Trabalho de Graduação do Curso de ADS da Faculdade de Tecnologia de São José do Rio Preto

IMPLEMENTAÇÃO DE UM AMBIENTE CLUSTERIZADO DE CONTAINERS UTILIZANDO KUBERNETES E CONTAINERD

Polisello, Breno Otavio; Gonçalves, Marcelo Augusto Silva;
Junior, Walter Gomes Pedroso; Viana, Jose Aparecido de Aguiar.

e-mail:
brenopolisello@hotmail.com; marcelo_itapa@hotmail.com
walter.pedroso@fatecriopreto.edu.br; prof.viana@fatecriopreto.edu.br

Resumo: Devido à necessidade de desenvolvedores e administradores de sistemas realizarem rapidamente o provisionamento de máquinas destinadas à execução de suas aplicações, soluções como conteinerização se sobressaíram das demais. Plataformas *open-source* como Kubernetes é as mais utilizadas atualmente pois executa tal tarefa de forma eficiente. Tendo isso em vista, o objetivo deste trabalho envolve a implementação do containerd e Kubernetes em uma máquina de testes com o intuído de aprender a instalar, configurar e utilizá-la a fim de implementá-la em empresas na modalidade *on premise*. Todo esse projeto não acarretará custos ou disponibilidade de servidores para a execução e testes dessa plataforma, levando em consideração que em um ambiente de produção, há necessidade de utilização de servidores reais, os quais poderiam ser alocados para outras finalidades, muitas vezes geradoras de renda. Não obstante, haverá um comparativo com as máquinas virtuais, comumente utilizadas no cenário atual.

Palavras-chave: Kubernetes, containerd, container runtime, container.

Abstract: *Due to the need for developers and system administrators to quickly perform the provisioning of machines destined to run their applications, solutions such as containerization stood out from the rest. Open-source platforms such as Kubernetes are the most used nowadays as they perform this task efficiently. With this, the objective of this work involves the implementation of containerd and Kubernetes in a test machine with the aim of learning how to install, configure and use it in order to implement it in companies in the on premise mode. This entire project will not entail costs or availability of servers for the execution and testing of this platform, taking into account that in a production environment, there is a need to use real servers, which could be allocated for other purposes, often generating income . However, there will be a comparison with virtual machines, commonly used in the current scenario.*

Keywords: *Kubernetes, containerd, container runtime, container.*

1 Introdução

Apesar de parecer que a conteinerização é uma nova tendência, que está inovando o setor da Tecnologia da Informação, ela já é conhecida em sistemas Unix há alguns anos, ou melhor dizendo, décadas (VITALINO; CASTRO, 2018). O comando *chroot* foi um dos primeiros a possibilitar a realização de isolamento de sistemas, nesse caso o do *filesystem*, depois o de processos e assim por diante, até chegarmos ao cenário que temos hoje. O isolamento faz a divisão de certas características, basicamente de dois grupos, o primeiro é o

de recursos (*cgroups*), como CPU, memória, entre outros, o segundo é o de escopo (*namespace*) como processos, rede etc.

A conteinerização trata-se, de forma superficial, do processo de distribuir uma aplicação de maneira compartimentada, portátil e autossuficiente, possibilitando um maior desempenho no desenvolvimento de ambientes de produção e suas aplicações. Com este recurso, é possível instalar inúmeras aplicações, que podem ou não ter a mesma finalidade em *containers* diferentes que são isolados entre si.

Em seu trabalho Pahl *et al*. (2017), relata que a conteinerização é uma tecnologia que se baseia na capacidade de desenvolver, testar e implantar aplicações em um grande número de servidores de forma leve e relativamente isolada, a qual resultou no ganho de uma parcela significativa no mercado de gerenciamento de aplicações em nuvem.

Em 2008 foi iniciado o projeto LXC (*Linux Container*), que nada mais era que uma virtualização leve de alguns sistemas, porém, para os desenvolvedores, ela não oferecia uma boa experiência. Foi então que em 2013 o Docker, trouxe consigo a implementação com base em imagens, possibilitando o compartilhamento de aplicações ou conjunto de serviços, incluindo dependências e disponibilidade para diversos ambientes, fazendo com que a implantação de aplicações seja realizada de forma prática. Isso fez com que o Docker ganhasse rapidamente popularidade, oferecendo aos seus usuários acesso sem precedentes a aplicações, total controle sobre o versionamento e distribuição, além da habilidade de implantação com rapidez e eficiência.

Já em 2014 os engenheiros da Google criaram o Kubernetes, que surgiu da necessidade de atender um projeto que a empresa estava trabalhando. Apesar do Kubernetes ter sido desenvolvido originalmente pela Google, hoje ele é um projeto de software livre que é mantido pela CNCF (*Cloud Native Computing Foundation*), que é impulsionado através de contribuições de softwares livres da IBM e da Red Hat (CLOUD... 2019).

O Kubernetes busca oferecer automatização de implementações, eliminado processos manuais, que são utilizados para ajustar e escalar aplicações em *container*. Com o Kubernetes, é possível ter uma plataforma para gerenciamento de *clusters*, trazendo facilidade e eficiência, para a realização do balanceamento de carga e a portabilidade de carga de trabalho. Segundo a RedHat (2018) os "*clusters* podem incluir hosts em nuvem pública, nuvem privada ou nuvem híbrida", tomando o Kubernetes uma plataforma ideal para hospedar aplicações nativas em nuvem, pois é possível ter uma escalabilidade rápida.

Para o Kubernetes realizar o gerenciamento do *container* é preciso utilizar os *container runtime,* que são softwares capazes de executar *containers* em um sistema operacional hospedeiro. Segundo a Aqua Security Software (2021) em "uma arquitetura de *containers*, os *container runtime* são responsáveis por carregar as imagens do *container* de um repositório, monitorar os recursos do sistema local, isolar os recursos do sistema para uso de um *container* e gerenciar seu ciclo de vida".

Um exemplo de *container runtime*, é o containerd que, segundo Aqua Security Software (2021), é um *daemon* de código aberto compatível com Linux e Windows, que facilita o gerenciamento dos ciclos de vida do *container* por meio de solicitações de API (*Application Programming Interface),* realiza a transferência e armazenamento de imagens, execução e supervisão do *container*, armazenamento de baixo nível, anexos de rede, além de adicionar uma camada de abstração e aprimora a portabilidade do *container*.

Para entender melhor a história do containerd, foi em fevereiro de 2019, que ele saiu do estágio de incubação para o de graduação. Apesar de ter nascido em 2014 como um gerenciador de tempo de execução da camada inferior do Docker, só após a CNCF aceitar o

containerd, é que ele se tornou um *container* de execução padrão focado na simplicidade, robustez e portabilidade (CLOUD... 2019).

Devido a estes e muitos outros motivos, pode-se atestar uma tendência maior da utilização de *containers* no passar dos anos. Para tanto, é natural que grandes empresas, as quais já possuem estrutura e pessoal qualificado para trabalhar com tais plataformas, estejam planejando sua implementação. Portanto, se faz necessário um estudo dos requisitos para a implementação de um sistema de *containers*, quais configurações de máquinas são necessárias, quantos servidores deverão ser disponibilizados, como é feita a instalação e configuração das plataformas, e como elas se comportarão no ambiente. Para tanto, o presente trabalho foi desenvolvido em um ambiente de testes, onde foram criadas duas máquinas virtuais fazendo uso de um virtualizador em um computador local de alto desempenho. A partir deste ambiente, será criado um *cluster* com a plataforma Kubernetes e containerd, onde serão realizados testes a fim de elucidar dúvidas referentes aos requisitos necessários, configurações que devem ser feitas no ambiente, como cada plataforma funciona e compará-la com as de máquinas virtuais utilizadas rotineiramente.

2 Justificativa

A implementação destas plataformas em um ambiente simulado em uma máquina local não necessitaria dispender de um servidor, podendo este ficar disponível para uso da empresa ou possíveis clientes. Tal implementação em um ambiente simulado e controlado é mais rápida, por fazer uso de um SSD (*Solid State Drive*), não envolve configurações avançadas, por se tratar de um teste, a resolução de problemas – *troubleshootings* – é mínima e não há tempo limite para ser tratada, pois em ambientes reais poderia impactar serviços vitais para a companhia, e por fim, prepara a equipe para os percalços de uma implementação real.

Dessa forma será demonstrado em um ambiente de testes as vantagens que podem ser obtidas usando Kubernetes, containerd e *container* em relação aos ambientes comuns de máquinas virtuais.

Através dos resultados obtidos por meio da implementação, será possível identificar, comprovar as vantagens e desvantagens que a utilização de ferramentas, como Kubernetes e containerd, podem trazer para um ambiente de produção.

3 Objetivos

O intuito deste projeto é implementar em uma máquina de testes a solução de *containers* com o containerd, os quais serão orquestrados por meio do Kubernetes. Este projeto foi um teste inicial para avaliar como é feita instalação, criação, *deploy* e gerenciamento de *containers* em geral em um ambiente simulado, avaliando a viabilidade de sua implementação *on premise* e comparar esta tecnologia com as máquinas virtuais utilizadas comumente.

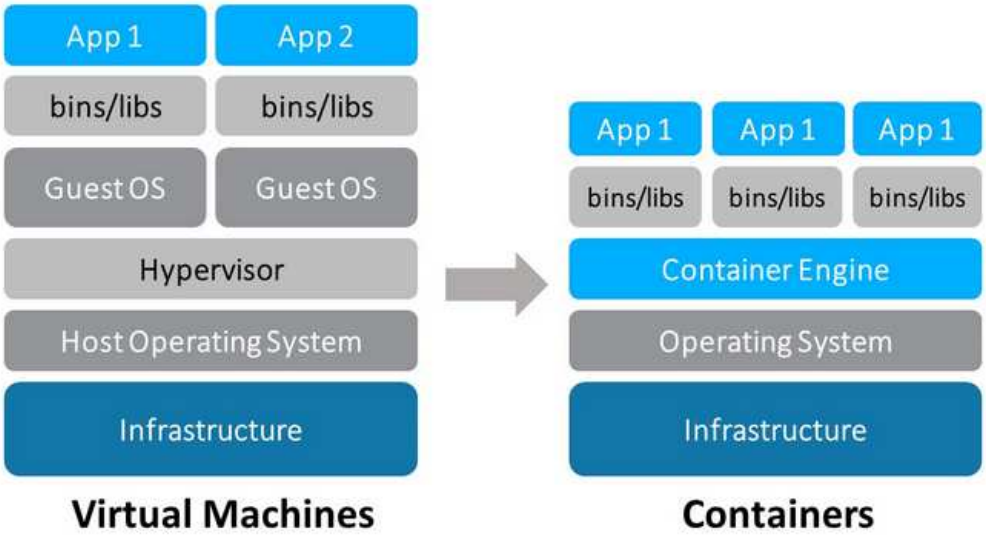
Com o desenvolvimento deste projeto, será possível adquirir conhecimento em sistemas que são utilizados para criar e manipular *containers*, e com isso, uma melhor qualificação para o mercado de trabalho, já que profissionais que saibam utilizar a tecnologia de conteinerização estão sendo requisitados.

4 Fundamentação Teórica

4.1 Container

A conteinerização é o processo de distribuição e implantação de aplicativos de uma forma portátil e previsível, onde há o empacotamento de componentes e suas dependências em um ambiente de processos padronizado, isolado e leve chamado *container*. Muitas empresas estão agora interessadas em projetar suas aplicações e serviços em sistemas distribuídos, permitindo assim escalá-lo facilmente e sobreviver a falhas de máquina ou de aplicação.

Figura 1- Estrutura máquinas virtuais vs *containers*



Fonte: (Kubernetes, 2021).

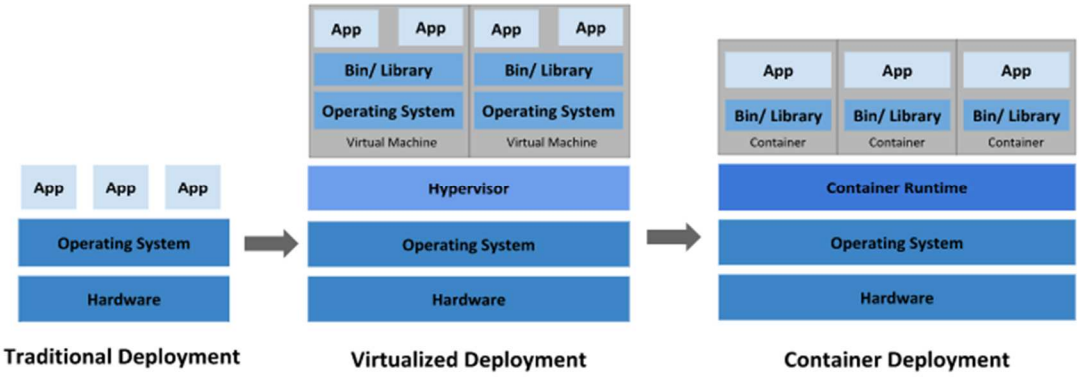
Na imagem acima, têm-se uma visão simplificada de como os *containers* se relacionam com o sistema hospedeiro (*host*) em comparação com as máquinas virtuais (VMs). Nos dois ambientes temos a infraestrutura e sistema operacional do *host* em comum, porém enquanto as VMs dependem de um virtualizador (*Hypervisor*) e de um sistema operacional completo para rodar suas aplicações, os *containers* utilizam recursos do sistema operacional de seu hospedeiro que foram abstraídos pelo *Container Engine* e isolam aplicações individuais.

4.2 Kubernetes

Kubernetes é uma plataforma de código aberto, portável e extensiva para o gerenciamento de cargas de trabalho e serviços distribuídos em *containers*, que facilita tanto a configuração declarativa quanto a automação. Ele possui um ecossistema grande, e de rápido crescimento. Serviços, suporte, e ferramentas para Kubernetes estão amplamente disponíveis tanto através de seu site oficial quanto em materiais da comunidade (KUBERNETES, 2021).

A figura 2 – Tipos de virtualização, mostra um comparativo entre um ambiente comum, a virtualização tradicional e a virtualização através de *containers* com o Kubernetes.

Figura 2 – Tipos de virtualização



Fonte: (Kubernetes, 2021).

Implementação tradicional: Antes de existir a possibilidade de executar os aplicativos em nuvem, não havia a possibilidade de definir limites para os aplicativos, dessa forma os servidores físicos, que possuíam mais de uma aplicação, apresentavam problemas de desempenho, pois algumas aplicações utilizavam mais recursos do que outras, com isso, acabavam onerando o desempenho de outra aplicação. A solução para esse problema era a utilização de um servidor para cada aplicação, gerando custo elevado para a empresa e inviabilizando manter muitos servidores físicos (KUBERNETES, 2021).

Implementação virtual: segundo Kubernetes (2021), a virtualização permite a execução de várias máquinas virtuais, em uma única CPU (Unidade de Central de Processamento) de um servidor físico, e com isso, possibilita o isolamento entre as máquinas virtuais, de forma que o nível de segurança da aplicação aumente. Através da virtualização, consegue-se realizar manutenções e melhorias de forma rápida e eficaz, tanto na aplicação como no hardware.

Implementação com *container*: apesar dos containers serem parecidos com as máquinas virtuais por terem seu próprio sistema de arquivos, compartilhamento de CPU, memória etc., o que os diferencia é que os *containers* compartilham o sistema operacional de seu *host*, tornando-

os leves. Como os containers são desacoplados da infraestrutura subjacente, eles são portáveis em nuvens e distribuições de sistema operacional (KUBERNETES, 2021).

O Kubernetes oferece uma estrutura para executar sistemas distribuídos de forma resiliente. Ele cuida do escalonamento e da recuperação à falha de sua aplicação, fornece padrões de implantação dentre outros recursos (KUBERNETES, 2021).

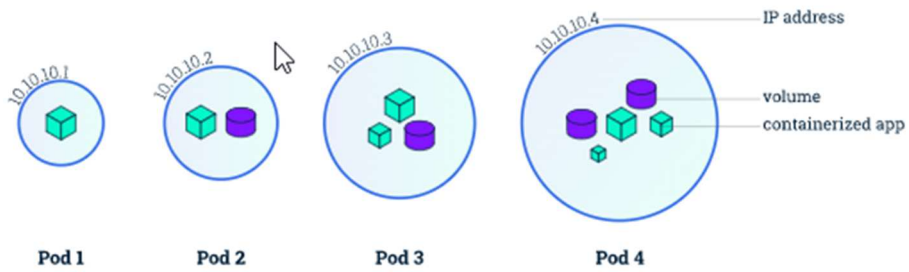
Através do Kubernetes é possível realizar o gerenciamento de forma fácil em uma implementação, pois ele oferece uma enorme gama de funcionalidades, tais como:

- Realizar descoberta de *IP (*Protocolo de Internet*)* utilizando o DNS (sistema de nomes de domínios) do container ou seu próprio IP;
- Balanceamento de cargas quando identifica que o tráfego em um *container* está alto, distribuindo o tráfego de rede;
- Variedade na montagem de armazenamento, tanto localmente quanto em nuvem;
- Manipulação dos *containers* e suas respectivas configurações e recursos;
- Realizar a reciclagem de *containers* que venham a falhar;
- Armazenar e gerenciar informações confidenciais, como senhas, tokens OAuth e chaves SSH.

Como o Kubernetes opera em nível de *container,* ele não consegue entregar todos os serviços PaaS (Plataforma como serviço), desta forma, ele fornece alguns recursos comuns ao PaaS, sendo eles: implementação, escalonamento, balanceamento de cargas, monitoração e alertas entre outros mais.

Cabe salientar que o Kubernetes não utiliza diretamente o container, mas sim os pods, que nada mais são do que uma abstração que representa um ou mais containers de aplicativos, e alguns recursos que são compartilhados com os containers. Pods são as menores unidades na plataforma Kubernetes. Segundo Kubernetes (2021) "Quando criamos um deployment no Kubernetes, esse deployment cria pods com containers dentro dele. Cada Pod está vinculado ao nó onde está programado (scheduled) e lá permanece até seu encerramento (de acordo com a política de reinicialização) ou exclusão. Em caso de falha do nó, pods idênticos são programados em outros nós disponíveis no cluster".

Figura 3 – Tipos de virtualização



Fonte: (Kubernetes, 2021).

Adicionalmente, o Kubernetes não é um mero sistema de orquestração. Na verdade, ele elimina a necessidade de orquestração. A definição técnica de orquestração é a execução de um fluxo de trabalho definido: primeiro faça A, depois B e depois C. Em contraste, o Kubernetes compreende um conjunto de processos de controle independentes e combináveis que conduzem continuamente o estado atual em direção ao estado desejado fornecido. Não importa como você vai de A para C. O controle centralizado também não é necessário. Isso resulta em um sistema que é mais fácil de usar e mais poderoso, robusto, resiliente e extensível (KUBERNETES, 2021).

4.3 Containerd

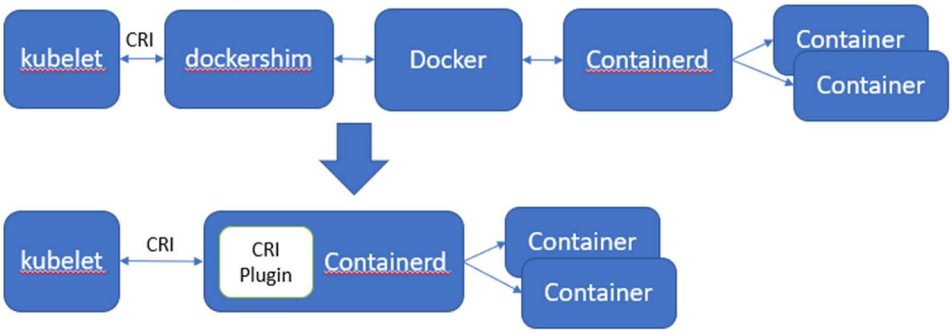
Com o containerd é possível realizar o gerenciamento do ciclo de vida de um container, desde que este seja compatível com as especificações da OCI (*Open Container Initiative*), responsável pela padronização da criação de imagens de aplicações, com isso qualquer *container runtime* pode utilizar o *container* (CONTAINERD, 2021).

Containerd é uma ferramenta que trabalha em segundo plano, executa e controla instâncias, gerenciando o ciclo de vida do *container,* assim como, a transferência e extração de imagens, à execução e supervisão do *container*, ao armazenamento de baixo nível, aos anexos de rede.

Através do containerd é possível abstrair grande parte da funcionalidade que o Docker acumulava como um todo, dessa forma ele executa um conjunto de funções mínimas que um *container runtime* precisa para executar qualquer *container*.

Diferente do Docker, a comunicação do containerd com o kubelet que é responsável pela comunicação entre o *Control Plane* e o nó, é curta, aumentando o desempenho e segurança da aplicação, pois o containerd segue o padrão CRI (*Container Runtime Interface*) do Kubernetes, reduzindo a carga de trabalho que é gerada, permitindo a interoperabilidade suave de diferentes tempos de execução do *container*.

Figura 4 – Comunicações com o Kubelet



Fonte: (Autores, 2021).

O containerd ajuda a abstrair as chamadas de kernel (syscalls) para que *containers* possam executar da mesma forma em qualquer sistema operacional, independente do que está acontecendo embaixo deles. Isso é chamado de supervisão e é um padrão muito comum quando estamos executando VMs (CONTAINERD, 2021).

Quando um sistema operacional (SO), é executado dentro de outro, o SO convidado não sabe que está sendo reproduzido dentro de uma VM, então ele continua chamando as syscalls necessárias para se comunicar com o hardware, é nesse momento que entra o trabalho do *Hypervisor*, que é abstrair essas chamadas de sistema para que o SO convidado não precise implementar cada kernel individual se ele estiver rodando dentro, por exemplo, de uma máquina Linux.

A seguir são apresentadas algumas vantagens obtidas com a utilização do containerd:

- Completo controle de imagens, podendo baixar e enviar imagens para registros;
- Totalmente configurável, permitindo gerenciar ciclos de vida de *containers* através de uma API;
- Realiza o gerenciamento de armazenamento e snapshots de forma rápida e flexível;
- Permite a utilização via API dentro da sua própria linguagem de programação;
- Abstrai completamente o SO que está em execução;
- Ótima compatibilidade com o Kubernetes e sua CRI (*Container Runtime Interface*).

5. Trabalhos Similares

Netto et al. (2017) explica o princípio da utilização do Kubernetes em um *data center* e as vantagens que podem ser proporcionadas através do controle dos *containers*. Um *cluster* Kubernetes é formado por máquinas, sendo elas virtuais ou físicas, as quais são denominadas como “nós” (*nodes*). O fato de os *containers* poderem ser replicados faz com que aumente a disponibilidade para hospedar as aplicações, dessa forma caso um *container* venha a falhar, o

Kubernetes o elimina e realiza sua substituição, garantindo a integridade da aplicação. Ao implementar um ambiente *cluster* Kubernetes, este orquestrará os *containers* utilizando o Docker, fazendo a gerência de todo este sistema, acarretando na alta disponibilidade das aplicações em execução, evitando que falhas possam deixar o sistema inoperante.

A palavra *container* foi citada várias vezes, no entanto em nenhum momento foi explicado o seu real valor. Vitalino e Castro (2016) expõem de forma simples o conceito do que é um *container*, que nada mais é do que um agrupamento de aplicações junto a suas dependências que utilizam o *kernel* do sistema em que está hospedado. Com isso o *container* pode prover microsserviços, sem a necessidade de que todo um sistema operacional seja instalado, trazendo agilidade quando for preciso subir um novo serviço (PINTO e PEREIRA, 2019).

6 Metodologia e Desenvolvimento

6.1 Montagem da Infraestrutura

Para este projeto, foi utilizado um computador de mesa com as seguintes configurações:

- Processador AMD Ryzen 5 3600, *cache* 32 MB, frequência de 3.6 GHZ, 6 núcleos e 12 *threads*, com *coolerbox* padrão;
- Placa mãe Asus TUF B450M-Pro Gaming (Socket AM4);
- duas memórias em *dual channel* DDR4 CL16 Geil EVO X II 8 GB (totalizando 16GB) com frequência de 3200 MHz em *stock*;
- Fonte Gamemax 550W;
- SSD Crucial MX500 com 500 GB de armazenamento;
- Placa de vídeo MSI GTX 1060 OCV1, memória 6 GB GDDR5 em 8008 MHz e com frequência de 1544 MHz em seu GPU*;*
- Sistema operacional Windows 10.

Para rodar máquinas virtuais neste computador, foi necessário habilitar a função de virtualização na BIOS. Procedimento que não será necessário se feito em servidores reais.

Nesta máquina foi instalado o virtualizador Oracle VM Virtualbox 6.1, onde criou-se duas máquinas virtuais, com sistema operacional Ubuntu Server 20.04 LTS. As máquinas atendem as seguintes configurações:

- 2 processadores;
- 4 GB de memória RAM;
- 20 GB de disco rígido;
- Memória de vídeo de 64 MB;
- 1 adaptador de rede Intel Pro/1000 MT Desktop configurado em uma rede NAT.

Figura 5 – Configuração Máquina Virtual Kubernetes - master

Fonte: (Autores, 2021).

Referente a configuração do sistema operacional Ubuntu, foi utilizada a linguagem como português do Brasil, quanto ao particionamento, criou-se partições automáticas no formato LVM (*Logical Volume Manager*), ficando a partição raiz com um total de 20 GB aproximadamente. As placas de rede foram configuradas para possuir apenas endereços IPV4 estáticos.

A partir do guia encontrado no site oficial do Kubernetes (2021), criou-se um *cluster* com um nó *master* e outro *node* fazendo uso das máquinas virtuais citadas anteriormente. O *master* foi designado como *control-plane*, que tem a função de orquestrar todo o ambiente, atuar na tomada de decisões gerais sobre o *cluster* (*kube-scheduler*), receber e executar os eventos do *cluster* (*kube-api-server*), balancear a carga de trabalho entre os *nodes* (*workers*), realizar o armazenamento persistente de dados chave-valor (banco *etcd*), exercer o gerenciamento dos demais controladores (*kube-controller-manager*), dentre outras tarefas. O *node*, que têm o papel de executar toda carga de trabalho (*work load*) advinda de aplicações conteinerizadas que rodam em cima destes.

A fim de comparar a tecnologia de *containers* com a de máquinas virtuais, foi criado outro servidor com configurações de hardware semelhantes, tendo como base os requisitos mínimos para rodar o sistema operacional Ubuntu server e a aplicação nginx que, segundo a documentação oficial destas, seria 1 GB de memória RAM, 1 processador e 20 GB de disco rígido. A placa de rede, no entanto, foi configurada em modo “bridge”. Tal máquina foi identificada com o nome “vm” e será utilizada no desenvolvimento dos testes que se seguirão.

Os passos que discutem a configuração do ambiente de *containers* foram baseados na documentação oficial do site Kubernetes, a instalação do Kubernetes e configuração do servidor “vm” foram anexados ao final deste projeto.

6.2 Metodologia

Os métodos aqui utilizados para comparar as tecnologias de conteinerização e máquinas virtuais são simples, com ambos os ambientes prontos (máquinas virtuais criadas, formatadas e *cluster* Kubernetes configurado), foi criado um *pod* nginx no *cluster* Kubernetes e instalada a mesma aplicação diretamente na máquina virtual "vm". Foi contabilizado então o tempo que se leva para a criação de um *pod* nginx com o tempo que é necessário para criar uma máquina virtual limpa e instalar o nginx. Não obstante, também foi analisado os recursos de memória ram e disco que cada uma dessas tecnologias utilizou.

Também foi verificado a eficiência de escalabilidade horizontal e vertical destas tecnologias, ou seja, a facilidade e praticidade de se aumentar os recursos de hardware e quantidade de *pods* e máquinas virtuais quando necessário.

Cabe salientar que foi abstraído toda a infraestrutura por trás dos *containers* e máquinas virtuais na aferição dos recursos e tempos de ambos os ambientes, pois o intuito é apenas comparar o *container* e máquina virtual em si.

6.3 Desenvolvimento

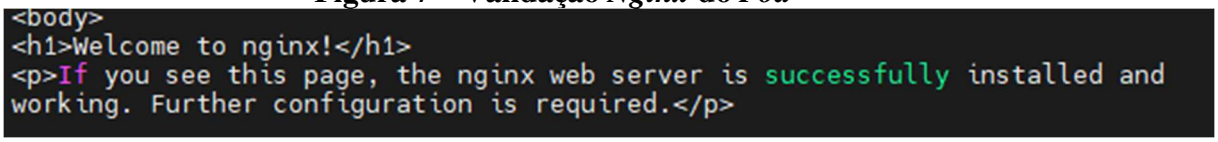
Os procedimentos citados na metodologia foram primeiramente realizados na máquina virtual "vm" e, com as configurações de hardware previamente feitas, iniciou-se o cronômetro após ligar a máquina e aparecer a primeira tela de formatação do sistema operacional Linux. Após uma formatação simples, atualização de repositórios e pacotes com o comando "apt update ; apt upgrade -y", ocorreu a instalação da aplicação nginx com o comando "apt install nginx -y", verificou-se com o comando "curl -v localhost:80" se o aplicativo estava funcionando adequadamente e, só então parou-se o cronômetro. Abaixo é possível visualizar parte da saída do comando "curl":

Figura 6 – Validação *Nginx* da máquina virtual

Fonte: (Autores, 2021).

Já para o ambiente de *containers* o desenvolvimento muda um pouco. Com o *cluster* Kubernetes ligado, cronometrou-se o tempo a partir do comando executado para criar o *pod* nginx, "kubectl create deployment nginx --image=nginx", e se encerrou a contagem quando o pod ficou com o status 1/1, verificado através do comando "kubectl get pods", o qual sinalizou que a aplicação estava operacional. Esta etapa teve que se repetir mais uma vez, pois a primeira execução a imagem da aplicação é baixada, ou seja, leva um tempo maior para a montagem do *container*, já na segunda vez, o processo ocorre mais rápido pois a imagem utilizada anteriormente fica armazenada na memória *cache* do Kubernetes, reduzindo o tempo de criação de *containers* consideravelmente. Para visualizar o nginx em funcionamento, foi acessado o *pod* através do comando "kubectl exec -it nginx-6799fc88d8-4gzvv -- /bin/bash", sendo nginx-"6799fc88d8-4gzvv" o nome do *pod*, e então novamente utilizamos o comando "curl -v localhost:80", o resultado é apresentado abaixo.

Figura 7 – Validação *Nginx* do *Pod*

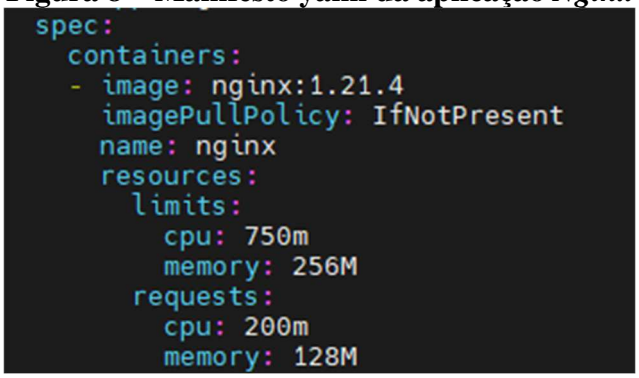


Fonte: (Autores, 2021).

Quanto ao quesito de escalabilidade vertical, ou seja, provisionamento de memória ram, processadores virtuais (vcpus), disco etc., ocorrem da mesma maneira entre o *cluster* Kubernetes e a máquina virtual "vm" por ambos se tratar de máquinas virtuais. O valor de memória ram, número de vcpus e placas de rede (este último requer configurações adicionais simples) foram alterados facilmente através do software Virtualbox, refletindo no servidor virtual após reinicialização. Porém modificações no disco envolvem configurações dentro das máquinas virtuais Linux, sendo necessário utilizar o pacote "fdisk" para configurar uma nova partição contendo o novo volume de dados adicionados através do Virtualbox. Para reverter as configurações também é simples, apenas se faz necessário alterar as mesmas configurações no Virtualbox e remover o novo volume utilizando o fdisk no Linux.

Em relação aos pods, por se tratar de uma estrutura voltada para microsserviços, a escalabilidade vertical é limitada. Porém, através de alterações simples no manifesto da aplicação nginx, consegue-se facilmente o provisionamento de memória ram e vcpu. Através do comando "kubectl edit deployment nginx" temos acesso e permissão de edição do manifesto da aplicação e, com isso, é possível alterar os limites de memória e cpu usados pelo *pod*.

Figura 8 – Manifesto yaml da aplicação *Nginx*



Fonte: (Autores, 2021).

Na imagem acima temos o manifesto da aplicação nginx, mais especificamente o *spec* (especificações) do *container*. Editando o atributo "*resources*" é possível alterar os valores requisitados pela aplicação ao *node* em "*requests"*, que seriam a memória e cpu que a aplicação utiliza por padrão, além de modificar os valores máximos que podem ser utilizados em "*limits",* sendo a sigla "M" definida para megabytes e "m" para "milis", a qual representa uma fração do todo. Tais alterações servem tanto para aumentar quanto para diminuir os recursos do *pod.*

Já a questão provisionamento de armazenamento em disco do *container* ocorre de maneira diferente. Um volume "emptyDir" é criado quando um *pod* é atribuído a um *node,* e existe enquanto o *pod* estiver em execução. Este volume não tem dimensionamento estipulado, e é utilizado pelo *container* para ler e gravar arquivos de forma temporária. Existem também outras inúmeras maneiras de configurar volumes para os *pods*, sendo eles persistentes, efêmeros, storages anexos etc., seja alterando o manifesto do deployment, ou utilizando recursos do próprio Kubernetes. Como o "emptyDir" é o volume padrão e não necessita de

configuração para ser provisionado mais espaço, as configurações para alterar volumes não serão abordadas, porém cabe lembrar que existe sim a possibilidade e sua configuração não é complexa.

Quanto a escalabilidade horizontal, para ambientes de máquinas virtuais é necessário criar uma nova máquina virtual, formatá-la, atualizá-la, configurar sua rede, instalar e configurar a aplicação, e inseri-la no balanceador de carga para que ela receba novas cargas de trabalho, ou seja, repetir todos os passos sobre a configuração da máquina virtual "vm" na parte de anexo deste projeto, além configurações adicionais no balanceador de carga. Já para o ambiente de conteinerização é muito mais simples, apenas temos que alterar o número de réplicas no manifesto de seu *deployment,* ou de maneira mais fácil ainda, basta executar o comando "kubectl scale deployment/nginx --replicas=X", onde "X" é o número desejado de réplicas para este *pod*. A parte de configuração do balanceador de carga não se faz necessário, pois existe um serviço atrelado ao *deployment* que já realiza o balanceamento de carga para todos os *pods* do *deployment* de forma automática.

Para melhor visualização dos dados adquiridos, foram adicionados imagens e gráficos que serão discutidos nos resultados.

7 Resultados e Discussões

A criação do *cluster* Kubernetes não foi algo complexo, pois os materiais fornecidos pela própria empresa são bem completos e explicativos. Porém em alguns momentos se fez necessário buscar fontes da comunidade para sanar algumas dúvidas que vieram durante o processo. O principal entrave foi encontrar informações sobre o *container runtime* containerd, onde até mesmo em seu site oficial, conceitos e dados mais aprofundados são vagos, até mesmo na comunidade é algo que ainda está se desenvolvendo, mesmo que este não seja nenhuma figura nova no mercado. No entanto, as informações encontradas nos sites oficiais a em comunidades sobre o assunto foram suficientes para atingir o objetivo de criar um *cluster* Kubernetes funcional, mesmo que ainda tenham ficado dúvidas sobre alguns procedimentos.

Para criar um cluster Kubernetes com alta disponibilidade e redundância são necessários ao menos dois servidores, dois links de internet, um sistema de backup e replicação, ambiente climatizado para evitar superaquecimento, um gerador de energia ou nobreak para manter o equipamento ligado caso tenha interrupção no fornecimento de energia, e infraestrutura redundante (switchs, cabeamento, servidor reserva etc.). A configuração mínima para os servidores é encontrada no próprio site do Kubernetes, vide imagem abaixo, porém a recomendação seria algo apenas voltado para testes, em um ambiente real de produção, quanto mais recursos, melhor.

Figura 9 – Configuração mínima *cluster* Kubernetes

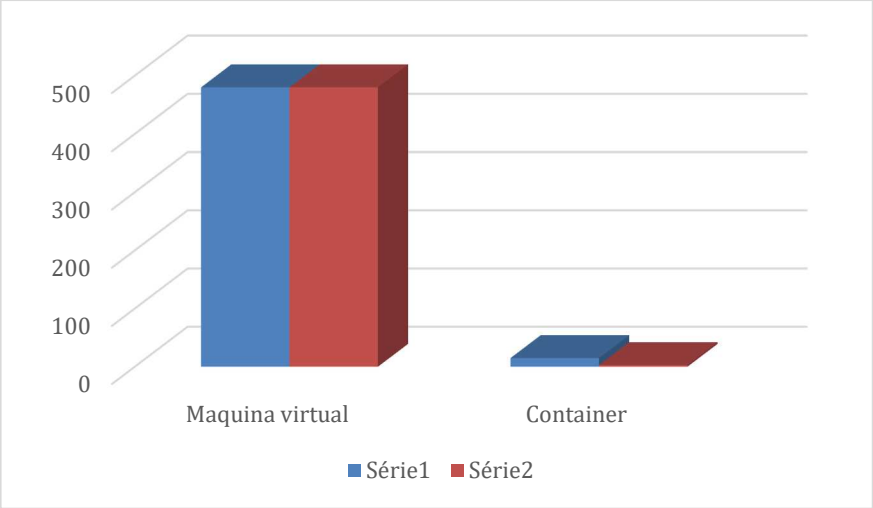
- Um host Linux compatível. O projeto Kubernetes fornece instruções genéricas para distribuições Linux baseadas em Debian e Red Hat, e aquelas distribuições sem um gerenciador de pacotes.
- 2 GB ou mais de RAM por máquina (qualquer menos deixará pouco espaço para seus aplicativos).
- 2 CPUs ou mais.
- Conectividade de rede total entre todas as máquinas no cluster (rede pública ou privada está bem).
- Nome de host exclusivo, endereço MAC e product_uuid para cada nó. Veja aqui para mais detalhes.
- Algumas portas estão abertas em suas máquinas. Veja aqui para mais detalhes.
- Troca desativada. Você **DEVE** desabilitar a troca para que o kubelet funcione corretamente.

Fonte: (KUBERNETES, 2021).

Recomenda-se também instalar um *Hypervisor* nos servidores *bare metal*, criando-se assim o *cluster* dentro deste, proporcionando assim uma maior gama de recursos como snapshots, possibilidade de criação de outros *clusters* para diferentes finalidades, facilidade em manutenções entre outros.

Referente aos dados comparativos de tempo para subir a aplicação nginx em cada um dos ambientes, *containers* e máquinas virtuais, recursos utilizados como memória ram, vcpu e disco, serão apresentados abaixo em gráficos.

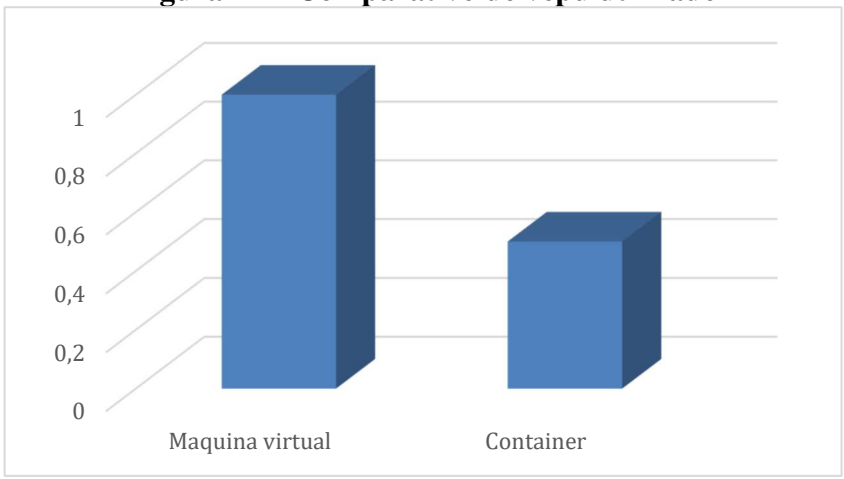
Figura 10 – Tempo em segundos para subir a aplicação Nginx



Fonte: (Autores, 2021).

Mediante ao teste realizado, verificou-se que o tempo médio para construir uma máquina virtual com a aplicação nginx levou em torno de oito minutos, enquanto a criação de um *pod* contendo a mesma aplicação, foi de apenas quinze segundos na primeira série, pois o *cluster* não tinha a imagem do nginx armazenada. Já na segunda série, o tempo da máquina virtual não mudou, e já com o download da imagem, o *container* levou apenas três segundos para ser criado.

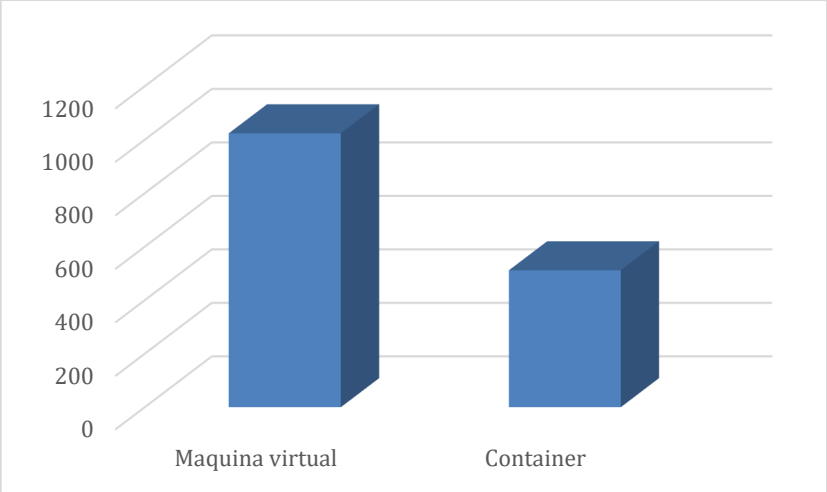
Figura 11 – Comparativo de vcpu utilizado



Fonte: (Autores, 2021).

No quesito de utilização vcpu, foi necessário alocar um vcpu para a máquina virtual e apenas meio vcpu para o pod. Tal configuração foi baseada tendo como base a documentação oficial da própria aplicação nginx e Ubuntu server (tamanho mínimo do servidor, 1 núcleo de CPU, 1 GB de RAM, 20 GB de armazenamento), salientando que esses dados têm como base uma máquina virtual. Já o *pod* utilizou apenas metade de um vcpu, sendo este totalmente dedicado a aplicação, diferente da máquina virtual que dividirá o vcpu com o sistema operacional Linux.

Figura 12 – Comparativo de memória ram utilizada em MB



Fonte: (Autores, 2021).

Seguindo novamente a documentação do nginx e Ubuntu server, o mínimo de memória ram para implementar a aplicação no ambiente Linux em máquinas virtuais foi de 1024Mb, novamente sendo dividido tanto pelo sistema quanto pelo nginx. O *container* utilizou apenas 512Mb, sendo totalmente dedicado a ele.

Quanto ao disco, a máquina virtual consumiu 20GB de armazenamento, sendo destinados tanto à aplicação quanto ao sistema operacional e, como não foi encontrado dados precisos sobre o tamanho do *pod* nginx, não foi realizada uma análise comparativa. Porém, estima-se que o *pod* consuma menos que 1GB.

Na questão de escalabilidade vertical, como o *cluster* e a "vm" são iguais, pois ambas são máquinas virtuais. O Virtualbox, assim como outros *Hypervisors*, facilitam muito nesse quesito, onde através de poucos cliques e alterações, as modificações já refletem nos servidores, com exceção de discos e volumes, que envolvem procedimentos adicionais mais complexos. Em relação aos *containers*, provisionamento de recursos é simples e rápido, apenas alterando o manifesto "yaml" de sua aplicação, as mudanças já refletem em um novo *pod* que é gerado automaticamente após sua alteração. Porém, como já foi dito, *containers* são destinados a microsserviços, ou seja, a ideia central é terem recursos o suficiente para rodar uma aplicação dedicada de forma performática, sendo estes leves, rápidos e tendo um ciclo de vida curto. Havendo a necessidade de ampliar a cobertura de tal serviço, são provisionados mais *pods* com a escalabilidade horizontal.

Por fim, no quesito de escalabilidade horizontal, as máquinas virtuais possuem grandes desvantagens. O tempo necessário para a criação de uma nova máquina virtual é muito elevado se comparado a tecnologia de *containers*, pois envolve diversos passos a serem realizados, como a configuração da nova máquina no virtualizador, formatação do sistema operacional e configuração da aplicação, o que pode ser feito utilizando apenas um comando no Kubernetes a fim de provisionar diversos novos *pods* em poucos segundos.

8 Conclusão

Concluiu-se que a criação e instalação do *cluster* Kubernetes com containerd em máquinas com o Sistema Operacional Ubuntu se faz de forma simples e sem muitos entraves, porém, a configuração do Kubernetes é um tanto quanto trabalhosa, pois envolve diversos conceitos de rede, DNS, virtualização, *firewall* e outros conhecimentos complexos, no entanto, a documentação oficial e conteúdo da comunidade é vasto e bem explicativo tornando estas tarefas mais fáceis.

O provisionamento de *container* é rápido, consome menos recursos computacionais de hardware que máquinas virtuais, consequentemente gerando economia de tempo e recursos financeiros, pois componentes de hardware são custosos.

A escalabilidade horizontal dos *pods* somada com a escalabilidade vertical dos nós e *clusters* através de um virtualizador faz com que o ambiente seja robusto e performático, trazendo praticidade para os gestores de infraestrutura, e agilidade na implementação e versionamento de novas aplicações para os desenvolvedores. Em contrapartida, é necessário um vasto conhecimento técnico em diversas áreas de infraestrutura e programação para gerir e configurar todo esse aparato.

Para concluir, a tecnologia de conteinerização é algo que empresas de médio e grande porte devem se atentar, sendo viável sua implementação, proporcionando economia de recursos, agilidade e praticidade em relação as máquinas virtuais, comumente utilizadas. Através deste projeto, muito conhecimento e conceitos foram adquiridos, o que pode e será utilizado num futuro próximo.

9 Referências

ANTON, M. Uma breve história do Kubernetes, do OpenShift e da IBM: Entenda o histórico e o futuro de uma plataforma empresarial de aplicativos Kubernetes. Site oficial IBM. 01/05/2019. Disponível em: https://developer.ibm.com/br/blogs/a-brief-history-of-red-hat-openshift/. Acesso em: 15 nov. 2021.

AQUA SECURITY SOFTWARE. **Três tipos de ambiente de execução de contêiner e a conexão Kubernetes**: o que é um container runtime?. O que é um Container Runtime?. 2021. Disponível em: https://www.aquasec.com/cloud-native-academy/container-security/container-runtime/. Acesso em: 30 out. 21.

CONTAINERD. ContainerD: Um dos substitutos do Docker. ContainerD: Um dos substitutos do Docker. 14/07/2021. Disponível em: https://www.coupled.com.br/site/artigo/containerd. Acesso em: 20 nov. 2021.

CLOUD Native Computing Foundation anuncia graduação da containerd. 2019. Disponível em: https://www.cncf.io/announcements/2019/02/28/cncf-announces-containerd-graduation/. Acesso em: 10 nov. 2021.

CROSBY, M. **O que é containerd?** Docker Blog. 07/08/2021. Disponível em: https://www.docker.com/blog/what-is-containerd-runtime/. Acesso em: 22 nov. 2021.

KUBERNETES. **Documentação do Kubernetes**. 12/11/2020. Disponível em: https://kubernetes.io/docs/home/. Acesso em: 29 nov. 2020.

KUBERNETES. **Documentação do Kubernetes**: visualizando pods e nós (nodes). Visualizando Pods e Nós (Nodes). 2021. Disponível em: https://kubernetes.io/pt-br/docs/_print/#pg-2771f4e8c45321b17cb0114a2d266453. Acesso em: 10 nov. 21.

NETTO, Hylson V.; LUNG, Lau Cheuk; CORREIA, Miguel; LUIZ, Aldelir Fernando; SOUZA, Luciana Moreira Sá de. **State machine replication in containers managed by Kubernetes**. 27/12/2017. Disponível em: https://www.gsd.inesc-id.pt/~mpc/pubs/smr-kubernetes.pdf. Acesso em: 31 out. 2020.

NGINX. **Documentação oficial do Nginx**. 25/10/2021. Disponível em: https://docs.nginx.com/nginx-instance-manager/reference/specs/. Acesso em: 25 out. 2021.

PAH, Claus; BROGI, Antonio. **Cloud Container Technologies: A State-of-the-Art Review**. 09/05/2017. Disponível em: https://www.researchgate.net/publication/316903410_Cloud_Container_Technologies_A_State-of-the-Art_Review. Acesso em: 02 out. 2020.

PINTO, Walker Douglas Garcia; PEREIRA, Fagner Coin. **DOCKER –CONTAINERS NÃO SÃO VM's**. 11/2019. Disponível em:

http://raam.alcidesmaya.com.br/index.php/SGTE/article/view/21/24. Acesso em: 01 out. 2020.

PROJECT CALICO. **Documentação oficial do Project Calico**. 07/09/2021. Disponível em: https://projectcalico.docs.tigera.io/getting-started/kubernetes/quickstart. Acesso em: 07 out. 2021.

REDHAT. **O que é Docker?** Site oficial RedHat. 09/01/2018. Disponível em: https://www.redhat.com/pt-br/topics/containers/what-is-docker. Acesso em: 22 fev. 2021.

REDHAT. **O que é Kubernetes?**: visão geral. Visão geral. 2018. Disponível em: https://www.redhat.com/pt-br/topics/containers/what-is-kubernetes. Acesso em: 10 out. 2021.

UBUNTU. **Documentação oficial do Ubuntu**. 02/10/2021. Disponível em: https://ubuntu.com/server/docs/installation. Acesso em: 02 out. 2021.

VITALINO, Jeferson Fernandes Noronha; CASTRO, Marcus André Nunes. **Descomplicando o Docker**. 2. ed. Rio de Janeiro: Brasport, 2018. 130 p.

WEAVEWORKS. **CONTÊINERES DE REDE EM QUALQUER AMBIENTE**. Disponível em: https://www.weave.works/oss/net/. Acesso em: 23 abr. 2021.

10 ANEXO

1. Configuração da máquina virtual "VM"

Já com o Ubuntu 20.04 LTS instalado, ligou-se a máquina e executou-se os comandos "*apt-get update"* e posteriormente "*apt-get upgrade"* para atualizar a lista de repositórios do Linux seus pacotes. Após a atualização, foram instaladas algumas ferramentas, como o *vim, telnet, openssh-server e net-tools* e configurado um IP estático no servidor, segue abaixo o endereço:

vm - 192.168.15.12

Para evitar complicações desnecessárias e, se tratando de um ambiente de testes, desabilitou-se o firewall UFW, padrão do Ubuntu, através dos comandos "systemctl stop ufw" e "systemctl disable ufw". Com isso, a máquina já está pronta para utilização.

2. Preparativos para a instalação do Kubernetes e configuração do cluster

Os procedimentos abaixo contemplam ambos os servidores Ubuntu e foram executados como usuário administrativo. Apenas na instalação do Kubernetes que haverá diferenças nas execuções entre uma máquina e outra.

Com as máquinas já ligadas, foram realizados os mesmos procedimentos para configuração da máquina virtual "vm", atualização de repositórios e pacotes, instalação das mesmas ferramentas e configuração do IP estático dos servidores, ficando da seguinte maneira:

Master - 192.168.15.10
Node - 192.168.15.11

Seguindo a documentação oficial do Kubernetes (2021), foi desabilitada a memória *swap*, caso contrário seriam perdidas as propriedades de isolamento, previsibilidade em relação ao desempenho, latência ou E/S. Pra isto, executou-se o comando "swapoff -a" e então utilizando o editor de texto *vim*, comentou-se a linha referente ao swap no arquivo */etc/fstab* , para finalizar o procedimento, foi necessário reiniciar os servidores.

Dando prosseguimento, foi desabilitado o também foi desabilitado o firewall UFW, pois utilizou-se apenas o iptables como firewall deste ambiente. Com o ufw desabilitado, configurou-se o *iptables* seguindo orientações sobre quais portas deveriam ser abertas em cada máquina, a fim de que todos os componentes funcionassem e se comunicassem adequadamente. Com o intuito de manter as modificações do *iptables* funcionando após a reinicialização das máquinas, criou-se um *script* em cada uma das VMs, segue abaixo como ficou cada um deles:

Master

Nome do script: iptables.sh

```
#!/bin/bash
#
### BEGIN INIT INFO
# Provides:          iptables.sh
# Required-Start:    $all
# Required-Stop:     $all

# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: libera portas iptables
# Description:       libera portas para o kubernetes no iptables
### END INIT INFO

sudo /sbin/iptables -A INPUT -p tcp --dport 6443 -j ACCEPT
sudo /sbin/iptables -A INPUT -p tcp --dport 10250:10252 -j ACCEPT
sudo /sbin/iptables -A INPUT -p tcp --dport 2379:2380 -j ACCEPT
```

Node

Nome do script: iptables.sh

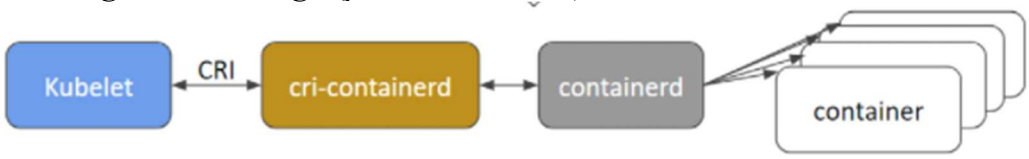
```
#!/bin/bash
#
### BEGIN INIT INFO
# Provides:          iptables.sh
# Required-Start:    $all
# Required-Stop:     $all
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: libera portas iptables
# Description:       libera portas para o kubernetes no iptables
### END INIT INFO

sudo /sbin/iptables -A INPUT -p tcp --dport 10250 -j ACCEPT
sudo /sbin/iptables -A INPUT -p tcp --dport 30000:32767 -j ACCEPT
```

Então moveu-se os scripts para o diretório */etc/init.d/* , garantiu-se a permissão de execução à eles com o comando "*chmod +x iptables.sh"* e foi utilizado o "*update-rc.d iptables.sh defaults*" para que o sistema os executa-se em sua inicialização.

Agora se faz necessário a instalação de um *daemon* de *container*, que nada mais é do que um conjunto de funções mínimas que um *container runtime* precisa para executar qualquer *container*, ou seja, construir, armazenar, compartilhar e rodar *containers*. Para esta finalidade foi escolhido o containerd como daemon.

A comunicação entre os *containers* e Kubernetes através do containerd acontece de acordo com a figura abaixo (figura X):

Figura 13 – Integração entre kubelet, cri-containerd e container



Fonte: (Kubernetes, 2021).

O Kubelet se comunica por soquetes Unix com o CRI (Container Runtime Interface), o qual consiste em buffers de protocolo e API gRPC e bibliotecas, com especificações e ferramentas adicionais em desenvolvimento ativo, onde kubelet atua como um cliente e o CRI como o servidor (Kubernetes, 2021). Por fim, o CRI se comunica com o containerd que executa os *containers.*

Para realizar a instalação do containerd, executou-se o comando "*apt install containerd*".

Visando a adequação do containerd ao Kubernetes CRI, algumas configurações devem ser implementadas, segundo a documentação oficial do Kubernetes (2021):

Carregando o módulo br_netfilter nas configurações do containerd:
cat <<EOF | sudo tee /etc/modules-load.d/containerd.conf
overlay
br_netfilter
EOF

Carregando os módulos overlay e br_netfilter no Linux:
sudo modprobe
sudo modprobe br_netfilter

Configurações dos parâmetros sysctl necessários, persistindo durante as reinicializações.
cat <<EOF | sudo tee /etc/sysctl.d/99-kubernetes-cri.conf
net.bridge.bridge-nf-call-iptables = 1
net.ipv4.ip_forward = 1
net.bridge.bridge-nf-call-ip6tables = 1
EOF

Aplicando as modificações do sistema
sudo sysctl --system

Configurando o containerd:
sudo mkdir -p /etc/containerd
containerd config default | sudo tee /etc/containerd/config.toml

#Usando o systemddriver cgroup
vim /etc/containerd/config.toml
Alterar o seguinte campo para "true"
[plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc]
...
[plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc.options]
SystemdCgroup = true

Aplicando as modificações
sudo systemctl restart containerd

3 Instalação Kubernetes e criação do *cluster*

Definidos o *master, node,* e realizadas as devidas instalações e configurações básicas, deu-se início a instalação do Kubernetes, o qual deve ser realizada tanto no *master* quanto no *node*. Porém, antes de dar início a sua instalação, segundo a documentação oficial, é necessário carregar o módulo *br_netfilter*, através do comando "*sudo modprobe br_netfilter",* a fim de permitir que o *iptables* (*firewall*) veja o tráfego interligado no *cluster*.

Em seguida, para que as tabelas de IP dos *nodes* vejam corretamente o tráfego na placa de rede, deve-se novamente garantir que a variável *net.bridge.bridge-nf-call-iptables* está definida como "1" na configuração do arquivo "*sysctl*".

Para tanto foram executados os seguintes comandos:

cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf
br_netfilter
EOF

cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
EOF
sudo sysctl –system

Após todos esses passos, baixou-se os pacotes necessários para utilizar o repositório do Kubernetes. Segue o comando abaixo:
sudo apt-get install -y apt-transport-https ca-certificates curl

Agora deve-se baixar a chave de assinatura pública do Google Cloud e adicioná-la ao repositório do *apt* (gerenciador de repositórios do Ubuntu):

sudo curl -fsSLo /usr/share/keyrings/kubernetes-archive-keyring.gpg https://packages.cloud.google.com/apt/doc/apt-key.gpg

echo "deb [signed-by=/usr/share/keyrings/kubernetes-archive-keyring.gpg] https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee /etc/apt/sources.list.d/kubernetes.list

Para concluir a pré-configuração do Kubernetes, foi atualizado o índice do *apt* e instalados os três pacotes necessários para executar o Kubernetes: *kubelet, kubeadm* e *kubectl*. O último comando que deve ser executado tem o objetivo de marcar estes três pacotes como retidos, o que vai prevenir que os pacotes sejam automaticamente atualizados ou removidos.

sudo apt-get update
sudo apt-get install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl

Para instalar o Kubernetes, executou-se seguinte comando no nó *master "kubeadm init --pod-network-cidr=192.168.0.0/16"* que não apenas instala, mas já define a rede que os *pods* pertencerão. Tal comando gera uma saída contendo alguns passos que devem ser seguidos para o correto funcionamento do k*ubectl*:

#Comando destinado à criação do diretório "*kube"* dentro do *Home* do usuário vigente
mkdir -p $HOME/.kube

Comando utilizado para copiar o arquivo *admin.conf* para o diretório kube recém-criado.
sudo cp -i /etc/kubernetes/adm.conf $HOME/.Kube/config

Comando com a finalidade de mudar o proprietário dos diretórios e seus conteúdos para o usuário e grupo de usuário vigente.
sudo chown $(id -u):$(id -g) $HOME/.kube/config

Opcionalmente, também foi acessado o usuário *root* onde executou-se o seguinte comando para criar a variável de ambiente informada na saída do comando *kubeadm init:*
export KUBECONFIG=/etc/kubernetes/admin.conf

Também foi necessário implantar uma rede *pod* para o *cluster,* o que nada mais é do que um kit de ferramentas de rede nativo em nuvem, o qual cria uma rede virtual que conecta *containers* em vários *hosts* e permite sua descoberta automática (WEAVE WORKS, 2021). O Kubernetes oferece algumas opções de redes *pod* por meio do *link* gerado na saída do comando *kubeadm init,* dentre elas optou-se por uma chamada Calico, devido alguns dos materiais que foram utilizados como referência também terem escolhido esta componente.

Foi iniciado o *kubeadm* no *master* com o comando "*kubeadm init"*, que ao final de sua execução, nos retornou um *token* necessário para adicionar os *nodes* ao *cluster*. Então, inserindo este *token* nos dois *workers*, houve a comunicação entre todas as máquinas virtuais dentro do *cluster* através do *api-server*. Pôde-se visualizar que o *cluster* foi criado a partir do retorno do comando "*kubectl get nodes*", executado no *master*.

A última saída do comando *kubeadm init* fornece um *token* para ser inserido nos *nodes* para que eles se comuniquem com o *control-plane (master)* e juntos criem um *cluster* Kubernetes. Segue exemplo do *token* gerado:

kubeadm join 192.168.0.114:6443 --token 7mtscv.bw2r267a8pmo9fmm --discovery-token-ca-cert-hash
sha256:73f665370f27b0f4d9695754d0d0eebf9d8e6ca6009cf035f815618be837b237

Após inserido tal *token* nos *nodes*, deve-se voltar ao *master* para rodar o comando *kubectl get nodes*, o qual valida se os *workers* fazem parte do *cluster* e estão prontos para serem utilizados.