SOCIEDADE EDUCACIONAL DO VALE DO ITAJAÍ MIRIM – ASSEVIM
FACULDADE DO VALE DO ITAJAÍ-MIRIM – FAVIM
GRUPO UNIASSELVI/ASSEVIM

VITOR SOBERANSKI

Kubernetes e Seus Benefícios

Brusque,
2022-1

SOCIEDADE EDUCACIONAL DO VALE DO ITAJAÍ MIRIM – ASSEVIM
FACULDADE DO VALE DO ITAJAÍ-MIRIM – FAVIM
GRUPO UNIASSELVI/ASSEVIM

VITOR SOBERANSKI

Kubernetes e Seus Benefícios

Monografia apresentada como requisito parcial para a aprovação na disciplina de TCC – Trabalho de Conclusão de Curso – no Curso de Sistema de informação do Grupo UNIASSELVI/ASSEVIM.

Orientador: Prof. Fernando César de Ornelas.

Brusque,
2022-1

SOCIEDADE EDUCACIONAL DO VALE DO ITAJAÍ MIRIM – ASSEVIM
FACULDADE DO VALE DO ITAJAÍ-MIRIM – FAVIM
GRUPO UNIASSELVI/ASSEVIM

VITOR SOBERANSKI

Kubernetes e seus benefícios

Trabalho de Conclusão de Curso, apresentado ao curso de graduação de Sistema de informação, da Sociedade Educacional do Vale do Itajaí Mirim – ASSEVIM.

AVALIADO EM ___/ /

Orientador: Fernando César de Ornelas.

Brusque,
2022-1

RESUMO

Com aumento na demanda de Com o aumento na demanda de *cloud computing* e as aplicações funcionando cada vez mais em forma de microsserviços, algumas dificuldades começaram a surgir para gerenciar o ambiente que hospedam essas aplicações, para resolver facilitar essa administração algumas ferramentais começaram a surgir, por exemplo o Kubernetes . O Kubernetes é uma plataforma de código aberto para gerenciar cargas de trabalho e serviços distribuídos em contêineres, ele facilita tanto a configuração quanto a automação. Ele possui um ecossistema grande e de rápido crescimento. Serviços, suporte, e ferramentas para Kubernetes estão amplamente disponíveis. Seus principais benefícios são velocidade, escalabilidade, abstração da infraestrutura e eficiência. O Kubernetes se torna muito útil no momento de administrar um ambiente com vários microsserviços e de alta carga. Desta forma, aqui está apresentado o Kubernetes, alguns de seus benefícios expondo alguns conceitos e ferramentas da *cloud computing.*

Palavras chaves: Kubernetes, Docker, Virtualização.

ABSTRACT

With the increase in demand for with the increase in demand for cloud computing and applications running increasingly in the form of microservices, some difficulties began to arise to manage the environment that host these applications, to resolve this administration some tools began to emerge, for example Kubernetes. Kubernetes is an open-source platform for managing distributed containerized workloads and services, it makes both setup and automation easy. It has a large and rapidly growing ecosystem. Services, support, and tools for Kubernetes are widely available. Its main benefits are speed, scalability, infrastructure abstraction and efficiency. Kubernetes becomes very useful when managing an environment with several microservices and high load. This work presents Kubernetes, some of its benefits and explains some concepts and tools of cloud computing.

Keywords: Kubernetes, Docker, virtualization.

LISTA DE FIGURAS

Sumário

1.Introdução

Com o aumento na demanda de *cloud computing* (computação em nuvem) e a evolução da arquitetura de software os meios de provisionar e administra a infraestrutura em nuvem também tiveram que evoluir.

No início as aplicações web eram projetadas e implementadas utilizando uma arquitetura de software monolítica, ou seja, era uma aplicação única um único código fonte que rodava em uma máquina host. Com o passar do tempo a arquitetura de software utilizada passou a ser de microsserviços. Uma aplicação construída no formato de microsserviços é uma aplicação composta por vários microsserviços que funcionam de forma independente e com objetivos muito específicos e que juntos formam a aplicação, cada microsserviço pode ser desenvolvido e implantado de forma autônoma.

Existem várias formas de executar esse microsserviços como em uma máquina física, uma máquina virtual ou em um container. A utilização de containers para rodar o microsserviços é a forma mais recente e veio para solucionar alguns problemas que apareceram com a alta demanda de *cloud computing*. Os containers são um ambiente vagamente isolado do sistema operacional da máquina host, com tudo que o microsserviço precisar para funcionar, esses containers são gerenciado pelo Docker.

Em uma aplicação com poucos microsserviços os containers já resolvem o problema, porém em uma aplicação com vários microsserviços se torna muito difícil gerenciar o ambiente de infraestrutura e garantir a alta disponibilidade. Uma solução para esse problema será o foco desse estudo, que é apresentar o Kubernetes, projeto de código aberto criado pela Google. Kubernetes é um orquestrador de containers com objetivo de garantir a estabilidade e automação e alta disponibilidade do ambiente.

Neste trabalho poderá ser visto como o Kubernetes ajuda a resolver alguns problemas de escalabilidade e confiabilidade de uma infraestrutura. Para isso foi realizada uma revisão bibliográfica explicando alguns conceitos de infraestrutura e Kubernetes em seguida apresentada a metodologia de pesquisa. Para apresentar o assunto, foi elaborado um conteúdo explicando como subir um *cluster* Kubernetes com uma aplicação funcionando. Finalizando com uma conclusão.

1.1. Caracterização do problema

Nos últimos tempos a busca por serviços online vem crescendo muito, como por exemplos os aplicativos de *delivery*, aplicativos de transporte, serviços de *streaming* e vários outros. Para conseguir disponibilizar todos esses serviços com alta disponibilidade e segurança a arquitetura de software e a *cloud computing* vem evoluindo muito e trazendo novas arquiteturas e tecnologias como, Arquitetura de microsserviços e ambientes vagamente isolados chamado de containers.

Gerenciar uma aplicação com muitos microsserviços rodando em vários ambientes de forma manual sem nenhuma automação exige muita mão de obra, o que acaba por se tornar inviável, visto que um ambiente automatizado com orquestração de container exigiria menos mão de obra e garantiria segurança, confiabilidade e alta disponibilidade.

1.2. Justificativa

O docker tem como objetivo a gerência do container, ele gerencia desde a criação, administração, e finalização do container, porém o Docker não é responsável pela automação e resiliência do ambiente. O Kubernetes permite criar um ambiente com orquestração de containers, e possibilita a automatização e dimensionamento desse ambiente (MONTEIRO et al, 2019). Segundo Hightower, Burns e Beda (2019) os principais benefícios que o Kubernetes traz são velocidade, escalabilidade, abstração da infraestrutura e eficiência.

> O Kubernetes foi criado para mudar radicalmente a maneira como os aplicativos são criados e implantados na nuvem. Fundamentalmente, ele foi projetado para dar aos desenvolvedores mais velocidade, eficiência e agilidade. Esperamos que as seções anteriores tenham dado uma ideia de porque você deve implantar seus aplicativos usando o Kubernetes. Agora que você está convencido de que, nos capítulos seguintes vão te ensinar *como* para implantar seu aplicativo (HIGHTOWER, BURNS, BEDA, 2019).

Mesmo utilizando docker para executar uma aplicação, administra os ambientes ainda é uma tarefa bem complicada, principalmente se a aplicação contiver muitos microsserviços(HIGHTOWER, BURNS, BEDA, 2019).

1.3. Objetivos do Trabalho

1.3.1. Objetivo geral

Apresentar a ferramenta Kubernetes, introduzindo os conceitos e seu funcionamento na prática, mostrando algumas diferenças sobre as outras ferramentas utilizadas na *cloud computing*, na intenção de mostrar seus benefícios e a importância do Kubernetes para a *cloud computing*.

1.3.2. Objetivo específicos

- Apresentar conceitos do Kubernetes
- Explicar o funcionamento do Kubernetes
- Montar um cluster Kubernetes configurado utilizando Minikube
- Disponibilizar aplicação web já existente para funcionar neste *cluster* Kubernetes.
- Avaliar e apresentar o quanto Kubernetes ajuda na administração da infraestrutura.

2. Revisão bibliográfica

Esse capítulo irá conter alguns tópicos que ajudaram na compreensão do assunto como um todo. Para o leitor entender os benefícios do Kubernetes é interessante que saiba alguns conceitos de sistemas operacionais, máquinas virtuais, arquitetura de software e docker.

2.1. Sistemas operacionais

Segundo Denardin e Barriquello (2019, p.42) “Um sistema operacional (SO) é um software ou conjunto de softwares cuja função é servir de interface entre um sistema microprocessado e o usuário.”

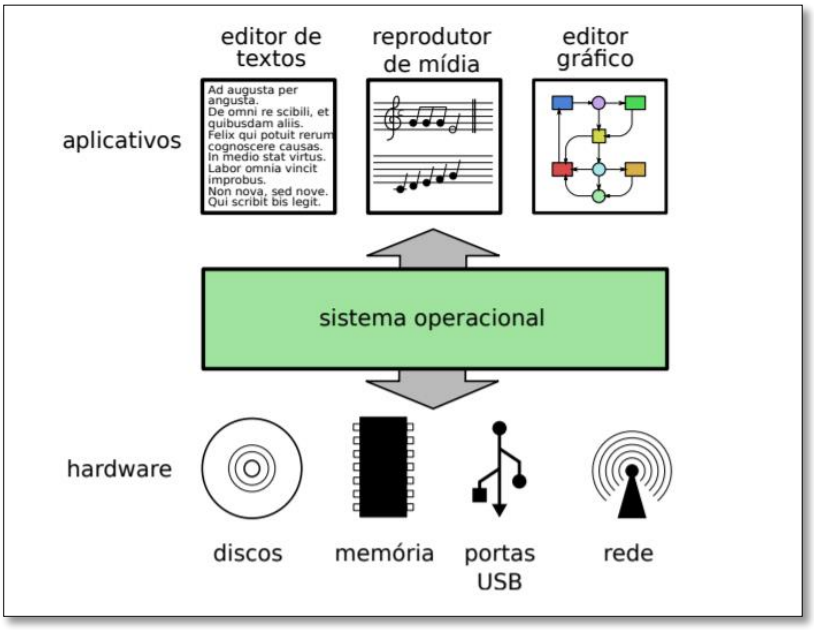
O Sistema operacional atua em uma camada de software entre o hardware e as aplicações que o usuário está utilizando. É uma estrutura de software complexa que atua desde o baixio nível, gerenciando recursos físicos da máquina (como disco memória física, periféricos), também atua no alto nível gerenciando aplicações e interfaces gráficas (MAZIERO, 2019). A figura 1 demonstra onde o sistema operacional se encontra em relação ao hardware.

Para Denardin e Barriquello (2019, p.42) existem duas formas de se conceituar um sistema operacional:

> Visão top-down: pela perspectiva do usuário ou projetista de aplicativos, provendo abstração do hardware, ou seja, fazendo o papel de intermediário entre o aplicativo e o hardware do sistema microprocessado;
>
> Visão bottom-up: opera como gerenciador de recursos, ou seja, controla quais tarefas podem ser executadas, quando podem ser executadas e que recursos (display, comunicação etc.) podem ser utilizados.

Para conseguir atingir seus objetivos os programa e aplicativos utilizam os recursos do *hardware*, processar uma informação, ler um arquivo, editar e imprimir um arquivo. Essas atividades podem ocorrer simultaneamente e as vezes pode haver conflitos, isso ocorre quando dois aplicativos precisam do mesmo recurso para funcionar. O sistema operacional é que resolve esses conflitos, definindo políticas que gerenciam esses recursos (MAZIERO, 2019).

Figura 1 Exemplificação de um sistema operacional



Fonte: Maziero, 2019

2.2. Linux

Linux é um sistema operacional que foi criado em 1991 por Linus Torvalds na universidade de Helsinki na Finlândia, hoje é mantido por uma comunidade mundial e até mesmo por grandes empresas (CAMPOS, 2006).

O linux é um sistema operacional de código aberto e é distribuído pela internet gratuitamente (SILVA, 2007).

> Seu código fonte é liberado como Free Software (software livre), sob licença GPL, o aviso de copyright do kernel feito por Linus descreve detalhadamente isto e mesmo ele não pode fechar o sistema para que seja usado apenas comercialmente (SILVA, 2007, p.4).

Com isso você não precisa pagar nada para utilizar o linux e pode ficar à vontade para criar copias do sistema e instalar em outros computadores. Ser um sistema de código aberto pode justificar a performance, estabilidade e velocidade que os recursos são implementados no sistema(SILVA, 2007).

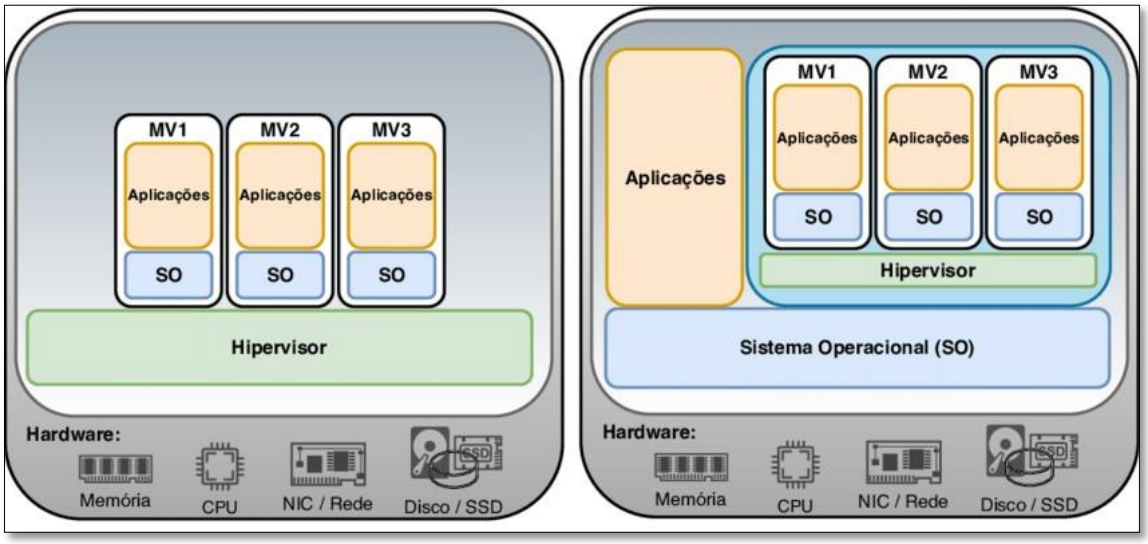
Um código fonte aberto significa que qualquer uma pode analisar seu funcionamento, analisar todos os detalhas do código, corrigir problemas do código, realizar sugestões de melhorias, e por isso consegue evoluir e crescer de forma rápida e aumentar sua compatibilidade com periféricos e hardwares (SILVA, 2007).

2.3. Máquinas virtuais

Uma máquina virtual é um tipo de virtualização que cria uma abstração dos recursos da máquina, basicamente é um sistema operacional rodando em cima de outro sistema operacional de forma isolada sem compartilhar processo com outras máquinas virtuais conforme pode ser observado na figura 2. O usuário pode acessar diretamente uma máquina virtual que está rodando dentro de outra máquina (EVEO, 2020).

> Nesse sentido, a partir de um gerenciador que funciona como uma interface, o hardware principal é dividido em vários pools de recursos, o que permite uso deles para objetivos diferentes e específicos. É uma solução que particiona a capacidade do CPU, da memória, dos recursos de rede, do disco rígido e outros componentes. Ou seja, cada parte desses recursos é separada para cada uma das máquinas, chamadas de guests. Contudo, todas essas partes ainda pertencem fisicamente a uma mesma máquina, o host (EVEO, 2020).

Figura 2 Representação dos dois principais tipos de virtualização



Fonte: researchgate, 2021

Atualmente as máquinas virtuais ainda são bem uteis no dia a dia, porém a algumas desvantagens no uso delas, uma das desvantagens é que o uso de um sistema operacional em uma máquina virtual sempre será mais lento que um sistema operacional em funcionamento direto sobre o hardware (SILVA, 2012).

Uma outra desvantagem das máquinas virtuais é o fato de que cada máquina virtual possui seu próprio Kernel e bibliotecas, isso quer dizer que cada vez que for necessário provisionar uma nova máquina virtual será provisionado com ela mais um Kernel e bibliotecas, isso deixa o processo mais lento(GARCIA, PEREIRA 2019).

2.4. Arquitetura de software

Segundo Rocha (2018) arquitetura de software é um conjunto de decisão que compõe a estrutura de um software, seus elementos estruturais seus elementos de interface

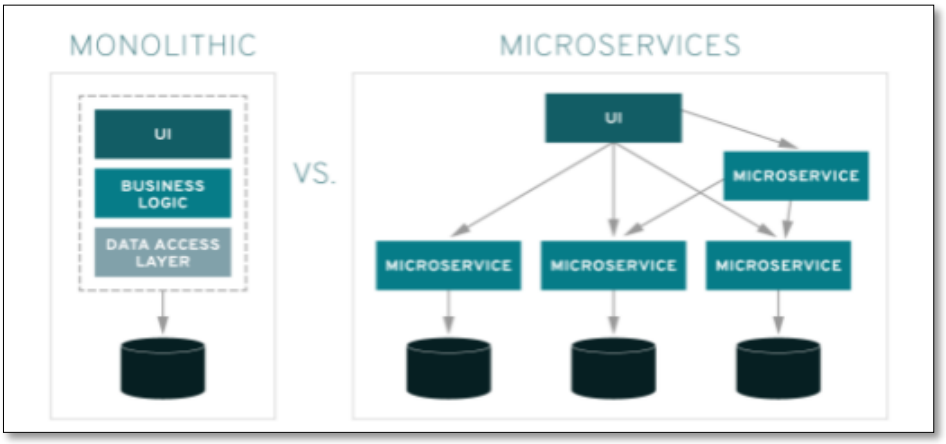
> Arquitetura de software: é um conjunto de decisões significativas sobre a organização de um sistema de software, a seleção dos elementos estruturais e suas interfaces pelas quais o sistema é composto, com seu comportamento, como especificado nas colaborações entre esses elementos, a composição desses elementos estruturais e comportamentais em subsistemas progressivamente maiores, e o estilo arquitetural que dirige essa organização – esses elementos e suas interfaces, suas colaborações e sua composição (ROCHA, 2018)

2.4.1 Microsserviços

Um microsserviço é um serviço que roda de forma independente e junto de vários outros serviços que formam uma aplicação (STOIBER, 2017). Microsserviço são serviços autônomos com tamanho reduzido, com suas responsabilidades bem especificas e muito bem definidas. Os microsserviço podem trabalhar em conjunto, porém não possuem Interdependência(coisas ligadas entre si por uma recíproca dependência, em virtude da qual realizam as mesmas finalidades pelo auxílio mútuo). Utilizar a arquitetura de microsserviços em uma aplicação proporciona flexibilidade na escolha de tecnologias e otimiza os tempos para implementar novas funcionalidades (SANTOS, 2020).

Quando se divide uma aplicação em vários pequenos pedaços e que cada pedaço é responsável por uma parte do sistema, cada pequeno pedaço pode ser implantado e escalonado de forma independente, pode ser feito por equipes deferentes, escrita por linguagens de programação diferente e testada de forma independente (STOIBER, 2017).

Figura 3 Diferença entre monolito e microsserviços



Fonte: researchgate, 2021

2.5. Docker

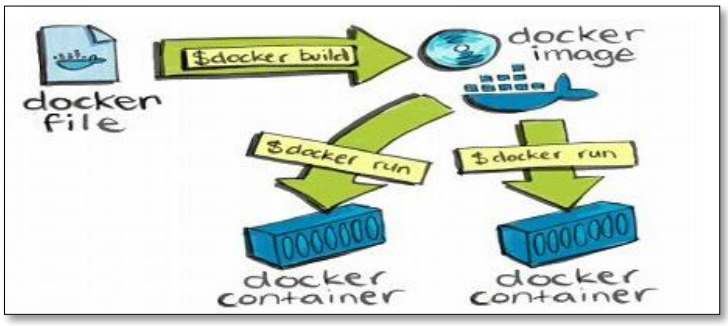
Docker é uma plataforma aberta que é utilizada para o desenvolvimento, envio e execução de aplicações. Para que você consiga entregar o seu software de forma rápida o Docker permite que você separe a aplicação de sua infraestrutura, utilizando o Docker você pode gerenciar sua infraestrutura do mesmo jeito que gerencia a sua aplicação. Aproveitando as metodologias do Docker no ambiente de desenvolvimentos para enviar, testar, e implantar o código da aplicação, é possível que você consiga uma boa redução no tempo entre a escrita do código e execução dele em produção(Docker, 2020).

O Docker consegue empacotar e executar a aplicação em um ambiente vagamente isolado, esse ambiente é denominado de container, na figura 4 consegue-se ver como isso é feito. O Docker permite a execução de vários containers simultaneamente em um determinado host (Docker, 2020).

O docker roda nativamente em linux e segundo Vitalino e Castro (2018) “ o Docker utiliza algumas *features* básicas do *kernel* Linux para seu funcionamento” . O docker utiliza as seguintes *features* do *kernel* linux: *cgroups*, *namespace*, *selinux*, *netfilte*, *capabilities*, *apparmor*, *netlink (VITALINO; CASTRO, 2018)*.

Mesmo o docker utilizando essas features do linux segundo Vitalino e Castro (2018) atualmente o docker já pode ser instalado e utilizado em outras plataformas como Windows e MacOS, porém sem a mesma estabilidade e performance do Docker rodando no Linux.

Figura 4 Componentes Docker

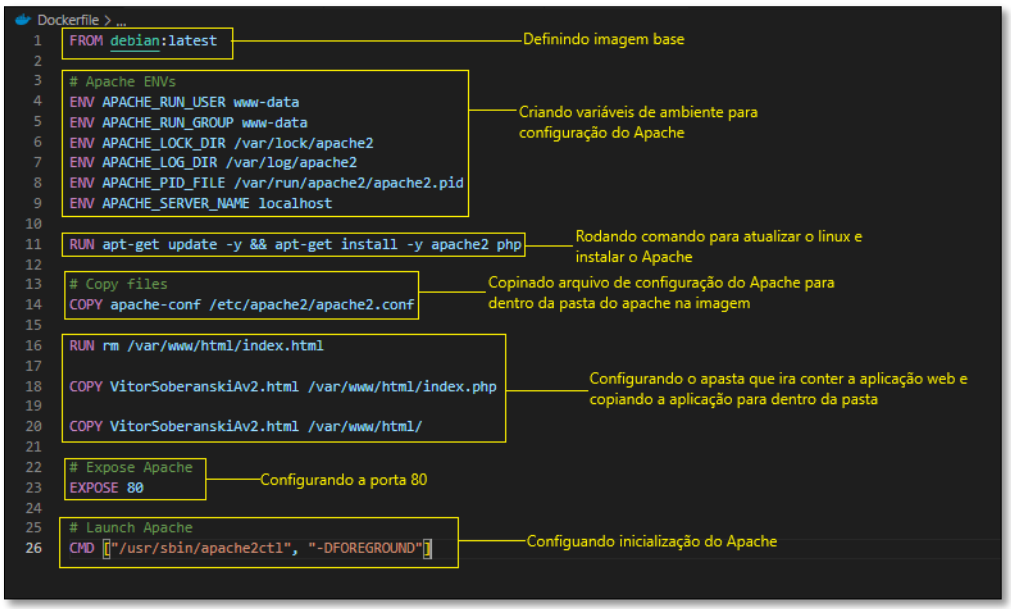


Fonte: Portal Tecnisys (2021)

2.5.1. Dockerfile

Dockerfile nada mais é do que um arquivo que contém todas as instruções através de comandos para a construção de uma imagem docker, com ele você define a imagem base que será utilizada e o que será acrescentado a essa imagem, como por exemplo na figura 5, um servidor web apache, suas configurações e a aplicação que o apache irá rodar(Diedrich, 2015)

Figura 5 Explicando *dockerfile*



Fonte: Elaborado pelo autor (2021)

2.5.2. Imagem docker

Uma imagem docker é um objeto *read only* base para se subir um container, uma imagem docker contém tudo que é preciso para o container

funcionar, por exemplo um sistema operacional Debian com uma *web server* Apache e o código do site. Como dito acima uma imagem docker só tem a camada leitura e quando se utiliza essa imagem para subir um container, o container adiciona a camada de escrita (SILVA 2017).

> Uma imagem nada mais é do que uma representação imutável de como será efetivamente construído um container. Por conta disso, nós não rodamos ou inicializamos imagens, nós fazemos isso com os containers(Artine, 2019)

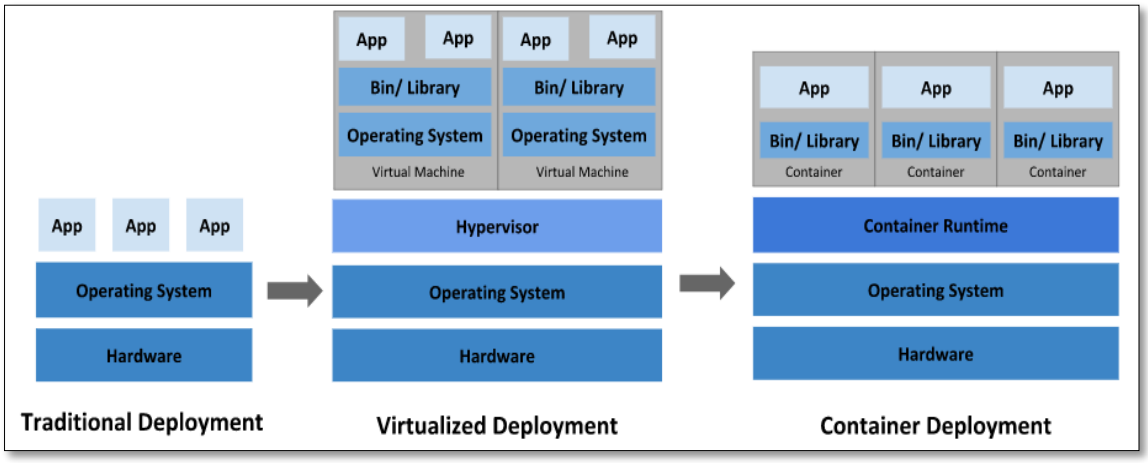
2.5.3. Container

Os containers são ambientes isolados que rodam dentro de um sistema operacional. Containers tem compatibilidade com vários sistemas operacionais sendo eles Windows, Linux ou Mac OS, e obrigatoriamente 64 bits.

> O principal objetivo de um container é prover micro serviços, sem a necessidade de que todo um sistema operacional seja instalado para que este um serviço funcione corretamente, isso traz maior agilidade quando é necessário subir um novo servidor web por exemplo com o Apache ou Tomcat (GARCIA, PEREIRA 2019).

Diferente de uma máquina virtual que possui seu próprio Kernel, binário e bibliotecas, o container compartilha o kernel, binário e bibliotecas do sistema operacional em que ele está sendo executado conforme representado na figura 6 (GARCIA, PEREIRA 2019).

Figura 6 Diferenças entre máquinas físicas, virtuais, container**.**



Fonte: Documentação Kubernetes (2021)

2.6. Kubernetes

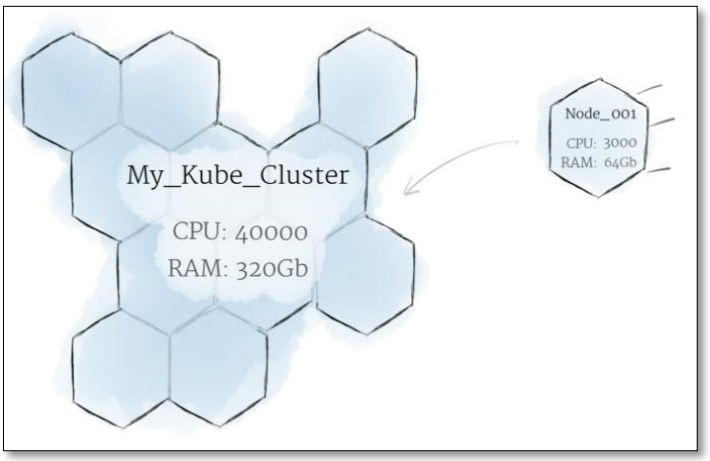
Segundo a própria documentação do Kubernetes (2021) “O Kubernetes é uma plataforma de código aberto, portável e extensiva para o gerenciamento de cargas de trabalho e serviços distribuídos em contêineres, que facilita tanto a configuração declarativa quanto a automação”.

> Uma dúvida comum e recorrente ao abordar a adoção de orquestradores de container, como o Kubernetes, decorre da eventual necessidade de sua utilização, considerando que o Docker já realiza a administração de containers. É justamente este aspecto que difere o Docker do Kubernetes. A fundamentação do Docker é a gestão do container, desde a criação, administração e finalização do mesmo. O Docker não cuida da automação e da resiliência do ambiente. O Kubernetes provê uma plataforma para automatizar a implantação, o dimensionamento e as operações de containers de aplicativos por meio de clusters de hosts (MONTEIRO et al, 2019, p9).

2.6.1. Cluster Kubernetes

Um conjunto de nós (*node*), que executam aplicações containerizadas (KUBERNETES; 2021). Um *cluster* Kubernetes é quando os nós são agrupados para formar uma unidade computacional única e com mais poder de processamento que um de seus nós separados. Quando se adiciona um node a um *cluster* os recursos desse node como por exemplo CPU, memória RAM, são somados ao *cluster* (MATOS,2022) a figura 7 exemplifica esse caso.

Figura 7 cluster Kubernetes



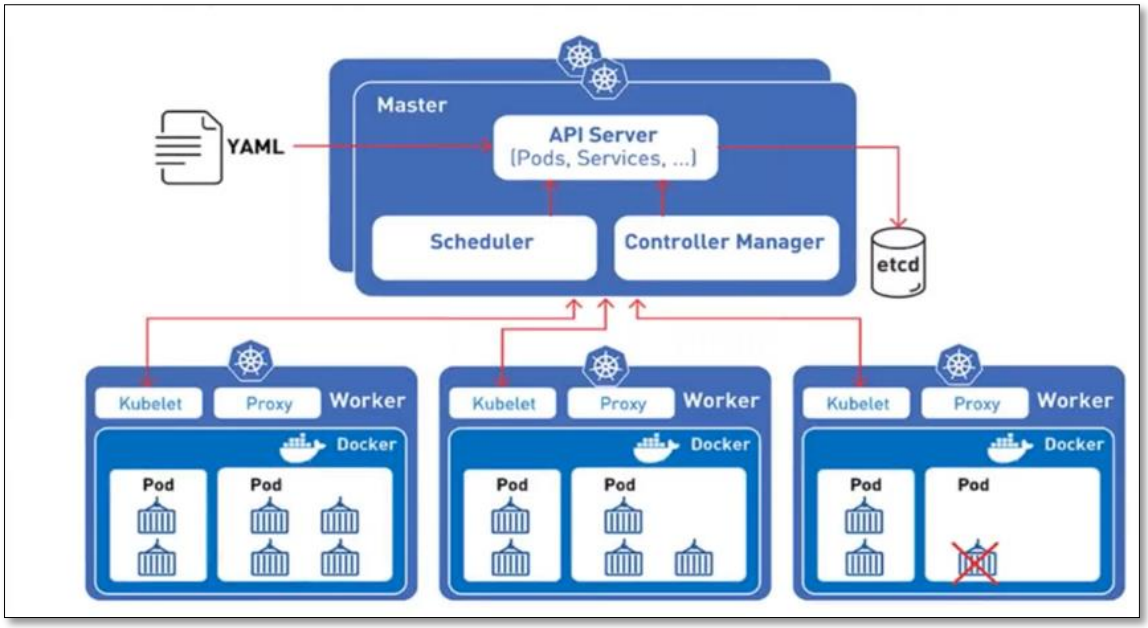
Fonte: Daniel Sanche 2018

Para montar um ambiente Kubernetes é necessário implantar um cluster de no mínimo três nós(node), sendo que um deles é *master node* e os outros dois são *worker nodes*. Para que o cluster Kubernetes funcione corretamente também é necessário que esse nós tenham instalados neles o um container runtime (por exemplo docker engine) (MONTEIRO et al, 2019).

O *Master Node* e responsável pela administração e por controlar e fazer a gestão do cluster, sendo assim ele é responsável por gerenciar os *worker nodes*. Em um ambiente de produção se recomenda sempre ter mais de uma master node para assegurar a alta disponibilidade(MONTEIRO et al, 2019).

O *Worker Node* é responsável por executar os Pods, dentro de um Pod pode haver um ou mais containers(MONTEIRO et al, 2019).

Figura 8 Cluster Kubernetes



Fonte: Developer Initiative

Em um *cluster* Kubernetes existem alguns componentes que gerenciam o *cluster* e que tomam decisões globais sobre o mesmo, também são capazes de identificar um evento e tomar medidas para resolver esse evento, alguns exemplos dessas decisões e medidas seriam, um agendamento de pods, iniciar ou finalizar pods para que atender um campo réplica de um deployment (KUBERNETES, 2021).

Esses componentes que ficam na camada de gerenciamento podem ser executados em qualquer nó do cluster e normalmente em até mais de um nó. Porém para as ficar mais simples os scripts que configuram o cluster colocam todos esses componentes no mesmo nó, esse nó não executa container dos Pods (KUBERNETES, 2021).

2.6.1.1. Kube-apiserver

O Kube-apiserver basicamente é a parte da frente da camada de gerenciamento do cluster, ele é que expõe a api do Kubernetes. Sendo a principal implementação do Kubernetes. Por ser projetado para ser escalonado horizontalmente o Kube-apiserver pode ser configurado para funcionar em mais de um nó, balanceando a carga de trabalho dele entre esses nós (KUBERNETES, 2021).

2.6.1.2. ETCD

Etcd é o armazenamento de apoio do Kubernetes, consistente e de alta disponibilidade, Etcd é um armazenamento do tipo chave-valor (KUBERNETES, 2021).

> Etcd – é um repositório de dados do Kubernetes no qual todas as configurações, as informações de tempo de execução e os status são armazenados. Os estados reais e os estados desejados de recursos são armazenados no Etcd, que é o único componente com estado nos componentes principais. O Etcd é um armazenamento de chave/valor de código aberto desenvolvido pelo CoreOS e é um dos componentes cruciais que tornam o Kubernetes confiável. O Etcd pode ser instalado em vários servidores principais, além de estar acessível dentro do cluster Kubernetes(MONTEIRO et al, 2019, p11).

2.6.1.3. Kube-scheduler

O kube-scheduler decide em que nó os novos Pods serão executados, ele analisa os requisitos do novo pod e a capacidade do nó (MONTEIRO et al, 2019). Para analisar esses dois casos ele leva em consideração os seguintes fatores: “requisitos de recursos individuais e coletivos, hardware/software/política de restrições, especificações de afinidade e antiafinidade, localidade de dados, interferência entre cargas de trabalho, e prazos” (KUBERNETES, 2021).

2.6.1.4. Kube-controller-manager

Controlador servem para gerenciar os ciclos de vida dos recursos, eles monitoram o estado desejado para que quando tenha alugam alteração eles possam ler as informações e tomar ações para que o estado desejado seja atingido (MONTEIRO et al, 2019).

Segue lista de alguns controladores de acordo com a documentação do Kubernetes(2021)

- Controlador de nó: Responsável por identificar e atuar caso algum nó pare de funcionar.
- Controlador de *job*: Responsável por criar pod para executar tarefas únicas até que elas sejam finalizadas, ele identifica essas tarefas observando os objetos job.
- Controlador de *endpoint*: Responsável por associar os serviços aos pods.

2.6.2. Nós(Node)

De acordo com a documentação do próprio Kubernetes (2021) “Um Nó é uma máquina de processamento em um cluster Kubernetes e pode ser uma máquina física ou virtual. Cada Nó é gerenciado pelo Control Plane”. Um Nó pode possuir vários pods e quem gerencia, de forma automática esses Pods dentro dos nós, é o Control Plane do Kubernetes(KUBERNETES, 2021).

Um nó Kubernetes tem vários componentes do Kubernetes executando nele, como por exemplo um kube-proxy, um runtime de contêiner (por exemplo o Docker) e um kubelet conforme mostrado na figura 9 (MONTEIRO et al, 2019).

2.6.2.1. Componente de um nó (node)

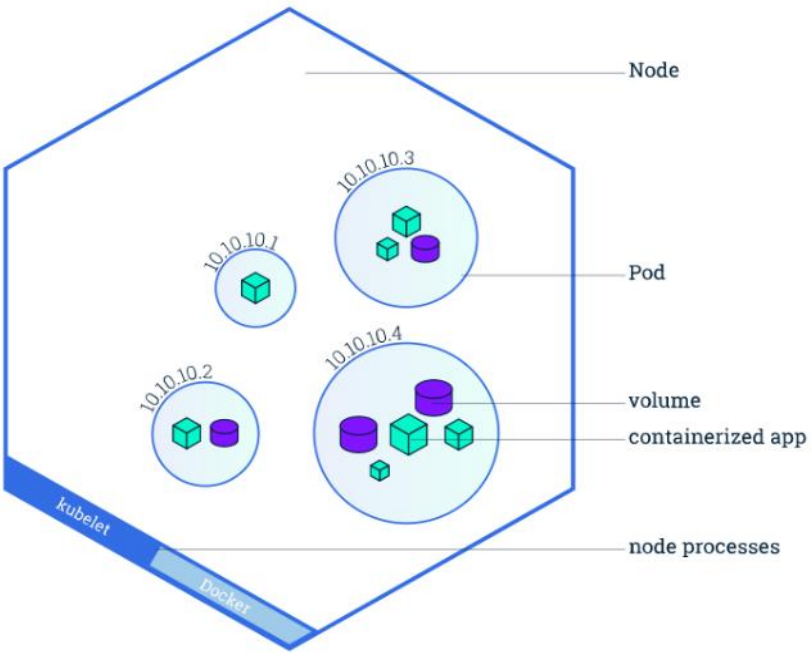
Em todos os nós existem componentes em execução para manter os pods funcionado e garantido a entrega do ambiente Kubernetes(KUBERNETES, 2021)

- **Kubelet:** Agente que garante que os containers estejam funcionando dentro de um Pod (KUBERNETS, 2021). “ O kubelet

utiliza um conjunto de PodSpecs que são fornecidos por vários mecanismos e garante que os contêineres descritos nesses PodSpecs estejam funcionando corretamente”(KUBERNETS, 2021). O Kubelet não irá gerenciar um container que não foi provisionado pelo Kubernetes (KUBERNETS, 2021).

- **Kube-proxy:** É um proxy de rede que é executado dentro dos um nós do *cluster*. “kube-proxy mantém regras de rede nos nós. Estas regras de rede permitem a comunicação de rede com seus pods a partir de sessões de rede dentro ou fora de seu cluster” (KUBERNETS, 2021).Normalmente o kube-proxy utiliza da própria camada de filtragem de pacotes do sistema operacional, caso o sistema não tenha uma ou esteja indisponível o próprio kube-proxy encaminha o tráfego.
- **Container runtime:** O software que é responsável pela execução dos containers. “O Kubernetes suporta diversos agentes de execução de contêineres: Docker, containerd, CRI-O, e qualquer implementação do Kubernetes CRI (Container Runtime Interface) (KUBERNETS, 2021).

Figura 9 Visão geral sobre um nó



Fonte: Documentação Kubernetes(2021)

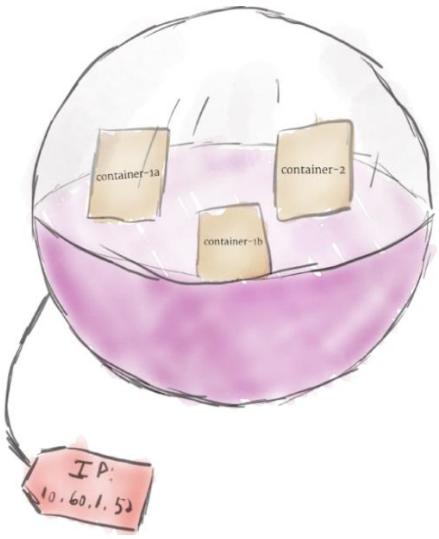
2.6.3. Pod

Pod é um conjunto de uma ou mais containers e é a menor unidade de uma aplicação Kubernetes. Na maioria das vezes os pods são compostos por um container geralmente nos casos mais simples em casos mais avançados os pods podem ser composto por vários containers altamente acoplados, isso maximiza os benefícios do compartilhamento de recursos gerenciados pelo *cluster*. Os containers que estão rodando no mesmo pod compartilham do mesmo recurso computacional. Os pods são executados dentro de um *nó* que por sua vez fazem parte de uma *cluster* (RED HAT, 2017).

Utilizando os pod o Kubernetes não precisa executar os containers diretamente. O Kubernetes executando os pods faz com que todos os containers que estão executados dentro de um pod compartilham dos mesmos recursos e da mesma rede local. Com isso os containers conseguem se comunicar com baixa latência e ainda assim manter um bom isolamento. (RED HAT, 2017).

> A organização de containers em pods é a base de uma das funcionalidades mais famosas do Kubernetes: a replicação. Quando existe esse tipo de organização, o Kubernetes pode usar controladores de replicação para escalar horizontalmente uma aplicação, conforme a necessidade. Efetivamente, isso significa que, quando um pod fica sobrecarregado, o Kubernetes pode replicá-lo e implantá-lo no cluster de forma automática. Além de ajudar a manter o funcionamento correto das operações durante os períodos de sobrecarga, os pods do Kubernetes são frequentemente replicados para que o sistema permaneça resistente a falhas(Red Hat).

Figura 10 Exemplificação do Pod

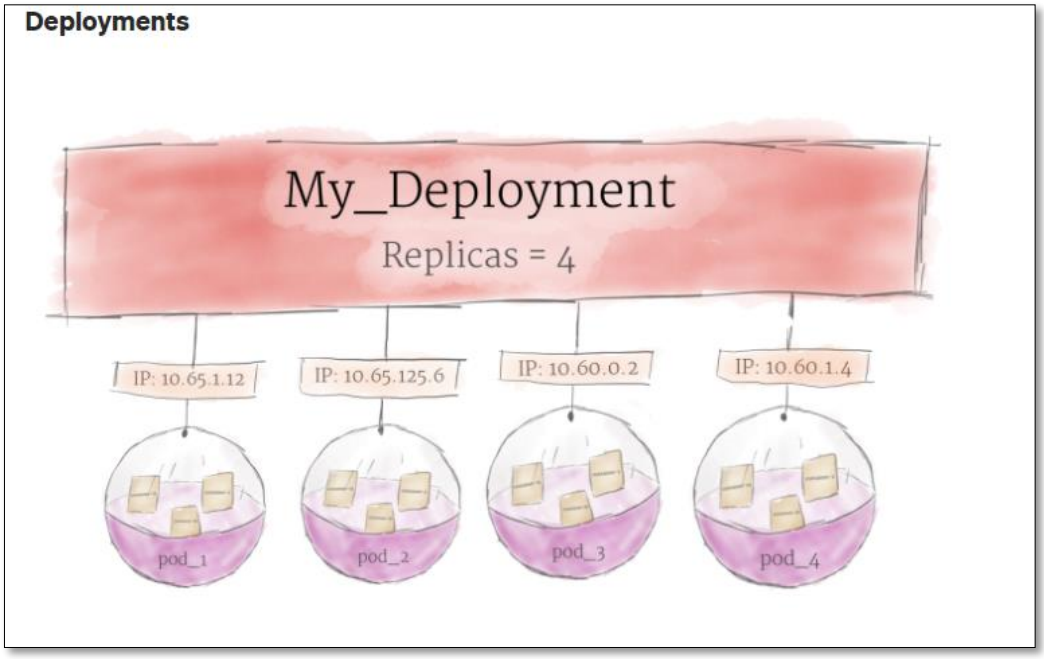


Fonte: Daniel Sanche 2018

2.6.4. Deplyment

Deployment é uma camada de abstração que gerencia os Pods. Por mais que os Pods sejam uma unidade bem básica dentro do Kubernetes, na maior parte das vezes eles não são jogados diretamente dentro do cluster. Por isso são utilizados os deployment (MATOS,2022). “O objetivo principal de um deployment é declarar quantas réplicas de um pod devem estar em execução por vez. Quando uma implantação é adicionada ao cluster, ela automaticamente aumenta o número solicitado de pods e os monitora (MATOS,2022)”. Caso um Pod venha parar de funcionar, o deployment irá automaticamente garantir que um novo Pod seja criado no lugar desse que parou de funcionar. Isso faz com que não precise lidar com Pod manualmente, basta declarar o estado desejado do sistema e o deployment irá garantir que esse estado seja atendido, a figura 9 exemplifica o caso (MATOS,2022).

Figura 11 Deployment



Fonte: Daniel Sanche 2018

2.6.5. Minikube

O Minikube e uma ferramenta que facilita a montagem de uma *cluster* Kubernetes para fins acadêmicos. Ele implanta em sua máquina um *cluster* simples contendo apenas um nó. Com foi falado acima uma *cluster* Kubernetes deve ter no mínimo 3 nós para ser utilizado em produção, por isso é importante destacar que o Minikube deve ser utilizado apenas para teste e estudo (MONTEIRO et al, 2019).

3. Metodologia

A pesquisa desenvolvida é de natureza aplicada, onde os conhecimentos obtidos durante a pesquisa bibliográfica são utilizados para explicar conceitos de *cloud computing* e Kubernetes.

Pesquisa de natureza aplicada segundo Francisco Paulo do Nascimento (2016) é dedicada a geração de conhecimento para solucionar problemas específicos, é voltada para buscar a verdade para uma aplicação pratica específica. No caso estudar os benefícios de utilizar Kubernetes em um ambiente de desenvolvimento e produção para organizar o ambiente e melhora a escalabilidade.

4. Desenvolvimento do Projeto

4.1. Explicação

O *cluster* Kubernetes rodará no sistema operacional Windows através do minikube. Como explicado anteriormente o minikube é uma maneira de montar uma *cluster* Kubernetes e tem algumas limitações por ser uma ferramenta utilizada para fins estudantis, mas atenderá as nossas necessidades para o desenvolvimento deste projeto. Importante observar que o minikube não é recomendado para ambientes de produção. O Kubernetes será configurado para rodar uma aplicação web desenvolvida pelo autor.

4.2. Instalando Minikube

As informações encontradas nesse tópico podem ser encontradas na documentação do Minikube que está nesse link: minikube.sigs.k8s.io/docs/start/ Para começar a utilizar o Minikube será necessário:

- 2 CPUs ou mais
- 2 GB de memória livre
- 20 GB de espaço livre em disco
- Conexão de internet
- Gerenciador de contêiner ou máquina virtual, como: Docker ou VMware Fusion/Workstation,

Para instalar o minikube no Windows basta baixar e executar o instalador para a versão mais recente:

https://storage.googleapis.com/minikube/releases/latest/minikube-installer.exe

Para instalar o minikube linux basta executar os comandos a seguir:

```
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
```

```
sudo install minikube-linux-amd64 /usr/local/bin/minikube
```

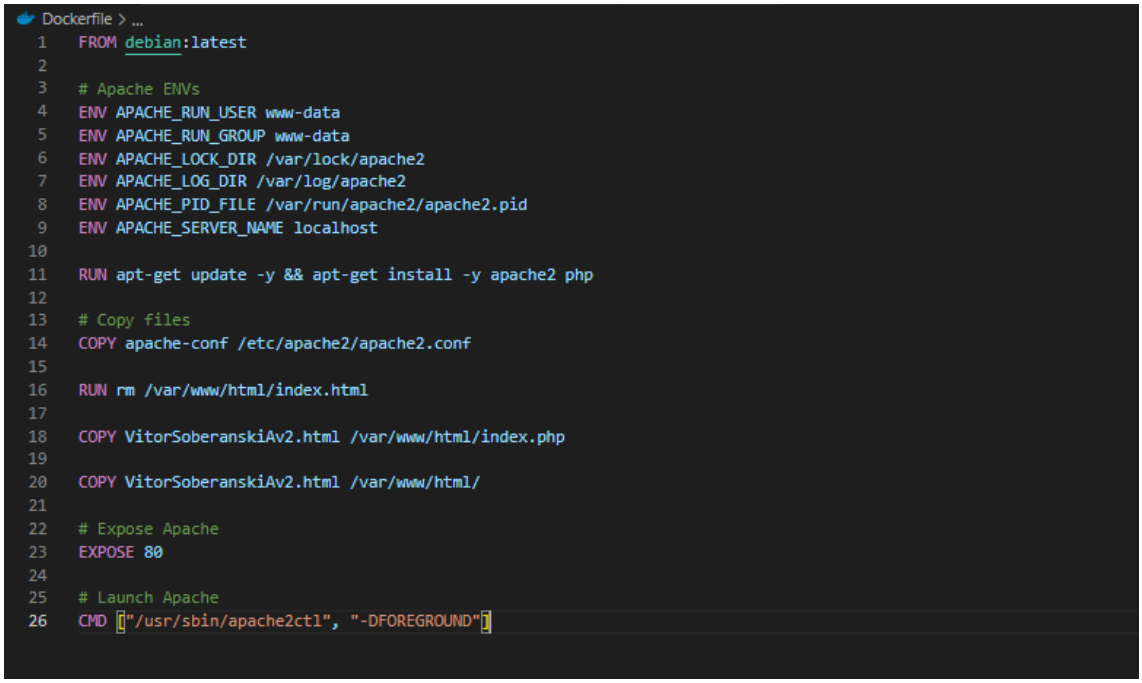
Feito as etapas tanto no linux quanto Windows é só executar o seguinte comando em um painel com acesso administrador: `minikube start`

Se o Minikube não iniciar consulte a página minikube.sigs.k8s.io/docs/drivers/ para obter ajuda na configuração.

4.3. Criando uma imagem Docker

Para começar a configurar o *cluster* é preciso criar a imagem Docker com a aplicação, para que os Pods possam rodar os containers com a aplicação. Essa imagem será criada utilizando o *dockerfile* com todas as configurações que são necessárias para rodar a aplicação, esse arquivo e demostrado na Figura 12.

Figura 12 *Dockerfile* com as configurações para cria a imagem



Fonte: Elaborado pelo autor (2021)

conforme demonstrado na Figura 13, com o *dockerfile* pronto basta executar o seguinte comando:

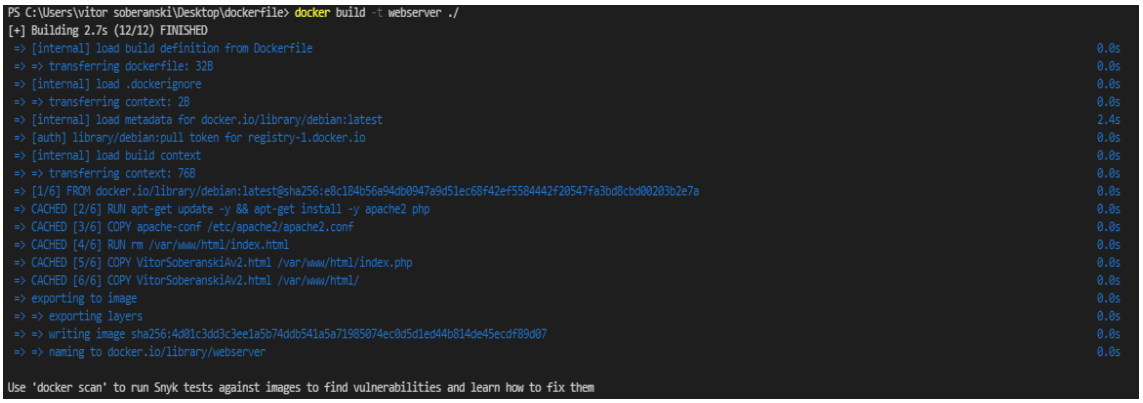
docker build -t webserver ./

Dividindo esse comando em algumas partes fica melhor exemplificando o seu funcionamento.

- **docker:** Essa primeira parte indica ao painel de comando que o comando que está sendo utilizado é um comando Docker.
- **build:** Build indica para o Docker a intenção de criar uma imagem.
- **-t webserver:** Esse é o nome que a imagem terá após ser criada antecedida por -t para o Docker entender que é um nome.
- **./:** Indica para o Docker onde está o *dockerfile* e o nome do arquivo que

ele precisa executar, nesse caso o arquivo está na pasta raiz e só tem um arquivo *dockerfile* na pasta então é só ./ mesmo.

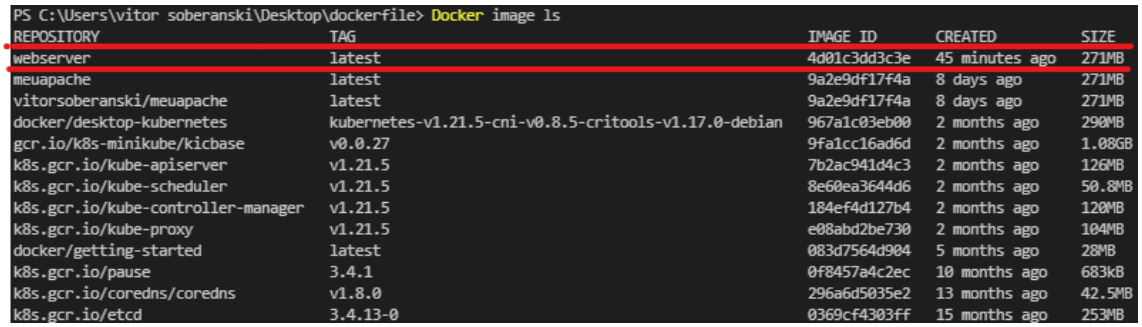
Figura 13 Comando para criar a imagem docker utilizando o *dockerfile*



Fonte: Elaborado pelo autor (2021)

Com a imagem criada é possível visualizar ela como o comando *docker image ls* conforme demonstrado na Figura 14.

Figura 14 Comado para visualizar todas as imagens criadas



Fonte: Elaborado pelo autor (2021)

4.4. Configurando e rodando o Deployment

Tendo a imagem pronta é possível criar um deployment que cuidara de subir os Pods e manter tudo rodando conforme as especificações do arquivo yaml. Para criar o deployment é preciso ter o arquivo yaml configurado de acordo com o que a aplicação precisa para funcionar conforme a Figura 15 demonstra.

Figura 15 Arquivo yaml com as configurações do deployment

Fonte: Elaborado pelo autor (2021)

Como o arquivo yaml feito é só executar o seguinte comando para criar o deployment, *kubectl create -f .\webServerDeployment.yaml* Conforme demonstrado na Figura 16.

- **kubectl:** Essa primeira parte indica ao painel de comando que o comando que está sendo utilizado é um comando kubectl.
- **create:** Indaca para o kubectl para criar o deployment.
- **-f:** Indica que será utilizado um arquivo para criação do deployment.
- **.\webServerDeployment.yaml:** Nome do arquivo yaml.

Figura 16 Executando o arquivo deployment.yaml para criar o deployment

Fonte: Elaborado pelo autor (2021)

O comando *kubectl get deployment* permite verificar se o deployment foi criado e se está funcionado, a Figura 17 demostra o resultado.

Figura 17 Status do deployment que está em execução

```
PS C:\Users\vitor soberanski\Desktop\dockerfile> kubectl get deployment
NAME            READY   UP-TO-DATE   AVAILABLE   AGE
webdeployment   1/1     1            1           40s
```

Fonte: Elaborado pelo autor (2021)

Com praticamente o mesmo comando trocando o deployment por pods *kubectl get pods* é possível visualizar os pods que o deployment criou e está gerenciando, a Figura 18 demostra o resultado.

Figura 18 Status do pod que está em execução

```
PS C:\Users\vitor soberanski\Desktop\dockerfile> kubectl get pods
NAME                             READY   STATUS    RESTARTS   AGE
webdeployment-5694874d6-54ptx    1/1     Running   0          11m
```

Fonte: Elaborado pelo autor (2021)

4.5. Criando os service

O deployment já foi criado e está rodando o pod já está rodando com a aplicação nele, e a aplicação está disponível na rede interna do Kubernetes. Para disponibilizar a aplicação em uma porta externa do Kubernetes é necessário a criação de um service(Serviço) que faça isso de forma automatizada. Os service também podem ser criados através de um arquivo yaml. A Figura 19 demonstra como o arquivo yaml de um serviço é configurado.

Figura 19 Arquivo service.yaml configurado para subir um service

services.yaml > {} spec

```yaml
apiVersion: v1
kind: Service
metadata:
  name: webdeployment-service
spec:
  selector:
    app: webdeployment
  ports:
  - protocol: TCP
    port: 80
  type: LoadBalancer

```

Fonte: Elaborado pelo autor (2021)

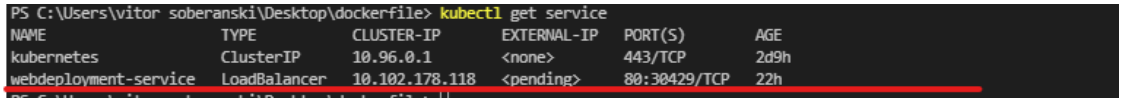
Da mesma forma que nas outras etapas com o arquivo yaml criado é necessário radar um comando para que o Kubernetes crie o service, o comando executado é o seguinte: *kubectl apply -f .\services.yaml.* A Figura 20 mostra a execução e o resultado.

Figura 20 Execução do arquivo service.yaml e o resultado

Fonte: Elaborado pelo autor (2021)

É possível visualizar o service criado utilizando o comando *kubectl get service* conforme demonstrado na figura 21.

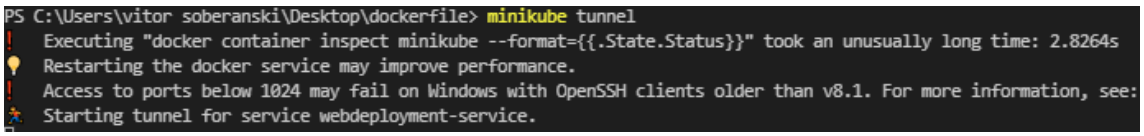
Figura 21 status do service em execução



Fonte: Elaborado pelo autor (2021)

Com o service em funcionamento já está tudo pronto a aplicação já está no ar e disponibilizada na porta 80, para acessar a aplicação web é só acessar o IP externo do service, porém repare que na Figura 21 o ESTERNAL-IP esta como pending, isso ocorre porque nosso cluster está rodando em Minikube, para contornar isso basta rodar o comando *minikube tunnel* como na Figura 22.

Figura 22 rodando o comando minikube tunnel



Fonte: Elaborado pelo autor (2021)

Com o comando minikube tunnel rondando basta rodar o comando *kubectl get service* novamente, figura 23 demonstra o caso.

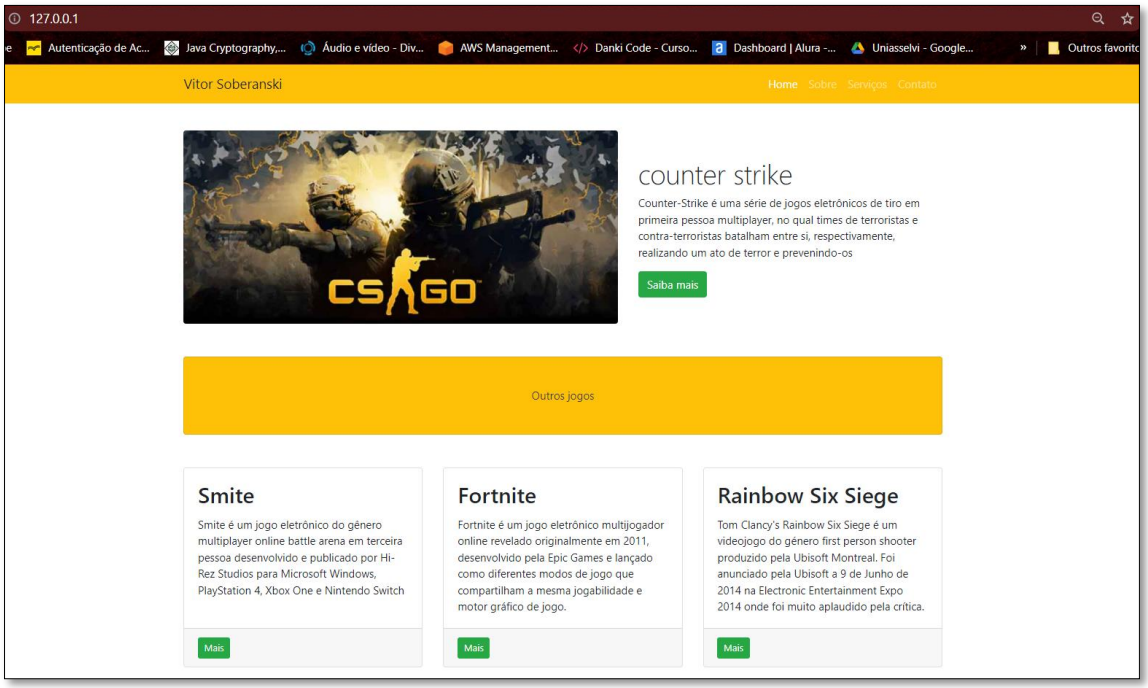
Figura 23 Status do service em execução com ip

```
PS C:\Users\vitor soberanski\Desktop\dockerfile> kubectl get service
NAME                  TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes            ClusterIP      10.96.0.1       <none>        443/TCP        35h
webdeployment-service LoadBalancer   10.102.178.118  127.0.0.1     80:30429/TCP   2m36s
```

Fonte: Elaborado pelo autor (2021)

Agora já está tudo rodando corretamente já se sabe o IP para acessar a aplicação, então é só ir ao navegador e digitar o endereço de IP que está sublinhado de vermelho na Figura 23.

Figura 24 Aplicação web rodando no cluster Kubernetes



Fonte: Elaborado pelo autor (2021)

Com isso concluísse a configuração do cluster Kubernetes e a disponibilização da aplicação web, conforme demonstrado na figura 24.

4. Considerações Finais

Perante as pesquisas realizadas e aos conceitos apresentados fica visível que Kubernetes traz alguns benefícios e ajuda na administração de uma infraestrutura cloud.

Um dos benefícios do Kubernetes é a otimização de uso dos recursos da máquina host, já que não precisa virtualizar todo um sistema operacional, visto que Kubernetes utiliza Pods como ambientes para rodar as aplicações e que dentro desses Pods temos os containers, que compartilham recursos do sistema operacional do host, ao invés de subir todo um novo sistema operacional por cima do sistema que já está rodando na máquina host.

O Kubernetes possibilita a automação do ambiente já que depois de configurado e definido o estado desejado o Kubernetes cuidara para que esse estado seja atendido, também possibilita a escalabilidade do ambiente visto que aumentando o número de Pods conseguisse aumentar o número de unidades logicas que estão rodando a aplicação. O Kubernetes também garante a confiabilidade e alta disponibilidade, isso porque se alguma coisa acontecer com o Pod responsável por uma aplicação a aplicação não ficara fora do ar por muito tempo, poque o Kubernetes identifica que o Pod não está com o funcionamento correto e sobe outro Pod no lugar desse que está com defeito.

Outro benefício bem importante do Kubernetes e a velocidade. Para subir uma nova versão da aplicação o Kubernetes não precisa subir todo um sistema operacional e esperar carregar, isso porque como dito anteriormente o Kubernetes só precisara subir o pod com os containers e nesse container só haverá o que é necessário para a aplicação, com isso o Kubernetes ganha muita agilidade.

O Kubernetes apresentou-se uma ferramenta muito útil para gerenciar uma infraestrutura e ambiente de uma aplicação como vários microsserviços, já que gerenciar tantas máquinas virtuais ou tantos containers de forma manual pode se tornar um pouco mais complicado que gerenciar tudo com Kubernetes.

REFERÊNCIAS

ARTINE, Daniel. **Docker: Desvendando o *Dockerfile***. 2019. Disponível em: https://www.alura.com.br/artigos/desvendando-o-*dockerfile*. Acesso em: 28 ago. 2019.

BURNS, Brendan; BEDA, Joe; HIGHTOWER, Kelsey. **Kubernetes: Up and Running, 2nd Edition**. O'Reilly Media, Inc., 2019.

CAMPOS, Augusto. **O que é Linux**. Florianópolis: Br-Linux, 2006. Disponível em: https://br-linux.org/2008/01/faq-linux.html. Acesso em: 20 jun. 2021.

DENARDIN, Gustavo Weber; BARRIQUELLO, Carlos Henrique. **Sistema operacionais tempo real e a sua aplicação em sistema embarcado**. São Paulo: Edgard Blucher Ltda, 2019. 474 p.

DIEDRICH, Cristiano. **O que é dockerfile**. 2015. Disponível em: https://www.mundodocker.com.br/o-que-e-dockerfile/. Acesso em: 25 maio 2022.

DOCKER. **Docker overview**. Disponível em: https://docs.docker.com/get-started/overview/#the-docker-platform. Acesso em: 26 jun. 2021

GARCIA, Walker; PEREIRA, Fagner Coin. **DOCKER – CONTAINERS NÃO SÃO VM's**. SEMINÁRIO DE TECNOLOGIA GESTÃO E EDUCAÇÃO. 2019. Disponível em: http://raam.alcidesmaya.com.br/index.php/SGTE/article/view/21. Acesso em: 23 jun. 2021.

KUBERNETES. **O que é Kubernetes?** 2021. Disponível em: https://kubernetes.io/pt-br/docs/concepts/overview/what-is-kubernetes/. Acesso em: 27 jun. 2021.

MATOS, David. **Kubernetes: Pods, Nodes, Containers e Clusters**. 2022. Disponível em: https://www.cienciaedados.com/kubernetes-pods-nodes-containers-e-clusters/. Acesso em: 11 jun. 2022.

MAZIERO, Carlos Alberto. **Sistemas Operacionais: Conceitos e Mecanismos**. Curitiba: Dinf - Ufpr, 2019. 472 p.

MONTEIRO, Luciano & Almeida, Washington & Lima, Anderson & Silva, Sahra & Escobar, Fernando. (2019). **Sustentando Microsserviços em Ambiente de Cloud Computing utilizando Kubernetes e Service Mesh**.

NASCIMENTO, Francisco Paulo do. **Metodologia da Pesquisa Científica: teoria e prática – como elaborar TCC**. 2016. Disponível em: http://franciscopaulo.com.br/arquivos/Classifica%C3%A7%C3%A3o%20da%20Pesquisa.pdf. Acesso em: 02 jul. 2021.

REDAÇÃO EVEO. **DATACENTER O que é Virtual Machine e para que serve?** 2020. Disponível em: https://www.eveo.com.br/blog/virtual-machine/. Acesso em: 21 jun. 21.

Red Hat. **O que é um pod do Kubernetes?** 2017. Disponível em: https://www.redhat.com/pt-br/topics/containers/what-is-kubernetes-pod#:~:text=Um%20pod%20do%20Kubernetes%20%C3%A9,acoplados%20em%20cen%C3%A1rios%20mais%20avan%C3%A7ados. Acesso em: 11 jun. 2022.

ROCHA, José Gladistone. **Arquitetura em Camadas com uso do Paradigma MVC e Processo Unificado na Programação de Software Orientado a Objetos**. 2018. Disponível em: http://revista.faculdadeprojecao.edu.br/index.php/Projecao4/article/view/1068. Acesso em: 21 jun. 2021.

SANTOS, Eduardo Fernandes Mioto de Oliveira dos Santos. **SUGESTÃO DE CRITICIDADE BASEADA EM MULTICRITÉRIO PARA ARQUITETURAS ESTABELECIDAS ORIENTADAS A MICROSSERVIÇOS**. 2020. 180 f. Dissertação (Mestrado) - Curso de Engenharia de Sistemas e Computação, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2020. Cap. 2.

SILVA, Flávio Henrique Rocha e. **Avaliação de Desempenho de Contêineres Docker para Aplicações do Supremo Tribunal Federal**. 2017. 80 f. Monografia (Doutorado) - Curso de Curso de Computação, Universidade de Brasília, Brasília, 2017.

SILVA, Gleydson Mazioli da. **Guia Foca GNU/Linux**., 2007. Disponível em: http://static.efetividade.net/archive/img/focalinux-iniciante.pdf. Acesso em: 20 jun. 2021.

SILVA, Lucas Martins. **Uma Máquina Virtual para uso didático**. 2012. 236 f. TCC (Graduação) - Curso de Sistemas de Informação, Universidade Federal de Santa Catarina - Ufsc, Florianópolis, 2012.

VITALINO, Jeferson Fernando Noronha; CASTRO, Marcus André Nunes. **Descomplicando o Docker**. 2. ed. São Paulo: Brasport, 2018

STOIBER, Max. **Build your first Node.js microservice**. 2017. Disponível em: https://mxstbr.blog/2017/01/your-first-node-microservice/. Acesso em: 21 jun. 2021.