

Evidencie executando em linha de comando:

- Rode com o curl e exiba o resultado

```
ribeiro@crosshairs:~/teste/kubernetes$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
node-teste-hello-world-deployment-6969875877-6bhjh   1/1     Running   0          10m
ribeiro@crosshairs:~/teste/kubernetes$ kubectl port-forward node-teste-hello-world-deployment-6969875877-6bhjh 3000:3000
Forwarding from 127.0.0.1:3000 -> 3000
Forwarding from [::1]:3000 -> 3000
Handling connection for 3000
Handling connection for 3000
Handling connection for 3000
]
```

```
ribeiro@crosshairs:~/teste/kubernetes$ curl -v http://localhost:3000
* Trying 127.0.0.1:3000...
* Connected to localhost (127.0.0.1) port 3000 (#0)
> GET / HTTP/1.1
> Host: localhost:3000
> User-Agent: curl/7.81.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< X-Powered-By: Express
< Accept-Ranges: bytes
< Cache-Control: public, max-age=0
< Last-Modified: Tue, 27 Aug 2024 19:33:42 GMT
< ETag: W/"14e-19195546876"
< Content-Type: text/html; charset=UTF-8
< Content-Length: 334
< Date: Tue, 27 Aug 2024 21:12:29 GMT
< Connection: keep-alive
< Keep-Alive: timeout=5
<
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">

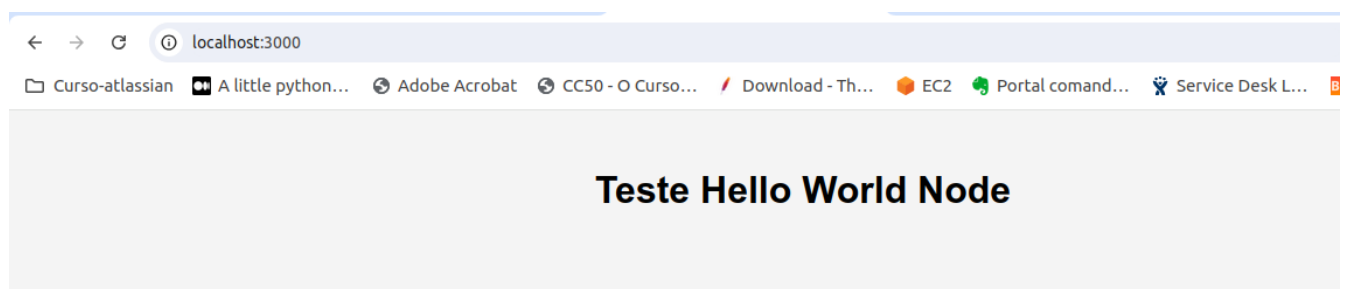
```

- Como foi executado o build da imagem

```
ribeiro@crosshairs:~/teste$ sudo docker build -t node-teste-hello-world .
[+] Building 17.2s (4/9)
=> [internal] load build definition from Dockerfile
=> == transferring Dockerfile: 188B
=> [internal] load metadata for docker.io/library/node:18
=> [internal] load .dockerignore
=> == transferring context: 2B
=> [1/5] FROM docker.io/library/node:18@sha256:a7ff16657263663c1e92ba3060cd8ba0e77329a0a4cb3c27bbbbe90c6e20bd87
=> == resolve docker.io/library/node:18@sha256:a7ff16657263663c1e92ba3060cd8ba0e77329a0a4cb3c27bbbbe90c6e20bd87
=> == sha256:903681d87777d28dc56866a07a2774c3fd5bf65fd734b24c9d0ecd9a13c9f636 48.23MB / 49.55MB
=> == sha256:cc72b58258b36bc48a43113d08f5a1e2957f12118fcd2ffade90f4c5e0c1 2.49kB / 2.49kB
=> == sha256:43f87525ada8df5b873def96c54f0fbf2f3b823638fb03cc19a57add84228fe0 6.55kB / 6.55kB
=> == sha256:3cbb86a28c2f6b3c1e8e8c6dcfba369e1ea050cf8daf69be789e0fe2105982b 24.05MB / 24.05MB
=> == sha256:6ed93aa58a52c9abc1ee472f1ac74b73d3adcccc2c30744498fd5f14f3f5d22c 3.15MB / 64.14MB
=> == sha256:a7ff16657263663c1e92ba3060cd8ba0e77329a0a4cb3c27bbbbe90c6e20bd87 6.41kB / 6.41kB
=> == sha256:787c78da3830beed988d34c7ee891f98d828510ce5478ca18a4933d655191bf 211.24MB / 211.24MB
=> == sha256:a22bad3cc9773638be9a43d84179b40cccaf4e9486ad5e39af9d01457ae263d4 0B / 3.32kB
=> [internal] load build context
=> == transferring context: 1.42kB
```

- Como validar que a imagem funcionou no Docker

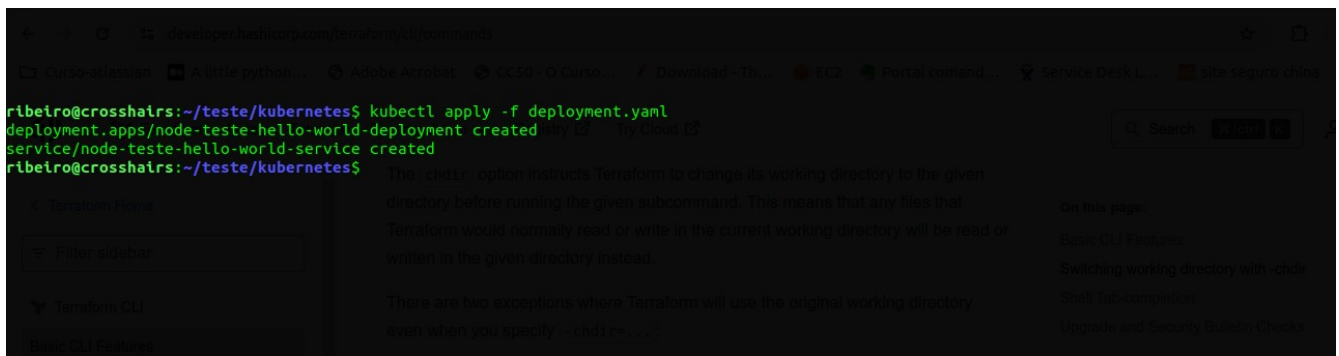
```
ribeiro@crosshairs:~/teste$ sudo docker run -p 3000:3000 node-teste-hello-world
Servidor rodando na porta 3000
```



```
ribeiro@crosshairs:~$ curl -v localhost:3000
* Trying 127.0.0.1:3000...
* Connected to localhost (127.0.0.1) port 3000 (#0)
> GET / HTTP/1.1
> Host: localhost:3000
> User-Agent: curl/7.81.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< X-Powered-By: Express
< Accept-Ranges: bytes
< Cache-Control: public, max-age=0
< Last-Modified: Tue, 27 Aug 2024 19:33:42 GMT
< ETag: W/"14e-19195546876"
< Content-Type: text/html; charset=UTF-8
< Content-Length: 334
< Date: Tue, 27 Aug 2024 19:59:47 GMT
< Connection: keep-alive
< Keep-Alive: timeout=5
<
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title> teste Hello World </title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <h1> Teste Hello World Node </h1>
  <script src="script.js"></script>
</body>
</html>

* Connection #0 to host localhost left intact
ribeiro@crosshairs:~$
```

- Como fez o deploy no Kubernetes



- Como obter que os recursos da aplicação no kubernetes foram aplicados

```
ribeiro@crosshairs:~/teste/kubernetes$ kubectl describe pod node-teste-hello-world-deployment-6969875877-6bhjh
Name: node-teste-hello-world-deployment-6969875877-6bhjh
Namespace: default
Priority: 0
Service Account: default
Node: minikube/192.168.59.100
Start Time: Tue, 27 Aug 2024 18:00:45 -0300
Labels: app=node-hello-world
pod-template-hash=6969875877
Annotations: <none>
Status: Running
IP: 10.244.0.16
IPs:
  IP: 10.244.0.16
Controlled By: ReplicaSet/node-teste-hello-world-deployment-6969875877
Containers:
  node-teste-hello-world:
    Container ID: docker://565172fe2065fdabc365537e224deabd6b18855f3ec084e46d459427024b2d3b
    Image: node-teste-hello-world:1.0
    Image ID: docker://sha256:dace8f8ee5173301ce5cfc6847365a468b61a364064a0e3042589f19a12dc318
    Port: 3000/TCP
    Host Port: 0/TCP
    State: Running
      Started: Tue, 27 Aug 2024 19:14:15 -0300
    Last State: Terminated
      Reason: Error
      Exit Code: 137
      Started: Tue, 27 Aug 2024 18:00:47 -0300
      Finished: Tue, 27 Aug 2024 19:13:12 -0300
    Ready: True
    Restart Count: 1
    Environment: <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-lxdd9 (ro)
Conditions:
  Type                               Status
  PodReadyToStartContainers         True
  Initialized                       True
  Ready                             True
  ContainersReady                   True
  PodScheduled                      True
```

- Como obter os recursos de CPU e memória dos Pods e do Node

kubectl top pod <sua pod>

kubectl top nodes

Questões

- Como criar um serviço com IP Público no Kubernetes e como obter o endereço deste serviço?

Resposta: Para criar um serviço com IP publico no kubernetes criamos um serviço do tipo Loadbalancer. Para obter o Ip do serviço criado usamos o comando abaixo:

kubectl get svc serviço-criado

- Como criar um serviço com IP Privado no Kubernetes e como obter o endereço deste serviço?

Resposta: declaramos o serviço com o tipo ClusterIP

Para obter o IP do serviço também usamos o comando :

kubectl get svc serviço-criado

- Como criar um serviço acessível apenas dentro do Cluster do Kubernetes?

Resposta: Para isso utilizamos o serviço do tipo cluster IP

- Como configurar um certificado SSL no Kubernetes?

- Qual a diferença do Deployment e StatefulSet?

Resposta: Deployment: Usado para aplicações stateless. Ele possibilita que o número desejado de réplicas do seu aplicativo esteja executando possibilitando deploys sem downtime. StatefulSet: Usado para aplicações stateful como bancos de dados. Ele garante a persistência do estado de cada Pod, mantendo um identificador

- Explique quando devemos usar um DNS do tipo CNAME e um do tipo A?

Um registro do tipo A devemos usar quando conhecemos o Ip do servidor Statico

Já cname aponta para um dominio o qual muitas vezes não temos a gerencia ou ciência do IP ou ele costuma mudar com frequencia. Com ele também é possível apontar vários subdominios para um dominio.

- Em um cenário em que o Desenvolvedor não consegue acessar uma base RDS retornando Connection Timeout, como é feito o troubleshoot e qual a possível causa?

Resposta: Eu para o troubleshooting desse cenário costumo utilizar o seguinte sequencia lógica.

Antes de tudo verifico se o RDS esta operando e as configurações das base de dados permitem acesso a conexões externas. Feito isso começo pela validação

das conexões ao RDS seguindo a sequencia abaixo:

- 1 - validar rede, verificando as regras do security group do RDS
- 2 - Validar se o cluster kubernetes ou a maquina do Dev pode acessar o RDS.
- 3 - Validar se o apontamento de DNS para o endpoint do RDS esta correto.
- 4 - Validar comunicação de acesso entre a VPN e o VPC funcionam.

Geralmente a causa mais comum para Connection Timeout costuma ser alguma regra de VPC que não permite o acesso do dev a base diretamente ou não estar habilitado conexões externas a base

- E se o erro no cenário acima for Connection Refused?

Identificado o ponto de bloqueio só ajustar as configurações

- Explique a diferença entre Jenkins e GithubActions. Qual sua preferência para fluxo de CI/CD e por quê?

Resposta: **Jenkins**: Um servidor de automação muito flexível, fácil de aumentar as funcionalidades usando os plugins. Tem como vantagem ser totalmente gerenciável e customizável. Porém, a desvantagem é que exigem constante manutenção e atualizações.

GitHub Actions : Também uma ferramenta de automação porém do Tipo SAS totalmente integrado ao GitHub, indicado para gerenciamento de pipelines.

- Já usou ArgoCD? Em que parte do Fluxo de CI/CD ele é utilizado e qual seu papel?

Respostas: Somente em POCs. Ele é usado no CD continuous delivery para a entrega dos objetos no kubernetes

=====