



CASO 1

O MISTÉRIO DO POD EM CRASHLOOPBACKOFF

LINUXTIPS



O CASO: O POD QUE NÃO QUERIA VIVER

Este é o rito de passagem para qualquer um que começa a usar o Kubernetes. Você passou horas criando o `deployment.yaml` perfeito para a sua aplicação. Você aplica o manifesto com `kubectl apply -f deployment.yaml` e o Kubernetes te responde com um amigável `deployment.apps/minha-app created`. Sucesso! Para confirmar, você vai listar os Pods, as unidades básicas onde seus containers vivem.

```
$ kubectl get pods
NAME           READY   STATUS      RESTARTS   AGE
minha-app-6bcf89c9d-z8k21  0/1   CrashLoopBackOff 5       2m
```

E aí está ele. O status que assombra os pesadelos de todo engenheiro: `CrashLoopBackOff`. Sim ou não, turma, que o coração até para por um segundo? O status `READY` está `0/1`, o que significa que o container dentro do Pod não está pronto. E a coluna `RESTARTS` não para de aumentar. O Kubernetes está tentando, bravamente, iniciar seu container, mas ele falha, morre, e o Kubernetes, como um bom maestro, tenta de novo, em um loop infinito de falha e sofrimento.

*O que está acontecendo?
Por que o nosso Pod se
recusa a viver?*



A INVESTIGAÇÃO: A AUTÓPSIA DO POD

Um **CrashLoopBackOff** não é o problema, é o sintoma.

O Kubernetes está nos dizendo:

"Eu estou fazendo o meu trabalho, que é tentar manter este Pod rodando, mas a aplicação dentro dele continua morrendo. O problema não é comigo, é com ele!".

Nossa missão é descobrir por que a aplicação está morrendo.



PISTA 1: O DOSSIÊ DO CRIME (KUBECTL DESCRIBE)

Nossa primeira e mais importante ferramenta de interrogatório é o **kubectl describe**. Ele nos dá um dossiê completo, a "autópsia" do Pod.

```
$ kubectl describe pod minha-app-6bcf89c9d-z8k21
```



O resultado é longo, mas duas seções são cruciais:

A SEÇÃO STATE

Perto do topo, você verá informações sobre o container.

A SEÇÃO EVENTS

Role até o final do **describe**. A seção de eventos é a linha do tempo do que aconteceu com o Pod.



PISTA 2: AS ÚLTIMAS PALAVRAS DA VÍTIMA (KUBECTL LOGS)

Agora que sabemos que a aplicação está falhando com um **Exit Code 1**, precisamos ouvir as últimas palavras dela. Precisamos ver os logs. Mas se o container está em loop de crash, como pegar o log?

```
$ kubectl logs minha-app-6bcf89c9d-z8k21
```



Se você rodar esse comando, pode ser que receba um log vazio ou um erro. Por quê? Porque você está pedindo os logs do container atual, que pode já ter crashado e ainda não produziu output. O truque de mestre aqui é usar a flag **--previous**.

```
$ kubectl logs minha-app-6bcf89c9d-z8k21  
--previous  
/app/server.js:25  
const db_password = process.env.DB_PASSWORD;  
  
TypeError: Cannot read properties of undefined  
(reading 'DB_PASSWORD')  
at Object. anonymous (/app/server.js:25:42)  
...  
...
```

Cai está!

A arma do crime, com a impressão digital do culpado. O log do container anterior (**--previous**) nos mostra exatamente o erro: um `TypeError` no `server.js`. A aplicação está tentando ler uma variável de ambiente chamada `DB_PASSWORD`, não a encontrou (`undefined`) e crashou.

Facil né?

PISTA 3: O SUICÍDIO POR SONDA DE VITALIDADE (LIVENESS PROBE)

Se os logs não mostrassem nada?

E se o **Exit Code** fosse **137**? Poderia ser um problema de memória, como vimos no capítulo do Docker. Você precisaria checar os resources (requests e limits) no seu manifesto YAML.

Mas há um culpado mais sutil: a Liveness Probe. É um mecanismo onde o Kubernetes periodicamente "cutuca" sua aplicação para ver se ela está viva e bem. Essa cutucada pode ser uma requisição HTTP, uma conexão TCP ou um comando. Se a aplicação não responder corretamente, o Kubernetes assume que ela travou e a reinicia. Se a Liveness Probe estiver mal configurada (ex: apontando para um endpoint errado, com um timeout muito curto), ela mesma pode ser a causa do CrashLoopBackOff, matando um container saudável.

Você pode ver a configuração da sonda no `kubectl describe pod` e testar o endpoint ou comando manualmente de dentro do Pod (`kubectl exec`) para ver se ele está funcionando como esperado.



A REVELAÇÃO: A MENTE POR TRÁS DO TROUBLESHOOTING

OBSERVAÇÃO DO SINTOMA:

O Pod está em CrashLoopBackOff.

DIAGNÓSTICO DE ALTO NÍVEL:

Usamos kubectl describe pod para obter o Exit Code e os Events. Isso nos deu a pista inicial (um erro na aplicação, Exit Code 1).

INVESTIGAÇÃO PROFUNDA:

Usamos kubectl logs --previous para ler os logs da última execução do container, encontrando a causa raiz exata: uma variável de ambiente faltando.

HIPÓTESES ALTERNATIVAS:

Se os logs fossem inconclusivos, investigaríamos outras causas, como limites de memória (conferindo resources no YAML) ou falhas na Liveness Probe (conferindo a configuração da sonda no descreve).

