UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

Informe PTI Lab 4:

Microservices

Arnau Garcia Gonzalez

PTI Quadrimestre tardor Curs 2025-26

Índex

1. Introducció

Aquest laboratori tracta sobre el desplegament d'una aplicació basada en microserveis utilitzant **Kubernetes** i **Minikube**. L'objectiu és desplegar una API web de lloguer de cotxes, composta per diversos Pods gestionats mitjançant un *Deployment* i exposats amb un *Service*.

Un **Deployment** és un recurs de Kubernetes que gestiona la creació, actualització i replicació de Pods. Permet definir el nombre de rèpliques del microservei, la imatge Docker a utilitzar i la política d'actualització (*rolling update*), assegurant alta disponibilitat i tolerància a errors.

Un **Service** és un recurs que proporciona un punt d'accés estable als Pods gestionats pel Deployment. Gestiona l'enrutament del tràfic intern o extern al clúster, i permet comunicar-se amb l'aplicació sense necessitat de conèixer l'adreça concreta dels Pods.

En aquest laboratori, es desplegarà la API de lloguer de cotxes amb dos endpoints principals:

- **Crear un lloguer**: rep dades com la marca, model, dies i unitats, i retorna la informació del lloguer amb el preu total.
- **Llistar lloguers**: retorna en format JSON la llista de tots els lloguers registrats.

Les dades dels lloguers es guardaran en un fitxer dins del Pod. Com que els Pods són efímers, aquestes dades poden perdre's en reiniciar-se, però per aquesta pràctica és suficient. L'objectiu final és comprendre com es crea, gestiona i exposa una aplicació a Kubernetes utilitzant manifestos YAML i el panell de control de Minikube.

Per la part d'utilitzar la API de carrental, s'utilitzarà la API programada a la sessió de laboratori 3.

2. Tutorial Inicial: Kubernetes

Abans de la realització de la pràctica s'ha dut a terme el tutorial inicial de Kubernetes proporcionat pel professor a https://repo.fib.upc.es/felix.freitag/pti/-/tree/master/microservices

2.1 Instal·lació d'un petit clúster Kubernetes: MiniKube

S'utilitzarà l'eina de MiniKube, aquesta permet còrrer Kubernetes de forma local, amb un single-node Kubernetes. El primer que s'ha fet ha estat comprovar que docker ja era instal·lat i instal·lar kubectl, eina que s'utilitzarà per control·lar els clústers. Finalment s'ha instal·lat també MiniCube.

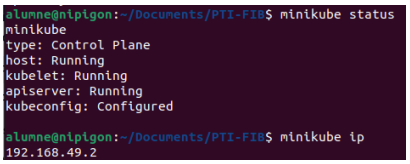
```Shell
docker -v    #Comprovació de que docker és instal·lat

#Instal·la kubectl
curl -LO https://storage.googleapis.com/kubernetes-release/release/$(curl -s
https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/li
nux/amd64/kubectl

#Modifica els permissos de kubectl i ho mou al directori indicat
chmod +x ./kubectl
sudo mv ./kubectl /usr/local/bin/kubectl

kubectl version --client   #Per comprobar la correcta instal·lació


#Per instal·lar MiniKube
curl -Lo minikube
https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64

#Modifica el permissos de MiniKube i ho mou al directori indicat
chmod +x minikube
sudo mv minikube /usr/local/bin
```

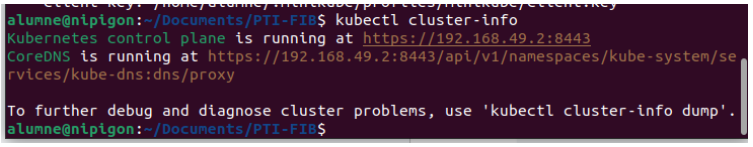
2.2 Llençar un clúster Kubernetes (MiniKube)

Per llençar el primer clúster Kubernetes s'han seguit els següents passos:

```Shell
#Configuracions inicials si kube s'ha instal·lat com a root
sudo mv /root/.kube /root/.minikube $HOME
sudo chown -R $USER $HOME/.kube $HOME/.minikube
sudo cp /etc/kubernetes/admin.conf $HOME/
sudo chown $(id -u):$(id -g) $HOME/admin.conf
```

```
export KUBECONFIG=$HOME/admin.conf
minikube update-context

#Crea i configura el clúster
minikube start

#Comandes útils
minikube status     #Comprova l'estat de MiniKube
minikube ip         #Comprova l'adreça IP
kubectl config view #Mostra el fitxer de configuració
kubectl cluster-info      #Comprova l'estat del cluster

minikube stop       #Atura MiniKube

#reinicia Minikube
export KUBECONFIG=$HOME/admin.conf
minikube --vm-driver=none start

minikube delete --all --purge    #Elimina els fitxers de Minikube
       rm -rf ~/.minikube  #Si l'anterior falla
```





2.3 Microservei HelloWorld!

Kubernetes necessita posar en contenidors els microserveis. Un microservei pot estar compost per diversos contenidors i altres recursos, però en aquest exemple només s'utilitzarà un contenidor. S'ha creat un directori /src i s'ha creat allà server.js amb el següent codi:

JavaScript

```
var http = require('http');

var handleRequest = function(request, response) {
  console.log('Received request for URL: ' + request.url);
  response.writeHead(200);
  response.end('Hello World!');
};
var www = http.createServer(handleRequest);
www.listen(8080);
```

També s'ha creat al mateix directori un Dockerfile amb el següent contingut:

```
Shell
FROM node:6.14.2
COPY server.js .
CMD node server.js
```

En un entorn real s'hauria construït la imatge localment i després s'hauria pujat a Docker Hub (o un altre registre) per tal que Kubernetes la pogués descarregar. Per simplificar, s'ha saltat l'ús d'un registre i s'ha indicat a Kubernetes que agafi les imatges directament del Docker intern de Minikube.

Com que Minikube utilitza el seu propi Docker, s'ha configurat el terminal perquè els comandos `docker` apuntessin cap al Docker de Minikube, i així s'han construït les imatges directament allà i han quedat accessibles al clúster. Per fer-ho s'ha utilitzat la següent comanda `eval $(minikube docker-env)`.

Després cal construir la imatge (primera comanda), però també s'ha provat de llençar-la per comprovar que el funcionament era correcte. Per això calen les següents comandes des del directori src/.

```
Shell
docker build -f Dockerfile -t helloworld:1.0 .
docker run --name helloworld -d -p 8080:8080 helloworld:1.0 #Per probar
curl $(minikube ip):8080   #Comprovar que està corrent
docker stop helloworld     #Per aturar el contenidor i alliberar el port
```

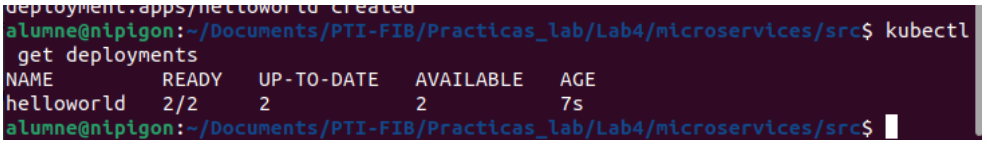
2.4 Desplegament del microservei

Aquest apartat consisteix a fer una configuració que comuniqui al Kubernetes com crear les instàncies del microservei.

Primer que tot crearem el deployment, proporcionant el nom i la localització de la imatge. Per això s'ha utilitzat la comanda següent:

```
Shell
kubectl get nodes #Comprova l'estat del clúster
#Crea el desplegament
kubectl create deployment helloworld --image=helloworld:1.0 --port=8080
--replicas=2

kubectl get deployments    #Per comprovar qu s'ha desplegat correctament
```

Després de comprovar l'estat s'ha comprovat que s'han creat dos pods, i que els dos estan ready.



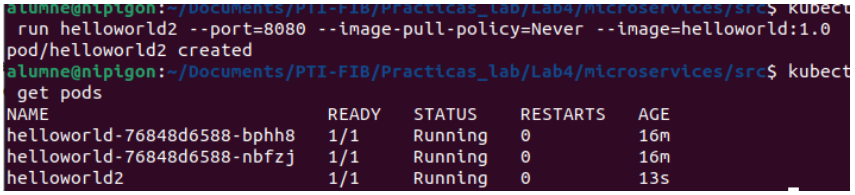
Si algun dels pods creats no estigués ready, seria necessari comprovar quin ha estat l'error, a continuació es poden veure algunes comandes útils per trobar-lo.

```shell
kubectl get pods    #Dona una llista del pods i el seu estat

#A partir d'aqui s'assumeix que s'ha guardat el nom del pod que no funcion
en una variable d'entorn amb
export MYPOD=helloworld-674c4d4dc8-6qn5r #Substituir amb el nom

kubectl describe pod $MYPOD      #Obté info del pod
kubectl logs pods/$MYPOD         #Veure logs del pod
kubectl logs pods/$MYPOD helloworld     # Per si hi ha més d'un contenidor

#Es pot accedir a endpoints de cada pot mitjançant un proxy, hja que l
clúster corre en xarxa privada
kubectl proxy &     #Crea el proxy per poder comunicar amb els endpoints
curl http://localhost:8001/api/v1/namespaces/default/pods/$MYPOD/proxy/
pkill kubectl       #Mata el proxy
```

2.4.1 Treballar sense desplegament

Si es vol treballar sense desplegament es pot utilitzar la comanda run de la següent manera:

```shell
#Comanda per correr
kubectl     run     helloworld2     --port=8080     --image-pull-policy=Never
--image=helloworld:1.0

#Elimina el pod que cabem de crear
kubectl delete pod helloworld2 --grace-period=0
```



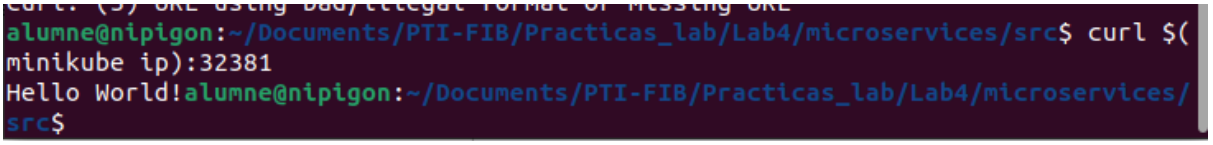
Aquest pod creat s'elimina a causa del fet que es treballarà amb desplegaments.

2.4.2 Serveis

L'execució d'un microservei es fa mitjançant un o més Pods que es creen i destrueixen dinàmicament. Per evitar que els clients hagin de gestionar aquesta complexitat, a Kubernetes es defineix un Service, que permet que el microservei rebi tràfic. El tipus per defecte és ClusterIP, que només és accessible dins del clúster.

S'ha creat un servei pel microservei HelloWorld, per fer-ho s'han executat les següents comandes. També s'han inclòs les comandes per eliminar el servei i desplegament.

```
Shell
#Comandes per crear el microservei i veure descripció
kubectl expose deployment/helloworld --type="NodePort" --port 8080
kubectl describe service helloworld

#Ja podem cridar al microservei
curl $(minikube ip):NODE_PORT #Substituir NODE_PORT pel vist en descripció

#Comandes per eliminar el servei/desplegament
kubectl delete service helloworld       #Elimina el servei
kubectl delete deployment helloworld    #Elimina el desplegament
```



2.5 Escalar microserveis

Si el tràfic augmenta són necessàries més instàncies de microserves (pods). Els microserveis ens permeten aquesta escalabilitat.

La següent comanda ens permet crear més rèpliques, 4 en aquest cas:

```
Shell
kubectl scale deployments/helloworld --replicas=4

kubectl get pods -o wide   #Comanda per veure el Nº de pods canviats
```

3. Realització de la pràctica

L'objectiu de la pràctica és crear i desplegar una API web de lloguer de cotxes amb dos endpoints:

- Crear un lloguer (rep dades: marca, model, dies, unitats → retorna dades i preu total).
- Llistar lloguers (retorna tots els lloguers en JSON).

Les dades es guardaran en un fitxer JSON dins del Pod (es poden perdre si el Pod es reinicia). Per aquesta pràctica s'utilitzarà la API REST realitzada en la Pràctica 3.

A més, es farà servir el Minikube Dashboard per visualitzar els Pods, Deployments i Services del clúster, i s'aprendrà a generar i utilitzar fitxers manifest YAML per gestionar el desplegament i el servei de l'aplicació.

3.1 Creació i desplegament de la API carrental

Per crear la imatge i desplegar la APi s'han seguit els passos esmentats a continuació, des del directori on es troba el codi de lAPI de la practica 3. S'han inclòs comentaris de la funcionalitat de cada comanda.

```shell
minikube start      #Per activar Minicube (en cas de que estigués inactiu)
eval $(minikube docker-env)      #Per construir l'imatge al Docker de Kube

#Construir la imatge docker
docker build -t carrental:1.0 .

#Crear el deployment a Kubernetes
kubectl create deployment carrental --image=carrental:1.0 --port=8080
--replicas=2

#Exposar el servei i comprovar que funciona
kubectl expose deployment carrental --type=NodePort --port=8080
kubectl get pods
kubectl get svc
```

```
pi$ kubectl get pods
NAME                         READY   STATUS    RESTARTS   AGE
carrental-754d85f666-9fph8   1/1     Running   0          13s
carrental-754d85f666-bzrfx   1/1     Running   0          13s
```

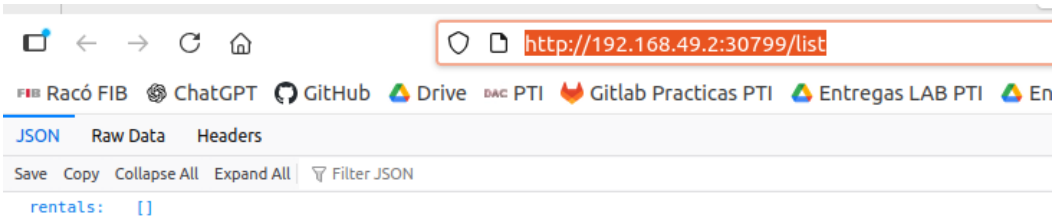
```
pi$ kubectl get svc
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
carrental    NodePort    10.98.235.177   <none>        8080:30799/TCP   50s
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP          3d21h
```

Després de comprovar que el desplegament s'ha efectuat correctament s'ha provat l'API per comprovar que funciona correctament.

Per a saber a quin URL s'està executant el servei s'ha executat la següent comanda: minikube service carrental --url. Així s'ha comprovat que l'URL és http://192.168.49.2:30799.

```
pi$ minikube service carrental --url
http://192.168.49.2:30799
```

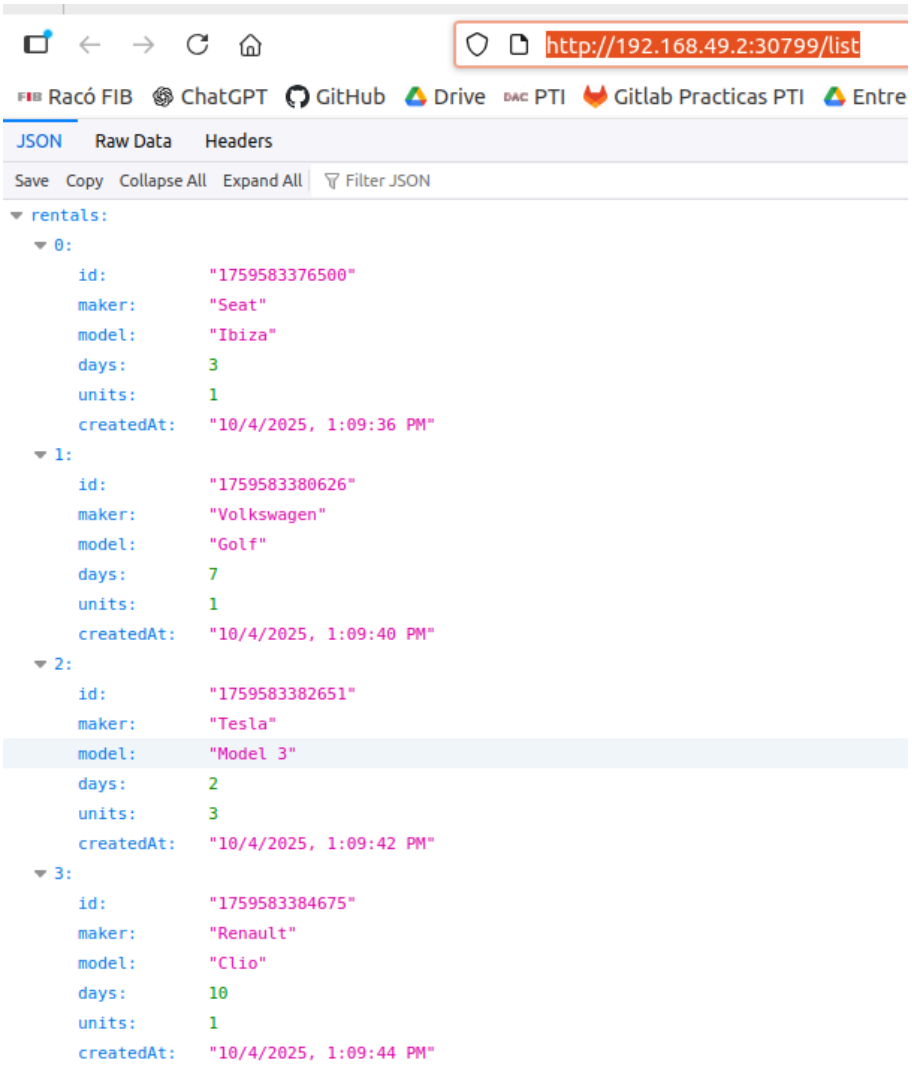
Per comprovar que l'API funciona correctament primer s'ha comprovat accedint a http://192.168.49.2:30799/list. S'ha obtingut el resultat esperat, ja que encara no hi ha cap lloguer registrat.



El següent pas per comprovar el funcionament de l'API ha estat provar la funció d'afegir un lloguer. Per a això s'ha executat el següent script, que afegeix 4 rentals mitjançant curl.

```shell
#Crea el primer lloguer
curl -i -H "Content-Type: application/json" \
     -d '{"maker":"Seat","model":"Ibiza","days":3,"units":1}' \
     http://192.168.49.2:30799/rentals

sleep 2    # Fa pausa per tenir diferent temps

#Crea el segon lloguer
curl -i -H "Content-Type: application/json" \
     -d '{"maker":"Volkswagen","model":"Golf","days":7,"units":1}' \
     http://192.168.49.2:30799/rentals
sleep 2    # Fa pausa per tenir diferent temps

#Crea el tercer lloguer
curl -i -H "Content-Type: application/json" \
     -d '{"maker":"Tesla","model":"Model 3","days":2,"units":3}' \
     http://192.168.49.2:30799/rentals
sleep 2    # Fa pausa per tenir diferent temps

#Crea el quart lloguer
curl -i -H "Content-Type: application/json" \
     -d '{"maker":"Renault","model":"Clio","days":10,"units":1}' \
     http://192.168.49.2:30799/rentals

echo Finished!
```

Després d'executar-lo i tornar a accedir a http://192.168.49.2:30799/list s'ha obtingut el següent output.

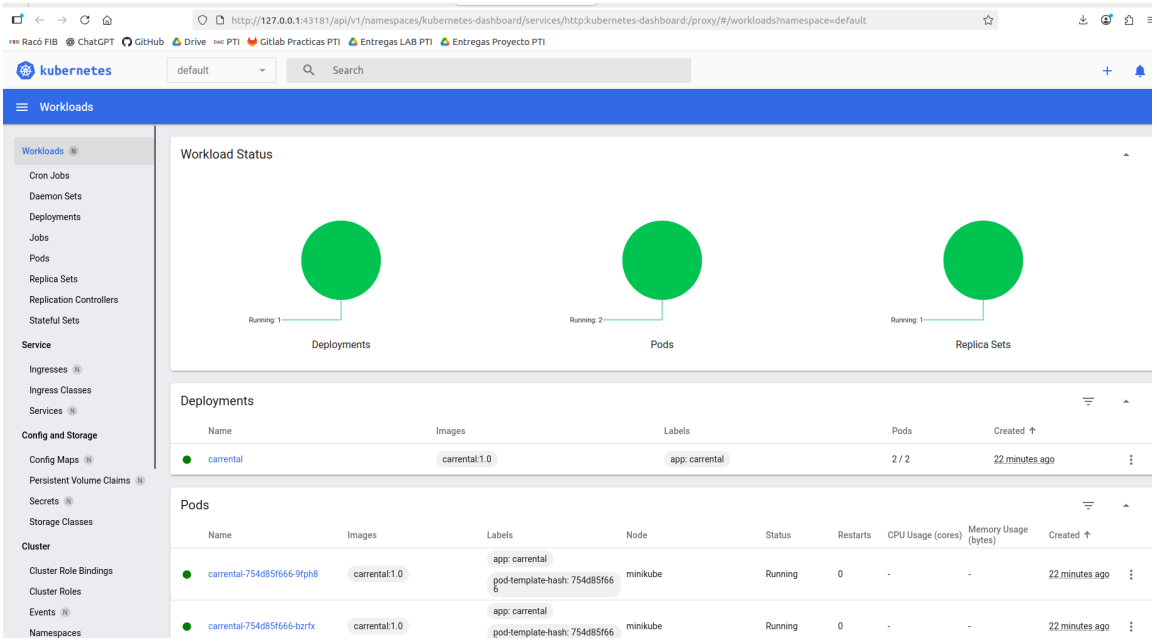


3.2 Dashboard MiniKube

Minicube ens permet veure els addons, aleshores podem activar un dashboard per gestió.

```shell
minikube addons list                    #Llista els addons
minikube addons enable dashboard        #Activa el dashboard
minikube dashboard                      #Obre el dashboard al navegador
```

S’han dut a terme aquestes accions amb el desplegament de carrental. I aquest és l’aspecte del dashboard.



3.3 Ús d’un fitxer Manifest

En aquest apartat s’ha practicat amb fitxers yaml. Primer, s’ha creat un fitxer YAML del Deployment del servei carrental, després s’ha eliminat aquest Deployment i s’ha tornat a crear a partir del fitxer obtingut. Després s’ha fet el mateix amb el Service: s’ha extret el manifest YAML, s’ha enviat el Service i s’ha tornat a desplegar amb aquest fitxer.

```shell
#Obté el Yaml del desplegament actual
kubectl get deployments/carrental -o=yaml > carrentaldeployment.yaml

#Elimina el desplegament
kubectl delete deployment carrental

#Torna a desplegar a partir del manifest
kubectl apply -f carrentaldeployment.yaml

#Crea el service
kubectl expose deployment carrental --type=NodePort --port=8080

#Exporta el manifest
kubectl get services/carrental -o=yaml > carrentalservice.yaml

#Elimina el Sevice
kubectl delete service carrental
```

```
#Crea el service a partir del manifest
kubectl apply -f carrentalservice.yaml

#Comprova la url
minikube service carrental --url
```

Finalment, s'ha comparat el contingut del dos yaml, contingut dels quals es pot trobar a continuació:

```
None
###### carrentaldeployment.yaml #########


apiVersion: apps/v1
kind: Deployment
metadata:
  annotations:
    deployment.kubernetes.io/revision: "1"
  creationTimestamp: "2025-10-04T12:55:10Z"
  generation: 1
  labels:
    app: carrental
  name: carrental
  namespace: default
  resourceVersion: "6022"
  uid: f3edf75b-59b6-4116-9cd0-52ddbc3b18a6
spec:
  progressDeadlineSeconds: 600
  replicas: 2
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app: carrental
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: carrental
    spec:
      containers:
```

```
      - image: carrental:1.0
        imagePullPolicy: IfNotPresent
        name: carrental
        ports:
        - containerPort: 8080
          protocol: TCP
        resources: {}
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      restartPolicy: Always
      schedulerName: default-scheduler
      securityContext: {}
      terminationGracePeriodSeconds: 30
status:
  availableReplicas: 2
  conditions:
  - lastTransitionTime: "2025-10-04T12:55:14Z"
    lastUpdateTime: "2025-10-04T12:55:14Z"
    message: Deployment has minimum availability.
    reason: MinimumReplicasAvailable
    status: "True"
    type: Available
  - lastTransitionTime: "2025-10-04T12:55:10Z"
    lastUpdateTime: "2025-10-04T12:55:14Z"
    message: ReplicaSet "carrental-754d85f666" has successfully progressed.
    reason: NewReplicaSetAvailable
    status: "True"
    type: Progressing
  observedGeneration: 1
  readyReplicas: 2
  replicas: 2
  updatedReplicas: 2
```

None

```
###### carrentalservice.yaml #########


apiVersion: v1
kind: Service
metadata:
  creationTimestamp: "2025-10-04T12:55:17Z"
  labels:
```

```
    app: carrental
  name: carrental
  namespace: default
  resourceVersion: "6026"
  uid: c89af9d7-cfed-49a1-8f1b-babe0eca9da1
spec:
  clusterIP: 10.98.235.177
  clusterIPs:
  - 10.98.235.177
  externalTrafficPolicy: Cluster
  internalTrafficPolicy: Cluster
  ipFamilies:
  - IPv4
  ipFamilyPolicy: SingleStack
  ports:
  - nodePort: 30799
    port: 8080
    protocol: TCP
    targetPort: 8080
  selector:
    app: carrental
  sessionAffinity: None
  type: NodePort
status:
  loadBalancer: {}
```