FACULDADE DE TECNOLOGIA DE AMERICANA
Curso de Tecnologia em Segurança da Informação

José Guilherme Guimarães de Oliveira 0040971621034
Gabriel e Silva Botelho 0040971711026

ORQUESTRAÇÃO DE CONTAINERS COM KUBERNETES

Americana, SP
2019

FACULDADE DE TECNOLOGIA DE AMERICANA
Curso de Tecnologia em Segurança da Informação

José Guilherme Guimarães de Oliveira 0040971621034
Gabriel e Silva Botelho 0040971711026

ORQUESTRAÇÃO DE CONTAINERS COM KUBERNETES

Trabalho de Conclusão de Curso desenvolvido em cumprimento à exigência curricular do Curso de Tecnologia em Segurança da Informação, sob orientação do(a) .

Área de concentração: Desenvolvimento de Jogos

Americana, SP
2019

RESUMO

Kubernetes é definido como uma sistema de código aberto para automatizar a implantação, escalabilidade e gerenciamento de aplicações em *containers*. Projetado pelo Google, baseia-se em anos de experiencia com o uso de *container*.

Palavras-chave:*Kubernetes*. *container*. código aberto.

ABSTRACT

Kubernetes is defined as an open source system to automate deployment, scalability and application management in containers. Designed by Google, it is based on years of experience with container use.

Keywords: Kubernetes. container. open source.

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

De acordo com a Cloud Native Computing Foundation (2019). Passamos por três eras no que diz respeito ha implantação de software: a tradicional, virtualizada e com *containers*.

Tradicionalmente, no início, as organizações executavam aplicativos em servidores físicos. Não havia como definir limites de recursos para aplicativos em um servidor físico, e isso causava problemas de alocação de recursos. Por exemplo, se vários aplicativos forem executados em um servidor físico, pode haver casos em que um aplicativo ocuparia a maioria dos recursos e, como resultado, os outros aplicativos teriam um desempenho inferior. Uma solução para isso seria executar cada aplicativo em um servidor físico diferente. Mas isso não escalou os recursos foram subutilizados, e era caro para as organizações manter muitos servidores físicos.

Como solução, a virtualização foi introduzida. Ele permite que você execute várias máquinas virtuais (VMs) na CPU de um único servidor físico. A virtualização permite que os aplicativos sejam isolados entre VMs e fornece um nível de segurança, pois as informações de um aplicativo não podem ser acessadas livremente por outro aplicativo.

A virtualização permite melhor utilização dos recursos em um servidor físico e melhor escalabilidade, porque um aplicativo pode ser adicionado ou atualizado facilmente, reduz os custos de hardware e muito mais. Com a virtualização, você pode apresentar um conjunto de recursos físicos como um cluster de máquinas virtuais descartáveis.

Cada VM é uma máquina completa executando todos os componentes, incluindo seu próprio sistema operacional, sobre o hardware virtualizado.

Os contêineres são semelhantes às VMs, mas possuem propriedades de isolamento relaxadas para compartilhar o Sistema Operacional (SO) entre os aplicativos. Portanto, os contêineres são considerados leves. Semelhante a uma VM, um contêiner possui seu próprio sistema de arquivos, CPU, memória, espaço de processo e muito mais. À medida que são dissociados da infraestrutura subjacente, eles são portáveis em nuvens e distribuições de SO.

Esse trabalho tem o proposito de mostrar a ferramenta Kubernetes, tem como objetivo mostrar como ele funciona, mostrando os problemas que ele resolve. Na segunda sessão irei abordar mais profundamente sobre virtualização e containes, na terceira sessão irei abordar sobre o Kubernetes

2 FUNDAMENTAÇÃO TEÓRICA

2.1 VIRTUALIZAÇÃO

Virtualização, basicamente, é a técnica de separar aplicação e sistema operacional dos componentes físicos. Por exemplo, uma máquina virtual possui aplicação e sistema operacional como um servidor físico, mas estes não estão vinculados ao software e pode ser disponibilizado onde for mais conveniente. Uma aplicação deve ser executada em um sistema operacional em um determinado software. Com virtualização de aplicação ou apresentação, estas aplicações podem rodar em um servidor ou ambiente centralizado e ser deportada para outros sistemas operacionais e hardwares.

2.1.1 *HYPERVISOR*

O *hypervisor* é um *firmware* ou hardware que cria e roda máquinas virtuais (*VMs*). O computador no qual o *hypervisor* roda uma ou mais VMs é chamado de máquina hospedeira (*host*), e cada *VM* é chamada de máquina convidada (*guest*). O *hypervisor* se apresenta aos sistemas operacionais convidados como uma plataforma de virtualização e gerencia a execução dos sistemas operacionais convidados.

Existem dois tipos de hypervisor, o primeiro tipo é conhecido como bare matal, onde o próprio sistema operacional que gerencia as VM, podemos citar o VMware: ESX, Xem, CubeOS, Microsoft Hyper-V. O segudo modelo é o hosted onde o hypervisor se encontra encima do sistema operacional, seja Linux, Windowns ou MacOS, podemos citar o virtualbox, VMware: Workstation, QEMU, OVirt.

2.1.2 VIRTUALIZAÇÃO COMPLETA E PARCIAL

Como o próprio nome sugere, a virtualização completa realiza toda a abstração do sistema físico, com o objetivo de fornecer ao sistema operacional hóspede uma réplica do hardware virtualizado pelo hospedeiro. Este tipo dispensa a necessidade de modificar o SO convidado, que trabalha desconhecendo que há virtualização.

Com a virtualização total, as instruções não críticas são executadas diretamente no hardware, enquanto as instruções críticas são interceptadas e executadas pela hypervisor. O sistema operacional visitante, no entanto, sequer tem o conhecimento de que está sendo executado sobre o hypervisor.

Já um SO convidado paravirtualizado tem a assistência de um compilador inteligente que atua na substituição de instruções de SO não virtualizáveis por hiperchamadas (hypercalls) quando for executar uma instrução sensível. Tal procedimento poupa o desempenho, quando comparado ao que foi descrito na virtualização total.

Em relação aos dispositivos de E/S, a paravirtualização permite que as máquinas virtuais usem os drivers do dispositivo físico real sob o controle do hipervisor, o que reduz os problemas de compatibilidade.

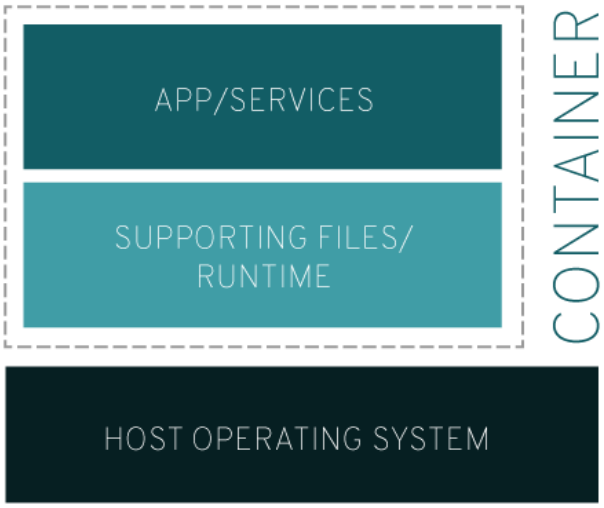
2.2 CONTAINER LINUX

A Red Hat (2019) em seu artigo "O que é um container Linux?"define container Linux como:

> Um container Linux é um conjunto de um ou mais processos organizados isoladamente do sistema. Todos os arquivos necessários à execução de tais processos são fornecidos por uma imagem distinta. Na prática, os containers Linux são portáteis e consistentes durante toda a migração entre os ambientes de desenvolvimento, teste e produção. Essas características os tornam uma opção muito mais rápida do que os pipelines de desenvolvimento, que dependem da replicação dos ambientes de teste tradicionais.

Por meio dessa definição pode-se perceber que os contaiers são bons no desenvolvimento de software, pois os programadores ao criar seus programas em um ambiente isolado e unificado, tanto na parte do desenvolvimento quanto no uso em produção.

Figura 1 – Container LXC



Fonte: Red Hat (2019)

Para rodar uma aplicação ou serviço, os containers Linux possuem as bibliotecas e as dependências dentro dele mesmo, assim possibilitando a migração entre os ambientes como o de desenvolvimento, teste e produção. A imagem mostra a camada de aplicações rodando em cima da camada de aquivos de suportes, que são as dependências e bibliotecas, isso tudo de maneira isolada do sistema operacional.

2.2.1 EVOLUÇÃO DOS CONTAINERS

Os conteiners não surgiram no Linux, mas surgiu inicialmente no FreeBSD em 2000 como jails, essa tecnologia ê capaz de particionar um sistema FreeBSD em vários subsistemas ou celas (por causa disso o nome "jails"). Eles foram desenvolvidos para serem ambientes seguros que podiam ser compartilhados entre os membros de equipe ou colegas de trabalho. Os jails tem como propósito, criar processos em um ambiente isolado utilizando o chroot ("Changing root"ou trocando a raiz), que consiste em mentir para os processos, para achar que a raiz ou o começo do sistema de arquivos está em uma determinada pasta, fazendo eles percam o acesso ao sistema de arquivos real. Um sistema operacional inteiro rodado de maneira isolada a partir de uma pasta. Contudo a implementação estava incompleta e com o tempo foram descobertos métodos de escapar do ambiente em jail.

Logo após, foram implementadas no Linux os grupos de controle (cgroups) que limita o uso de recursos por um processo ou grupo de processos. Com a mudança do sistema de inicialização padrão do sysV para systemd, que possui uma melhor integração com o cgroups, possibilitou ter mais controle sobre os processos isolados. Ambas as tecnologias, além de adicionarem um controle geral ao Linux, serviram como estrutura para a separação eficaz de ambientes.

Com os avanços em namespaces de kernel, pode-se virtualizar, desde IDs de processos a nomes de rede. E os namespaces de usuários "permitem realizar mapeamentos de IDs de usuários e grupos por namespace. No que diz respeito a containers, isso faz com que os usuários e grupos podem ter privilégios para executar determinadas operações dentro de um container, sem ter esses mesmos privilégios fora dele. Por conta disso surgiu o projeto Linux Containers (LXC) que melhorou a experiência do usuário na utilização de containers, contribuindo com ferramentas para facilitar a inicialização de containers com uma interface de linha de comando simples.

Porém, mesmo com os avanços do LXC, estava longe de ser amigável com o publico. Com isso surgiu o docker, mudando completamento a forma de criar, gerenciar, compartir, depurar e de iniciar containers. Levando a tecnologia de containers a outro patamar, sendo utilizado pelas maiores impressas no mundo.

2.3 CONTAINER ENGINE

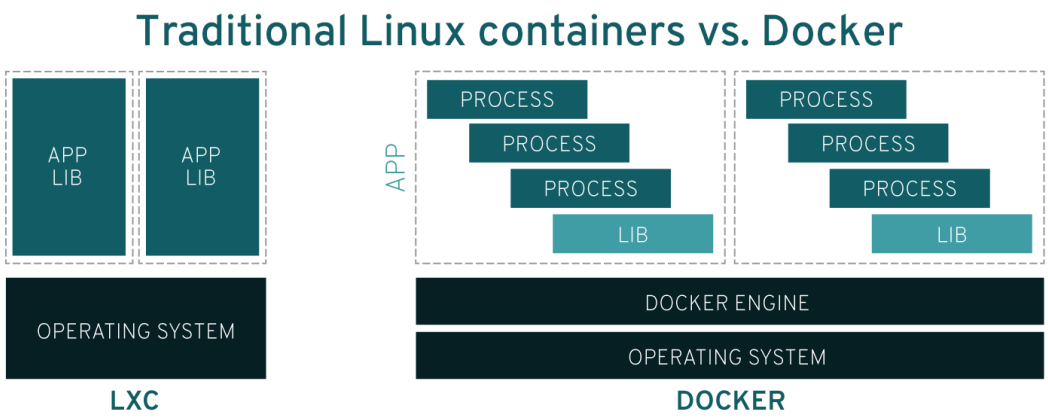
Em 2008, o Docker entrou em cena (por meio do dotCloud) com sua tecnologia de container homônima. A tecnologia Docker adicionou muitos dos novos conceitos e ferramentas: uma interface de linha de comando simples para executar e criar novas imagens em camadas, um daemon de servidor, uma biblioteca de imagens de container pré-criadas e o conceito de servidor de registros. Combinadas, essas tecnologias possibilitaram aos usuários criar novos containers em camadas com rapidez e compartilhá-los facilmente com outras pessoas.

A Red Hat reconheceu o poder da colaboração nesse novo ecossistema e usou a

tecnologia subjacente no nosso OpenShift Container Platform. Para afastar o receio de haver um único fornecedor controlando uma tecnologia tão importante, a Docker Inc. contribuiu com muitos dos componentes subjacentes utilizados em projetos open source realizados pela comunidade (o runc faz parte da Open Containers Initiative (OCI) e o containerd foi transferido para o CNCF).

Há três padrões principais que garantem a interoperabilidade das tecnologias de containers: as especificações Image, Distribution e Runtime da OCI. A combinação dessas especificações permite que projetos da comunidade, soluções comerciais e provedores de cloud criem tecnologias de container interoperáveis (por exemplo, pense em uma situação em que você precisa introduzir imagens personalizadas no servidor de registro do provedor de cloud). Atualmente, a Red Hat e a Docker, juntamente com muitas outras organizações, são membros da Open Container Initiative, cujo objetivo é padronizar as tecnologias de containers no setor open source.

Figura 2 – LXC vs docker



Fonte: Red Hat (2019)

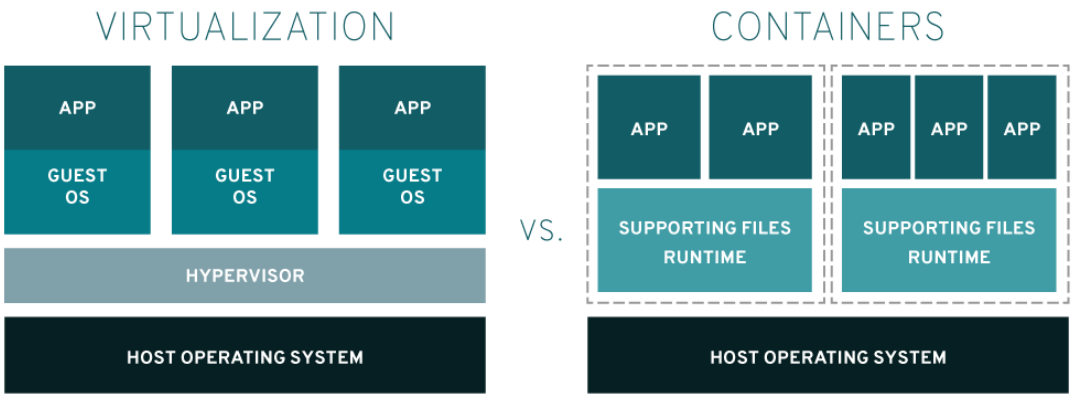
2.4 VIRTUALIZAÇÃO VS CONTAINER

Em bora a virtualização e os containers se pareçam muito, sáo duas tecnologias diferentes, que complementam-se. De acordo com a Red Hat (2019), a virtualização tem o proposito de executar sistemas operacionais simultaneamente em um único sistema de hardware. Já os containers compartilham o mesmo núcleo do sistema operacional e isolam os processos da aplicação do restante do sistema. Os containers Linux são extremamente portáteis, mas devem ser compatíveis com o sistema operacional subjacente.

A virtualização usa um hipervisor para emular o hardware e essa não é uma solução tão leve quanto o uso de containers. Os containers são executados de maneira nativa no sistema operacional. Em comparação com as máquinas virtuais, executar containers Linux consome

menos recursos e facilita o gerenciamento dos processos. Além disso, é possível orquestrar as aplicações em vários containers em diversas nuvens.

Figura 3 – Virtualização vs container



Fonte: Red Hat (2019)

3 KUBERNETES

O Kubernetes é uma plataforma portátil, extensível e de código aberto para gerenciar cargas de trabalho e serviços em contêiner, que facilita a configuração declarativa e a automação. Possui um ecossistema grande e de rápido crescimento. Os serviços, suporte e ferramentas do Kubernetes estão amplamente disponíveis.

O nome Kubernetes é originário do grego, significando timoneiro ou piloto. O Google abriu o projeto Kubernetes em 2014. O Kubernetes tem uma década e meia de experiência que o Google tem em executar cargas de trabalho de produção em grande escala, combinadas com as melhores idéias e práticas da comunidade.

Ao implantar o Kubernetes, você obtém um cluster.

Um cluster é um conjunto de máquinas, chamadas nós, que executam aplicativos em contêiner gerenciados pelo Kubernetes. Um cluster possui pelo menos um nó de trabalho e pelo menos um nó principal.

Este documento descreve os vários componentes necessários para ter um cluster Kubernetes completo e funcionando.

Aqui está o diagrama de um cluster Kubernetes com todos os componentes unidos.

3.1 COMPONENTES PRINCIPAIS

Os componentes principais fornecem o plano de controle do cluster. Os componentes principais tomam decisões globais sobre o cluster (por exemplo, agendamento) e detectam e respondem a eventos do cluster (por exemplo, iniciando um novo pod quando o campo de réplicas de uma implantação não está satisfeito).

Os componentes principais podem ser executados em qualquer máquina no cluster. No entanto, para simplificar, os scripts de configuração geralmente iniciam todos os componentes principais na mesma máquina e não executam contêineres de usuários nessa máquina. Consulte Criando clusters de alta disponibilidade para obter um exemplo de instalação da VM com várias master.

3.1.1 kube-apiserver

O servidor da API é um componente do plano de controle do Kubernetes que expõe a API do Kubernetes. O servidor da API é o front end do plano de controle do Kubernetes.

A principal implementação de um servidor de API do Kubernetes é o kube-apiserver. O kube-apiserver foi desenvolvido para ser dimensionado horizontalmente, ou seja, é dimensionado com a implantação de mais instâncias. Você pode executar várias instâncias do kube-apiserver e equilibrar o tráfego entre essas instâncias.

3.1.2 etcd

Armazenamento de valores-chave consistente e de alta disponibilidade usado como repositório do Kubernetes para todos os dados do cluster.

Se o cluster do Kubernetes usa o etcd como repositório de backup, verifique se você tem um plano de backup para esses dados.

Você pode encontrar informações detalhadas sobre o etcd na documentação oficial

3.1.3 kube-scheduler

Componente no mestre que assiste os pods recém-criados que não têm nenhum nó designado e seleciona um nó para execução.

Os fatores levados em consideração nas decisões de agendamento incluem requisitos de recursos individuais e coletivos, restrições de hardware/software/política, especificações de afinidade e anti-afinidade, localidade dos dados, interferência entre cargas de trabalho e prazos.

3.1.4 kube-controller-manager

Componente no mestre que executa controladores.

Logicamente, cada controlador é um processo separado, mas para reduzir a complexidade, todos são compilados em um único binário e executados em um único processo.

Esses controladores incluem:

- Controlador de nó: responsável por perceber e responder quando os nós são desativados.
- Controlador de replicação: responsável por manter o número correto de pods para cada objeto do controlador de replicação no sistema.
- Controlador de pontos finais: preenche o objeto Pontos finais (ou seja, junta-se a Serviços e pods).
- Conta de serviço e controladores de token: crie contas padrão e tokens de acesso à API para novos namespaces

3.1.5 cloud-controller-manager

O cloud-controller-manager executa controladores que interagem com os provedores de nuvem subjacentes. O binário cloud-controller-manager é um recurso alfa introduzido no Kubernetes versão 1.6.

O cloud-controller-manager executa apenas loops de controlador específicos do provedor de nuvem. Você deve desativar esses loops de controlador no kube-controller-manager. Você pode desativar os loops do controlador configurando o sinalizador –cloud-provider como externo ao iniciar o kube-controller-manager.

o gerenciador de controlador de nuvem permite que o código do fornecedor de nuvem e o código Kubernetes evoluam independentemente um do outro. Em versões anteriores, o código principal do Kubernetes dependia do código específico do provedor de nuvem para

funcionalidade. Em versões futuras, o código específico para os fornecedores de nuvem deve ser mantido pelo próprio fornecedor de nuvem e vinculado ao gerenciador de controlador de nuvem durante a execução do Kubernetes.

Os seguintes controladores têm dependências do provedor de nuvem:

- Node Controller: For checking the cloud provider to determine if a node has been deleted in the cloud after it stops responding
- Route Controller: For setting up routes in the underlying cloud infrastructure
- Service Controller: For creating, updating and deleting cloud provider load balancers
- Volume Controller: For creating, attaching, and mounting volumes, and interacting with the cloud provider to orchestrate volumes

3.2 Componentes do nó

Node components run on every node, maintaining running pods and providing the Kubernetes runtime environment.

3.2.1 kubelet

Um agente que é executado em cada nó no cluster. Ele garante que os contêineres estejam sendo executados em um pod.

O kubelet utiliza um conjunto de PodSpecs que são fornecidos por vários mecanismos e garante que os contêineres descritos nesses PodSpecs estejam funcionando e funcionando corretamente. O kubelet não gerencia contêineres que não foram criados pelo Kubernetes.

3.2.2 kube-proxy

O kube-proxy é um proxy de rede que é executado em cada nó do cluster, implementando parte do conceito de Serviço Kubernetes.

O kube-proxy mantém regras de rede nos nós. Essas regras de rede permitem a comunicação em rede com seus Pods a partir de sessões de rede dentro ou fora do cluster.

O kube-proxy usa a camada de filtragem de pacotes do sistema operacional, se houver uma disponível. Caso contrário, o kube-proxy encaminha o próprio tráfego.

3.2.3 Container Runtime

O tempo de execução do contêiner é o software responsável pela execução de contêineres.

O Kubernetes suporta vários tempos de execução do contêiner: Docker, container, cri-o, rktlet e qualquer implementação do Kubernetes CRI (Container Runtime Interface).

4 CONSIDERAÇÕES FINAIS

Referências

CLOUD NATIVE COMPUTING FOUNDATION. **Concepts**. 2019. Disponível em: <https://kubernetes.io/docs/concepts/>. Acesso em: 28 de novenbro de 2019. Citado na página 1.

RED HAT. **O que um é container Linux?** 2019. Disponível em: <https://www.redhat.com/pt-br/topics/containers/whats-a-linux-container>. Acesso em: 26 de setembro de 2019. Citado 3 vezes nas páginas 3, 5 e 6.