DevOps

UNIDADE 05

Kubernetes 

UMA INTRODUÇÃO AO KUBERNETES

O Kubernetes, (também abreviado como K8s), tem as suas origens no Google. Ele começou como um projeto interno para gerenciar a infraestrutura gigantesca de serviços da empresa. Para falar a verdade, eles tinham um sistema interno chamado Borg. As melhores ideias do Borg foram convertidas no Kubernetes, e ele se tornou um projeto de código aberto (*open source*).

A ideia do Kubernetes era a de criar uma plataforma que pudesse orquestrar *containers* de forma automática, escalável e resiliente, mas acessível a qualquer organização. Em 2015, o Kubernetes foi transferido para a Cloud Native Computing Foundation (CNCF),

que o mantém e promove seu desenvolvimento.

A principal função do Kubernetes é automatizar a implantação, o dimensionamento e a gestão de aplicações em *containers.* Vamos supor que você tenha uma aplicação composta por vários *containers* que precisam se comunicar, ser escalados automaticamente com base na carga e se recuperar de falhas sem intervenção manual. A ideia do Kubernetes é a de automatizar isso.

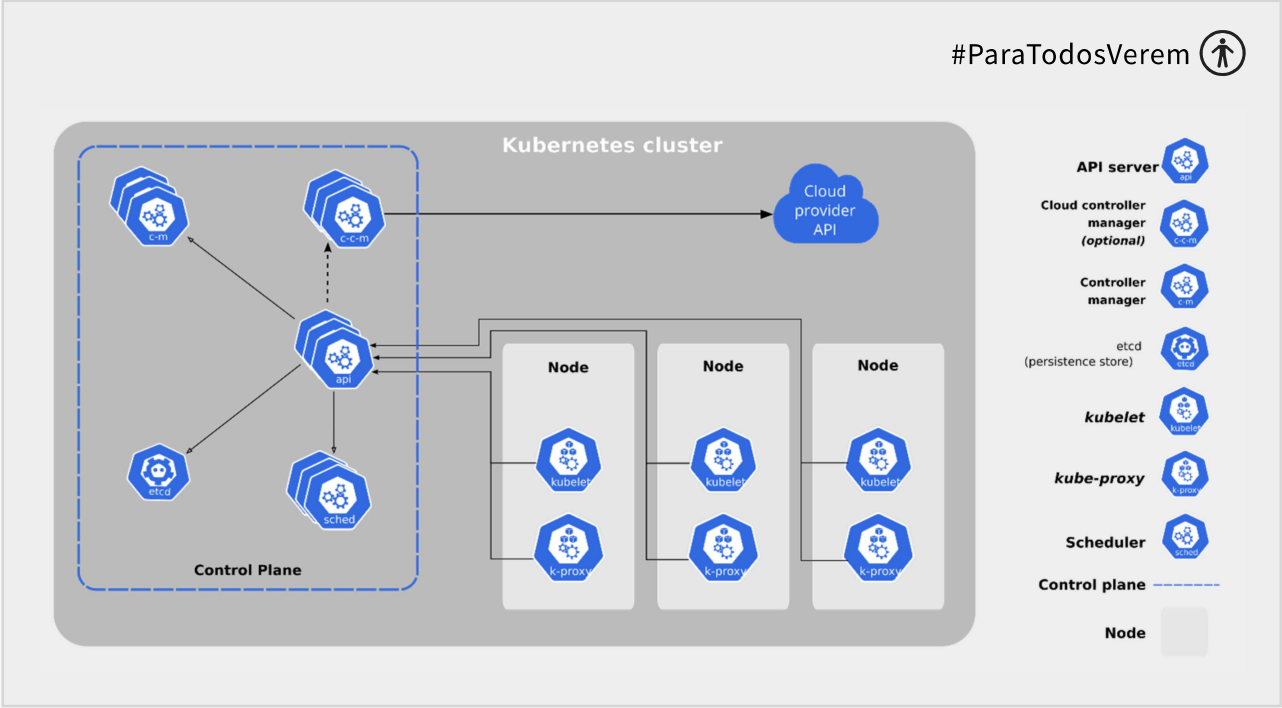
Para fins de estudo, é interessante termos algumas tabelas comparativas. Isso ajuda a entender melhor os prós e contras de algumas soluções – até porque às vezes os termos e aplicações se confundem, como o Docker Compose e o Kubernetes. Portanto, veja a tabela comparativa abaixo. Também está incluso o OpenShift, que é uma outra solução concorrente de mercado.

Solução	Docker	Docker Compose	Kubernetes	OpenShift
Para que serve?	Criação e gerenciamento de *containers* **individuais**.	Orquestração de **múltiplos *containers*** para aplicações.	Orquestração de *clusters* de *containers* em **grande escala**.	Orquestração de *clusters* com funcionalidades **corporativas**.

Como ele funciona?	*Containers* individuais.	Conjuntos de serviços em *containers*.	*Clusters* de *containers* distribuídos.	*Clusters* de *containers* com gestão empresarial.
Capacidade de escalabilidade	Limitada.	Limitada.	Alta, com autoescalonamento.	Alta, com autoescalonamento e gestão avançada.
Se der algum erro ele tenta consertar sozinho?	Não.	Não.	Ideal para ambientes de produção.	Ideal para ambientes corporativos de produção.
Como usamos no mercado?	Geralmente para desenvolvimento e teste.	Geralmente para ambientes de desenvolvimento.	Ideal para ambientes de produção.	Ideal para ambientes corporativos de produção.
Quando usamos?	Desenvolvimento e teste, ou aplicações simples.	Desenvolvimento e teste de aplicações compostas por múltiplos serviços.	Produção de larga escala, microsserviços.	Produção empresarial, microsserviços.

COMPONENTES DO KUBERNETES

Dito isso, como funcionam os componentes do Kubernetes? Primeiro, vamos ao conceito de *cluster*: toda vez que falarmos essa palavra, significa que estamos trabalhando com um **conjunto** de recursos (ex.: computadores, *containers*, *nodes* ou nós). No caso do Kubernetes, é um *cluster* composto por *nodes*, e cada *cluster* precisa ter, pelo menos, um *node worker*. Pense nisso como se fosse uma casa: toda casa precisa ter pelo menos uma porta. Ela pode ter mais de uma porta, mas **pelo menos uma** para funcionar.



Fonte: Github, 2020

Agora, o que significam esses componentes do diagrama? Vamos lá:

Nodes

Os *nodes* (às vezes traduzidos como nós, mas prefiro manter a nomenclatura original para evitar confusões) são as máquinas em que os *containers* realmente rodam. No exemplo do diagrama, temos três *nodes*, mas é possível termos qualquer número entre 1 e 5000. Cada *node* tem dois componentes principais:

a. kubelet: ele é um agente que roda em cada *node* e garante que os *containers* estejam em execução conforme definido pelas instruções do API Server (já falaremos sobre ele).

a. O kubelet monitora a saúde dos containers e reporta ao Control Plane.

b. kube-proxy: um proxy de rede que mantém regras de rede em cada node, permitindo a comunicação de rede entre os containers e o mundo externo.

a. É ele quem gerencia o tráfego de entrada e saída dos *nodes*.

Control plane

O *control plane* (à esquerda, no diagrama) é o cérebro do *cluster* Kubernetes. Ele gerencia todo o ciclo de vida dos *containers*. Os principais componentes do *control plane* são:

c. API *server* (api): esse é o ponto de entrada para todas as instruções do *cluster*. Veja que os outros quatro componentes do *control plane* e os *nodes* estão conectados a ele.

a. É ele quem recebe comandos do usuário ou de outros componentes e garante que essas instruções sejam executadas.

d. *Controller manager* (c-m): ele está no canto superior esquerdo do *control plane*. Ele gerencia os controladores que mantêm o estado desejado do *cluster*. É ele quem garante que tudo está funcionando como deveria.

a. Exemplo: se um *container* falhar, o *controller manager* é responsável por criar um novo *container*.

e. Cloud *controller manager* (c-c-m): ele está acima da *api* e ao lado do *c-m*. Como você deve imaginar, as empresas usam o Kubernetes dentro de uma infraestrutura em nuvem, como Amazon (AWS), Microsoft (Azure) e Google (GCP).

a. O *cloud controller manager* faz a ponte entre o Kubernetes e esses serviços de nuvem.

f. *Scheduler* (sched): ele está logo abaixo da *api*, no diagrama. É ele quem decide em qual *node* um novo *container* deve ser executado.

a. Ou seja: ele analisa a capacidade de recursos de cada *node* e escolhe o melhor local para executar cada *container*.

g. etcd: ele está no canto inferior esquerdo do diagrama. Ele funciona como um livro de registros que mantém tudo o que acontece no *cluster*.

a. Ele é responsável por guardar todas as informações de configuração e estado do *cluster*.

No dia a dia, é comum interagirmos com esses componentes usando o *kubectl*. O que acha de testarmos um pouco dessas interações? Para a videoaula abaixo, usaremos o Minikube, uma versão do Kubernetes preparada para funcionar diretamente no seu computador.

Brincando com os componentes do Kubernetes

Conclusão

Tratamos nesta semana sobre Kubernetes. De fato, compreendemos a arquitetura do Kubernetes, incluindo o *control plane* e os *nodes*, e como cada componente desempenha um papel importante na garantia de que as nossas aplicações de grande escala rodem. Quer dizer: o conceito do Docker funciona bem até certo ponto: quando temos milhões de usuários, é provável que queiramos algo mais robusto, como os Kubernetes. Perceba que todo o conteúdo que aprendemos sobre Docker se aplica aqui – não há como falarmos sobre Kubernetes sem falarmos sobre *containers*, por exemplo.

Também exploramos como o Kubernetes facilita a implantação, o escalonamento e a gestão de aplicações complexas, e vimos como ele automatiza muitas das tarefas que antes exigiam intervenção manual.

Referências

DOCKER DOCS. Play with Kubernetes. Disponível em: https://labs.play-with-k8s.com. Acesso em: 29 ago. 2024.

FREEMAN, E. **DevOps para leigos**. Rio de Janeiro: Editora Alta Books, 2021. E-book.

KIM, G.; BEHR, K.; SPAFFORD, G. **O projeto Fênix**. Rio de Janeiro: Editora Alta Books, 2020. E-book.

KIM, G.; HUMBLE, J.; DEBOIS, P.; WILLIS, J. **Manual de DevOps**. Rio de Janeiro: Editora Alta Books, 2018. E-book.