THIAGO ADRIANO

PDS TECH

SOFTWARE ARCHITECTURE

KUBERNETES

AULA 01

SUMÁRIO

O QUE VEM POR AÍ?

Aprenderemos como os containers nos ajudam no desenvolvimento do dia a dia, mas e quando temos dezenas ou centenas de containers para administrar? Para isso, vamos aprender sobre o Kubernetes.

HANDS ON

Na primeira videoaula, os professores apresentaram virtualização por meio de alguns desenhos apresentados pelo professor Thiago, e finalizaram falando sobre Docker.

Avançando para segunda videoaula, os professores demonstraram como o Docker funciona na prática, criando uma imagem por meio de um projeto desenvolvido pelo professor Thiago, e depois avançamos com o básico de Kubernetes, criando o nosso primeiro Pod.

SAIBA MAIS

MOTIVAÇÃO

Para começarmos a falar de Kubernetes, precisamos primeiramente entender em qual cenário, contexto e problema essa solução tecnológica se encaixa. Precisamos falar sobre infraestrutura de sistemas de software!

Existem dois problemas comuns e latentes em todos os projetos de software: gestão de múltiplos ambientes (desenvolvimento, testes, produção) e escalabilidade (capacidade que o sistema tem de ‘crescer’ dada uma determinada necessidade).

Quando falamos de ambientes, estamos colocando em apenas uma palavra, os recursos de hardware, rede e infraestrutura necessários para que desenvolvedores construam e testem suas aplicações; equipes de QA testem e homologuem as aplicações antes de serem disponibilizadas para os usuários finais; aplicações estejam disponíveis para o usuário final.

Poderíamos abordar os custos envolvidos na criação e manutenção desses três ambientes, mas temos também a grande dificuldade de mantermos três ambientes, que em termos de configuração, capacidade computacional, dados e rede, devem ser minimamente parecidos.

A pessoa desenvolvedora constrói sua aplicação tendo em vista um conjunto de atributos de infraestrutura, que por sua vez deve estar de acordo com o ambiente a ser utilizado pelo usuário final. O ambiente de testes também deve estar alinhado com aquilo que fora previsto no desenvolvimento e também muito próximo ao ambiente final.

Para problemas complexos, não temos soluções simples! A solução passa por uma composição de: Devops, Cloud Computing e Docker, mas para o momento, vamos nos atentar ao cenário e contexto do problema que temos.

Outro desafio é escalabilidade, ou se preferir, scaling. Este termo está ligado à capacidade que um sistema tem de crescer ou encolher em infraestrutura de acordo com as variações da sua demanda.

Por exemplo, um site de vendas on-line. Durante todo o ano, ele possui uma certa demanda de uso e sua infraestrutura: capacidade de hardware, redes, banco de

dados, etc. A infraestrutura é dimensionada (e adquirida) para essa demanda. Em determinadas épocas do ano, como na black friday, essa demanda aumenta. É no mínimo complexo e caro adquirir capacidade computacional para atender esse cenário. O que fazer com o excedente? O que fazer quando não precisarmos de mais? O que fazer em casos em que a demanda não foi bem dimensionada?

VIRTUALIZAÇÃO

Tanto para o problema de ambientes, quanto para o de escalabilidade, temos uma peça comum que faz parte do quebra-cabeças que compõe a solução para ambos os cenários: o conceito de container.

Poderíamos dizer que o conceito de container é igual à virtualização de máquinas (VM's). Mas é importante entender que eles são semelhantes, não iguais, muito longe disso!

A virtualização atua na criação de uma "pequena máquina" completa: hardware e sistema operacional dentro de uma máquina que chamamos de hospedeira, compartilhando assim apenas os recursos de hardware dessa máquina.

Figura 1 – Exemplo de arquitetura tradicional x virtualizada
Fonte: Elaborado pelo autor (2023)

A arquitetura baseada em containers é criada na camada do sistema operacional da máquina hospedeira, ou seja, além de compartilhar os recursos de hardware dessa máquina, também compartilha dos recursos do próprio sistema operacional dela, fazendo com que um container seja uma "pequena máquina" muito mais leve e barata do que uma máquina virtual, e por tabela, deixando nela tão e

somente os recursos de software, arquivos e bibliotecas necessárias para a execução da aplicação.

Por exemplo, se precisarmos criar um ambiente com banco de dados MySQL com base no S.O. Debian, teremos uma imagem que contém todos os arquivos, binários e bibliotecas necessárias para “subirmos” esse ambiente. Se precisarmos de um ambiente com um servidor web que suporte uma aplicação Java Spring Boot, teremos uma imagem com tudo o que precisamos para essa missão e nada mais.

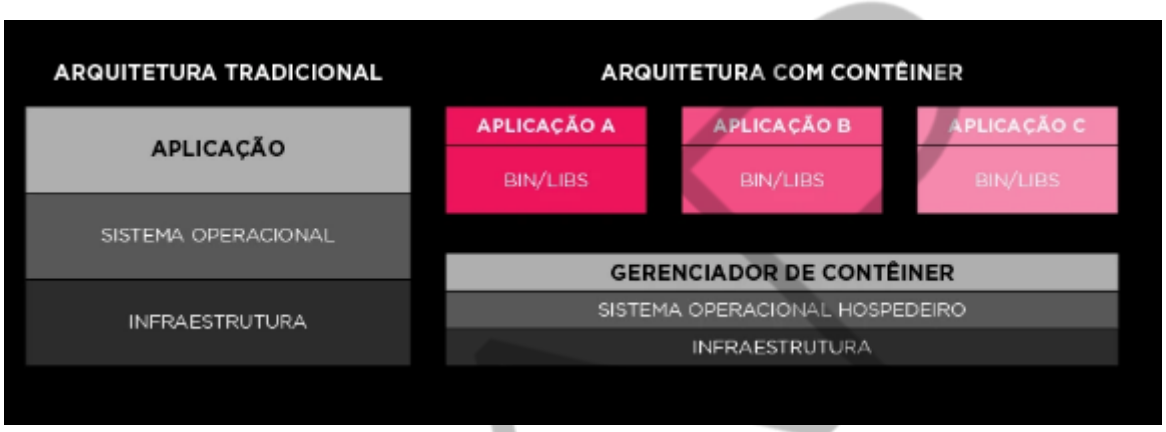

Figura 2 – Exemplo de arquitetura tradicional e com containers
Fonte: Elaborado pelo autor (2023)

Com relação a gestão de ambientes, perceba que temos a facilidade de criarmos e replicarmos ambientes de maneira mais simples e rápida, evitando possíveis diferenças de configurações provenientes de ambientes diferentes. Se falarmos de escalabilidade, podemos criar e destruir recursos computacionais por demanda, atingindo aquilo que chamamos de escalabilidade horizontal.

A escalabilidade vertical é a capacidade de expandirmos os recursos físicos, tanto em ambientes on-premise quanto na nuvem. Englobando os recursos de hardware já existentes, bem como a adição de novos componentes, visando otimizar o desempenho do sistema.

É nesse ponto que entra o Kubernetes (ou K8S). Ele é o responsável por tarefas que, se forem feitas de forma manual, seriam custosas e pouco eficientes: criar, destruir containers a partir de uma imagem, bem como manter a infraestrutura operacional. Ou seja, em caso de falha de um determinado container, é responsabilidade do K8S criar um novo container.

KUBERNETES

A arquitetura do Kubernetes é baseada no conceito de clusters, em que cada nó pode ser composto por um ou mais containers (a depender da otimização necessária), chamado de “POD”.

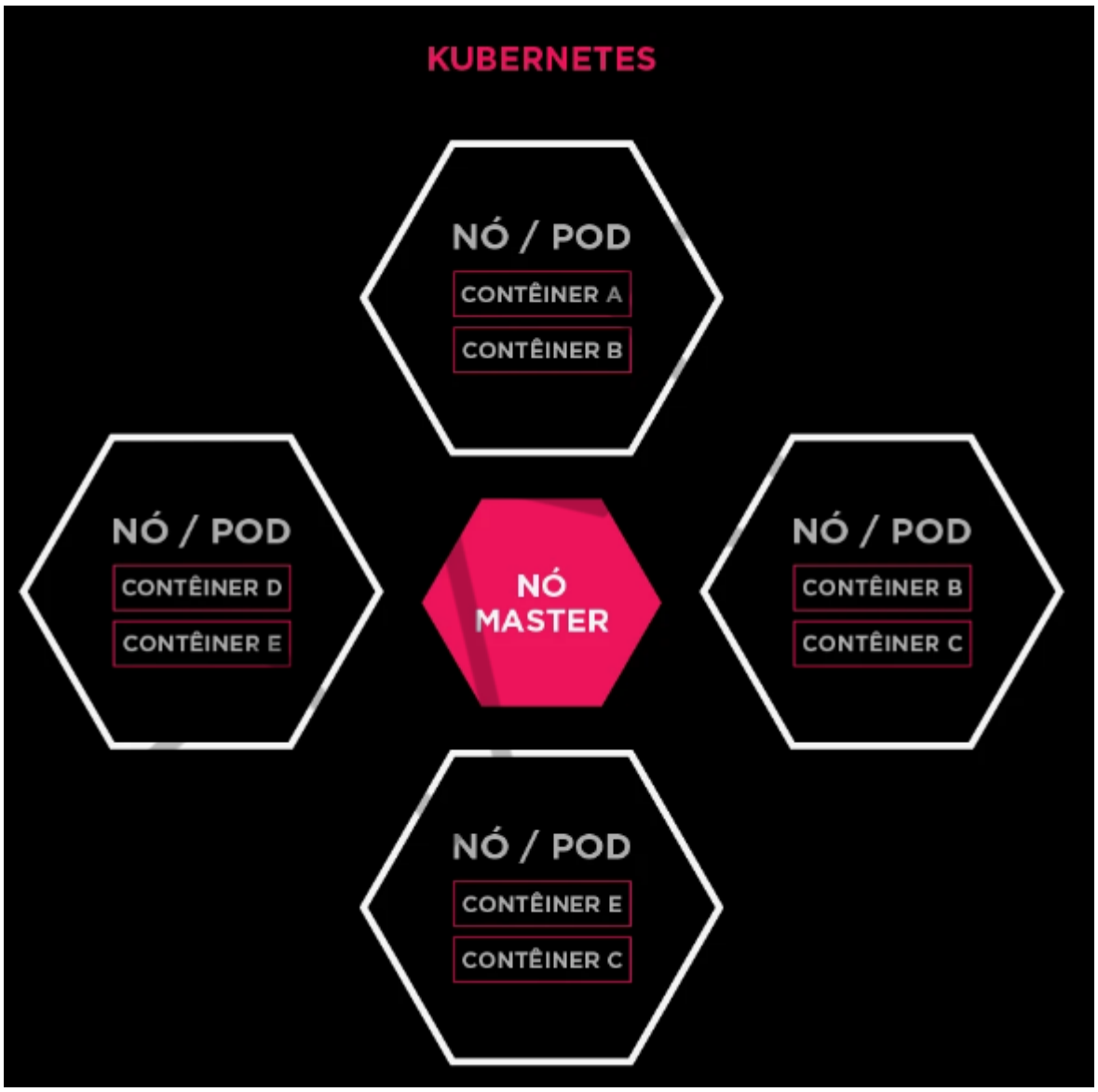

Figura 3 – Exemplo de fluxo do kubernetes
Fonte: Elaborado pelo autor (2023)

Nós precisamos de recursos para a manipulação desses nós! Para isso, o Kubernetes possui uma série de API’s REST disponíveis no nó master da arquitetura. O acesso a essa API pode ser feito a partir de ferramentas. A mais usada é a Kube Control, ou também chamada de Kubectl.

Essa ferramenta, que pode ser instalada em qualquer sistema operacional, atua como client do nó master do cluster e é sem dúvida a forma mais produtiva de trabalharmos com o Kubernetes.

Figura 4 – Exemplo do kubectl
Fonte: Elaborado pelo autor (2023)

PREPARANDO O AMBIENTE

1º PASSO – INSTALAÇÃO DO KUBECTL

Para instalação do kubectl, siga os procedimentos da documentação nos links abaixo, de acordo com o seu sistema operacional:

- **Windows**.

- Linux.

Para validar a instalação, executar o comando “kubectl version --output=yaml”.

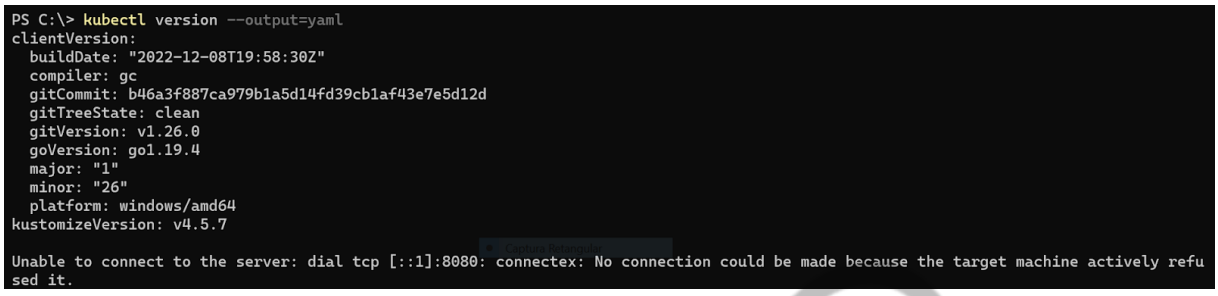

Figura 5 – Executando o output do kubectl
Fonte: Elaborado pelo autor (2023)

2º PASSO - INSTALAÇÃO DO DOCKER

Antes de tudo, o Docker é um produto de software de plataforma aberta que implementa o conceito de container. Ele utiliza uma arquitetura cliente servidor e podemos utilizá-lo em infraestrutura de data center, cloud ou híbrido.

Para instalar, siga o procedimento para o sistema operacional do host (ou o seu, no caso), no site do Docker.

Ao final da instalação, execute o comando “docker -v” para validar se a instalação do Docker foi feita com sucesso.

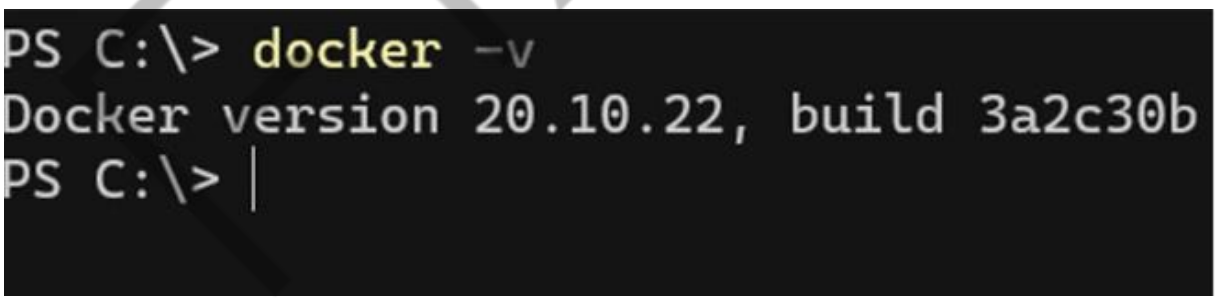

Figura 6 – Executando o docker version
Fonte: Elaborado pelo autor (2023)

3º PASSO – INSTALAÇÃO DO MINIKUBE

O Minikube é uma implementação leve do Kubernetes que cria uma VM em sua máquina local e implanta um cluster simples, contendo apenas um nó. É a melhor indicação para estudos e para ambientes de desenvolvimento local.

Para instalar o Minikube, precisamos antes instalar um mecanismo de virtualização (no nosso caso, o Docker precisa estar instalado no ambiente). Após isso, seguimos o procedimento descrito no site do Minikube (Atenção! Verifique os pré-requisitos de hardware necessários!).

Para validarmos a instalação do Minikube, execute o comando "minikube start". Esse comando buscará uma imagem do Kubernetes diretamente do repositório oficial e a instalará no seu ambiente local (o que pode demorar alguns minutos).

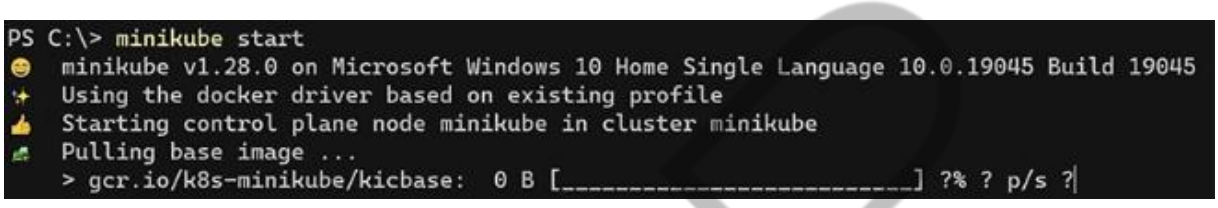

Figura 7 – Iniciando o Minikube
Fonte: Elaborado pelo autor (2023)

Para validar se a instalação foi feita com sucesso, vamos executar o comando "kubectl get pods". Este comando listará todos os Pods criados no nosso ambiente.

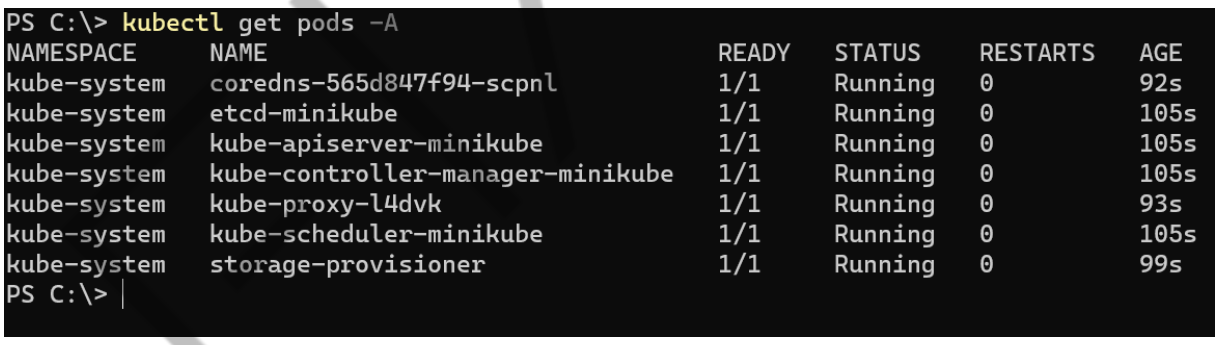

Figura 8 – Buscando os Pods do kubectl
Fonte: Elaborado pelo autor (2023)

O QUE VOCÊ VIU NESTA AULA?

Nesta aula, os professores apresentaram a motivação de se utilizar o Kubernetes por meio de um cenário imaginário. Após isso, apresentaram alguns conceitos sobre virtualização e, por fim, uma introdução ao Kubernetes, que é o tema da nossa disciplina.

O que achou do conteúdo? Conte-nos no Discord! Estamos disponíveis na comunidade para tirar dúvidas, fazer networking, enviar avisos e muito mais.

REFERÊNCIAS

BASS, L.; CLEMENTS, P. KAZMAN, R. Software Architecture in Practice. 3rd ed. [s.l.]: Addison-Wesley Professional, 2012.

PALAVRAS-CHAVE

Arquitetura de Software. Kubernets. Pods.