INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SERGIPE

COORDENADORIA DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO

Um Guia Prático para Migração de Workloads para Containerização em Ambientes de Nuvem

Trabalho de Conclusão de Curso

Marconi Nascimento Oliveira

Lagarto – Sergipe

2024

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SERGIPE

COORDENADORIA DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO

Marconi Nascimento Oliveira

Um Guia Prático para Migração de Workloads para Containerização em Ambientes de Nuvem

Trabalho de conclusão de curso apresentado a Coordenadoria de Bacharelado em Sistemas de Informação do Instituto Federal de Sergipe como requisito parcial para a obtenção do título de bacharel em Sistemas de Informação.

Orientador(a): Rubens de Souza Matos Junior

Lagarto – Sergipe

2024

Oliveira, Marconi do Nascimento.
O48g Um guia prático para migração de workloads para containerização em ambientes de nuvem / Marconi do Nascimento Oliveira. – Lagarto, 2024.
40 f. ; il.

Monografia (Graduação) Bacharelado em Sistemas de Informação. Instituto Federal de Educação Ciência e Tecnologia de Sergipe – IFS, 2024.
Orientador: Prof. Dr. Rubens de Souza Matos Junior.

1. Nuvem. 2. Tecnologia da informação. 3. Recuperação da informação. 4. Computação em nuvem. I. Instituto Federal de Educação Ciência e Tecnologia de Sergipe – IFS. II. Título.

CDU 004

Agradecimentos

Gratidão a Deus pela vida, por me guiar e me dar forças durante todo o meu trabalho de conclusão de curso.

À minha família, por sempre acreditarem em mim. Às mulheres da minha vida; minha mãe Evanilda, minha esposa Maria Aparecida, minas filhas Maria Clara e Lara. Ao meu pai José Alberto. Sem vocês, os caminhos teriam sido bem mais árduos.

Aos Professores: Rubens Matos e Luana Barreto, pelas orientações e pela oportunidade de executar esse trabalho e pela paciência. Todos professores do curso de BSI do IFS-SE Campos Lagarto. Em especial: Wilhelm, Glauco, Francisco, João Paulo, Adriana Fontes, Catuxe e Jislaine pelos ensinamentos, orientações, força, prontidão e serenidade, em momentos críticos, que foram essenciais para continuar seguindo em frente.

Aos amigos do Trabalho, Alexandre Andrade, Juliana Góis, Aline Medeiros, Emilly Almeida, Allaphy e Nilton que contribuíram para o meu crescimento intelectual, minha formação e minha vida, seja em uma aula, dúvida ou tomando um café durante os intervalos.

Saudações especiais ao meu compadre Danilo Souza que sempre contribuindo com meu crescimento profissional intelectual e sempre me incentivando a continuar.

A todas as pessoas que passaram pela minha vida e contribuíram para a construção de quem sou hoje.

Dedicatória

Dedico este trabalho aos meus pais, pelo exemplo de vida e valores que me inspiram;
à minha esposa, pelo apoio incondicional e paciência em cada etapa desta jornada;
às minhas filhas, que são minha maior motivação;
ao meu compadre, pela amizade sincera e parceria de longa data;
e a toda minha família, por estarem sempre presentes, com amor e incentivo, em todos os momentos.

Resumo

Nos últimos anos, a adoção de contêineres tem impulsionado a migração de aplicações para a nuvem, destacando-se pela eficiência no uso de recursos de *hardware*, escalabilidade, portabilidade e facilidade de gerenciamento. Neste cenário, este trabalho foca na migração de uma aplicação desenvolvida em .NET Core, originalmente executada em um ambiente local, para uma arquitetura baseada em contêineres. O estudo explora as vantagens e os desafios desse processo, oferecendo uma visão detalhada que pode servir como um guia para auxiliar no processo de migração e na evolução para tecnologias que garantam alta disponibilidade e escalabilidade. Como resultado, a pesquisa apresenta uma análise aprofundada sobre o tema e busca incentivar outros trabalhos futuros, contribuindo com mais conhecimento e convidando o leitor a reconsiderar e otimizar a forma como gerenciam e utilizam seus recursos de infraestrutura computacional.

- **Palavra Chave:** Container, Docker, Kubernetes, Nuvem.

Lista de ilustrações

Lista de abreviaturas e siglas

AWS	Amazon Web Services
BaaS	Backend as a Service
CaaS	Container as a Service
chroot	Change Root
CI/CD	Continuous Integration / Continuous Deployment
CLI	Command Line Interface
CPU	Central Processing Unit
DNS	Domain Name System
Docker	Plataforma de contêineres
EC2	Elastic Compute Cloud
ELB	Elastic Load Balancer
ELK	Elasticsearch, Logstash, Kibana
EKS	Elastic Kubernetes Service
GB	Gigabyte
Host	Máquina que hospeda serviços ou aplicativos
IaaS	Infrastructure as a Service
IDE	Integrated Development Environment
I/O	Input/Output
MVC	Model-View-Controller
PaaS	Platform as a Service
Pod	Menor unidade de implantação no Kubernetes
RAM	Random Access Memory
RDS	Relational Database Service
SaaS	Software as a Service

SQLite	Banco de dados leve
SSD	Solid State Drive
TI	Tecnologia da Informação
Worker	Instância de trabalho
Workload	Carga de trabalho
WSL 2	Windows Subsystem for Linux 2
YAML	YAML Ain’t Markup Language

Sumário

1 Introdução

A computação em nuvem tem se tornado cada vez mais popular devido a seu modelo de negócios atrativo. De acordo com (MOÇO; CUNHA, 2020) a nuvem é comumente associada à internet, que funciona como uma infraestrutura de comunicação. Essa infraestrutura é composta por uma variedade de elementos, incluindo *hardware*, *software*, *interfaces*, redes de telecomunicação e dispositivos de controle e armazenamento, permitindo que a computação seja disponibilizada como um serviço. A possibilidade de gerenciamento remoto, pagamento conforme o uso, escalabilidade e o uso de tecnologias como virtualização de servidores, armazenamento, rede e ambientes de desenvolvimento atraem muitas empresas. Esses fatores levam as organizações a considerarem a migração de seus serviços para a nuvem. A redução de custos, a otimização de processos e a melhoria na qualidade dos produtos e serviços são algumas das principais razões que despertam o interesse de diferentes tipos de organizações para mover suas aplicações para a nuvem. Contudo, apesar dos benefícios, existem desafios que dificultam essa adoção, especialmente quando se trata de migrar aplicações complexas para plataformas de nuvem pública. A migração para a nuvem é um processo complexo, no qual diversos fatores, como custos, desempenho e segurança, devem ser cuidadosamente avaliados na tomada de decisões.

1.1 Justificativa e Relevância

Existem diferentes maneiras de executar um *workload* (carga de trabalho) na nuvem, cada uma delas com características diferentes, como por exemplo flexibilidade, elasticidade, custo e administração. Alguns *workloads* possuem requisitos que podem permitir que se consiga obter mais benefícios a partir do uso de um determinado tipo de serviço de computação, dentre os disponíveis nas plataformas de nuvens populares no mercado. Conteinerizar um *workload* é uma possibilidade de execução, porém a migração de um *workload* entre modelos de computação

não é algo simples e requer a avaliação de alguns aspectos do *workload*, a obediência a alguns critérios importantes, assim como a observação de alguns pontos de atenção.

No que se refere ao desenvolvimento de software, levando em conta o avanço tecnológico, no contexto da computação em nuvem, vêm surgindo ferramentas que possam facilitar o trabalho do desenvolvedor. A migração de cargas de trabalho para a nuvem tem sido amplamente otimizada pela conteinerização, que permite encapsular aplicações e suas dependências em contêineres leves. Essa abordagem não apenas simplifica o processo de implantação, mas também proporciona maior portabilidade entre diferentes ambientes, seja em nuvens públicas, privadas ou híbridas. A adoção de contêineres em aplicações web, por exemplo, tem sido analisada por (CARVALHO; SANTOS, 2019), que destacam os benefícios e os desafios dessa tecnologia.

Além disso, a conteinerização possibilita a escalabilidade eficiente das aplicações, permitindo que os desenvolvedores ajustem rapidamente os recursos conforme a demanda, melhorando a performance e reduzindo custos. Ferramentas como Kubernetes e Docker têm se destacado nesse cenário, facilitando a orquestração e gestão dos contêineres. (JúNIOR; GOMES, 2020) também discutem a adoção de computação em nuvem em pequenas e médias empresas, enfatizando a importância dessa tecnologia para a inovação e eficiência operacional.

Com essas inovações, os desenvolvedores podem focar na criação de *software* de alta qualidade, enquanto as complexidades da infraestrutura são gerenciadas de maneira mais automatizada e eficaz.

1.2 Objetivos

1.2.1 Objetivos Gerais

Apresentar a proposta de um guia para o processo de migração de um *workloads* para uma infraestrutura de conteiners na nuvem exemplificado através de um estudo de caso.

1.2.2 Objetivos específicos

- Apresentar o conceito de computação em nuvem e os tipos de serviço.;
- Apresentar as principais categorias;
- Associar características de um *workload* a pontos de atenção do processo conteinerização;
- Relacionar as possíveis etapas desse processo e pontos de atenção.

1.3 Estrutura do Documento

Para facilitar a navegação e melhor entendimento, este documento está estruturado em capítulos e seções, que são:

- **Capítulo 1 - Introdução**: Apresenta a justificativa argumentações sobre o tema e as definições preliminares de literatura;
- **Capítulo 2 - Fundamentação teórica**: Demonstra os trabalhos correlatos com a revisão de literatura adotada, bem como são apresentados os resultados dessa revisão;
- **Capítulo 3 - Estudo de caso**:Este capítulo apresenta os métodos utilizados para a efetivação deste trabalho;
- **Capítulo 4 - Conclusões e Considerações**: Este capítulo apresenta o resultado obtido, desafios e considerações..
- **Referencias**

2

Fundamentação Teórica

Neste capítulo são definidos e delimitados os paradigmas e características dos temas abordados. Para tanto, foram realizadas consultas nas literaturas mais relevantes da área, com o objetivo de fornecer o embasamento teórico necessário para este estudo, o qual está focado na análise dos serviços oferecidos por provedores de nuvem pública. Este conhecimento inicial é fundamental para a execução e desenvolvimento adequado da pesquisa.

2.1 Computação em nuvem

2.1.1 Conceito de Computação em nuvem

Em (TAURION, 2009) a computação em nuvem está definida como um conjunto de recursos com capacidade de processamento, armazenamento, conectividade, plataformas, aplicações e serviços disponibilizados na internet. O resultado é que a nuvem pode ser vista como o estágio mais evoluido de conceito de virtualização. A partir dessa definição podemos entender também que computação em nuvem está relacionada ao aluguel da capacidade ociosa de datacenters, que constituem-se em provedores de serviços computacionais sob demanda com elasticidade escalável.

O trabalho apresentado por (CHAVES; CASTRO; NASCIMENTO, 2021) também define que o modelo de computação em nuvem foi criado com o objetivo de oferecer redução de custos, maior flexibilidade, escalabilidade e acesso global, superando as limitações de uma infraestrutura local nas organizações. mundiais. Apresenta a vantagem de tornar os processos de TI independentes da infraestrutura, dando mais elasticidades ao negócio. Para (VERAS, 2012) a computação em nuvem facilita a separação entre os processos empresariais e a TI necessária para executá-los. Além disso, traz o conceito de elasticidade no uso da infraestrutura, permitindo que as organizações adaptem de maneira mais flexível o uso dos recursos tecnológicos às demandas

comerciais.

2.1.2 Tipos de nuvem

São conhecidos por 3 tipos, conforme (Penso, 2022) descreve:

- **Nuvem Pública:** A nuvem pública é o modelo mais difundido e popular de computação em nuvem entre os usuários. Ele é particularmente atraente para empresas que buscam economizar custos. Nesse modelo, os servidores da nuvem são normalmente fornecidos por terceiros, tornando o acesso ao sistema mais acessível para aqueles que desejam utilizá-lo.
- **Nuvem Privada:** A nuvem privada é um ambiente de armazenamento em nuvem dedicado exclusivamente ao uso pessoal ou empresarial restrito. Isso resulta em um serviço de nuvem altamente personalizado, projetado para atender às necessidades específicas do usuário.
- **Nuvem Híbrida:** Como o próprio nome sugere, a nuvem híbrida combina elementos dos modelos público e privado, permitindo a alocação de aplicativos e dados entre esses dois ambientes.

2.1.3 Tipos de Serviço

2.1.3.1 Infraestrutura como Serviço (IaaS)

No contexto do IaaS (*Infraestructure as a Service*), segundo (PAULO, 2021) encontramos uma categoria de serviços em nuvem que oferece elementos fundamentais, como capacidade de processamento, espaço de armazenamento, conectividade de rede e uma variedade de outros recursos de infraestrutura. Esses recursos são disponibilizados por meio da criação de máquinas virtuais, as quais são geradas e hospedadas nos data centers dos provedores de infraestrutura em nuvem.

O IaaS permite aos usuários a flexibilidade de configurar e gerenciar recursos de TI em um ambiente virtualizado, eliminando a necessidade de investir em *hardware* físico e infraestrutura de data center local. Além disso, esse modelo oferece escalabilidade sob demanda, permitindo que as organizações dimensionem seus recursos de acordo com suas necessidades em evolução. Exemplos notáveis de serviços IaaS incluem *Amazon Web Services* (AWS) EC2 - *Elastic Compute Cloud* e *Google Cloud Platform*, que fornecem uma ampla gama de recursos de infraestrutura em nuvem para atender às necessidades de diversas aplicações e empresas.

2.1.3.2 Plataforma como Serviço (PaaS)

O modelo *Platform as a Service* (PaaS) é uma solução de computação em nuvem que disponibiliza uma plataforma para desenvolver, executar e administrar aplicações, eliminando a

necessidade de gerenciar a infraestrutura subjacente. Conforme descrito por (DAVIS, 2023), o PaaS proporciona um ambiente de desenvolvimento abrangente, equipado com ferramentas e serviços integrados que simplificam tanto a criação quanto a implementação de aplicativos.

Um aspecto fundamental do PaaS é que os usuários não precisam se preocupar em adquirir ou administrar recursos de *hardware* e *software* por conta própria, nem contratar especialistas para gerenciar esses recursos. Essa abordagem oferece uma notável flexibilidade na implantação de *software* no ambiente, permitindo que as empresas dimensionem suas operações de acordo com suas necessidades. No entanto, é importante observar que uma desvantagem do PaaS é a potencial falta de interoperabilidade e portabilidade entre diferentes provedores de serviços em nuvem, o que pode limitar a flexibilidade em casos de migração entre plataformas (RASHID; CHATURVEDI, 2019).

2.1.3.3 Software como Serviço (SaaS)

De acordo com (SILVA, 2017) o modelo de *Software as a Serviçe* (SaaS) oferece software com funções específicas que estão disponíveis para os usuários por meio da internet. Esses *softwares* podem ser acessados a partir de diversos dispositivos dos usuários, geralmente através de uma interface de cliente leve, como um navegador *web*. No contexto do SaaS, o usuário não precisa gerenciar ou controlar a infraestrutura subjacente, que engloba aspectos como rede, servidores, sistemas operacionais, armazenamento, e até mesmo as características individuais da aplicação, exceto por configurações específicas. Para (OLIVEIRA; JR; ALBUQUERQUE, 2010) o prestador de serviços oferece seus servidores e o suporte necessário para o funcionamento das aplicações aos clientes. Em troca, ele recebe uma taxa de assinatura, garantindo sua viabilidade financeira. Dessa forma, tanto o cliente quanto o provedor se beneficiam no modelo SaaS.

2.1.3.4 Container como serviço (CaaS)

Conforme descrito em (HUSSEIN; MOUSA; ALQARNI, 2019), CaaS ou *Container as a Serviçe* se enquadram na categoria de serviços em nuvem nos quais os provedores de serviços possibilitam aos usuários implantar e gerenciar *clusters* e aplicativos em contêiner. Algumas pessoas consideram essa plataforma um subtipo de IaaS, mas a diferença surge no ponto em que a principal mercadoria do CaaS são os *containers* e não as máquinas virtuais ou o *hardware* físico.

2.2 Principais categorias de serviços de nuvem

2.2.1 Armazenamento

O armazenamento em nuvem tem se consolidado como uma solução estratégica para o gerenciamento de dados em várias indústrias. De acordo com (SILVA, 2022), essa tecnologia

envolve o uso de servidores remotos para o armazenamento e processamento de informações, permitindo o acesso via internet. Esse novo paradigma transformou a maneira como as empresas gerenciam grandes volumes de dados, proporcionando flexibilidade, escalabilidade e redução de custos em relação aos métodos de armazenamento tradicionais.

2.2.2 Processamento

Um dos grandes benefícios da computação em nuvem é sua habilidade de realizar processamento em grande escala. Através da virtualização, os prestadores de serviços podem maximizar o uso de seus recursos. (SILVA; ALMEIDA, 2020) afirmam que "a virtualização é uma das tecnologias essenciais que tornam possível a eficiência do processamento em nuvem, permitindo que várias máquinas virtuais funcionem em um único servidor físico".

2.2.3 Bancos de dados

A utilização de bancos de dados na nuvem traz várias vantagens, como a eliminação da necessidade de manter infraestrutura e a facilidade de acesso remoto. No entanto, (RAMESH; LOGESHWARAN; ARAVINDARAJAN, 2022) observa que um dos principais desafios é a dependência da conexão com a internet e as questões de segurança relacionadas ao armazenamento de dados fora do controle direto da organização.

2.2.4 Segurança

A segurança na nuvem é um aspecto fundamental da computação em nuvem, abrangendo uma ampla gama de práticas, controles e tecnologias projetadas para proteger dados, aplicações e infraestruturas associadas ao uso de serviços em nuvem. À medida que as organizações adotam cada vez mais soluções de computação na nuvem, a necessidade de garantir a proteção dos ativos digitais armazenados nesses ambientes torna-se crítica. Em (COUTINHO; NEVES; LOPES, 2021) o controle de acesso aos dados é uma maneira eficaz de garantir segurança de dados na nuvem.A segurança na nuvem não se refere apenas à proteção contra ameaças externas, como *hackers* e *malware*, mas também à garantia de conformidade com normas regulatórias, gestão de acesso, privacidade e integridade dos dados.

2.3 Workload

Segundo (HUSSEIN; MOUSA; ALQARNI, 2019) *workload* é a carga de trabalho e a quantidade de processamento que o computador recebe em um determinado momento. A carga de trabalho refere-se à quantidade de aplicativos sendo executados em um computador, normalmente envolvendo um número de usuários conectados que interagem com esses aplicativos.

2.4 Tecnologia de Contêineres e sua Evolução

Os fundamentos que deram origem à tecnologia de *containers* modernos podem ser rastreados até os anos 1970, especificamente com o desenvolvimento dos conceitos de "jails" e "chroot" nos sistemas Unix. O comando chroot, introduzido em 1979, representou uma inovação significativa ao permitir que administradores de sistema isolassem processos em um diretório específico, criando um ambiente controlado e restrito para sua execução (IMESH, 2016). Esse conceito de isolamento foi um precursor importante para o que mais tarde evoluiria para a tecnologia de *containers*.

O uso do chroot possibilitou uma forma primitiva de contenção e segurança dentro dos sistemas Unix, estabelecendo uma base para a evolução das técnicas de virtualização. A partir desse ponto, a ideia de criar ambientes isolados e independentes dentro de um mesmo sistema operacional continuou a se desenvolver, eventualmente contribuindo para a criação dos containers modernos.

2.4.1 Infraestrutura de containers

Segundo (PAHL, 2015) um contêiner visa prover uma unidade de computação auto-suficiente para execução de uma aplicação em um ambiente computacional e contém partes autocontidas e prontas para implantação de aplicativos e se necessário, *middleware* e lógica de negócios (em binários e bibliotecas) para execução de aplicações . Os *containers* têm se destacado como uma solução eficaz para a virtualização de aplicações, facilitando o empacotamento e a distribuição de software de maneira eficiente. Diferentemente das máquinas virtuais, que realizam a virtualização a partir do hardware, os contêineres operam a nível do sistema operacional. Isso permite que diversos containers utilizem o mesmo *kernel* do sistema operacional, enquanto cada um mantém suas próprias bibliotecas e dependências. Esses containers são projetados para incluir todos os componentes necessários para a execução de uma aplicação, como bibliotecas e dependências específicas. Esse encapsulamento assegura que o *software* se comporte de forma consistente em diferentes ambientes. Esse método é especialmente vantajoso em contextos de desenvolvimento ágil e integração contínua, onde a garantia de desempenho uniforme em diversos ambientes é fundamental.

2.4.2 Docker

Segundo (DOCKER, 2023), o Docker é uma plataforma aberta que possibilita o desenvolvimento, envio e execução de aplicativos de maneira eficiente. Permitindo a separação dos aplicativos da infraestrutura subjacente, o que facilita a entrega ágil de *software*. Com essa abordagem, é possível gerenciar a infraestrutura de modo similar à forma como se gerenciam os próprios aplicativos. Dessa forma, ao adotar as metodologias proporcionadas pelo Docker

para o envio, teste e implantação de código, a latência entre o desenvolvimento e a execução em produção é significativamente reduzida.

A plataforma Docker fornece ferramentas que possibilitam a gestão completa do ciclo de vida dos contêineres. No início, o desenvolvimento do aplicativo e de seus componentes ocorre utilizando contêineres. Em seguida, o contêiner se torna a unidade central para a distribuição e o teste do aplicativo. Assim, após estar pronto, o aplicativo pode ser implantado em um ambiente de produção, seja ele local, em nuvem ou uma combinação de ambos, com a mesma eficácia. Isso torna o Docker uma solução viável para ambientes de produção orquestrados.

Adicionalmente, o Docker simplifica o ciclo de vida do desenvolvimento ao permitir que os desenvolvedores operem em ambientes padronizados utilizando contêineres locais, que fornecem tanto aplicativos quanto serviços. Essa padronização é especialmente benéfica para fluxos de trabalho que envolvem integração contínua e entrega contínua (CI/CD) (DOCKER, 2023).

2.4.3 Kubernetes

Conforme citado por (MORAES et al., 2021) uma das plataformas mais amplamente adotadas para a criação de contêineres é o Docker, enquanto o Kubernetes é amplamente utilizado para gerenciar esses contêineres. O Kubernetes opera sob um modelo hierárquico, onde cada unidade de processamento é referida como "nó". Neste sistema, o nó que exerce funções de gerenciamento é conhecido como "nó Master", enquanto os nós que realizam as tarefas são chamados de "nós *Workers*". A função do nó Master é orquestrar os contêineres Docker distribuídos entre os diversos nós do Kubernetes, que podem ser tanto máquinas físicas quanto virtuais, dedicadas à execução dos *containers*. Juntas, essas máquinas formam um "*cluster* Kubernetes".

Dessa forma, a implantação de aplicações em contêineres Docker é realizada por meio do nó Master do Kubernetes. Comumente, uma aplicação é segmentada em vários componentes, conhecidos como contêineres, que são executados em um ou mais "*pods*". Cada *pod*, por sua vez, possui limites definidos quanto à quantidade máxima de recursos que pode consumir. Uma das principais vantagens do Kubernetes é sua capacidade de otimizar e gerenciar o uso de hardware, como RAM, disco e CPU, resultando em uma economia significativa de recursos (MORAES et al., 2021).

2.4.3.1 Benefícios

O emprego de *containers* na nuvem tem se tornado cada vez mais frequente, graças aos vários benefícios que essa tecnologia proporciona. Entre as principais vantagens estão a portabilidade e a escalabilidade. Os *containers* possibilitam que as aplicações sejam agrupadas com todas as suas dependências, o que simplifica a migração entre diferentes ambientes de

nuvem. De acordo com (COSTA; LIMA, 2021), "a portabilidade dos contêineres é uma característica fundamental que permite que as aplicações sejam executadas de maneira consistente em ambientes diversos, seja na nuvem pública ou privada".

3

Estudo de Caso sobre Migração de Workloads para Containerização

Este trabalho baseia-se na metodologia de estudo de caso, conforme descrito por (YIN, 2015). O estudo de caso envolve uma investigação aprofundada em um contexto contemporâneo do mundo real, especialmente quando os limites entre o fenômeno e o contexto não são claramente definidos. O objetivo deste estudo foi realizar a migração de uma aplicação desenvolvida em .Net Core para *containers* Docker, abordando todas as etapas desse processo.

Para a migração, foi selecionada uma aplicação de conta bancária desenvolvida em .Net Core, utilizando o padrão de design de software MVC (*Model-View-Controller*), que separa a lógica da aplicação em três componentes principais: *Model*, responsável pela gestão dos dados; *View*, que trata da interface do usuário; e *Controller*, que gerencia a lógica de controle e a comunicação entre os outros dois componentes. A persistência dos dados da aplicação é realizada em um banco de dados em arquivo SQLite. A escolha desta aplicação baseou-se em uma avaliação criteriosa de arquitetura, dependências e recursos necessários, com foco na conteinerização deste *workload*. Esse planejamento inicial considerou tanto os requisitos técnicos quanto às necessidades de infraestrutura, estabelecendo uma base sólida para a migração.

A aplicação selecionada para o estudo de caso serviu como base para demonstrar os processos e desafios envolvidos na migração para um ambiente conteinerizado. Com a persistência de dados gerida por um banco de dados em arquivo SQLite, a aplicação foi ideal para ilustrar a transição para Docker compose, dado seu equilíbrio entre complexidade técnica e relevância prática.

O estudo de caso foi conduzido utilizando uma máquina local com configuração básica,

incluindo um processador Intel Core i5, 16GB de memória RAM e um disco SSD. O sistema operacional utilizado foi o Windows 11 Pro, e a aplicação foi desenvolvida na IDE Microsoft Visual Studio 2022, com suporte adicional do Docker Desktop versão 24.06 para a gestão dos containers. Para a execução de comandos específicos, foi utilizada a IDE Microsoft Visual Studio Community 2022

3.1 Geração da Imagem Docker no Visual Studio para uma Aplicação em .NET Core MVC

.

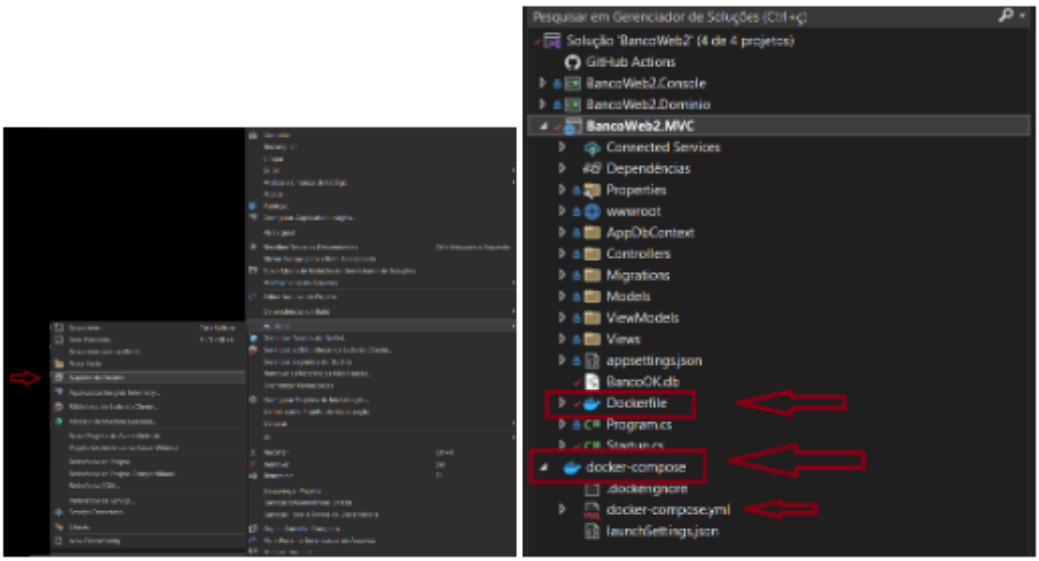
Ao definir a escolha da aplicação em .NET Core MVC no Visual Studio, você pode gerar uma imagem Docker para uso em um ambiente Docker Compose. A seguir, estão as etapas principais para criar e configurar a imagem Docker diretamente no Visual Studio:

1. **Configurar o Projeto para Suporte ao Docker:** Antes de mais nada, é importante garantir que o projeto está configurado para ser compatível com o Docker. Se o suporte ao Docker ainda não estiver habilitado, segue-se estas etapas:

    - No Visual Studio, clicar com o botão direito no projeto da aplicação em Solution Explorer.
    - Selecionar a opção Add e depois Docker Support.
    - Escolher Linux como a plataforma, já que a maioria dos ambientes Docker são otimizados para Linux, embora você possa usar Windows se necessário.

    Em seguida, o Visual Studio criará um Dockerfile dentro do projeto, que será utilizado para construir a imagem Docker da aplicação, conforme ilustrado na Figura 1 abaixo:

Figura 1 – Habilitando o suporte Docker no projeto



Fonte: Elaborado pelo autor

2. **Entendendo o Dockerfile Gerado:** O *Dockerfile* define o processo de construção da imagem Docker. A estrutura gerada geralmente inclui:

   - **Imagem base:** Geralmente uma imagem oficial do .NET, como `mcr.microsoft.com/dotnet/aspnet:3.1`, que é utilizada para executar aplicações ASP.NET Core 3.1.
   - **Cópia do código fonte:** O código da aplicação é copiado para dentro do contêiner.
   - **Restauração de dependências:** Usa o comando dotnet restore para restaurar todas as dependências do projeto.
   - **Build da aplicação:** Usa o comando dotnet build para compilar a aplicação.
   - **Execução da aplicação:** Define o comando de execução, como dotnet <nome do arquivo>.dll.O *Dockerfile* gerado deve ficar conforme Figura 2:

Figura 2 – Geração Imagem Docker

```
## Imagem Base
FROM mcr.microsoft.com/dotnet/aspnet:3.1 AS base
WORKDIR /app
EXPOSE 80
EXPOSE 443

## Compilação
FROM mcr.microsoft.com/dotnet/sdk:3.1 AS build
WORKDIR /src
COPY ["BancoWeb2.MVC/BancoWeb2.MVC.csproj", "BancoWeb2.MVC/"]
COPY ["BancoWeb2.Dominio/BancoWeb2.Dominio.csproj", "BancoWeb2.Dominio/"]
RUN dotnet restore "BancoWeb2.MVC/BancoWeb2.MVC.csproj"
COPY . .
WORKDIR "/src/BancoWeb2.MVC"
RUN dotnet build "BancoWeb2.MVC.csproj" -c Release -o /app/build

##Publicação
FROM build AS publish
RUN dotnet publish "BancoWeb2.MVC.csproj" -c Release -o /app/publish /p:UseAppHost=false

##Geração da imagem
FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "BancoWeb2.MVC.dll"]
```

Fonte: Elaborado pelo autor

3. **Personalizando o Dockerfile:** O *Dockerfile* pode ser ajustado conforme as necessidades do projeto. Algumas personalizações comuns incluem:

   - Adicionar variáveis de ambiente para configurar a aplicação.
   - Definir volumes para persistir dados, especialmente se a aplicação interage com bancos de dados ou arquivos.
   - Expor portas específicas que a aplicação utiliza.

4. **Gerar a Imagem no Visual Studio:** Com o *Dockerfile* configurado, é possível criar a imagem Docker diretamente pelo Visual Studio:

   - No Visual Studio, seleciona-se Build e depois Build Solution.
   - Alternativamente, clica-se com o botão direito no projeto e escolhe-se *Container* Tools > Build Docker Image.

   O Visual Studio então usa o *Dockerfile* para construir a imagem e a armazena localmente. Esse processo inclui:

   - Baixar as dependências da imagem base (caso ainda não estejam no sistema).
   - Construir o código da aplicação em contêiner.
   - Publicar a aplicação em um ambiente pronto para execução.

5. **Executando a Aplicação no Docker Compose** Após a geração da imagem, é hora de configurá-la para ser executada dentro de um ambiente Docker Compose. Isso envolve

a criação de um arquivo docker-compose.yml, que define como os serviços (incluindo a aplicação) serão orquestrados.

3.2 Composição do Dockerfile e Docker Compose

A criação de um *Dockerfile* é um passo essencial para garantir que todas as dependências necessárias sejam incluídas e que as variáveis de ambiente sejam configuradas corretamente. O *Dockerfile* é um arquivo de texto que contém uma série de instruções que o Docker utiliza para construir uma imagem. Essas instruções especificam a base da imagem, as dependências a serem instaladas, os arquivos a serem copiados, e as variáveis de ambiente que a aplicação precisará ao ser executada dentro de um *container*.

Além do Dockerfile, utiliza-se o Docker Compose para orquestrar a criação e execução de múltiplos containers. O Docker Compose permite que todas as definições de configuração para os containers sejam centralizadas em um único arquivo docker-compose.yml, escrito em YAML. Esse arquivo especifica as configurações necessárias para a execução de um ou mais containers, facilitando a criação de sistemas complexos onde há interação entre múltiplos containers.

Para a composição do Docker Compose, alguns pontos importantes foram considerados:

1. **Serviço "app":** Conforme ilustrado na Figura 3, o serviço chamado “app” inclui todas as especificações necessárias para executar a aplicação web, como a definição da imagem Docker, o apontamento para o banco de dados, as variáveis de ambiente, o mapeamento de portas e o volume que armazena dados persistentes.

Figura 3 – Configuração do serviço “app” no arquivo docker-compose.yml

```
services:
  app:
    image: bancook:latest
    container_name: app
    links:
      - db
    environment:
      - ASPNETCORE_ENVIRONMENT=Development
      - ASPNETCORE_URLS=https://+:443;http://+:80
      - DATABASE_CONNECTION=server=db; user=root; password=root; database=db_bancook
    ports:
      - 60951:80
      - 60952:443
    volumes:
      - ${APPDATA}/Microsoft/UserSecrets:/root/.microsoft/usersecrets:ro
      - ${APPDATA}/ASP.NET/Https:/root/.aspnet/https:ro
```

Fonte: Elaborado pelo autor

2. **Serviço "db":** Conforme mostrado na Figura 4, foi criado um serviço chamado "db"para o banco de dados, utilizando uma imagem pública do MySQL 8. Esse serviço segue os

mesmos parâmetros para criação especificados no Docker Compose, garantindo que o banco de dados esteja corretamente integrado à aplicação.

Figura 4 – Configuração do serviço "db"utilizando MySQL no arquivo docker-compose.yml

```yaml
db:
  image: mysql:8.0
  container_name: db
  cap_add:
    - SYS_NICE
  restart: always
  environment:
    - MYSQL_DATABASE=db_bancook
    - MYSQL_ROOT_PASSWORD=root
    #- MYSQL_USER: db_user
    #- MYSQL_PASSWORD: db_user_pass
  ports:
    - '3306:3306'
  volumes:
    - db:/var/lib/mysql
```

Fonte: Elaborado pelo autor

3. **Serviço "phpMyAdmin":** Finalmente, na Figura 5, é mostrado o serviço "phpMyAdmin", criado para acesso e gerenciamento do banco de dados MySQL. Esse serviço utiliza uma imagem pública do phpMyAdmin, sendo configurado para se conectar ao serviço de banco de dados, com a definição de variáveis de ambiente, rede e volumes necessários.

Figura 5 – Configuração do serviço "phpMyAdmin"no arquivo docker-compose.yml

```yaml
 phpmyadmin:
   image: phpmyadmin/phpmyadmin
   container_name: pma
   links:
     - db
   environment:
     PMA_HOST: db
     PMA_PORT: 3306
     PMA_ARBITRARY: 1
   restart: always
   ports:
     - 8081:80

volumes:
  db:
    driver: local
```

Fonte: Elaborado pelo autor

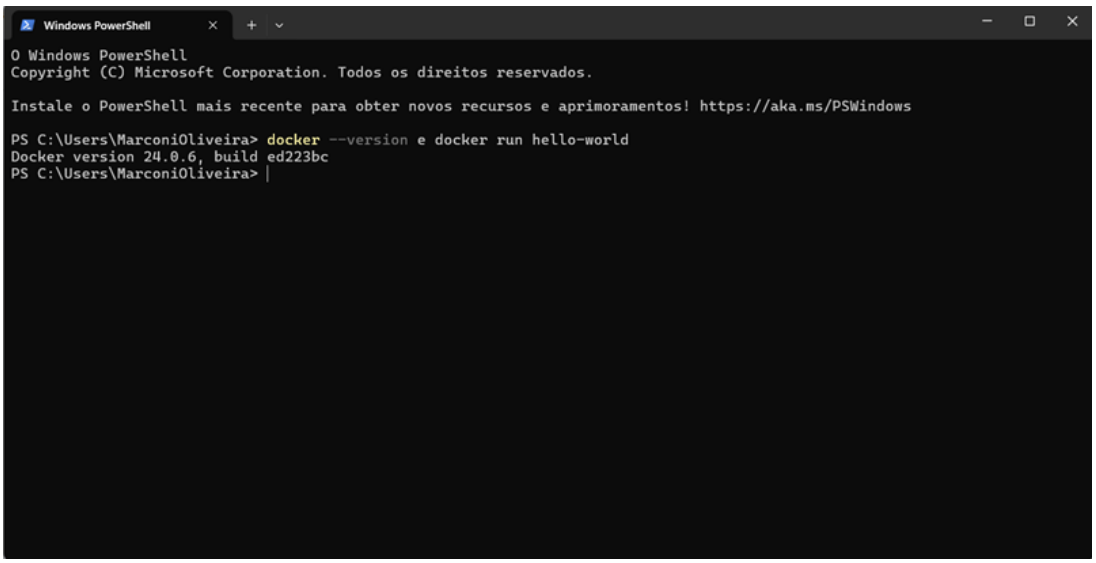
Após a composição do arquivo docker-compose.yml, o comando docker-compose up foi utilizado para iniciar e executar os containers definidos no arquivo. Quando esse comando é

executado, o Docker Compose cria e inicia os containers conforme as configurações especificadas. Caso os containers já existam, o comando apenas os iniciará. Por padrão, docker-compose up também exibe os logs dos containers no terminal, permitindo monitorar a saída em tempo real. Adicionalmente, o comando pode ser complementado com diversas opções, como -d para executar os containers em segundo plano (modo detached) ou –build para forçar a reconstrução das imagens antes de iniciar os containers. Tambem é possível escalar serviços facilmente através de comandos simples, como docker-compose up –scale <serviço>=<número>, permitindo ajustar a capacidade da aplicação conforme a demanda.

3.3 Instalação do Docker Desktop

Para executar uma imagem usando Docker Compose, é necessário ter o Docker Desktop instalado. É importante certificar-se de que a opção "*Install required WSL 2* components for Windows"esteja marcada durante a instalação, pois o Docker Desktop usa o WSL 2 para executar contêineres no Windows. A instalaçcao do Docker Desktop pode ser verificada por meio da execução de comandos como "docker –version"e "docker run *hello-world*", conforme mostrado na Figura 6.

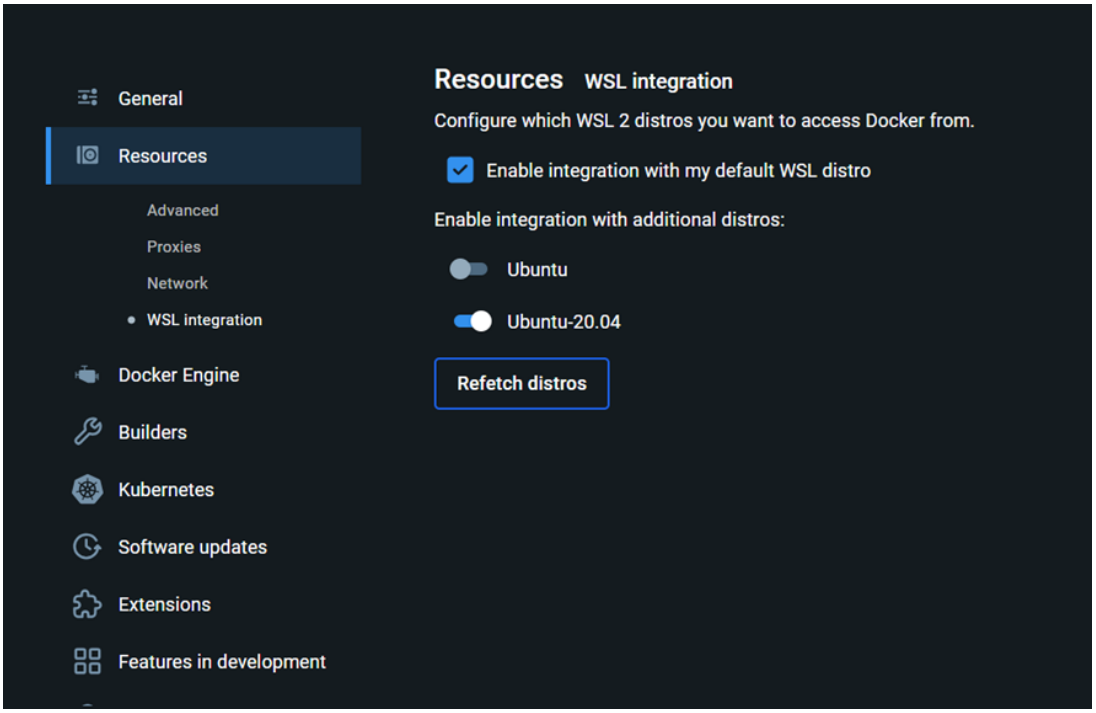
Figura 6 – Comando docker run hello-world



Fonte: Elaborado pelo autor

Após a instalação, foi feita a configuração para integrar o Docker com o WSL 2 (*Windows Subsystem for Linux*), otimizando o ambiente para o desenvolvimento com containers, como pode ser visto na Figura 7.

Figura 7 – Configuração WSL 2 (Windows Subsystem for Linux),



Fonte: Elaborado pelo autor

3.4 Migração para Containers Docker na Nuvem

A definição do layout dos containers e a escolha das tecnologias apropriadas são cruciais para assegurar uma arquitetura eficiente e com bom desempenho. Foram considerados diversos elementos, incluindo:

- **Comunicação entre componentes:** Garantir que os containers possam se comunicar de forma segura e eficiente.
- **Segurança do ambiente:** Implementar medidas de segurança adequadas para proteger os dados e serviços.
- **Ambientes de desenvolvimento e produção:** Configurar ambientes que suportem o ciclo completo de desenvolvimento, testes e implantação.

Esses aspectos precisam ser cuidadosamente planejados para facilitar a futura migração da aplicação para um ambiente de nuvem, garantindo flexibilidade e escalabilidade. Em nosso estudo de caso, as seguintes etapas fizeram parte do processo para migração dos containers docker para um ambiente de nuvem:

Configuração do Ambiente de Destino: Preparou-se o ambiente de destino, incluindo a configuração dos serviços de orquestração e a criação de redes e volumes necessários para suportar os contêineres.

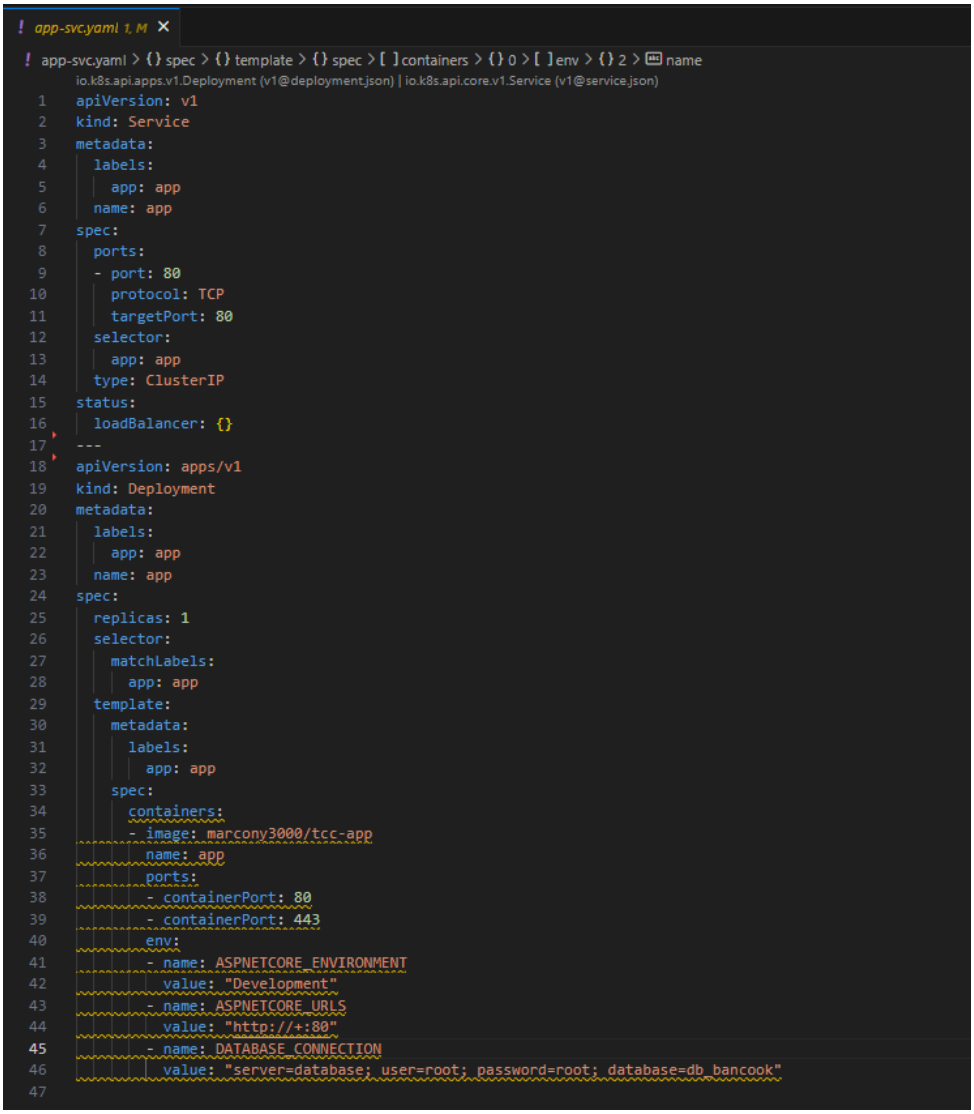
Geração de Imagens no Docker Hub: Foram construídas e registradas no Docker Hub as imagens necessárias para os diferentes componentes da aplicação, permitindo o compartilhamento e reutilização delas nos diversos ambientes.

Criação dos *Dockerfiles*: Foram criados *Dockerfiles* específicos para cada componente, contendo instruções detalhadas para a construção das imagens Docker, incluindo a configuração de variáveis de ambiente e dependências necessárias.

As duas imagens principais desenvolvidas foram:

- **Aplicação .Net Core:** Contém o código-fonte da aplicação e suas dependências.

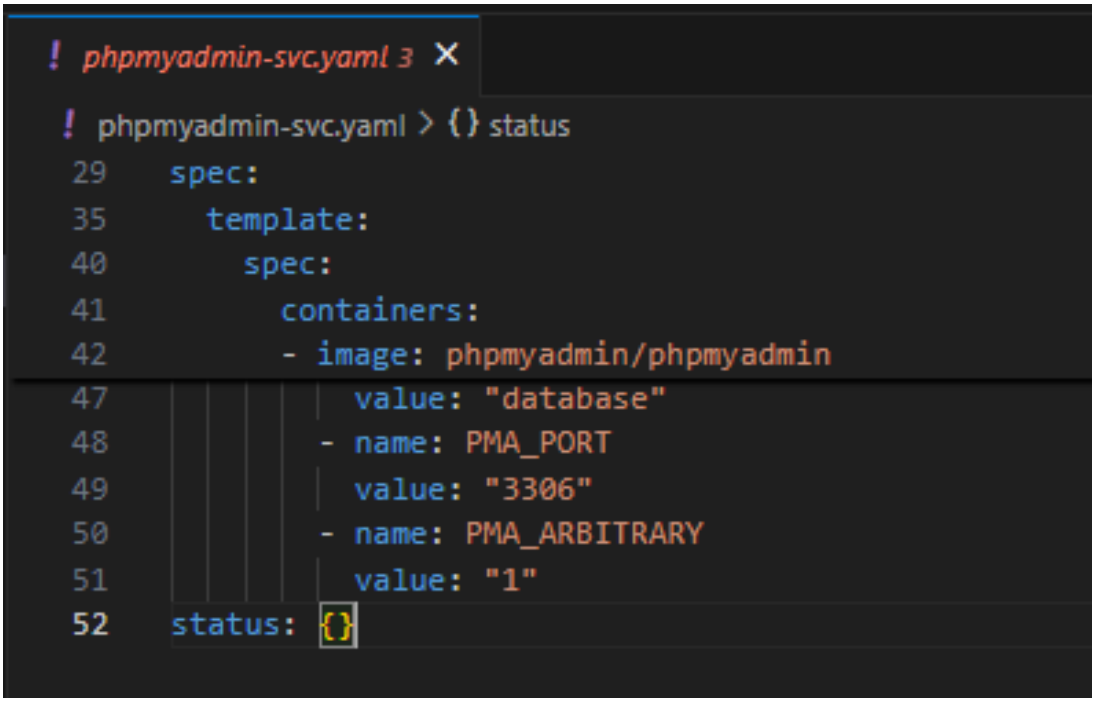
Figura 8 – Manifesto dockerfile da aplicação



Fonte: Elaborado pelo autor

- **Administração de Banco de Dados (phpMyAdmin):** Fornece uma interface gráfica para gerenciamento do banco de dados MySQL.

Figura 9 – Manifesto *dockerfile* de imagem pública "phpmyadmin" no arquivo docker file.yml



Fonte: Elaborado pelo autor

O próximo passo no estudo de caso é a configuração do banco de dados MySQL como serviço. A decisão de optar pelo banco de dados como serviço, em vez de usá-lo em um container, visa adotar de maneira mais robusta o conceito de nuvem.

Para configurar esse ambiente, utilizamos diversos recursos que asseguram a conectividade da aplicação conteinerizada com o banco de dados. Implementamos uma rede virtual que permite a comunicação segura entre os serviços, além de configurar variáveis de ambiente para armazenar credenciais de forma segura.

3.5 Migração para Containers Kubernetes e Implantação na Nuvem

A migração para o ambiente de orquestração de *containers* Kubernetes tem por objetivo permitir um gerenciamento facilitado de *clusters* de *containers*, em uma infraestrutura mais escalável e dinâmica.

3.5.1 Avaliação e Planejamento da Migração para a Nuvem

Baseado nos resultados obtidos nas etapas anteriores, foi realizada uma avaliação detalhada dos pré-requisitos para a migração completa para a nuvem. Isso incluiu:

- **Seleção do provedor de nuvem:** Optou-se por utilizar os serviços da AWS (*Amazon Web Services*), especificamente o EKS (*Elastic Kubernetes Service*), para orquestração dos containers.
- **Configuração de rede e segurança:** Implementação de VPC (*Virtual Private Cloud*) para isolamento de rede e uso do RDS (*Relational Database Service*) para gerenciamento do banco de dados MySQL, garantindo alta disponibilidade e segurança dos dados.
- **Configuração de recursos:** O provisionamento de instâncias de computação, configuração do orquestrador Kubernetes, e estabelecimento de políticas de segurança e armazenamento adequadas.

3.5.2 Configuração do EKS (Elastic Kubernetes Service)

- **Criação de um *Cluster* EKS:** Pode-se utilizar o AWS Management Console, a AWS CLI ou o AWS CloudFormation para criar um cluster EKS. A CLI é um método comum, no qual temos apenas de certificar de que o cluster tenha nós suficientes para suportar a aplicação conforme exemplo da Figura 10.

Figura 10 – Comando para criação do cluster EKS

```
eksctl create cluster --name nome-do-cluster --region us-east-1 --nodes 3
```

Fonte: Elaborado pelo autor

- **Configuração do Kubectl para o *Cluster* EKS:** Foi executado o comando o kubectl exemplificado no modelo da Figura 11 para se comunicar com o *cluster* EKS :

Figura 11 – Comando para configuração do Kubctl para o cluster EKS

```
aws eks --region us-east-1 update-kubeconfig --name nome-do-cluster
```

Fonte: Elaborado pelo autor

- **Criação de um Namespace (Opcional):** No Kubernetes, um *Namespace* é uma forma de organizar e dividir recursos dentro de um *cluster*. Os Namespaces são úteis para segmentar ambientes de produção, desenvolvimento e teste, ou para diferentes departamentos de uma empresa. exemplificado no comando conforme Figura 12:

Figura 12 – Comando para criação Namespace

```
kubectl create namespace nome-do-namespace
```

Fonte: Elaborado pelo autor

3.6 Implantação da Aplicação no Kubernetes

- **Criação de Manifests do Kubernetes:** Foram criados os arquivos de configuração YAML para *Deployment, Service, e Ingress*, se necessário mostrado na Figura 13.

Figura 13 – Arquivos Manifests para Kubernetes

Fonte: Elaborado pelo autor

- **Aplicação dos Manifests no Cluster:** Os arquivos manifest YAML podem ser aplicados usando o comando kubectl conforme Figura 14.

Figura 14 – Comandos para executar a aplicação do Manifest

```
kubectl apply -f deployment.yaml -n nome-do-namespace
kubectl apply -f service.yaml -n nome-do-namespace
```

Fonte: Elaborado pelo autor

- **Verificação do Status:** Podemos verificar se os *pods*: Que é menor e mais básica unidade de computação que pode ser criada e gerenciada. Como mostrado no exemplo da Figura 15:

Figura 15 – Comando para verificação de execução dos pods

```
kubectl get pods -n nome-do-namespace
```

Fonte: Elaborado pelo autor

- **Exposição da Aplicação:** Para a exposição da aplicação utilizamos o serviço de ELB (*Elastic Load Balancer*) configurando um Service ou um Ingress para expor a aplicação externamente.
- **Acesso à Aplicação:** Após o provisionamento do *Load Balancer*, obtém-se o endereço IP público ou DNS e pode-se acessar então a aplicação.

3.7 Otimização e Escalabilidade

Com a aplicação operando na nuvem, foram realizadas otimizações adicionais para aprimorar o desempenho e suportar crescimentos na demanda. As medidas implementadas incluíram:

- **Utilização de serviços gerenciados:** Aproveitamento de serviços gerenciados da AWS para simplificar a administração e aumentar a confiabilidade.
- **Balanceamento de carga:** Implementação de balanceadores de carga para distribuir o tráfego de forma equilibrada entre as instâncias da aplicação.
- **Dimensionamento automático:** Configuração de políticas de *auto-scaling* para ajustar automaticamente os recursos computacionais de acordo com as necessidades em tempo real.

3.8 Problemas e Desafios

Migrar uma aplicação para Kubernetes, especialmente usando serviços como Amazon EKS (*Elastic Kubernetes Service*) e Amazon RDS (*Relational Database Service*) com MySQL, pode envolver diversos desafios e problemas. Aqui estão alguns dos principais:

1. Complexidade na Configuração
    - **Configuração do Docker Compose:** A definição de serviços, redes e volumes no arquivo docker-compose.yml pode ser complexa, especialmente em aplicações que possuem múltiplas dependências pois cada serviço representa um *container* que pode executar uma parte da aplicação (banco de dados, *backend*, *frontend*, *cache* dentre outros.). A complexidade pode surgir ao definir como os serviços se conectam, as variáveis de ambiente, portas e dependências entre eles. Em aplicações maiores, pode haver dependências complexas, como múltiplas instâncias de banco de dados, comunicação entre microserviços, ou até a necessidade de balanceamento de carga entre instâncias de um mesmo serviço. Definir redes no Docker Compose é fundamental para controlar a comunicação entre serviços. A configuração padrão conecta os *containers* automaticamente, mas ao lidar com aplicações mais complexas, pode ser necessário configurar várias redes para isolar certos serviços. Para a configuração dos volumes podem ser usados para evitar perda de dados em caso de reinicialização de *containers*. Também são usados para persistir dados e compartilhar dados entre *containers*. Isso pode ser particularmente importante ao lidar com bancos de dados ou arquivos temporários que precisam ser compartilhados entre serviços. O desafio é garantir que os volumes corretos sejam montados e gerenciados corretamente.

- **Configuração do EKS:** O EKS facilita o gerenciamento de *clusters* Kubernetes na AWS, mas sua configuração envolve diversos aspectos que podem aumentar a complexidade, como por exemplo a a configuração da quantidade de nós (workers) para lidar com a carga da aplicação. ajustar as métricas de escalabilidade, como a utilização de CPU ou memória, para garantir que os pods sejam escalados adequadamente conforme a demanda. Parâmetros necessários para garantir que os nós do EKS possam se comunicar com outros serviços da AWS.
- **Integração com RDS:** Garantir que os serviços possam se conectar ao RDS corretamente e que as regras de segurança e permissões estejam adequadas. Para garantir esse acesso, é necessário configurar certos serviços, como a VPC e suas sub-redes, com regras de roteamento apropriadas, além de definir regras de segurança que permitam o tráfego, como no caso do MySQL, que utiliza por padrão a porta 3306.

2. Gerenciamento de Estado

   - **Gerenciamento de Estado**: Aplicações que dependem de armazenamento persistente podem enfrentar dificuldades na gestão do estado entre contêineres, necessitando de soluções adequadas para garantir a integridade dos dados.
   - **Persistência de Dados:** Manter a consistência e a persistência dos dados pode ser um desafio, especialmente quando se trata de aplicações que exigem alta disponibilidade e recuperação de desastres.
   - ***Backup* e Recuperação:** Configurar e testar *backups* regulares e estratégias de recuperação para o banco de dados MySQL são cruciais.

3. Escalabilidade

   - **Desempenho:** A utilização de contêineres pode introduzir uma sobrecarga de virtualização que impacta a performance das aplicações, especialmente em cenários de alta demanda. Essa sobrecarga ocorre porque os contêineres compartilham os recursos do sistema host, o que pode gerar latência e limitar o uso eficiente de CPU, memória e rede. Para mitigar esse impacto e garantir um desempenho satisfatório, é fundamental adotar otimizações específicas, como ajustes finos na alocação de recursos, utilização de tecnologias de orquestração, como Kubernetes, e o uso de práticas recomendadas de monitoramento e balanceamento de carga.
   - **Dimensionamento:** Gerenciar a escalabilidade de pods e serviços pode ser complicado, especialmente quando se lida com cargas de trabalho variáveis.
   - **Performance do Banco de Dados:** Garantir que o MySQL possa lidar com o aumento da carga enquanto é escalado pode exigir ajustes na configuração e monitoramento contínuo.

4. Segurança

- **Segurança da Comunicação:** Assegurar que a comunicação entre os *pods* e o banco de dados seja segura e criptografada.
- **Gerenciamento de Segredos:** Gerenciar e proteger credenciais e segredos dentro do Kubernetes usando ferramentas como o AWS *Secrets Manager*.

5. Configuração e *Deploy*

   - **Configuração de Aplicações:** Garantir que a configuração da aplicação (como variáveis de ambiente e configurações específicas de serviço) esteja corretamente definida no Kubernetes.
   - ***Pipeline* de CI/CD:** Configurar pipelines de integração e entrega contínua (CI/CD) para automatizar o *deploy* pode ser desafiador e requer integração com ferramentas como o AWS *CodePipeline* ou *Jenkins*.

6. Monitoramento e Logs

   - **Monitoramento de *Cluster*:** Implementar soluções de monitoramento para o EKS e para a aplicação em execução, como o AWS *CloudWatch* ou *Prometheus*.
   - **Logs:** Centralizar e gerenciar logs de aplicação e do cluster pode exigir configuração de ferramentas como o ELK *Stack* ou *Fluentd*.

7. Treinamento e Conhecimento

   - **Treinamento:** Garantir que o conhecimento suficiente para estar familiarizado com Kubernetes, EKS e a utilização de práticas recomendadas para o gerenciamento de contêineres e serviços. Cada uma dessas áreas pode ter suas próprias complexidades e pode exigir planejamento cuidadoso, testes e, muitas vezes, a ajuda de especialistas.

4

Conclusão e Recomendação de Trabalhos Futuros

4.1 Conclusão

Neste trabalho, foi realizada a migração de uma aplicação de conta bancária originalmente desenvolvida em .Net Core. Essa aplicação, estruturada segundo o padrão de design MVC (*Model-View-Controller*), para o ambiente Kubernetes em uma infraestrutura de nuvem. A decisão de realizar essa migração foi fundamentada na busca por maior flexibilidade, escalabilidade e resiliência operacional, benefícios amplamente associados ao uso do Kubernetes. Entretanto, para garantir o sucesso desse processo, foi necessário um planejamento estratégico detalhado cuidadoso.

A abordagem adotada envolveu, primeiramente, um estudo aprofundado para obter o conhecimento necessário sobre as práticas de orquestração de *containers*. Em seguida, a configuração de medidas de segurança robustas foi priorizada, abordando a segurança tanto no nível da aplicação quanto no nível da infraestrutura. Por fim, foi estabelecido um monitoramento contínuo da aplicação e do ambiente Kubernetes, permitindo a identificação proativa de possíveis problemas e a manutenção da alta disponibilidade e desempenho da aplicação.

Kubernetes provê uma base sólida para a orquestração de *containers*, mas o sucesso da migração depende não apenas da sua implementação técnica, mas também da capacidade de gerenciar a complexidade operacional que acompanha esse tipo de ambiente. O êxito desta iniciativa ficou evidente na melhoria da escalabilidade da aplicação e na resiliência proporcionada pelo Kubernetes, resultando em um ambiente de produção mais robusto, eficiente e preparado para demandas futuras.

4.2 Dificuldades encontradas

- Durante o processo de migração do banco de dados como desafio na conversão dos dados, surgiram inconsistências devido às diferenças nos tipos de dados suportados pelos dois sistemas de gerencia- mento de banco de dados. O MySQL oferece uma variedade maior de tipos de dados, como **TINYINT, TIMESTAMP, MEDIUMBLOB, MEDIUMTEXT e DATETIME**, enquanto o SQLite suporta apenas **INTEGER, REAL, TEXT e BLOB**, para isso tivemos que realizar uma pesquisa para a conversão dos dados e também ajustes foram realizados.

- Outro desafio foi garantir a persistência dos dados e geranciamento de volumes foram encontrados desafios com a necessidade da criação de volumes no processo inicial executado no Doker compose e a escolha do uso do banco de dados como serviço para a migração para a nuvem seguindo os conceitos da computação na nuvem.

- Na migração para o serviço de EKS (*Elastic Container Service*) obrigatoriamente foi realizado a configuraçaão de variaveis de ambientes para garantir a comunicação entre os serviços de aplicação, gerenciamento e banco de dados.

- Para garantir a alta disponibilidade e o acesso externo, houve a necessidade da implantação de um *load balancer*. Ele distribui o tráfego de rede ou aplicação entre vários contêineres, evitando que algum fique sobrecarregado. Isso melhora o desempenho e previne falhas. Para configurar o acesso externo de forma eficaz, é necessário seguir boas práticas e possuir conhecimento adequado, garantindo que o balanceamento de carga seja eficiente e que os serviços permaneçam disponíveis e estáveis, mesmo durante picos de demanda.

4.3 Recomendações para Trabalhos Futuros

Para trabalhos futuros, sugere-se a extensão deste guia com abordagem na migração para diferentes provedores de nuvens públicas, além da consideração de diversos tipos de orquestradores. Outro foco possível para trabalhos futuros está na identificação de soluções para os desafios enfrentados durante este processo, de forma a garantir sucesso na migração mesmo envolvendo casos de aplicações mais complexas ou em maior escala.

Referências

CARVALHO, M. A. V. de; SANTOS, T. F. R. Adoção de contêineres em aplicações web: Uma revisão da literatura. *Revista Brasileira de Computação Aplicada*, v. 11, n. 1, p. 45–59, 2019. Citado na página 12.

CHAVES, B. D.; CASTRO, B. G. D. d.; NASCIMENTO, L. J. S. Estudo comparativo entre cloud computing e infraestrutura de rede local. 2021. Citado na página 14.

COSTA, L. F.; LIMA, R. P. Portabilidade de aplicações em contêineres: Desafios e oportunidades. *Revista Brasileira de Sistemas de Informação*, v. 15, n. 2, p. 33–47, 2021. Citado na página 20.

COUTINHO, L. R.; NEVES, H. P. O. d.; LOPES, L. C. Abordagens sobre computação na nuvem: uma breve revisão sobre segurança e privacidade aplicada a e-saúde no contexto do programa conecte sus e rede nacional de dados em saúde (rnds). *Brazilian Journal of Development*, v. 7, n. 4, p. 35152–35170, 2021. Citado na página 17.

DAVIS, M. *Understanding Platform as a Service*. [S.l.]: Tech Insights, 2023. Citado na página 16.

DOCKER. *Docker overview*. 2023. Disponível em: <https://docs.docker.com/get-started/docker-overview/>. Citado 2 vezes nas páginas 18 e 19.

HUSSEIN, M. K.; MOUSA, M. H.; ALQARNI, M. A. A placement architecture for a container as a service (caas) in a cloud environment. *Journal of Cloud Computing*, Springer, v. 8, p. 1–15, 2019. Citado 2 vezes nas páginas 16 e 17.

IMESH, G. *The evolution of linux containers and their future*. 2016. Urlhttps://dzone.com/articles/evolution-of-linux-containers-future. Citado na página 18.

JúNIOR, E. T. P.; GOMES, A. S. D. Cloud computing: Um estudo sobre a adoção em pequenas e médias empresas. In: *Anais do Simpósio Brasileiro de Sistemas de Informação*. [S.l.: s.n.], 2020. p. 123–136. Citado na página 12.

MOÇO, P. A. B.; CUNHA, P. Análise da implementação da indústria 4.0 nas gestões de qualidade e de conhecimento. *Boletim do Gerenciamento*, v. 16, n. 16, p. 40–48, 2020. Citado na página 11.

MORAES, J. et al. Implementação de um cluster kubernetes com a plataforma dojot para aplicações de internet das coisas. In: SBC. *Anais do XLVIII Seminário Integrado de Software e Hardware*. [S.l.], 2021. p. 1–8. Citado na página 19.

OLIVEIRA, B. T. de; JR, M. P. R.; ALBUQUERQUE, J. P. de. Implantação de um sistema integrado de gestão no modelo software as a service (saas): um estudo de caso em uma pequena empresa de engenharia. *Revista Eletrônica de Sistemas de Informação*, v. 9, n. 1, 2010. Citado na página 16.

PAHL, C. Containerization and the paas cloud. *IEEE Cloud Computing*, IEEE, v. 2, n. 3, p. 24–31, 2015. Citado na página 18.

PAULO, K. P. d. Avaliação de desempenho para elasticidade de ambientes conteinerizados: estudo experimental e de modelagem do kubernetes. Pós-Graduação em Ciência da Computação, 2021. Citado na página 15.

Penso. *Quais os diferentes tipos de Cloud Computing?* 2022. Acessado em: 22 de setembro de 2023. Disponível em: <https://www.penso.com.br/quais-os-diferentes-tipos-de-cloud-computing/>. Citado na página 15.

RAMESH, G.; LOGESHWARAN, J.; ARAVINDARAJAN, V. A secured database monitoring method to improve data backup and recovery operations in cloud computing. *BOHR International Journal of Computer Science*, v. 2, n. 1, p. 1–7, 2022. Citado na página 17.

RASHID, A.; CHATURVEDI, A. Cloud computing characteristics and services: a brief review. *International Journal of Computer Sciences and Engineering*, v. 7, n. 2, p. 421–426, 2019. Citado na página 16.

SILVA, A. R.; ALMEIDA, T. C. Virtualização e sua importância para a computação em nuvem. In: *Anais do Simpósio Brasileiro de Computação em Nuvem*. [S.l.: s.n.], 2020. p. 78–92. Citado na página 17.

SILVA, F. H. R. Avaliação de desempenho de contêineres docker para aplicações do supremo tribunal federal. 2017. Citado na página 16.

SILVA, J. Modelos de armazenamento na nuvem: Uma análise comparativa. *Journal of Cloud Computing*, v. 14, n. 1, p. 25–40, 2022. Citado na página 16.

TAURION, C. *Cloud computing-computação em nuvem*. [S.l.]: Brasport, 2009. Citado na página 14.

VERAS, M. *Cloud Computing: Nova Arquitetura da TI*. Rio de Janeiro: Brasport, 2012. Citado na página 14.

YIN, R. K. *Case Study Research: Design and Methods*. 5. ed. Thousand Oaks, CA: Sage Publications, 2015. Citado na página 21.