

Cluster Kubernetes Local

Sinopse:

Neste capítulo vamos criar um **namespace** e colocar todos os recursos dentro dele, conectaremos nossa aplicação a um banco de dados.

1. Vamos excluir os recursos que criamos até aqui e recriaremos em um namespace, você pode excluir pelo Lens ou linha de comando.

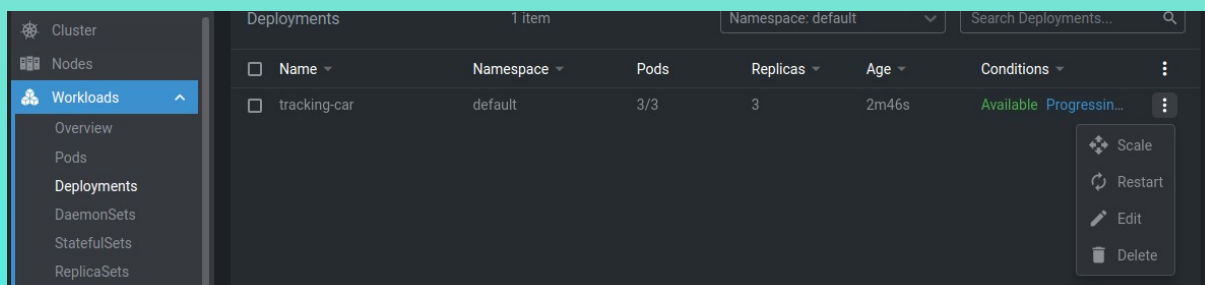
a) Na linha de comando digite;

```
~/Documentos/ProjetoK8S$ kubectl delete -f manifesto.yml
```

você receberá a saída;

```
deployment.apps "tracking-car" deleted
service "tracking-port" deleted
```

b) No Lens abra workloads em Deployments clique nos três pontos e **“Delete”**



- 1.
2. Agora criaremos o namespace com o nome **“app-tracking-car”**, podemos criar por linha de comando ou pelo Lens.

a) No terminal digite;

```
~/Documentos/ProjetoK8S$ kubectl create namespace app-tracking-car
```

Se ocorrer tudo OK você receberá a saída. `namespace/app-tracking-car created`

Você pode executar o comando **“kubectl get namespaces”** para verificar os namespaces existentes.

3. Com o namespace criado realizaremos as implantações para dentro do namespace **app-tracking-car** de duas formas que será declarando na linha de comando e outra dentro do manifesto.
 - a) Na linha de comando vamos implantar a aplicação digite o seguinte comando; **“kubectl apply -f manifesto.yml -n app-tracking-car”** aqui acrescentamos o **“-n”** abreviatura de namespace, e o nome **app-tracking-car**
 - b) **“kubectl get all -n app-tracking-car”** é o namespace onde o recurso deve ser alocado. Se você executar o comando **“kubectl get all”** para ver os recursos não encontrará nada porque ele estará buscando no namespace **“Default”** para verificar os recursos devemos passar o **-n** e o nome do namespace que desejamos verificar, então devemos executar o seguinte comando: **“kubectl get all -n app-tracking-car”**.

```
$kubectl get all -n app-tracking-car
```

NAME	READY	STATUS	RESTARTS	AGE
pod/tracking-car-6848f66cd5-68tlc	1/1	Running	0	18s
pod/tracking-car-6848f66cd5-c49qb	1/1	Running	0	18s
pod/tracking-car-6848f66cd5-c94mh	1/1	Running	0	18s

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/tracking-port	NodePort	10.109.221.60	<none>	80:31292/TCP	18s

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/tracking-car	3/3	3	3	18s

NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/tracking-car-6848f66cd5	3	3	3	18s

Nota: Existe outras formas que não abordaremos aqui sobre namespaces de deploy e visualização de recursos, criando contextos para facilitar o dia-dia e questões de segurança de acessos ao namespaces.

- c) Agora vamos ver o manifesto “db-mysql.yml” do nosso banco e os novos recursos que implementaremos nesta aula.

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: mysql
  namespace: app-tracking-car
spec:
  serviceName: mysql
  replicas: 1
  selector:
    matchLabels:
      app: mysql
kind: StatefulSet
metadata:
  name: mysql
  namespace: app-tracking-car
spec:
  serviceName: mysql
  replicas: 1
  selector:
    matchLabels:
      app: mysql
  template:
    metadata:
      labels:
        app: mysql
    spec:
      containers:
        - name: mysql
          image: mysql:5.6
          ports:
            - containerPort: 3306
              protocol: TCP
          env:
            - name: MYSQL_ROOT_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: mysql-secret
                  key: MYSQL_ROOT_PASSWORD
            - name: MYSQL_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: mysql-secret
                  key: MYSQL_PASSWORD
            - name: MYSQL_DATABASE
              valueFrom:
                secretKeyRef:
                  name: mysql-secret
                  key: MYSQL_DATABASE
```

- a) Estamos criando um recurso do tipo statefulSet que é indicado quando temos que ter persistência de dados.

Vamos a um resumo rápido de termos conhecidos com o conceito stateless&stateful. Quando precisamos que algo seja persistente mas ele pode sofrer quedas falhas mas sua resiliência ainda garantirá que nada seja perdido estamos falando de “stateful”, agora quando temos algo que pode sofrer avarias, encaixam-se neste conceito;

- StatefulSet que pelo nome já diz!! Esse recurso se compara ao conceito de stateful.
- ReplicaSet esse se compara com o stateless que no caso utilizamos dentro de um Deployment por convenção quando usamos esse recurso sempre usamos ele dentro de um Deployment melhor gerenciamento e controle do nosso pods.

- b) Após a declaração do tipo do recurso temos o Metadata que já vimos anteriormente só que vamos acrescentar o namespace onde esse recurso vai ser criado e não precisaremos explicitar na linha de comando como realizamos anteriormente
- c) Agora não muda muita coisa de um Deployment no caso temos o algumas coisas para apresentar que os nosso pods vão precisar como variáveis de ambientes criaremos mais a frente configMaps e Secrets. Como você pode ver eu estou declarando as variáveis de ambiente uma a uma, mas como elas estão no mesmo recurso poderia declarar de uma única vez, lembre-se disso pois mostrarei mas para frente o outro modo. Como você pode ver estamos declarando no nome do banco de dados senhas, aqui no arquivo só declaramos as variáveis nada de texto claro por questões de segurança elas ficam em um recurso chamado secrets que veremos a seguir.
4. Em secrets temos algumas formas de declarar aqui vou utilizar uma básica mas reforço na leitura da documentação. Como você pode ver abaixo na imagem os valores estão criptografados no nosso caso de secret sempre precisa ser codificado em base64 e declaramos o tipo Opaque

```
apiVersion: v1
kind: Secret
metadata:
  name: mysql-secret
  namespace: app-tracking-car
data:
  MYSQL_DATABASE: dHJhY2s=
  MYSQL_PASSWORD: cGFzc3dvcmQ=
  MYSQL_ROOT_PASSWORD: YWRtdHQ=
type: Opaque
```

Para codificar eu usei o próprio terminal do linux com o comando “echo {string} |base64”

```
$echo 'admtt' |base64
YWRtdHQK
```

Existem sites que pode fazer isso para você como estou utilizando o linux já codifique, fique tranquilo vou mostrar depois como a secret ficará dentro do kubernetes.

5. Agora vamos ver recurso de ConfigMap esse nós vamos utilizar dentro da nossa aplicação então vou lhe apresentar uma de muitas formas de utilizar esse recurso da mesma forma que citei sobre a secrets você deve sempre ver a documentação. No caso esse configMap vai substituir o arquivo de configuração xml da aplicação veja abaixo. Como você pode ver as declarações de namespace esta igual em metadata em todos os recursos que vamos criar

```

apiVersion: v1
kind: ConfigMap
metadata:
  name: mysql-datasources
  namespace: app-tracking-car
data:
  traccar.xml: |-
    <?xml version='1.0' encoding='UTF-8'?>

    <!DOCTYPE properties SYSTEM 'http://java.sun.com/dtd/properties.dtd'>

    <properties>

      <entry key='config.default'>/opt/traccar/conf/default.xml</entry>

      <!--

      This is the main configuration file. All your configuration parameters should be placed in this f

      Default configuration parameters are located in the "default.xml" file. You should not modify it
      with upgrading to a new version. Parameters in the main config file override values in the default
      remove "config.default" parameter from this file unless you know what you are doing.

      For list of available parameters see following page: https://www.traccar.org/configuration-file/

      -->

      <entry key='database.driver'>com.mysql.cj.jdbc.Driver</entry>
      <entry key='database.url'>jdbc:mysql://svc-mysql:3306/track?serverTimezone=UTC&allowPublicKey
      <entry key='database.user'>root</entry>
      <entry key='database.password'>admtt</entry>

    </properties>

  default.xml: |-
    <?xml version='1.0' encoding='UTF-8'?>

    <!DOCTYPE properties SYSTEM 'http://java.sun.com/dtd/properties.dtd'>

```

Em **Data** vamos declara chave e valor como vamos criar um arquivo você pode ver o **traccar.xml** colocamos o “|-” que significa que vai existir diversas linhas e mais abaixo temos outra chave com o nome de **default.xml** como você pode reparar ela já reconheceu da nova chave repare da indentação do arquivo.

6. Por ultimo temos um tipo de Service ClusterIP como o banco não precisa de comunicação externa igual um NodePort esse tipo de service é ideal para isso apenas para acesso interno das aplicações dentro da rede do kubernetes.

```

apiVersion: v1
kind: Service
metadata:
  name: svc-mysql
  namespace: app-tracking-car
  labels:
    app: mysql
spec:
  ports:
    - name: svc-mysql
      protocol: TCP
      port: 3306
  clusterIP: None
  selector:
    app: mysql

```

7. Vamos realizar o deploy dos recursos e listá-los.

```

$kubectl apply -f db-mysql.yml
statefulset.apps/mysql created
secret/mysql-secret created
configmap/mysql-datasources created
service/svc-mysql created

```

```
$kubectl get all -n app-tracking-car
```

NAME	READY	STATUS	RESTARTS	AGE
pod/mysql-0	1/1	Running	0	77s
pod/tracking-car-6848f66cd5-c49qb	1/1	Running	0	4h42m

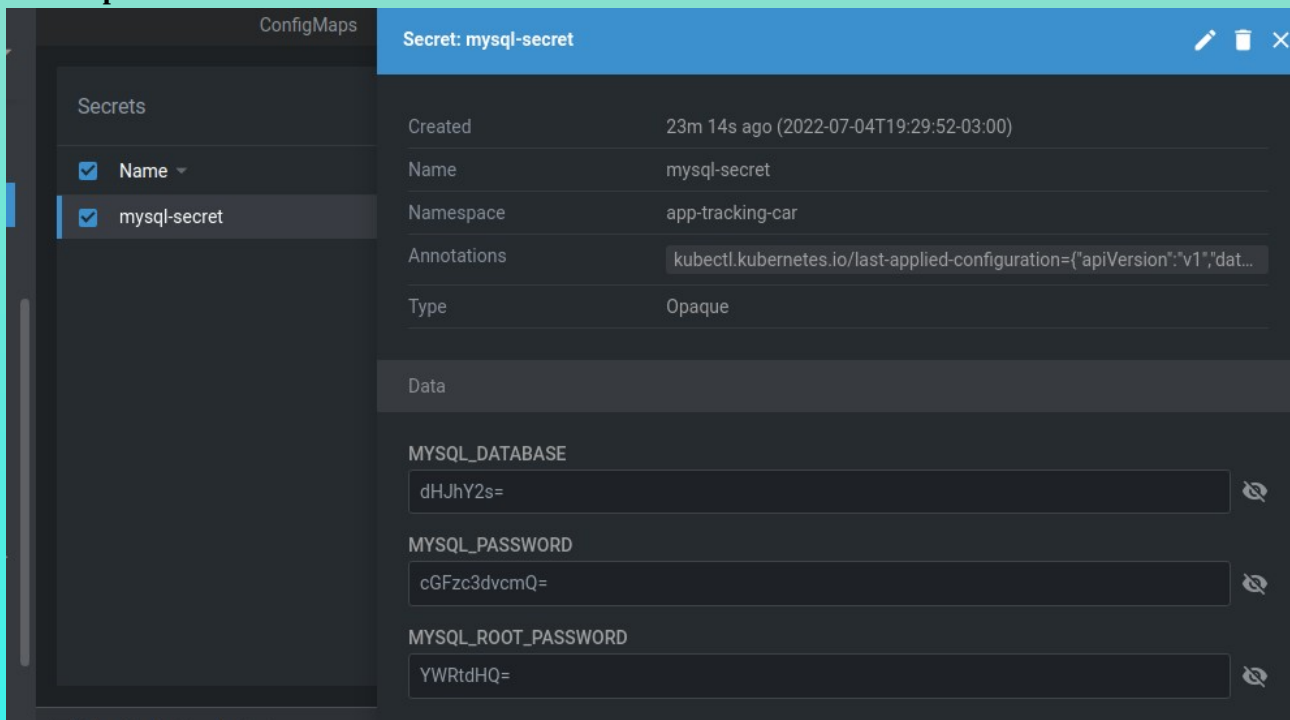
NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/svc-mysql	ClusterIP	None	<none>	3306/TCP	77s
service/tracking-port	NodePort	10.109.221.60	<none>	80:31292/TCP	4h42m

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/tracking-car	1/1	1	1	4h42m

NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/tracking-car-6848f66cd5	1	1	1	4h42m

NAME	READY	AGE
statefulset.apps/mysql	1/1	77s

8. Como pode verificar os serviços estão dentro do namespace que desejavamos vamos dar uma olhada pelo Lens os recursos que criamos e como havia dito anteriormente olharemos a Secret primeiro.



Se você clicar para desocultar a secret consegue ver em texto claro as credencias aqui conseguimos ver porque estamos utilizando o Lens faça se você quiser ver pelo terminal sairá assim

```
$kubectl describe secret mysql-secret -n app-tracking-car
```

Name: mysql-secret
 Namespace: app-tracking-car
 Labels: <none>
 Annotations: <none>

Type: Opaque

Data

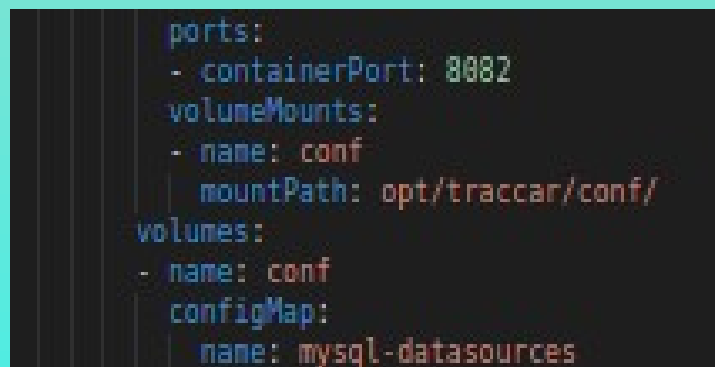
```
====
MYSQL_DATABASE:      5 bytes
MYSQL_PASSWORD:      8 bytes
MYSQL_ROOT_PASSWORD: 5 bytes
```

9. Explore os recursos que criamos e estude um pouca sua documentação no kubernetes. Ainda temos muito o que fazer, lembra que o ConfigMap ainda não esta declarado no manifesto da aplicação e por tanto só implantamos os recursos do banco, a aplicação ainda não esta conectada.
10. Abra o manifesto da aplicação e vamos configurar a variável de ambiente aqui vamos configurar ela para serem usadas como arquivos dentro Pod, lembra que havia dito que demonstraria a outra forma de declarar variáveis de um único arquivo uma única vez.

a) No arquivo de manifesto inclua abaixo do containerPort

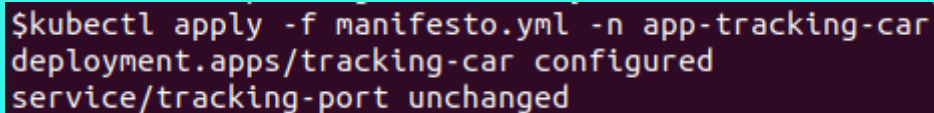
```
volumeMounts:
  - name: conf
    mountPath: opt/traccar/conf/
volumes:
  - name: conf
    configMap:
      name: mysql-datasources
```

b) Observe as indentações veja a imagem abaixo.



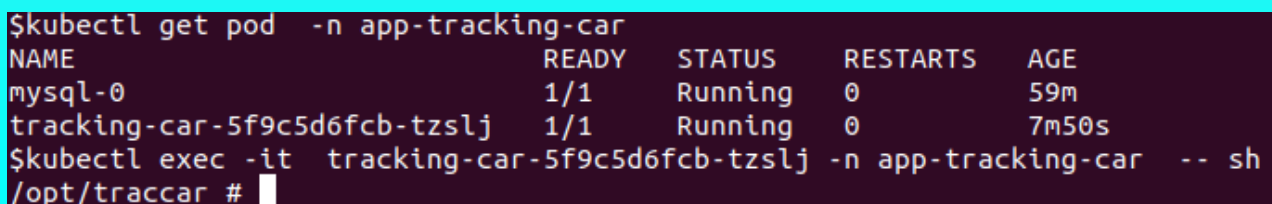
```
ports:
  - containerPort: 8082
volumeMounts:
  - name: conf
    mountPath: opt/traccar/conf/
volumes:
  - name: conf
    configMap:
      name: mysql-datasources
```

11. Faça o deploy novamente da aplicação lembre-se que o manifesto não tem o namespaces então execute declarando o namespace.



```
$kubectl apply -f manifesto.yml -n app-tracking-car
deployment.apps/tracking-car configured
service/tracking-port unchanged
```

12. Vamos acessar nosso Pod para verificar o arquivos do configMap, execute;
“kubectl exec -it {Pod} -n app-tracking-car – sh”



```
$kubectl get pod -n app-tracking-car
NAME                READY   STATUS    RESTARTS   AGE
mysql-0             1/1     Running   0           59m
tracking-car-5f9c5d6fcb-tzslj  1/1     Running   0           7m50s
$kubectl exec -it tracking-car-5f9c5d6fcb-tzslj -n app-tracking-car -- sh
/opt/traccar #
```

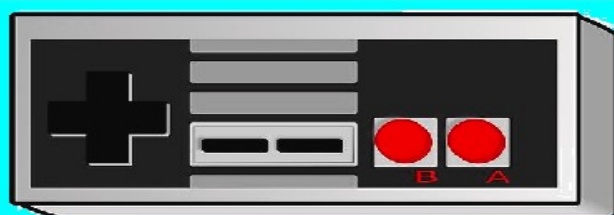

Acesse a o diretório conf e liste o que a dentro como a imagem abaixo.

```
/opt/traccar # cd conf/  
/opt/traccar/conf # ls  
default.xml  traccar.xml
```

De um cat no traccar.xml, como você pode ver ele é o arquivo que criamos no configMap.

```
default.xml  traccar.xml  
/opt/traccar/conf # cat traccar.xml  
<?xml version='1.0' encoding='UTF-8'?>  
  
<!DOCTYPE properties SYSTEM 'http://java.sun.com/dtd/properties.dtd'  
  
<properties>  
  
  <entry key='config.default'>/opt/traccar/conf/default.xml</entry>  
  
  <!--  
  
  This is the main configuration file. All your configuration parameters should be placed in this file.  
  
  Default configuration parameters are located in the "default.xml" file. You should not modify it to avoid issues  
  with upgrading to a new version. Parameters in the main config file override values in the default file. Do not  
  remove "config.default" parameter from this file unless you know what you are doing.  
  
  For list of available parameters see following page: https://www.traccar.org/configuration-file/  
  
  -->  
  
  <entry key='database.driver'>com.mysql.cj.jdbc.Driver</entry>  
  <entry key='database.url'>jdbc:mysql://svc-mysql:3306/track?serverTimezone=UTC&allowPublicKeyRetrieval=true&useSSL=false&allowMu  
ltiQueries=true&autoReconnect=true&useUnicode=yes&characterEncoding=UTF-8&sessionVariables=sql_mode=''</entry>  
  <entry key='database.user'>root</entry>  
  <entry key='database.password'>admtt</entry>  
  
</properties>/opt/traccar/conf # command terminated with exit code 137
```

Tudo ocorreu certo agora você já tem uma aplicação que está conectada a um banco de dados, você pode realizar alguma configuração no site e salvar destrua o pod da aplicação e suba novamente e verá que sua alteração ainda esta salva, mas lembre até aqui você viu o básico do básico se seu banco falhar todos os dados serão perdidos porque não temos nenhum recurso criado para o armazenamento de dados veremos nos capitulos futuros, como já venho falando não deixe de ler a documentação.



Congratulations
on your level up!