Aluno: Lucas Piacenti Graciano - 2020.1907.029-1
Orientador: Dionisio Machado Leite Filho
Componente Curricular: Atividade Orientada a Ensino

Avaliações de CI/CD: Um estudo sobre Kubernetes

O Kubernetes, líder entre os orquestradores de containers, é essencial para a automação, escalabilidade e gerenciamento eficaz de aplicações em containers, sendo considerado o padrão de mercado. Ele proporciona alta disponibilidade, auto-recuperação e simplificações operacionais, contribuindo para a padronização e efemeridade de ambientes em nuvem.

A evolução dos containers, impulsionada pelo Docker, revolucionou a hospedagem de aplicações. Conceitos como Dockerfiles, Union Filesystem e Registros de Containers simplificaram o empacotamento e compartilhamento de imagens. A analogia "Gado x Pet" destaca a natureza efêmera dos containers, enquanto o Kubernetes surge como uma solução aberta, mantida pela Cloud Native Computing Foundation, para gerenciar de forma eficiente clusters de containers em ambientes empresariais de computação em nuvem.

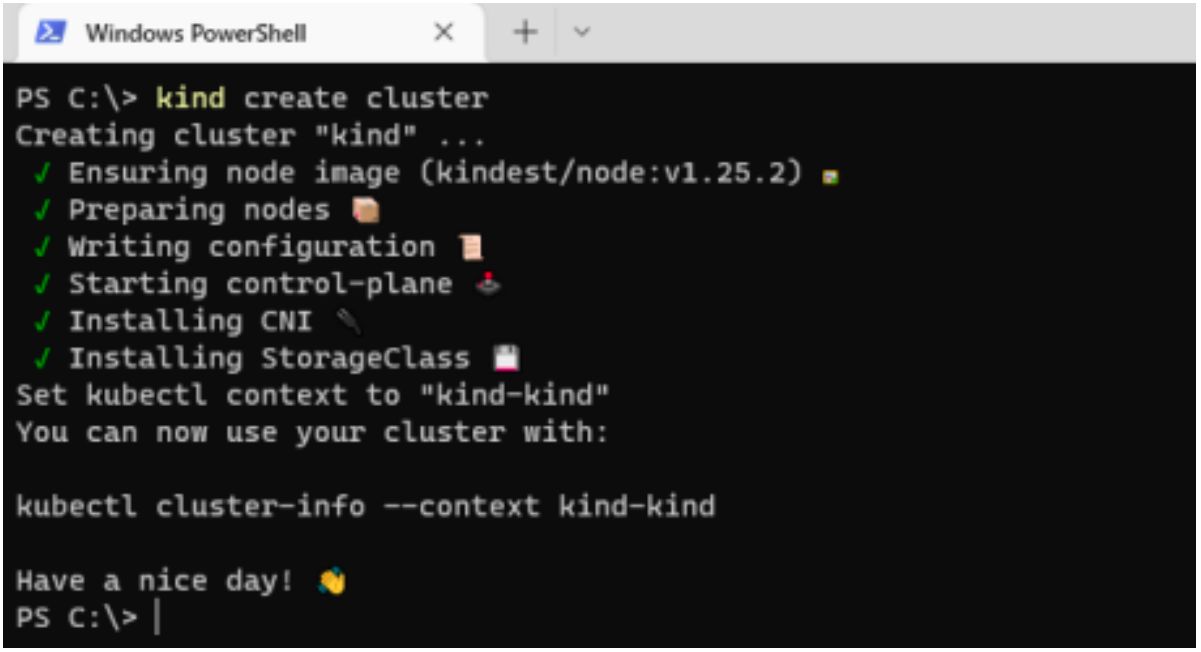
Componentes

O Kubernetes é composto por diferentes camadas de componentes que compartilham um estado centralizado no banco de dados chave-valor ETCD. Essas camadas incluem o Control Plane, responsável pelas decisões administrativas; o Node Plane, que abriga componentes em todos os servidores do cluster para gerenciar containers e redes; e os Addons, que adicionam funcionalidades extras ao cluster. O Kube-apiserver é a porta de entrada para o Control Plane, enquanto o ETCD armazena todos os dados do Kubernetes, priorizando a consistência sobre a disponibilidade. Outros componentes, como o Kube-scheduler, Kube-controller-manager, Kubelet e Kube-proxy, desempenham funções cruciais no gerenciamento e execução de containers no cluster. O Kubectl é uma ferramenta fundamental para interagir com o Kubernetes, enquanto o capítulo aborda ainda o Cloud Controller, responsável pela interação entre o Kubernetes e a nuvem onde está hospedado.

Criando um Cluster com kind

Para criar e gerenciar um cluster Kubernetes local para desenvolvimento, o documento propõe o uso da ferramenta Kind, que utiliza o Docker. Antes de começar, são listados os requisitos, incluindo Powershell 5, Docker for Windows 4.1.1, Kind 0.11.1 e Git 2.33.0.windows.1.

Os passos foram executados em um ambiente Windows 11 com Powershell 5. O Docker utilizado é a versão 4.1.1, e o Kind é baixado da última versão disponível em seu repositório. Após adicionar o Kind ao Path do Windows, é possível criar clusters de desenvolvimento com o comando **kind create cluster**. O cluster é executado em um container Docker, e sua existência pode ser verificada com **docker ps**. Para deletar o cluster, utiliza-se o comando **kind delete cluster**.



Namespaces

Para criar e gerenciar Namespaces em um cluster Kubernetes, é essencial entender que Namespaces são divisões lógicas dentro do cluster, permitindo isolar recursos entre equipes, aplicações ou ambientes.

A criação de Namespaces pode ser feita com o comando **kubectl create ns** seguido do nome do Namespace ou por meio de um arquivo YAML. Por exemplo, um Namespace chamado production pode ser criado usando o arquivo YAML namespace.yaml. Recursos podem ser associados a um Namespace usando a opção -n ou adicionando namespace: production nos arquivos YAML.

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: production
```

Nem todos os recursos são vinculados a Namespaces. Classes de armazenamento e servidores do cluster (Nodes) são exemplos de recursos não vinculados à Namespaces.

Em resumo, Namespaces são utilizados para organizar o cluster de diversas maneiras, seja dividindo-o entre equipes, aplicações ou ambientes de produção e desenvolvimento. A flexibilidade do Kubernetes permite adaptar o cluster conforme as necessidades da equipe ou empresa, possibilitando a criação de ambientes distintos dentro de um único cluster.

POD

Um Pod é o menor recurso que podemos criar, consistindo em um grupo de um ou mais containers que compartilham recursos de rede e armazenamento. Podemos criar Pods manualmente usando comandos ou descrevendo-os em YAML.

No exemplo apresentado, foi criado um Pod chamado static-web com um container baseado na imagem nginx:latest, definindo uma porta TCP 80 e aplicando rótulos (labels) ao recurso. É importante notar que os Pods são recursos "Namespaced", ou seja, devem estar relacionados a um Namespace específico.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: static-web
  namespace: default
  labels:
    role: myrole
spec:
  containers:
  - name: web
    image: nginx:latest
    ports:
    - name: web
      containerPort: 80
```

Em resumo, este capítulo forneceu uma introdução prática à criação, execução e monitoramento de Pods no Kubernetes, preparando o terreno para a exploração de recursos mais avançados nos capítulos subsequentes.

ReplicaSets

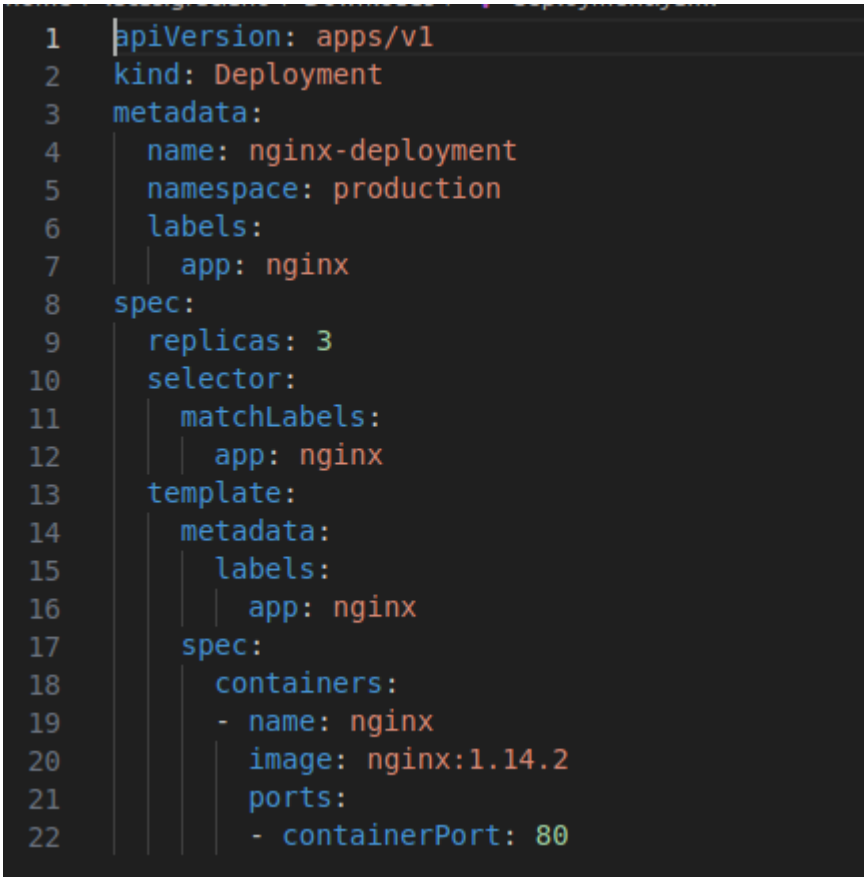
Os ReplicaSets no Kubernetes são recursos que garantem que um número específico de réplicas (cópias) de um Pod esteja sempre em execução. Se o número de réplicas cair abaixo do especificado, o ReplicaSet inicia automaticamente novos Pods para restaurar o número desejado. Essa abordagem contribui para a disponibilidade e confiabilidade dos aplicativos, garantindo que a quantidade desejada de instâncias de um Pod esteja sempre em operação, mesmo diante de falhas ou interrupções.

Os ReplicaSets são frequentemente usados em conjunto com Pods para escalar aplicativos, garantindo que a quantidade desejada de instâncias esteja sempre ativa para gerenciar a carga e distribuir serviços de forma eficiente. Eles são uma evolução do conceito anterior de "Replication Controllers" no Kubernetes.

Deployment

No contexto de Kubernetes, um Deployment é um recurso que gerencia aplicativos em contêineres. Ele facilita a definição declarativa do estado desejado do aplicativo, orquestra ReplicaSets para garantir o número desejado de Pods e oferece recursos como atualizações, rollbacks automáticos e auto reparo em caso de falhas. Os Deployments são uma abordagem declarativa para definir o estado desejado do aplicativo, e você os configura usando arquivos YAML. Eles simplificam a operação e a escalabilidade de aplicativos em ambientes Kubernetes.

Sendo assim, foi realizado um experimento utilizando o arquivo YAML a seguir, o qual descreve um Deployment para criar um ReplicaSet com três Pods usando a imagem do Nginx. Destacamos a importância dos rótulos (labels) para a identificação e gerenciamento dos recursos. Demonstramos a criação de Deployments através de arquivos YAML e sua aplicação com o comando **kubectl apply**.



```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  namespace: production
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

Em suma, Deployments são importantes para o gerenciamento eficiente de aplicações no Kubernetes, proporcionando a capacidade de atualizações sem interrupções e a facilidade de desfazer alterações quando necessário.

Services

Services são ferramentas essenciais para garantir a comunicação estável entre Pods no cluster. Eles fornecem IPs fixos para os Pods, facilitando a comunicação interna e externa. Existem diferentes tipos de Services, como ClusterIP, NodePort e LoadBalancer, cada um com finalidades específicas.

Para criar o Service, foi criado um Deployment para os Pods, utilizando seletores baseados em Labels.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: whoami-deployment
  namespace: production
  labels:
    app: whoami
spec:
  replicas: 2
  selector:
    matchLabels:
      app: whoami
  template:
    metadata:
      labels:
        app: whoami
    spec:
      containers:
      - name: whoami
        image: traefik/whoami:v1.8.0
        ports:
        - containerPort: 80
```

Em seguida, criamos o Service, escolhendo o tipo apropriado, como ClusterIP para comunicação interna ou NodePort/LoadBalancer para acesso externo. O Service foi testado usando comandos como **kubectl port-forward** ou **kubectl exec**.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: whoami-service
  namespace: production
spec:
  selector:
    app: whoami
  type: ClusterIP
  ports:
  - name: http
    port: 80
    targetPort: 80
```

Os Services também têm nomes DNS, facilitando o acesso interno. Por fim, é notável a importância dos Services no Kubernetes para facilitar a comunicação entre os componentes da aplicação e fornecer acesso controlado a usuários externos.

Ingress

O Ingress atua como um Prox HTTP para facilitar o acesso externo às aplicações dentro do cluster. Enquanto os Services resolvem muitas necessidades de acesso, os Ingresses oferecem recursos adicionais, como redirecionamento de tráfego baseado em paths e customizações HTTP.

Para criar um Ingress, é necessário primeiro instalar um Ingress Controller no cluster. O Nginx Ingress é uma escolha comum e pode ser instalado usando ferramentas como Kind. Após a instalação, criamos recursos para a aplicação que serão acessados através do Ingress.

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: minimal-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - host: localhost
    http:
      paths:
      - path: /testpath
        pathType: Prefix
        backend:
          service:
            name: testpath
            port:
              number: 80
```

Demonstramos um exemplo de Ingress YAML que redireciona tráfego do path "/testpath" para os Pods do Deployment "whoami-deployment" através do Service "whoami-service" criado anteriormente.

Neste sentido podemos concluir que o Ingress Controllers é utilizado para facilitar o acesso unificado às aplicações no Kubernetes, além de oferecer funções avançadas, como manipulação de tráfego HTTP e suporte a certificados digitais.