

# Orquestração de Containers

Resolução de problemas  
no Kubernetes

# Tópicos abordados

- Resolução de problemas
- *Troubleshooting* de aplicações
- *Troubleshooting* do *control plane*
- Software de apoio

# Resolução de problemas

Invariavelmente, problemas irão acontecer no *cluster*, seja em aplicações ou no *control plane*

O processo de resolução é conhecido como *troubleshooting*

É fundamental conhecer os recursos disponíveis para responder de forma rápida e precisa

Essa habilidade é útil não apenas num contexto de certificação, mas também no dia-a-dia de operações

# Esferas de resolução de problemas

Aplicações

*Control plane*

Software e recursos de apoio

# Troubleshooting de aplicações

A resolução de problemas em aplicações passa primeiro pela *triagem*

Qual é o recurso afetado?  
*Pods, replication controllers* ou serviços?

# Solucionando problemas em *Pods*

Primeiramente, deve-se determinar o estado corrente do *pod*

Os containers do *pod* estão em estado *Running*? Há reinícios recentes?

```
kubectl describe pods ${POD_NAME}
```

# Pods em estado *Pending*

O estado *Pending* indica que um *pod* não pôde ser agendado em um *node*

As mensagens nesse sentido são emitidas pelo *kube-scheduler*. Além das já vistas neste curso, temos também:

Falta de recursos disponíveis  
(CPU ou RAM)

O *kube-scheduler* encontra-se indisponível

Mapeamento de *hostPort*  
indisponível no *node*

Não há *node* agendável  
devido a *taints* ou afinidades

# Pods em estado Pending

```
root@s2-master-1:~# k apply -f manifests/deploy-nginx-limits.yaml
deployment.apps/deploy-nginx created
root@s2-master-1:~#
root@s2-master-1:~# k get pod
```

NAME	READY	STATUS	RESTARTS	AGE
deploy-nginx-788cb948bb-9d6gq	1/1	Running	0	9s

```
root@s2-master-1:~# k describe pod deploy-nginx-788cb948bb-9d6gq
Name:                deploy-nginx-788cb948bb-9d6gq
Namespace:           color
```

## Events:

Type	Reason	Age	From	Message
Normal	Scheduled	27s	default-scheduler	Successfully assigned color/deploy-nginx-788cb948bb-9d6gq to s2-node-1
Normal	Pulling	26s	kubelet	Pulling image "nginx"
Normal	Pulled	23s	kubelet	Successfully pulled image "nginx" in 2.135774998s (2.135781388s including waiting)
Normal	Created	23s	kubelet	Created container nginx
Normal	Started	23s	kubelet	Started container nginx



# Pods em estado Pending

```
root@s2-master-1:~# k scale deployment deploy-nginx --replicas=3
deployment.apps/deploy-nginx scaled
root@s2-master-1:~#
root@s2-master-1:~# k get pods
```

NAME	READY	STATUS	RESTARTS	AGE
deploy-nginx-788cb948bb-8vmkf	0/1	Pending	0	21s
deploy-nginx-788cb948bb-9d6gq	1/1	Running	0	3m22s
deploy-nginx-788cb948bb-h5986	0/1	Pending	0	21s

```
root@s2-master-1:~# k describe pod deploy-nginx-788cb948bb-8vmkf
Name:          deploy-nginx-788cb948bb-8vmkf
Namespace:     color
```

```
Events:
  Type      Reason             Age   From              Message
  ----      -
Warning    FailedScheduling   85s   default-scheduler 0/2 nodes are available: 1 Insufficient cpu, 1 node(s) had untolera
aint {node-role.kubernetes.io/control-plane: }. preemption: 0/2 nodes are available: 1 No preemption victims found for incoming pod, 1 Preemption is not helpful for scheduling..
```

# Pods em estado *Waiting*

O estado *Waiting* indica que um *pod* foi agendado, mas não consegue executar no *node*-alvo

Frequentemente, isso significa que as imagens dos containers do *pod* não puderam ser baixadas. Possíveis motivos:

O nome da imagem está escrito incorretamente

A imagem ou *tag* não encontra-se no repositório

Não há permissões para obter a imagem

Tente fazer um pull manual via *docker pull* e buscar por erros

# Pods em estado Waiting

```
1 apiVersion: v1
2 kind: Pod
3 metadata:
4   labels:
5     run: nginx
6   name: pod-nginx
7 spec:
8   containers:
9     - image: nginx:alpine
10      name: pod-nginx
```

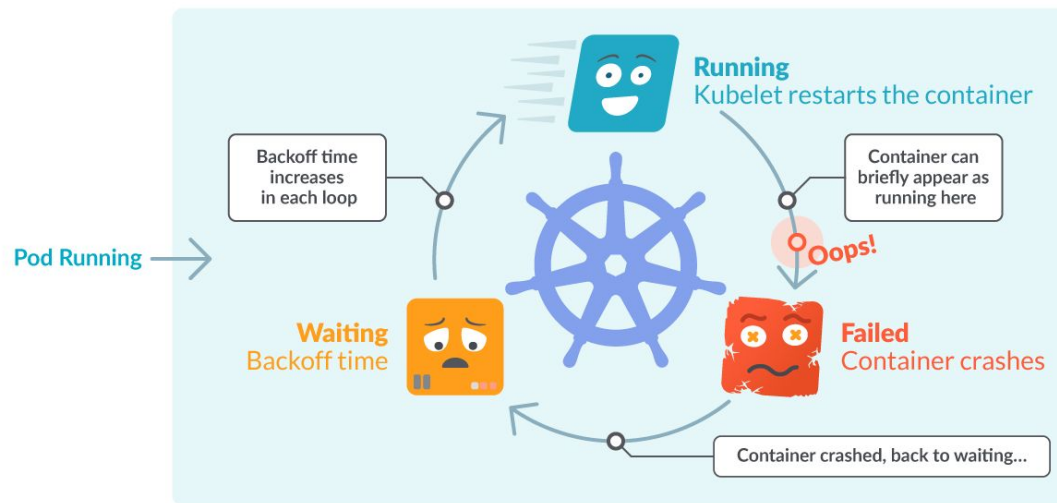
```
root@s2-master-1:~# k apply -f manifests/pod-nginx.yaml
pod/pod-nginx created
root@s2-master-1:~#
root@s2-master-1:~# k get pod
NAME          READY   STATUS      RESTARTS   AGE
pod-nginx     0/1     ErrImagePull 0           15s
```

```
Events:
  Type     Reason      Age           From          Message
  ----     -
  Normal   Scheduled   88s          default-scheduler Successfully assigned color/pod-nginx to s2-node-1
  Normal   Pulling     42s (x3 over 87s) kubelet        Pulling image "nginx:alpine"
  Warning   Failed      38s (x3 over 84s) kubelet        Failed to pull image "nginx:alpine": rpc error: code = Unknown de
sc = Error response from daemon: manifest for nginx:alpine not found: manifest unknown: manifest unknown
  Warning   Failed      38s (x3 over 84s) kubelet        Error: ErrImagePull
  Normal   BackOff     9s (x4 over 83s) kubelet        Back-off pulling image "nginx:alpine"
  Warning   Failed      9s (x4 over 83s) kubelet        Error: ImagePullBackOff
```

# Pods em estado *CrashLoopBackOff*

```
$ kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
flask-7996469c47-d7z12	1/1	Running	1	77d
flask-7996469c47-tdr2n	1/1	Running	0	77d
nginx-5796d5bc7c-2jdr5	0/1	CrashLoopBackOff	2	1m
nginx-5796d5bc7c-xs16p	0/1	CrashLoopBackOff	2	1m



# Pods em estado *CrashLoopBackOff*

```
1 |apiVersion: v1
2 kind: Pod
3 metadata:
4   labels:
5     run: nginx
6   name: pod-nginx
7 spec:
8   containers:
9   - image: nginx:alpine
10     name: pod-nginx
11     command: ['sleep 5']
```

```
root@s2-master-1:~# k get pod
```

NAME	READY	STATUS	RESTARTS	AGE
pod-nginx	0/1	CrashLoopBackOff	1 (24s ago)	27s

```
Warning BackOff      5s (x7 over 109s)  kubelet
color(222d3482-547a-4944-9703-9978c6647b3f)
```

```
Back-off restarting failed container pod-nginx in pod pod-nginx_c
```

# *Pods* falhando ou com problemas diversos

Caso um pod esteja corretamente agendado e operacional, mas tendo outros problemas, deve-se explorar outras possibilidades

# Analizando *logs* de *pods*

Um bom primeiro passo é analisar os logs dos containers afetados

```
kubectl logs ${POD_NAME} ${CONTAINER_NAME}
```

Caso tenha ocorrido um *crash* recente, pode-se acessar os *logs* antigos com:

```
kubectl logs --previous ${POD_NAME} ${CONTAINER_NAME}
```

# Realizando *debugging* com *kubectl exec*

Caso a imagem de container possua utilitários de *debugging*, o *kubectl exec* é uma boa opção:

```
kubectl exec ${POD_NAME} -c ${CONTAINER_NAME} -- ${CMD} ${ARG1} ${ARG2} ... ${ARGN}
```

Considere também o uso de um *shell* interativo via:

```
kubectl exec -it cassandra -- sh
```



# Realizando *debugging* com containers efêmeros

A partir da versão 1.25 do Kubernetes é possível utilizar containers efêmeros para *debugging*

Isso é ideal num cenário de *crash* de containers ou imagens minimalistas (distroless images)

Os containers efêmeros compartilham o mesmo *pod* dos recursos afetados

É possível, portanto, realizar *troubleshooting* de forma interativa e conveniente

# Realizando *debugging* com containers efêmeros

```
kubectl run ephemeral-demo --image=registry.k8s.io/pause:3.1 --restart=Never
```

```
kubectl exec -it ephemeral-demo -- sh
```

OCI runtime exec failed: exec failed: container\_linux.go:346: starting container process caused "exec: \"sh\": executable file not found in \$PATH": unknown

```
kubectl debug -it ephemeral-demo --image=busybox:1.28
```

```
> k describe pod ephemeral-demo
Name:          ephemeral-demo
Namespace:     color
```

```
Containers:
  ephemeral-demo:
    Container ID:   docker://e952c3ec59d7634775e84791d
    Image:          registry.k8s.io/pause:3.1
```

```
Ephemeral Containers:
  debugger-k4cw5:
    Container ID:   docker://c50686093da010c9d3916776
    Image:          busybox:1.28
```

# Realizando o *troubleshooting* via shell do NODE

Se você não tiver acesso ao console do node, é possível também com `kubectl debug` criar um POD interativo que se liga ao shell do NODE

Como no exemplo abaixo:

```
kubectl debug node/mynode -it --image=ubuntu
```

O root filesystem do NODE (/) será montado no diretório /host do POD de debug

# Solucionando problemas em serviços

Serviços funcionam como um balanceador de carga entre diversos *pods*

Comece verificando se os *endpoints* do mesmo estão identificados e corretos

```
kubectl get endpoints ${SERVICE_NAME}
```

# Solucionando problemas em serviços

Vamos executar um deployment de uma aplicação chamada hostnames que escuta na porta 9376 e responde com o nome do seu hostname

```
kubectl create deployment hostnames --image=registry.k8s.io/serve_hostname
```

```
kubectl scale deployment hostnames --replicas=3
```

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

```
kubectl run -it curl --image=curlimages/curl --restart=Never -- sh
```

```
$ curl -s endpoint_ip:9376
```

```
$ curl -s hostnames
```

O service hostnames existe?

# Solucionando problemas em serviços

Vamos executar um deployment de uma aplicação chamada hostnames que escuta na porta 9376 e responde com o nome do seu hostname

```
kubectl expose deployment hostnames --port=8080 --target-port=9376
```

```
kubectl get svc -l app=hostnames
```

```
kubectl run -it curl --image=curlimages/curl --restart=Never -- sh
```

```
curl -s hostnames:8080
```

E agora, foi?

# Verificando seletores

Caso os *endpoints* não correspondam ao esperado, verifique os seletores aplicados

Cheque tanto o arquivo de definição do serviço, quanto os *Pods* com o mesmo *label* aplicado

```
...  
spec:  
  - selector:  
    name: nginx  
    type: frontend
```

```
kubectl get pods --selector=name=nginx,type=frontend
```

# Verificando o estado do *kube-proxy*

Se tudo o mais estiver  
correto, o método de  
exposição dos serviços fica  
sob suspeita

Verifique se os pods do  
*kube-proxy* estão ativos, bem  
como seus *logs*



# Verificando o estado do *kube-proxy*

```
root@s2-master-1:~# k -n kube-system get pods | grep kube-proxy
kube-proxy-6glz7          1/1      Running          12 (21h ago)      40d
kube-proxy-8bvv6          1/1      Running          11 (21h ago)      40d
root@s2-master-1:~# k -n kube-system logs kube-proxy-6glz7
I0821 02:10:19.039059      1 node.go:141] Successfully retrieved node IP: 192.168.68.20
I0821 02:10:19.041332      1 server_others.go:110] "Detected node IP" address="192.168.68.20"
I0821 02:10:19.041568      1 server_others.go:554] "Using iptables proxy"
I0821 02:10:20.504168      1 server_others.go:192] "Using iptables Proxier"
I0821 02:10:20.507850      1 server_others.go:199] "kube-proxy running in dual-stack mode" ipFamily=IPv4
I0821 02:10:20.507972      1 server_others.go:200] "Creating dualStackProxier for iptables"
I0821 02:10:20.508047      1 server_others.go:484] "Detect-local-mode set to ClusterCIDR, but no IPv6 cluster CIDR defined, defaulting
to no-op detect-local for IPv6"
```

```
> k stern -n kube-system ds/kube-proxy
+ kube-proxy-6glz7 > kube-proxy
+ kube-proxy-8bvv6 > kube-proxy
kube-proxy-6glz7 kube-proxy I0821 02:10:19.039059      1 node.go:141] Successfully retrieved node IP: 192.168.68.20
kube-proxy-6glz7 kube-proxy I0821 02:10:19.041332      1 server_others.go:110] "Detected node IP" address="192.168.68.20"
kube-proxy-6glz7 kube-proxy I0821 02:10:19.041568      1 server_others.go:554] "Using iptables proxy"
kube-proxy-6glz7 kube-proxy I0821 02:10:20.504168      1 server_others.go:192] "Using iptables Proxier"
kube-proxy-6glz7 kube-proxy I0821 02:10:20.507850      1 server_others.go:199] "kube-proxy running in dual-stack mode" ipFamily=IPv4
kube-proxy-6glz7 kube-proxy I0821 02:10:20.507972      1 server_others.go:200] "Creating dualStackProxier for iptables"
kube-proxy-6glz7 kube-proxy I0821 02:10:20.508047      1 server_others.go:484] "Detect-local-mode set to ClusterCIDR, but no IPv6 clus
ter CIDR defined, defaulting to no-op detect-local for IPv6"
```

# Troubleshooting do control plane

Estando afastadas as possibilidades de problemas na aplicação, passamos ao *cluster* em si

```
kubectl get nodes
```

Antes de mais nada, verifique os *nodes* e se configuração geral está correta

```
kubectl cluster-info dump
```

# Visualizando *logs* relevantes

Caso os componentes do *control plane* sejam *pods* estáticos, utilize o *kubectl logs*. Do contrário, verifique:

## Master

- `/var/log/kube-apiserver.log` - API Server, responsible for serving the API
- `/var/log/kube-scheduler.log` - Scheduler, responsible for making scheduling decisions
- `/var/log/kube-controller-manager.log` - Controller that manages replication controllers

## Worker Nodes

- `/var/log/kubelet.log` - Kubelet, responsible for running containers on the node
- `/var/log/kube-proxy.log` - Kube Proxy, responsible for service load balancing

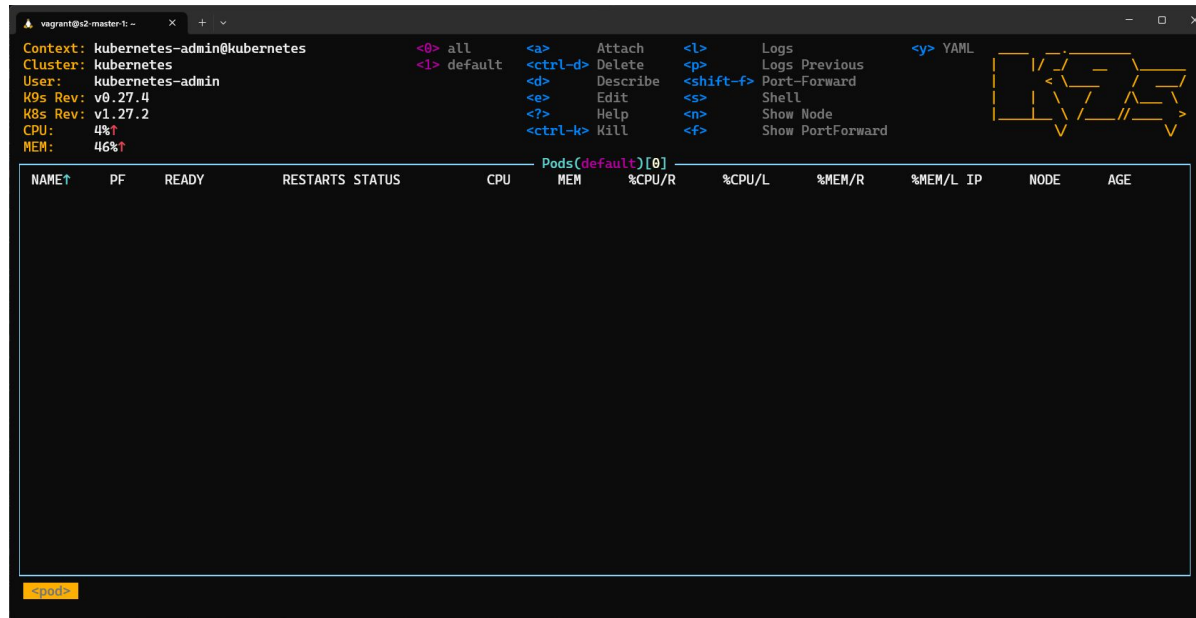
# Troubleshooting do cluster

<https://kubernetes.io/docs/tasks/debug/debug-cluster>

# E agora...

Vamos ver algumas soluções e ferramentas que podem auxiliar a tarefa de *troubleshooting*

# Visibilidade em linha de comando: k9s

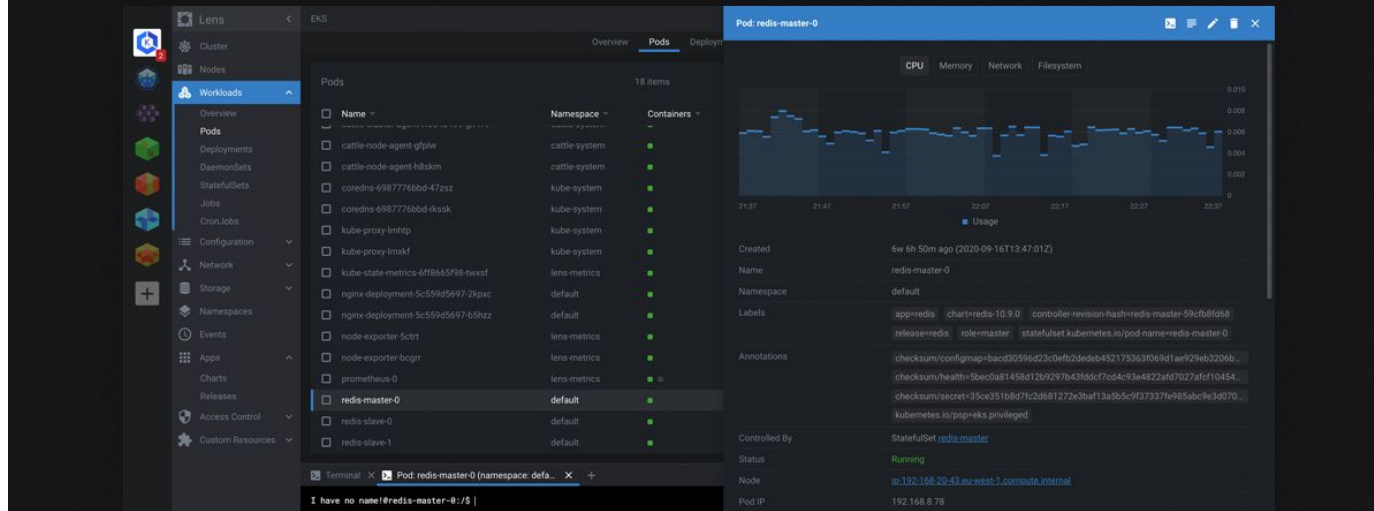


The screenshot shows the k9s terminal interface. At the top left, it displays system information: Context: kubernetes-admin@kubernetes, Cluster: kubernetes, User: kubernetes-admin, K9s Rev: v0.27.4, K8s Rev: v1.27.2, CPU: 4%↑, and MEM: 46%↑. A keyboard shortcuts menu is visible on the right, listing actions like Attach, Delete, Describe, Edit, Help, Kill, Logs, Logs Previous, Port-Forward, Shell, Show Node, and Show PortForward. Below this, a table titled 'Pods(default) [0]' is shown with columns: NAME↑, PF, READY, RESTARTS, STATUS, CPU, MEM, %CPU/R, %CPU/L, %MEM/R, %MEM/L, IP, NODE, and AGE. The table is currently empty. At the bottom left, there is a small orange button labeled '<pod>'. The terminal window title is 'vagrant@k2-master-1: ~'.

<https://github.com/derailed/k9s>

# Kubernetes - LENS

## LENS // THE KUBERNETES IDE



The screenshot displays the Kubernetes Lens IDE interface. On the left, a sidebar contains navigation icons for Cluster, Nodes, Workloads, Overview, Pods, Deployments, DaemonSets, StatefulSets, Jobs, CronJobs, Configuration, Network, Storage, Namespaces, Events, Apps, Charts, Releases, Access Control, and Custom Resources. The main panel is divided into two sections. The top section, titled 'Pods', shows a list of 18 items with columns for Name, Namespace, and Containers. The bottom section, titled 'Pod: redis-master-0', provides detailed information about the selected pod, including its creation time, name, namespace, labels, annotations, and status. A terminal window at the bottom shows the command 'I have no name@redis-master-0:/#'.

Name	Namespace	Containers
redis-master-0	default	redis
redis-slave-0	default	redis
redis-slave-1	default	redis

**Pod: redis-master-0**

Created: 6w 6h 50m ago (2020-09-16T13:47:01Z)

Name: redis-master-0

Namespace: default

Labels: app=redis, chart=redis-10.9.0, controller=revision-hash=redis-master-59cfb8f68, release=redis, role=master, statefulset.kubernetes.io/pod-name=redis-master-0

Annotations: checksum/configmap-bacd30596d23cdefb2dedeb452175363f04d1a929eb3206b..., checksum/health=5becba81458d12b9297b43fddc7fcd4c93e4822a4d7027afcf10454..., checksum/secret=35ce351b8d7fc2d681272x3baf13a5e5cf937337e985abc9c3d070..., kubernetes.io/psp=eks.privileged

Controlled By: StatefulSet/redis-master

Status: Running

Node: ip-192-168-20-43.eu-west-3.compute.internal

Pod IP: 192.168.8.78

<https://k8slens.dev/>

# Introduzindo o caos no dia-a-dia: *kube-monkey*

<https://github.com/asobti/kube-monkey>



# Visibilidade com Grafana/Loki



<https://grafana.com/docs/loki/latest/installation/helm>

# Tarefa 9

As atividades práticas desta sessão podem ser obtidas em formato HTML via:

<https://bit.ly/ads19-tarefas-s9>



ESCOLA  
SUPERIOR  
DE REDES

# Resolução de problemas no Kubernetes