

UNIVERSIDADE FEDERAL DO MARANHÃO

Curso de Ciência da Computação

Felipe Breno Aragão Chaves

**O modelo DevOps para execução de integração
contínua e entrega contínua em ambiente de
nuvem**

São Luís - MA

2021

Felipe Breno Aragão Chaves

O modelo DevOps para execução de integração contínua e entrega contínua em ambiente de nuvem

Monografia apresentada ao curso de Ciência da Computação da Universidade Federal do Maranhão, como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof^o. Dr^o. Mário Antonio Meireles Teixeira

São Luís - MA

2021

Ficha gerada por meio do SIGAA/Biblioteca com dados fornecidos pelo(a) autor(a).
Diretoria Integrada de Bibliotecas/UFMA

Chaves, Felipe Breno Aragão.

O modelo DevOps para execução de integração contínua e entrega contínua em ambiente de nuvem / Felipe Breno Aragão Chaves. - 2021.

51 f.

Orientador(a): Mário Antonio Meireles Teixeira.

Monografia (Graduação) - Curso de Ciência da Computação, Universidade Federal do Maranhão, UFMA, 2021.

1. DevOps. 2. Docker. 3. Entrega Contínua. 4. Integração Contínua. 5. Nuvem. I. Teixeira, Mário Antonio Meireles. II. Título.

Felipe Breno Aragão Chaves

O Modelo DevOps para execução e de Integração Contínua e Entrega Contínua em ambiente de nuvem

Monografia apresentada ao curso de Ciência da Computação da Universidade Federal do Maranhão, como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Trabalho de Monografia. São Luís - MA, 03 de maio de 2021:

Mário Antonio Meireles Teixeira
Orientador
Universidade Federal do Maranhão

Prof. Dr. Geraldo Braz Junior
Examinador Interno
Universidade Federal do Maranhão

**Prof. Me. Carlos Eduardo Portela
Serra de Castro**
Examinador Interno
Universidade Federal do Maranhão

São Luís - MA
2021

Agradecimentos

Primeiramente gostaria de agradecer à minha família pelo apoio durante toda essa jornada e por não terem me deixado desistir nos momentos de dificuldade. A todos amigos conquistados nesses longos anos. Ao meu orientador Mário, pela paciência, disponibilidade e excelente orientação. Aos professores, por todos os ensinamentos e a toda equipe da coordenação pela ajuda e direcionamento. E por fim, a minha filha Valentina, carinhosamente chamada de Popota, que passou a ser minha maior motivação diária de crescer depois que veio ao mundo.

Resumo

Esse projeto visa elucidar o modelo de desenvolvimento utilizando Integração Contínua e Entrega Contínua, explanando sua definição e aplicando a teoria na prática através do desenvolvimento de uma aplicação de exemplo. Além disso, serão utilizadas ferramentas amplamente usadas no mercado como o Gitlab, para que possa ser possível o versionamento de código e a aplicação dos conceitos de DevOps.

Palavras-chaves: Entrega Contínua, Integração Contínua, Gitlab, DevOps.

Abstract

This project aims to elucidate the development model that uses Continuous Integration and Continuous Delivery, explaining its definition and applying a theory about the practice of developing an example application. In addition, the tools used are often used in the market, such as Gitlab, so that code versioning and DevOps concepts can be applied.

Keywords: Continuous Integration, Continuous Delivery, Gitlab, Devops.

Lista de ilustrações

Figura 1 – Tabela de custos de hadwares	18
Figura 2 – Tabela de custos de softwares	18
Figura 3 – Tabela de custos de despesas mensais	19
Figura 4 – Tabela de resumo de investimento em hospedagem interna	19
Figura 5 – Tabela de custos de ambiente na nuvem	19
Figura 6 – Tabela de resumo de custo de todo ambiente na nuvem	20
Figura 7 – Tabela de resumo de custo de todo ambiente na nuvem	20
Figura 8 – Navio cargueiro	22
Figura 9 – Diferenças entre Máquina virtual e contêiner	23
Figura 10 – Instalação Docker Desktop para Windows Home	24
Figura 11 – Instalação Docker Desktop para Windows Home Boas vindas	24
Figura 12 – Instalação Docker Desktop para Windows Home tutorial	24
Figura 13 – Comando <i>docker build</i> sendo executado	26
Figura 14 – Comando <i>docker run</i> sendo executado	27
Figura 15 – Executando o <i>bash</i> do contêiner MySQL	28
Figura 16 – Tela inicial de login da aplicação	29
Figura 17 – Tela inicial: login inválido	29
Figura 18 – Tela de registro da aplicação	30
Figura 19 – Tela de registro da aplicação com as validações de campos obrigatório	30
Figura 20 – Tela de registro da aplicação após registrado um usuário com sucesso	31
Figura 21 – Tela de acesso restrito a usuários cadastrados	31
Figura 22 – Tabelas criadas no banco de dados	32
Figura 23 – Modelo ER das tabelas user, role e user_role	32
Figura 24 – Opção SQL no menu lateral para iniciar a configuração	33
Figura 25 – Opção nova instância para começar a criação da nova instância do banco de dados	33
Figura 26 – SGBDs disponíveis para utilização do serviço SQL da Google Cloud	34
Figura 27 – Definição de nome da instância, senha do usuário root, região e zona	34
Figura 28 – Adicionado IP para autorizar acesso a instância do banco de dados	35
Figura 29 – Criar novo banco de dados	36
Figura 30 – Definido nome para o banco de dados	36
Figura 31 – Banco de dados criado	36
Figura 32 – Visão Geral - IP da instância do banco de dados	37
Figura 33 – Teste de conexão com o banco criado pela DataGrip	37
Figura 34 – Lista de arquivos do projeto java	40

Figura 35 –Lista de imagens docker	40
Figura 36 –Criação do contêiner login-app	41
Figura 37 –contêiners ativos: visualização pela interface gráfica do Docker Desktop .	41
Figura 38 –Resumo de funcionamento de uma publicação de um aplicação no Kubernetes	42
Figura 39 –Instalação concluída do Google Cloud SDK Shell	43
Figura 40 –Lista de pipelines já executado na aplicação	46
Figura 41 –Ciclo de vida resumido de todas as etapas da publicação	48

Lista de abreviaturas e siglas

CD	<i>Continuos Delivery</i>
CI	<i>Continuos Integration</i>
DevOps	<i>Developments and Operations</i>
GCR	<i>Google Container Registry</i>
GKE	<i>Google Kubernetes Engine</i>
kubectl	<i>Kubernetes Command Line Tool</i>
OS	<i>Operating System</i>
SDK	<i>Software Development Kit</i>
URL	<i>Uniform Resource Locator</i>
VM	<i>Virtual Machine</i>
WSL	<i>Windows Subsystem for Linux</i>

Sumário

1	Introdução	9
1.1	Contextualização	9
1.2	Justificativa da escolha do tema	10
1.3	Metodologia	10
1.4	Trabalhos Relacionados	11
1.5	Objetivos	11
1.5.1	Objetivos Específicos	11
1.6	Organização do Trabalho	12
2	Fundamentação Teórica	13
2.1	DevOps	13
2.2	Integração contínua e Entrega contínua (CI / CD)	14
2.3	Ambientes On-Cloud x Ambientes On-Premise	15
2.3.1	Infraestrutura	15
2.3.2	Operacional	16
2.3.3	Financeiro	17
3	Integração Contínua e Entrega Contínua na prática	21
3.1	Docker	21
3.1.1	Diferenças entre Máquina Virtual e contêiner	22
3.1.2	Instalação do Docker	23
3.1.3	Contêiner Docker	24
3.2	A aplicação web	28
3.2.1	Configurando o banco de dados na Google Cloud	32
3.2.2	Contêinerização da aplicação web	37
3.3	Kubernetes, o orquestrador do Docker	42
3.3.1	Instalação Kubernetes	42
3.3.2	Implantar aplicação no GKE	43
3.4	Gitlab e automatização da publicação	45
3.4.1	Criação do pipeline	46
3.5	Ciclo de vida das aplicações usando CI/CD	48
4	Conclusão	50
	Referências	51

1 Introdução

1.1 Contextualização

A ideia do cloud computing surgiu em 1950 quando mainframes foram disponibilizados para universidades e corporações. O mainframe é um computador de grande porte feito especialmente para processar grandes volumes de dados. Esses computadores possuíam uma grande infraestrutura de hardware e era instalado no que poderia ser chamado de sala de servidores. Vários usuários podiam acessar o mainframe através de “terminais burros” - estações com a única função de enviar comandos aos mainframes e mostrar na tela o seu resultado. Devido ao custo de compra e manutenção de mainframes, uma organização não poderia pagar por um mainframe para cada usuário. Então, tornou-se prática permitir que vários usuários compartilhassem o acesso ao mesmo computador, através de terminais, compartilhando recursos de armazenamento e processamento oferecido por essas máquinas. Ao permitir o acesso ao mainframe compartilhado, uma organização obteria um retorno melhor de seu investimento (IBM, 2017).

O início da mudança desse paradigma começa em 1970. A IBM, maior empresa de tecnologia da época, lança um sistema operacional chamado VM que permitia que os administradores em seus sistemas de mainframe tivessem vários sistemas virtuais, ou seja, “máquinas virtuais (VMs)” em um único nó físico. Esse sistema operacional levou o acesso compartilhado de um mainframe ao próximo nível. O sistema operacional permitia que vários ambientes de computação distintos vivessem no mesmo ambiente físico. A "virtualização" tornou-se um grande recurso de tecnologia e acelerou a evolução da computação no que se refere à comunicação.

Para acompanhar tais mudanças, as empresas de telecomunicações da década de 90, que historicamente ofereciam apenas conexões dedicadas de dados ponto a ponto, começaram a oferecer conexões de rede privada virtualizadas. Ou seja, em vez de construir uma infraestrutura física para permitir que mais usuários tivessem suas próprias conexões, as empresas de telecomunicações forneceram aos usuários acesso compartilhado à mesma infraestrutura física com a mesma qualidade que serviços dedicados que ofereciam, porém a um custo muito menor. Essa mudança permitiu que as empresas de telecomunicações mudassem o tráfego conforme necessário, levando a um melhor equilíbrio da rede e a um maior controle sobre o uso da largura de banda.

Como a internet se tornou mais acessível, o natural era que a virtualização atingisse o ambiente online. Nesse momento, o setor de comunicação começou a usar o símbolo da nuvem para denotar o ponto de demarcação entre o que o provedor de recursos era

responsável e o que os usuários eram responsáveis. A computação em nuvem superou seus limites e cobriu ainda a parte dos servidores bem como a infraestrutura de rede.

1.2 Justificativa da escolha do tema

O desenvolvimento de software se move rapidamente e exige mais dos desenvolvedores do que nunca. Tal necessidade sugere a criação de ferramentas e conceitos para ajudar os desenvolvedores a fornecer valor de maneira mais rápida e transparente. A entrega contínua e integração contínua, conhecida comumente como CI / CD (*Continuous Integration / Continuous Delivery*), vieram para permitir que os desenvolvedores entreguem código com mais frequência e confiabilidade. Embora essa ideia pareça mais adequada para software para empresas, CI / CD é uma prática valiosa, independentemente do mercado precisar criar aplicativos corporativos em grande escala ou simplesmente tentar colocar algum projeto em funcionamento. Os métodos convencionais de desenvolvimento e entrega de software estão rapidamente se tornando obsoletos. Historicamente, na era da agilidade, a maioria das empresas implantavam ou enviavam softwares em versões mensais, trimestrais, bianuais ou mesmo anuais. Agora, no entanto, semanalmente, diariamente e até várias vezes por dia é a norma para entrega de resultados. Isso é real, já que a computação na nuvem está dominando (NAIR, 2018).

1.3 Metodologia

Nesse trabalho será mostrado como integrar os conceitos de entrega contínua e integração contínua para utilizar da melhor maneira possível os recursos que os ambientes on-cloud oferecem. Então será mostrado na prática como reduzir custos de infraestrutura e de rede de aplicações, escalar as aplicações de acordo com a demanda requisitada pelos usuários do sistema, simplificar a integração de mudanças de evolução ou de manutenção do produto e mostrar a rapidez para ser integrada ao código do produto final.

Para alcançar o objetivo esperado, será usado a ferramenta GitLab para o versionamento dos códigos fontes dos exemplos utilizados aqui. Como padrão para o modelo de desenvolvimento o Trunk-based development será utilizado, o que significa que nesse modelo as mudanças no código fonte do projeto sempre são integrados a um único ramo principal (*branch master*) do projeto. A aplicação será executada dentro de uma estrutura criada no Docker (ferramenta que auxiliará na implementação do CI / CD). Por fim há o Kubernetes, ele nos entregará um ambiente de fácil manutenção das aplicações dentro do Docker, de fácil monitoramento e totalmente escalável, o que demonstra uma estrutura moderna utilizada pelo mercado atual.

1.4 Trabalhos Relacionados

Em (BRAGA, 2015) é feita uma pesquisa a respeito das práticas DevOps e tem como um dos objetivos identificar e entender como está o movimento DevOps. Lá chega-se a conclusão de que a maioria dos estudos encontrados se concentra em integração, desenvolvimento, entrega e implantação contínua, em virtualização e nuvem, automação de infraestrutura, ferramentas e princípios DevOps. O artigo demonstra que o DevOps é uma área muito grande para ser explorada mas que falta uma certa padronização quando se trata da definição das áreas de pesquisa.

O trabalho (SENAPATHI JIM BUCHAN, 2018) apresenta um caso de estudo que explora a implementação do DevOps numa organização da Nova Zelândia em um produto que está em produção. Tal estudo envolveu engenheiros de software que monitoravam continuamente os reflexos das práticas e dos princípios implementados pelo DevOps. Para tal estudo de caso o DevOps trouxe benefícios significantes como o incremento do número de implantações (o número passou de 30 por mês para 120 por mês) e também uma evolução natural na comunicação entre o departamento de tecnologia com o pessoal de operações. Baseado nesse artigo o presente trabalho irá aplicar os conceitos de DevOps em um projeto menor mas que demonstra o impacto da mudança cultural.

1.5 Objetivos

O objetivo desse trabalho é propor a implantação de uma arquitetura que utilize os conceitos de CI / CD em ambientes on-cloud.

1.5.1 Objetivos Específicos

O objetivo específico proposto será de abordar a construção dessa infraestrutura de desenvolvimento.

- Explanar as vantagens em abordar os conceitos de Devops, CD / CI e ambientes on-cloud.
- Construir uma infraestrutura utilizando as principais ferramentas do mercado nessa área, Gitlab para controle de versão de código fonte e automatização de deploy, o Docker para criação das imagens que conterão as builds do projeto e o Kubernetes para orquestrar as imagens criadas pelo Docker.
- Fazer um estudo de viabilidade econômica comparando a infraestrutura construída aqui com os modelo on-premise.

1.6 Organização do Trabalho

Essa monografia está dividida em quatro capítulos, sendo o primeiro a introdução, que contém a contextualização, justificativa da escolha do tema, a metodologia, os trabalhos relacionados e os objetivos, tanto geral quanto específico. O capítulo dois irá abordar sobre as *Fundamentação Teórica* do trabalho, explorar os conceitos do DevOps, Integração contínua e Entrega contínua e os Ambientes On-Cloud. No terceiro capítulo, será explorado a *Integração Contínua e a Entrega Contínua na prática* e irá explicar as tecnologias utilizadas e como implementá-las. Por fim, o capítulo quatro é constituído das conclusões e considerações finais a respeito do trabalho.

2 Fundamentação Teórica

2.1 DevOps

O conceito de *DevOps* foi descrito como ambíguo e difícil de definir (SENAPATHI JIM BUCHAN, 2018). Neste capítulo serão abordadas as definições e as aplicações dessa nova forma de se trabalhar com desenvolvimento de softwares.

Existem duas principais visões, a primeira considera que é como um cargo específico onde requer uma combinação de habilidades de desenvolvimento de software e operações de TI. A segunda visão argumenta que o *DevOps* nasceu para atender uma necessidade emergente do desenvolvimento de software contemporâneo. O fato é que ambos os conceitos estão certos e precisam estar interligados um ao outro. Na tentativa de resolver esse problema, aqueles que o consideravam como um cargo específico, tem se esforçado para alcançar um entendimento claro das definições e caracterização do cargo e definido também suas práticas associadas. Já aqueles que consideram que o *DevOps* surgiu para atender uma necessidade atual, se esforçam para entender os benefícios do mesmo e os desafios que este trás para os projetos. Por exemplo, o *DevOps* é descrito como um movimento cultural que permite dentro de uma organização concentrar-se no desenvolvimento rápido com quatro características definidas: comunicação aberta, alinhamento de incentivos e responsabilidades, respeito e confiança. Essa cultura resultaria na melhoria da colaboração entre o desenvolvimento e a operação, a fim de acelerar a entrega de mudanças. Mas também existe o fato de que os aspectos culturais por si só não podem ser a definição das características do *DevOps*, mas que agem como facilitadores para oferecer suporte a um conjunto de recursos do processo de engenharia.

Em resumo, embora o primeiro fluxo de pesquisa tenha se concentrado amplamente na compreensão dos conceitos e das definições características do *DevOps*, o segundo fluxo focou-se na compreensão dos benefícios e desafios da adoção de algumas dessas práticas. Portanto, é pertinente entender que os desenvolvedores experientes de softwares adotam uma abordagem gradual e customizada do DevOps quando irão fabricar um produto real. Acreditamos que as lições aprendidas com sua implementação em um contexto real de desenvolvimento de software são inestimáveis, pois poucos estudos foram publicados. Dado o exposto, será usada a definição de *DevOps* desenvolvida por (SENAPATHI JIM BUCHAN, 2018): "Um conjunto de recursos de processos de engenharia suportados por facilitadores culturais e tecnológicos. Os recursos definem processos que uma organização deve ser capaz de executar, enquanto permitem uma maneira fluente, flexível e eficiente de se trabalhar"(SENAPATHI JIM BUCHAN, 2018) como um guia para investigar a implementação do *DevOps* na prática real.

Em resumo, o principal objetivo em implementar o DevOps é conseguir desenvolver um ciclo de entrega contínua e integração contínua, ou seja, automatizar seus passos desde o desenvolvimento do código do programador até a integração dessa mudança na aplicação em produção. Os conceitos de integração contínua (*Continuous Integration* - CI) e entrega contínua (*Continuous Delivery* - CD) serão abordados na seção a seguir.

2.2 Integração contínua e Entrega contínua (CI / CD)

Historicamente, o padrão da maioria das empresas era implantar ou enviar softwares em versões mensais, trimestrais, bianuais ou mesmo anuais. Agora, no entanto, semanalmente, diariamente e até várias vezes por dia é a norma. Muitas vezes, usuários nem percebem que os softwares que equipes de desenvolvimento fazem manutenção estão mudando.

Para entendermos melhor o que isso significa devemos esclarecer o que é exatamente integração contínua (CI) e entrega contínua (CD). O objetivo da integração contínua é refinar a integração de código em uma tarefa de desenvolvimento simples, facilmente repetível, que servirá para reduzir os custos gerais de construção e revelar defeitos logo no início do ciclo. O sucesso na integração contínua dependerá de mudanças na cultura da equipe de desenvolvimento, de modo que haja incentivo para prontidão, compilações frequentes e iterativas e vontade de lidar com erros quando forem encontrados muito antes.

Uma forma de elucidar o funcionamento da integração contínua na prática é definir um pseudocódigo que descreva seu passo a passo do início até o fim do seu ciclo. Então, ao modificar o código-fonte dentro de um repositório, um script responsável pelas operações da CI executaria, as seguintes etapas:

1. Execute varreduras automáticas de qualidade de código e gere um relatório de como suas alterações mais recentes aderem a boas práticas de codificação.
2. Crie o código e execute todos os testes automatizados que você possa ter escrito para garantir que suas alterações não quebrem nenhuma funcionalidade.
3. Gere e publique um relatório de cobertura de teste para ter uma ideia de como seus testes automatizados são completos.
4. Executar a entrega do código (entrega contínua)

As etapas citadas acima resumem todas as responsabilidades da integração contínua. Percebemos que a integração contínua percorre por diferentes áreas do desenvolvimento de softwares. Utiliza-se de conceitos de qualidade código para que mantenha um padrão de codificação de acordo com as normas estabelecidas pelos órgãos competentes. Após a

avaliação da qualidade, existe a etapa de execução de testes automatizados. Aqui pode-se ter N tipos de testes (unitários, integração, segurança, integridade, etc), tudo que possa garantir ao desenvolvedor a segurança de que sua mudança não impactará negativamente na aplicação. Na entrega contínua, não há necessidade de uma pessoa decidir quando e o quê entrará em ambiente de produção, pois subentende-se de que após aprovação nos testes o código esteja pronto para ser incorporado ao código fonte. A última etapa em um sistema de CI é o CD. Esse implantará automaticamente todos os componentes e pacotes de compilação que forem bem-sucedidos na saída do pipeline de entrega contínua. Essas implantações automáticas podem ser configuradas para distribuir rapidamente componentes, recursos e correções para os clientes e também fornecer clareza sobre quais recursos estarão atualmente em produção.

2.3 Ambientes On-Cloud x Ambientes On-Premise

A computação em nuvem (ou cloud computing) tornou fácil o acesso a todos os aplicativos e softwares pela internet. Embora o termo “nuvem” possa parecer abstrato, os benefícios da cloud computing para os clientes são muito reais e tangíveis. O mundo está adotando a computação em nuvem e assim, espera-se inaugurar uma nova era de capacidade de resposta e eficiência na prestação de serviços de TI. Anteriormente, os softwares precisavam ser instalados fisicamente, o que não é mais necessariamente uma realidade. Com a evolução da tecnologia e a viabilidade de acessar aplicativos diretamente da internet, as empresas estão obtendo enormes benefícios da computação em nuvem.

Existem dois tipos possíveis de ambientes de infraestrutura que servem uma aplicação. São eles on-cloud e on-premise . On-premise, ou ambientes locais, refere-se a infraestruturas privadas mantidas pela própria empresa em suas instalações e que podem ser mantidas de modo independente. Já o on-cloud, ou ambientes na nuvem, refere-se a infraestruturas robustas que ficam instaladas em data centers (SOLUÇÕES, 2020). A principal diferença entre eles é que os recursos de ambientes na nuvem são disponibilizados de acordo com a necessidade do cliente, eles tem acesso aos dados pela internet na hora que quiser, em qualquer lugar do mundo onde estiverem e de forma segura. Além dessa, há outras diferenças entre os ambientes e por consequência, cada um tem suas vantagens e desvantagens. A seguir, isso será mostrado com mais detalhes em cada subárea da infraestrutura tanto na perspectiva de ambiente on-cloud quanto na de ambiente on-premise. (SOLUÇÕES, 2020) (CL9, 2020) (MANDIC, 2020)

2.3.1 Infraestrutura

- On-premise: A construção de uma infraestrutura on-premise necessita da compra de hardwares e softwares para o projeto, além disso, ter a sensibilidade do possível

crescimento potencial da demanda de clientes para o projeto, pois daí será necessário o provisionamento de mais recursos para a compra de uma quantidade a mais de hardware que supra este crescimento. É necessário o desembolso de valores elevados de ativos, que são essenciais para que seu projeto funcione adequadamente. Outro ponto é o ambiente onde esses equipamentos serão instalados, este local deve prover alguns requisitos para que seus equipamentos funcionem de acordo com as especificações. Necessitam de climatização, raques para a instalação dos equipamentos, sistemas de combate a incêndio, links para conectividade, energia elétrica, um ambiente seguro e que conte com medidas de segurança física e computacional.

- On-cloud: Os ambientes on-cloud disponibilizam uma estrutura física construída para atender todos os requisitos de segurança dos dados. Contam com segurança física com equipe armada, salas monitoradas, controle de acesso, salas cofre, câmeras, ambiente climatizado e sistemas de combate a incêndio moderno. Os Data Centers além de possuir a estrutura física mencionada conta com medidas de segurança física e lógica, redundância de conectividade, de energia, climatização, e etc. Ou seja, tudo o que seria necessário investir em equipamentos e sistemas para construir um ambiente on-premise, no ambiente on-cloud já está pronto e estruturado.

2.3.2 Operacional

Se faz necessário a implementação desse ambiente e também o seu gerenciamento para que ele possa funcionar da maneira desejada. O funcionamento dos softwares também deve entrar no planejamento de um projeto. Após montada a infraestrutura física, precisa-se das licenças dos softwares adquiridos (caso sejam pagos). Vale lembrar que em alguns casos é necessário a customização dos sistemas que serão usados conforme a necessidade.

- On-premise: Aqui o gerenciamento das aplicações usadas ou será feito pela própria empresa ou de forma terceirizada. Quando é feito de forma terceirizada todo o suporte, gerenciamento e monitoramento do ambiente é de responsabilidade da empresa contratada. Esta pode fazer ajustes, caso necessário, para que o projeto possa seguir em pleno funcionamento. Uma das grandes preocupações é a segurança tanto das aplicações quanto dos sistemas instalados. Como é de se esperar a segurança dos softwares adquiridos fica por conta da equipe terceirizada já contratada. Desse modo, faz-se necessário profissionais com diferentes qualificações para cada tipo de solução. Em grande parte dos casos, é necessária a contratação de uma equipe somente para gerenciamento e controle do ambiente, principalmente em áreas que podem ser críticas como a de segurança da aplicação.
- On-cloud: Com o ambiente na nuvem, são necessários pessoas para gerenciar e instalar recursos, tais como relatórios de tráfego, análises de melhoria, atualizações de sistemas

operacionais, ferramentas e etc. A diferença está na facilidade da contratação desse tipo de serviço. Grande parte das empresas que oferecem recursos na nuvem também possuem pacotes de serviços que englobam implantação, gerenciamento e suporte. Então, contratar profissionais da área que sejam qualificados para atender a demanda deixa de ser uma preocupação, afinal a decisão aqui seria apenas de encontrar uma empresa que ofereça os recursos necessários para sua demanda com um menor custo e uma maior eficiência.

2.3.3 Financeiro

Ao unir os recursos utilizados na estrutura e no operacional pode-se deduzir os custos financeiros de cada tipo de ambiente.

- On-premise: Hardwares, softwares e estruturas em geral são apenas o investimento inicial. Existe também um custo mensal de uma equipe focada na gestão do ambiente, seja ela terceirizada ou interna. Manutenções e atualizações frequentes serão aplicadas aos servidores, somado as horas de trabalho dos profissionais especializados e de consumo de energia elétrica. Aumentar recursos, atualizar licenças e profissionais especializados trazem um custo mensal tão alto quanto das estruturas em geral.
- On-cloud: Em analogia a realidade dos ambientes on-premise, na nuvem os investimentos iniciais são menores. Independente do pacote escolhido pela empresa, seja ele o mais simples ou o mais robusto, ambientes on-cloud dispõem da possibilidade de escalabilidade desses recursos conforme necessidade. Vale também, a disponibilidade do cliente, equipes multidisciplinares para a gestão, suporte e monitoramento de todo o ambiente. Em suma, o desperdício de recursos é contido e, se preciso de mais poder de processamento, a mudança não é onerosa ao usuário dos serviços on-cloud. Percebe-se que há redução no custo total do projeto, o investimento é somente no necessário com possibilidade de ampliação ou redução dos hardwares e softwares. A preocupação da empresa é voltada a funcionalidade de sua aplicação.

A seguir serão mostradas tabelas de custos de um ambiente on-premise para melhor ilustração e também para quantificar em valores monetários. Tais informações foram consultadas em um orçamento do site (SOLUçõES, 2020), empresa que atua hoje no mercado de Devops. Vale lembrar que tais recursos e valores são estimativas que se aproximam da realidade.

A primeira tabela refere-se ao custo dos hardwares de um sistema, ou seja, inclui toda estrutura física que faz uma aplicação funcionar, desde servidores, no-breaks e raques, até o firewall e equipamentos de segurança. A tabela possui descrição que nomeia o tipo de recurso, a quantidade necessária dele, o seu valor unitário e por fim o valor total. Na última linha está o somatório total de todos os componentes necessários.

HARDWARES			
Descrição	Quantidade	Valor Unitário	Valor Total
Servidores (hardware)	1	R\$ 25.000,00	R\$ 25.000,00
Infraestrutura de rede (cabos e conectores)	1	R\$ 3.000,00	R\$ 3.000,00
Switch 24 portas GigaBit Gerenciável	1	R\$ 2.000,00	R\$ 2.000,00
Rack (para colocar os Servidor (es))	1	R\$ 5.000,00	R\$ 5.000,00
No-break	1	R\$ 2.000,00	R\$ 2.000,00
Gerador (diesel ou gasolina)	1	R\$ 20.000,00	R\$ 20.000,00
Sistema de Climatização (Ar Condicionado)	1	R\$ 5.000,00	R\$ 5.000,00
Segurança Física - Equipamentos (trancas, sistemas de segurança)	1	R\$ 2.000,00	R\$ 2.000,00
Unidade de Backup (storage)	1	R\$ 5.000,00	R\$ 5.000,00
Sistema de Backup (Software específico para gestão do Backup)	1	R\$ 3.500,00	R\$ 3.500,00
Firewall (proteção lógica da rede)	1	R\$ 2.500,00	R\$ 2.500,00
Total Hardwares			R\$ 75.000,00

Figura 1: Tabela de custos de hadwares

A seguir, a tabela de softwares necessários estimados para o sistema e o seu custo total.

SOFTWARES			
Descrição	Quantidade	Valor unitário	Valor total
Licença Windows Server	1	R\$ 5.400,00	R\$ 5.400,00
Licença de Virtualização por processador	1	R\$ 3.000,00	R\$ 3.000,00
Serviços de Implementação do Ambiente	1	R\$ 3.000,00	R\$ 3.000,00
Total Hardwares			R\$ 11.400,00

Figura 2: Tabela de custos de softwares

Por fim a tabela de despesas mensais. Tais despesas são referentes a custo de aluguel, segurança pessoal, energia elétrica etc. A tabela possui a descrição da despesa, o seu valor mensal e seu valor num período de 36 meses que equivale aos próximos 3 anos. Ressalta-se que essa despesa acontece na mesma medida do tempo de vida do sistema.

DESPESAS MENSAIS		
Descrição	Valor Mensal	Valor em 36 meses
Estrutura Física (Predial)	R\$ 250,00	R\$ 9.000,00
Segurança Física - Pessoal	R\$ 300,00	R\$ 10.800,00
Link de Internet 4Mb Dedicado	R\$ 850,00	R\$ 30.600,00
Manutenção / Gerenciamento	R\$ 500,00	R\$ 18.000,00
Energia Elétrica	R\$ 200,00	R\$ 7.200,00
Custo de Depreciação	R\$ 120,00	R\$ 4.320,00
Custo Total Despesas - Serviços Mensais	R\$ 2.220,00	R\$ 79.920,00

Figura 3: Tabela de custos de despesas mensais

Em resumo, para uma hospedagem feita internamente, devemos considerar o valor pago em hardwares, softwares e as despesas. Em um período de 36 meses, teremos o seguinte custo total:

Resumo Investimento Hospedagem Interna		
Hardware + Software	R\$ 86.400,00	
Despesas	Valor mensal	Valor em 36 meses
	R\$ 2.220,00	R\$ 79.920,00
Custo total	R\$ 166.320,00	

Figura 4: Tabela de resumo de investimento em hospedagem interna

Agora iremos mostrar os custos no ambiente na nuvem para depois fazermos uma comparação no aspecto financeiro.

A primeira tabela descreve o recurso, a quantidade, o valor mensal e o valor em 36 meses (de maneira similar as tabelas anteriores). Uma observação aqui é que existe a atividade de implantação do ambiente e de migração, pois considera-se que o cliente irá mudar de um ambiente on-premise para o on-cloud.

AMBIENTE EM NUVEM (CLOUD)			
Descrição	Quantidade	Valor mensal	Valor em 36 meses
Cloud Privada	1	R\$ 2.354,00	R\$ 84.744,00
Firewall/VPN	1	R\$ 215,00	R\$ 7.740,00
Implantação do ambiente e migração	cobrado apenas uma vez	R\$ 2.800,00	
Custo Total - Serviços Mensais Nuvem		R\$ 2.569,00	R\$ 92.484,00

Figura 5: Tabela de custos de ambiente na nuvem

O resumo do investimento do sistemas em nuvem, considerando o custo de implantação, migração e o valor mensal pago ao serviço de ambiente na nuvem.

Resumo Investimento Sistemas em Nuvem		
Implantação do ambiente e migração	R\$ 2.800,00	
Ambiente nuvem	Valor mensal	Valor em 36 meses
	R\$ 2.569,00	R\$ 92.484,00
Custo total	R\$ 95.284,00	

Figura 6: Tabela de resumo de custo de todo ambiente na nuvem

Por fim, podemos comparar os valores. Percebe-se que o valor total para uma hospedagem utilizando uma estrutura interna é maior que quando utiliza-se a estrutura de hospedagem em nuvem. A diferença é alta e passa dos 170 por cento. Além disso, os problemas no gerenciamento de recursos físicos e de softwares podem acarretar na contratação de mais profissionais especializados nessa área o que pode gerar mais custos. Também é perceptível que o ambiente on-cloud simplifica os recursos necessários para implantação de um ambiente pois a empresa fornecedora de tal encapsula os recursos e os oferecem de maneira mais simples e resumida.

Investimento total em 36 meses	
Estrutura Interna	R\$ 166.320,00
Hospedagem em Nuvem	R\$ 95.284,00

Figura 7: Tabela de resumo de custo de todo ambiente na nuvem

3 Integração Contínua e Entrega Contínua na prática

Aqui iremos acompanhar como ocorreu a implementação dessa arquitetura. Será mostrado desde a criação da aplicação web até a automatização de sua publicação no ambiente da nuvem.

3.1 Docker

O docker é um software de alto desempenho que garante a facilidade para a criação de ambientes isolados e a rápida distribuição de programas para o usuário final. Essa ferramenta tem como objetivo implementar aplicações em um ambiente separado da máquina do desenvolvedor da máquina de produção. A esse ambiente separado damos o nome de contêiner (GOMES, 2018). Então, o desenvolvedor consegue empacotar seu software de maneira padronizada sem se preocupar se a máquina de produção terá os recursos necessários para executar a aplicação que foi desenvolvida por ele.

Para que os conceitos sobre o docker fiquem claro é válido o uso de uma analogia com um navio cargueiro feito por (DIEDRICH, 2017). A princípio o navio cargueiro foi feito apenas para o transporte de minério. Caso precisasse levar outro tipo de carga, como por exemplo a soja, teriam que ser feitas várias adaptações para que atendesse a esse tipo de demanda. Se imaginarmos que o navio cargueiro é o nosso host, cada vez que queremos utilizar um recurso diferente teríamos que atualizar a máquina do host com novos plugins, ferramentas, etc e checar possíveis incompatibilidades para o correto funcionamento da aplicação. Esse tipo de atualização (tanto no navio cargueiro quanto no host) era muito comum visto que a tecnologia sempre está evoluindo e mudando e as soluções que eram adotadas eram as que causavam o menor impacto no ambiente possível.

Os contêiners foram criados para solucionar esse problema. Vamos ver a imagem abaixo:



Figura 8: Navio cargueiro

Esse é um navio cargueiro transportando contêineres de qualquer tipo de produto e não mais só de minério. Os contêineres apesar de um simples conceito, permitiu tal façanha para esse navio. Então, da mesma forma que o navio consegue hospedar vários contêineres de diferentes produtos, os contêineres do docker permite o encapsulamento de uma aplicação e que ela execute em qualquer lugar. Não é necessário se preocupar se a máquina destino terá os requisitos para sua aplicação executar pois tudo o que ela precisa estará dentro do contêiner já construído pelo desenvolvedor. Agora a única forma de quebrar o ambiente de produção é errando na construção dos contêiner. Os engenheiros de softwares passaram a se preocupar somente em construir um host que tenha alguma tecnologia capaz de executar contêineres docker.

3.1.1 Diferenças entre Máquina Virtual e contêiner

É comum depois de entender o conceito do docker confundir e/ou comparar com as máquinas virtuais. Se verificarmos apenas a funcionalidade, ambas as soluções são parecidas. As grandes diferenças aparecem no emprego da tecnologia utilizada em cada uma. Enquanto na virtualização tradicional o ambiente emulado contempla memória, processamento, disco, rede e ainda todo um sistema operacional instalado, nos contêineres isso é bem diferente, não existe toda essa sobrecarga de recursos desnecessários, pois o isolamento e controle de recursos é realizado por chamadas de sistemas diretamente e não por uma camada de virtualização. A seguir, uma imagem bastante conhecida, mas que ajuda no entendimento das diferenças de ambas tecnologias:

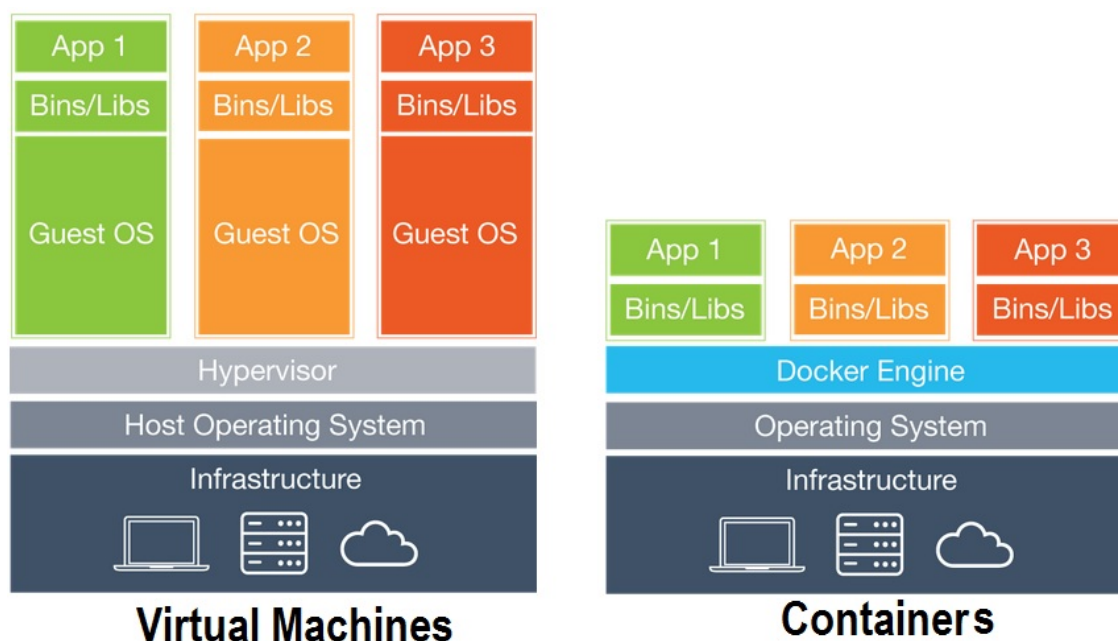


Figura 9: Diferenças entre Máquina virtual e contêiner

Na imagem percebe-se que nas máquinas virtuais existem as camadas de Guest OS e Hypervisor para que a aplicação consiga acessar o recurso do sistema. O Guest OS é justamente o sistema operacional da máquina virtual, ou seja, a parte que sobrecarrega as máquinas virtuais. Ao contrário dos contêineres que utilizam apenas a interface Docker Engine para se comunicar com o sistema operacional do host. O hipervisor é o responsável por executar máquinas virtuais, ele faz isso isolando o sistema operacional do próprio hipervisor e os recursos das máquinas virtuais e permite a criação e o gerenciamento dessas (HAT, 2020).

3.1.2 Instalação do Docker

A instalação do Docker é bem simples. De maneira geral os passos são apenas executar um instalador que pode ser baixado no site da Docker Hub. O sistema operacional tem que permitir a utilização dos recursos de virtualização (em alguns casos isso deverá ser habilitado manualmente). Há também uma observação para usuários do windows, as únicas versões que permitem a utilização do Docker desktop completo são a pro, enterprise e education. Para o Windows na versão home a Docker recomenda a utilização de uma versão diferente que será abordada passo a passo aqui.

Após o download do instalador no site da docker hub da versão para Windows Home, é necessário instalar com a opção WSL 2 Windows Features habilitada. A opção WSL 2, ou melhor, Windows Subsystem for Linux, permite que o sistema operacional windows execute sem precisar de emulação contêineres linux no sistema. Será necessário atualizar o kernel do windows para completar a instalação. Anteriormente isso não era possível e usuários

do Windows Home deveriam utilizar uma versão minimizada do Docker chamada de DockerToolBox que caiu em desuso após essa significativa adição na arquitetura Windows. Além disso o WSL 2 prover melhorias no sistema de compartilhamento de arquivos, tempo de boot e permite acesso a mais recursos do Docker desktop (DOCKER, 2020a).

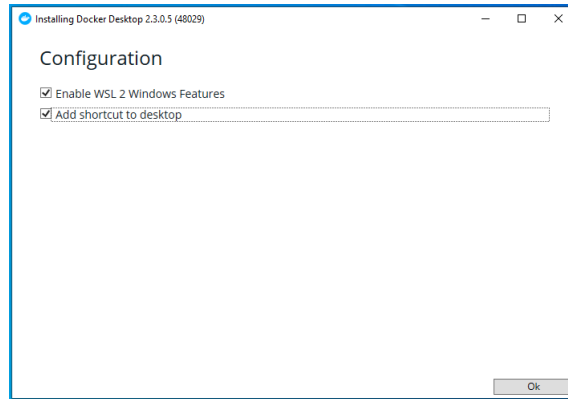


Figura 10: Instalação Docker Desktop para Windows Home

Após o procedimento concluir o Docker Desktop está pronto. Uma tela de boas vindas seguida de um tutorial para criação de um primeiro contêiner já configurado salvo no GitHub são mostrados na primeira vez que o Docker é executado na máquina.

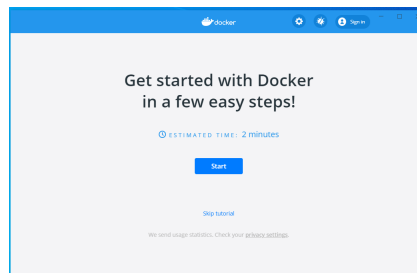


Figura 11: Instalação Docker Desktop para Windows Home Boas vindas

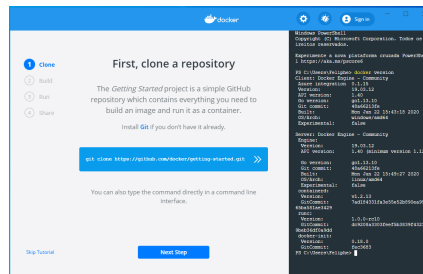


Figura 12: Instalação Docker Desktop para Windows Home tutorial

3.1.3 Contêiner Docker

Como já foi explicado, o Docker consegue criar contêineres de aplicações de acordo com as diretivas pré-programadas. Nessa seção trataremos de como isso é feito e como

criar tais regras para que o Docker entenda o que é para ser feito dentro do contêiner.

O artefato mais importante nesse processo de criação chama-se Dockerfile. O Dockerfile é um documento de texto que contém todos os comandos que o usuário pode executar para montar uma imagem. Usando o *docker build*, os usuários podem criar uma build automatizada que executa várias instruções de linha de comando em sucessão (DOCKER, 2020b). A imagem Docker (Docker image) funciona como um template para o contêiner. É a partir da imagem, criada de acordo com as instruções no Dockerfile, que o contêiner será criado.

Nesse trabalho usaremos um banco de dados MySQL, então a princípio, será criado um contêiner da imagem MySQL para que o processo de criação seja explanado do início ao fim. Os passos para criação do Dockerfile serão:

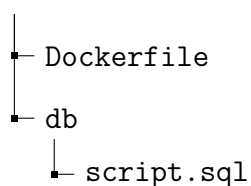
- Criar arquivo de texto sem extensão nomeado de Dockerfile.
- Definir uma imagem oficial como base para ser modificada.
- Definir as regras de criação para a imagem.
- Executar o contêiner a partir da imagem criada

Criado o arquivo de texto sem extensão e nomeado de Dockerfile as instruções que estão dentro dele serão as seguintes:

```
FROM mysql:latest
COPY ./db/ /docker-entrypoint-initdb.d/
```

O comando *FROM mysql:latest* define a imagem base para ser modificada. Essa imagem, é buscada pelo nome no repositório oficial de imagens da docker (Dockerhub). Quando essa linha é executada pelo Docker, temos a certeza de que nossa imagem terá todas os recursos que são necessários para rodar uma base de dados do tipo MySQL dentro do contêiner. Vale lembrar que uma vez baixada para a máquina do usuário, ela fica disponível para futuras criações de imagem o que otimiza tempo.

As regras de criação para nossa imagem é bem simples. Antes, devemos definir a estrutura do nosso diretório em que terá o Dockerfile e os scripts para dar início ao banco de dados. Será a seguinte:

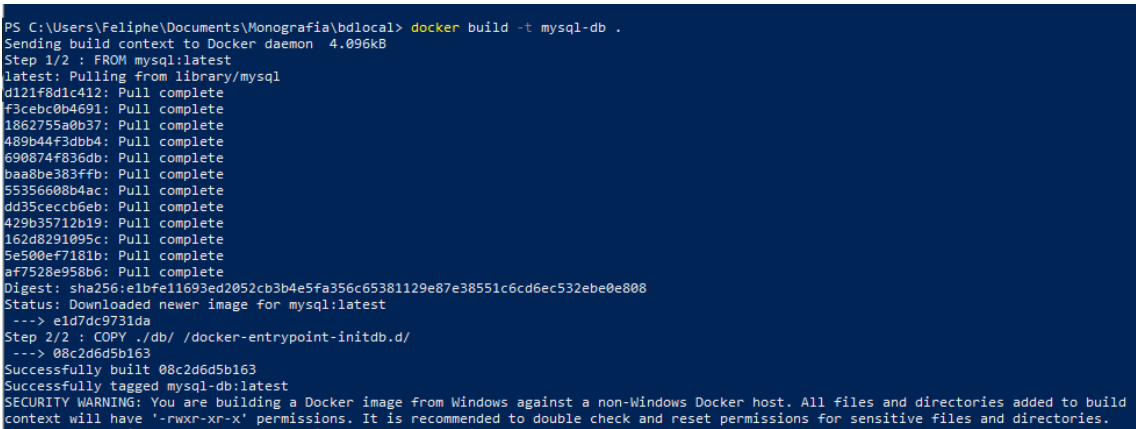


O comando `COPY ./db/ /docker-entrypoint-initdb.d/` apenas move os arquivos da pasta `db` (máquina local do usuário) para a pasta `docker-entrypoint-initdb.d/`. (imagem criada). Isso é feito pois o MySQL enxerga essa pasta logo na criação do banco e executa os arquivos com extensão `sql` que estão lá dentro. Assim podemos iniciar a base de dados já com a estrutura necessária para o funcionamento da aplicação que será usada como objeto de estudo nesse trabalho.

Agora com o Dockerfile criado e com a estrutura de pastas com o script de banco inicial, executaremos o primeiro comando `docker` para criar e imagem. O comando para fazer isso é o seguinte:

```
docker build -t mysql-db .
```

Aqui é criada uma imagem `docker` tendo como referência o Dockerfile, por isso é necessário estar na mesma pasta em que o Dockerfile estar. O valor `mysql-db` é o nome da imagem. O caractere de ponto (`.`) significa que que o Dockerfile está na pasta em que o comando em questão está sendo executado, caso contrário pode-se colocar o caminho completo ali. A seguinte imagem mostra o comando sendo executado:



```
PS C:\Users\Felipe\Documents\Monografia\bdlocal> docker build -t mysql-db .
Sending build context to Docker daemon 4.096kB
Step 1/2 : FROM mysql:latest
latest: Pulling from library/mysql
d121f8d1c412: Pull complete
f3cebc0b4691: Pull complete
1862755a0b37: Pull complete
489b44f3dbb4: Pull complete
690874f836db: Pull complete
baa8be383ffb: Pull complete
55356608b4ac: Pull complete
dd35ceccb6eb: Pull complete
429b35712b19: Pull complete
162d8291095c: Pull complete
5e500ef7181b: Pull complete
af7528e958b6: Pull complete
Digest: sha256:e1bfe11693ed2052cb3b4e5fa356c65381129e87e38551c6cd6ec532ebe0e808
Status: Downloaded newer image for mysql:latest
--> e1d7dc9731da
Step 2/2 : COPY ./db/ /docker-entrypoint-initdb.d/
--> 08c2d6d5b163
Successfully built 08c2d6d5b163
Successfully tagged mysql-db:latest
SECURITY WARNING: You are building a Docker image from Windows against a non-Windows Docker host. All files and directories added to build context will have '-rwxr-xr-x' permissions. It is recommended to double check and reset permissions for sensitive files and directories.
```

Figura 13: Comando `docker build` sendo executado

Após criada a imagem, é necessário executá-la. A imagem em execução leva o nome de `contêiner`. Para executar uma imagem é necessário o comando `docker run`, ele é responsável pela criação do `contêiner`. Também é necessário passar alguns parâmetros para esse comando para personalizar o `contêiner` de acordo com a necessidade da imagem criada. O comando a ser executado completo é o seguinte:

```
docker run -d -p 3306:3306 --name login -e MYSQL_ROOT_PASSWORD=admin -e
MYSQL_DATABASE=login -e MYSQL_USER=admin -e MYSQL_PASSWORD=admin
```

mysql-db

O parâmetro `-p` diz respeito a porta em que a máquina local vai responder a porta virtualizada pelo docker, que no caso, é a mesma para ambas por padrão de uso do banco de dados MySQL. O parâmetro `-e` refere-se a alteração de variáveis de ambiente do contêiner, então aqui é possível mudar os valores delas de acordo com a necessidade. No caso mudamos a senha do usuário root, o nome do banco de dados e definimos o usuário *admin* com a senha também *admin*. O parâmetro `MYSQL_DATABASE` é o nome do banco que será criado inicialmente. O valor `mysql-db` é a imagem a ser usada para a criação do contêiner com os parâmetros passados e `-name` é o nome do contêiner. A imagem a seguir mostra o comando executado:

```
PS C:\Users\Felipe\Documents\Monografia\bdlocal> docker run -p 3306:3306 --name container-db -e MYSQL_ROOT_PASSWORD=admin -e MYSQL_DATABASE=admin -e MYSQL_USER=admin -e MYSQL_PASSWORD=admin mysql-db
1553966b68105152f754c6cd0b65122b41df56c5d10c426c316aa44e93dfe316d
PS C:\Users\Felipe\Documents\Monografia\bdlocal> docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                    NAMES
1553966b6810   mysql-db      "docker-entrypoint.s..." 13 seconds ago Up 12 seconds  0.0.0.0:3306->3306/tcp, 33060/tcp  container-db
```

Figura 14: Comando `docker run` sendo executado

Após o comando `docker run` foi executado o comando `docker ps`. Esse último lista todos os contêineres em execução naquele momento e retorna informações úteis do contêiner como o *contêiner ID* então confirma-se que o contêiner foi criado com sucesso.

Depois do contêiner está em execução, podemos acessá-lo e executar programas dentro dele, como se fosse uma máquina virtual. O comando seguinte é capaz de executar o bash do contêiner:

```
docker exec -it 1553966b6810 bash
```

O valor `182211a2a023` é o identificador do contêiner que pode ser recuperado através do comando `docker ps -a`. O valor `bash` é a aplicação que será usada dentro do contêiner e espera-se que o usuário informe que recurso será utilizado e com quais permissões. Como é um contêiner de um imagem MySQL, a entrada será a seguinte:

```
mysql -uroot -p
```

Então é esperado o password de root pré-configurado na criação do contêiner. A partir daqui já é possível utilizar comandos sql e acessar os objetos do banco, tudo pelo prompt de comando. Segue uma imagem com todos comandos explicados acima em execução na prática:

```
PS C:\Users\Felipe\Documents\Monografia\bdlocal> docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                    NAMES
1553966b6810  mysql-db      "docker-entrypoint.s..." 6 hours ago   Up 6 hours   0.0.0.0:3306->3306/tcp, 33060/tcp  container-db
PS C:\Users\Felipe\Documents\Monografia\bdlocal> docker exec -it 1553966b6810 bash
root@1553966b6810:/# mysql -uroot -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 12
Server version: 8.0.21 MySQL Community Server - GPL

Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> select * from login.users;
Empty set (0.00 sec)

mysql>
```

Figura 15: Executando o *bash* do contêiner MySQL

Nós também podemos criar um contêiner MySQL utilizando a imagem oficial disponibilizada no docker hub. Ela exige menos passos para ser feita e dispensa a criação de um Dockerfile. Isso acontece pois o nome do contêiner é *mysql*, assim, o Docker, após verificar que não existe uma imagem local com esse nome, usa a imagem oficial do *mysql* disponível no DockerHub. Nós definimos isso dentro do Dockerfile a nível informativo, mas na verdade não precisaria. O comando completo segue a seguir:

```
docker run -d -p 3306:3306 --name my-mysql -v
  /c/projeto_mono/mysql/db/:/docker-entrypoint-initdb.d/ -e
  MYSQL_ROOT_PASSWORD=supersecret -e MYSQL_DATABASE=company mysql
```

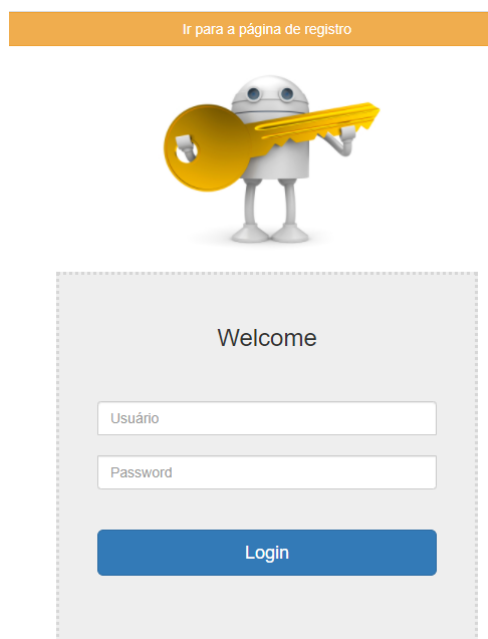
3.2 A aplicação web

O intuito desse projeto é mostrar os conceitos de DevOps aplicados na prática e para aplicarmos tais conceitos precisamos de uma aplicação já desenvolvida. Nesse momento, trataremos essa aplicação que terá as mudanças necessárias para utilizar dessas tecnologias. Vale lembrar que não é objeto desse estudo analisar detalhadamente a linguagem, plugins e/ou frameworks utilizados por ela.


A aplicação original está disponibilizada no artigo (PONCE, 2016). É uma aplicação simples que gerencia um controle de acesso. Existe um cadastro de novos usuários e, após cadastrados, os usuário terão acesso a uma página exclusiva para usuários cadastrados, ou seja, é uma demonstração de implementação possível de um controle de acesso. A seguir serão apresentadas algumas imagens do sistema sendo executado localmente para ilustrar melhor seu funcionamento.

A primeira tela corresponde a tela de login, é a tela inicial da aplicação. Nela possui os campos para usuário inserir o login e a senha. Caso o usuário já possua o registro basta inserir as informações e clicar no botão de login. Caso contrário existe um botão na parte

superior para o usuário sem cadastro seguir para fazer o registro.



Ir para a página de registro



Welcome

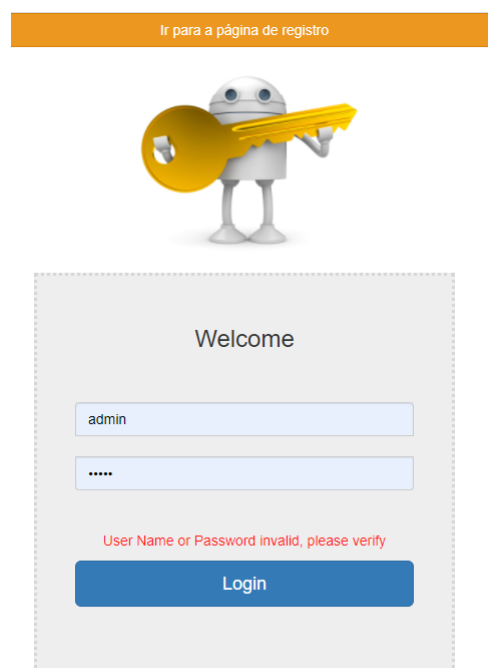
Usuário

Password


Login

Figura 16: Tela inicial de login da aplicação

Caso o usuário entre com um login e senha inválido o sistema irá fazer a verificação e retornar uma mensagem de erro, como mostrado a seguir:



Ir para a página de registro



Welcome

admin

.....

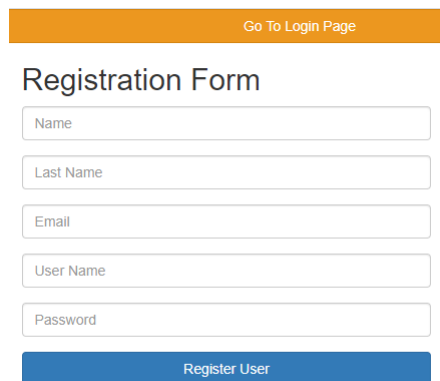
User Name or Password invalid, please verify

Login

Figura 17: Tela inicial: login inválido

Para o usuário se cadastrar no sistema o mesmo deve seguir para a página de

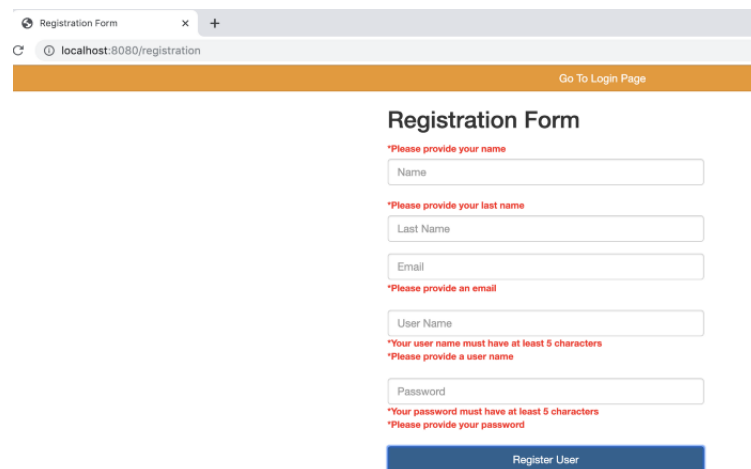
registro, para isso basta clicar no botão da tela inicial nomeado de *Ir para página de registro*. A seguir, a tela de registro de usuário:



The image shows a registration form with a blue header bar containing the text "Go To Login Page". Below the header, the title "Registration Form" is displayed. The form consists of five input fields: "Name", "Last Name", "Email", "User Name", and "Password". At the bottom of the form is a blue button labeled "Register User".

Figura 18: Tela de registro da aplicação

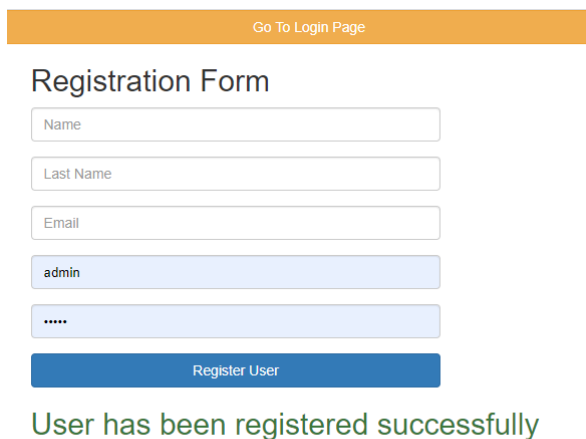
Os campos obrigatório do registro são validados e caso o usuário não informe os dados, as mensagens de validação irão aparecer como mostra a imagem:



The image shows the registration form with validation messages. The browser address bar shows "localhost:8080/registration". The form has the same layout as Figure 18, but with red error messages above each input field: "Name" has "*Please provide your name"; "Last Name" has "*Please provide your last name"; "Email" has "*Please provide an email"; "User Name" has "*Your user name must have at least 5 characters" and "*Please provide a user name"; "Password" has "*Your password must have at least 5 characters" and "*Please provide your password". The "Register User" button is still visible at the bottom.

Figura 19: Tela de registro da aplicação com as validações de campos obrigatório

Depois de preenchido todos os dados corretamente, o usuário clica em *Register User*, assim o cadastro do usuário estará completo e disponível para logar. Uma mensagem de sucesso será mostrada conforme a seguir:



The image shows a web registration form. At the top, there is an orange button labeled "Go To Login Page". Below it, the title "Registration Form" is displayed. The form consists of several input fields: "Name", "Last Name", and "Email", each with a light blue border. Below these are two password fields; the first contains the text "admin" and the second contains "....". At the bottom of the form is a dark blue button labeled "Register User". Below the form, a green message states "User has been registered successfully".

Figura 20: Tela de registro da aplicação após registrado um usuário com sucesso

Para logar na aplicação com o usuário cadastrado e acessar a área restrita basta retornar a página de login e inserir login e senha cadastrados. A tela a ser carregada agora é uma página que só pode ser acessada por usuários cadastrados. Segue a tela restrita para usuários:

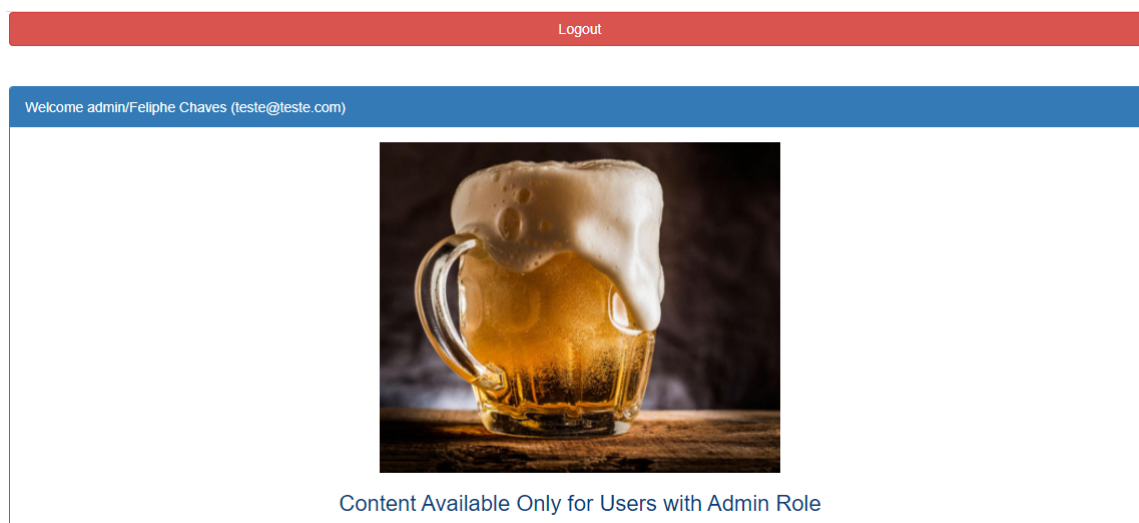


Figura 21: Tela de acesso restrito a usuários cadastrados

A tela tem uma imagem e uma mensagem que explica que esse conteúdo somente é acessado para usuário com o papel de administrador que nesse caso, todos os usuários cadastrados serão atribuídos a esse papel pois é o único existente nessa aplicação. Além de mostrar o login, nome e email do usuário logado.

O modelo de dados utilizado nessa aplicação é também bem simplista. No banco de dados nomeado de *Login*, teremos apenas as seguintes tabelas: *hibernate_sequence*, *roles*, *user_role* e *users*.

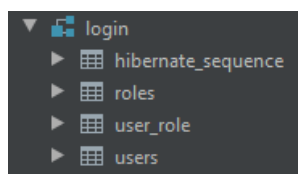
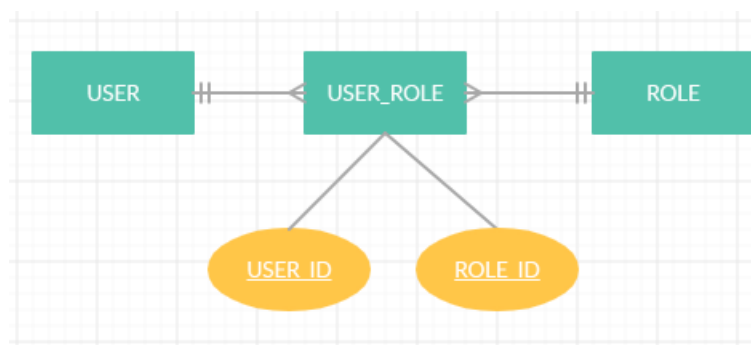


Figura 22: Tabelas criadas no banco de dados

A tabela `hibernate_sequence` é requerida pelo framework do `springboot` utilizado no desenvolvimento dessa aplicação. Ela é responsável por atribuir identificadores únicos para os dados inseridos nas tabelas mapeadas como classes modelo.

A tabela `role` guarda os possíveis papéis que os usuários poderão ter, nesse caso, essa tabela a princípio possui apenas um dado referente ao papel de administrador. Assim como a tabela `role`, a tabela `users` guarda os usuários cadastrados na aplicação, dados como nome, email e senha. E por fim a tabela `user_role` é a tabela que faz o relacionamento de usuário com o papel que ele pode ter. Esse é um relacionamento de 1 usuário para N papéis. A imagem a seguir, mostra no modelo entidade relacionamento a relação entre eles:

Figura 23: Modelo ER das tabelas `user`, `role` e `user_role`

A aplicação em questão, até o momento, utiliza um banco de dados `MySQL` e esse está num contêiner `docker` que executa na máquina local. A utilização de serviços da nuvem ajuda a manter a segurança e integridade de informação, principalmente quando se trata de uma base de dados. A `Google Cloud` oferece, dentro do ambiente de serviços da nuvem dela, também um serviço de base de dados. Visto que isso pode otimizar as aplicações que utilizam de base de dados, iremos abordar esse método de usar o banco de dados como serviço na nuvem também. Ou seja, ao invés de criar um contêiner `docker` de um banco de dados `MySQL`, iremos utilizar um serviço de banco de dados `MySQL` fornecido pelo `Google Cloud`.

3.2.1 Configurando o banco de dados na `Google Cloud`

Para a aplicação, a mudança do local do banco de dados não trará grandes mudança. Na verdade, a única informação que precisa ser alterada, é a URL de conexão com o banco,

pois o endereço de ip mudará e o mesmo faz parte da composição da URL de conexão. Essa geralmente é formada pelo: drive de conexão (muda de acordo com o Sistema Gerenciador de Banco de Dados utilizado), endereço ip, porta de reposta do banco de dados e o nome do banco de dados. Atualmente a URL está assim:

```
jdbc:mysql://localhost:3306/login
```

Depois de mudar o banco de dados, o valor *localhost* mudará para o novo endereço ip fornecido pelo serviço de MySQL do Google Cloud. Então, agora irar ser tratado a criação e configuração desse banco de dados.

O Google Cloud é uma plataforma da Google que oferece diversas soluções para diversos problemas. O que vamos utilizar por hora é o MySQL. O requisito para acessar os recursos do Google Cloud é ter uma conta Google. Após logar na plataforma e criar um projeto novo, basta acessar ao SQL no menu lateral e criar uma nova instância de banco conforme as imagens abaixo:

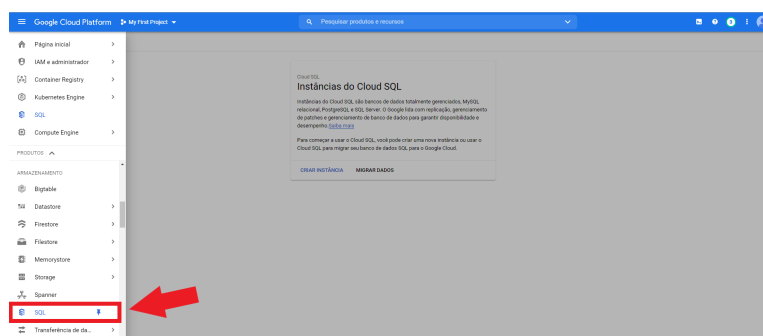


Figura 24: Opção SQL no menu lateral para iniciar a configuração



Figura 25: Opção nova instância para começar a criação da nova instância do banco de dados

Na criação da instância, o serviço disponibilizará os tipos de bancos que são trabalhados na plataforma. Atualmente serão três SGBDs diferentes MySQL, PostgreSQL e SQL Server. Afim de manter a mesma versão do banco trabalhado localmente, iremos utilizar o MySQL na versão 8 (mais atual).

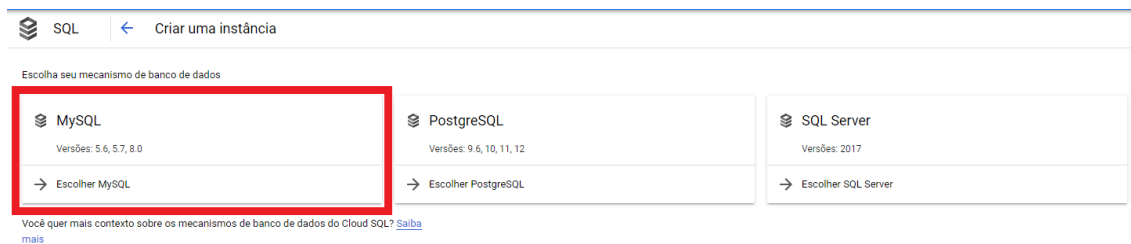


Figura 26: SGBDs disponíveis para utilização do serviço SQL da Google Cloud

A última parte da criação do banco é definir algumas informações sobre a instância. Na próxima tela a ser exibida pelo assistente de criação é um formulário que o usuário precisará informar o nome da instância, a senha do usuário root, definir em que região será alocado o serviço do banco de dados dentro dos servidores da google e também definir uma zona. Aqui, a recomendação é que se mantenha os dados próximos dos serviços que utilizam ele, ou seja, escolher a mesma região em que a aplicação ficará hospedada para diminuir a distância física e por consequência evitar algum problema possível de tempo de resposta.

Figura 27: Definição de nome da instância, senha do usuário root, região e zona

Por padrão de segurança, a instância de banco de dados que será criada inicialmente não aceita conexões de qualquer IP pois seria uma falha de segurança. Então ainda nessa tela existem várias opções de permitir a conexão no banco de dados: utilizando um proxy do Cloud SQL, configurando um IP privado (que depois de ativada não pode ser revertida) e por último a de IP público onde o usuário pode adicionar uma lista de IPs que são

permitido acesso ao banco. O recomendado pela própria Google Cloud é a utilização do proxy do Cloud SQL, assim dispensando a configuração da autorização de IPs específicos para acessá-lo. Porém, utilizaremos a configuração de IP Público e assim adicionaremos o IP da máquina local que roda a aplicação, para que quando for iniciada, ela tenha acesso ao banco de dados na nuvem.

Por fim, basta clicar no botão de criar para que o Google Cloud inicie o processo de criação da instância. Após concluído o procedimento, o usuário é redirecionado a um painel de controle da instância onde várias informações úteis serão mostradas como um gráfico de utilização da CPU, o nome da conexão, a atual configuração da instância, um histórico de alterações na mesma e etc.

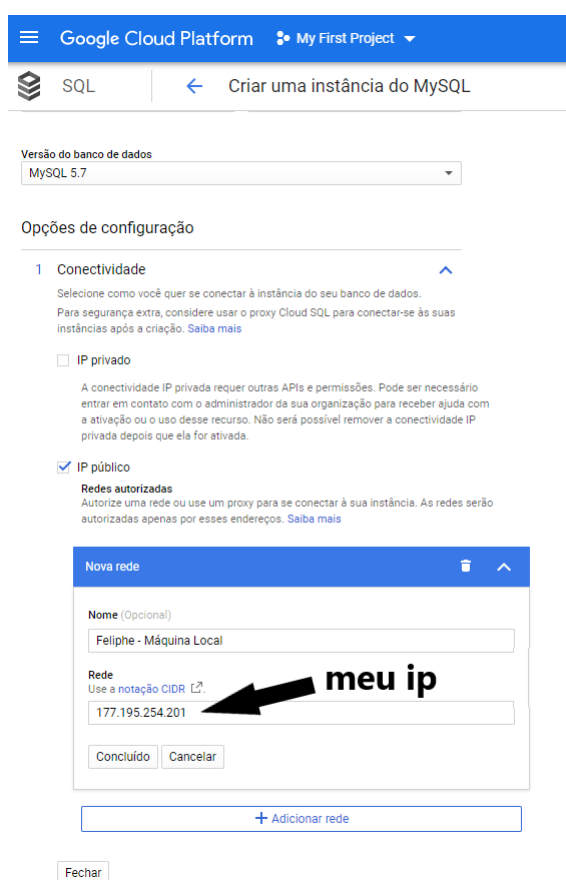


Figura 28: Adicionado IP para autorizar acesso a instância do banco de dados

Com a instância do banco criada, agora basta criar um banco de dados nessa instância para finalizar a configuração nessa parte. A criação do banco também é muito simples, basta selecionar a opção *Criar novo banco de dados* e definir um nome a ele.

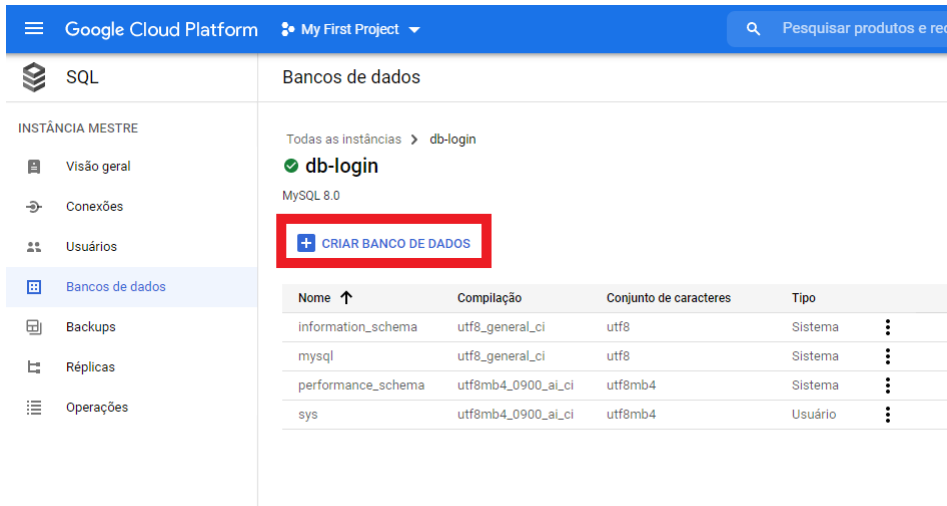


Figura 29: Criar novo banco de dados

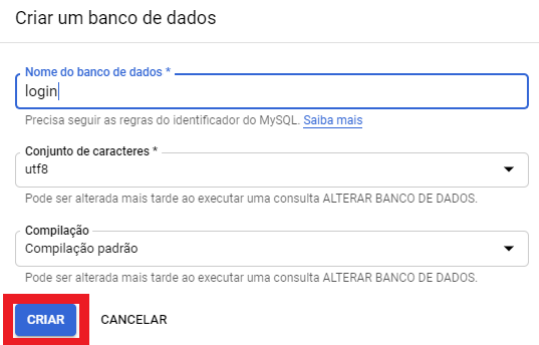


Figura 30: Definido nome para o banco de dados

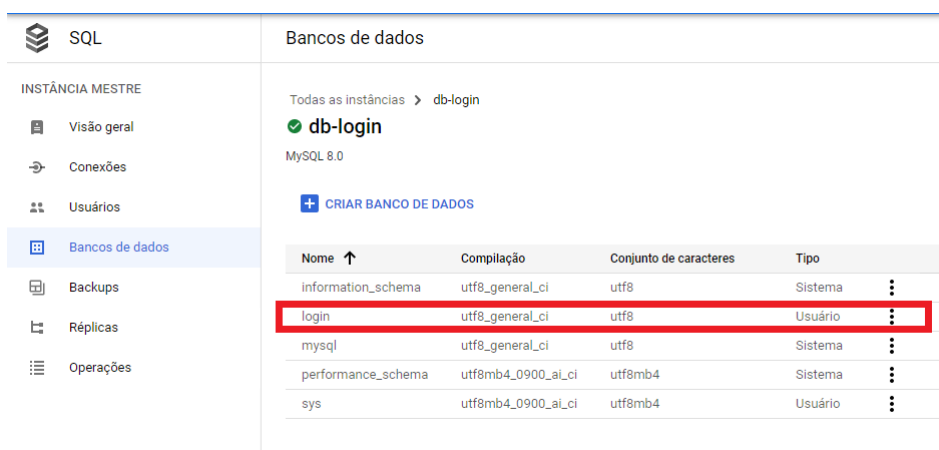


Figura 31: Banco de dados criado

Na *Visão geral* consegue-se saber o IP para acesso ao banco e assim configurar e testar a conexão com o novo banco de dados. Faremos a configuração e conexão do banco utilizando a ferramenta DataGrip.



Figura 32: Visão Geral - IP da instância do banco de dados

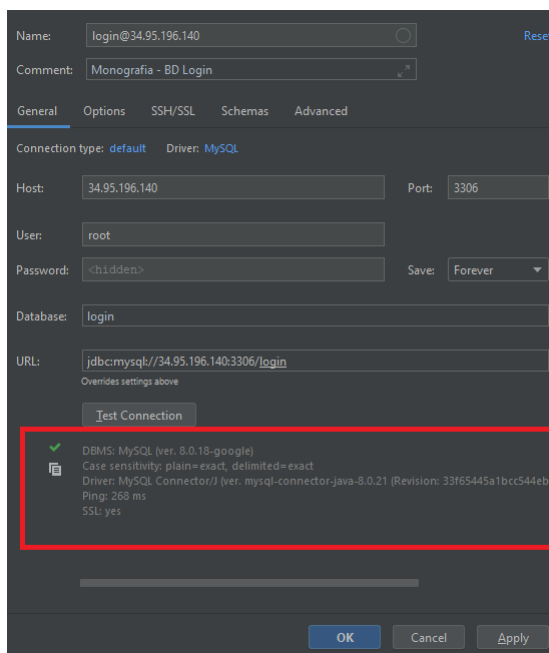


Figura 33: Teste de conexão com o banco criado pela DataGrip

Testada a conexão, basta mudar a URL de conexão da aplicação e já conectará no novo banco de dados. A nova URL ficará assim:

```
jdbc:mysql://34.95.196.140:3306/login
```

3.2.2 Contêinerização da aplicação web

Detalhada a aplicação que servirá como objeto a ser publicado automaticamente em nosso ambiente e descrito como foi criado o configurado o banco de dados na nuvem, agora entraremos num processo de *contêinerização* da aplicação, ou seja, iremos agora criar um Dockerfile para a aplicação definindo tudo o que ela irá precisar pra funcionar dentro de um contêiner Docker. Esse processo será de maneira similar ao contêiner MySQL já criado nesse trabalho, portanto, devemos definir as diretivas de criação da imagem com todos os recursos necessitados pela aplicação em questão e coloca-los dentro de um arquivo Dockerfile, então o Docker, para criar a imagem, irá seguir as regras criadas nesse arquivo.

Para iniciar a criação do dockerfile precisamos identificar quais são os recursos necessários para a aplicação executar, ou seja, qual a linguagem e quais frameworks foram utilizados em seu desenvolvimento. A aplicação de login, já detalhada seu funcionamento anteriormente, utilizou da linguagem Java em sua versão 8 e também utiliza o Maven - framework que facilita o processo de download de bibliotecas utilizadas pela aplicação durante o desenvolvimento da aplicação pelo programador. Tendo como base essas informações iniciaremos o dockerfile com o seguinte comando:

```
FROM maven:3.5.2-jdk-8-alpine AS MAVEN
```

Ou seja, aqui o docker iniciará a base da imagem a ser criada com uma versão do maven, possibilitando a aplicação utilizar tal recurso. Devemos definir um diretório de trabalho, ele nos auxiliará quando precisarmos copiar ou fazer download de arquivos durante a execução do script do dockerfile. Para definir um diretório dessa forma o código é o seguinte:

```
WORKDIR /tmp/
```

Feito isso, devemos fazer o download das bibliotecas que foram usadas no desenvolvimento da aplicação e colocá-las em cache. A definição dessas bibliotecas ficam dentro do arquivo pom.xml de acordo com o padrão de utilização do framework Maven. Para que o docker faça o download de todas as dependências, copiaremos o pom.xml do projeto para a pasta tmp (diretório de trabalho definido) e executaremos um maven build para que as dependências descritas no arquivo xml sejam baixadas:

```
COPY pom.xml /tmp/  
RUN mvn -B -f ./pom.xml dependency:go-offline
```

Agora usaremos o diretório tmp para copiar o código fonte da aplicação e passar para a imagem a ser criada. Um simples copy será executado para isso.

```
COPY src /tmp/src/
```

Para finalizar a parte do Maven, executar o comando o clean package, para que o código fonte seja compilado juntamente com as dependências.

```
RUN mvn -Dmaven.test.skip=true -B -s clean package
```

Com a parte do Maven configurada, devemos agora configurar o Java dentro do dockerfile também. Da mesma forma devemos partir de uma imagem de base do Java e definir um diretório de trabalho.

```
FROM openjdk:8-jdk-alpine  
WORKDIR /root
```

Uma outra necessidade da aplicação, é se comunicar com a porta do host onde será hospedada. No caso dos contêiners, devemos informar qual porta ficará exposta para receber requisições do host, dessa maneira, como padrão das aplicações web, usaremos a porta 8080:

```
EXPOSE 8080
```

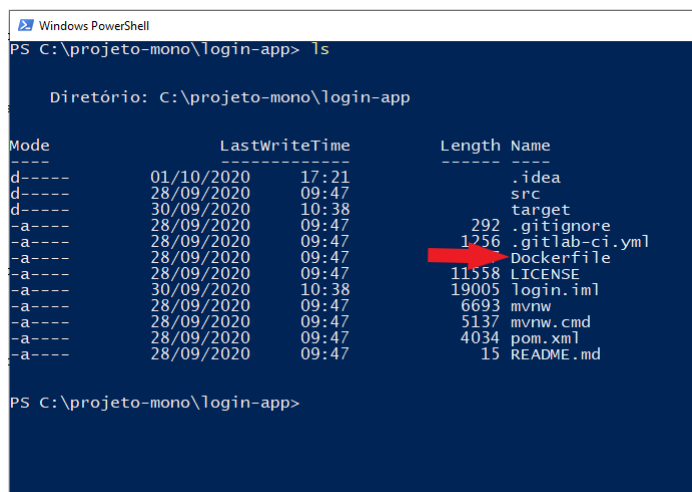
Para finalizar o dockerfile, pegaremos o objeto java já compilado (arquivo jar), e o colocaremos na pasta de trabalho e assim iniciar a aplicação com o comando java (comando ENTRYPOINT). O fim da instrução ficará assim:

```
COPY --from=MAVEN /tmp/target/login*.jar app.jar
ENTRYPOINT
    ["java", "-Djava.security.egd=file:/dev/./urandom", "-jar", "app.jar"]
```

Então, o arquivo completo ficará dessa forma:

```
FROM maven:3.5.2-jdk-8-alpine AS MAVEN
WORKDIR /tmp/
COPY pom.xml /tmp/
RUN mvn -B -f ./pom.xml dependency:go-offline
COPY src /tmp/src/
RUN mvn -Dmaven.test.skip=true -B -s clean package
FROM openjdk:8-jdk-alpine
WORKDIR /root
EXPOSE 8080
COPY --from=MAVEN /tmp/target/login*.jar app.jar
ENTRYPOINT
    ["java", "-Djava.security.egd=file:/dev/./urandom", "-jar", "app.jar"]
```

Com o arquivo Dockerfile configurado, devemos agora seguir as etapas de criação de imagem e do contêiner. Com isso, conseguiremos acessar uma aplicação que está funcionando dentro de um contêiner docker. Então, para criar a imagem devemos acessar o console e começar os comandos de criação de imagem (docker build) e executar a imagem transformando a imagem num contêiner (docker run). Para executar os comandos de criação do contêiner, é necessário que o dockerfile criado esteja dentro do diretório do projeto.



```
Windows PowerShell
PS C:\projeto-mono\login-app> ls

Diretório: C:\projeto-mono\login-app

Mode                LastWriteTime         Length Name
----                -
d-----            01/10/2020    17:21      .idea
d-----            28/09/2020    09:47      src
d-----            30/09/2020    10:38      target
-a-----            28/09/2020    09:47          292      .gitignore
-a-----            28/09/2020    09:47          1256     .gitlab-ci.yml
-a-----            28/09/2020    09:47          1256     Dockerfile
-a-----            28/09/2020    09:47          11558    LICENSE
-a-----            30/09/2020    10:38          19005    login.iml
-a-----            28/09/2020    09:47          6693    mvnw
-a-----            28/09/2020    09:47          5137    mvnw.cmd
-a-----            28/09/2020    09:47          4034    pom.xml
-a-----            28/09/2020    09:47           15    README.md

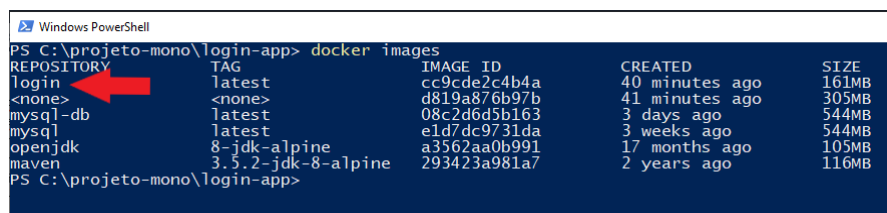
PS C:\projeto-mono\login-app>
```

Figura 34: Lista de arquivos do projeto java

Então, o próximo passo é gerar a imagem docker da aplicação de login. A imagem a ser gerada terá o nome *login*:

```
docker build -t login .
```

Nesse momento, o docker irá pegar as informações do dockerfile e executar. Uma série de downloads irão acontecer, tanto das dependências quanto das imagens usadas como base para criação do contêiner. Tudo isso para preparar o contêiner para a aplicação que rodará dentro dele. Para ver as imagens que foram criadas pode ser usado o comando *docker images* que lista as imagens disponíveis na máquina.



```
Windows PowerShell
PS C:\projeto-mono\login-app> docker images

REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
login         latest   cc9cde2c4b4a   40 minutes ago 161MB
<none>        <none>   d819a876b97b   41 minutes ago 305MB
mysql-db      latest   08c2d6d5b163   3 days ago    544MB
mysql         latest   e1d7dc9731da   3 weeks ago   544MB
openjdk       8-jdk-alpine a3562aa0b991   17 months ago 105MB
maven         3.5.2-jdk-8-alpine 293423a981a7   2 years ago   116MB

PS C:\projeto-mono\login-app>
```

Figura 35: Lista de imagens docker

E finalmente pode ser gerado o contêiner da aplicação. O contêiner, como já sabemos, é uma imagem do docker que entra em execução. O comando para executar tal operação é o *docker run* e nele passaremos o parâmetro *-p* para indicar qual porta da máquina local irá responder a porta exposta pela aplicação, definiremos um nome do contêiner e a qual imagem esse contêiner faz referência. Abaixo segue o comando executado e o resultado:

```
docker run -p 8080:8080 --name login-app login
```

```

PS C:\projeto-mono\login-app> docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
login               latest             cc9de2c4b4a       40 minutes ago    161MB
mysql               latest             0819a87d9b7       41 minutes ago    209MB
mysql-db            latest             08c2d6d5b163     3 days ago        544MB
mysql               latest             e1d7dc9731da     3 weeks ago       544MB
openjdk             3.5.2-jdk-alpine  a3302ca4b991     17 months ago    109MB
maven               3.5.2-jdk         20342a981a7       7 years ago       116MB

PS C:\projeto-mono\login-app> docker run -p 8080:8080 --name login-app login
:: Spring Boot :: (v2.2.2.RELEASE)

2020-10-02 17:51:32.171 INFO 1 --- [main] com.gpch.login.LoginApplication : Starting LoginApplication v0.0.1-SNAPSHOT on bf987c6ad7c3 with PID 1 (C:/root/app.jar started by root in /root)
2020-10-02 17:51:32.177 INFO 1 --- [main] com.gpch.login.LoginApplication : No active profile set, falling back to default profiles: default
2020-10-02 17:51:33.640 INFO 1 --- [main] s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2020-10-02 17:51:33.736 INFO 1 --- [main] s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 83ms. Found 2 JPA repository interfaces.
2020-10-02 17:51:34.329 INFO 1 --- [main] org.springframework.transaction.annotation.ProxyTransactionManagementConfiguration : Bean 'org.springframework.transaction.annotation.ProxyTransactionManagementConfiguration' is not eligible for getting processed by all BeanPostProcessors (for example: not eligible for auto-proxying)
2020-10-02 17:51:34.823 INFO 1 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2020-10-02 17:51:34.840 INFO 1 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2020-10-02 17:51:34.840 INFO 1 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.29]
2020-10-02 17:51:34.914 INFO 1 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded webApplicationContext
2020-10-02 17:51:34.914 INFO 1 --- [main] o.s.web.context.ContextLoader : Root webApplicationContext: initialization completed in 2182 ms
2020-10-02 17:51:35.171 INFO 1 --- [main] o.hibernate.jpa.internal.util.LogHelper : HH000041: Hibernate Core [5.4.9.Final]
2020-10-02 17:51:35.267 INFO 1 --- [main] org.hibernate.Version : HH000001: Hibernate Commons Annotations [5.1.0.Final]
2020-10-02 17:51:35.433 INFO 1 --- [main] org.hibernate.dialect.Dialect : HH000040: Using dialect: org.hibernate.dialect.MySQL5Dialect
2020-10-02 17:51:35.589 INFO 1 --- [main] org.hibernate.dialect.Dialect : HH000040: Using dialect: org.hibernate.dialect.MySQL5Dialect
2020-10-02 17:51:36.392 INFO 1 --- [main] o.h.e.t.j.p.i.JtaPlatformInitiator : HH000040: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
2020-10-02 17:51:36.404 INFO 1 --- [main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2020-10-02 17:51:37.057 WARN 1 --- [main] jp.datasource.configuration.JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may be performed during view rendering. Explicitly configure spring.jpa.open-in-view to disable this warning
2020-10-02 17:51:37.263 INFO 1 --- [main] o.s.s.web.DefaultSecurityFilterChain : Creating filter chain: Ant [pattern='/resources/**'] []
2020-10-02 17:51:37.264 INFO 1 --- [main] o.s.s.web.DefaultSecurityFilterChain : Creating filter chain: Ant [pattern='/static/**'] []
2020-10-02 17:51:37.264 INFO 1 --- [main] o.s.s.web.DefaultSecurityFilterChain : Creating filter chain: Ant [pattern='/css/**'] []
2020-10-02 17:51:37.264 INFO 1 --- [main] o.s.s.web.DefaultSecurityFilterChain : Creating filter chain: Ant [pattern='/js/**'] []
2020-10-02 17:51:37.264 INFO 1 --- [main] o.s.s.web.DefaultSecurityFilterChain : Creating filter chain: Ant [pattern='/images/**'] []
2020-10-02 17:51:37.303 DEBUG 1 --- [main] edFilterInvocationSecurityMetadataSource : Adding web access control expression 'permitAll', for Ant [pattern='/']
2020-10-02 17:51:37.307 DEBUG 1 --- [main] edFilterInvocationSecurityMetadataSource : Adding web access control expression 'permitAll', for Ant [pattern='/login']
2020-10-02 17:51:37.307 DEBUG 1 --- [main] edFilterInvocationSecurityMetadataSource : Adding web access control expression 'permitAll', for Ant [pattern='/registration']
2020-10-02 17:51:37.310 DEBUG 1 --- [main] edFilterInvocationSecurityMetadataSource : Adding web access control expression 'hasAuthority('ADMIN')', for Ant [pattern='/admin/**']
2020-10-02 17:51:37.320 DEBUG 1 --- [main] o.s.s.w.a.l.FilterSecurityInterceptor : Validated configuration attributes
2020-10-02 17:51:37.323 DEBUG 1 --- [main] o.s.s.w.a.l.FilterSecurityInterceptor : Validated configuration attributes
2020-10-02 17:51:37.327 INFO 1 --- [main] o.s.s.web.DefaultSecurityFilterChain : Creating filter chain: any request, [org.springframework.security.web.context.request.async.NoAsyncManagerIntegrationFilter@211651, org.springframework.security.web.context.SecurityContextPersistenceFilter@30c278e, org.springframework.security.web.header.HeaderWriterFilter@7c214ec0, org.springframework.security.web.authentication.logout.LogoutFilter@68fc0f2, org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter@7d9e8ef, org.springframework.security.web.savedrequest.RequestCacheAwareFilter@3d246a3, org.springframework.security.web.servletapi.SecurityContextHolderAwareRequestFilter@99c6baa, org.springframework.security.web.authentication.AnonymousAuthenticationFilter@2410f4, org.springframework.security.web.session.SessionManagementFilter@609b546, org.springframework.security.web.access.ExceptionTranslationFilter@3851c410, org.springframework.security.web.access.intercept.FilterSecurityInterceptor@2d778add]
2020-10-02 17:51:37.541 INFO 1 --- [main] o.s.s.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicationTaskExecutor'
2020-10-02 17:51:38.570 INFO 1 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2020-10-02 17:51:38.574 INFO 1 --- [main] com.gpch.login.LoginApplication : Started LoginApplication in 7.011 seconds (JVM running for 7.649)
    
```

Figura 36: Criação do contêiner login-app

Observa-se o log e percebe-se que a aplicação java foi iniciada devido ao comando java colocado dentro do dockerfile no comando ENTRYPOINT. Lembrando que a aplicação está rodando na máquina local porém conecta-se ao serviço de banco de dados da Google Cloud. Após a execução de um comando de criação de contêiner, o Docker Desktop fornece uma interface gráfica que facilita a visualização dos contêineres ativo.

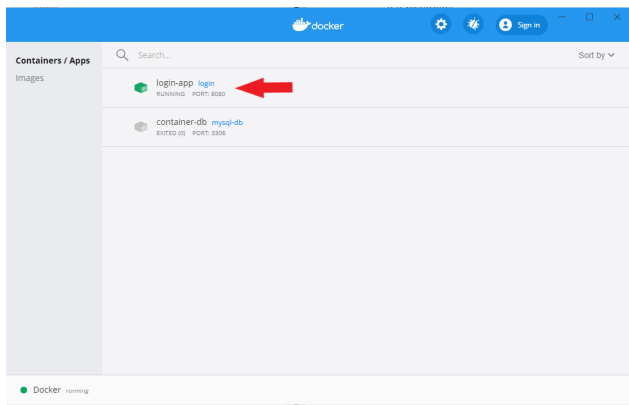


Figura 37: contêineres ativos: visualização pela interface gráfica do Docker Desktop

Na próxima seção, faremos todo esse procedimento porém, dessa vez, o contêiner criado ficará num outro serviço, um repositório de imagens da Google Cloud, e lá é que será criado um contêiner, utilizando o Google Kubernetes se terá acesso a outros recursos muito importante para a aplicação.

3.3 Kubernetes, o orquestrador do Docker

O Kubernetes é sistema de código aberto que tem como objetivo automatizar a publicação, o escalonamento e o gerenciamento de aplicações que estão containerizadas. Nele possui ferramentas que facilitam a configuração e a publicação de aplicações. Então ao realizar uma publicação, é obtido um cluster Kubernetes, que é um grupo de máquinas de trabalho, chamado de nó (nodes), estes executam aplicações containerizadas. A implementação do Kubernetes pela google é chamada Google Kubernetes Engine (GKE). Nesse ambiente, são oferecidos serviços para implantação, gerenciamento e escalonamento de aplicativos em contêineres, como gerenciamento automático, sondas de monitoramento e estabilidade para contêineres de aplicativos, escalonamento automático e atualizações contínuas. Em resumo, o funcionamento consiste em criar um cluster Kubernetes, publicar a aplicação nesse cluster, expor a aplicação para conexões públicas, pode-se escalar a publicação a depender da necessidade e por fim, atualizar a aplicação para futuras mudanças. A figura 38 ilustra essas operações:

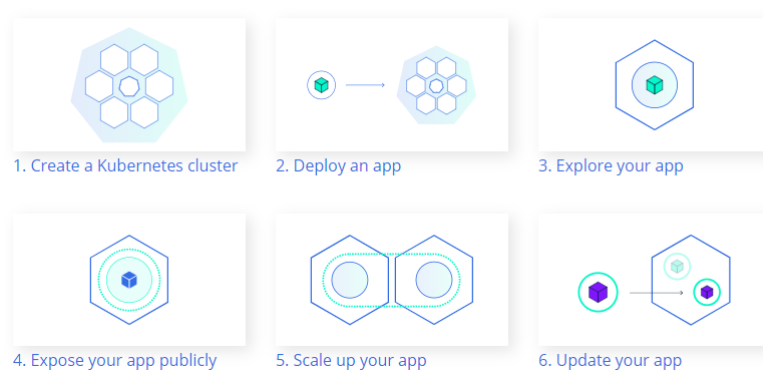


Figura 38: Resumo de funcionamento de uma publicação de um aplicação no Kubernetes

O Kubernetes não é um sistema tradicional PaaS (Platform as a Service) pois opera a nível de contêiner e não a nível de hardware como é feito normalmente. Os recursos oferecidos são semelhantes aos dos PaaS, como publicação, escalonamento, balanço de carga e monitoramento feito por usuários. Entretanto o Kubernetes não é um serviço monolítico, ou seja, essas soluções são opcionais e moduladas, ou seja, o Kubernetes prover blocos de serviço que serão utilizados conforme necessidade, preservando a flexibilidade de escolha do usuário do que é importante para ele no momento.

3.3.1 Instalação Kubernetes

A instalação da ferramenta Kubernetes será feita utilizando o SDK do Google Cloud, que é o *kit de desenvolvimento de software* para utilizar os recursos do Google Cloud (inclusive o Kubernetes). A instalação requer uma versão do Python e é feita através de um executável obtido no site da Google. Após instalado, a tela a seguir irá aparecer

para que possa executar o Google Cloud SDK Shell. Será a partir dele que será executado o comando para instalação da ferramenta do Kubernetes.

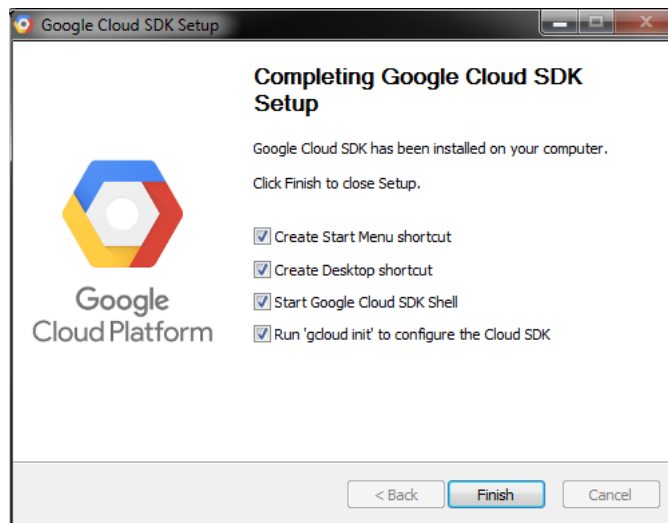


Figura 39: Instalação concluída do Google Cloud SDK Shell

Como sabemos, podemos utilizar só os módulos que necessita dentro de um projeto, no caso iremos utilizar o Kubernetes. No Google Cloud SDK Shell, será executado o comando a seguir. Este irá instalar a ferramenta de linha de comando do Kubernetes, o *kubectl*, que como já foi descrito, é o sistema de orquestração de clusters do Google Kubernetes Engine:

```
gcloud components install kubectl
```

Após instalado a ferramenta *kubectl* poderemos utilizar tanto os comandos do Kubernetes para criar o serviço da nossa aplicação de login, quanto os comandos do Google Cloud que permitem a criação de um cluster para executar o contêiner de nosso aplicativo.

3.3.2 Implantar aplicação no GKE

Para implantar uma aplicação no ambiente do GKE, é preciso primeiro realizar a criação de um projeto lá, pois precisaremos utilizar o identificador gerado para esse projeto. Feito isso, deve-se criar um cluster e definir suas configurações. O GKE possui um assistente de criação de cluster onde pode ser definido o nome, a zona ou região onde será criado. Aqui também há a opção de escalonamento automático. É importante que se autorize a utilização do *kubectl* dentro do Google Cloud, essa autorização é realizada executando o comando:

```
gcloud container clusters get-credentials [CLUSTER NAME]
```

O valor *CLUSTER NAME* refere-se ao nome do cluster que será utilizado pelo kubectl. O comando a seguir lista os clusters existentes:

```
gcloud container clusters list
```

O segundo passo, após a autorizar o kubectl no cluster que será trabalhado, é criar a imagem da aplicação. Aqui existe a diferença de quando foi criado a imagem no Docker local que é o nome da imagem que será criada. Para que o GKE enxergue essa imagem, ela precisa estar num repositório específico que se chama Google Container Registry. O comando a seguir faz o que foi descrito aqui:

```
docker build -t gcr.io/prefab-mapper-290913/login-app .
```

O valor *gcr.io/login-app* é o caminho para onde a imagem criada será enviada. O valor *prefab-mapper-290913* é o identificador do projeto criado automaticamente pelo GKE e seu diretório *gcr.io* é o diretório do Google Container Registry (GCR). O valor *login-app* é o nome atribuído a imagem dentro do GCR.

Após criada imagem com o nome correto, identificando o caminho do GCR, é necessário envia-la para o GCR com o comando *push*:

```
docker push gcr.io/prefab-mapper-290913/login-app
```

Após a execução desse comando podemos checar se a imagem foi enviada acessando a ferramenta do google. Agora deve-se criar um cluster de container. Podemos criar por linhas de comando no shell ou pela própria interface do GKE.

```
gcloud config set project stellar-perigee-253518
gcloud config set compute/zone southamerica-east1-a
gcloud container clusters create my-first-cluster --num-nodes=2
--no-enable-autoupgrade
```

Os dois primeiros comandos mudam para o projeto em questão e também define a zona de trabalho respectivamente. O terceiro comando cria o cluster e é aqui onde se define o número de nós a serem utilizados. Esse valor pode ser alterado posteriormente ou ainda pode-se utilizar um upgrade automático de acordo com a necessidade da demanda habilitando o autoupgrade. Após execução, será retornado o nome do cluster, o status e outras informações importantes sobre o mesmo. Pode-se checar os nodes daquele cluster que foram criados, nesse caso dois. No comando usa-se:

```
gcloud compute instances list
```

Percebemos aqui que temos dois nós que executam a aplicação e dividem as requisições recebidas entre eles:

NAME	ZONE	MACHINE-TYPE	INTERNAL-IP	EXTERNAL-IP	STATUS
gke-my-first-cluster-default-pool-28e362f4-6d0v	southamerica-east1-a	n1-standard-1	10.158.0.6	34.95.215.30	RUNNING
gke-my-first-cluster-default-pool-28e362f4-xd63	southamerica-east1-a	n1-standard-1	10.158.0.5	34.95.232.30	RUNNING

Agora com a imagem docker da aplicação dentro do container registry do google e o cluster com dois nós, deve-se colocar a imagem dentro do cluster para que a aplicação possa ser executada e posteriormente acessada. Para isso cria-se o que se chama de deployment:

```
kubectl create deployment login-app
  --image=gcr.io/stellar-perigee-253518/login-app
```

Assim, finaliza-se a criação de uma unidade implantável no Kubernetes. A aplicação já está em produção. Agora é necessário expor as portas de acesso para a internet pois assim poderemos acessa-la de uma conexão qualquer. Por padrão, os contêineres executados no GKE não podem ser acessados pela Internet porque não têm endereços IP externos. Então é necessário publicar explicitamente o aplicativo no tráfego da internet:

```
kubectl expose deployment login-app --type=LoadBalancer --port 80
  --target-port 8080
```

Um outro detalhe é que é preciso liberar o acesso das instâncias criadas pelo cluster para acessar o banco. Por mais que os dois estejam no mesmo ambiente (Google Cloud), ainda assim é necessário fazer essa configuração de adicionar o ip externo da instância na lista de ips públicos permitidos para acessar o Google SQL.

3.4 Gitlab e automatização da publicação

O Gitlab é uma ferramenta que utiliza a tecnologia Git. O Git, é um sistema de controle de versão de código aberto e gratuito, projetado para manter um histórico de alterações de arquivos com informações do tipo **por que, quem e quando** um arquivo foi editado. A grande vantagem de utilizar o Gitlab há outros controladores de versão, é a integração que o mesmo tem com as ferramentas do DevOps. Ele proporciona nativamente ferramentas de integração contínua e de entrega contínua que serão necessárias para a conclusão do projeto desse trabalho.

Para utilizar o Git dentro de um projeto, é necessário que se faça um uso de algum modelo de versionamento. Existem vários tipos, mas os principais são o GitFlow e o Trunked-based. O GitFlow recomenda a utilização de branches features, ou seja, para cada alteração é criada uma branch diferente o que dificulta a utilização da entrega contínua, pois, além da automatização o DevOps também é responsável pela entrega contínua de valor ao produto final (CARMO, 2018). Então, o recomendado é o Trunked-based. O modelo Trunked Based, é um modelo onde os desenvolvedores integram seu código a cada novo recurso, bugfix ou qualquer mudança para um ramo central (branch master). Essa branch, normalmente, é chamada de master.

3.4.1 Criação do pipeline

Então definido o modelo de branch, o objetivo agora é criar um pipeline que faça as publicações automáticas no Google Cloud e o Gitlab nos fornece essa integração.

O Gitlab oferece um serviço de integração contínua e também as instruções documentadas de como utilizá-lo. Segundo essa documentação, todas as mudanças incorporadas ao código da branch master disparam um gatilho para iniciar o pipeline. Tal pipeline é configurado dentro de um arquivo `.gitlab-ci.yml`, esse arquivo tem de estar dentro do projeto da aplicação e ter esse nome para ser reconhecido pelo o que chama-se de *runner*. Ou seja, basicamente o arquivo `.gitlab-ci.yml` diz o que o *runner* irá fazer, esse último é o responsável por executar as linhas de comando do arquivo yml. Após executado, podemos enxergar dentro da aplicação do Gitlab se ocorreu algum erro na execução do script ou se foi um sucesso.

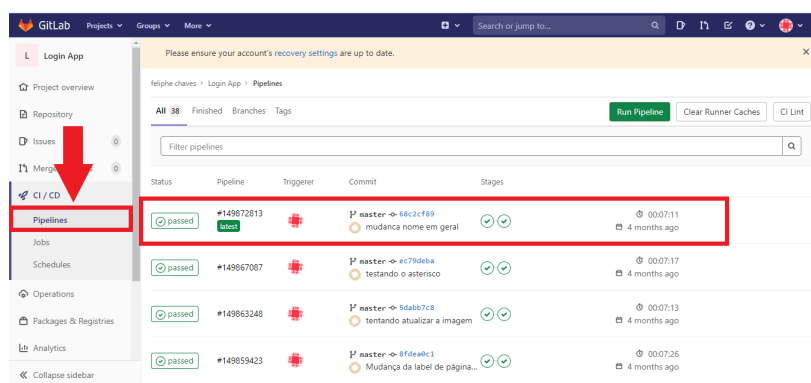


Figura 40: Lista de pipelines já executado na aplicação

Como foi descrito, quem define o que o runner do Gitlab irá fazer é o `.gitlab-ci.yml`. Esse é o arquivo que, nesse caso, o Gitlab procura e executa as linhas de comando colocadas por ele. Na documentação do Gitlab existe as regras para criação desse script e como cada linha funciona. O principal a se observar é que existem duas etapas dentro do pipeline aqui configurado, são elas build e deploy. Os comandos dentro da tag script são os comandos já

utilizados aqui porém agora eles estarão sendo feito de maneira automatizada. A tag *only* define em qual branch o código deverá comitado para disparar o gatilho que faz o build da aplicação. Ou seja, ao comitar, será executado o bloco *build* e logo em seguida o bloco *deploy*.

```
image: google/cloud-sdk:latest

variables:
  DOCKER_DRIVER: overlay
  PROJECT_ID: 'stellar-perigee-253518'
  IMAGE_NAME: 'gcr.io/stellar-perigee-253518/login-app'
  DOCKER_HOST: tcp://docker:2375
  CLUSTER_NAME: cluster-1
  CLUSTER_ZONE: southamerica-east1-a
  APP_NAME: login-app

services:
  - docker:dind

stages:
  - build
  - deploy

build:
  stage: build
  script:
    - echo $SERVICE_ACCOUNT_KEY > key.json
    - gcloud auth activate-service-account --key-file=key.json
    - gcloud auth configure-docker --quiet
    - rm -f key.json
    - docker build -t $IMAGE_NAME:latest .
    - docker push $IMAGE_NAME:latest
  only:
    - master

deploy:
  stage: deploy
  script:
    - echo $SERVICE_ACCOUNT_KEY > key.json
    - gcloud auth activate-service-account --key-file=key.json
    - gcloud --project $PROJECT_ID container clusters get-credentials
```

```

$CLUSTER_NAME --zone $CLUSTER_ZONE
- rm -f key.json
- kubectl set image deployment/$APP_NAME $APP_NAME=$IMAGE_NAME:latest
- kubectl create deployment login-app --image=$IMAGE_NAME
- kubectl expose deployment login-app --type=LoadBalancer --port 80
  --target-port 8080
- kubectl get service

```

O bloco Build realiza a autenticação dentro do Google Cloud e para essa autenticação ser realizada com sucesso é necessário uma chave gerada anteriormente, então após utilizada, por segurança a mesma é excluída. Após exclusão da chave, há a criação da imagem da aplicação e o envio para o Google Container Registry. Ao entrar no bloco Deploy e depois de realizar a mesma autenticação necessária no bloco Build, o comando muda a imagem dentro do nó do Kubernetes. Após isso, é criado o serviço da aplicação e exposto as suas portas para a internet. O próprio gitlab dará um retorno se houve algum erro ou se a publicação da aplicação foi um sucesso (figura 40).

3.5 Ciclo de vida das aplicações usando CI/CD

A aplicação de login foi alterada para utilizar o banco de dados MySQL fornecido pelo Google Cloud. Após isso, esta foi dockerizada, ou seja, conseguimos criar imagens Docker dela para posteriormente enviar ao GCR (Google Container Registry). O GKE (Google Kubernetes Engine) enxerga o repositório de imagens do google (GCR) e consegue fazer uso dessas imagens para a criação das publicações (deployment) através dos comandos utilizados na plataforma kubectl.

A princípio foi feito essa publicação manualmente para ilustração, mas posteriormente foi criado um ciclo de vida (pipeline) dentro do Gitlab para executar todos esses passos de maneira automatizada.

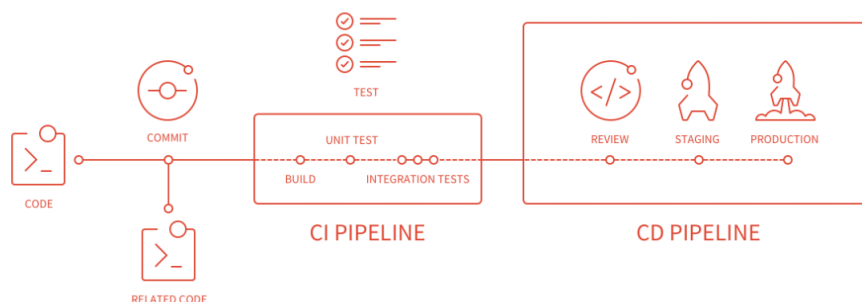


Figura 41: Ciclo de vida resumido de todas as etapas da publicação

Então, no final, conseguimos fazer a integração de código pelo programador que executa um *push* no git. O push dispara o pipeline criado no Gitlab e o mesmo executa

todos esses passos. O pipeline é configurado dentro do arquivo `gitlab-ci.yml` anexado aqui no presente trabalho. A figura 41 resume o processo implementado aqui.

4 Conclusão

O trabalho desenvolvido aqui conseguiu alcançar o objetivo proposto. Foi realizado a construção de uma infraestrutura que utiliza os conceitos DevOps e de Integração Contínua e Entrega Contínua e sempre utilizando ferramentas e recursos da nuvem.

A forma como esse trabalho foi construído e o desenvolvimento desse modelo de infraestrutura exigiu que se aplicasse muitos conceitos teóricos a prática, assim, surgiram diversas situações não previstas devido a integração de diversas ferramentas diferentes que aumentou o nível de complexidade deste.

O DevOps é um modelo cultural e estrutural a ser implantado dentro de um projeto ou organização. É focado no melhoramento dos processos integração de mudanças nas aplicações e também é direcionado para a otimização da entrega de valor ao produto. As ferramentas e recursos que este dispõe são todas voltadas para esse objetivo e é altamente ligado a utilização de recursos da nuvem.

O Docker consegue simplificar os elementos de infraestruturas necessários para se publicar uma aplicação na internet. Ao unir esses benefícios ao Kubernetes do Google percebemos que existe um controle maior das aplicações que são colocadas em ambientes de produção e que podemos aumentar a escalabilidade da aplicação, por exemplo, com apenas simples comandos como demonstrado.

É perceptível um incremento significativo na agilidade de entrega que a utilização da ferramenta de controle de versão Gitlab trouxe ao fazer uma implementação de um ciclo de vida de CI / CD para automatizar as integrações de código ao ramo principal da aplicação e em seguida já publicar tais alterações no ambiente. O ciclo de vida de integração contínua está pronto, porém é válido a adição de testes automatizados para que se garanta a confiabilidade do código a ser entregue.

Na computação não podemos definir uma única solução para diversos tipos de problemas, mas, nesse trabalho, consegue-se concluir que a união do DevOps com ferramentas disponíveis em ambientes On-cloud trás muitas vantagens para projetos em que não há problemas em mudança cultural e de seus processos já existentes.

Referências

- BRAGA, F. A. M. Um panorama sobre o uso de práticas devops nas indústrias de software. 2015.
- CARMO, G. M. do. *Por que trunk-based é o melhor modelo de desenvolvimento para DevOps*. 2018. Online. <https://churrops.io/2018/04/09/por-que-trunk-based-e-o-melhor-modelo-de-desenvolvimento-para-devops/>.
- CL9. *On Premise x Cloud Computing: qual é o melhor para sua empresa?* 2020. Online. <https://cl9.com.br/on-premise-x-cloud-computing-qual-e-o-melhor-para-sua-empresa>.
- DIEDRICH, C. dos S. *Como essa tecnologia pode alavancar suas aplicações*. 2017. Online. <https://blog.umbler.com/br/containers-101-como-essa-tecnologia-pode-alavancar-suas-aplicacoes/>.
- DOCKER. *Docker Desktop WSL 2 backend*. 2020. Online. <https://docs.docker.com/docker-for-windows/wsl/>.
- DOCKER. *Dockerfile reference*. 2020. Online. <https://docs.docker.com/engine/reference/builder/>.
- GOMES, P. C. T. *AFINAL, O QUE É DOCKER?* 2018. Online. <https://www.opservices.com.br/o-que-e-docker/: :text=De>
- HAT, R. *O que é um hipervisor?* 2020. Online. <https://www.redhat.com/pt-br/topics/virtualization/what-is-a-hypervisor>.
- IBM. *The concept of “cloud computing” has been around much longer than you think. Let’s dive into its history*. 2017. Online. <https://www.ibm.com/cloud/blog/cloud-computing-history>.
- MANDIC. *On Premises vs Cloud Computing*. 2020. Online. <https://blog.mandic.com.br/artigos/on-premises-vs-cloud-servers/>.
- NAIR, S. *What is CICD — Concepts in Continuous Integration and Deployment*. 2018. Online. <https://medium.com/@nirespire/what-is-cicd-concepts-in-continuous-integration-and-deployment-4fe3f6625007>.
- PONCE, G. *Spring Boot + Spring MVC + Spring Security + MySQL*. 2016. Online. <https://medium.com/@gustavo.ponce.ch/spring-boot-spring-mvc-spring-security-mysql-a5d8545d837d>.
- SENAPATHI JIM BUCHAN, H. O. M. Devops capabilities, practices, and challenges: Insights from a case study. *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering*, p. 57–67, 2018.
- SOLUÇÕES, D. *ON-PREMISE OU CLOUD: EM QUAL INVESTIR?* 2020. Online. <https://www.dallasolucoes.com.br/cloud-ou-on-premise-em-qual-ambiente-investir/>.