. Acessado em 2023-08-13.

Jennings, B.; Stadler, R. Resource Management in Clouds: Survey and Research Challenges. Journal of Network and Systems Management, v. 23, n. 3, p. 567–619, 2015. ISSN 10647570. <https://doi.org/10.1007/s10922-014-9307-7>.

Buyya, R. et al. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Generation Computer Systems, Elsevier B.V., v. 25, n. 6, p. 599–616, 2009. ISSN 0167739X. <http://dx.doi.org/10.1016/j.future.2008.12.001>

K. Maeda, "Performance evaluation of object serialization libraries in XML, JSON and binary formats" 2012 Second International Conference on Digital Information and Communication Technology and it's Applications (DICTAP), Bangkok, Thailand, 2012, pp. 177-182.,

Amazon Web Services (2023). API Gateway | API Management. Amazon Web Services, Inc. https://aws.amazon.com/api-gateway/. Acessado em 2023-08-19.

Belshe, M. Hypertext Transfer Protocol Version 2 (HTTP/2).[S.l.]: RFC Editor, 2017. <https://datatracker.ietf.org/doc/html/rfc7540>. Acessado em 2023-08-19.

MOZILLA. An overview of HTTP. 2017. <https://developer.mozilla.org/en-US/docs/Glossary/Serialization>. Acessado em 2023-08-13.

TANENBAUM, A. S. et al. Computer networks, 4-th edition. ed: Prentice Hall, ‘2003.

J. Manner, M. Endreß, T. Heckel and G. Wirtz, "Cold Start Influencing Factors in Function as a Service," 2018 IEEE/ACM International Conference on Utility and

Cloud Computing Companion (UCC Companion), Zurich, Switzerland, 2018, pp. 181-188.

Análise da estrutura e formatação de dados para comunicação eficiente entre ambientes de computação em nuvem

Rafael Begnini de Castilhos, Carlos Becker Westphall

Departamento de Informática e Estatística - Universidade Federal de Santa Catarina
88.040-900 - Florianópolis - SC - Brasil

{rafaelbcastilhos, carlosbwestphall}@gmail.com

Abstract: Cloud computing has seen exponential growth, particularly for commercial web applications. Applications increasingly frequently exchange information between services with the purpose of sharing, replicating, storing and other features characteristic of a distributed system. The choice of this format and protocol directly affects the size, latency and efficiency of the communication. Therefore, this work presents the development of an application that will act as a transmitter that will send data to a recipient. Subsequently, the data transmitted will be monitored, making it possible to extract, analyze and evaluate the results obtained in a comparative and exploratory manner.

Resumo: A computação em nuvem teve um crescimento exponencial, principalmente para aplicativos da web comerciais. Os aplicativos cada vez mais frequentemente realizam trocas de informações entre serviços com finalidade de compartilhar, replicar, armazenar e entre outras funcionalidades características de um sistema distribuído. A escolha deste formato e protocolo implica diretamente no tamanho, na latência e eficiência da comunicação. Com isso, este artigo apresenta o desenvolvimento de uma aplicação que atuará como emitente que enviará dados para um destinatário. Posteriormente, será efetuado monitoramento dos dados trafegados, sendo possível extrair métricas e avaliar os resultados obtidos de maneira comparativa e exploratória.

1. Introdução

A computação em nuvem é amplamente adotada na indústria como uma tecnologia que permite acesso fácil e barato ao processamento e armazenamento de dados. Durante as últimas décadas, muitas aplicações de *software* que até então operava de maneira *standalone,*

como por exemplo: sistemas de reserva, sistema bancário e comércio eletrônico e entre outras, passaram a ser distribuídas.

A comunicação entre os serviços ocorre através de mensagens que são geradas por um serviço emitente, trafegadas e recebidas por um serviço destinatário. Para isso, é necessário que o emitente serialize o dado que será transacionado, estruture de acordo com o protocolo que será utilizado e por fim que o destinatário deserialize. Entretanto, diferentes mecanismos podem ser aplicados para realizar a serialização e desserialização, além de que variados protocolos podem ser utilizados.

Este artigo propõe desenvolver uma aplicação responsável por emitir e receber informações utilizando diferentes abordagens, a fim de obter um aglomerado de dados para extrair métricas, analisar e comparar os variados mecanismos responsáveis. Com isso será possível identificar os métodos mais satisfatórios, objetivando aumentar a eficiência, reduzir os custos e a latência, e também identificar os que devem ser evitados.

2. Conceitos:

Nesta seção serão expostos alguns conceitos importantes relacionados ao tráfego de dados em aplicações de ambiente de computação em nuvem.

2.1. Computação em nuvem

A computação em nuvem é um modelo ubíquo, conveniente e de acesso compartilhado a recursos computacionais como servidores, armazenamento, aplicações e redes (MELL; GRANCE, 2011). Estes recursos compartilhados podem ser rapidamente provisionados e liberados com um esforço mínimo de gestão ou interação com o provedor de serviços (MELL; GRANCE, 2011).

Os objetivos dos provedores de nuvem estão centrados no uso eficiente dos recursos, dentro de limites estabelecidos por um Acordo de Nível de Serviço, ou Service Level Agreement (SLA), que é um contrato formal entre o provedor e o usuário, cujo objetivo é definir os aspectos funcionais e não funcionais do serviço em termos quantitativos (JENNINGS; STADLER, 2015). De acordo com a sua infraestrutura e o modelo de provisionamento de recursos, podemos classificar a nuvem em quatro principais categorias (BUYYA, *et al*., 2009):

- Nuvem pública: a nuvem abrange o provisionamento de recursos a terceiros alugando-os com base no uso.

- Nuvem privada: compreende as infraestruturas privadas mantidas e utilizadas por indivíduos e organizações.
- Nuvem híbrida: pode ser definida como o cenário em que uma organização estende a sua nuvem privada ao alugar parte dos seus recursos de uma nuvem pública.
- Nuvens comunitárias: nesta categoria, os recursos são contribuídos por vários indivíduos e/ou organizações de forma descentralizada.

Em (MELL; GRANCE, 2011) são definidas como características básicas de um ambiente de computação em nuvem:

1. Acesso aos recursos sob demanda: O usuário pode escolher quais serviços/componentes ele irá usar dentro do que é disponibilizado pela nuvem sem a necessidade de interação com administradores;

2. Disponibilidade: Ambientes de computação em nuvem devem estar sempre disponíveis para o usuário pela internet independente da plataforma de acesso (tablet, smartphone, desktop e outros);

3. Recursos compartilhados: Os recursos físicos da nuvem devem poder atender diferentes usuários simultaneamente;

4. Elasticidade: Conforme um usuário solicita mais recursos, a nuvem deve alocar mais recursos para este usuário, evitando a degradação dos serviços prestados, assim como se um usuário deixar de usar recursos, a nuvem pode remover recursos desnecessários;

5. Monitoramento de serviços: A nuvem deve ser capaz de monitorar o uso de recursos por cada serviço, proporcionando transparência aos fornecedores e usuários do serviço.

2.2. Microsserviços

O termo “microsserviço” tem sido amplamente utilizado desde 2012 para se referir a aplicativos desenvolvidos como um conjunto de aplicativos relativamente pequenos, consistentes, isolados e serviços autônomos implantados de forma independente, com um propósito único e claramente definido.

Os microsserviços são o oposto das arquiteturas monolíticas, onde os aplicativos geralmente são implantados como um único pacote em um contêiner da Web, como o Tomcat (Apache Tomcat, http://tomcat.apache.org/) ou JBoss (JBoss, http://www.jboss.org), e podem ser desenvolvidos independentemente por diferentes equipes de desenvolvimento. (TAIBI; LENARDUZZI; PAHL, 2017).

2.3. Formato de dados

A distinção entre diferentes formatos de dados é essencial para compreender como cada um se adapta a distintas necessidades de aplicativos e sistemas. Características

específicas moldam a utilidade e eficácia de cada formato em contextos variados. A presença ou ausência de uma especificação formal é um fator determinante. Formatos como XML e JSON possuem especificações que definem a estrutura e semântica dos dados, proporcionando consistência e interoperabilidade.

Outro aspecto crucial é a natureza binária ou textual do formato. Formatos binários, exemplificados por Protobuf, BSON, Avro, MessagePack, Kryo e entre outros, focam em eficiência e compactação, enquanto formatos textuais, como JSON e XML, oferecem legibilidade humana, embora possam ser mais extensos (SUMARAY; MAKKI. 2012).

2.4. Serialização e Desserialização:

Serializar é o processo de transformar objetos, ou estruturas de dados, em uma sequência de bytes, para que possam ser salvos em disco ou transportados pela rede (MICROSOFT, 2018). O processo inverso, a desserialização, por sua vez, envolve a conversão dessa sequência de bytes de volta para objetos ou estruturas de dados originais. A serialização e desserialização de objetos e dados são processos essenciais na programação que envolvem a conversão de estruturas de dados complexas, como objetos, em um formato que possa ser facilmente armazenado, transmitido e posteriormente recriado.

3. Trabalhos correlatos

Os autores Audie Sumaray e S. Kami Makki, investigam quatro formatos de serialização amplamente utilizados: XML, JSON, Protobuf e Thrift. Eles conduzem uma série de experimentos para avaliar o desempenho desses formatos em termos de tamanho do arquivo serializado, velocidade de serialização e desserialização, e consumo de recursos (CPU e memória) em dispositivos móveis.

Petersen et al, propõem uma comparação de diferentes formatos de serialização para aplicação em redes elétricas inteligentes (smart grids). É analisado e comparado formatos de serialização comumente utilizados em aplicações de smart grid: XML, JSON, Protobuf, Protostuff, Kryo e Avro. Eles exploram características como tamanho do arquivo serializado, velocidade de serialização e desserialização, capacidade de extensibilidade e interoperabilidade dos formatos.

Mohammed Chnar e Zeebaree Subhi fornecem uma visão geral dos três modelos de serviço de nuvem. O IaaS oferece recursos de infraestrutura, como servidores virtuais, armazenamento e redes, permitindo que os usuários tenham controle total sobre o ambiente em nuvem e implantem suas próprias aplicações. O PaaS oferece uma plataforma completa para desenvolvimento e implantação de aplicativos, fornecendo ferramentas e serviços

adicionais para facilitar o desenvolvimento. Já o SaaS oferece aplicativos prontos para uso, hospedados e disponíveis para os usuários acessarem pela Internet.

4. Proposta

Analisando e estudando os trabalhos relacionados, constatou-se a necessidade de comparar os formatos de serialização e desserialização no cenário de computação em nuvem, abrangendo diferentes modelos de computação e tipos de dados. Como apresentado anteriormente nos trabalhos correlatos, existem diferentes comparações entre formatos, porém nenhum deles leva em consideração os modelos de computação em nuvem.

Este artigo tem como proposta desenvolver uma aplicação seguindo a arquitetura cliente x servidor, que permita o cliente realizar requisições utilizando o protocolo HTTP para o servidor, conforme apresentado na figura 4, comparando os formatos XML, JSON, MSGPACK e KRYO, em que o conteúdo dessas mensagens esteja inserido no contexto de um comércio eletrônico. O conteúdo das mensagens serão geradas aleatoriamente utilizando o site 4devs. Objetivando abranger diversos tipos de dados, o conteúdo das mensagens serão vetores de booleanos, caracteres, inteiros, doubles, strings e objetos

Além disso, a aplicação fará uso dos diferentes modelos de computação fornecidos pela AWS, como AWS EC2, AWS ECS e AWS Lambda Functions. O cliente será localizado na região de disponibilidade *us-east-1* (Norte da Virgínia, Estados Unidos), já o servidor, será localizado na região eu-central-1 (Frankfurt, Alemanha).

No caso dos modelos AWS EC2 e AWS ECS, será utilizado a linguagem Java, juntamente com o framework Spring Boot, na qual possui como objetivo facilitar o desenvolvimento e integração entre cliente e servidor. Já para o modelo AWS Lambda, não será necessário nenhum framework, pois será utilizado o AWS API Gateway, na qual atua como a "porta da frente" para que os aplicativos acessem dados, lógica de negócios ou funcionalidade de seus serviços de back-end (AMAZON WEB SERVICES, 2023).

Desse modo, serão criadas combinações onde para cada modelo de computação, serão realizadas 100 iterações com cada tipo de dado, e com cada tamanho de vetor. Após realizar os experimentos, os principais objetivos da análise serão: Identificar o tempo de serialização e desserialização e compará-lo entre os diferentes tamanhos, serviços e formatos; Identificar a quantidade de bytes trafegados e compará-los entre os diferentes tamanhos, serviços e formatos; Identificar o tempo total de requisição/resposta e compará-lo entre os diferentes tamanhos, serviços e formatos.

Ao analisar o tempo médio de serialização e desserialização entre tamanhos e formatos, nota-se o custo da representação do formato KRYO em comparação aos outros formatos em todos os tamanhos. O tempo médio mais satisfatório de formatos textuais é o JSON, e de formatos binários é o MSGPACK. É notável o grande tempo levado para o KRYO, isso acontece devido ao processo de serialização, conforme pode ser visto nas figuras 1 e 2.

Ao avaliar os diferentes serviços sob a ótica da quantidade de bytes trafegados, pode-se perceber uma média muito próxima entre os serviços, mas bem espaçosa entre os formatos. O resultado mais satisfatório foi o do KRYO, tendo uma grande diferença para o MSGPACK, que possui pequena diferença para JSON e XML. A disparidade entre formatos pode ser melhor comparada observando a figura 3.

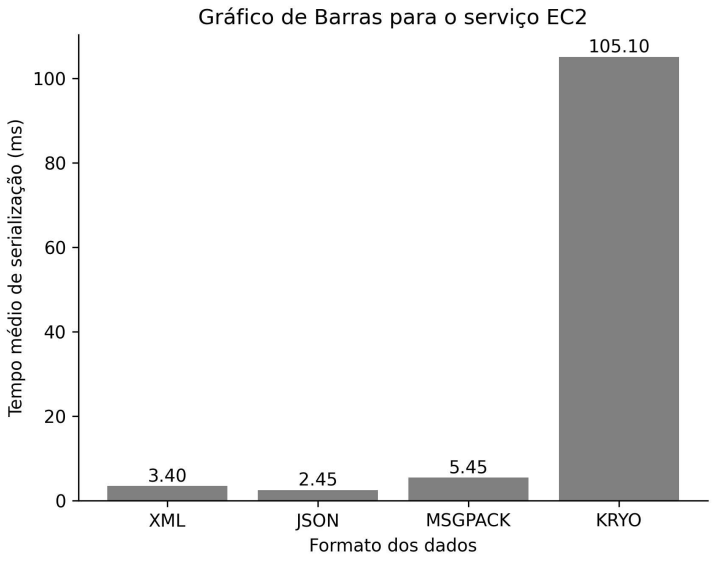


Figura 1. Gráfico do tempo médio em milissegundos de serialização no serviço EC2.

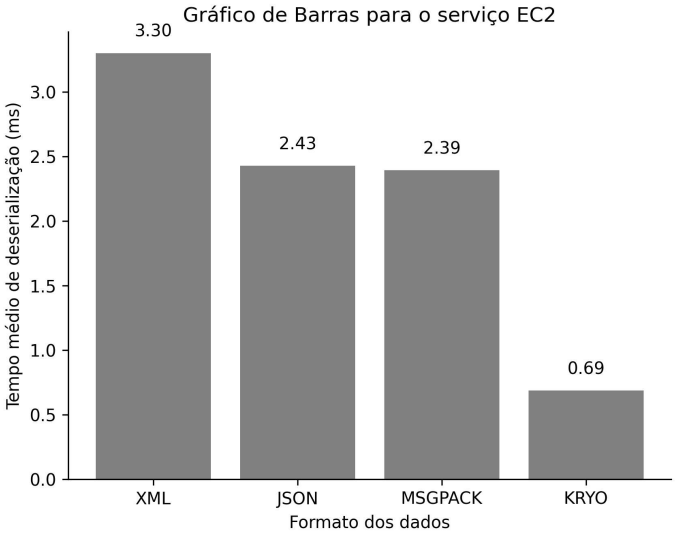


Figura 2. Gráfico do tempo médio de desserialização em milissegundos no serviço EC2.

Considerando-se apenas os formatos binários, os valores são muito próximos. MSGPACK se destaca na representação de números inteiros, valores booleanos e caracteres, porém não performou muito bem para os outros tipos. Isso é consequência da otimização presente em todos os formatos para números inteiros, booleanos ou caracteres, que utilizam representações de tamanho variável. Já para outros tipos primitivos, os valores têm tamanho fixo, e o MSGPACK utiliza um byte adicional para informar o tipo do dado, impactando o resultado de STRUCT, já que este engloba outros tipos apresentados, justificando os resultados apresentados na tabela 1.

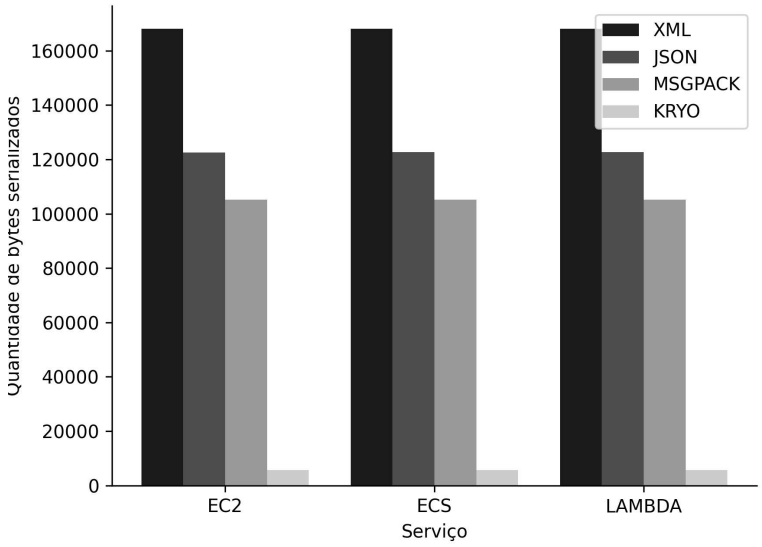


Figura 3. Gráfico de quantidade de bytes serializados em diferentes serviços.

Tabela 1 - Quantidade de bytes serializados entre tipos de dados e formatos.

Tipo/Formato	**XML**	**JSON**	**MSGPACK**	**KRYO**
STRUCT	955345.33	708604.45	608129.33	27076.33
STRING	23890.43	18714.73	18211.75	1100.72
CHAR	6640.00	1479.00	744.33	1125.33
DOUBLE	8254.14	2359.94	3313.33	3327.33
INTEGER	6640.00	745.00	377.33	758.33
BOOLEAN	7961.41	2066.44	377.33	758.33

Tabela 2 - Tempo médio de requisição e resposta em milissegundos entre serviços e formatos.

Serv./Formato	**XML**	**JSON**	**MSGPACK**	**KRYO**
EC2	412.94	387.60	367.62	332.14
ECS	418.02	389.40	365.95	319.76
LAMBDA	494.15	489.56	483.56	435.04

Ao analisar o tempo médio de requisição e resposta nos diferentes serviços, percebe-se que existe similaridade entre os formatos na tabela 2. Nota-se que o AWS Lambda possui uma média mais elevada ao comparar com os demais serviços, isso ocorre devido ao tempo de inicialização do container, conhecido como *cold start*. A primeira execução de uma função de nuvem enfrenta um início frio, já que o contêiner deve ser iniciado antes da execução. O melhor resultado utilizando KRYO e MSGPACK é utilizando ECS, já o melhor resultado utilizando XML e JSON é utilizando EC2, conforme pode ser visto na figura 4.

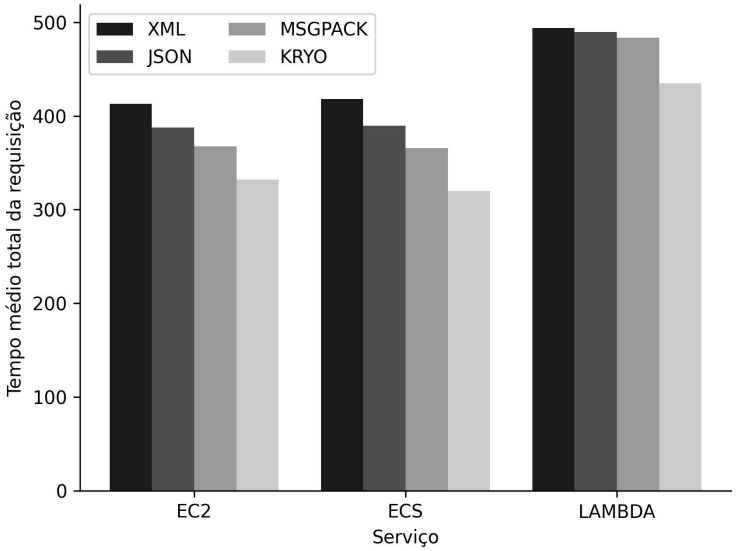


Figura 4: Gráfico de tempo médio em milissegundos em diferentes serviços.

5. Conclusões e considerações finais

Neste estudo, foi examinada a eficácia de diferentes métodos de serialização junto às suas respectivas bibliotecas em Java durante a comunicação em ambientes de nuvem empregando diferentes serviços. Ao explorar os conceitos essenciais, tornou-se evidente a importância da interoperabilidade na nuvem, o que influenciou a seleção dos métodos de serialização a serem analisados, utilizando como base a abordagem dos trabalhos correlatos,

foi estabelecida a relação entre o tamanho dos dados, o tempo de serialização, desserialização e o tempo de resposta das requisições.

A compreensão dos fatores que influenciam o desempenho dos métodos de serialização, como o tamanho dos dados e o tempo de processamento, é fundamental para a tomada de decisões informadas na arquitetura e implementação de sistemas na nuvem. A capacidade de selecionar o método de serialização mais adequado para cada contexto específico pode resultar em ganhos significativos em termos de desempenho e eficiência operacional.

Na análise dos resultados obtidos neste estudo, observamos que a eficácia de um formato de comunicação sobre HTTP torna-se menos relevante quando os dados são pequenos o suficiente. Por outro lado, à medida que o tamanho dos dados aumenta, torna-se evidente a disparidade de desempenho entre os diferentes formatos.

Reafirmou-se que, formatos textuais são ineficientes quando contrapostos a formatos binários. Ainda assim, a escolha deste também pode depender de parâmetros qualitativos, como, por exemplo, o requisito de um formato legível num cenário onde os dados são usualmente acessados diretamente por humanos, ou a necessidade de lidar com dados dinâmicos e flexíveis, que não possuam um schema os definindo. Nos formatos textuais, XML deve ser evitado a menos que seja necessário, já que JSON é uma alternativa superior, uma vez que pode ser escrito e inspecionado em qualquer plataforma. Em suma, o uso de ECS com KRYO é identificado como a melhor opção, seguido pela utilização de EC2, representando os cenários mais favoráveis.

Referências

Mell, P.; Grance, T. et al. The nist definition of cloud Mell, P.; Grance, T. et al. The nist definition of cloud computing. 2011.computing. 2011.

Buyya, R. et al. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Generation Computer Systems, Elsevier B.V., v. 25, n. 6, p. 599–616, 2009.

Jennings, B.; Stadler, R. Resource Management in Clouds: Survey and Research Challenges. Journal of Network and Systems Management, v. 23, n. 3, p. 567–619, 2015.

Taibi, D., Lenarduzzi, V., & Pahl, C. (2017). Processes, Motivations, and Issues for Migrating to Microservices Architectures: An Empirical Investigation. IEEE Cloud Computing, 4(5), 22–32.

Sumaray, Audie & Makki, S.. (2012). A comparison of data serialization formats for optimal efficiency on a mobile platform.

MICROSOFT. Serialization (C#). 2018. <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/serialization/>. Acessado em 2023-08-13.

Amazon Web Services (2023). API Gateway | API Management. Amazon Web Services, Inc. https://aws.amazon.com/api-gateway/. Acessado em 2023-08-19