Introdução à Arquitetura do Kubernetes

Esta lição abordará a Arquitetura do Kubernetes. O que abordaremos aqui será suficiente para entender e raciocinar sobre os tópicos que aprenderemos mais tarde neste curso. A intenção é construir uma base sólida, em vez de ser uma revisão exaustiva. O próprio Kubernetes é um sistema distribuído. Ele introduz seu próprio dialeto no espaço de orquestração. Internalizar o vocabulário é uma parte importante do sucesso com o Kubernetes. Definiremos vários termos à medida que surgirem, mas saiba que também há um glossário do Kubernetes disponível na introdução ao caminho de aprendizado do Kubernetes, de modo que você tenha um único ponto de referência para os termos que precisa conhecer. Um glossário mais abrangente mantido pelo Kubernetes também está vinculado a partir daí.

Você também deve entender a arquitetura para ter uma compreensão básica de como os recursos funcionam nos bastidores. O cluster do Kubernetes é o mais alto nível de abstração para começar. Os clusters do Kubernetes são compostos por nós, e o termo cluster refere-se a todas as máquinas coletivamente e pode ser considerado como o sistema inteiro em execução.

As máquinas no cluster são referidas como nós. Um nó pode ser uma VM ou uma máquina física. Os nós são categorizados como nós de trabalho ou nós mestres. Cada nó de trabalho inclui software para executar contêineres gerenciados pelo plano de controle do Kubernetes, e o plano de controle é executado nos nós mestres. O plano de controle é um conjunto de APIs e softwares com os quais os usuários do Kubernetes interagem. Essas APIs e softwares são coletivamente chamados de componentes mestres.

O plano de controle agenda contêineres nos nós. Portanto, o termo agendamento não se refere ao tempo neste contexto. Pense nisso do ponto de vista do Kernel: o Kernel agenda processos na CPU de acordo com vários fatores. Certos processos precisam de mais ou menos recursos de computação ou podem ter diferentes regras de qualidade de serviço. Em última análise, o scheduler faz o seu melhor para garantir que cada contêiner seja executado. O agendamento, neste caso, refere-se ao processo de decisão de colocar contêineres nos nós de acordo com seus requisitos declarados de computação.

No Kubernetes, os contêineres são agrupados em Pods. Os Pods podem incluir um ou mais contêineres. Todos os contêineres em um Pod são executados no mesmo nó. E o Pod é, na verdade, o menor bloco de construção no Kubernetes. Abstrações mais complexas e úteis são construídas sobre os Pods. Serviços definem regras de rede para expor Pods a outros Pods ou expor Pods à internet. E o Kubernetes também usa implantações para gerenciar a configuração de implantação e mudanças nos Pods em execução, bem como o escalonamento horizontal. Estes são termos fundamentais que você precisa entender antes de podermos avançar.

Vamos elaborar sobre esses termos e introduzir mais termos à medida que avançamos ao longo do curso, mas não posso enfatizar demais a importância deles. Sugiro que você repita esta seção quantas vezes for necessário até que todas essas informações sejam assimiladas. Vamos recapitular o que aprendemos até agora. Kubernetes é uma ferramenta de orquestração. Um grupo de nós forma um cluster Kubernetes. Kubernetes executa contêineres em grupos chamados Pods. Os serviços do Kubernetes expõem Pods ao cluster, bem como à internet pública. E as implantações do Kubernetes controlam a implantação e a reversão dos Pods.

Como vimos, os componentes mestres fornecem o plano de controle do Kubernetes. A maneira de recuperar e modificar informações de estado no cluster é enviando uma solicitação ao servidor da API do Kubernetes, que é o componente mestre que atua como front-end para o plano de controle. Isso nos leva ao primeiro método de interação com o Kubernetes: comunicação direta via chamadas de API REST. É possível, mas incomum, precisar trabalhar diretamente com o servidor da API. Você pode precisar fazer isso se estiver usando uma linguagem de programação que não possui uma biblioteca cliente do Kubernetes.

As bibliotecas clientes são nosso segundo método de interação com o Kubernetes. Elas podem lidar com a tediosa tarefa de autenticação e gerenciamento de solicitações e respostas individuais da API REST. O Kubernetes mantém bibliotecas clientes oficiais para Go, Python, Java, .NET e JavaScript. Existem também muitas bibliotecas mantidas pela comunidade, caso não haja suporte oficial para sua linguagem de escolha. As bibliotecas clientes são uma ótima escolha para desenvolvedores escrevendo código para interagir com o Kubernetes.

O próximo método de interação com o Kubernetes é o mais comum e no qual focaremos neste curso: a ferramenta de linha de comando do Kubernetes chamada kubectl. Com o kubectl, você pode emitir comandos em um alto nível de abstração, com cada comando sendo traduzido na solicitação apropriada ao servidor da API. Com o kubectl, você pode acessar clusters localmente, bem como remotamente. Seu sucesso com o Kubernetes está diretamente relacionado à sua habilidade com o kubectl. Você pode realizar todo o seu trabalho diário usando o kubectl.

Portanto, é vital aprender este comando, pois ele gerencia todos os diferentes tipos de recursos do Kubernetes e fornece recursos de depuração e introspecção. Felizmente, o kubectl segue um padrão de design fácil de entender. Quando você aprende a gerenciar um recurso, aprende a gerenciar todos. Vamos introduzir alguns subcomandos comuns para ver como eles são e o que o kubectl pode fazer.

Começando com o `kubectl create`. O `kubectl create` cria um novo recurso do Kubernetes. Você pode criar vários recursos usando os subcomandos integrados do create, ou pode usar recursos especificados em um arquivo. Os arquivos são mais comumente no formato YAML e são chamados de manifestos.

O `kubectl delete` faz o oposto do create, pois exclui um recurso específico. Você pode fazer o mesmo com um arquivo, com recursos declarados dentro dele. O `kubectl get` retorna uma lista de todos os recursos para um tipo especificado. Por exemplo, `kubectl get pods` lista todos os pods no namespace atual. O `kubectl describe` imprime informações detalhadas sobre um recurso específico ou uma lista de recursos. Por exemplo, `kubectl describe pod server` fornece informações detalhadas sobre o pod chamado server. O `kubectl logs` imprime os logs do contêiner para um pod específico ou um contêiner específico dentro de um pod com múltiplos contêineres.