Practice 3: **Microservices**

PTI - G12

Josep Martínez

Adrià Martínez

Index

1. Environment.....3

2. Initialization of the cluster.....4

2.1 Initialization..... 4

2.2 Deployment.....5

2.3 Services.....6

3. Extras..... 8

3.2 Kubernetes Dashboard..... 8

3.2 Manifest file.....10

4.ANNEX..... 12

carrentaldeployment.yaml..... 12

carrentalservice.yaml.....14

1. Environment

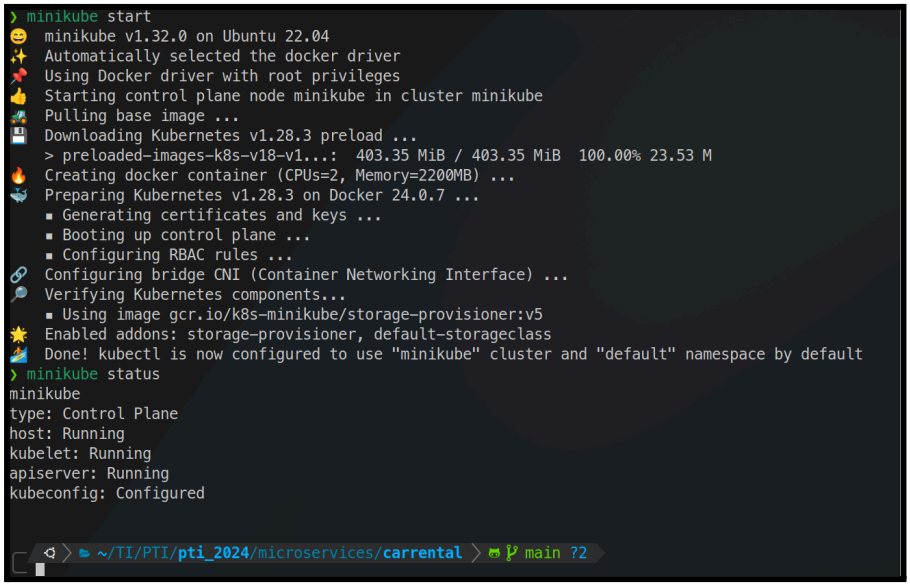
For this lab session, we will be using only **Zsh** as a command-line interpreter, since it is not necessary to use any programming IDE for the completion of this session. Also, **Firefox** is used to make and render web requests.

2. Initialization of the cluster

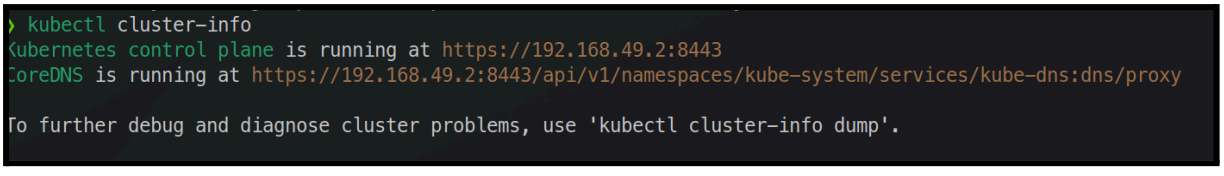
2.1 Initialization

The first step is to start up a local cluster with ***minikube***.

```
Unset
$ minikube start
$ minikube status
```



With `kubectl cluster-info` we can see the cluster's status.



Now we need to point our Docker commands directly to the Minikube built-in Docker daemon, building our images there, and making them directly accessible to Minikube.

```
Unset
$ eval $(minikube docker-env)
```

2.2 Deployment

Then we build a docker image with tag ***carrental:1.0*** based on the previous lab session Dockerfile.

```
Unset
$ docker build -f Dockerfile -t carrental:1.0 .
```

Before performing any deployment we can check the status of our nodes.

```
Unset
$ kubectl get nodes
```

```
❯ kubectl get nodes
NAME       STATUS   ROLES           AGE     VERSION
minikube   Ready    control-plane   9m42s   v1.28.3
```

Now it is time to create a deployment with our docker image and two replicas (two pods).

```
Unset
$ kubectl create deployment carrental --image=carrental:1.0 --port=8080
--replicas=2
```

As we can see, the deployment is created successfully.

```
❯ kubectl create deployment carrental --image=carrental:1.0 --port=8080 --replicas=2

deployment.apps/carrental created
❯ kubectl get deployments

NAME        READY   UP-TO-DATE   AVAILABLE   AGE
carrental   2/2     2            2           5s
```

We can see now that our deployment is running two pods.

```
Unset
$ kubectl get pods
```

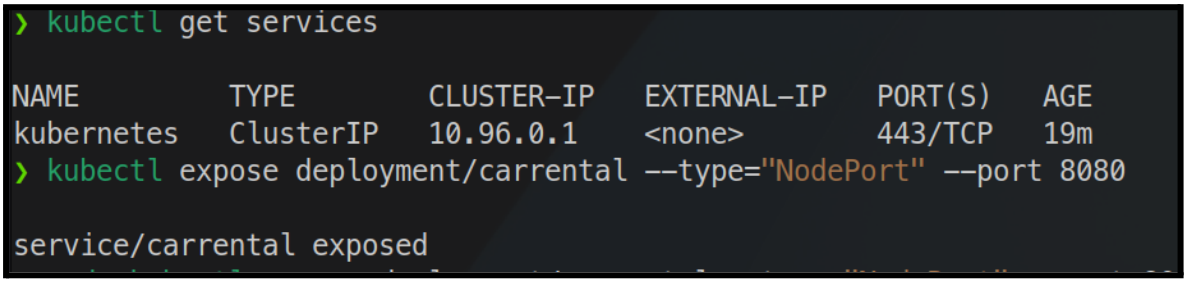
```
❯ kubectl get pods

NAME                         READY   STATUS    RESTARTS   AGE
carrental-86c84d86cc-dct96   1/1     Running   0          75s
carrental-86c84d86cc-jwgfk   1/1     Running   0          75s
```

2.3 Services

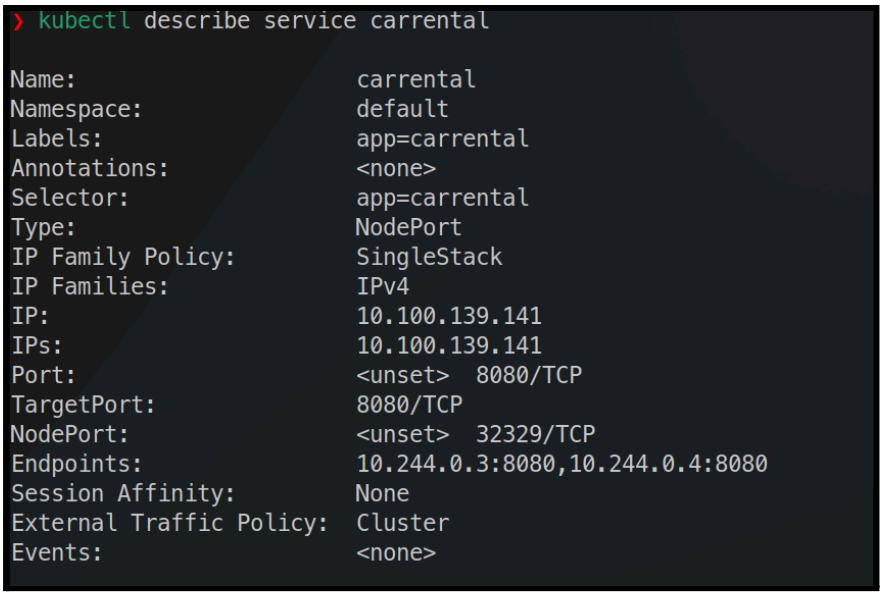
First we can see that the only running service for now is the default *kubernetes* service, created by ***minikube***.
Then we create a new service with ***expose*** command for our deployment named ***carrental***.

```
Unset
$ kubectl get services
$ kubectl expose deployment/carrental --type="NodePort" --port 8080
```



We can list all info related to the recently created service.

```
Unset
$ kubectl describe service carrental
```

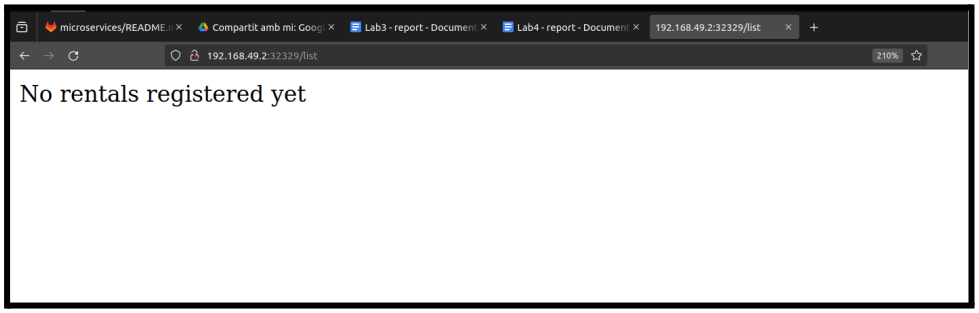


Note that we will be using **NodePort** to access our service.

Now that our deployment is exposed via a service, we can access it through the NodePort present in the previous dump. The endpoint is ClusterIP:NodePort, in this case:

`192.168.49.2:32329`

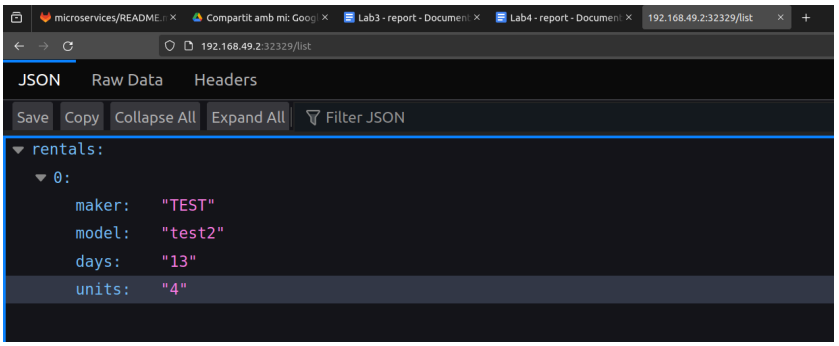
To check that our service is 100% functional we try to list all rentals via ***/list*** endpoint, like the previous lab session.



Then we forge a request with a new car rental.

```
Unset
curl -X POST -H "Content-Type: application/json" -d
'{"maker":"TEST","model":"test2", "days":"13", "units":"4"}'
http://192.168.49.2:32329/newrental
```

And finally we try to list all car rentals again. As we can see, the service registered and processed the request successfully.



3. Extras

3.2 Kubernetes Dashboard

As an additional feature, we added the kubernetes Dashboard in order to deploy containerized applications to a Kubernetes cluster, troubleshoot our containerized application, and manage the cluster resources. The dashboard is a useful tool to get the overviews of applications running in our cluster and get additional information in an understandable way, with some graphical visualizations of everything. The followed steps where the following:

```
Unset
minikube addons
minikube addons enable dashboard
```

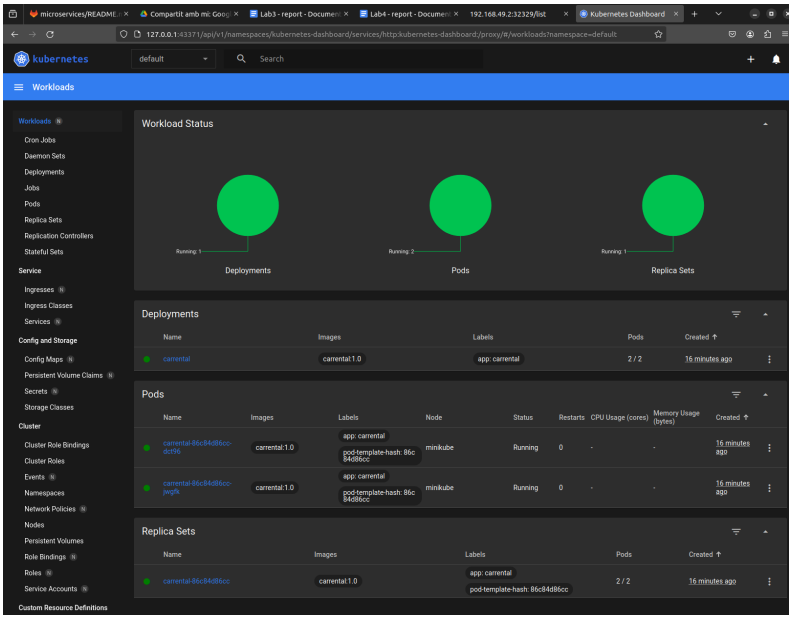
```
❯ minikube addons enable dashboard

💡  dashboard is an addon maintained by Kubernetes. For any concerns contact minikube on GitHub.
You can view the list of minikube maintainers at: https://github.com/kubernetes/minikube/blob/master/OWNERS
    ▪ Using image docker.io/kubernetesui/metrics-scraper:v1.0.8
    ▪ Using image docker.io/kubernetesui/dashboard:v2.7.0
💡  Some dashboard features require the metrics-server addon. To enable all features please run:

        minikube addons enable metrics-server


🌟  The 'dashboard' addon is enabled
```

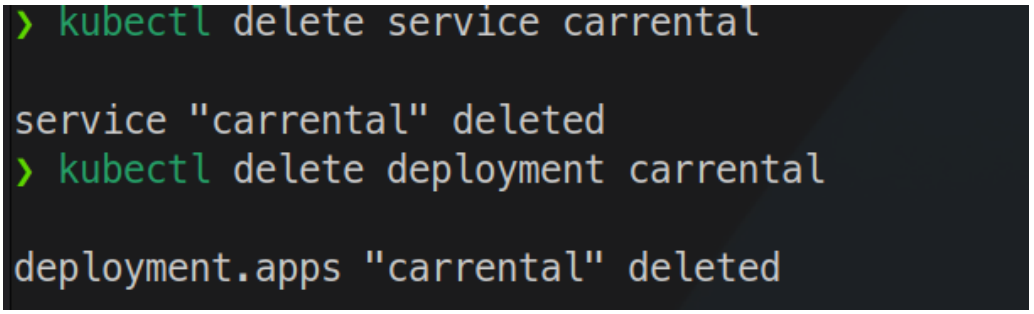
```
Unset
minikube dashboard
```



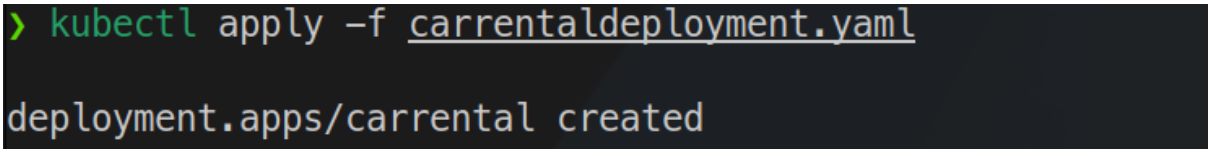
3.2 Manifest file

Kubernetes manifest files are YAML or JSON files that describe objects in the cluster. They're the primary way to manage the objects as they let you version configurations alongside your code, then declaratively apply them to your cluster.
The main reason to adopt this solution in our kubernetes project is the fact that you can get the configuration of the objects we are creating for our deployments and services. Thanks to that we can recreate the same status if we are working on different machines, edit the configurations manually and also use them for our CI/CD controls since they will define the status of everything in our cluster.
The steps followed to implement this solution were:

```
Unset
kubectl get deployments/carrental -o=yaml > carrentaldeployment.yaml
kubectl delete service carrental
kubectl delete deployment carrental
```




```
Unset
kubectl apply -f carrentaldeployment.yaml
```




```
Unset
kubectl expose deployment/carrental --type="NodePort" --port 8080
```

```
kubectl get services/carrental -o=yaml > carrentalservice.yaml
```

```
❯ kubectl expose deployment/carrental --type="NodePort" --port 8080

service/carrental exposed
❯ kubectl get services/carrental -o=yaml > carrentalservice.yaml
```

Unset

```
kubectl delete service carrental
kubectl apply -f carrentalservice.yaml
```

```
❯ kubectl delete service carrental
service "carrental" deleted
❯ kubectl apply -f carrentalservice.yaml

service/carrental created
```

Unset

```
kubectl describe service carrental
```

```
❯ kubectl describe service carrental

Name:                     carrental
Namespace:                default
Labels:                   app=carrental
Annotations:              <none>
Selector:                 app=carrental
Type:                     NodePort
IP Family Policy:         SingleStack
IP Families:              IPv4
IP:                       10.111.31.51
IPs:                      10.111.31.51
Port:                     <unset>  8080/TCP
TargetPort:               8080/TCP
NodePort:                 <unset>  32087/TCP
Endpoints:                10.244.0.7:8080,10.244.0.8:8080
Session Affinity:         None
External Traffic Policy:  Cluster
Events:                   <none>
```

4.ANNEX

---

carrentaldeployment.yaml

```
Unset
apiVersion: apps/v1
kind: Deployment
metadata:
 annotations:
       deployment.kubernetes.io/revision: "1"
 creationTimestamp: "2024-03-09T11:21:14Z"
 generation: 1
 labels:
       app: carrental
 name: carrental
 namespace: default
 resourceVersion: "1003"
 uid: f503d974-0c69-4ded-9013-c93c95571645
spec:
 progressDeadlineSeconds: 600
 replicas: 2
 revisionHistoryLimit: 10
 selector:
       matchLabels:
       app: carrental
 strategy:
       rollingUpdate:
       maxSurge: 25%
       maxUnavailable: 25%
       type: RollingUpdate
 template:
       metadata:
       creationTimestamp: null
       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: "2024-03-09T11:21:17Z"
      lastUpdateTime: "2024-03-09T11:21:17Z"
      message: Deployment has minimum availability.
      reason: MinimumReplicasAvailable
      status: "True"
      type: Available
 - lastTransitionTime: "2024-03-09T11:21:14Z"
      lastUpdateTime: "2024-03-09T11:21:17Z"
      message: ReplicaSet "carrental-86c84d86cc" has successfully progressed.
      reason: NewReplicaSetAvailable
      status: "True"
      type: Progressing
 observedGeneration: 1
 readyReplicas: 2
 replicas: 2
 updatedReplicas: 2
```

carrentalservice.yaml

```
Unset
apiVersion: v1
kind: Service
metadata:
 creationTimestamp: "2024-03-09T11:44:40Z"
 labels:
       app: carrental
 name: carrental
 namespace: default
 resourceVersion: "2260"
 uid: 0b97e6e1-7966-4c04-9cc9-e44b422df7a1
spec:
 clusterIP: 10.111.31.51
 clusterIPs:
 - 10.111.31.51
 externalTrafficPolicy: Cluster
 internalTrafficPolicy: Cluster
 ipFamilies:
 - IPv4
 ipFamilyPolicy: SingleStack
 ports:
 - nodePort: 32087
       port: 8080
       protocol: TCP
       targetPort: 8080
 selector:
       app: carrental
 sessionAffinity: None
 type: NodePort
status:
 loadBalancer: {}
```