Descomplicando o Kubernetes - Expert Mode

DAY-3

Inicio da aula do Day-3

O que iremos ver hoje?

Durante o dia de hoje nós iremos aprender sobre um objeto muito importante no Kubernetes, o Deployment. Nós iremos ver todos os detalhes para que possamos ter uma visão completa sobre o que é um Deployment e como ele funciona. Agora que já sabemos tudo sobre como criar um Pod, acho que já podemos colocar um pouco mais de complexidade no nosso cenário, não é mesmo? Bora lá!

O que é um Deployment?

No Kubernetes um Deployment é um objeto que representa uma aplicação. Ele é responsável por gerenciar os Pods que compõem uma aplicação. Um Deployment é uma abstração que nos permite atualizar os Pods e também fazer o rollback para uma versão anterior caso algo dê errado.

Quando criamos um Deployment é possível definir o número de réplicas que queremos que ele tenha. O Deployment irá garantir que o número de Pods que ele está gerenciando seja o mesmo que o número de réplicas definido. Se um Pod morrer, o Deployment irá criar um novo Pod para substituí-lo. Isso ajuda demais na disponibilidade da aplicação.

Um Deployment é declarativo, ou seja, nós definimos o estado desejado e o Deployment irá fazer o que for necessário para que o estado atual seja igual ao estado desejado.

Quando criamos um Deployment, nós automaticamente estamos criando um ReplicaSet. O ReplicaSet é um objeto que é responsável por gerenciar os Pods para o Deployment, já o Deployment é responsável por gerenciar os ReplicaSets.Como eu disse, um ReplicaSet é um objeto que é criado automaticamente pelo

Deployment, mas nós podemos criar um ReplicaSet manualmente caso necessário. Nós vamos falar mais sobre isso no próximo dia, hoje o foco é o Deployment.

Como criar um Deployment?

Para criar um Deployment nós precisamos de um arquivo YAML. Vamos criar um arquivo chamado deployment.yaml e vamos adicionar o seguinte conteúdo:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: nginx-deployment
  name: nginx-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx-deployment
  strategy: {}
  template:
    metadata:
      labels:
        app: nginx-deployment
    spec:
      containers:
      - image: nginx
        name: nginx
        resources:
          limits:
            cpu: "0.5"
            memory: 256Mi
          requests:
            cpu: 0.25
            memory: 128Mi
```


O que cada parte do arquivo significa?

```
apiVersion: apps/v1
kind: Deployment
```

Aqui nós estamos definindo que o tipo do objeto que estamos criando é um Deployment e a versão da API que estamos utilizando é a apps/v1.

```
metadata:
  labels:
    app: nginx-deployment
  name: nginx-deployment
```

Aqui nós estamos definindo o nome do Deployment, que é nginx-deployment e também estamos definindo as labels que serão adicionadas ao Deployment. As labels são utilizadas para identificar os objetos no Kubernetes.

```
spec:
  replicas: 3
```

Aqui nós estamos definindo o número de réplicas que o Deployment irá ter. Nesse caso nós estamos definindo que o Deployment irá ter 3 réplicas.

```
selector:
  matchLabels:
    app: nginx-deployment
```

Aqui nós estamos definindo o seletor que será utilizado para identificar os Pods que o Deployment irá gerenciar. Nesse caso nós estamos definindo que o Deployment irá gerenciar os Pods que possuírem a label app: nginx-deployment.

```
strategy: {}
```

Aqui nós estamos definindo a estratégia que será utilizada para atualizar os Pods. Nesse caso nós estamos deixando a estratégia padrão, que é a estratégia Rolling Update, ou seja, o Deployment irá atualizar os Pods um por um. Iremos entrar em detalhes sobre as estratégias mais para frente.

```
template:
  metadata:
    labels:
      app: nginx-deployment
  spec:
    containers:
    - image: nginx
      name: nginx
      resources:
        limits:
          cpu: "0.5"
          memory: 256Mi
```

```
        requests:
          cpu: 0.25
          memory: 128Mi
```

Aqui nós estamos definindo o template que será utilizado para criar os Pods. Nesse caso nós estamos definindo que o template irá utilizar a imagem nginx e que o nome do container será nginx. Também estamos definindo os limites de CPU e memória que poderão ser utilizados pelo container. Essa definição é identica ao que vimos no Day-2, quando criamos um Pod.

Simples demais, não? Agora vamos aplicar o Deployment.

Como aplicar o Deployment?

Para aplicar o Deployment nós precisamos executar o seguinte comando:

```
kubectl apply -f deployment.yaml
```

Como verificar se o Deployment foi criado?

Para verificar se o Deployment foi criado nós precisamos executar o seguinte comando:

```
kubectl get deployments -l app=nginx-deployment
```

O resultado será o seguinte:

```
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-78cd4b8fd-r4zk8    1/1     Running   0          5s
```

Como verificar os Pods que o Deployment está gerenciando?

Para verificar os Pods que o Deployment está gerenciando nós precisamos executar o seguinte comando:

```
kubectl get pods -l app=nginx
```

O resultado será o seguinte:

```
NAME                               READY   STATUS    RESTARTS   AGE
nginx-deployment-78cd4b8fd-8b8mm   1/1     Running   0          44s
nginx-deployment-78cd4b8fd-kn4v8   1/1     Running   0          44s
nginx-deployment-78cd4b8fd-xqn5g   1/1     Running   0          44s
```

Isso acontece porque o seletor do Deployment é app: nginx e as labels dos Pods que o Deployment está gerenciando são app: nginx, lembra que definimos isso no template do Deployment?

Como verificar o ReplicaSet que o Deployment está gerenciando?

Caso eu queria listar os ReplicaSets que o Deployment está gerenciando eu posso executar o seguinte comando:

```
kubectl get replicasets -l app=nginx
```

O resultado será o seguinte:

```
NAME                         DESIRED   CURRENT   READY   AGE
nginx-deployment-78cd4b8fd   3         3         3       88s
```

Lembrando novamente que iremos entrar em detalhes sobre os ReplicaSets mais para frente.

Como verificar os detalhes do Deployment?

Para verificar os detalhes do Deployment nós precisamos executar o seguinte comando:

```
kubectl describe deployment nginx-deployment
```

Na saída do comando `kubectl describe deployment nginx-deployment` nós vamos encontrar informações importantes sobre o Deployment, como por exemplo:

- O nome do Deployment
- O Namespace que o Deployment está
- Os labels que o Deployment possui
- A quantidade de réplicas que o Deployment possui
- O selector que o Deployment utiliza para identificar os Pods que ele irá gerenciar
- Limites de CPU e memória que o Deployment irá utilizar
- O pod template que o Deployment irá utilizar para criar os Pods
- A estratégia que o Deployment irá utilizar para atualizar os Pods

- O ReplicaSet que o Deployment está gerenciando
- Os eventos que aconteceram no Deployment

Vamos dar uma olhada em uma parte da saída do comando `kubectl describe deployment nginx-deployment`:

```
Name:                   nginx-deployment
Namespace:              default
CreationTimestamp:      Fri, 20 Jan 2023 19:05:29 +0100
Labels:                 app=nginx-deployment
Annotations:            deployment.kubernetes.io/revision: 1
Selector:               app=nginx-deployment
Replicas:               3 desired | 3 updated | 3 total | 3 available | 0
unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=nginx-deployment
  Containers:
   nginx:
    Image:      nginx
    Port:       <none>
    Host Port:  <none>
    Limits:
      cpu:     500m
      memory:  256Mi
    Requests:
      cpu:        250m
      memory:     128Mi
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
  Progressing    True    NewReplicaSetAvailable
OldReplicaSets:  <none>
NewReplicaSet:   nginx-deployment-78cd4b8fd (3/3 replicas created)
Events:
  Type    Reason             Age    From                   Message
  ----    ------             ----   ----                   -------
  Normal  ScalingReplicaSet  3m13s  deployment-controller  Scaled up replica
set nginx-deployment-78cd4b8fd to 3
```


Como atualizar o Deployment?

Vamos imaginar que agora precisamos passar uma versão especifica da imagem do Nginx para o Deployment, para isso nós precisamos alterar o arquivo deployment.yaml e alterar a versão da imagem para nginx:1.16.0, por exemplo.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: nginx-deployment
  name: nginx-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx-deployment
  strategy: {}
  template:
    metadata:
      labels:
        app: nginx-deployment
    spec:
      containers:
      - image: nginx:1.16.0
        name: nginx
        resources:
          limits:
            cpu: "0.5"
            memory: 256Mi
          requests:
            cpu: 0.25
            memory: 128Mi
```


Agora que já alteramos a versão da imagem do Nginx, nós precisamos aplicar as alterações no Deployment, para isso nós precisamos executar o seguinte comando:

```
kubectl apply -f deployment.yaml
```

O resultado será o seguinte:

```
deployment.apps/nginx-deployment configured
```

Vamos ver os detalhes do Deployment para verificar se a versão da imagem foi alterada:

```
kubectl describe deployment nginx-deployment
```

Na saída do comando, podemos ver a linha onde está a versão da imagem do Nginx:

```
Containers:
   nginx:
```

```
Image:      nginx:1.16.0
Port:       <none>
Host Port:  <none>
Limits:
  cpu:     500m
  memory:  256Mi
Requests:
  cpu:        250m
  memory:     128Mi
Environment:  <none>
Mounts:       <none>
```

E qual é a estratégia de atualização padrão do Deployment?

Quando criamos o nosso Deployment, nós não informamos nenhuma estratégia de atualização, por isso o Kubernetes utiliza a estratégia de atualização padrão, que é a estratégia RollingUpdate.

A estratégia RollingUpdate é a estratégia de atualização padrão do Kubernetes, ela é utilizada para atualizar os Pods de um Deployment de forma gradual, ou seja, ela atualiza um Pod por vez, ou um grupo de Pods por vez.

Nós podemos definir como será a atualização dos Pods, por exemplo, podemos definir a quantidade máxima de Pods que podem ficar indisponíveis durante a atualização, ou podemos definir a quantidade máxima de Pods que podem ser criados durante a atualização.

Vamos entender um pouco melhor como isso funciona no próximo tópico.

As estratégias de atualização do Deployment

O Kubernetes possui 2 estratégias de atualização para os Deployments:

- RollingUpdate
- Recreate

Vamos entender um pouco melhor cada uma dessas estratégias.

Estratégia RollingUpdate

A estratégia RollingUpdate é a estratégia de atualização padrão do Kubernetes, ela é utilizada para atualizar os Pods de um Deployment de forma gradual, ou seja, ela atualiza um Pod por vez, ou um grupo de Pods por vez.

Nós podemos definir como será a atualização dos Pods, por exemplo, podemos definir a quantidade máxima de Pods que podem ficar indisponíveis durante a

atualização, ou podemos definir a quantidade máxima de Pods que podem ser criados durante a atualização.

Vamos também aumentar a quantidade de réplicas do Deployment para 10, para que possamos ter um pouco mais de Pods para atualizar.

E para que possamos testar a estratégia RollingUpdate, vamos alterar a versão da imagem do Nginx para 1.15.0.

Para que possamos definir essas configurações, nós precisamos alterar o arquivo deployment.yaml e adicionar as seguintes configurações:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: nginx-deployment
  name: nginx-deployment
spec:
  replicas: 10
  selector:
    matchLabels:
      app: nginx-deployment
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 2
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: nginx-deployment
    spec:
      containers:
      - image: nginx:1.15.0
        name: nginx
        resources:
          limits:
            cpu: "0.5"
            memory: 256Mi
          requests:
            cpu: 0.25
            memory: 128Mi
```


O que nós fizemos foi adicionar as seguintes configurações:

```
strategy:
  type: RollingUpdate
  rollingUpdate:
    maxSurge: 1
```

```
    maxUnavailable: 2
```

Onde:

- maxSurge: define a quantidade máxima de Pods que podem ser criados a mais durante a atualização, ou seja, durante o processo de atualização, nós podemos ter 1 Pod a mais do que o número de Pods definidos no Deployment. Isso é útil pois agiliza o processo de atualização, pois o Kubernetes não precisa esperar que um Pod seja atualizado para criar um novo Pod.
- maxUnavailable: define a quantidade máxima de Pods que podem ficar indisponíveis durante a atualização, ou seja, durante o processo de atualização, nós podemos ter 1 Pod indisponível por vez. Isso é útil pois garante que o serviço não fique indisponível durante a atualização.
- type: define o tipo de estratégia de atualização que será utilizada, no nosso caso, nós estamos utilizando a estratégia RollingUpdate.

Agora que já alteramos o arquivo deployment.yaml, nós precisamos aplicar as alterações no Deployment, para isso nós precisamos executar o seguinte comando:

```
kubectl apply -f deployment.yaml
```

O resultado será o seguinte:

```
deployment.apps/nginx-deployment configured
```

Vamos verificar se as alterações foram aplicadas no Deployment:

```
kubectl describe deployment nginx-deployment
```

Na saída do comando, podemos ver que as linhas onde estão as configurações de atualização do Deployment foram alteradas:

```
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  2 max unavailable, 1 max surge
```

Com essa configuração estamos falando para o Kubernetes que ele pode criar até 1 Pod a mais durante a atualização, e que ele pode ter até 2 Pods indisponíveis durante a atualização, ou seja, ele vai atualizar de 2 em 2 Pods.

Um comando muito útil para acompanhar o processo de atualização dos Pods é o comando:

```
kubectl rollout status deployment nginx-deployment
```

O comando `rollout status` é utilizado para acompanhar o processo de atualização de um Deployment, ReplicaSet, DaemonSet, StatefulSet, Job e CronJob. O comando `rollout status` é muito útil pois ele nos informa se o processo de atualização está em andamento, se ele foi concluído com sucesso ou se ele falhou.

Vamos executar o comando `rollout status` para acompanhar o processo de atualização do Deployment:

```
kubectl rollout status deployment nginx-deployment
```

O resultado será o seguinte:

```
Waiting for deployment "nginx-deployment" rollout to finish: 9 of 10 updated
replicas are available...
deployment "nginx-deployment" successfully rolled out
```

Como podemos ver, o processo de atualização foi concluído com sucesso.

Vamos verificar se os Pods foram atualizados:

```
kubectl get pods -l app=nginx-deployment -o yaml
```

Na saída do comando, podemos ver que os Pods foram atualizados:

```
...
  - image: nginx:1.15.0
...
```

Podemos também verificar a versão da imagem do Nginx no Pod:

```
kubectl exec -it nginx-deployment-7b7b9c7c9d-4j2xg -- nginx -v
```

O resultado será o seguinte:

```
nginx version: nginx/1.15.0
```

O comando `nginx -v` é utilizado para verificar a versão do Nginx.

Nem sempre essa é a melhor estratégia de atualização, pois você pode encontrar por aí aplicações que não suportam duas versões do mesmo serviço rodando ao mesmo tempo, por exemplo.

Tudo funcionando como queriamos, agora vamos testar a estratégia Recreate.

Estratégia Recreate

A estratégia Recreate é uma estratégia de atualização que irá remover todos os Pods do Deployment e criar novos Pods com a nova versão da imagem. A parte boa é que o deploy acontecerá rapidamente, porém o ponto negativo é que o serviço ficará indisponível durante o processo de atualização.

Vamos alterar o arquivo deployment.yaml para utilizar a estratégia Recreate:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: nginx-deployment
  name: nginx-deployment
spec:
  replicas: 10
  selector:
    matchLabels:
      app: nginx-deployment
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: nginx-deployment
    spec:
      containers:
      - image: nginx:1.15.0
        name: nginx
        resources:
          limits:
            cpu: "0.5"
            memory: 256Mi
          requests:
            cpu: 0.25
            memory: 128Mi
```


Perceba que agora somente temos a configuração `type: Recreate`. O Recreate não possui configurações de atualização, ou seja, não é possível definir o número máximo de Pods indisponíveis durante a atualização, afinal o Recreate irá remover todos os Pods do Deployment e criar novos Pods.

Agora que já alteramos o arquivo deployment.yaml, nós precisamos aplicar as alterações no Deployment, para isso nós precisamos executar o seguinte comando:

```
kubectl apply -f deployment.yaml
```

O resultado será o seguinte:

```
deployment.apps/nginx-deployment configured
```

Vamos verificar se as alterações foram aplicadas no Deployment:

```
kubectl describe deployment nginx-deployment
```

Na saída do comando, podemos ver que as linhas onde estão as configurações de atualização do Deployment foram alteradas:

```
StrategyType:           Recreate
```

Vamos novamente alterar a versão da imagem do Nginx para 1.16.0 no arquivo deployment.yaml:

```
image: nginx:1.16.0
```

Agora que já alteramos o arquivo deployment.yaml, nós precisamos aplicar as alterações no Deployment, para isso nós precisamos executar o seguinte comando:

```
kubectl apply -f deployment.yaml
```

O resultado será o seguinte:

```
deployment.apps/nginx-deployment configured
```

Vamos verificar os Pods do Deployment:

```
kubectl get pods -l app=nginx-deployment
```

O resultado será o seguinte:

```
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-7d9bcc6bc9-24c2j   0/1     Pending   0          0s
nginx-deployment-7d9bcc6bc9-5r69s   0/1     Pending   0          0s
nginx-deployment-7d9bcc6bc9-78mc9   0/1     Pending   0          0s
nginx-deployment-7d9bcc6bc9-7pb2v   0/1     Pending   0          0s
nginx-deployment-7d9bcc6bc9-gvtvl   0/1     Pending   0          0s
nginx-deployment-7d9bcc6bc9-kb9st   0/1     Pending   0          0s
nginx-deployment-7d9bcc6bc9-m69bm   0/1     Pending   0          0s
nginx-deployment-7d9bcc6bc9-qvppt   0/1     Pending   0          0s
nginx-deployment-7d9bcc6bc9-sqn6q   0/1     Pending   0          0s
nginx-deployment-7d9bcc6bc9-zthn4   0/1     Pending   0          0s
```

Podemos ver que os Pods estão sendo criados novamente, porém com a nova versão da imagem do Nginx.

Vamos verificar a versão da imagem do Nginx no Pod:

```
kubectl exec -it nginx-deployment-7d9bcc6bc9-24c2j -- nginx -v
```

O resultado será o seguinte:

```
nginx version: nginx/1.16.0
```

Pronto, agora nós temos a versão 1.16.0 do Nginx rodando no nosso cluster e já entendemos como funciona a estratégia Recreate.

Fazendo o rollback de uma atualização

Agora que já entendemos como funciona as estratégias Rolling Update e Recreate, vamos entender como fazer o rollback de uma atualização.

Vamos alterar a versão da imagem do Nginx para 1.15.0 no arquivo deployment.yaml:

```
image: nginx:1.15.0
```

Agora que já alteramos o arquivo deployment.yaml, nós precisamos aplicar as alterações no Deployment, para isso nós precisamos executar o seguinte comando:

```
kubectl apply -f deployment.yaml
```

O resultado será o seguinte:

```
deployment.apps/nginx-deployment configured
```

Vamos verificar os Pods do Deployment:

```
kubectl get pods -l app=nginx-deployment
```

Vamos verificar a versão da imagem do Nginx no Pod:

```
kubectl exec -it nginx-deployment-7d9bcc6bc9-24c2j -- nginx -v
```

O resultado será o seguinte:

```
nginx version: nginx/1.15.0
```

Vamos imaginar que nós queremos fazer o rollback para a versão 1.16.0 do Nginx, para isso nós precisamos executar o seguinte comando:

```
kubectl rollout undo deployment nginx-deployment
```

O resultado será o seguinte:

```
deployment.apps/nginx-deployment rolled back
```

O que estamos fazendo nesse momento é falar para o Kubernetes que queremos fazer o rollback para a versão anterior do Deployment.

Vamos verificar os Pods do Deployment:

```
kubectl get pods -l app=nginx-deployment
```

Vamos verificar a versão da imagem do Nginx no Pod:

```
kubectl exec -it nginx-deployment-7d9bcc6bc9-24c2j -- nginx -v
```

O resultado será o seguinte:

```
nginx version: nginx/1.16.0
```

Pronto, agora nós temos a versão 1.16.0 do Nginx rodando no nosso cluster e já entendemos como fazer o rollback de uma atualização. Mas como nós visualizamos o histórico de atualizações do Deployment?

Essa é fácil, nós precisamos executar o seguinte comando:

```
kubectl rollout history deployment nginx-deployment
```

Com isso ele vai nos mostrar o histórico de atualizações do Deployment:

```
deployment.apps/nginx-deployment
REVISION  CHANGE-CAUSE
1         <none>
2         <none>
```

Na saída do comando podemos ver que temos duas revisões do Deployment, a revisão 1 e a revisão 2.

Vamos verificar o histórico de atualizações da revisão 1:

```
kubectl rollout history deployment nginx-deployment --revision=1
```

O resultado será o seguinte:

```
deployment.apps/nginx-deployment with revision #1
Pod Template:
  Labels:   app=nginx-deployment
      pod-template-hash=c549ff78
  Containers:
   nginx:
    Image:  nginx:1.16.0
    Port:   <none>
    Host Port:    <none>
    Limits:
      cpu:  500m
```

```
      memory:     256Mi
    Requests:
      cpu:  250m
      memory:     128Mi
    Environment:  <none>
    Mounts: <none>
  Volumes:  <none>
```

Vamos verificar o histórico de atualizações da revisão 2:

```
kubectl rollout history deployment nginx-deployment --revision=2
```

O resultado será o seguinte:

```
deployment.apps/nginx-deployment with revision #2
Pod Template:
  Labels:   app=nginx-deployment
      pod-template-hash=7d9bcc6bc9
  Containers:
   nginx:
    Image:  nginx:1.15.0
    Port:   <none>
    Host Port:    <none>
    Limits:
      cpu:  500m
      memory:     256Mi
    Requests:
      cpu:  250m
      memory:     128Mi
    Environment:  <none>
    Mounts: <none>
  Volumes:  <none>
```


Ou seja, como podemos notar, a revisão 1 é a versão 1.16.0 do Nginx e a revisão 2 é a versão 1.15.0 do Nginx.

Se você quiser fazer o rollback para a revisão 1, basta executar o seguinte comando:

```
kubectl rollout undo deployment nginx-deployment --to-revision=1
```

O resultado será o seguinte:

```
deployment.apps/nginx-deployment rolled back
```

Pronto, agora nós temos a versão 1.16.0 do Nginx rodando no nosso cluster e já entendemos como fazer o rollback de uma atualização, simples né?

O comando `kubectl rollout` é muito útil para nós, pois ele nos ajuda a visualizar o histórico de atualizações do Deployment, fazer o rollback de uma atualização e muito mais.

Nós já vimos algumas opções do comando `kubectl rollout` como por exemplo o `kubectl rollout history`, `kubectl rollout undo` e `kubectl rollout status`, mas existem outras opções que nós podemos utilizar, vamos ver algumas delas:

```
kubectl rollout pause deployment nginx-deployment
```

O comando `kubectl rollout pause` é utilizado para pausar o Deployment, ou seja, ele vai pausar o Deployment e não vai permitir que ele faça nenhuma atualização.

```
kubectl rollout resume deployment nginx-deployment
```

O comando `kubectl rollout resume` é utilizado para despausar o Deployment, ou seja, ele vai despausar o Deployment e vai permitir que ele faça nenhuma atualização.

```
kubectl rollout restart deployment nginx-deployment
```

O comando `kubectl rollout restart` é utilizado para reiniciar o Deployment, ou seja, ele vai reiniciar o Deployment recriando os Pods.

```
kubectl rollout status deployment nginx-deployment
```

O comando `kubectl rollout status` é utilizado para verificar o status do Deployment, ou seja, ele vai verificar o status do rollout do Deployment.

```
kubectl rollout undo deployment nginx-deployment
```

O comando `kubectl rollout undo` é utilizado para fazer o rollback de uma atualização, ou seja, ele vai fazer o rollback de uma atualização para a revisão anterior.

```
kubectl rollout history deployment nginx-deployment
```

O comando `kubectl rollout history` é utilizado para visualizar o histórico de atualizações do Deployment.

```
kubectl rollout history deployment nginx-deployment --revision=1
```

Lembrando que podemos utilizar o comando `kubectl rollout` em Deployments, StatefulSets e DaemonSets.

Removendo um Deployment

Para remover um Deployment nós precisamos executar o seguinte comando:

```
kubectl delete deployment nginx-deployment
```

O resultado será o seguinte:

```
deployment.apps "nginx-deployment" deleted
```

Caso queira remover o Deployment utilizando o manifesto, basta executar o seguinte comando:

```
kubectl delete -f deployment.yaml
```

O resultado será o seguinte:

```
deployment.apps "nginx-deployment" deleted
```

Pronto, agora nós removemos o Deployment do nosso cluster.

Conclusão

Durante o dia de hoje, nós aprendemos o que é um Deployment, como criar um Deployment, como atualizar um Deployment, como fazer o rollback de uma atualização, como remover um Deployment e muito mais. Com isso nós já temos uma excelente base para começar a trabalhar com Deployments no Kubernetes.

Ainda falaremos muito sobre os Deployments e conheceremos muitas outras opções que eles nos oferecem, mas por enquanto é isso, espero que tenham gostado e aprendido bastante com o conteúdo de hoje.

Na Day-4 nós vamos falar sobre os ReplicaSets e DaemonSets, então fiquem ligados que a Day-4 está chegando.

#VAIIII