História e Conceitos Básicos do Kubernetes

1. História

Kubernetes é uma plataforma de orquestração de contêineres que surgiu da necessidade de gerenciar e escalar aplicações em contêineres em larga escala. O projeto foi iniciado pelo Google em 2014, mas suas raízes remontam a projetos internos do Google, como o Borg e o Omega.

- **Origens**: O Google enfrentava desafios significativos ao gerenciar sua vasta infraestrutura, que executava bilhões de contêineres semanalmente. Para resolver isso, eles desenvolveram o Borg, um sistema interno de gerenciamento de clusters. A experiência adquirida com Borg e Omega levou à criação do Kubernetes, que foi projetado como uma solução de código aberto para a comunidade.

- **Lançamento**: Em 2014, o Kubernetes foi lançado como um projeto de código aberto, sob a gestão da Cloud Native Computing Foundation (CNCF). Ele rapidamente ganhou popularidade por sua capacidade de abstrair a complexidade de gerenciar contêineres, tornando-o uma escolha ideal para empresas e desenvolvedores

2. Evolução do Kubernetes

Desde o seu lançamento, o Kubernetes passou por uma evolução significativa, tanto em termos de funcionalidades quanto de adoção.

- **Adoção em Massa**: Empresas de todos os tamanhos, de startups a grandes corporações, começaram a adotar o Kubernetes para gerenciar suas aplicações em contêineres. A capacidade do Kubernetes de funcionar em diferentes ambientes, como nuvem, on-premises e híbrido, contribuiu para sua rápida adoção.

- **Comunidade e Contribuições**: O Kubernetes se beneficiou de uma comunidade ativa, com contribuições de empresas líderes como Google, Red Hat, IBM, Microsoft, e muitas outras. Isso resultou em um rápido desenvolvimento de novas funcionalidades e melhorias.

- **Integração com Outras Tecnologias:** Kubernetes se integrou com outras ferramentas e tecnologias, como Prometheus para monitoramento, Istio para malha de serviços, e Helm para gerenciamento de pacotes, ampliando suas capacidades e tornando-o um componente central no ecossistema de DevOps

3. Princípios Básicos do Kubernetes

Kubernetes é baseado em vários princípios fundamentais que o tornam robusto, flexível e escalável.

- **Automação:** Kubernetes automatiza a implantação, escalabilidade, e gerenciamento de contêineres. Isso inclui a automação de tarefas como o escalonamento de aplicações com base na carga, o reinício de contêineres falhos, e o balanceamento de carga.

- **Desacoplamento:** Kubernetes promove o desacoplamento entre as aplicações e a infraestrutura subjacente. Isso significa que as aplicações

são executadas de maneira consistente, independentemente do ambiente em que estão hospedadas.

- **Escalabilidade:** Um dos principais objetivos do Kubernetes é permitir que aplicações sejam escaladas horizontalmente (adicionando mais instâncias de contêineres) de maneira eficiente e sem intervenção manual.

- **Resiliência:** Kubernetes foi projetado para ser resiliente a falhas. Ele pode detectar falhas em aplicações e automaticamente reiniciar ou substituir os contêineres falhos.

4. Arquitetura do Kubernetes

A arquitetura do Kubernetes é composta por um conjunto de componentes que trabalham juntos para orquestrar a execução de contêineres.

- **Master Node:** O master node é o cérebro do cluster Kubernetes. Ele gerencia o agendamento dos contêineres, a manutenção do estado desejado do cluster, e a coordenação de todas as atividades no cluster.
  - **Componentes do Master Node.**
    - API Server: O ponto de entrada para todas as operações no cluster.
    - Scheduler: Responsável por distribuir as cargas de trabalho nos nodes.
    - Controller Manager: Garante que o estado do cluster esteja conforme especificado nas definições.
    - etcd: Armazena dados de configuração e estado do cluster.

- **Worker Nodes:** São as máquinas que executam os contêineres. Cada worker node contém um kubelet, que é responsável por garantir que os

contêineres estejam rodando como esperado.

- **Componentes do Worker Node:**
  - Kubelet: Agente que comunica com o master node e aplica as decisões feitas por ele.
  - Container Runtime: O software responsável por rodar os contêineres (por exemplo, Docker).
  - Kube-proxy: Gerencia a rede do node e permite a comunicação entre os serviços.

5. Principais Componentes do Kubernetes

Kubernetes tem vários componentes-chave que permitem o gerenciamento eficiente de contêineres.

- **Pods:** A menor unidade de implantação no Kubernetes. Um pod pode conter um ou mais contêineres que compartilham recursos de rede e armazenamento.
- **Services:** Um serviço define um conjunto lógico de Pods e uma política para acessá-los. Ele abstrai os pods subjacentes, fornecendo uma única interface para comunicação.
- **Deployments**: Um deployment gerencia a criação e atualização de Pods. Ele permite a definição de estratégias de atualização, como rolling updates, para minimizar o tempo de inatividade.
- **ConfigMaps e Secrets:** Permitem o gerenciamento de dados de configuração e segredos (como senhas e tokens) usados pelos contêineres.

- **Namespaces:** Fornecem um escopo para nomes no Kubernetes, permitindo a segmentação de recursos em diferentes contextos ou ambientes.

6. Comparação do Kubernetes com Outras Ferramentas

Kubernetes é frequentemente comparado com outras ferramentas de orquestração de contêineres e plataformas de infraestrutura como serviço.

- **Docker Swarm:**
  - Simplicidade: Docker Swarm é mais simples de configurar e usar, mas oferece menos recursos em comparação com Kubernetes.
  - Escalabilidade: Kubernetes é mais robusto e escalável, sendo mais adequado para ambientes complexos e de grande escala.
- **Apache Mesos:**
  - Versatilidade: Mesos é uma plataforma mais genérica que pode orquestrar não apenas contêineres, mas também outros tipos de workloads. No entanto, Kubernetes oferece uma experiência mais focada e otimizada para contêineres.
  - Adoção: Kubernetes tem uma adoção muito maior na indústria, com um ecossistema mais rico e uma comunidade mais ativa.
- **OpenShift:**
  - Enterprise Features: OpenShift, baseado em Kubernetes, oferece recursos adicionais voltados para empresas, como suporte integrado a CI/CD e maior segurança.
  - Complexidade: Embora ofereça mais funcionalidades, OpenShift pode ser mais complexo de configurar e gerenciar do que o Kubernetes puro.

VMs, Contêineres e o Cenário Antes da Computação em Nuvem

1. O Cenário Antes da Computação em Nuvem

Antes do advento da computação em nuvem, a maneira tradicional de hospedar e gerenciar aplicações envolvia o uso de servidores físicos dedicados. Cada aplicação geralmente rodava em seu próprio servidor, o que trazia diversos desafios:

- **Capacidade Subutilizada:** Servidores físicos eram frequentemente subutilizados, pois cada servidor era configurado com recursos para lidar com picos de carga, mas na maior parte do tempo, esses recursos ficavam ociosos.

- **Manutenção e Escalabilidade:** A manutenção de servidores físicos era complexa e cara. Escalar a infraestrutura significava adquirir e configurar novos servidores, o que demandava tempo e planejamento.

- Isolamento e Segurança: Aplicações rodando no mesmo servidor podiam interferir umas nas outras, resultando em problemas de segurança e isolamento.

2. Máquinas Virtuais (VMs)

A virtualização surgiu como uma solução para muitos dos problemas associados ao uso de servidores físicos. Máquinas Virtuais (VMs) permitem a execução de múltiplos sistemas operacionais isolados em um único servidor físico.

- **Funcionamento:**
  - Uma VM é uma emulação de um computador físico, com seu próprio sistema operacional e aplicativos. Ela roda em cima de um hypervisor,

que é o software que gerencia e aloca os recursos do hardware físico para as VMs.

- Exemplo de hypervisores: VMware ESXi, Microsoft Hyper-V, Xen e KVM.

- **Vantagens:**
  - Melhor Utilização dos Recursos: VMs permitem que vários sistemas operacionais e aplicações rodem simultaneamente no mesmo hardware físico, maximizando a utilização dos recursos.
  - Isolamento: Cada VM é isolada das outras, o que melhora a segurança e a estabilidade.
  - Facilidade de Gerenciamento: A criação, clonagem, e backup de VMs é mais simples e rápida do que a de servidores físicos.
- **Desvantagens:**
  - Overhead de Recursos: Cada VM requer seu próprio sistema operacional completo, o que pode resultar em overhead significativo de recursos (CPU, memória e armazenamento).

3. Contêineres

Os contêineres são uma evolução da virtualização que oferece uma abordagem mais leve e eficiente para rodar múltiplas aplicações em um único servidor.

- **Funcionamento:**
  - Ao contrário das VMs, os contêineres compartilham o kernel do sistema operacional host e isolam apenas o espaço de usuário. Isso significa que múltiplos contêineres podem rodar em cima de um único

sistema operacional, com cada contêiner contendo apenas as bibliotecas e dependências necessárias para a aplicação.

  - Exemplo de tecnologias de contêinerização: Docker, LXC, rkt.

- **Vantagens:**
  - Leveza: Como os contêineres compartilham o kernel do sistema operacional, eles são muito mais leves que as VMs, consumindo menos recursos.

  - Rapidez: Contêineres iniciam e param quase instantaneamente, o que os torna ideais para ambientes de desenvolvimento e implantação contínua.

  - Portabilidade: Contêineres encapsulam tudo que a aplicação precisa para rodar, garantindo que ela funcione da mesma forma em qualquer ambiente (desenvolvimento, teste, produção).

- **Desvantagens:**
  - Isolamento Menos Rigoroso: Embora os contêineres ofereçam isolamento entre aplicações, ele não é tão robusto quanto o isolamento proporcionado por VMs, uma vez que todos os contêineres compartilham o mesmo kernel.

4. Comparação Entre VMs e Contêineres

- **Arquitetura:**
  - VMs: Cada VM possui seu próprio sistema operacional completo, rodando em cima de um hypervisor.

  - Contêineres: Compartilham o kernel do sistema operacional host, isolando apenas o ambiente de aplicação.

- **Desempenho e Recursos:**
  - VMs: Têm overhead maior devido à necessidade de emular hardware e rodar múltiplos sistemas operacionais completos.
  - Contêineres: São mais eficientes em termos de uso de recursos, com menor overhead.
- **Tempo de Inicialização:**
  - VMs: Podem demorar minutos para iniciar, dependendo do sistema operacional e dos recursos disponíveis.
  - Contêineres: Iniciam em segundos ou até menos, sendo muito mais ágeis.
- **Isolamento:**
  - VMs: Proporcionam isolamento forte, ideal para ambientes onde a segurança é uma grande preocupação.
  - Contêineres: Oferecem isolamento suficiente para a maioria dos casos de uso, mas dependem do mesmo kernel.

5. O Impacto da Computação em Nuvem

A chegada da computação em nuvem transformou ainda mais o panorama de VMs e contêineres, ao introduzir a possibilidade de escalar recursos de maneira elástica e sob demanda.

- Infraestrutura como Serviço (IaaS): Serviços como Amazon EC2, Google Compute Engine, e Microsoft Azure Virtual Machines permitem a criação e gestão de VMs na nuvem, proporcionando escalabilidade e flexibilidade incomparáveis.

- Plataforma como Serviço (PaaS) e Contêineres na Nuvem: Com a popularização de contêineres, surgiram plataformas como Google Kubernetes Engine (GKE), AWS Elastic Kubernetes Service (EKS), e Azure Kubernetes Service (AKS), que facilitam o gerenciamento e a orquestração de contêineres em larga escala.

- Serverless: A computação serverless, embora não seja baseada em VMs ou contêineres diretamente, exemplifica a evolução para abstrações ainda mais altas, onde os desenvolvedores focam apenas no código, e a infraestrutura subjacente é completamente gerenciada pelo provedor de nuvem.

Conteúdo Bônus

Artigo: "Introdução à arquitetura do Kubernetes"

Este artigo da Red Hat explora a origem do Kubernetes, sua evolução e os conceitos fundamentais que o sustentam.

Referências Bibliográficas

BASSO, D. E. **Administração de Redes de Computadores**. Contentus, 2020.

KUROSE, J. F.; ROSS, K. W. **Redes de Computadores e a Internet: Uma Abordagem Top-Down**. 8. ed. Pearson, 2021.

MARINHO, A. L.; CRUZ, J. L. da. (Orgs.). **Desenvolvimento de Aplicações para Internet**. 2. ed. Pearson, 2020.

PUGA, S.; RISSETTI, G. **Lógica de Programação e Estruturas de Dados, com Aplicações em Java**. 3. ed. Pearson, 2016.

ROHLING, L. J. **Segurança de Redes de Computadores**. Contentus, 2020.

SILVA, C. F. da. **Projeto Estruturado e Gerência de Redes**. Contentus, 2020.

TANENBAUM, A. S.; FEAMSTER, N.; WETHERALL, D. J. **Redes de Computadores**. 6. ed. Pearson, 2021.

TOCCI, R. J.; WIDMER, N. S.; MOSS, G. L. **Sistemas Digitais: Princípios e Aplicações**. 12. ed. Pearson, 2018.

Ir para exercício