Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

FaaS em Nuvem Privada para Otimização do Consumo de Recursos Computacionais em Operações de Armazenamento

Marcelo Augusto da Cruz Motta

Dissertação apresentada como requisito parcial para conclusão do
Mestrado Profissional em Computação Aplicada

Orientadora
Prof.a Dr.a Aletéia Patrícia Favacho de Araújo Von Paumgartten

Brasília
2024

Ficha catalográfica elaborada automaticamente,
com os dados fornecidos pelo(a) autor(a)

da Cruz Motta, Marcelo Augusto
dM314f FaaS em Nuvem Privada para Otimização do Consumo de Recursos Computacionais em Operações de Armazenamento / Marcelo Augusto da Cruz Motta; orientador Aletéia Patrícia Favacho de Araújo Von Paumgartten. -- Brasília, 2024.
83 p.

Dissertação(Mestrado Profissional em Computação Aplicada) -- Universidade de Brasília, 2024.

1. FaaS. 2. Função como serviço. 3. Nuvem Privada. 4. Dataprev. 5. OpenWhisk. I. de Araújo Von Paumgartten, Aletéia Patrícia Favacho , orient. II. Título.

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

FaaS em Nuvem Privada para Otimização do Consumo de Recursos Computacionais em Operações de Armazenamento

Marcelo Augusto da Cruz Motta

Dissertação apresentada como requisito parcial para conclusão do
Mestrado Profissional em Computação Aplicada

Prof.a Dr.a Aletéia Patrícia Favacho de Araújo Von Paumgartten (Orientadora)
CIC/UnB

Prof.a Dr.a Maria Clicia de Castro
UERJ

Prof. Dr.a Maristela Terto de Holanda
CIC/UnB

Prof. Dr. Gladston Luiz da Silva
Coordenador do Programa de Pós-graduação em Computação Aplicada

Brasília, 21 de junho de 2024

Dedicatória

Dedico este trabalho a minha esposa Priscilla e ao nosso filho Eduardo, que mesmo diante da minha ausência em inúmeras ocasiões e mesmo diante de todas as dificuldades da vida, sempre me motivaram e permaneceram ao meu lado, acreditando sempre em todos nossos objetivos. Nossa família sempre foi a maior motivação para tudo que me tornei e conquistei nessa vida.

Agradecimentos

Primeiramente, gostaria de agradecer a Deus pela minha família, pela minha saúde e por todas as oportunidades e ensinamentos que tive em toda minha vida, agradeço meu pai Carlos e minha mãe Patrícia, por formarem os meus valores e minha índole que me permitiram chegar até aqui.

Agradeço também, de coração, a Universidade de Brasília - UNB, por me proporcionar a oportunidade de evoluir pessoalmente e academicamente, a todos os colegas e professores do PPCA do qual tive a oportunidade de trocar experiências e adquirir conhecimento. Agradeço ainda o professor Dr Marcos Fagundes Caetano e a professora Dra Maria Clicia de Castro, por terem me direcionado e contribuído com este trabalho como membros da banca de qualificação.

Agradeço a excepcional família DATAPREV, empresa pública do qual me orgulho muito de fazer parte, e que me permitiu construir uma carreira profissional que nunca imaginei alcançar.

E por último, com muito carinho, gostaria de expressar a minha profunda e eterna gratidão pela querida orientadora e amiga, professora Dr.a Aletéia Araújo, pela parceria e paciência, por todo o aprendizado e pelo privilégio das lições que levarei para o resto da minha vida. Muito obrigado !!

Resumo

Desde o seu surgimento em meados de 2014, o modelo de serviço em nuvem *Function-as-a-Service* (FaaS) tem experimentado um notável crescimento em sua adoção pelos usuários. Este modelo oferece uma série de benefícios significativos, incluindo redução de custos, escalabilidade automática e desenvolvimento ágil baseado em microsserviços. Diante da gama de modelos de serviço em nuvem disponíveis, este trabalho propõe uma abordagem para a adoção do FaaS, focando na avaliação, seleção e provisionamento de plataformas FaaS de código aberto. Assim, esta proposta visa a implementação do FaaS no ambiente de nuvem privada de uma grande empresa pública. O objetivo central é realizar a implantação do modelo de serviço FaaS em uma nuvem privada real, permitindo comparações significativas em operações de infraestrutura relacionadas ao armazenamento e manipulação de grande volume de dados. Mais especificamente, o estudo concentra-se nos casos de uso relacionados à interação com o armazenamento de objetos, como o Dell ECS S3, da empresa em questão. Os experimentos iniciais demonstraram resultados promissores, especialmente em relação à disponibilidade do recurso de elasticidade e à economia de recursos computacionais. Este trabalho representa um passo significativo na direção da otimização e eficiência das operações de armazenamento de documentos em grande escala, dentro de um contexto corporativo de nuvem privada.

Palavras-chave: FaaS, Função como serviço, Nuvem Privada, Dataprev, OpenWhisk.

Abstract

Since its emergence in mid-2014, the Function-as-a-service (FaaS) model has experienced notable growth in its user adoption. This model offers some significant benefits, including cost savings, automatic scalability, and agile development based on microservices. Given the wide range of available cloud service models, this work proposes an approach to FaaS adoption, focusing on assessing, selecting, and providing open-source FaaS platforms. Therefore, this proposal aims to implement FaaS in the private cloud environment of a large public company. The central objective is to deploy the FaaS service model in a real private cloud, allowing meaningful comparisons in storage-related infrastructure operations and handling large volumes of data. Specifically, the study focuses on use cases related to interacting with object storage, such as Dell ECS S3, from the company in question. Initial experiments demonstrated promising results, especially concerning the availability of the elasticity resource and the saving of computational resources. This work represents a significant step in the direction of optimization and efficiency of document storage operations on a large scale within a corporate private cloud context.

Keywords: FaaS, Function-as-a-Service, Private Cloud, Dataprev, OpenWhisk.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

API Application programming interface.

AWS Amazon web services.

CNIS Cadastro Nacional de Informações Sociais.

DATAPREV Empresa de Tecnologia e Informações da Previdência.

FaaS Function-as-a-service.

IaaS Infrastructure as a Service.

K8S Kubernetes Cluster.

MGI Ministério da Gestão e Inovação.

NIST National Institute of Standards and Technology.

RHEL RedHat Enterprise Linux.

SDK Software development kit.

SGD Secretaria de governo digital.

Capítulo 1

Introdução

Atualmente, há vários tipos de serviços de nuvem sendo ofertados pelos diversos provedores. Dentre essas ofertas, destaca-se o *Function as a Service* (FaaS) [3], também conhecido como *Serverless*, no qual o cliente submete um trecho de código ao provedor, que irá garantir seu processamento, independentemente da escala.

Em 2014, a Amazon web services (AWS) lançou o AWS Lambda [5], sendo pioneira no lançamento do tipo de serviço FaaS [6]. Desde então, diversos outros provedores de nuvem pública têm ofertado em seus catálogos serviços orientados a esse novo modelo, por exemplo o *Google Cloud Plattform* oferece o serviço de *Google Cloud Function* [7], a Microsoft tem o *Azure Functions* [8], e outros provedores como Alibaba, Oracle e IBM também possuem suas opções de FaaS.

Diante das novas possibilidades para esse modelo, o interesse sobre ele cresceu notadamente e, considerando sua representatividade, existe a perspectiva de que em pouco tempo ele se torne a forma de adoção de nuvem predominante dentre os modelos disponíveis atualmente [6]. Isso ocorre porque o FaaS simplifica a execução de software na nuvem, suprimindo a necessidade de provisionamento, configuração e gestão sobre a infraestrutura necessária para o processamento de softwares escritos nas principais linguagens de programação. Além disso, com o FaaS ainda é possível racionalizar o investimento em recursos computacionais [6].

Nesse modelo de serviço a cobrança dos provedores é baseada no efetivo tempo de processamento que o software utiliza, e não no tempo de operação como ocorre em serviços de *Infrastructure as a Service* - IaaS tradicionais, nos quais a infraestrutura precisa permanecer alocada continuamente. Essas características, em conjunto com o comportamento auto escalável presente nesse modelo, tem feito com que diversos projetos migrem suas cargas de trabalho para ambientes orientados a FaaS.

Diante desse cenário, diversos trabalhos [2], [9], [10], têm dedicado esforços no sentido de avaliar o desempenho desse tipo de serviço, seja com relação à utilização de recursos,

consumo energético, tempo de processamento, dentre outras métricas.

Entretanto, ainda existe na literatura um interesse maior no uso de plataformas de nuvem pública. Todavia, há situações nas quais o uso de nuvem privada é o mais indicado, ou até mesmo a única solução possível.

O governo federal brasileiro, por meio do Decreto 10.332 de 28/04/2020, instituiu a estratégia de governo digital para o período de 2020 a 2022 com a missão de oferecer políticas públicas e serviços de melhor qualidade, mais simples, acessíveis a qualquer hora e lugar, e a um custo menor para o cidadão. A Secretaria de governo digital (SGD), do Ministério da Gestão e Inovação (MGI), coordena os planos de transformação digital dos órgãos e entidades da administração pública federal. Nesse cenário, destaca-se a Empresa de Tecnologia e Informações da Previdência - DATAPREV, que além de ser mantenedora do Cadastro Nacional de Informações Sociais (CNIS), possui a missão institucional de fornecer soluções de tecnologia da informação e da comunicação para a execução e o aprimoramento das políticas sociais do Estado brasileiro. Com sede em Brasília, a empresa possui três *datacenters*, localizados no Distrito Federal, Rio de Janeiro e em São Paulo, e cinco Unidades de Desenvolvimento distribuídas nos estados Ceará-CE, Paraíba-PB, Rio Grande do Norte-RN, Rio de Janeiro-RJ e Santa Catarina-SC.

Dispondo de uma infraestrutura robusta, distribuída entre seus três *datacenters*, atualmente, a empresa hospeda inúmeros sistemas de vários clientes (órgaos) públicos, em diferentes casos. De um modo geral, ambientes com missão crítica para atendimento de políticas sociais para a população Brasileira. Diante desse cenário, e diante do seu considerável tamanho, de 2019 para os dias atuais, a empresa tem investindo fortemente no modelo de serviço em nuvem, tanto para atendimento interno, com provisionamentos de ambientes automatizados nos modelos IaaS e PaaS (nuvem privada), quanto para o atendimento externo com diversos outros modelos de serviço.

Nesse sentido, acredita-se que o uso de serviço de FaaS em um ambiente de nuvem privada, visando operações de infraestrutura relacionadas a operações de armazenamento de documentos em grande escala utilizando um *storage* de objetos, proporcione uma melhor relação custo benefício em relação ao uso das plataformas tradicionais.

Diante desse cenário, o presente trabalho de pesquisa avaliou diversas ferramentas de FaaS atualmente utilizadas no mercado, procurando analisar e qualificar a performance e confiabilidade de cada uma. Os experimentos foram realizados utilizando a infraestrutura de nuvem privada da DATAPREV, viabilizando a produção de trabalhos que originaram três publicações importantes, que são produtos desse trabalho relacionados ao modelo de serviço FaaS e armazenamento:

- Comparison of FaaS Platform Performance in Private Clouds - 12th International Conference on Cloud Computing and Services Science CLOSER 2022 [11].

- Análise de Desempenho de Funções como Serviço em Nuvem Privada - DATAPREV - 2021 ANAIS DA IV ESCOLA REGIONAL DE ALTO DESEMPENHO DO CENTRO-OESTE [12].
- Performance Analysis of Main Public Cloud Big Data Services Processing Brazilian Government Data. CARLA 2020. Communications in Computer and Information Science, vol 1327. Springer, Cham. [13].

1.1 Objetivos

O objetivo geral deste trabalho é implementar o serviço do tipo FaaS na nuvem privada da DATAPREV, a fim de investigar os benefícios deste serviço em relação ao uso de máquinas virtuais (plataformas tradicionais de alto desempenho). A proposta é identificar os benefícios que o modelo de serviço FaaS proporcionam, nos desafios operacionais relacionados a manipulação de grande volume de dados, bem como ao consumo de infraestrutura tecnológica desta empresa.

Para cumprir o objetivo geral deste trabalho, faz-se necessário atingir os seguintes objetivos específicos:

- Identificar uma ferramenta de FaaS de melhor performance para o ambiente de nuvem privada;
- Selecionar uma operação de infraestrutura relacionada a operações de armazenamento de documentos em grande escala, adequada para ser executada por meio do serviço do tipo FaaS;
- Avaliar os benefícios e limitações do uso de FaaS na aplicação relacionada a operações de armazenamento de documentos em grande escala.

1.2 Descrição deste Documento

Em adição a este capítulo de introdução, o presente trabalho possui mais seis capítulos. No Capítulo 2 são apresentados os principais conceitos da computação em nuvem, além de uma visão geral sobre a evolução dessa plataforma em ambientes de alto desempenho. No Capítulo 3 são abordados os conceitos do modelo FaaS, com uma descrição das principais plataformas estudadas. O Capítulo 4 apresenta uma análise de desempenho das plataformas FaaS de código aberto, bem como a metodologia utilizada no estudo, e os resultados obtidos. No Capítulo 5 é apresentada a implantação da plataforma FaaS selecionada em conjunto com a implantação do *framework* Orama [14], com foco na análise

comparativa em operações de infraestrutura, relacionadas a manipulação e gravação de documentos em grande escala. Em seguida, no Capítulo 6 são apresentados os resultados com o comparativo das cargas de trabalho realizadas na plataforma FaaS. Por último, o Capítulo 7 traz as conclusões desta dissertação, e destaca alguns trabalhos futuros para continuidade desta pesquisa.

Capítulo 2

Computação em Nuvem

Neste capítulo são apresentados os principais conceitos de computação em nuvem, bem como seus modelos de provisionamento e de serviços. Este capítulo está estruturado em cinco seções. A Seção 2.1 apresenta as características da computação em nuvem. Na Seção 2.2 são apresentados os modelos de provisionamento. Na Seção 2.3 são abordados os modelos de serviços. Na Seção 2.4, a evolução dos modelos de serviço é abordada, considerando o comportamento dos provedores em relação aos serviços ofertados. Por último, a Seção 2.5 descreve especificamente as características e os desafios de um ambiente de nuvem privada, considerando o caso da empresa DATAPREV.

2.1 Características da Computação em Nuvem

Atualmente, o conceito de computação em nuvem é uma realidade bem estabelecida, sendo possível o acesso sob demanda a vários recursos computacionais configurados [15], usando modelos de oferta como serviço para infraestrutura, plataforma e software. De acordo com a definição de NIST (National Institute of Standards and Technology) em 2011 [1], a computação em nuvem é um modelo que visa fornecer acesso de rede onipresente, conveniente e sob demanda a um conjunto de recursos de computação configuráveis, tais como redes, servidores, armazenamento, aplicativos e serviços. Ela deve usufruir da possibilidade de ser rapidamente provisionada e liberada, com o mínimo esforço de gerenciamento ou interação com o provedor de serviços.

Assim sendo, para o usuário a capacidade dos ambientes de nuvem deve parecer ser ilimitada, podendo a demanda dos usuários ser atendida em qualquer quantidade a qualquer momento. A Figura 2.1 apresenta, de acordo com NIST [1], as principais características essenciais, os modelos de serviço e os modelos de implantação. Logo, o modelo de serviço em nuvem deve ser composto, considerando-se cinco características essenciais, as quais são [1]:

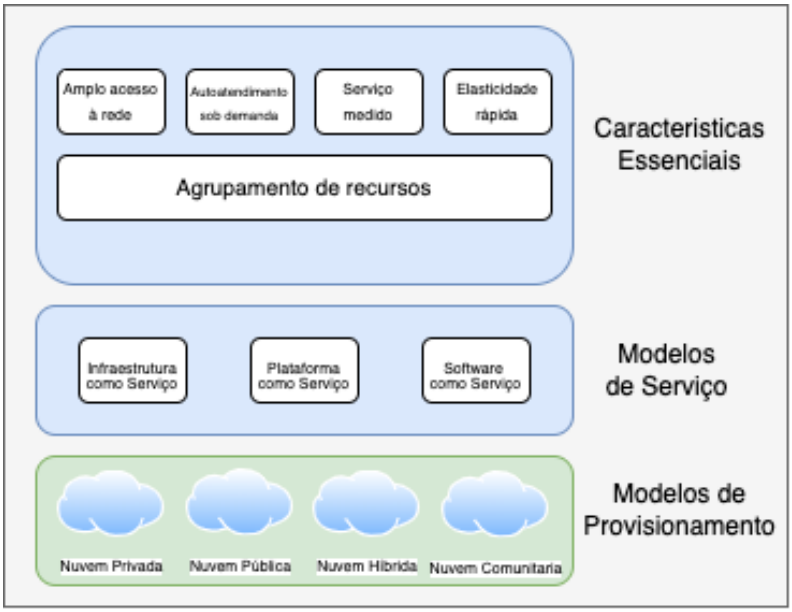


Figura 2.1: Características, modelos de provisionamento e serviços, adaptado de [1].

- **Autoatendimento sob demanda:** no qual um usuário pode unilateralmente provisionar e desprovisionar recursos de computação sob demanda, como servidores, armazenamento e rede, conforme sua necessidade;
- **Amplo acesso à rede:** os recursos estão disponíveis na rede e são acessados por meio de mecanismos que promovem sua utilização (por exemplo, telefones celulares, *tablets, laptops* e estações de trabalho);
- **Agrupamento de recursos:** os recursos de computação do provedor são agrupados para atender a vários consumidores usando um modelo *multi-tenant*, com diferentes recursos físicos e virtuais, dinamicamente atribuídos e reatribuídos de acordo com a demanda do consumidor. O cliente, geralmente, não tem controle ou conhecimento sobre a localização dos recursos fornecidos, mas pode especificar a localização em um nível mais alto de abstração (por exemplo, país, estado ou *datacenter*);
- **Serviço medido:** os sistemas em nuvem controlam e otimizam automaticamente o uso de recursos computacionais, fornecendo aos provedores e usuários os recursos apropriados para a medição e cobrança de cada tipo de serviço fornecido (por exemplo, armazenamento, processamento, largura de banda, contas de usuário ativas e etc...);
- **Elasticidade rápida:** a elasticidade é a capacidade de um sistema de sustentar cargas de trabalho crescentes com desempenho adequado. Nesse caso, os recursos computacionais podem ser incrementados ou reduzidos, elasticamente, em alguns

casos automaticamente, possibilitando dessa forma ao consumidor aumentar ou diminuir rapidamente a infraestrutura, de acordo com a necessidade da sua demanda.

A elasticidade é uma característica fundamental dos serviços de nuvem que permite que eles se adaptem de forma dinâmica à demanda, garantindo disponibilidade e desempenho do serviço. Existem dois tipos de elasticidade no serviço de nuvem, as duas são horizontal e vertical [16]. A elasticidade horizontal se refere à capacidade do serviço de adicionar ou remover instâncias de recursos, tais como máquinas virtuais, containers ou servidores, para lidar com picos de demanda ou quedas na utilização. Isso significa que o serviço pode aumentar ou diminuir a quantidade de recursos disponíveis para garantir a qualidade e a disponibilidade da oferta. Já a elasticidade vertical se refere à capacidade do serviço de adicionar ou remover recursos dentro de uma única instância de um recurso, como uma máquina virtual. Isso permite que o serviço ajuste a capacidade de processamento, de armazenamento ou de memória de forma mais granular e eficiente, sem a necessidade de adicionar ou remover instâncias inteiras.

A elasticidade automática é uma característica dos serviços de nuvem que permite que o serviço se adapte automaticamente à demanda, sem a necessidade de intervenção humana [16]. Isso pode melhorar a disponibilidade e o desempenho do serviço, reduzindo o tempo de resposta e os custos operacionais. Além disso, a elasticidade pode ser implementada de maneira manual ou automática [16].

Na elasticidade manual, o ajuste da capacidade do serviço é feito manualmente por um administrador ou equipe responsável pelo serviço [16]. Essa abordagem requer que alguém monitore constantemente a utilização de recursos do serviço e tome ações para aumentar ou diminuir a capacidade do serviço de acordo com a demanda. O ajuste pode ser feito por meio de ferramentas de gerenciamento de nuvem ou diretamente no painel de controle do provedor de nuvem.

Por outro lado, na elasticidade automática, o ajuste da capacidade do serviço é feito automaticamente pelo próprio serviço de nuvem [16]. Isso significa que o serviço monitora constantemente a utilização de recursos e ajusta a capacidade do serviço de forma autônoma, sem a necessidade de intervenção humana. As regras da elasticidade são definidas previamente e podem ser personalizadas de acordo com as necessidades do serviço.

A Figura 2.2 mostra o comportamento da demanda por serviço sob um ambiente durante determinado tempo, sendo possível constatar o alto custo de implementação da elasticidade horizontal, mesmo sendo possível o não atendimento efetivo da demanda (perda de clientes). Entretanto, verifica-se que, considerando a elasticidade, a curva acompanha a demanda tanto nos picos altos como baixos, deixando evidente a vantagem econômica e operacional dessa característica.

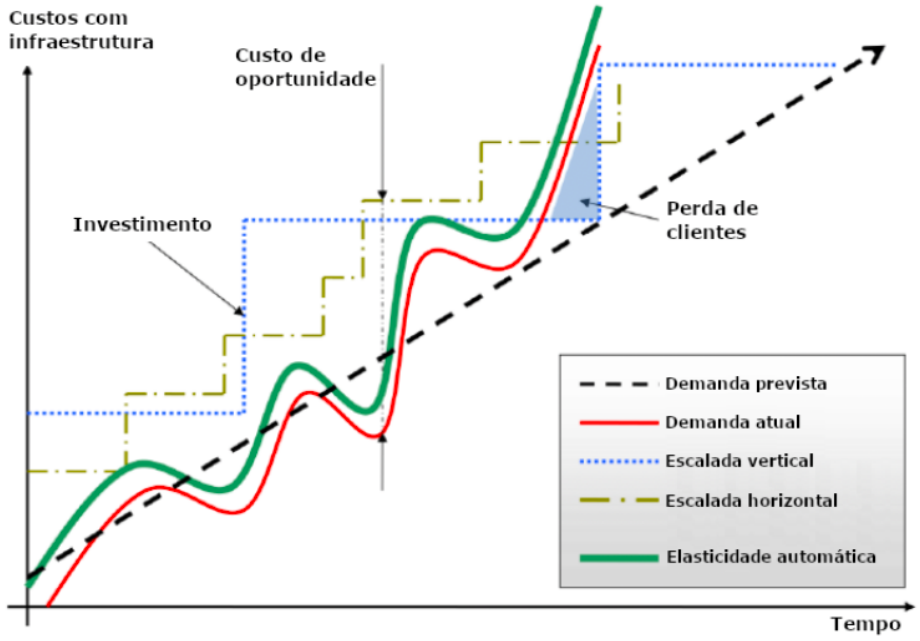


Figura 2.2: Escalabilidade versus elasticidade, adaptado de [2].

2.2 Modelos de Provisionamento

Além das principais características de nuvem, o National Institute of Standards and Technology (NIST) [1] também definiu as formas de implantação dos ambientes de nuvem, conforme apresentados na Figura 2.1, e descritos a seguir:

- **Nuvem Pública:** a infraestrutura de nuvem é provisionada para uso aberto pelo público em geral. Pode ser de propriedade, gerenciada e operada por uma organização empresarial, acadêmica ou governamental, ou alguma combinação deles. Ela existe nas instalações dos *datacenters* do provedor de nuvem, que normalmente são geodistribuídas;
- **Nuvem Privada:** nesse modelo de implantação a infraestrutura de nuvem é provisionada para uso exclusivo de uma única organização, compreendendo vários consumidores (por exemplo, diferentes unidades de negócios). Pode ser de propriedade, gerida e operada pela própria organização, um terceiro, ou alguma combinação deles, podendo existir dentro ou fora das instalações;

  A nuvem privada é bastante semelhante à nuvem pública, pois oferece os mesmos serviços. Entretanto, esses serviços são oferecidos por meio de uma infraestrutura proprietária e controlada, na qual os usuários, que também são os próprios proprietários, podem controlar o nível de segurança e privacidade.

No geral, o modelo de provisionamento de nuvem privada oferece uma infraestrutura segura, escalável, personalizável e confiável, adequada para grandes empresas de TI, com cargas de trabalho complexas e exigentes. Aproveitando as nuvens privadas, as empresas podem otimizar o desempenho, reduzir custos, e garantir a conformidade com os regulamentos do setor;

- **Nuvem Comunitária:** neste modelo de provisionamento, a infraestrutura de nuvem é provisionada para uso exclusivo por um comunidade de consumidores de organizações que compartilham objetivos (por exemplo, missão, requisitos de segurança, política e considerações de conformidade). Pode ser de propriedade, gerenciada e operada por uma ou mais organizações da comunidade, ou por terceiros, ou alguma combinação deles, podendo também existir dentro ou fora das instalações [1];
- **Nuvem Híbrida:** a infraestrutura computacional é uma composição de duas ou mais infraestruturas de nuvens distintas (privadas, comunitárias ou públicas), que permanecem como entidades únicas [1]. A consolidação das nuvens distintas nos permite abordar os conceitos de nuvens federadas.

As nuvens federadas podem ser definidas como um conjunto de nuvem pública, privada e comunitária, conectados por meio da Internet, as quais também são chamadas de *inter-cloud* ou *cross-cloud* [17]. Dentre seus objetivos, destaca-se a ilusão de recursos ilimitados, a eliminação da exclusividade no uso de um provedor de infraestrutura específico e a otimização quanto ao uso dos recursos dos provedores federados.

Elas são integradas por tecnologia padronizada ou proprietária que permite o contingenciamento dos serviços fornecidos entre elas (por exemplo, alto consumo de recursos com a carga sendo balanceada entre as nuvens). Entretanto, em ambientes *multicloud*, é preciso ter uma ferramenta que gerencie os papéis de cada tipo de recursos dentro do ambiente, bem como a origem (qual provedor) e suas respectivas integrações [17]. A adoção dessas ferramentas visa abstrair a complexidade inerente à integração dos provedores de nuvem.

Dada a importância do modelo de nuvem privada para este trabalho, o mesmo está melhor detalhado na Seção 2.4. Analisando a Figura 2.3 é possível verificar os três principais modelos de implantação.

2.3 Modelos de Serviço

A computação em nuvem tem proporcionado uma forma diferente de tratar os recursos computacionais. Isso ocorre porque as nuvens entregam, por serem orientadas a serviços,

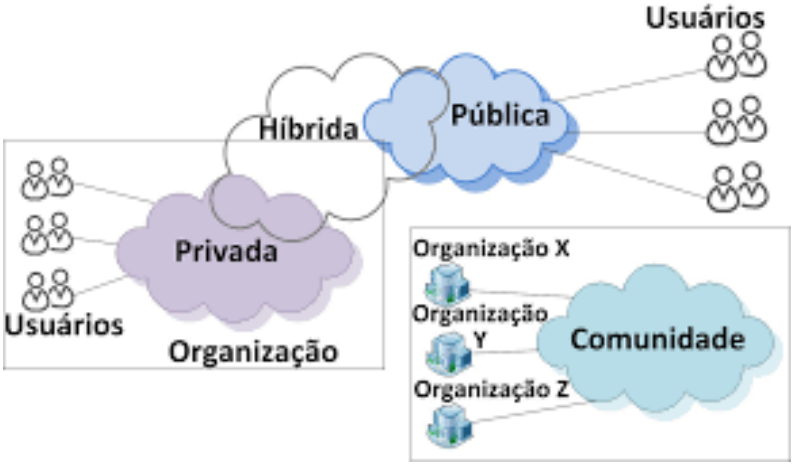


Figura 2.3: Modelos de provisionamento de nuvem.

diferentes modelos de utilização da capacidade computacional, abstraindo complexidades de acordo com a necessidade do cliente.

Assim, os usuários decidem qual o nível de envolvimento que desejam ter com o recurso computacional que estão utilizando, seja uma máquina virtual, com acesso privilegiado às configurações; ou um serviço de *e-mail*, cuja complexidade operacional é totalmente encapsulada, oferecendo aos usuários apenas o *front-end* do serviço. Assim sendo, de acordo com o NIST [1], a infraestrutura em nuvem fornece três modelos de serviço:

- **Infraestrutura como Serviço (*Infrastructure as a Service - IaaS*)**

  Este modelo de serviço se baseia na capacidade do provedor em fornecer ao consumidor processamento, armazenamento, redes e outros recursos computacionais fundamentais, permitindo-o implantar e executar software arbitrário, incluindo sistemas e aplicativos.

  Nesse caso, o consumidor não gerencia ou controla totalmente a infraestrutura, mas tem controle sobre sistemas operacionais, armazenamento, aplicativos implantados e possivelmente controle limitado sobre componentes de rede (por exemplo, *firewalls*).

- **Plataforma como Serviço (*Plattaform as a Service - PaaS*)**

  Este modelo de serviço se baseia no fornecimento de recursos por parte do provedor, ao consumidor para se implantar aplicativos de infraestrutura criados pelo consumidor, ou adquiridos usando linguagens de programação, bibliotecas, serviços e ferramentas suportadas pelo provedor.

  Nesse caso, o consumidor não pode gerenciar ou controlar a infraestrutura de nuvem subjacente, incluindo rede, servidores, sistemas operacionais ou armazenamento.

Entretanto, o consumidor tem controle sobre os aplicativos implantados e, possivelmente, a definições de configuração para o ambiente de hospedagem de aplicativos.

- **Software como Serviço (*Software as a Service - SaaS*)**

  Este modelo de serviço se baseia na capacidade fornecida ao consumidor para utilização dos aplicativos do provedor executados em sua infraestrutura de nuvem. Nesse caso, as aplicações são acessíveis a partir de vários dispositivos clientes por meio de uma interface, como um navegador *Web* ou um aplicativo. O consumidor não gerencia ou controla a infraestrutura de nuvem subjacente, incluindo rede, servidores, sistemas operacionais, armazenamento ou até mesmo recursos de aplicativos individuais, com a possível exceção de configurações limitadas de aplicativos específicos do usuário.

Contudo, desde a definição dos modelos de serviço, os provedores têm então usado a criatividade que a indústria exige para contemplar todos os tipos de siglas que definam seus modelos de serviço em nuvem. Atualmente, é possível encontrar siglas como DaaS para definir serviços de banco de dados (*Database-as-a-Service*) [18], ou definir serviços de entrega de *desktop* (*Desktop-as-a-Service*) [19], ou ainda especificando serviços de entrega de dados (*Data-as-a-Service*) [20]. Além disso, é possível encontrar MaaS para definir monitoramento como serviço (*Monitoring-as-a-Service*) [21], e STaaS definindo armazenamento como serviço (*STorage-as-a-service*) [22].

Assim sendo, o que todas essas siglas têm em comum é o fato de representarem alguma atividade computacional oferecida no formato de serviço (*as-a-Service*). Por conta disso, a sigla XaaS (*Everything-as-a-service*) [23] foi criada para abranger qualquer definição de serviço em nuvem que apresente alguma especificidade em relação aos modelos tradicionais definidos pelo NIST (IaaS , PaaS e SaaS) [1]. Dentre as diversas formas de oferecer serviços de computação em nuvem, o *Function-as-a-Service* (FaaS) tem obtido grande destaque no mercado [3].

No modelo FaaS, o cliente envia um trecho de código ao provedor, cujo processamento será garantido pelo provedor, independente da quantidade de requisições que receber [6]. A complexidade relacionada ao gerenciamento da infraestrutura para permitir isso é encapsulada pelo provedor, incluindo a elasticidade automática. Assim, dada a importância deste modelo de serviço para este trabalho, o Capítulo 3 apresenta a descrição deste modelo de serviço, descrevendo suas vantagens e limitações. Além disso, é apresentado o cenário de sua implementação e a utilização em um ambiente de nuvem privada.

2.4 Nuvem Privada - DATAPREV

A DATAPREV é uma empresa pública que fornece soluções de Tecnologia da Informação e Comunicação (TIC) para o aprimoramento, e a execução de políticas sociais do Estado brasileiro. Com sede em Brasília, ela possui cinco unidades nos seguintes estados: Ceará, Paraíba, Rio Grande do Norte, Rio de Janeiro e Santa Catarina.

A empresa conta com três *datacenters* com certificação internacional TIER III [24]. As infraestruturas são projetadas para promover a alta disponibilidade e a segurança dos sistemas, e estão localizadas no Distrito Federal, Rio de Janeiro e em São Paulo. Com 48 anos de experiência na gestão e no desenvolvimento de soluções de TIC, a DATAPREV dispõe de capacidade computacional e logística para hospedar, manter, gerir e proteger informações e sistemas.

Além disso, a DATAPREV também tem a capacidade de analisar e qualificar dados, antecipar demandas de parceiros, prestar serviços de consultoria, apoiar a elaboração e a realização de projetos. Atualmente, essa empresa opera ofertando vários tipos de serviços de TI aos órgãos do governo brasileiro, possuindo estrutura computacional robusta que a permite operacionalizar os modelos de serviços em nuvem, tanto pública quanto privada, para hospedagem de seus sistemas internos. De acordo com o foco deste trabalho, a infraestrutura de nuvem privada da DATAPREV foi utilizada para provisionamento de infraestrutura automatizada em ambientes de desenvolvimento, homologação e produção. Assim, a nuvem privada usada neste trabalho é apresentada na Figura 2.4.

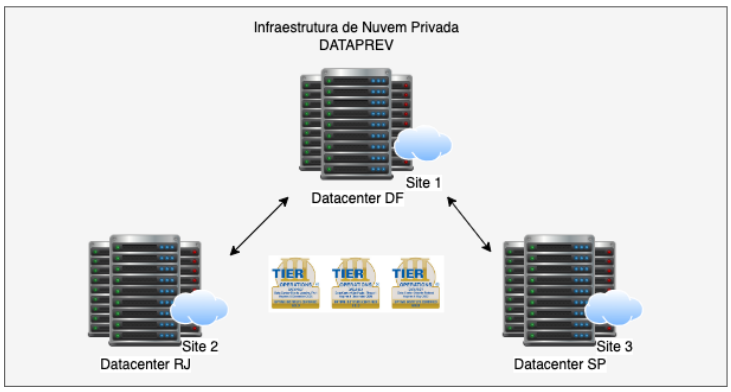


Figura 2.4: *Datacenters* DATAPREV - Arquitetura nuvem privada.

Diante da criticidade dos serviços atualmente hospedados nessa infraestrutura de nuvem privada, o presente trabalho objetiva explorar o uso do modelo de serviço do tipo FaaS em aplicações de infraestrutura relacionadas a manipulação de grande volume de dados, a fim de obter uma melhor relação custo x benefício de aplicações reais da DATAPREV. Para isso, o próximo capítulo apresenta o modelo FaaS em detalhes, destacando suas vantagens e limitações.

Capítulo 3

Modelo de Serviço FaaS

Neste capítulo é apresentado o modelo de serviço FaaS. Para isso, a Seção 3.1 apresenta as definições do modelo de serviço, bem como as suas características e tendências. Na Seção 3.2 são apresentadas as principais plataformas de FaaS do mercado, considerando seu nível de maturidade e a utilização por provedores públicos. Na Seção 3.3 são apresentados os trabalhos relacionados, com suas principais características e abordagens.

3.1 Definição

O modelo de serviço *Function-as-a-Service* (FaaS) também é conhecido por *Serverless* [25], ou *Backend-as-a-Service*, (BaaS) [26] quando usado especificamente para atuar como APIs. Neste modelo de serviço os provedores fornecem interfaces (gráfico ou linha de comando) para que os usuários permitam a entrada e a manipulação de código fonte escrito em qualquer uma das linguagens de programação suportadas, supervisionando o processamento de aplicativos a partir de gatilhos pré-configurados. Esses acionadores, também chamados de gatilhos (*triggers*), podem se originar dos serviços do provedor ou por meio de uma API [6].

Neste modelo, mesmo considerando a flutuação da carga exercida sobre o serviço, espera-se que o provedor garanta a resposta aos acionamentos, eventualmente disponibilizando, de maneira transparente, mais recursos computacionais para contemplar o efetivo processamento de forma ágil, mesmo em situações de sobrecarga. Da mesma forma, em situações de baixa carga, o provedor deve desalocar recursos adicionais para racionalizar o custo operacional da plataforma de forma transparente para seus usuários [2]. Esse efeito, conhecido por elasticidade no contexto de computação em nuvem é, "o grau em que um sistema é capaz de se adaptar às mudanças de carga de trabalho, provisionando e desprovisionando recursos de maneira autônoma, de modo que, em cada ponto no tempo, os recursos disponíveis correspondam à demanda atual o mais próximo possível"[16], con-

forme apresentado na Seção 2.1. E essa é uma grande vantagem do FaaS porque os provedores disponibilizam a elasticidade automática neste tipo de serviço.

Outra importante característica do FaaS está relacionada ao aspecto que preocupa os administradores de sistemas, que é a ocorrência de falhas nos ambientes que gerenciam. O desafio de manter o ecossistema preparado para lidar com esses eventos a fim de restabelecer o serviço da forma mais rápida e transparente possível, pode ser muito exigente e é conhecido como resiliência. A resiliência ajuda a aumentar a taxa de disponibilidade e pode ser fundamental para a integridade de uma solução computacional [3]. Com o modelo de serviço FaaS, a resiliência é garantida, de maneira transparente, pelo provedor de serviço. A Figura 3.1 apresenta o *workflow* do *Functions-as-a-Service*, demonstrando como ocorre a interação entre o desenvolvedor e o serviço de FaaS, carregando código, consultando as Application programming interface (API)s dos Software development kit (SDK)s do provedor, ou configurando gatilhos para invocação do serviço, bem como a elasticidade do serviço e seu modelo de pagamento sob demanda (*pay-as-you-go*).

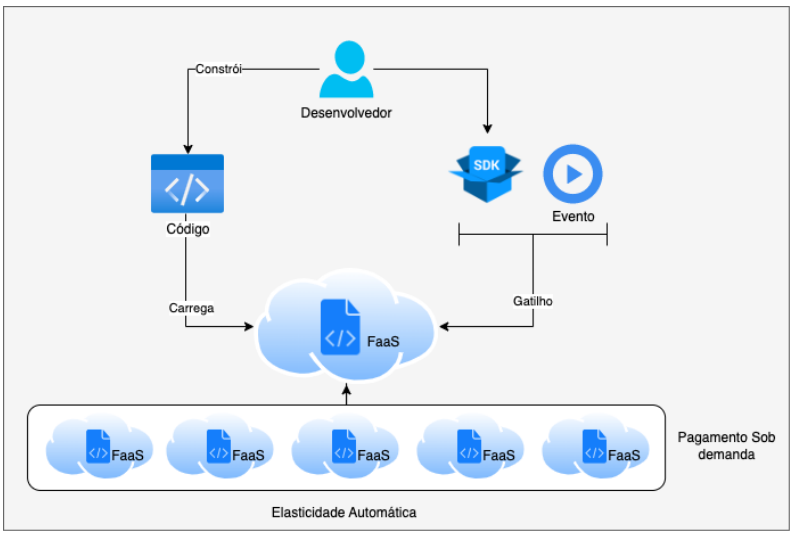


Figura 3.1: Fluxo de trabalho do FaaS, adaptado de [3].

A cobrança neste modelo de serviço é realizada, geralmente, por meio de uma política baseada no número de recursos que a função alocou e no seu tempo de execução. Essa tarifação não gera cobrança por inatividade, como ocorre nos modelos tradicionais de serviços em nuvem. Além disso, dada a sua característica elástica, o FaaS permite a aplicação de sobrecargas operacionais sem a necessidade de intervenção do usuário.

Nesse cenário, o FaaS permite que uma série de casos de uso prospere, principalmente, nos serviços de Internet, os quais estão frequentemente sujeitos a grandes variações de demanda. Assim, é fundamental contar com uma infraestrutura elástica que se adapte rapidamente às novas necessidades. Como resultado, o interesse por esse serviço em nuvem tem aumentado bastante nos últimos anos [27]. Todavia, em cenários onde há necessidade

de manter o estado, é necessário utilizar outros recursos do provedor para armazenamento, pois com FaaS o estado não é mantido pelo servidor.

Manter o estado em um ambiente de microsserviços é crucial para garantir a consistência e a confiabilidade das operações. Isso permite que cada microsserviço mantenha informações relevantes sobre o contexto da aplicação e do usuário, facilitando a comunicação entre os serviços e garantindo uma experiência contínua para o usuário final. Além disso, o estado persistente pode ser essencial para garantir a integridade dos dados e suportar operações que exigem coesão e coordenação entre diferentes serviços.

Diante do exposto, vários provedores têm ofertado serviços de FaaS em seus catálogos, existindo atualmente várias ferramentas proprietárias e de código aberto para este propósito. Assim sendo, na próxima seção serão descritas as principais ferramentas ofertadas pelos provedores públicos e privados para o modelo FaaS.

3.2 Plataformas de FaaS

Atualmente, os maiores provedores de nuvem pública já oferecem opções de serviço orientadas ao modelo FaaS. Contudo, cada provedor tem sua própria estratégia para sustentar operacionalmente esses serviços baseados em FaaS. Em sua maioria adotam soluções de gestão FaaS proprietárias, cujo conhecimento de como opera é limitado aos usuários e a comunidade acadêmica quando estudado. No entanto, há provedores que optam por usar plataformas de código aberto. A Figura 3.2 apresenta as principais tecnologias relacionadas ao modelo de serviço FaaS estudadas no presente trabalho.

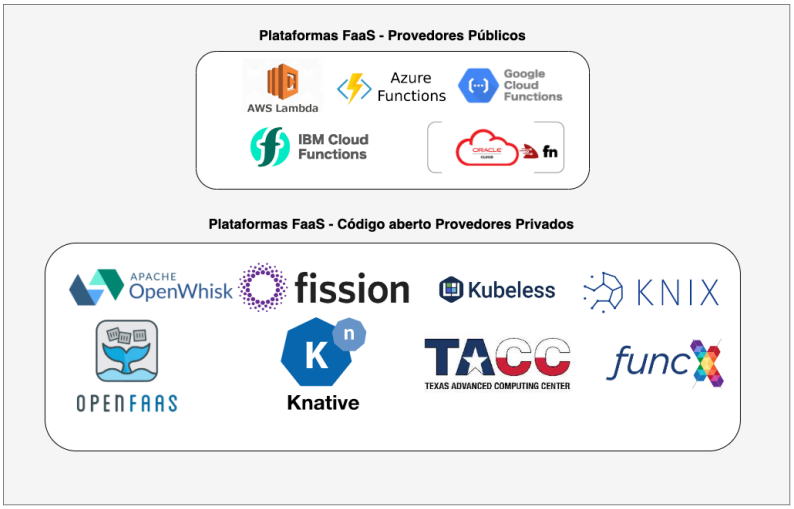


Figura 3.2: Plataformas de FaaS - Provedores públicos X código aberto.

3.2.1 Plataformas FaaS de Código Fechado

As plataformas FaaS de código proprietário ou fechado, são aquelas desenvolvidas e gerenciadas por uma empresa específica, e usam uma tecnologia proprietária e não está disponível para uso geral. Isso significa que os usuários não têm acesso ao código-fonte da plataforma ou não podem modificá-lo para atender às suas necessidades específicas. Dentre esssas plataformas, destacam-se:

- ***AWS Lambda***

  A pioneira *Amazon Web Services* - AWS, oferece o serviço AWS Lambda [5]. Ele permite processamento das linguagens NodeJS, Python, Java, Go, Ruby e C-sharp por meio do .NET Core. Além disso, ele disponibiliza um pacote mensal de um milhão de requisições gratuitas, antes de iniciar a cobrança pelo serviço. O provedor possui diversas parcerias para a implantação, o monitoramento, o gerenciamento de código, e a segurança. A AWS usa a plataforma Firecracker [28] para orquestrar as *microVMs* consumidas pelo AWS Lambda.

  O Lambda é, geralmente, usado para criar aplicativos e serviços de *back-end* escaláveis, mas também pode ser usado para executar tarefas automatizadas, como processamento de arquivos ou transformação de dados. Ele pode ser integrado com outros serviços da AWS, tais como o Amazon S3 [29], o Amazon DynamoDB [30] e o Amazon API Gateway [31], para criar aplicativos completos, sem a necessidade de gerenciar infraestrutura ou servidores.

  O modelo de pagamento do serviço Lambda é baseado no uso, o que significa que os usuários são cobrados apenas pelo tempo de execução de seus códigos e pela quantidade de recursos computacionais usados. Isso torna o serviço uma opção econômica para aplicativos de baixo tráfego, ou para aqueles que precisam de recursos computacionais de maneira intermitente [5].

- ***Azure Functions***

  Por meio do seu serviço de nuvem Azure, a Microsoft disponibiliza o serviço FaaS *Azure Functions* [8], o qual permite o carregamento de funções escritas nas linguagens C-sharp, F-sharp, NodeJS, Java, Python ou PHP. É informado pelo provedor em seu portal que as funções são disponibilizadas em ambiente Windows, mesmo que isso fique transparente para o usuário. Há uma previsão de disponibilização de ambientes usando Linux. Ele oferece gratuidade mensal para o primeiro milhão de chamadas ao serviço.

  As funções do Azure são acionadas automaticamente em resposta a eventos, como a inserção de um novo item em um banco de dados ou o *upload* de um arquivo

em um armazenamento em nuvem. Isso permite que os usuários criem fluxos de trabalho automatizados e aplicativos completos, sem a necessidade de gerenciar a infraestrutura subjacente.

O modelo de pagamento do Azure Functions também é baseado no uso, sendo os usuários cobrados apenas pela quantidade de recursos computacionais usados, e do tempo de execução de suas funções. Isso torna o serviço uma opção econômica para aplicativos com baixo tráfego ou para aqueles que precisam de recursos computacionais de maneira intermitente.

O *Azure Functions* é frequentemente usado em conjunto com outros serviços da Microsoft Azure, tais como o Azure Storage [32], o Azure Event Grid [33] e o Azure Service Bus [34], para criar aplicativos e fluxos de trabalho complexos e automatizados na nuvem [8].

- ***Google Cloud Function***

  O Google Cloud Platform também oferece um serviço de FaaS, que é o *Google Cloud Function* [7]. Esse serviço disponibiliza uma cota inicial de dois milhões de requisições por mês, e inicia a cobrança apenas quando essa cota é ultrapassada. O Google usa o orquestrador de *container* Borg [35] para renderizar seus serviços de nuvem, incluindo o Cloud Functions. O Borg foi a base para o desenvolvimento do Kubernetes[1] [36].

  O serviço suporta várias linguagens de programação, incluindo JavaScript (Node.js), Python, Go e outros. Ele foi projetado para lidar com cargas de trabalho que variam de pequenas tarefas a aplicativos de grande escala. Os usuários podem criar funções de acordo com suas necessidades específicas e, em seguida, publicá-las na plataforma do Google Cloud Functions.

  As funções do Google Cloud são acionadas automaticamente em resposta a eventos, como a inserção de um novo item em um banco de dados, ou o *upload* de um arquivo em um armazenamento em nuvem. Isso permite que os usuários criem fluxos de trabalho automatizados e aplicativos completos, sem a necessidade de gerenciar a infraestrutura subjacente. Ele é frequentemente usado em conjunto com outros serviços da Google Cloud Platform, como o Google Cloud Storage [37], o Google Cloud Pub/Sub [38] e o Google Cloud Firestore [39], para criar aplicativos e fluxos de trabalho complexos e automatizados na nuvem.

- ***Oracle Cloud Functions***

[1] Kubernetes é um orquestrador de contêineres de código aberto usado para automatizar a implantação, dimensionamento e o gerenciamento de aplicativos.

O *Oracle Functions* é sustentado por uma plataforma de processamento de código livre. Ela adota o Fn [40] para suporte operacional do serviço Oracle Cloud Functions. O projeto Fn [40] é uma plataforma *serverless* nativa de contêiner de código aberto que pode ser executada em nuvens públicas e privadas. O Oracle Functions suporta todas as linguagens de programação. Portanto, dada sua capacidade de sustentar a operação em um provedor público de grande escala, o Fn atualmente possui um alto grau de maturidade. Assim, como o Fn executa sobre Docker [41], ele potencialmente pode processar códigos em qualquer linguagem de programação.

As funções do Oracle Cloud são acionadas automaticamente em resposta a eventos, como a inserção de um novo item em um banco de dados, ou o *upload* de um arquivo em um armazenamento em nuvem. Isso permite que os usuários criem fluxos de trabalho automatizados e aplicativos completos, sem a necessidade de gerenciar a infraestrutura subjacente.

O modelo de pagamento do Oracle Cloud Functions também se baseia na utilização, sendo os usuários cobrados apenas pelo tempo de execução e pela utilização de recursos computacionais utilizados. Isso torna o serviço uma opção econômica para aplicativos com baixo tráfego ou para aqueles que precisam de recursos computacionais de maneira variável.

O Oracle Cloud Functions é frequentemente usado em conjunto com outros serviços da Oracle Cloud Infrastructure, tais como o Oracle Cloud Object Storage [42] e o Oracle Cloud Notification Service [43], para criar aplicativos e fluxos de trabalho complexos e automatizados na nuvem.

- ***IBM Cloud Functions***

  O *IBM Cloud Functions* é um serviço de FaaS fornecido pela IBM Cloud, que permite aos usuários executar códigos de forma escalável. O serviço suporta várias linguagens de programação, incluindo JavaScript (Node.js), Python e Swift. Ele é projetado para lidar com cargas de trabalho que variam de pequenas tarefas a aplicativos de grande escala. Os usuários podem criar funções de acordo com suas necessidades específicas e, em seguida, publicá-las na plataforma do IBM Cloud Functions.

  A IBM utiliza uma solução *open-source*, neste caso o Open-Whisk [44]. O Apache OpenWhisk [44] é uma plataforma sem servidor distribuída, de código aberto que executa funções em resposta a eventos, em qualquer escala. O seu modelo de pagamento também é baseado no uso pelo tempo de execução e pela quantidade de recursos computacionais utilizados, ofertando também uma opção economicamente viável para aplicativos com carga variada.

O IBM Cloud Functions é, frequentemente, usado em conjunto com outros serviços da IBM Cloud, tais como o IBM Cloud Object Storage [45] e o IBM Cloud Message Hub [46], para criar aplicativos e fluxos de trabalho complexos e automatizados na nuvem.

3.2.2 Plataformas de FaaS de Código Aberto

As plataformas FaaS de código aberto são ferramentas que permitem aos usuários executarem códigos de forma escalável, e sem a necessidade de gerenciar a infraestrutura subjacente, mas que são desenvolvidas e mantidas pela comunidade de código aberto. Essas plataformas permitem que os desenvolvedores criem, implantem e gerenciem funções sem servidor em suas próprias infraestruturas, ou em provedores de nuvem pública. Dentre as principais plataforma de FaaS de código aberto, destacam-se:

- ***Openwhisk***

  O OpenWhisk gerencia infraestrutura, servidores e dimensionamento usando contêineres do Docker [44]. O projeto inclui uma interface de linha de comando (CLI, do inglês, *Command-Line Interface*) baseada em API REST [4], juntamente com outras ferramentas para oferecer suporte a pacotes, serviços de catálogo e muitas opções populares de implantação de contêiner. Ele oferece suporte a várias linguagens de programação, incluindo Java, Node.js, Python, Swift e PHP, além de permitir que os usuários personalizem e estendam a plataforma com seus próprios componentes e serviços.

  Ele também possui uma API de gerenciamento *RESTful* para permitir que os usuários gerenciem suas funções, acionadores e regras de forma programável e funciona com base em uma arquitetura distribuída, na qual cada função é executada em um contêiner separado. Isso permite a escalabilidade horizontal em resposta a cargas de trabalho variáveis. As funções do OpenWhisk são ativadas por eventos (acionamentos de API, operações agendadas e etc.), que podem ser gerados por fontes externas, como dispositivos IoT, bancos de dados ou mensagens de fila.

  O OpenWhisk pode ser implantado em diferentes plataformas de nuvem, tais como AWS, Microsoft Azure, Google Cloud Platform e IBM Cloud. Ele também pode ser executado em infraestruturas locais (nuvem privada), permitindo que as empresas criem soluções personalizadas e privadas de computação sem servidor. Ele é uma plataforma de código aberto, gerenciada pela Apache Software Foundation [47], o que significa que a comunidade pode contribuir com código, correções de *bugs* e

melhorias na plataforma. Isso ajuda a garantir que a plataforma continue evoluindo e atendendo as necessidades em constante mudança dos usuários.

- ***Fission***

  O Fission [48] é um *framework* de código aberto executado no Kubernetes e extensível a qualquer linguagem de programação. A arquitetura do Fission é baseada em contêineres, permitindo que as funções sejam implantadas em qualquer ambiente que suporte contêineres. Cada função do Fission é executada em um contêiner separado, o que garante a elasticidade horizontal em resposta a cargas de trabalho variáveis. Além disso, o Fission permite que os desenvolvedores personalizem e estendam a plataforma com seus próprios componentes e serviços.

  No Fission, o núcleo é escrito na linguagem Go [48], e as partes específicas das linguagens são isoladas em algo chamado de ambiente inferior. Esta plataforma mantém um *pool* de contêineres "quentes", cada um com um pequeno carregador dinâmico. Quando uma função é chamada pela primeira vez, ou seja, inicializada a frio, um contêiner em execução é escolhido e a função é carregada. Esse *pool* é o que torna o Fission rápido, atingindo latências de partida a frio de cerca de 100 milissegundos.

  O Fission também oferece suporte a vários gatilhos (*triggers*) de eventos, como HTTP, cron, Kafka e outros eventos personalizados. Os gatilhos permitem que as funções do Fission sejam acionadas automaticamente em resposta a eventos específicos, o que pode ajudar a automatizar fluxos de trabalho complexos. Também é uma plataforma de código aberto mantida pela comunidade, permitindo que os usuários contribuam com o código, correções de *bugs* e melhorias na plataforma.

- ***Knative***

  O Knative [49] permite que os desenvolvedores usem uma arquitetura orientada a eventos. Uma arquitetura orientada a eventos é baseada no conceito de relacionamentos desacoplados entre produtores de eventos que criam eventos; e consumidores de eventos, ou coletores, que recebem eventos [49]. O Knative é construído em cima do Kubernetes, tornando-o uma extensão dessa plataforma. Ele utiliza conceitos de Kubernetes, tais como serviços, *pods* e *deployments*, para gerenciar funções sem servidor, oferecendo recursos avançados, como por exemplo, escala automática e recursos para diferentes estratégias de *deploy*, como implantações canário e implantações graduais. Além disso, ele garante que as funções sem servidor sejam implantadas e gerenciadas de maneira confiável e escalável.

Ele oferece suporte a várias linguagens de programação, incluindo Java, Node.js, Python e Ruby, e também fornece uma API aberta para integração com outras ferramentas de nuvem, tais como serviços de banco de dados e filas de mensagens. O Knative é altamente configurável e pode ser facilmente personalizado para atender às necessidades específicas de diferentes aplicativos. Ele é modular e permite que os desenvolvedores escolham os componentes que desejam usar para criar suas funções sem servidor.

O Knative também é uma plataforma de código aberto mantida por sua comunidade, na qual as melhorias e correções são evoluídas por meio da contribuição de seus integrantes [49].

- ***OpenFaaS***

  O OpenFaaS [50] é uma ferramenta de código aberto que executa no Kubernetes, e tem como objetivo facilitar a implementação de microsserviços e funções orientadas a eventos pelos desenvolvedores. Ele fornece um ambiente de execução sem servidor para funções, em que cada função é executada em sua própria instância de contêiner. Isso permite que as funções sejam escalonadas independentemente, com base na demanda, e que sejam executadas sem a necessidade de gerenciar ou provisionar servidores.

  O OpenFaaS também inclui uma API de *gateway* para roteamento de solicitações de funções, e gerenciamento de autorização e autenticação, sendo altamente personalizável e extensível. Ele permite que os desenvolvedores integrem facilmente seus próprios serviços e ferramentas com a plataforma. Outra vantagem é que ele possui uma ampla variedade de recursos, tais como monitoramento, *logging* e métricas, que ajudam a garantir a confiabilidade e a escalabilidade das funções.

  Em resumo, o OpenFaaS é uma plataforma flexível e poderosa que permite aos desenvolvedores criar e implantar funções sem servidor, em qualquer ambiente de computação em nuvem. Com ele, as empresas podem reduzir custos, aumentar a agilidade e melhorar a escalabilidade de suas aplicações [51].

- ***Kubeless***

  O Kubeless [52] foi projetado para ser executado em um *cluster* Kubernetes existente, o que significa que ele aproveita a escalabilidade, a resiliência e a segurança oferecidas pelo Kubernetes. Ele suporta várias linguagens de programação, incluindo Python, Node.js, Ruby, PHP e Go, permitindo que os desenvolvedores escrevam funções usando sua linguagem de programação favorita. Ele também fornece uma CLI (*Command Line Interface*) para implantar funções facilmente, e uma API *RESTful*

para gerenciar funções e *triggers*. Os desenvolvedores podem implantar funções com ou sem um *trigger*, e usar gatilhos para disparar funções em resposta a eventos.

O kubeless suporta vários tipos de *trigger*, incluindo gatilhos HTTP, gatilhos de mensagens (como por exemplo, Kafka ou NATS), gatilhos de banco de dados (tais como MySQL ou MongoDB), e gatilhos de eventos de nuvem (tais como AWS S3 ou Azure Blob Storage). Com o Kubeless, os desenvolvedores podem implantar funções rapidamente e facilmente, sem precisar gerenciar infraestrutura ou lidar com preocupações de escalabilidade, resiliência e segurança. Isso permite que eles se concentrem no desenvolvimento de código de negócios de alto valor, e na entrega rápida de recursos.

- ***Knix***

  O KNIX MicroFunctions (conhecido anteriormente por SAND) [53] é uma plataforma de computação sem servidor de código aberto de alto desempenho, projetada para minimizar atrasos de inicialização para execução de funções. A ferramenta visa fornecer suporte para funções persistentes e otimizar o uso de recursos. As microfunções do KNIX MicroFunctions são pequenos trechos de código que realizam tarefas específicas, como por exemplo processamento de eventos, transformação de dados ou execução de tarefas simples. Ele suporta várias linguagens de programação, incluindo Python, Node.js, Go e Ruby. Ele também fornece uma interface visual para facilitar a criação e a implantação de microfunções, bem como uma API *RESTful* para gerenciamento e automação.

  Outra vantagem do KNIX MicroFunctions, é que ele permite que os desenvolvedores criem fluxos de trabalho (*workflows*) conectando várias microfunções para formar um processo mais complexo. Isso permite que os desenvolvedores criem aplicativos sem servidor completos e altamente escaláveis, que podem ser dimensionados automaticamente para lidar com picos de demanda. Com ele os desenvolvedores podem criar microfunções de forma ágil e implantá-las em um ambiente altamente escalável e gerenciável. Isso permite que eles se concentrem no desenvolvimento de código de negócios de alto valor e na entrega rápida de recursos.

- ***Abaco***

  O Texas Advanced Computing Center - TACC usa Abaco [54] como sua solução FaaS. A TACC é um provedor de nicho, focado em pesquisas acadêmicas. O Abaco foi projetado para ser executado em um *cluster* de contêineres gerenciados pelo Docker Swarm ou Kubernetes. Ele fornece uma API RESTful para a criação e gerenciamento de funções sem servidor, bem como gatilhos para acionar a execução

de funções em resposta a eventos específicos. Ele suporta várias opções de armazenamento de dados, como bancos de dados NoSQL e armazenamento em nuvem, permitindo que os desenvolvedores criem aplicativos sem servidor altamente escaláveis e resilientes.

Com o Abaco os desenvolvedores podem criar funções sem servidor e implantá-las em um ambiente gerenciável e escalável. O Abaco é uma plataforma de FaaS de código aberto, o que significa que é altamente personalizável, e pode ser estendido para atender às necessidades específicas de um aplicativo.

- ***FuncX***

  O FuncX [55] é uma plataforma FaaS distribuída que permite a execução de funções remotas flexíveis, escalonáveis e de alto desempenho. Ao contrário das plataformas FaaS centralizadas, o FuncX permite que os usuários executem funções em computadores heterogêneos remotos, de *laptops* a *clusters* remotos, nuvens e supercomputadores [55]. A heterogeneidade característica do FuncX é sua grande vantagem sobre outras ferramentas. Ele foi projetado para ser executado em um *cluster* gerenciado pelo Kubernetes ou Apache Mesos, e suporta várias linguagens de programação, incluindo Python, Java e C++. Ele fornece uma API *RESTful* para criar, implantar e gerenciar funções sem servidor, bem como gatilhos para acionar a execução de funções em resposta a eventos específicos.

  O FuncX também suporta várias opções de armazenamento de dados, incluindo armazenamento em nuvem, bancos de dados NoSQL e sistemas de arquivos distribuídos. Isso permite que os desenvolvedores criem aplicativos sem servidor altamente escaláveis e resilientes, que possam manipular grandes volumes de dados. O FuncX tem como objetivo fornecer um ambiente escalável e eficiente para funções sem servidor, permitindo que os desenvolvedores se beneficiem da sua rápida entrega de recursos. Ele é uma plataforma de FaaS de código aberto, personalizável e flexível, capaz de atender as demandas especificas de um aplicativo.

- ***TinyFaaS***

  TinyFaaS [56] é uma plataforma FaaS projetada para ser executada em sistemas embarcados e de borda, como dispositivos IoT e *gateways* de borda. Depois que uma função é implantada no TinyFaaS, os manipuladores de função são criados automaticamente. Ele suporta várias linguagens de programação, incluindo Python e Rust. Ele fornece uma API RESTful para criar, implantar e gerenciar funções sem servidor, bem como gatilhos para acionar a execução de funções em resposta a eventos específicos.

O TinyFaaS foi projetado para ser altamente leve e escalável, permitindo que as funções sejam executadas em recursos limitados. Ele suporta a execução de funções em contêineres, e fornece um sistema de gerenciamento de recursos para garantir que as funções tenham acesso aos recursos necessários para executar com eficiência, permitindo a construção de aplicativos sem servidor altamente escaláveis e resilientes, que possam ser executados perto dos dados gerados pelos dispositivos. Ele também é uma plataforma FaaS de código aberto, o que garante a possibilidade de personalização e pode ser ampliado visando o atendimento de necessidades específicas.

3.3 Análise das Ferramentas FaaS de Código Aberto

Para descrever e definir o grau de maturidade de cada ferramentas FaaS de código livre estudadas neste trabalho, foram considerados alguns critérios tais como escalabilidade, suporte a idiomas, operação em larga escala, suporte e documentação. A Tabela 3.1 apresenta a análise feita com as plataformas de FaaS de código aberto.

Tabela 3.1: Características das plataformas de FaaS.

Ferramenta	**Ambiente**	**Linguagens**	**Provedor**	**Maturidade**
Fission [48]	Kubernetes	NodeJS, Python, Go, Java, Ruby, Binary, PHP, .NET, .NET 2.0 e Perl	Privado	Alta
Fn [40]	Kubernetes	Várias (Imagens Docker)	Oracle	Alta
Knative [49]	Kubernetes	Várias (Imagens Docker)	Privado	Alta
OpenFaaS [51]	Kubernetes	Várias (Imagens Docker)	Privado	Alta
Openwhisk [44]	Kubernetes, Openshift e Docker-Compose	Go, Java, NodeJS, .NET, PHP, Python, Ruby, Rust, Scala, Swift,Ballerina e Deno	IBM	Alta
Kubeless [52]	Kubernetes	Java, NodeJS e VertX	Privado	Média
Knix [53]	Kubernetes	Python and Java	Privado	Média
Abaco [54]	Docker-Compose	Várias (Imagens Docker)	TACC	Baixa
FuncX [55]	Aplicação específica	Python	Privado	Baixa
TinyFaaS [56]	Docker	NodeJS	Privado	Baixa

De acordo com a seção anterior, nota-se que há diferentes plataformas de FaaS ofertadas no mercado, cada uma com características específicas. Assim sendo, notou-se a

necessidade de avaliar, dentre essas plataformas de FaaS, qual seria a mais indicada para ser usada no ambiente de nuvem privada da DATAPREV. Para isso, foram feitos diversos testes que resultaram na publicação do artigo *Comparison of FaaS Platform Performance in Private Clouds* [11], cujos resultados são apresentados no Capítulo 4.

Capítulo 4

Análise de Desempenho das Plataformas FaaS de Código Aberto

Ao comparar plataformas FaaS, é crucial considerar o caso de uso específico e os requisitos do aplicativo para determinar qual plataforma oferecerá o melhor desempenho. Isso inclui avaliar fatores como tempo de execução, escalabilidade, custos, suporte a linguagens de programação, integração com outros serviços e ferramentas, facilidade de uso e segurança. Cada plataforma tem suas próprias características e vantagens, portanto, uma análise detalhada de como elas se alinham com os objetivos e demandas é essencial para fazer a escolha mais adequada. Assim sendo, este capítulo descreve na Seção 4.1 a metodologia utilizada nos testes e nas análises dos resultados. Na sequência, a Seção 4.2 apresenta o planejamento fatorial do experimento, em seguida, na Seção 4.3 é apresentada a dinâmica do teste de confiabilidade realizado. Na Seção 4.4 são apresentadas os resultados das latências obtidas no experimento. Na Seção 4.5, a análise comparativa entre as plataformas FaaS é descrita, seguida da Seção 4.6 que descreve a análise fatorial utilizada no experimento. Na Seção 4.7, através do planejamento fatorial são apresentadas as análises de impacto dos fatores em relação aos resultados do experimento, e por fim, na Seção 4.8, são apresentados os resultados da análise de confiabilidade executada nas plataformas.

4.1 Metodologia

A infraestrutura utilizada neste experimento foi provisionada na solução de nuvem privada da DATAPREV, na qual é possível fornecer configurações personalizadas de memória, CPU, disco, entre outros. As plataformas FaaS de código aberto, classificadas no capítulo anterior, com alto grau de maturidade na Tabela 3.1, foram implementadas em uma infraestrutura, composta por três máquinas virtuais, contemplando um *Cluster Kubernetes* com Docker para gerenciamento das imagens utilizadas. A Tabela 4.1 mostra os valores

para cada recurso que foi usado para provisionar os três *clusters* Kubernetes, que suportam cada plataforma aplicada neste experimento.

Tabela 4.1: Parâmetros dos *clusters*.

Número de Servidores	3
Sistema Operacional	RHEL 8.3
Armazenamento	30GB
Memória RAM	4GB
CPU	4 (2.5Ghz)
Hospedeiro	Power Egde R900 - Intel Xeon E7 4870
Hypervisor	VMware ESXi 6
Docker	v20.10.7
Kubernetes	v1.21.3
Apache JMeter	v5.4.1

A plataforma Kubernetes foi escolhida para hospedar todas as ferramentas a serem comparadas. Isso permitiu a experiência com foco nas estratégias adotadas pela ferramenta, evitando que diferenças de plataforma interferissem nos resultados. Contudo, embora as plataformas Fn e Knative tenham sido classificadas com alto grau de maturidade, devido a dificuldades decorrentes do ambiente *on-premise* trabalhando atrás de um *proxy* corporativo, não foi possível atingir a configuração almejada para uma comparação justa com outras plataformas, deixando apenas OpenFaaS, Fission e OpenWhisk como o foco da avaliação realizada. As instalações das plataformas seguiram as instruções registradas nos manuais oferecidos pelos fornecedores. Todavia, nos casos do OpenFaaS e do OpenWhisk, seus limites de ativação por minuto foram desconsiderados para equalizar os limites das três plataformas analisadas. Nenhuma alteração foi feita no padrão da configuração do Fission.

Após a implementação das plataformas de FaaS, quatro funções para testes de *performance*, oriundas do *benchmark FaaSDom* [57], foram adaptadas e publicadas em cada uma das três plataformas analisadas. Para efetuar os disparos de cada função, em cada plataforma, e se obter as métricas de avaliação dessas ativações, foi utilizada a ferramenta *Apache Jmeter* [58]. Ela foi implementada em um servidor apartado, porém na mesma rede de dados dos *clusters* que abrigam as plataformas, no intuito de não haver nenhuma interferência na rede durante os testes. A Figura 4.1 apresenta a arquitetura do experimento, na qual e possível notar que foi usada a mesma infraestrutura para cada ferramenta testada, garantindo a correta comparação entre as plataformas.

Além dos testes realizados por meio do *benchmark* FaasDom, uma análise fatorial 2^k foi realizada sob as métricas obtidas pelo resultado, permitindo uma análise estatística mais profunda dos fatores que influenciaram nos resultados dos testes.

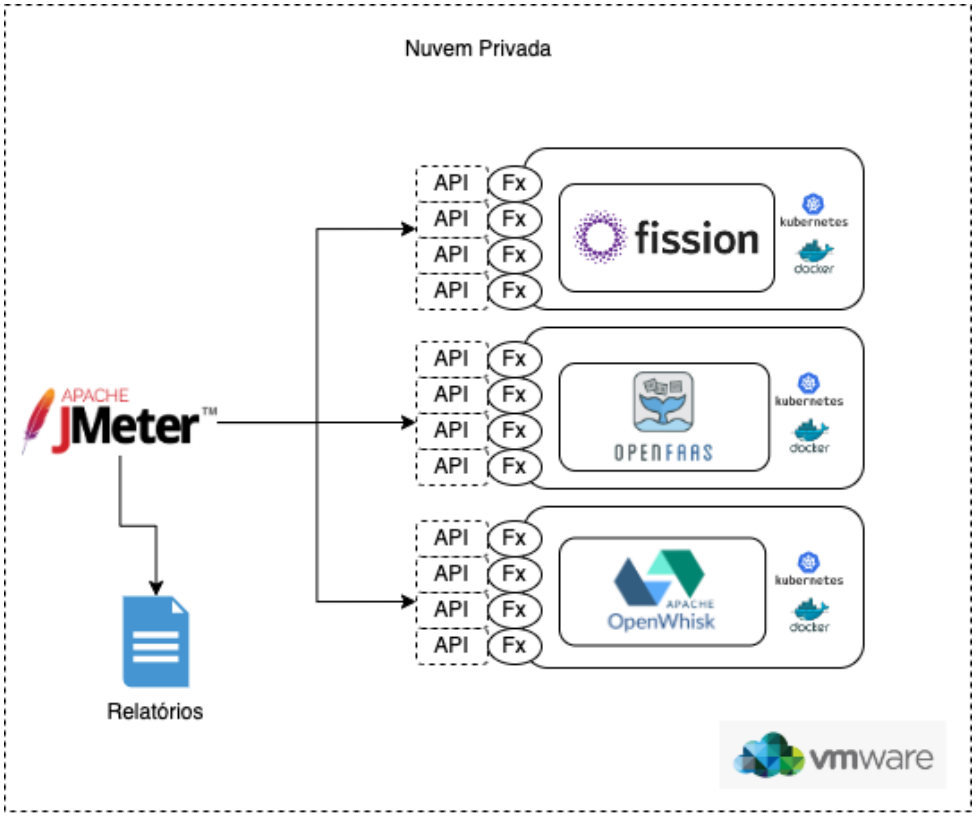


Figura 4.1: Arquitetura do experimento no ambiente de nuvem privada da DATAPREV.

4.2 Planejamento Fatorial 2^k

Os planejamentos fatoriais 2^k [59] são um tipo de *design* experimental usado na análise estatística para estudar o efeito de duas ou mais variáveis independentes, cada uma com dois níveis, em uma variável dependente. O 2^k refere-se ao número de variáveis independentes, ou fatores, no projeto e a parte "fatorial", refere-se à forma como os níveis de cada fator são sistematicamente combinados no projeto.

O objetivo dos projetos fatoriais 2^k é permitir que os pesquisadores estudem com eficiência os efeitos de múltiplos fatores em uma variável dependente, usando um número relativamente pequeno de execuções experimentais. Ao variar sistematicamente os níveis de cada fator, os pesquisadores podem determinar quais fatores têm efeitos significativos na variável dependente, bem como quaisquer interações entre os fatores. Os benefícios de usar um experimento fatorial 2^k incluem [60]:

- Eficiência: o *design* permite que os pesquisadores estudem vários fatores em um número relativamente pequeno de execuções experimentais, o que pode economizar tempo e recursos;

- Simplicidade: o *design* é fácil de configurar e analisar, tornando-o uma escolha popular para pesquisadores em muitos campos;

- Flexibilidade: o projeto pode ser adaptado para incluir fatores ou níveis adicionais, tornando-se uma ferramenta versátil para estudar uma ampla gama de questões de pesquisa.

Ao selecionar e manipular cuidadosamente os fatores no projeto, os pesquisadores podem obter informações valiosas sobre as relações entre as variáveis, e melhorar sua compreensão dos processos subjacentes no trabalho.

Para então identificar quais dos fatores tiveram a maior influência nos resultados, após a realização dos testes, foi utilizado o planejamento fatorial 2^k para iniciar a análise de cada cenário de comparação entre as plataformas utilizadas. Em cada planejamento fatorial, os fatores analisados foram: plataforma, concorrência e função.

A Tabela 4.2 apresenta a matriz de planejamento fatorial 2^k, que é uma tabela que exibe as diferentes combinações de níveis para cada um dos k fatores em um planejamento fatorial 2^k.

Tabela 4.2: Fatores do desenho fatorial - Openwhisk, OpenFaaS e Fission.
Openwhisk (1), OpenFaaS (2) e Fission (3)

Concorrência	**Função**	**Plataforma**		
Baixa (1)	Matrix	Openwhisk	Openwhisk	OpenFaaS
Alta (16)	Delay	Fission	OpenFaaS	Fission
Baixa (1)	Delay	Openwhisk	Openwhisk	OpenFaaS
Alta (16)	Factors	Fission	OpenFaaS	Fission
Baixa (1)	Delay	Openwhisk	Openwhisk	OpenFaaS
Alta (16)	Filesystem	Fission	OpenFaaS	Fission
Baixa (1)	Factors	Openwhisk	Openwhisk	OpenFaaS
Alta (16)	Filesystem	Fission	OpenFaaS	Fission
Baixa (1)	Matrix	Openwhisk	Openwhisk	OpenFaaS
Alta (16)	Factors	Fission	OpenFaaS)	Fission
Baixa (1)	Matrix	Openwhisk	Openwhisk	OpenFaaS
Alta (16)	Filesystem	Fission	OpenFaaS	Fission

4.3 Teste de Confiabilidade

Para observar o comportamento das plataformas em situações extremas, foi elaborado um teste de estresse (*stress test*) utilizando a função Matrix. Esse teste foi projetado para avaliar como as plataformas se comportam sob pressão intensa e para identificar os pontos de falha em cenários de alta demanda. O uso da função Matrix permitiu simular condições adversas que as plataformas poderiam enfrentar no mundo real, proporcionando uma visão clara de suas capacidades de resposta.

Neste teste, as plataformas foram submetidas a níveis exponenciais de concorrência da função Matrix, até que fossem incapazes de responder. Isso envolveu a criação de múltiplas

instâncias da função sendo executadas simultaneamente, aumentando progressivamente o número de solicitações. O objetivo era observar o ponto em que cada plataforma começava a falhar, permitindo uma análise detalhada de sua robustez e capacidade de manutenção de desempenho sob carga extrema.

Os resultados do experimento relacionado à confiabilidade são detalhados na Seção 4.8. Esses resultados fornecem *insights* valiosos sobre o desempenho das plataformas e suas limitações. A análise dos dados obtidos revelou quais plataformas são mais resilientes e quais precisam de melhorias significativas. Essa informação é crucial para desenvolvedores e engenheiros que buscam otimizar a confiabilidade e a eficiência de suas plataformas em ambientes de produção.

4.4 Análise de Latência

Para a análise de latência foram utilizadas no experimento, as quatro principais funções presentes no *benchmark* FaaSDom [57] para se avaliar a latência dos tempos de resposta em diferentes cenários. A função de *Latency* é responsável por medir e avaliar o tempo de resposta das funções executadas na plataforma FaaS. Ela simula solicitações de usuários e registra o tempo necessário para que a plataforma responda a cada uma delas.

A função *Matrix* é projetada para testar a capacidade de escalabilidade da plataforma FaaS. Ela cria uma matriz de solicitações concorrentes, simulando picos de tráfego, e monitora como a plataforma responde a essas solicitações.

A função *Factors* é responsável por realizar testes de estresse na plataforma FaaS, expondo-a a condições extremas para avaliar sua estabilidade e confiabilidade. Ela simula cenários de uso intensivo de recursos, como a execução de um grande número de funções simultaneamente ou o processamento de cargas de trabalho pesadas.

Por último, a função *Filesystem* é responsável por testar o sistema de arquivos da plataforma FaaS, avaliando sua capacidade de armazenar, recuperar e manipular dados de forma eficiente e segura. Ela simula operações de leitura e gravação em arquivos, medindo o tempo necessário para realizar essas operações e analisando a integridade dos dados armazenados. Todas as funções utilizadas buscam avaliar o desempenho e a confiabilidade da plataforma, ajudando os desenvolvedores a garantir que suas aplicações *serverless* possam utilizar a plataforma de maneira eficaz e segura.

A análise de latência foi calculada por meio dos registros obtidos pela ferramenta *JMeter* [58] durante os testes. Assim sendo, conforme apresentado na Tabela 4.3, foram geradas as médias dos valores de latência em milisegundos (ms) para todos os níveis de concorrência de cada função, em cada plataforma. Além disso, também é possível verificar os valores de Log_{10}, calculados pela média obtida em cada nível de acionamento.

Tabela 4.3: Latência (ms) das funções *Delay (Latency), Matrix, Factors* e *Filesystem.*

Plataforma	-	Função							
		Delay (latency)		Matrix		Factors		Filesystem	
		Média	Log_{10}	Média	Log_{10}	Média	Log_{10}	Média	Log_{10}
Fission	1	80,50	1,91	1,00	446,30	12762,50	4,11	4788,40	3,68
	2	34,65	1,54	423,50	2,63	12781,80	4,11	4785,10	3,68
	4	23,80	1,38	443,53	2,65	13414,83	4,13	5332,23	3,73
	8	26,39	1,42	547,56	2,74	14820,28	4,17	9001,76	3,95
	16	30,68	1,49	889,91	2,95	17589,30	4,25	27677,16	4,44
OpenFaaS	1	14,10	1,15	393,00	2,59	9084,00	3,96	4617,00	3,66
	2	11,50	1,06	371,55	2,57	18433,90	4,27	9373,70	3,97
	4	10,85	1,04	896,65	2,95	38854,73	4,59	26730,43	4,43
	8	11,84	1,07	1907,26	3,28	33404,96	4,52	24423,61	4,39
	16	10,63	1,03	3518,90	3,55	30479,75	4,48	27124,71	4,43
OpenWhisk	1	71,10	1,85	332,70	2,52	8963,00	3,95	3950,80	3,60
	2	52,80	1,72	500,90	2,70	8890,80	3,95	5074,35	3,71
	4	34,90	1,54	391,60	2,59	9582,50	3,98	4573,18	3,66
	8	128,29	2,11	448,53	2,65	13465,26	4,13	10784,60	4,03
	16	2951,51	3,47	3974,74	3,60	23349,06	4,37	12842,34	4,11

Na execução sem concorrência (somente com uma requisição) foram gerados dez valores de resultado para cada função em cada plataforma. Para concorrência nível dois foram gerados 20 valores de resultado, para concorrência nível quatro foram 40, para concorrência nível oito foram 80, e para concorrência nível dezesseis foram 160 valores de latência para cada função. Para representar adequadamente as amostras em cada bloco de execução, e permitir uma análise consolidada dos dados, foi necessário, por meio da análise de gráficos quantil-quantil, definir um valor para essa representação.

A Figura 4.2 apresenta todas as execuções de uma das funções na plataforma Fission, no nível 16 de concorrência. A Figura 4.3 apresenta todas as execuções de uma das funções na Plataforma OpenFaaS também no nível 16 de concorrência. Por último, a Figura 4.4, mostra todas as execuções de uma das funções na plataforma Openwhisk, no nível 16 de concorrência.

Em todos os casos, verifica-se que os valores não seguem completamente uma distribuição normal, pois a forma dos gráficos não se parece com uma linha reta seguindo uma linha de tendência. Entretanto verifica-se o Fission apresentando menor latência, seguido pela OpenFaaS e Openwhisk, evidenciando as diferenças na estrategia de cada plataforma. Assim, a média foi adotada como métrica representativa das amostras em cada concorrência, representando melhor os dados com características de dispersão.

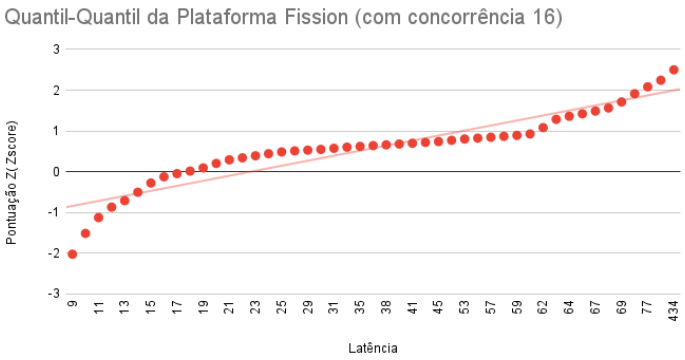


Figura 4.2: Quantil-quantil da plataforma Fission.

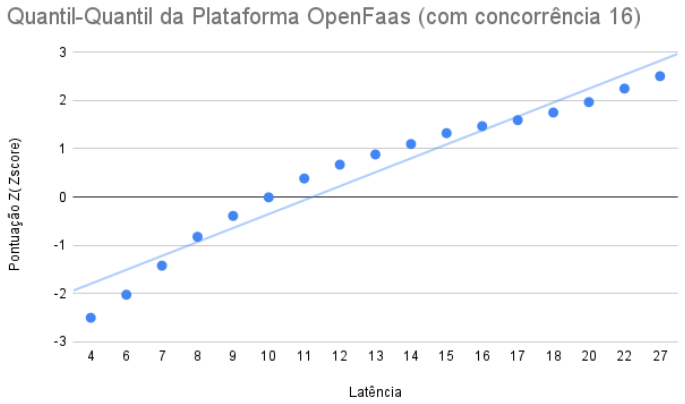


Figura 4.3: Quantil-quantil da plataforma OpenFaaS.

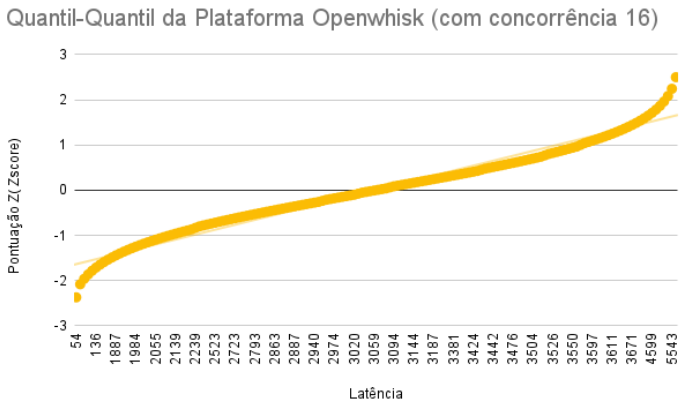


Figura 4.4: Quantil-quantil da plataforma OpenWhisk.

4.5 Análise Comparativa entre as Plataformas FaaS

A Figura 4.5 mostra a evolução das execuções da função *Delay (latency)* em cada nível de competição, nas três plataformas. Vale ressaltar que em todos os gráficos, as médias

de latência são logaritimamente transformadas para proporcionar uma melhor visualização das informações. Assim, conforme apresentado nas Figuras 4.2, 4.3 e 4.4 é possível perceber que ao executar sem concorrência, o OpenFaaS obteve um resultado de latência muito menor do que o valor obtido pelo Fission, seguido ligeiramente pelo OpenWhisk. E, conforme o nível de concorrência aumenta, o OpenWhisk aumenta a distância para Fission e OpenFaaS. Até a concorrência nível 4, a plataforma Fission experimenta uma leve diminuição na latência, enquanto o OpenWhisk apresenta um crescimento considerável. Logo, é possível inferir que o OpenFaaS foi mais eficiente do que as outras plataformas no processamento da função *Delay (Latency)* em todos os níveis de simultaneidade.

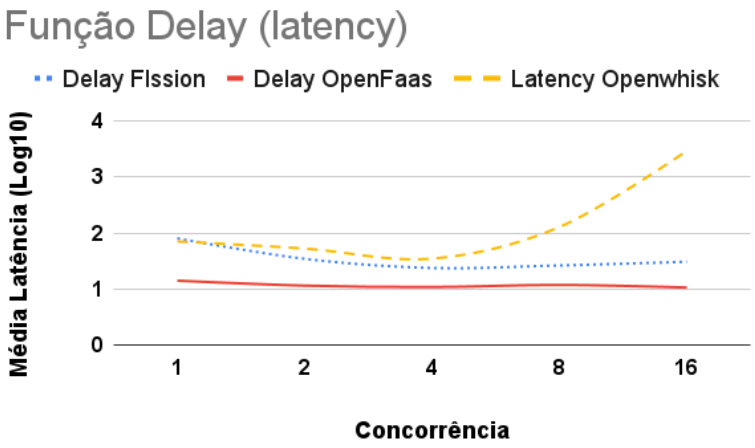


Figura 4.5: Função *Delay (Latency)*.

A Figura 4.6 também mostra a evolução das execuções, nesse caso da função Matrix, em cada nível de competição em ambas as plataformas.

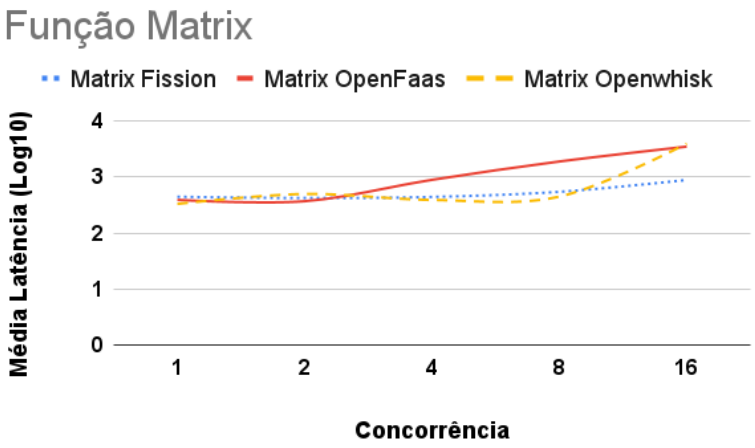


Figura 4.6: Função *Matrix* nas três plataformas.

É possível observar que, sem concorrência, todas as plataformas apresentam resultados equivalentes para latência. No entanto, quando o nível de concorrência aumenta, enquanto o Fission se mantém estável, o OpenFaaS sofre um crescimento na latência a partir da concorrência nível 2, e o OpenWhisk a partir da concorrência nível 8. Isso mostra que a plataforma Fission foi capaz de processar a função Matrix de forma mais eficaz que as demais plataformas, quando a concorrência é superior ao nível 8.

Os resultados mostraram que na função *Delay (latency)*, a Plataforma OpenFaaS se destaca em eficiência, seguido pelo Fission e depois pelo OpenWhisk. Na função Matrix (que requer mais processamento), a plataforma OpenWhisk em execuções com concorrência de até 8 apresenta menor eficiência quanto à latência, sendo superado pelo Fission após execuções com concorrência 10. O OpenFaaS, por outro lado, diminui sua eficiência na concorrência de nível 4.

A Figura 4.7 apresenta a evolução da latência na execução da função *Factors*, em cada plataforma ao longo de cada nível de concorrência.

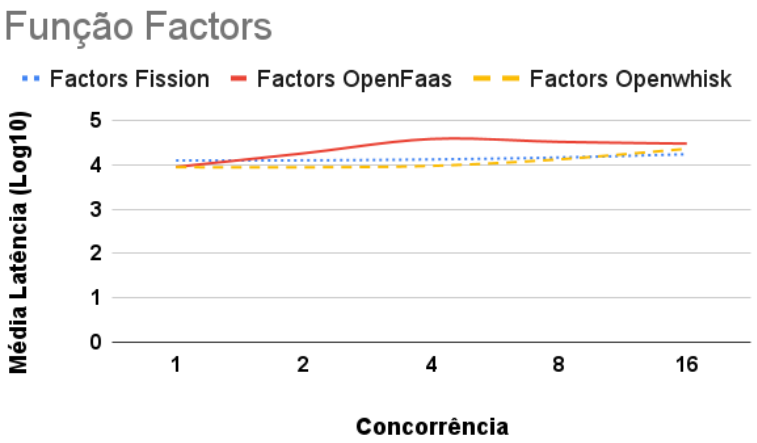


Figura 4.7: Função *Factors* nas três plataformas.

Nesse caso, o OpenWhisk tem o menor registro de latência sem concorrência, seguido pelo OpenFaaS e o Fission. No entanto, à medida que o número de concorrência aumenta, o OpenWhisk e Fission mostram consistência em seus níveis de latência. Por outro lado, o OpenFaaS mostra um aumento a partir da concorrência 2.

Também transformada de forma logarítmica, a Figura 4.8 apresenta a evolução da latência na execução da função *Filesystem* em cada plataforma, em todos os níveis de simultaneidade. Quando os recursos relacionados ao disco são usados, é possível notar que o OpenWhisk tem o menor registro de latência no primeiro teste (sem concorrência), seguido pelo OpenFaaS, e Fission com latência um pouco maior, mostrando extrema similaridade nos testes iniciais. Dado o aumento da concorrência, a plataforma OpenFaaS

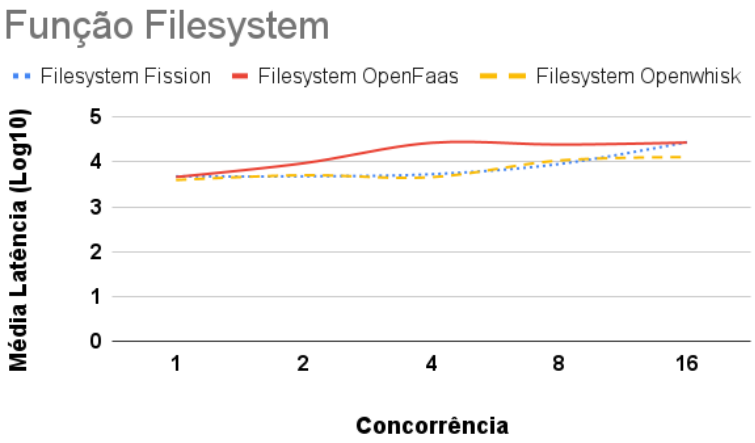


Figura 4.8: Função *Filesystem* nas três plataformas.

experimenta um crescimento maior do que outras plataformas já nos níveis 2 e 4 de concorrência, mantendo-se constante em 8 e 16. As plataformas Fission e OpenWhisk mantêm seus níveis de latência até o nível 4 de concorrência, obtendo ligeiro crescimento em ambos os casos. Com nível de concorrência 16, o Openwhisk mostrou os melhores registros entre as plataformas analisadas.

4.6 Análise Fatorial

A Tabela 4.4 apresenta a matriz de *design* fatorial utilizada em todos os casos, sendo possível observar os oito testes performados, bem como os efeitos e suas correlações. A matriz de *design* é um componente chave do *design* experimental, pois descreve as condições específicas que serão testadas no experimento. Os níveis inferiores são indicados por -1, enquanto os superiores são denotados por 1. Todos os dados usados para extração das médias, bem como a soma da diferença entre cada média, o quadrado médio, e o cálculo parcial da soma dos erros quadrados neste experimento, para cada teste (em cada linha), foram disponibilizados no GitHub[1]. As colunas denominadas PxC, PxF, CxF e PxFxC representam respectivamente as correlações entre: Plataforma (P) e Concorrência (C) - PxC, Plataforma (P) e Função (F) - PxF, Concorrência (C) e Função (F) - CxF e Plataforma (P) e Função (F) e Concorrência (C) - PxFxC.

Em cada caso, as plataformas, as funções e as concorrências foram classificadas de acordo com as Tabelas 4.4 e 4.2, calculando-se assim os efeitos, as variações, o *RSS - Residual Sum of Squares*, o *SSY - Sum of the Square of Y* e o *SST - Sum of the Total Square*, permitindo-se obter o valor das frações de cada fator e suas relações em cada

[1] `https://github.com/unb-faas/private-platform-benchmark`

resultado. A soma residual dos quadrados (RSS) é uma medida da diferença entre os valores observados de uma variável dependente, e os valores previstos por um modelo de regressão. A Soma do Quadrado de Y (SSY) e a Soma do Quadrado Total (SST) são medidas da variabilidade total ou dispersão de uma variável dependente.

Tabela 4.4: Matriz de *design* fatorial.

Teste	**Plataforma**	**Concorrência**	**Função**	**PxC**	**PxF**	**CxF**	**PxFxC**
1	-1	-1	-1	1	1	1	-1
2	1	-1	-1	-1	-1	1	1
3	-1	1	-1	-1	1	-1	1
4	1	1	-1	1	-1	-1	-1
5	-1	-1	1	1	-1	-1	1
6	1	-1	1	-1	1	-1	-1
7	-1	1	1	-1	-1	1	-1
8	1	1	1	1	1	1	1

4.7 Planejamento Fatorial

Um planejamento fatorial é um planejamento experimental em que vários fatores (ou variáveis) são testados simultaneamente. O objetivo é determinar os efeitos individuais e combinados de cada fator em uma variável de resposta. Em termos simples, permite que você teste várias condições ao mesmo tempo, e veja como elas interagem umas com as outras para afetar o resultado final. Os fatores podem variar em diferentes níveis para observar seu efeito, e os resultados podem ser analisados para identificar quaisquer interações significativas existem entre os fatores.

4.7.1 Análise do Impacto da Função *Delay (latency)*

A Figura 4.9 mostra que, na comparação entre as Plataformas OpenFaaS e Fission, o efeito do fator Função é de 26,35% nos resultados. Em relação ao fator Concorrência, no caso do OpenWhisk e OpenFaaS chega a 20%, e no caso do OpenWhisk e Fission o fator Função atinge 81,35%. Considerando o cenário das funções *Delay* e *Matrix*, é possível concluir que, neste caso, o fator de Função tem o maior efeito sobre os resultados dos testes realizados.

Observando a Figura 4.10, em relação à função *Factors*, todas as três comparações entre as plataformas analisadas apresentam altos níveis de fração para o fator Função. O OpenWhisk com o Fission 76,14%, o OpenWhisk com OpenFaaS 72,73%, e o OpenWhisk com Fission 96,25%. Assim como na análise anterior, quando se correlaciona as funções

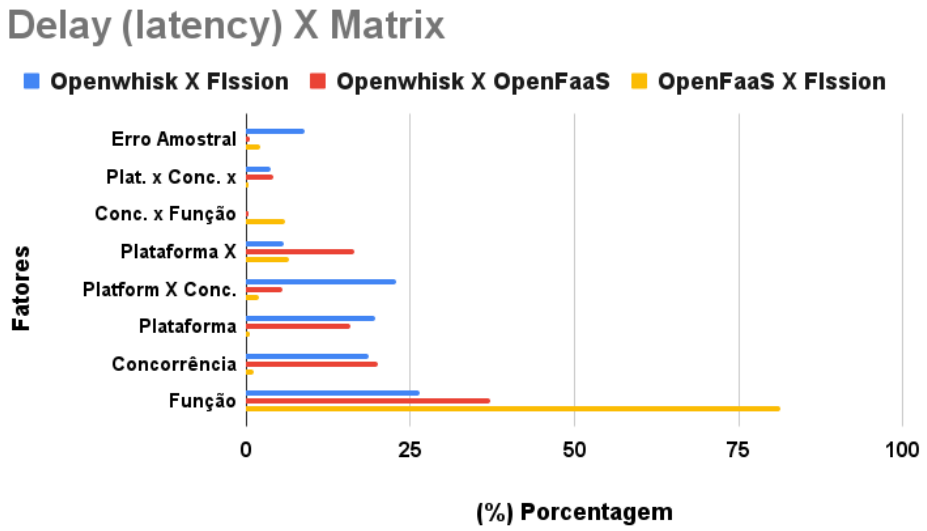


Figura 4.9: Projeto Fatorial *Delay (latency)* X Matrix.

Delay e Matrix, o fator Função tem a maior representatividade nos resultados, com certa distância para a segunda maior.

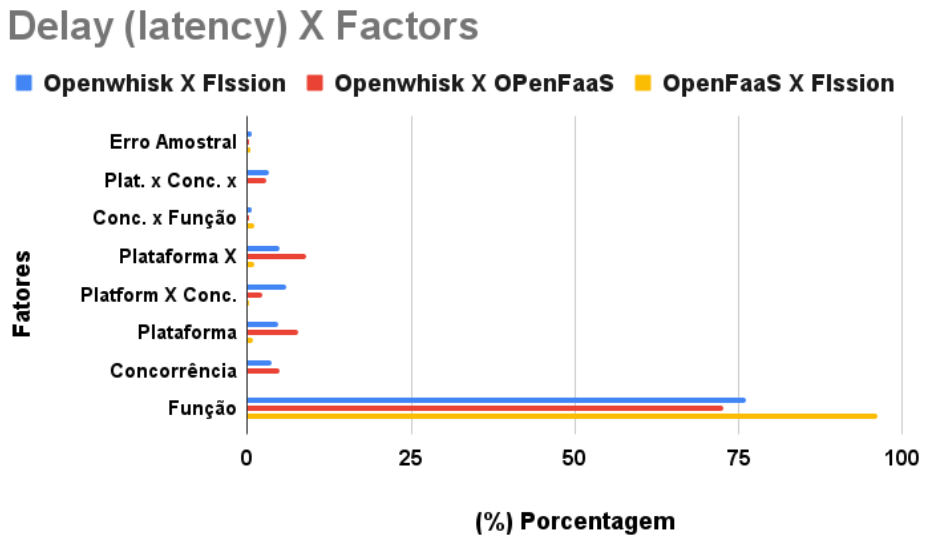


Figura 4.10: Projeto Fatorial *Delay (latency)* X Factors.

A correlação entre as funções *Delay (latency)* e Filesystem, calculadas pelo projeto fatorial, conforme demonstrado na Figura 4.11, mostra novamente que o fator Função foi predominantemente maior nesta fase da análise, com correlação de OpenFaaS e Fission obtendo a fração de 68,90%, OpenWhisk e OpenFaaS com 68,84%, seguidos pelo OpenWhisk e Fission com 93,18%.

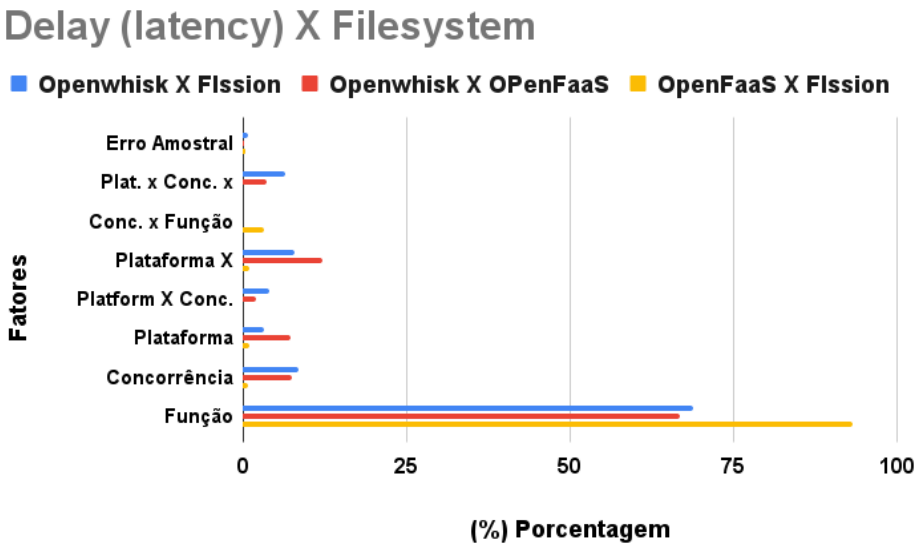


Figura 4.11: Projeto Fatorial *Delay (latency)* X Filesystem.

No fator Concorrência, as três plataformas também apresentam comportamento semelhante, com frações abaixo de 10%. Novamente, assim como nas duas análises anteriores, o fator Função tem maior representatividade nos resultados, nas três plataformas analisadas.

4.7.2 Análise do Impacto da Função Factors

Nesta fase, resultados representativos foram encontrados usando outras funções.

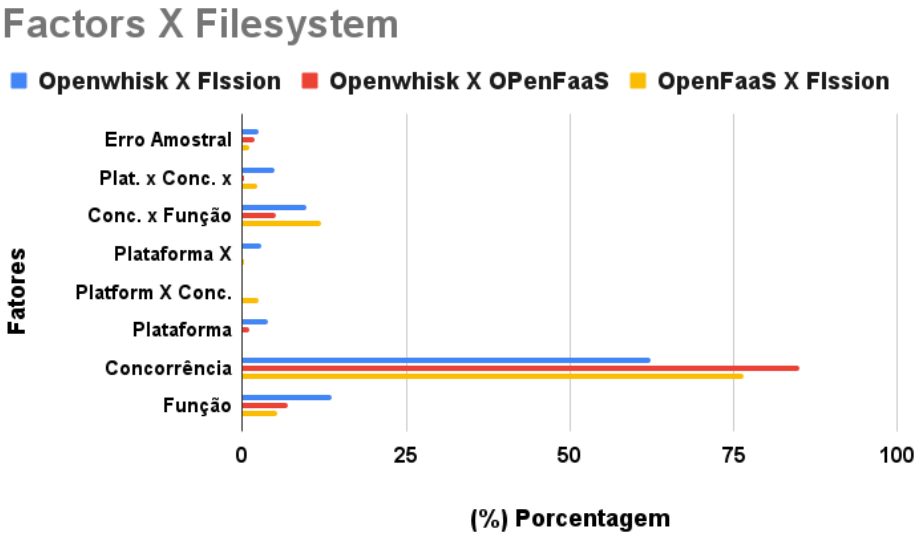


Figura 4.12: Projeto Fatorial Factors X Filesystem.

Ao analisar a correlação entre as funções Factors e FileSystem na Figura 4.12, o fator Função não é mais tão representativo como visto anteriormente, atingindo apenas a fração de 5,21% na correlação OpenFaaS e Fission, na OpenWhisk e OpenFaaS 6,84%, e na OpenWhisk e Fission 13,5%.

Nesse novo cenário é possível perceber a predominância do fator de Concorrência nas três plataformas, com OpenFaaS e Fission 62,37%, OpenWhisk e OpenFaaS 84,95%, e OpenWhisk e Fission 76,58%. Considerando a relação das funções analisadas em relação ao consumo de CPU e I/O, é possível notar que o fator Concorrência tem a fração predominante nos resultados das três plataformas correlacionadas.

4.7.3 Análise de Impacto da Função Matrix

Na análise da correlação entre as funções Matrix e Factors, o fator mais representativo foi novamente a Função. Na Figura 4.13 é possível ver as frações de 78,89% entre OpenWhisk com Fission, 66,66% entre OpenFaaS e Fission, e 80,77% entre OpenWhisk com OpenFaaS.

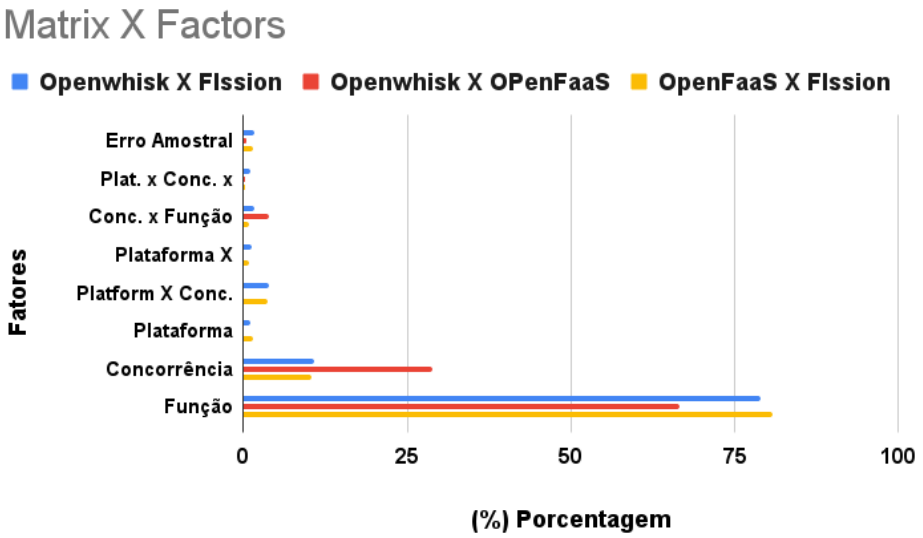


Figura 4.13: Projeto Fatorial Matrix X Factors.

Com relação ao fator Concorrência, em todos os três cenários analisados para as duas funções, nas três plataformas, seus valores de fração de fator variaram em média de 10% a 28,8%. Considerando a correlação das funções analisadas neste caso, os resultados novamente permitiram inferir que o fator função predomina na composição dos resultados.

Quanto à correlação das funções *Matrix* e *Filesystem*, obtiveram-se dois fatores com considerável representatividade. Os fatores Função e Concorrência, neste caso, possuem valores representativos. Conforme a Figura 4.14, o fator Função para a correlação OpenFaaS e Fission a fração de 63,95% influencia os resultados.

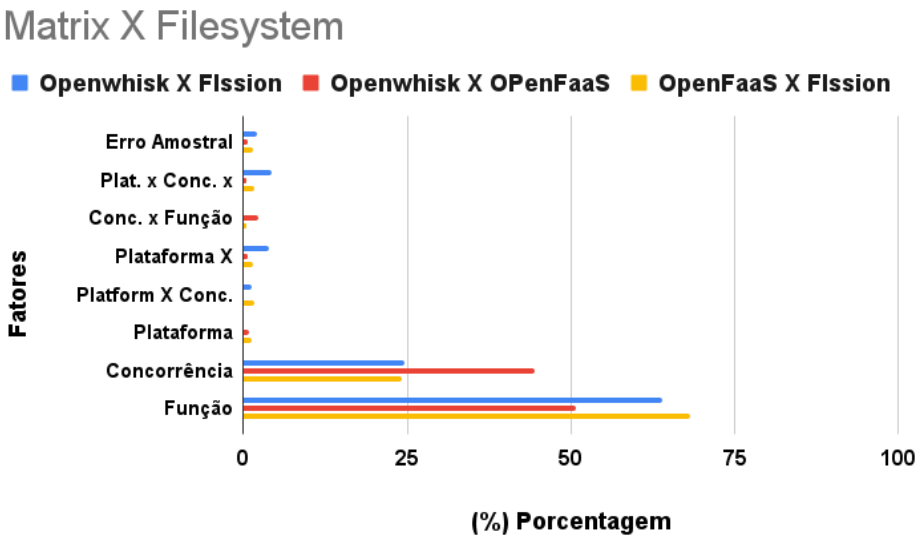


Figura 4.14: Projeto Fatorial Matrix X Filesystem.

Enquanto que para o fator de Concorrência tem a fração de 24,49%. No caso da correlação OpenWhisk e OpenFaaS, 50,71% para Função e 44,49% para Concorrência. Considerando OpenWhisk e Fission, 68,09% para Função e 24,18% para Concorrência. Considerando o erro amostral médio de apenas 1,19% nos 18 cenários calculados no experimento, é possível ter uma visão mais detalhada dos fatores que influenciam cada resultado.

Nos casos estudados, a influência dos fatores Função e Concorrência foi predominantemente representativa dos resultados de latência obtidos, por meio dos testes realizados nas funções de cada plataforma.

4.8 Análise de Confiabilidade

No teste de estresse realizado com a função Matrix, conforme apresentado na Figura 4.15, apenas o Openwhisk conseguiu atingir o limite de 16.384 requisições simultâneas antes de começar a falhar. Ele foi seguido pelo OpenFaaS com aproximadamente 1024 requisições, e o Fission, com 512 requisições. A Figura 4.15 mostra o grau de confiabilidade de cada plataforma, calculado com base no percentual de requisições atendidas.

É possível observar que até 128 requisições simultâneas, as três plataformas mantiveram o nível máximo de confiabilidade, atendendo 100% das requisições. Com cerca de 512 solicitações simultâneas, o OpenFaaS começou a sofrer uma queda gradual em sua taxa de confiabilidade e, nessa concorrência, o Fission também apresentou uma queda no percentual de sucesso. Embora o OpenFaaS tenha registrado algum nível de confiabilidade

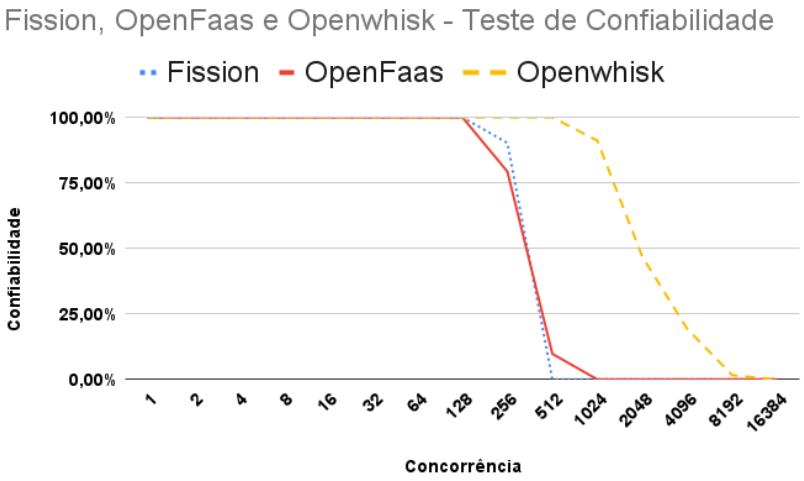


Figura 4.15: Análise de confiabilidade.

entre 512 e 1024 solicitações simultâneas, sua confiabilidade foi muito baixa e considerada irrelevante. Já o OpenWhisk demonstrou maior confiabilidade entre as três plataformas, apresentando erro em aproximadamente 10% das requisições somente, após uma carga maior do que 1024 requisições simultâneas.

Diante dos testes realizados nas plataformas selecionadas e, por meio das análises nas métricas obtidas pelos testes, foi possível a constatação de algumas conclusões que fomentam a estratégia de implementação do modelo de serviço na infraestrutura da DATAPREV. Primeiramente, a identificação da plataforma mais performática, no caso o OpenWhisk, fornecendo assim uma maior confiança na plataforma, considerando-se a expectativa de utilização em cargas de trabalho com alto nível de consumo de recursos.

Em seguida, diante da análise do planejamento fatorial, foi possível identificar que o tempo de resposta em todas as plataformas, na maioria dos casos, sofre variação influenciada pelo fator função, ou seja, o tempo de resposta das ativações das funções é diretamente impactado, dependendo do recurso de infraestrutura a ser consumido (consumo de rede, E/S de disco, uso de memória e de processador). Nesses testes foi possível também constatar que nas correlações entre as plataformas, na maioria dos casos, o OpenWhisk demonstrou sofrer menos influência pelo fator função nos testes estudados.

Nesse sentido, e considerando os benefícios e particularidades do modelo de serviço evidenciadas neste trabalho, a utilização do FaaS na nuvem privada da DATAPREV para operações de infraestrutura torna-se uma oportunidade viável, visando o incremento na eficiência relacionada ao consumo de recursos, bem como em uma maior resiliência dos serviços a serem implantados. Diante do exposto, o Capítulo 5 apresenta a solução proposta neste trabalho de pesquisa que é viabilizar o uso de FaaS no ambiente de uma grande empresa, como é o caso da DATAPREV.

Capítulo 5

Proposta de Uso do FaaS em Operações de Infraestrutura

Neste trabalho é proposto o uso de FaaS em uma plataforma de nuvem privada, para a execução de tarefas relacionadas às operações de infraestrutura que manipulam grande volume de documentos. Assim sendo, este capítulo descreve a definição da estratégia para o caso de uso a ser analisado, bem como a implantação da infraestrutura do *cluster* Kubernetes a ser utilizado para implantação modelo de serviço FaaS através da ferramenta OpenWhisk [4], e também da implementação do *benchmark* a ser utilizado para os experimentos, na nuvem privada da DATAPREV, descrevendo também as principais ferramentas usadas na solução.

5.1 Armazenamento de Documentos

Especificamente, para o caso da DATAPREV, é proposto o uso de FaaS para manipulação de arquivos (documentos) bem como sua leitura e gravação em uma de suas soluções de armazenamento (*storage* de objetos S3) [29], que é atualmente o protocolo mais comum utilizado para serviço de armazenamento da maioria dos provedores de nuvem. Assim sendo, o cenário escolhido para utilização do modelo de serviço FaaS, na infraestrutura de nuvem privada da DATAPREV, trata-se da utilização de funções pré-configuradas a serem executadas no serviço FaaS, para geração de documentos em *pdf*, bem como sua posterior gravação utilizando-se a solução de armazenamento (Dell ECS - S3) da empresa de forma a propiciar a análise de viabilidade de utilização desse modelo de serviço em sistemas de missão crítica relacionado a gestão de documentos.

A solução Dell ECS S3 é uma plataforma de armazenamento de objetos em nuvem altamente escalável e segura, que permite às organizações armazenar, gerenciar e acessar grandes quantidades de dados não estruturados, tais como imagens, vídeos, arquivos de

áudio e documentos [61]. O ECS S3 é baseado em hardware de servidor Dell EMC, software de armazenamento em nuvem, e tecnologia de gerenciamento de dados em escala de petabyte. Isso permite que os usuários armazenem e gerenciem dados em grande escala com facilidade. A solução é altamente escalável e pode ser dimensionada horizontalmente para lidar com cargas de trabalho em constante crescimento [62].

Além disso, a solução Dell ECS S3 fornece segurança de nível empresarial para dados armazenados na nuvem, incluindo recursos avançados de criptografia e autenticação de usuário. A solução é compatível com os padrões S3 da Amazon Web Services (AWS) [29], permitindo que os usuários implementem aplicativos que dependem do armazenamento em nuvem, sem a necessidade de alterar o código ou a infraestrutura existente. Em resumo, a solução Dell ECS S3 é uma plataforma de armazenamento em nuvem altamente escalável e segura, que oferece aos usuários a capacidade de gerenciar grandes quantidades de dados não-estruturados de maneira eficiente e econômica.

É esperado que diante das operações de escrita realizadas nesse serviço de armazenamento, conforme apresentado na Figura 5.1, por meio do uso de funções customizadas, o paralelismo e a confiabilidade providos pelo modelo de serviço FaaS demonstrem uma performance mais eficiente em relação ao modelo tradicional de operação desses arquivos.

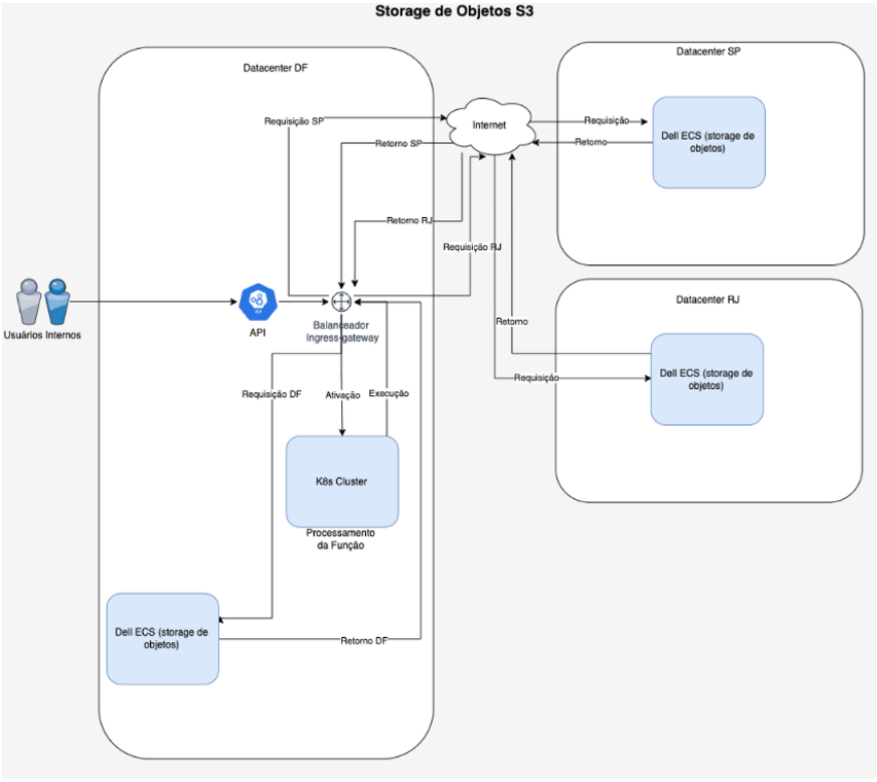


Figura 5.1: Arquitetura de alto nível - Interação FaaS com *storage* de objetos Dell ECS - S3.

Ainda conforme apresentado na Figura 5.1, é possível verificar que o serviço de armazenamento (*storage* de objetos - S3) encontra-se disponível em cada um dos três sites da empresa. Isso permite uma flexibilidade em sua utilização e também possibilita uma resiliência *multi-site*, quando utilizado com replicação dos dados. Entretanto, visando a não interferência de rede ao se trafegar pela internet, os experimentos foram realizados isoladamente no site do Distrito Federal, estando o *cluster* Kubernetes com a solução FaaS instalada no mesmo *datacenter* que o *storage* de objetos alvo da gravação dos arquivos.

5.2 Implementação do Cluster Kubernetes (K8S)

A arquitetura ideal para um *cluster* Kubernetes deve ser altamente disponível, escalável, segura e compatível com uma ampla variedade de cargas de trabalho. Assim, considerando as melhores práticas encontradas na documentação, e conforme apresentado na Figura 5.2, é possível verificar os componentes e tecnologias presentes na arquitetura do *cluster* alvo da implantação proposta nesta dissertação.

Para implementação do Kubernetes Cluster (K8S) a ser utilizado, por meio do modelo Infrastructure as a Service (IaaS), foram efetuados no ambiente de nuvem privada da empresa, o provisionamento de um balanceador virtual de cargas (NSX Load Balacer), e de 13 servidores virtuais com sistema operacional RedHat Enterprise Linux (RHEL) 8.3.

Conforme é possível verificar na Figura 5.2, um servidor foi utilizado como Bastião. Em seguida foram utilizados três servidores *Worker Ingress*, responsáveis por gerenciar o trafego de entrada e saída do *cluster*, três servidores *Master Nodes*, responsáveis por gerenciar as configurações e distribuição de carga no cluster, bem como 6 servidores *Worker Nodes* responsáveis pela efetiva execução da carga de trabalho.

A Tabela 5.1 apresenta os parâmetros relacionados a configuração do hardware em cada componentes do *cluster*, de acordo com a arquitetura apresentada na Figura 5.2. Diante do exposto, espera-se o provisionamento da plataforma OpenWhisk em um ambiente mais robusto e realista, de forma a propiciar a execução de testes mais acertivos e elegíveis para adoção por parte da empresa.

Tabela 5.1: *Cluster* Kubernetes alvo da implantação.

Componente	**Cores (vCPU)**	**Memória RAM (GB)**	**Quantidade de nodes**
Bastião	4	2	1
Master	4	16	3
Worker	8	32	6
Worker Ingress/gw	4	8	3

Para a facilitar a configuração de todas essas camadas, a ferramenta Rancher Kubernetes Engine (RKE) [63] foi utilizada. O Rancher Kubernetes Engine (RKE) é uma

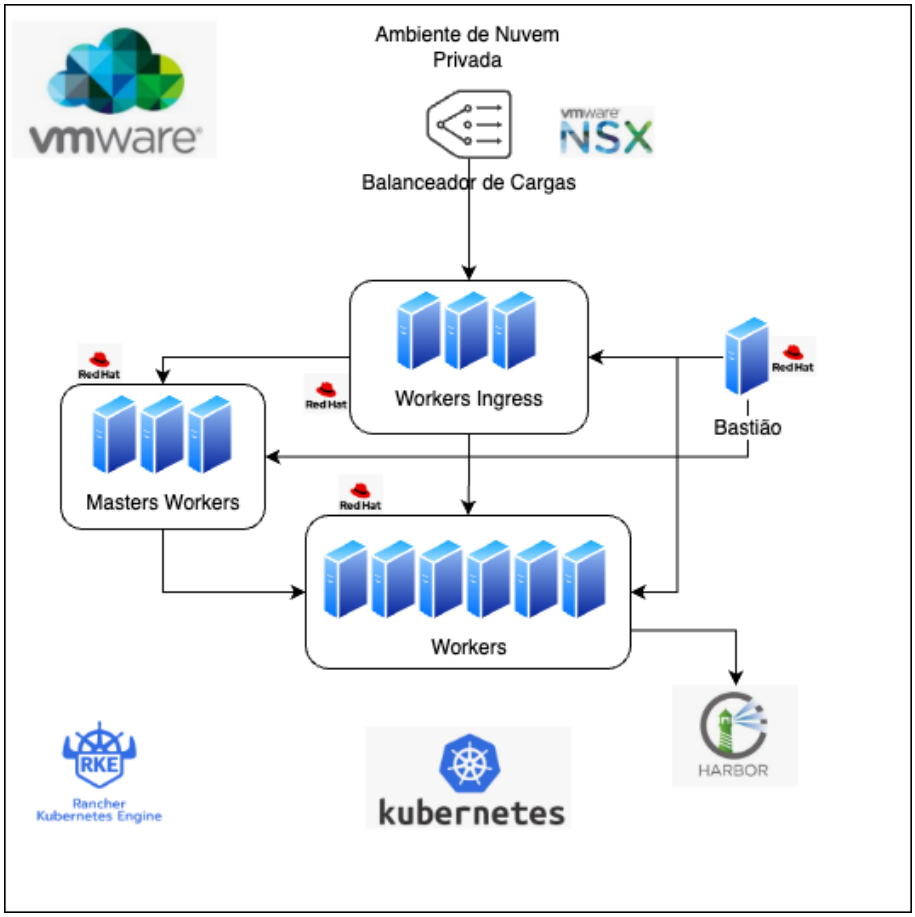


Figura 5.2: Arquitetura do Cluster Kubernetes.

ferramenta que simplifica significativamente o processo de instalação e gerenciamento de *clusters* Kubernetes. Ele oferece uma abordagem simplificada e modular para implantar e manter *clusters* Kubernetes em qualquer ambiente, seja em nuvem pública, privada ou híbrida. Com o RKE os usuários podem criar *clusters* Kubernetes altamente disponíveis e escaláveis com apenas alguns comandos, reduzindo drasticamente a complexidade e o tempo necessário para configurar um ambiente Kubernetes robusto. Além disso, o RKE é conhecido por sua confiabilidade e segurança, garantindo que os *clusters* implantados estejam configurados corretamente e protegidos contra possíveis vulnerabilidades, proporcionando assim maior tranquilidade aos administradores de sistemas.

A Tabela 5.2 apresenta as versões dos softwares utilizados no *cluster* alvo da implementação.

Tabela 5.2: Versões dos softwares da plataforma de *conteiners*.

Componente	**Versão**
rke (k8s)	1.20.4 (kubernetes)
Docker	20.10.5, build 55c4c88

5.2.1 Instalação Openwhisk

Para a correta instalação do OpenWhisk no *cluster* kubernetes implantado e descrito anteriormente, foram seguidas as instruções contidas na documentação oficial de seu projeto no github [1] - *OpenWhisk Deployment on Kubernetes* [64], do qual se baseia na utilização da ferramenta Helm [65]. Conforme pode ser verificado na Figura 5.3, no intuito

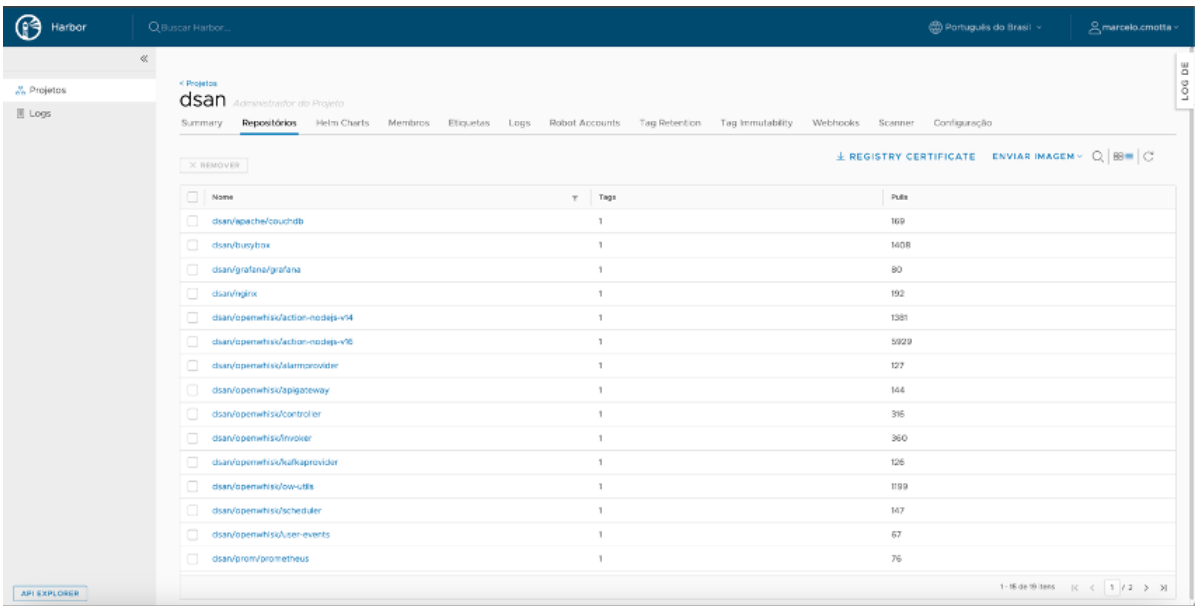


Figura 5.3: *Harbor - registry* interno.

de garantir não haver interferências de rede na questão do consumo das imagens, todas as 19 imagens docker necessárias para o correto funcionamento da solução Openwhisk, presentes em seu arquivo *values.yaml* no referido projeto [64], foram internalizadas para um servidor interno (*Harbor - registry*), de forma a servi-las localmente para o *cluster* Kubernetes.

O Helm [65] é uma ferramenta amplamente utilizada para simplificar o processo de implantação e gerenciamento de aplicativos em *clusters* Kubernetes. Ao usar o Helm para instalar o OpenWhisk, uma plataforma de computação sem servidor (*serverless*), os usuários podem aproveitar uma série de benefícios significativos. Primeiramente, o Helm oferece um mecanismo de empacotamento e distribuição de aplicativos chamados "*charts*", que contêm todos os recursos e dependências necessárias para implantar o OpenWhisk de maneira consistente e repetível. Isso simplifica o processo de instalação, pois os usuários podem implantar o OpenWhisk em seu *cluster* Kubernetes com apenas alguns comandos simples, economizando tempo e esforço.

[1] https://github.com/apache/openwhisk-deploy-kube

Além disso, o Helm permite que os usuários gerenciem facilmente o ciclo de vida do OpenWhisk, facilitando a atualização, o *rollback* e a remoção da instalação quando necessário. Em resumo, o uso do Helm para instalação do OpenWhisk em um *cluster* Kubernetes oferece uma maneira conveniente, confiável e eficiente de implantar e gerenciar aplicativos *serverless*, impulsionando a inovação e a produtividade das equipes de desenvolvimento.

Assim, conforme apresentado, nota-se que a arquitetura de implantação para o caso de uso escolhido neste trabalho explora diferentes características a serem consideradas no uso de FaaS. Isso é importante porque viabilizará um estudo completo das vantagens e limitações a serem consideradas no uso de FaaS integrado com aplicações que manipulam grandes volumes de documentos. A Tabela 5.3 apresenta as versões dos softwares utilizados para implementação do OpenWhisk.

Tabela 5.3: Versões dos softwares utilizados na instalação OpenWhisk.

Componente	**Versão**
Helm	v3.11.3
Openwhisk	20221014

5.2.2 Openwhisk - Fluxo de Ativação

Para descrever o fluxo de ativação de uma ação (função) no OpenWhisk, é necessário considerar quatro principais etapas. Primeiramente, o usuário envia uma solicitação para o OpenWhisk para executar uma ação. Essa solicitação pode ser feita por meio de uma API REST, uma interface de linha de comando, ou uma interface de usuário baseada em web. Na segunda etapa, a recepção é distribuída entre os controladores por meio do *proxy* Nginx. Na terceira, o controlador do OpenWhisk recebe a solicitação e verifica se o usuário tem permissão para executar a ação. Na quarta etapa, se a solicitação for válida, o controlador cria uma instância da ação em um ambiente de execução isolado. O ambiente de execução carrega a imagem da ação e as suas dependências, e executa o código da ação. Após a conclusão da execução da ação, o resultado é retornado ao usuário que fez a solicitação. O ambiente de execução é desativado e pode ser reutilizado para outras ações em um momento posterior, permitindo uma escalabilidade mais eficiente e uma redução de custos de infraestrutura.

Considerando a arquitetura de referência da soluçao FaaS Apache Openwhisk, conforme apresentado na Figura 5.4, é possivel verificar o fluxo de ativação das *actions* (funções) conforme descrito anteriormente.

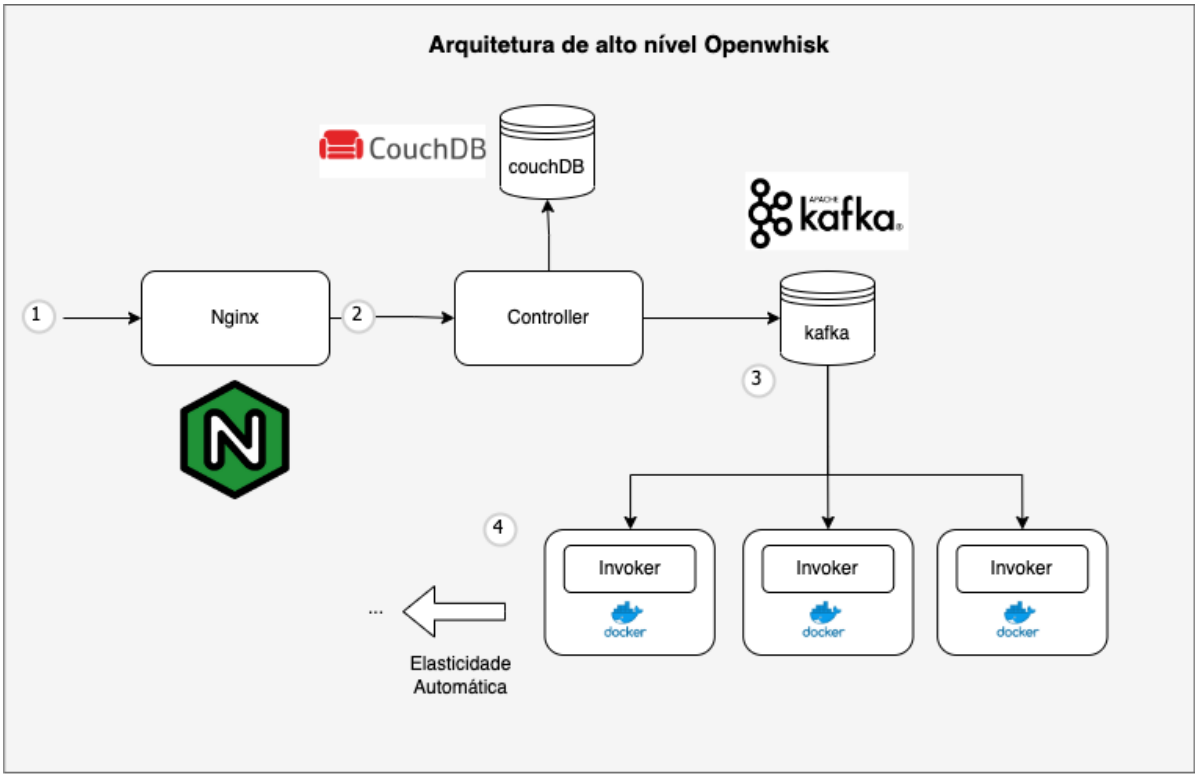


Figura 5.4: Arquitetura de alto nível Openwhisk, adaptado de [4].

Configurações de Ajuste (Tunning) do Openwhisk

De acordo com a documentação oficial do projeto *OpenWhisk Deployment on Kubernetes,* presente no *github* [2], a solução FaaS Apache OpenWhisk [44], fornece uma série de opções de configurações em diferentes cenários, sejam eles utilizando um *cluster* Kubernetes customizado em um ambiente de nuvem privada como o caso foco deste estudo, seja para utilização em outro serviço de nuvem pública, como por exemplo as soluções em Kubernetes do Google, o Google GKE [66], da Amazon, o Amazon EKS [67] e da IBM, o IBM Cloud Kubernetes Service (IKS) [68].

Visando a configuração ideal para usufruir-se da escalabilidade proposta pelo modelo de serviço, foram utilizados, de acordo com a documentação de referência, o incremento do recurso *replicaCount*, de forma a permitir a implantação de mais de um *Invoker*. Em adição ao recurso descrito acima, o parâmetro *whisk.containerPool.userMemory* também foi significativamente incrementado, visando a correspondência de utilização da memória disponível nos *Worker Nodes* dividida pelo número de *Invokers* disponibilizados pelo parâmetro *replicaCount*.

Assim sendo, considerando o fluxo de ativação das funções descrito, bem como o caso de uso pretendido, a proposta de utilização da solução FaaS nesses cenário necessitará

[2]https://github.com/apache/openwhisk-deploy-kube/blob/master/docs/configurationChoices.md

de aferição das métricas de desempenho e de custo relacionado ao consumo de recursos computacionais para efetiva comparação com o modelo tradicional. Para isso, o *framework* Orama [14] foi selecionado para viabilizar a aplicação dos testes de carga em cima da infraestrutura implantada de maneira semelhante aos estudos realizados anteriormente e descritos no Capítulo 4.

5.3 O *framework* Orama

O framework Orama[14] é uma ferramenta de suporte para avaliação de ambientes orientados a Função como Serviço. Ele auxilia nas tarefas de provisionamento e desprovisionamento de ambientes de casos de uso, configuração e execução de *benchmarks*, e análise dos resultados por meio de *design* fatorial e testes. Ele oferece uma abordagem abrangente e eficiente para a realização de testes em plataformas FaaS (Function-as-a-Service), trazendo uma série de benefícios significativos para desenvolvedores e equipes de qualidade. Ao adotar o Orama, os usuários podem automatizar a criação e execução de diversos tipos de testes, incluindo testes de unidade, integração e carga, de maneira rápida e precisa. Isso permite uma validação mais completa e confiável das funções implantadas na plataforma FaaS, garantindo que estejam funcionando conforme o esperado e atendendo aos requisitos de qualidade.

O *framework* Orama é composto pelos seguintes componentes rodando em Docker:

- *Frontend*: uma aplicação Node.Js + React;
- *Backend*: uma aplicação Node.js + Express + Swagger que recebe e processa solicitações de outros serviços;
- Banco de dados: uma instância autônoma do Postgres;
- *Orchestrator*: uma aplicação Node.js + Express + Swagger + Terraform que processa solicitações de provisionamento e desprovisionamento;
- *Benchmarker*: uma aplicação Node.js + Express + Swagger + JMeter que processa solicitações de benchmark e pode ser instalada no mesmo host onde os demais serviços estão instalados ou ser instalada isoladamente em máquinas removidas;
- Kafka: um gerenciador de mensagens Apache Kafka + Zookeeper que distribui os trabalhos de benchmarks para executores remotos.

No intuito de se viabilizar as análises comparativas dentre o modelo de serviço FaaS executando a mesma carga de trabalho em um ambiente tradicional com máquina virtual, dentro do ambiente de nuvem privada utilizado neste trabalho, foi providenciada também

a implantação do framework Orama visando sua utilização para viabilizar o estudo da performance diante do cenário proposto.

5.3.1 Implantação do Orama Framework

Sendo necessário o provisionamento de mais um servidor dedicado, foi providenciado outro servidor com sistema operacional RedHat Enterprise Linux na versão 8.3, com 4 CPUs e 4GB de memória, no intuito de não haver interferência de nenhuma carga externa aos testes a serem realizados. Os softwares necessários e suas versões podem ser conferidos na Tabela 5.4.

Tabela 5.4: Versões dos softwares para execução do *framework* Orama.

Componente	**Versão**
Docker Engine - Community	24.0.6
Docker Compose	1.29.2

Para a efetiva instalação do *framework* Orama [69], foi utilizado o método simplificado utilizando-se a ferramenta *docker compose*, seguindo-se as instruções presentes em sua documentação do projeto no github[3]. Diante de fácil customização de parâmetros no descritor, obteve-se sucesso em sua instalação e utilização no experimento.

Conforme pode ser verificado na Figura 5.5, a interface do *framework* Orama oferece uma série de facilidades que simplificam e otimizam o processo de teste em plataformas FaaS. Com uma interface intuitiva, criação simplificada de testes, execução flexível e suporte para CI/CD, o Orama capacita os desenvolvedores a realizar testes de maneira eficiente, garantindo a qualidade e confiabilidade de suas aplicações na nuvem.

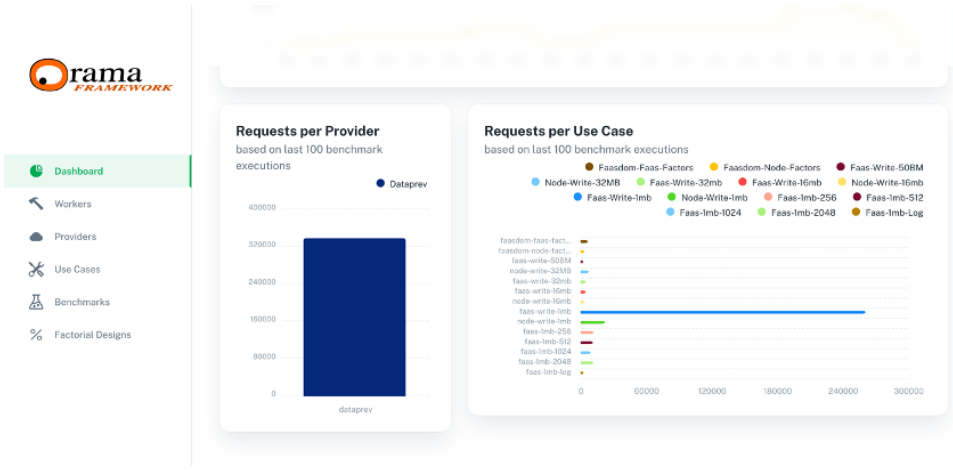


Figura 5.5: Interface do *framework* Orama.

[3]https://github.com/unb-faas/orama

5.3.2 Experimento Comparativo FaaS X Servidor Virtual Tradicional

No intuito de se aferir os benefícios ofertados pela tecnologia do modelo de serviço FaaS, para viabilizar o comparativo de cargas de trabalho originadas do Orama, mais um servidor foi providenciado, também com sistema operacional RedHat Enterprise Linux na versão 8.3, com 4 CPUs e 4GB de memória, com adição do pacote *nodejs* na versão 16.19.1-1, de forma a permitir a execução de códigos de forma semelhante a plataforma FaaS.

Considerando então o foco da presente pesquisa, foram desenvolvidas funções customizadas na liguagem NodeJS, tanto para sua execução pela plataforma OpenWhisk (com API nativa), quanto por parte do servidor tradicional (sem API nativa) com foco na geração e escrita de documentos de tamanhos variados no serviço de armazenamento de *storage* de Objetos dentro do ambiente de nuvem privada. As funções customizadas para esse propósito bem como os resultados obtidos neste experimento, podem ser conferidas no projeto do github[4].

Utilizando a interface do Orama, foram construídos dois casos de uso dentro do provedor DATAPREV que foi previamente configurado, respectivamente o caso de uso *s3-write-1mb-faas* e *s3-write-1mb-node* do qual respectivamente possibilitam a parametrização do tamanho do arquivo bem como apontam para as APIs do FaaS e do Servidor tradicional.

Conforme pode ser verificado através do exemplo na Figura 5.6, os casos de uso no Framework Orama são configurados de forma a registrar o endereço da API das plataformas alvo dos testes de carga.

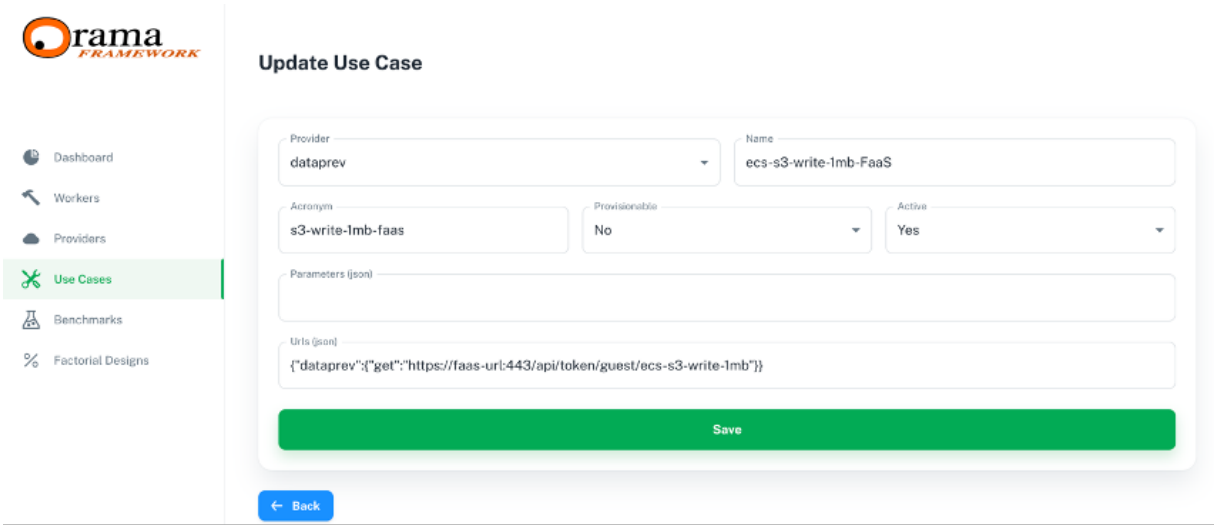


Figura 5.6: Caso de Uso Orama - s3-write-1mb-faas.

[4]https://github.com/marcelocmotta/benchmark

Foram então criados 2 casos de uso no *framework*, um apontando para a API da plataforma FaaS, bem como outro apontando para a API da máquina virtual tradicional, visando permitir o comparativo entre as duas plataformas através dos *benchmarks* a serem configurados.

Benchmarks

Após a devida configuração dos casos de uso no *framework* Orama, foram providenciados então os cenários para execução dos *benchmarks* através da ferramenta. Diante do objetivo proposto, pretende-se analisar as métricas obtidas por meio da execução de três distintos *benchmarks* em cada plataforma, variando-se o tamanho do arquivo gerado entre 1Mb, 16Mb e 32Mb. Conforme pode ser verificado no exemplo da Figura 5.7, por meio do método GET que aponta para API da plataforma FaaS, foram nesse caso configuradas as concorrências de 1 a 32, bem como o número de 10 repetições da carga de trabalho.

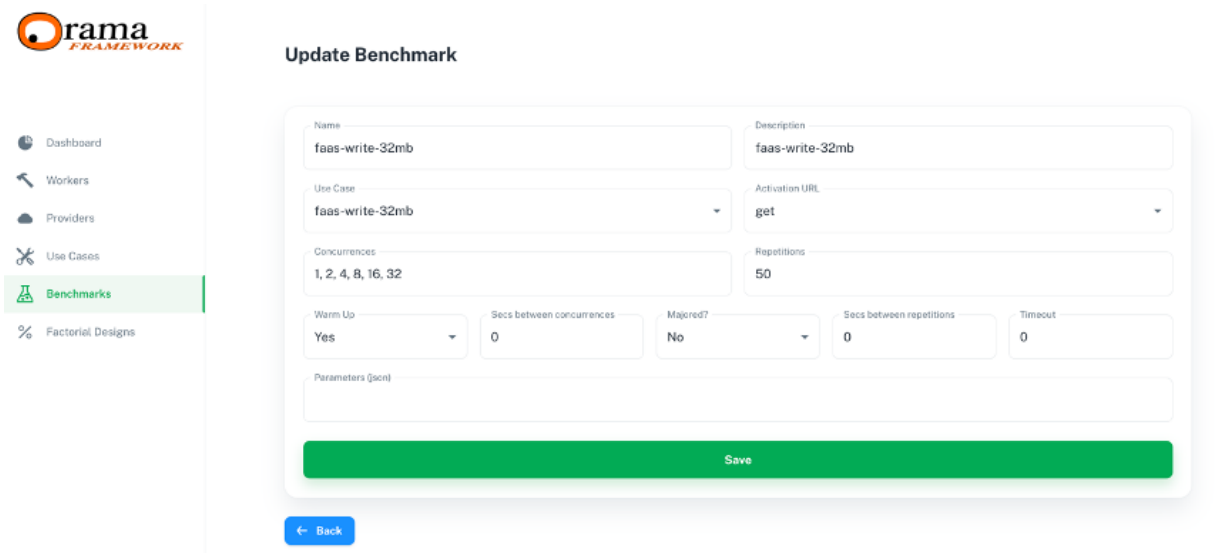


Figura 5.7: Exemplo *benchmark* 1 Orama - *faas-write-32mb*.

O mesmo processo foi repetido para criação dos outros *benchmarks*, variando o *endpoint* da API pretendida, bem como as concorrências e o número de repetições. O presente experimento pretende analisar as execuções dos *benchmarks* pré-configurados na ferramenta de forma crescente e exponencial, conforme apresentado na Figura 4.1.

Este processo permite ao *framework* Orama armazenar todas as métricas de latência previstas, proporcionando uma análise detalhada do comportamento dos sistemas testados. A abordagem exponencial visa testar os limites e a capacidade de escalabilidade dos provedores FaaS e dos servidores tradicionais. Para realizar um comparativo efetivo entre os provedores FaaS e os servidores tradicionais, diferentes cargas de trabalho das funções pré-configuradas serão utilizadas. Durante o tempo de execução, essas funções irão gerar

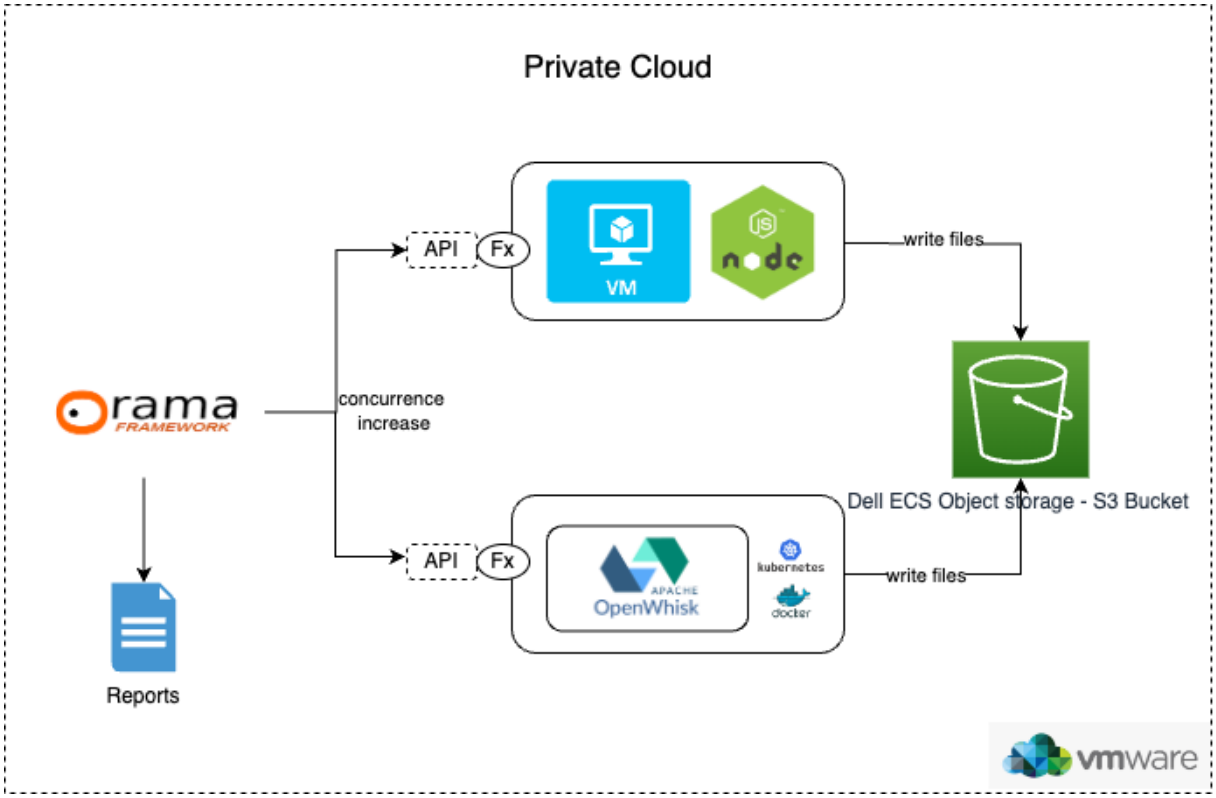


Figura 5.8: Arquitetura do experimento - FaaS - Servidor Tradicional - *framework* Orama.

arquivos que serão posteriormente gravados no *Storage* de Objetos. Este método garante que todas as variáveis relevantes sejam consideradas e que os dados coletados demonstrem fielmente o desempenho sob diferentes condições de carga.

Os resultados deste experimento são detalhados no Capitulo 6, onde são apresentados os dados coletados e as conclusões derivadas. A análise dos resultados permite uma compreensão clara das capacidades e limitações de cada provedor, fornecendo *insights* valiosos para a escolha da melhor solução de infraestrutura conforme as necessidades específicas.

Capítulo 6

Resultados

Diante da implementação da infraestrutura do *cluster* Kubernetes apresentada na Seção 5.2, da instalação da plataforma OpenWhisk na Seção 5.2.1, da inmplatação do *framework* Orama na Seção 5.3.1, e do desenvolvimento das funções customizadas em NodeJS para criação de documentos em tamanhos variados e escrita dos mesmos no *storage* de objetos, este capítulo apresenta, primeiramente as análises comparativas de latência referente aos resultados obtidos através da aplicação dos *benchmarks* criados no *framework* Orama, e em seguida, os resultados obtidos através das análises fatoriais são considerados.

6.1 Análise das Latências

Os fatores de baixo e alto nível para análise da latência considerados podem ser conferidos na Tabela 6.1, onde, utilizou-se o mesmo provedor DATAPREV (plataforma FaaS e servidor tradicional) com a concorrência mínima de 1 e máxima de 64 para os casos de arquivos de 1Mb e 16Mb e, para o caso de geração de arquivos de 32Mb, foi atingido o máximo de 32 requisições simultâneas, atingindo com concorrência alta os totais de 64Mb para o caso de arquivos de 1Mb, 1024 Mb para o caso de arquivos com 16Mb e também 1024 Mb para os arquivos de 32Mb.

Tabela 6.1: Fatores de baixo e alto nível para Análise fatorial.

Tamanho Arquivo	**Fator**	**Nível Baixo**	**Nível Alto**
1Mb	Provedor	dataprev	dataprev
1Mb	Concorrência	1	64
16Mb	Provedor	dataprev	dataprev
16Mb	Concorrência	1	64
32Mb	Provedor	dataprev	dataprev
32Mb	Concorrência	1	32

Nos *benchmarks* aplicados nas plataformas, onde, em cada caso, a ativação da função resulta na geração de um arquivo PDF de 1Mb, 16Mb e 32Mb com seu armazenamento em um *bucket* S3 remoto no *Storage* de Objetos, a carga de trabalho foi repetida 10 vezes em todos os casos de concorrência. Essa abordagem permite avaliar o desempenho da plataforma FaaS em comparação com a máquina virtual tradicional, diante de diferentes cargas de trabalho e níveis de concorrência. Ao iniciar com uma única ativação e aumentar gradualmente o número de ativações simultâneas, é possível observar como as plataformas respondem à escalabilidade e se mantém estável sob pressão.

Considerando o caso de uso de arquivos de 1Mb, e conforme apresentado no gráfico extraído diretamente da interface do Orama, representado pela Figura 6.1, é possível verificar que nos casos onde a concorrência é leve (1 e 2), ambas as plataforma mantém um comportamento similar relacionado a latência no tempo de resposta.

Em seguida, após o incremento das requisições simultâneas, diferentemente do servidor tradicional (na cor verde), que demonstra um crescimento exponencial em seu tempo de resposta, a plataforma FaaS (na cor azul) demonstra um desempenho melhor, distanciando-se da outra plataforma após o nível 4 de concorrência e mantendo-se constante até o nível de 32, com um leve aumento no caso de 64 requisições simultâneas.

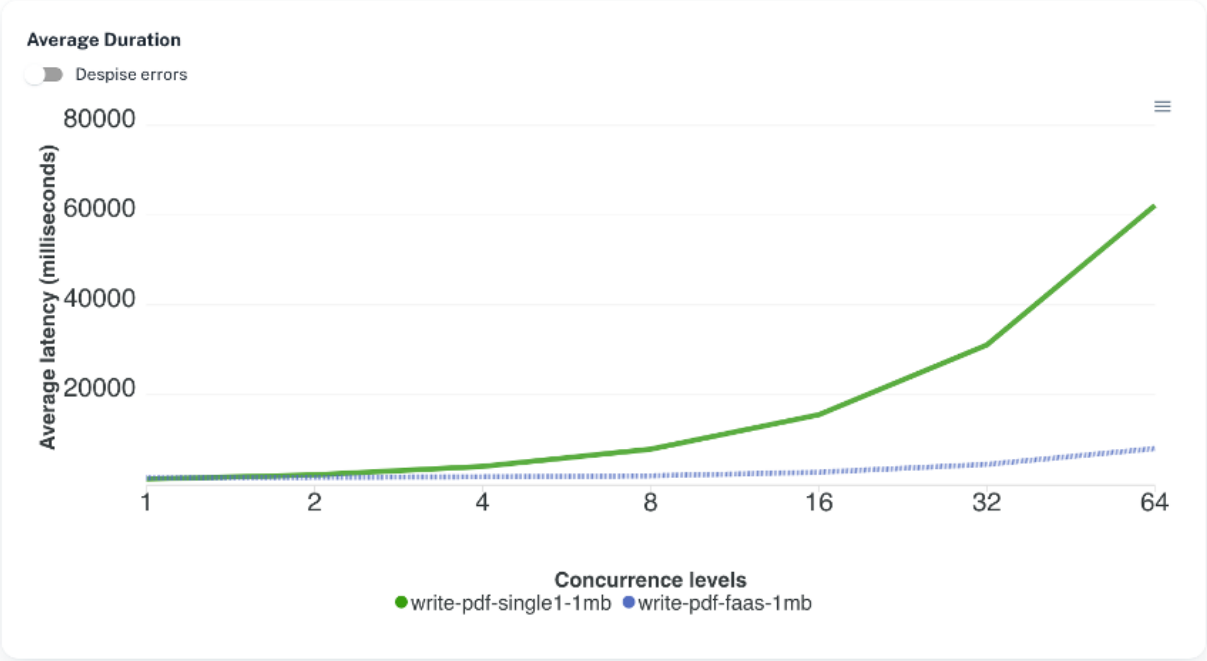


Figura 6.1: Análise de latência - arquivos PDF de 1Mb.

Esse comportamento evidencia que a plataforma FaaS, usufruindo do seu nativo recurso de escalabilidade, demonstrou maior eficiência que a arquitetura tradicional para lidar com a carga de trabalho escolhida para o caso dos arquivos de 1Mb.

A variedade de tamanho dos arquivos de documentos armazenados em um *bucket* S3 em um *Storage* de Objetos reflete a diversidade de tipos e usos desses documentos na prática cotidiana. Arquivos pequenos, como documentos de texto simples, planilhas ou apresentações em formato compactado, costumam variar de alguns kilobytes (KB) a poucos megabytes (MB). Esses arquivos normalmente são utilizados para tarefas administrativas diárias, relatórios e compartilhamento de informações básicas. Devido ao seu tamanho reduzido, são fáceis de armazenar e transferir, permitindo acesso rápido e eficiente. Por outro lado, arquivos maiores, exigem mais espaço de armazenamento e, frequentemente, maior largura de banda para transferência.

Considerando então a geração de arquivos com tamanho de 16Mb, é possível analisar, por meio da Figura 6.2, que nos casos onde a concorrência é leve (1 e 2), a plataforma FaaS (na cor azul) obteve um tempo de resposta levemente maior que o servidor tradicional (na cor verde) até o nível 2 de concorrência.

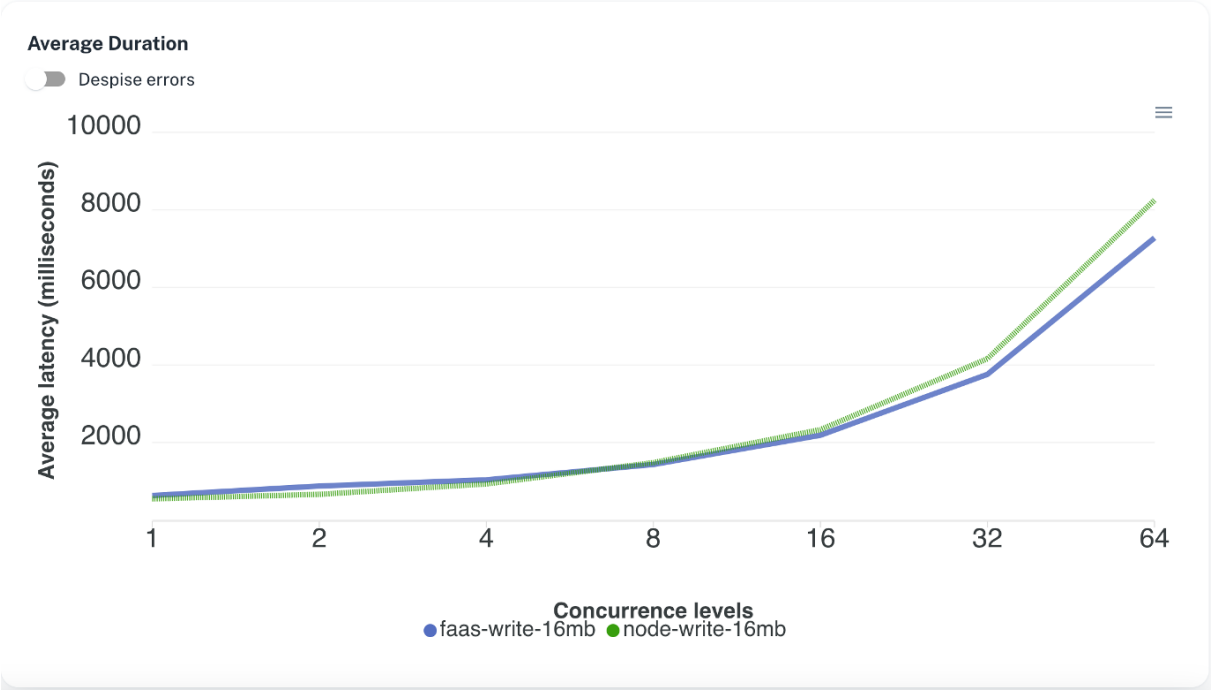


Figura 6.2: Análise de latência - arquivos PDF de 16Mb.

Em seguida, após o nível 3 de concorrência, as duas plataformas tiveram um comportamento bastante similar, mantendo-se constante sob o incremento de até 15 requisições simultâneas. Após o nível 16, a plataforma FaaS demonstra um tempo de resposta mais eficiente em relação ao servidor tradicional, mantendo-se na frente também nos casos onde a concorrência foi incrementada a 32 e a 64 requisições simultâneas.

No caso apresentado, utilizando arquivos de 16Mb, a plataforma FaaS demonstrou desempenho superior na maioria dos cenários de concorrência. Esse comportamento su-

gere que a plataforma FaaS é capaz de gerenciar a escalabilidade de maneira eficiente, alocando recursos adicionais conforme necessário para lidar com a carga de trabalho. A capacidade de escalar automaticamente permite que a plataforma mantenha um desempenho consistente mesmo quando a demanda aumenta, garantindo que os tempos de resposta e a eficiência geral não sejam comprometidos. Essa característica é particularmente importante em ambientes de alta concorrência, no qual a capacidade de resposta rápida é crucial. Apesar do impacto observado no desempenho devido ao processo de escalabilidade, a plataforma FaaS manteve um comportamento estável em cenários de concorrência mais elevada. Isso indica que, embora a escalabilidade possa introduzir uma sobrecarga inicial, a plataforma consegue se ajustar rapidamente, continuando a oferecer um desempenho robusto. No último caso de uso utilizado na análise, do qual se refere a geração e gravação de arquivos com 32Mb, conforme apresentado no gráfico representado pela Figura 6.3, é possível constatar que nos casos onde a concorrência é moderada (1 e 4), ambas as plataforma FaaS (na cor azul) e servidor tradicional (na cor verde) se mantém com comportamento equalizado, obtendo um tempo de resposta bastante similar nas duas plataformas.

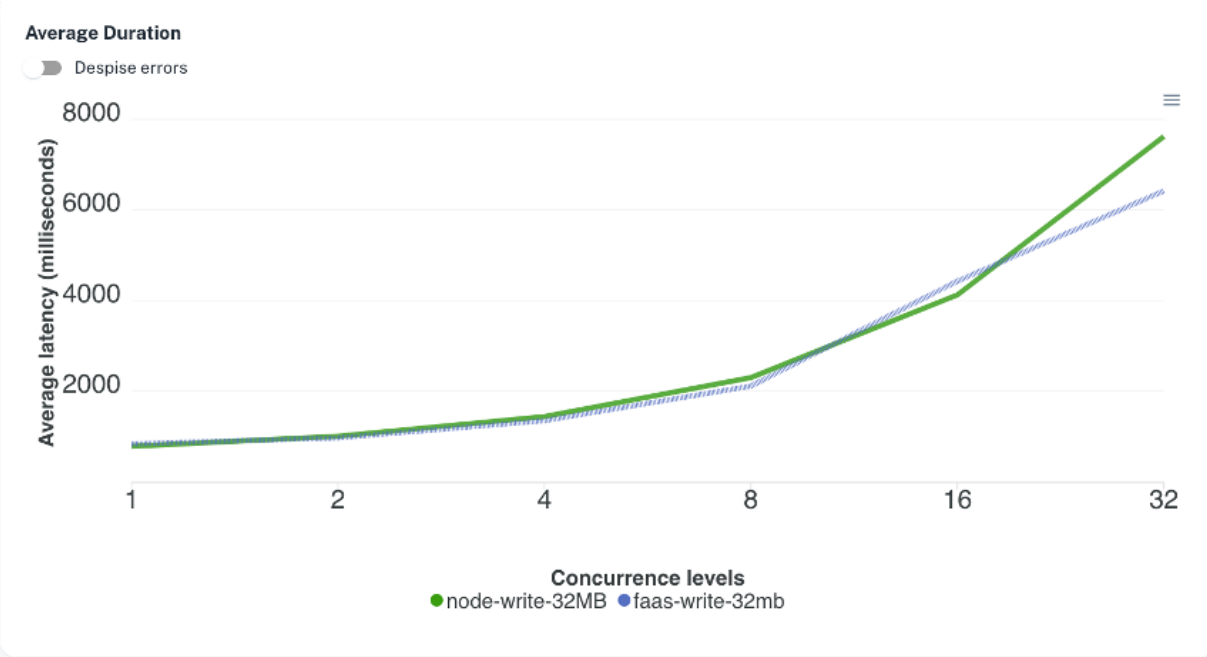


Figura 6.3: Análise de latência - arquivos PDF de 32Mb.

Após o incremento na concorrência, a partir do nível 4 até o nível 8, a plataforma FaaS demonstra um discreto melhor desempenho, obtendo resultados de latência menores que o servidor tradicional. Entre os níveis de concorrência 8 e 16, a plataforma FaaS sofre um relevante aumento em seu tempo de resposta, obtendo, nesse intervalo de concorrência, resultados piores em comparação com o servidor tradicional. Em seguida, ao se atingir

o nível 16, a plataforma FaaS, conforme nos casos anteriores, melhora consideravelmente seu tempo de resposta em relação ao servidor tradicional, novamente demonstrando um melhor desempenho após sua adaptação ao novo nível de carga recebido até o nível 32.

A análise do comportamento das plataformas fornece *insights* valiosos sobre a capacidade da plataforma FaaS de lidar com cargas de trabalho intensas e fornecer uma experiência consistente e confiável para os usuários finais, no caso de armazenamento de documentos.

6.2 Análise Fatorial

Explorando mais o *framework* Orama, em relação aos seus recursos de análise, conforme indicado na Figura 6.4, é possível perceber que no cenário dos arquivos de 32Mb, com 630 ativações em ambas as plataformas, a plataforma FaaS demonstra uma melhor média na latência, atingindo uma média de 4327,66 milissegundos contra 5138,76 milissegundos na máquina tradicional no registro de latências por repetição, já no registro de latências por concorrência, a plataforma FaaS também demonstra uma melhor média de 2457,28 milissegundos contra 2755,61 milissegundos registrados na máquina tradicional.

Conforme apresentado no Capítulo 4, em um planejamento fatorial, vários fatores são testados simultaneamente em um experimento. Isso permite a testagem de cenários simultaneamente e demonstra a interação de uns com as outros para afetar o resultado final. O objetivo é determinar os efeitos individuais e combinados de cada fator em uma variável de resposta.

Os resultados podem ser analisados para identificar quaisquer interações significativas entre os fatores, e os fatores podem variar em diferentes níveis para observar seu efeito. Em termos simples, um planejamento fatorial oferece uma abordagem eficaz para explorar as relações entre os fatores em um experimento. Assim, o planejamento definido no *framework* Orama para o *benchmark*, realizado com arquivos de 32Mb, pode ser verificado na Figura 6.5.

Ao se conduzir uma análise fatorial, os efeitos podem ser evidenciados de diversas maneiras, influenciando diretamente os resultados obtidos. No caso em questão, os efeitos calculados pela ferramenta são apresentados na Figura 6.6.

Em uma análise fatorial, as frações são fundamentais para compreender a contribuição de cada parte em relação ao todo. Essa abordagem permite uma compreensão mais profunda das interações entre as diferentes partes do sistema e como elas contribuem para o resultado geral, a Figura 6.7 demonstra os resultados obtidos no comparativo dos dois benchmarks aplicados, evidenciando 1,03% de influência em relação ao provedor, 97,33%

Figura 6.4: Análise de latência - arquivos PDF de 32Mb.

Plan

Test	I	Factor A	Factor B	Factor AxB	Y	Avg of Y	SSY (in line)
0	1	1	1	1	6484.78125, 6062.84375, 6315.71875, 5967.1875, 5689.5, 6116.8125, 5876.5625, 6061.34375, 5996.375, 5935.3125	6050.6438	447322.1879
1	1	-1	1	-1	7627.5, 7457.34375, 7379.5, 7557.875, 7539.4375, 7090.875, 7350.6875, 7224.125, 7314.6875, 7082.34375	7362.4375	322828.4434
2	1	1	-1	-1	895, 843, 853, 890, 804, 929, 786, 789, 1004, 958	875.1000	48936.9000
3	1	-1	-1	1	741, 788, 788, 741, 822, 764, 757, 704, 820, 798	772.3000	12826.1000

Figura 6.5: Planejamento fatorial arquivos PDF de 32Mb.

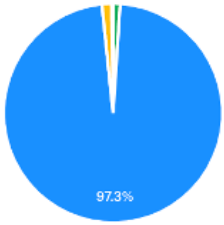
em relação a concorrência, 1,31% considerando o Provedor X Concorrência, atingindo 0,23% de taxa de erro.

Effects

	i	Provider	Concurrence	Provider x Concurrence
Values	3765.1203	-302.2484	2941.4203	-353.6484
Confidence interval (99.95)%	(3679.02, 3851.22)	(-388.35, -216.15)	(2855.32, 3027.52)	(-439.75, -267.55)
Confidence interval (97.5)%	(3716.37, 3813.87)	(-351.00, -253.50)	(2892.67, 2990.17)	(-402.40, -304.90)
Confidence interval (95)%	(3724.54, 3805.70)	(-342.83, -261.67)	(2900.84, 2982.00)	(-394.23, -313.07)
Confidence interval (90)%	(3733.74, 3796.50)	(-333.63, -270.87)	(2910.04, 2972.80)	(-385.03, -322.27)
Confidence interval (80)%	(3744.65, 3785.59)	(-322.72, -281.78)	(2920.95, 2961.89)	(-374.12, -333.18)
Confidence interval (70)%	(3752.40, 3777.84)	(-314.97, -289.53)	(2928.70, 2954.14)	(-366.37, -340.93)
Confidence interval (60)%	(3758.99, 3771.25)	(-308.38, -296.11)	(2935.29, 2947.55)	(-359.78, -347.51)

Figura 6.6: Efeitos - arquivos PDF de 32Mb.

Fractions



Provider Concurrence Provider X Concurrence Error

Provider	Concurrence	Provider x Concurrence	Error
1.03%	97.33%	1.41%	0.23%

Figura 6.7: Frações - arquivos Pdf de 32Mb.

Os testes desempenham um papel fundamental na análise fatorial, fornecendo uma estrutura sólida para investigar as relações complexas entre as variáveis. Eles permitem que pesquisadores e analistas identifiquem padrões subjacentes nos dados, ajudando a entender como diferentes fatores se interrelacionam e contribuem para o fenômeno em estudo. Por meio de testes rigorosos, é possível determinar a relevância e a força das correlações entre variáveis, oferecendo uma visão detalhada das dinâmicas envolvidas. Esta abordagem sistemática não apenas simplifica a complexidade dos dados, mas também ajuda a revelar fatores latentes que podem não ser imediatamente aparentes, proporcionando uma

compreensão mais profunda e completa do objeto de estudo.

A importância dos testes na análise fatorial vai além da mera identificação de relações; eles são cruciais para a validação das hipóteses e modelos teóricos. Através da aplicação de testes estatísticos, é possível avaliar a significância dos resultados obtidos, garantindo que as conclusões sejam baseadas em evidências robustas e não em coincidências ou padrões espúrios. Isso é vital para a credibilidade da pesquisa, pois assegura que os *insights* derivados sejam confiáveis e replicáveis em diferentes contextos. Além disso, os testes permitem ajustar e refinar os modelos, melhorando continuamente a precisão das análises.

Por fim, conforme apresentado na Figura 6.8, os resultados obtidos a partir dos testes realizados, utilizando o referido *benchmark* nas duas plataformas analisadas, demonstram que foi atingido um intervalo de confiança de 99,95% nas amostras.

Tests

Test Name	Difference	Difference Standard Deviation	Effective DoF
t	811.10	47.50	14.66

Confidence	Confidence Interval of the Difference	Result
99.95%	(616.60 , 1005.61)	✓
97.50%	(709.65 , 912.56)	✓
95.00%	(727.70 , 894.51)	✓
90.00%	(747.35 , 874.85)	✓
80.00%	(769.93 , 852.28)	✓
70.00%	(785.64 , 836.57)	✓
60.00%	(798.85 , 823.36)	✓

Benchmark	Count	Mean	Sum of Values	Sum of Squared Values	Sum of Squared Errors	Standard Deviation
node-write-32MB	10	5138.76	51387.62	264149550.62	80811.49	89.90
faas-write-32mb	10	4327.66	43276.57	187430998.07	144834.61	120.35

Figura 6.8: Testes arquivos Pdf de 32Mb.

Esse elevado nível de confiança indica que os dados coletados são altamente precisos e confiáveis, refletindo com grande certeza o desempenho real das plataformas em condições de teste específicas. A alta confiança nos resultados é fundamental para validar a eficácia e a eficiência das plataformas analisadas, assegurando que as conclusões tiradas a partir do estudo sejam robustas e replicáveis.

O intervalo de confiança de 99,95% significa que há apenas uma pequena margem de erro nos resultados, permitindo que os *stakeholders* confiem plenamente nas recomendações feitas com base nesses testes. Além disso, a metodologia rigorosa empregada para atingir esse nível de confiança pode servir como um padrão para futuros estudos comparativos, garantindo que os resultados sejam tanto precisos quanto aplicáveis a uma ampla gama de cenários operacionais.

Capítulo 7

Conclusão

O modelo de serviço do tipo Function-as-a-service (FaaS) mostrou ser uma solução de serviço de nuvem aderente às necessidades atuais para a utilização de *storage* de objetos, como solução de armazenamento de documentos. Porém, as dificuldades de estruturação e consumo desse modelo ameaçam sua adesão devido sua complexidade de implementação. Os resultados dos experimentos realizados neste trabalho mostraram que esse modelo confere, na maioria das situações abordadas, resultados positivos para sua utilização como solução para otimização de operações de infraestrutura relacionada ao armazenamento de documentos utilizando-se como alvo um *storage* de objetos. Dessa forma, maximizar a eficiência da utilização desse serviço no contexto de nuvem privada é uma necessidade e um desafio constante.

Os resultados acadêmicos alcançados diante deste trabalho, primeiramente, no quesito relacionado a ferramentas de armazenamento em nuvem, geraram a publicação do artigo *Performance Analysis of Main Public Cloud Big Data Services Processing Brazilian Government Data* [13], apresentado no evento *7th Latin American High Performance Computing Conference, CARLA 2020*, em Cuenca, Equador, atualmente com estrato CAPES na classificação B3.

Em seguida, diante do aprofundamento da pesquisa, já dentro do escopo referente a análise das ferramentas FaaS de código livre, foi conquistada também, a publicação do artigo Análise de Desempenho de Funções como Serviço em Nuvem Privada - DATAPREV [12], no evento ERAD-CO 2021 - Anais da IV Escola Regional de Alto Desempenho do Centro-Oeste, onde foi possível, através da utilização da análise fatorial a identificação dos fatores que mais geravam interferência nos resultados de desempenho na execução de funções dentro do ambiente de nuvem privada.

Após isso, ao se adequar e incrementar a infraestrutura de testes e se aprofundar nos resultados comparativos focados na identificação da plataforma FaaS mais resiliente e com melhor desempenho, foi conquistada a publicação do artigo *Comparison of FaaS Platform*

Performance in Private Clouds [11], apresentado no evento *12th International Conference on Cloud Computing and Services Science CLOSER 2022*, que atualmente possui estrato CAPES com classificação A3.

Essa fase do estudo permitiu a seleção da ferramenta a ser utilizada em uma implementação mais realista e condizente com uma situação relacionada as operações de infraestrutura em ambiente de nuvem privada. A partir desse ponto, foi implementado um *cluster* Kubernetes na infraestrutura de nuvem privada da Empresa de Tecnologia e Informações da Previdência (DATAPREV), seguindo as melhores práticas de arquitetura. Em seguida, a ferramenta FaaS selecionada, OpenWhisk, foi configurada de acordo com as instruções da documentação para obter a melhor performance possível. Além disso, foram utilizados um repositório de imagens *Docker* (*registry*) e um armazenamento de objetos com protocolo S3, ambos internos.

Nesse cenário foi utilizado o *framework* Orama [14], de forma a possibilitar a aplicação dos *benchmarks* que forneceram cenários propícios ao incentivo da adoção desse modelo de serviço para utilização em aplicações de ambiente produtivo, que utilizam armazenamento de documentos.

Diante do exposto, nota-se que a abordagem proposta neste trabalho pode conferir a aplicações que utilizam armazenamento de documentos hospedadas em ambiente de nuvem privada, significativos ganhos do tempo de processamento se comparados ao tempo de processamento em ambiente tradicional. Todavia, conforme os resultados apresentados, as configurações da plataforma FaaS necessitam ser aprimoradas e corretamente dimensionadas para a carga de trabalho a ser submetida, para de fato oferecer uma maior eficiência na utilização dos recursos computacionais.

Portanto, em trabalhos futuros essa perspectiva pode ser incorporada, por exemplo, a um sistema de gestão de documentos, de modo a oferecer uma maior confiabilidade, desempenho e otimização de recursos computacionais, bem como todos os benefícios que o modelo de serviço FaaS proporciona.

Referências

[1] MELL, Peter e Tim Grance: *The NIST definition of cloud computing*. National Institute of Standards and Tecnology, September 2011. x, 5, 6, 8, 9, 10, 11

[2] Carvalho, L. e Aletéia P. F. Araújo: *Framework node2faas: Automatic nodejs application converter for function as a service*. Em *Proceedings of the 9th International Conference on Cloud Computing and Services Science - Volume 1: CLOSER,*, páginas 271–278. INSTICC, SciTePress, 2019, ISBN 978-989-758-365-0. x, 1, 8, 13

[3] SPOIALA, Cristian: *Pros and cons of serverless computing.*, 2017. `https://assist-software.net/blog/pros-and-cons-serverless-computing-faas-/comparison-aws-lambda-vs-azure-functions-vs-google`, acesso em 13/10/2017. x, 1, 11, 14

[4] Foundation, Apache: *Apache openwhisk documentation*, 2023. `https://openwhisk.apache.org/documentation.html`, acesso em 2023-03-07. x, 19, 42, 48

[5] Amazon Web Services: *AWS lambda*, 2021. `https://aws.amazon.com/pt/lambda`. 1, 16

[6] Schleier-Smith, Johann, Vikram Sreekanti, Anurag Khandelwal, Joao Carreira, Neeraja J. Yadwadkar, Raluca Ada Popa, Joseph E. Gonzalez, Ion Stoica e David A. Patterson: *What serverless computing is and should become: The next phase of cloud computing*. Commun. ACM, 64(5):76–84, abril 2021, ISSN 0001-0782. `https://doi.org/10.1145/3406011`. 1, 11, 13

[7] Google: *Cloud functions*, 2021. `https://cloud.google.com/functions/`, acesso em 2021/08/10. 1, 17

[8] Microsoft: *Azure functions*, 2021. `https://azure.microsoft.com/pt-br/services/functions/`, acesso em 2021/08/10. 1, 16, 17

[9] García López, Pedro, Marc Sánchez-Artigas, Gerard París, Daniel Barcelona Pons, Alvaro Ruiz Ollobarren e David Arroyo Pinto: *Comparison of faas orchestration systems*. Em *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*, páginas 148–153, 2018. 1

[10] Malawski, Maciej, Adam Gajek, Adam Zima, Bartosz Balis e Kamil Figiela: *Serverless execution of scientific workflows: Experiments with hyperflow, aws lambda and google cloud functions*. Future Generation Computer Systems, 110:502–514, 2020, ISSN 0167-739X. `https://www.sciencedirect.com/science/article/pii/S0167739X1730047X`. 1

[11] Motta., Marcelo Augusto Da Cruz, Leonardo Reboucas De Carvalho., Michel Junio Ferreira Rosa. e Aleteia Patricia Favacho De Araujo.: *Comparison of faas platform performance in private clouds.* Em *Proceedings of the 12th International Conference on Cloud Computing and Services Science - CLOSER,*, páginas 109–120. INSTICC, SciTePress, 2022, ISBN 978-989-758-570-8. 2, 25, 64

[12] Motta, Marcelo, Leonardo Carvalho e Aleteia Araujo: *Análise de desempenho de funções como serviço em nuvem privada - dataprev.* Em *Anais da IV Escola Regional de Alto Desempenho do Centro-Oeste*, páginas 17–21, Porto Alegre, RS, Brasil, 2021. SBC. `https://sol.sbc.org.br/index.php/eradco/article/view/18418`. 3, 63

[13] Carvalho, Leonardo Rebouças de, Marcelo Augusto da Cruz Motta e Aleteia Patricia Favacho de Araújo: *Performance analysis of main public cloud big data services processing brazilian government data.* Em Nesmachnow, Sergio, Harold Castro e Andrei Tchernykh (editores): *High Performance Computing*, páginas 49–61, Cham, 2021. Springer International Publishing, ISBN 978-3-030-68035-0. 3, 63

[14] Carvalho, Leonardo e Aleteia Araujo: *Orama: A benchmark framework for function-as-a-service.* Em *Proceedings of the 12th International Conference on Cloud Computing and Services Science - Volume 1: CLOSER,*, páginas 313–322. INSTICC, SciTePress, 2022, ISBN 978-989-758-570-8. 3, 49, 64

[15] Huane, L.: *Big data drives cloud adoption in enterprise.* IEEE Internet Comput, 2013. 5

[16] Herbst, Nikolas, Samuel Kounev e Ralf Reussner: *Elasticity in cloud computing: What it is, and what it is not.* junho 2013. 7, 13

[17] Celesti, Antonio, Francesco Tusa, Massimo Villari e Antonio Puliafito: *How to enhance cloud architectures to enable cross-federation.* páginas 337–345, julho 2010. 9

[18] ZHENG, Xi: *Database as a service - current issues and its future.* CoRR, abs/1804.00465, 2018. `http://arxiv.org/abs/1804.00465`, acesso em 05/07/2020. 11

[19] CITRIX: *Deliver virtual windows apps and desktops simply and securely*, 2019. `https://www.citrix.com/products/citrix-managed-desktops.html`, acesso em 26/05/2019. 11

[20] ABE, John Olorunfemi e Burak BERK: *A data as a service (daas) model for gpu-based data analytics.* Em *IEEE/IFIP NTMS Workshop on Big Data and Emerging Trends.* IEEE/IFIP, 2017. 11

[21] Meng, Shicong e Ling Liu: *Monitoring-as-a-service in the cloud: Spec phd award (invited abstract).* Em *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, ICPE '13, página 373–374, New York, NY, USA, 2013. Association for Computing Machinery, ISBN 9781450316361. `https://doi.org/10.1145/2479871.2479929`. 11

[22] DELL: *Storage as a service.*, 2019. https://www.dellemc.com/pt-br/glossary/storage-as-a-service.htm, acesso em 26/05/2019. 11

[23] DUAN, Y., G. Fu, N. Zhou, X. Sun, N. C. Narendra e B. Hu: *Everything as a service (xaas) on the cloud: Origins, current and future trends.* Em *2015 IEEE 8th International Conference on Cloud Computing (CLOUD)*, volume 00, páginas 621–628, Junho 2015. doi.ieeecomputersociety.org/10.1109/CLOUD.2015.88, acesso em 05/07/2020. 11

[24] Institute, Uptime: *Dataprev: Data center distrito federal conquista certificação tier iii facility - datacenterdynamics.com.br*, 2019. https://uptimeinstitute.com/blog/26-news-external/, acesso em 2023-03-07. 12

[25] CHAPIN, John e Michael Roberts: *What is Serverless.* Oreilly, Junho 2017, ISBN 9781491984161. 13

[26] Castro, Paul, Vatche Ishakian, Vinod Muthusamy e Aleksander Slominski: *The rise of serverless computing.* Commun. ACM, 62(12):44–54, novembro 2019, ISSN 0001-0782. https://doi.org/10.1145/3368454. 13

[27] Grambow, Martin, Tobias Pfandzelter, Luk Burchard, Carsten Schubert, Max Zhao e David Bermbach: *Befaas: An application-centric benchmarking framework for faas platforms*, 2021. 14

[28] Amazon: *Firecracker*, 2021. https://firecracker-microvm.github.io/, acesso em 2021/08/10. 16

[29] Amazon Web Services, Inc. .: *What is amazon s3?*, 2023. https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html, acesso em 2023-03-07. 16, 42, 43

[30] Services, Amazon Web: *Amazon dynamodb*, 2023. https://aws.amazon.com/pt/dynamodb/, acesso em 2023-03-07. 16

[31] Services, Amazon Web: *Amazon api gateway*, 2023. https://aws.amazon.com/pt/api-gateway/, acesso em 2023-03-08. 16

[32] Corporation, Microsoft: *Armazenamento em nuvem seguro, de alto desempenho, confiável e escalonável*, 2023. https://azure.microsoft.com/pt-br/products/category/storage, acesso em 2023-03-09. 17

[33] Corporation, Microsoft: *What is azure event grid?*, 2023. https://learn.microsoft.com/en-us/azure/event-grid/overview, acesso em 2023-03-09. 17

[34] Corporation, Microsoft: *Barramento de serviço*, 2023. https://azure.microsoft.com/pt-br/products/service-bus, acesso em 2023-03-09. 17

[35] Verma, Abhishek, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune e John Wilkes: *Large-scale cluster management at google with borg.* Em *Proceedings of the Tenth European Conference on Computer Systems*, EuroSys '15, New York, NY, USA, 2015. Association for Computing Machinery, ISBN 9781450332385. https://doi.org/10.1145/2741948.2741964. 17

[36] Burns, Brendan, Brian Grant, David Oppenheimer, Eric Brewer e John Wilkes: *Borg, omega, and kubernetes*. Communications of the ACM, 59(5):50–57, 2016. 17

[37] Cloud, Google: *Armazenamento de objetos para empresas de todos os tamanhos.*, 2023. https://cloud.google.com/storage?hl=pt-br, acesso em 2023-03-09. 17

[38] Cloud, Google: *O que é o pub/sub?*, 2023. https://cloud.google.com/pubsub/docs/overview?hl=pt-br, acesso em 2023-03-09. 17

[39] Cloud, Google: *Firestore*, 2023. https://cloud.google.com/firestore?hl=pt-br, acesso em 2023-03-09. 17

[40] Fn-Project: *Open source. container-native. serverless platform.*, 2021. https://fnproject.io/. 18, 24

[41] Inc., Docker: *Docker docs.*, 2023. https://docs.docker.com/, acesso em 2023-03-09. 18

[42] Oracle: *Object storage.*, 2023. https://www.oracle.com/br/cloud/storage/object-storage/, acesso em 2023-03-09. 18

[43] Oracle: *Oracle notification service (ons).*, 2023. https://docs.oracle.com/en-us/iaas/pl-sql-sdk/doc/ons-package.html, acesso em 2023-03-09. 18

[44] Djemame, Karim, Matthew Parker e Daniel Datsev: *Open-source serverless architectures: an evaluation of apache openwhisk*. Em *2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*, páginas 329–335, 2020. 18, 19, 24, 48

[45] IBM: *Object storage.*, 2023. https://www.ibm.com/cloud/object-storage, acesso em 2023-03-09. 19

[46] IBM: *Ibm watson iot platform - message gateway.*, 2023. https://www.ibm.com/docs/en/wip-mg/5.0.0?topic=concepts-message-hubs-endpoints-policies, acesso em 2023-03-09. 19

[47] Foundation., Apache Software: *Software for the public good.*, 2023. https://www.apache.org/, acesso em 2023-03-09. 19

[48] Fission: *Open source, kubernetes-native serverless framework*, 2021. https://fission.io, acesso em 2021/08/10. 20, 24

[49] Knative: *Knative - enterprise-grade serverless on your own terms.*, 2021. https://knative.dev, acesso em 2021/08/10. 20, 21, 24

[50] Kaewkasi, Chanwit: *Docker for Serverless Applications: Containerize and Orchestrate Functions Using OpenFaas, OpenWhisk, and Fn.* Packt Publishing, 2018, ISBN 1788835263. 21

[51] OpenFaaS: *Serverless functions, made simple.*, 2021. https://www.openfaas.com/. 21, 24

[52] KUBELESS: *Kubeless - the kubernetes native serverless framework: Build advanced applications with faas on top of kubernetes*, 2019. https://kubeless.io/, acesso em 14/06/2019. 21, 24

[53] Knix: *Knix: A high-performance, open-source serverless computing platform*, 2021. https://knix.io/, acesso em 2021/08/10. 22, 24

[54] STUBBS, Joe, Rion Dooley e Matthew Vaughn: *Containers-as-a-service via the Actor Model*. Em *Gateways 2016 proceedings*, janeiro 2017. https://figshare.com/articles/Containers-as-a-service_via_the_Actor_Model/4490747, acesso em 05/07/2020. 22, 24

[55] CHARD, Ryan, Tyler J. Skluzacek, Zhuozhao Li, Yadu Babuji, Anna Woodard, Ben Blaiszik, Steven Tuecke, Ian Foster e Kyle Chard: *Serverless supercomputing: High performance function as a service for science*, 2019. 23, 24

[56] Pfandzelter, Tobias e David Bermbach: *tinyfaas: A lightweight faas platform for edge environments*. Em *2020 IEEE International Conference on Fog Computing (ICFC)*, páginas 17–24, 2020. 23, 24

[57] FAASDOM: *The faasdom benchmark suite*, 2021. https://github.com/faas-benchmarking/faasdom. 27, 30

[58] Halili, Emily H: *Apache JMeter*. Packt Publishing Birmingham, 2008. 27, 30

[59] Jain, Raj: *The art of computer systems: Techniques for experimental design, measurement, simulation, and modeling*. John Wiley & Sons,, 1991. 28

[60] N VOLPATO, M. CUNICO e M. CUNICO e O. MIGUEL e S. ZAWADZKI e P. PERALTA-ZAMORA e: *Planejamento fatorial: Uma ferramenta estatÍstica valiosa para a definiÇÃo de parÂmetros experimentais empregados na pesquisa cientÍfica.* Visão Acadêmica, 9(1), 2008, ISSN 1518-8361. https://revistas.ufpr.br/academica/article/view/14635. 28

[61] Dell.: *Dell ecs série ex.*, 2024. https://www.delltechnologies.com/asset/pt-br/products/storage/technical-support/h13117-emc-ecs-appliance-ss.pdf, acesso em 2024-05-05. 43

[62] Technologies, Dell: *Armazenamento empresarial em objeto do dell ecs*, 2023. https://www.dell.com/pt-br/dt/storage/ecs/index.htm#tab0=0&tab1=0, acesso em 2023-03-07. 43

[63] Suse, Rancher by: *What is rancher?*, 2023. https://ranchermanager.docs.rancher.com/, acesso em 2023-03-07. 44

[64] Foundation., Apache Software: *Openwhisk deployment on kubernetes.*, 2023. https://github.com/apache/openwhisk-deploy-kube, acesso em 2024-01-04. 46

[65] Foundation., The Linux: *The package manager for kubernetes.*, 2024. https://helm.sh/, acesso em 2024-01-04. 46

[66] Apache.: *Deploying openwhisk on google gke.*, 2024. https://github.com/apache/openwhisk-deploy-kube/blob/master/docs/k8s-google.md, acesso em 2024-01-04. 48

[67] Apache.: *Deploying openwhisk on amazon eks.*, 2024. https://github.com/apache/openwhisk-deploy-kube/blob/master/docs/k8s-aws.md, acesso em 2024-01-04. 48

[68] Apache.: *Deploying openwhisk on ibm cloud kubernetes service (iks).*, 2024. https://github.com/apache/openwhisk-deploy-kube/blob/master/docs/k8s-ibm-public.md, acesso em 2024-01-04. 48

[69] Carvalho., Leonardo Reboucas de: *Orama framework.*, 2024. https://github.com/unb-faas/orama, acesso em 2024-01-04. 50