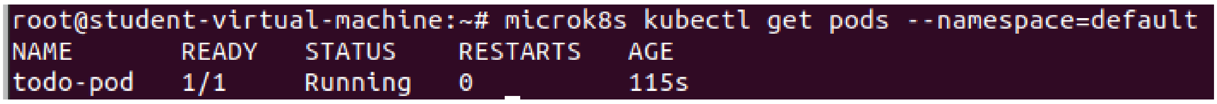
Operating Systems 3: lab 3 report

1. “Todo” web service pod deployment

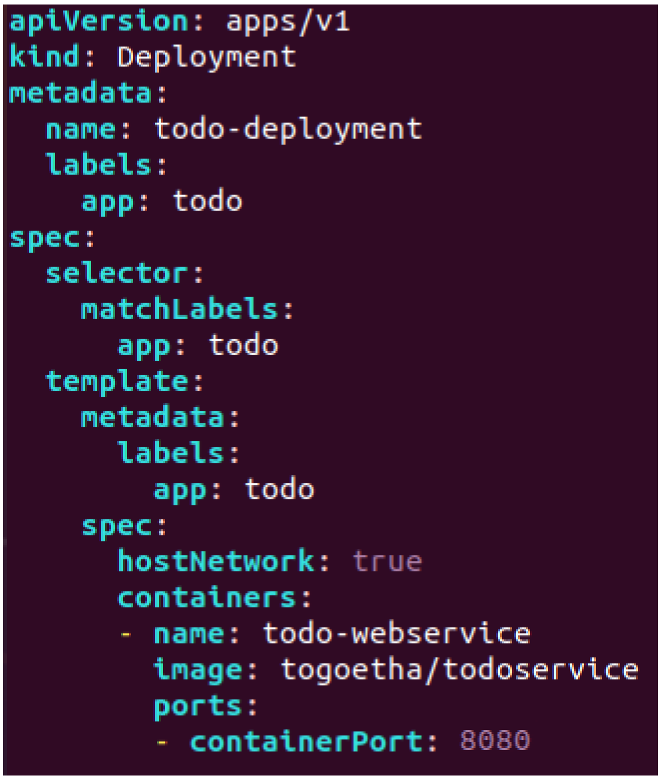
After creating the pod using the command **microk8s kubectl create -f pod.yaml**, we can verify that our pod is running using the following command:



```
root@student-virtual-machine:~# microk8s kubectl get pods --namespace=default
NAME       READY   STATUS    RESTARTS   AGE
todo-pod   1/1     Running   0          115s
```

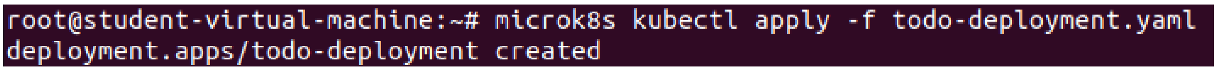
2. Switching to a Deployment

Instead of using a raw pod, we will use a deployment so that we can use the benefits of lifecycle and scalability. We convert the YAML previously used for our raw pod to a deployment YAML. Requirements are that we maintain a single replica, thus we don’t use the replicas field yet, and that the labels and matchLabels fields are all named ‘todo’.



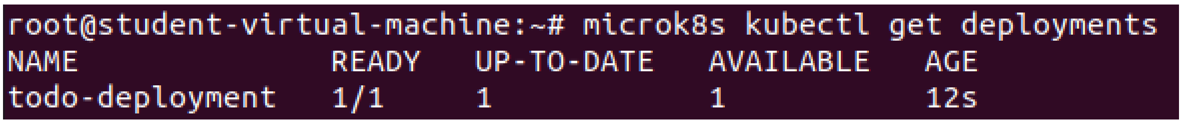
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: todo-deployment
  labels:
    app: todo
spec:
  selector:
    matchLabels:
      app: todo
  template:
    metadata:
      labels:
        app: todo
    spec:
      hostNetwork: true
      containers:
      - name: todo-webservice
        image: togoetha/todoservice
        ports:
        - containerPort: 8080
```

We can create our Deployment by running the following command:



```
root@student-virtual-machine:~# microk8s kubectl apply -f todo-deployment.yaml
deployment.apps/todo-deployment created
```

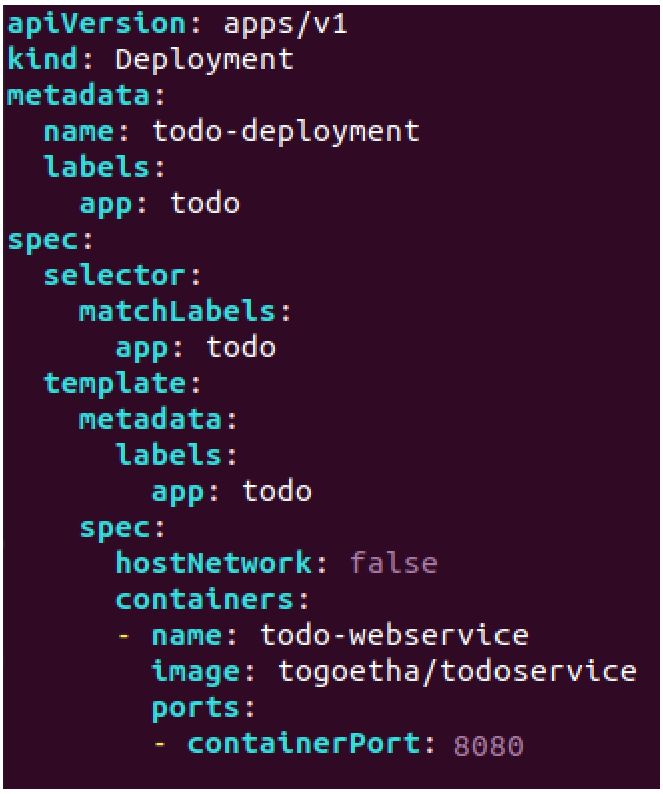
We verify that our Deployment was correctly created using the following command:



```
root@student-virtual-machine:~# microk8s kubectl get deployments
NAME              READY   UP-TO-DATE   AVAILABLE   AGE
todo-deployment   1/1     1            1           12s
```

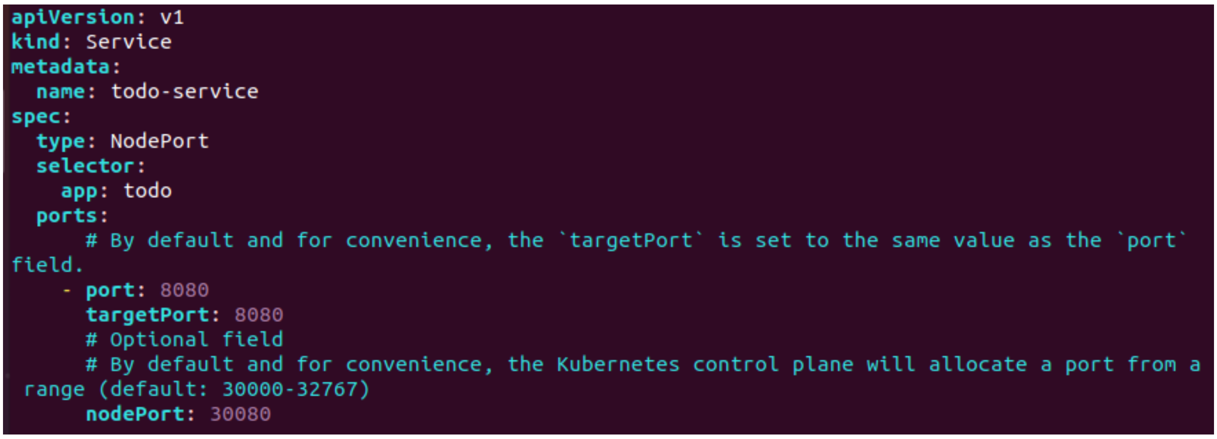
3. Put the pods in the container network

Next, we would prefer not to use our host network due to various security reasons as well as scalability and flexibility. We now change the deployment YAML so that the pods are managed by the CNI plugin, i.e. Calico, rather than the host’s network. We set the hostNetwork to false.



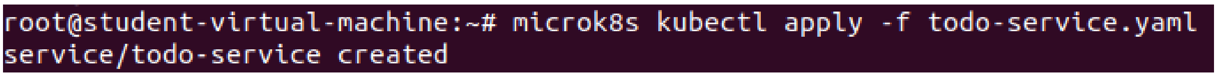
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: todo-deployment
  labels:
    app: todo
spec:
  selector:
    matchLabels:
      app: todo
  template:
    metadata:
      labels:
        app: todo
    spec:
      hostNetwork: false
      containers:
      - name: todo-webservice
        image: togoetha/todoservice
        ports:
        - containerPort: 8080
```

We now have to create a Service, here a NodePort service, that acts as an entry point and load balancer for the Deployment’s pods. Here is the YAML for this NodePort service:



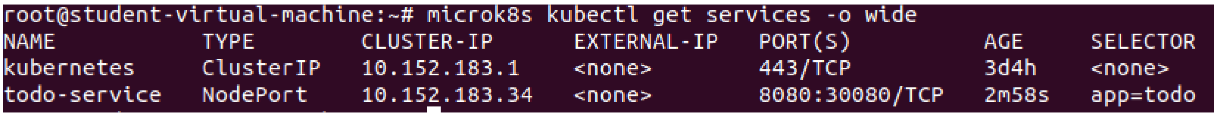
```
apiVersion: v1
kind: Service
metadata:
  name: todo-service
spec:
  type: NodePort
  selector:
    app: todo
  ports:
      # By default and for convenience, the `targetPort` is set to the same value as the `port`
field.
    - port: 8080
      targetPort: 8080
      # Optional field
      # By default and for convenience, the Kubernetes control plane will allocate a port from a
 range (default: 30000-32767)
      nodePort: 30080
```

We can create our Service by executing the following command:



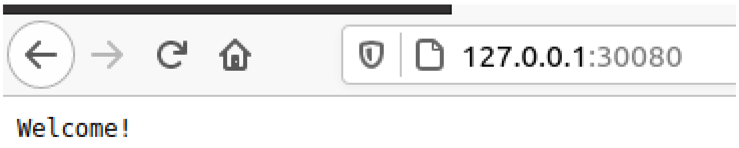
```
root@student-virtual-machine:~# microk8s kubectl apply -f todo-service.yaml
service/todo-service created
```

We verify that all is well:



```
root@student-virtual-machine:~# microk8s kubectl get services -o wide
NAME           TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE    SELECTOR
kubernetes     ClusterIP   10.152.183.1    <none>        443/TCP          3d4h   <none>
todo-service   NodePort    10.152.183.34   <none>        8080:30080/TCP   2m58s  app=todo
```

And in the browser:

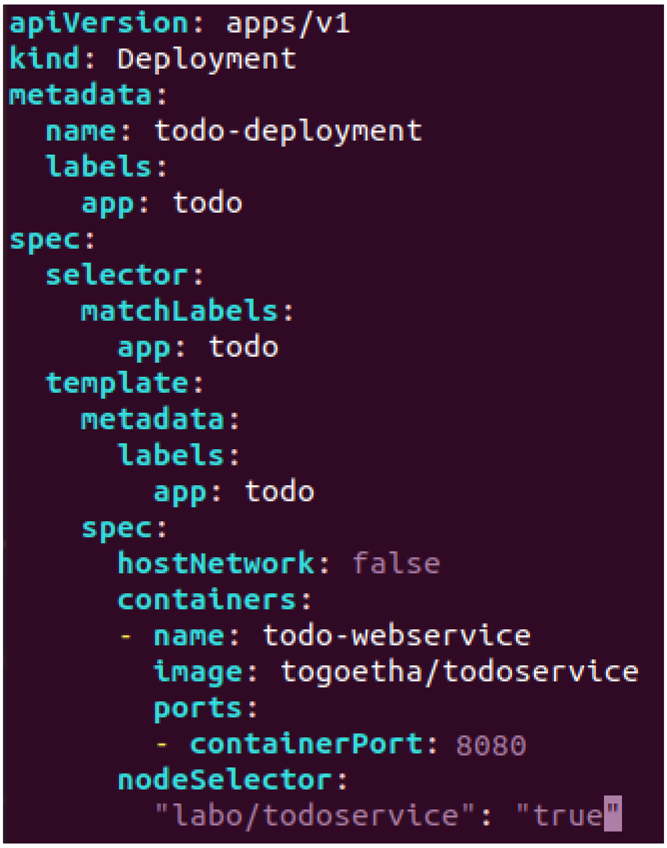


127.0.0.1:30080

```
Welcome!
```

4. Node selection and deployment metadata

We now add a nodeSelector for the label ‘labo/todoservice’ with value ‘true’ to our Deployment, resulting in the following, updated YAML:

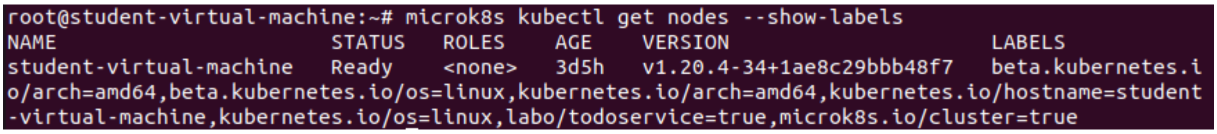


```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: todo-deployment
  labels:
    app: todo
spec:
  selector:
    matchLabels:
      app: todo
  template:
    metadata:
      labels:
        app: todo
    spec:
      hostNetwork: false
      containers:
      - name: todo-webservice
        image: togoetha/todoservice
        ports:
        - containerPort: 8080
      nodeSelector:
        "labo/todoservice": "true"
```

We must also add the proper labels to the node student-virtual-machine using the following command:

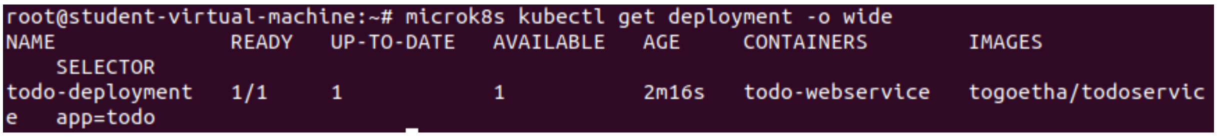
microk8s kubectl label nodes student-virtual-machine labo/todoservice=true

We verify our work using the following command and indeed see our K-V pair present as a label:



```
root@student-virtual-machine:~# microk8s kubectl get nodes --show-labels
NAME                      STATUS   ROLES    AGE    VERSION                     LABELS
student-virtual-machine   Ready    <none>   3d5h   v1.20.4-34+1ae8c29bbb48f7   beta.kubernetes.i
o/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/arch=amd64,kubernetes.io/hostname=student
-virtual-machine,kubernetes.io/os=linux,labo/todoservice=true,microk8s.io/cluster=true
```

Indeed, our Deployment is running:

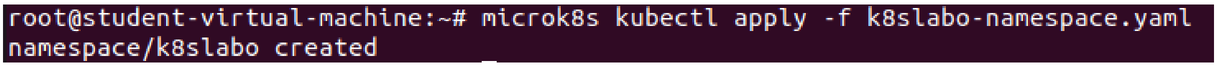


```
root@student-virtual-machine:~# microk8s kubectl get deployment -o wide
NAME              READY   UP-TO-DATE   AVAILABLE   AGE     CONTAINERS        IMAGES
    SELECTOR
todo-deployment   1/1     1            1           2m16s   todo-webservice   togoetha/todoservic
e    app=todo
```

5. Moving to another namespace

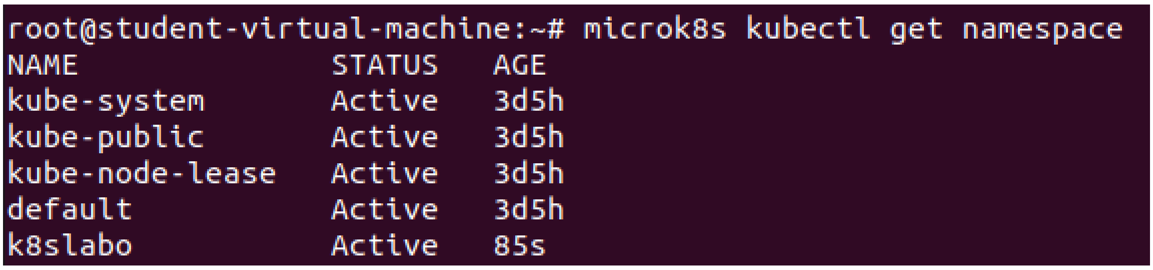
We would now like to put the Deployment in its own namespace, i.e. “k8slabo”. We will also have to change the namespace of the NodePort service.

First, we create the namespace using the given YAML.



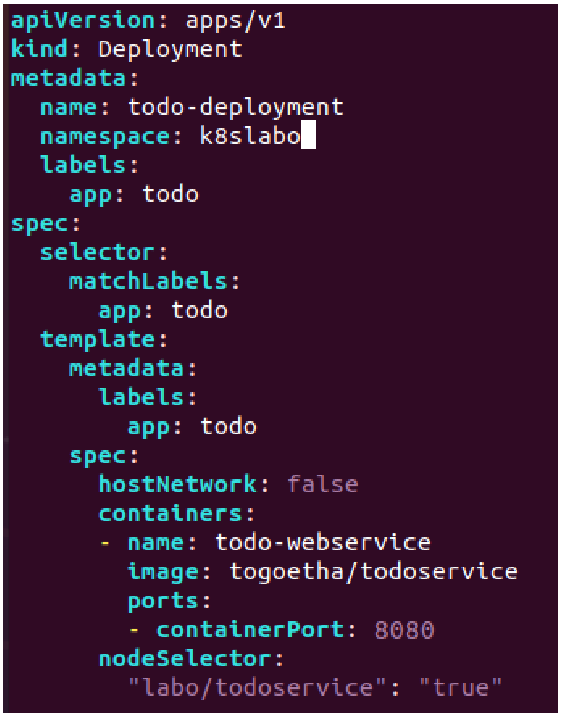
```
root@student-virtual-machine:~# microk8s kubectl apply -f k8slabo-namespace.yaml
namespace/k8slabo created
```

We verify that our namespace has been created and indeed see it listed as the very last namespace:

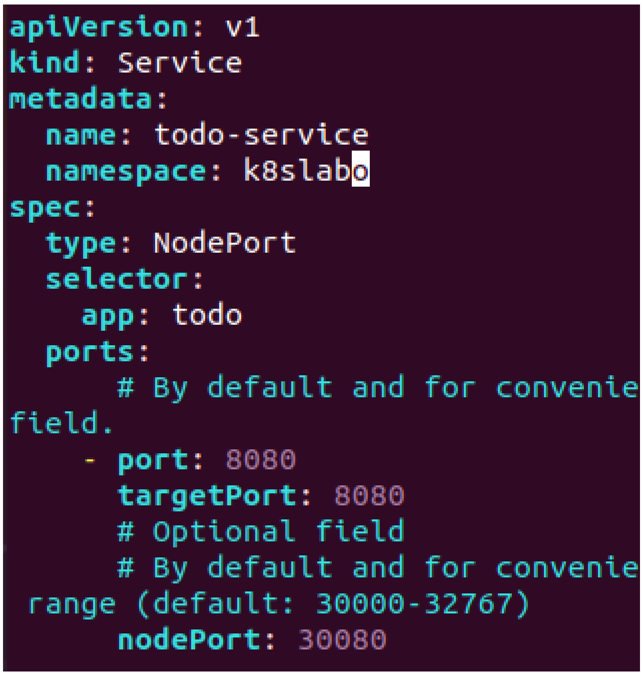


```
root@student-virtual-machine:~# microk8s kubectl get namespace
NAME              STATUS   AGE
kube-system       Active   3d5h
kube-public       Active   3d5h
kube-node-lease   Active   3d5h
default           Active   3d5h
k8slabo           Active   85s
```

We will now update both the Deployment and Service YAML to include our namespace.

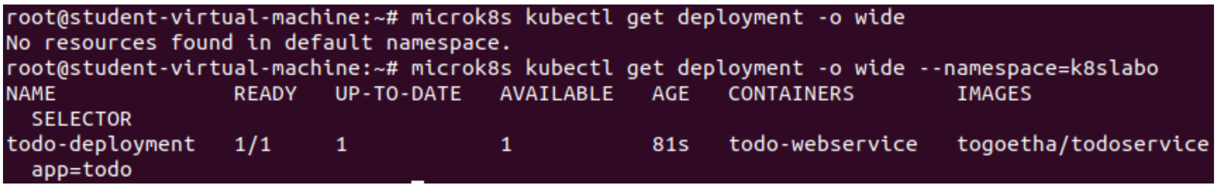


```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: todo-deployment
  namespace: k8slabo
  labels:
    app: todo
spec:
  selector:
    matchLabels:
      app: todo
  template:
    metadata:
      labels:
        app: todo
    spec:
      hostNetwork: false
      containers:
      - name: todo-webservice
        image: togoetha/todoservice
        ports:
        - containerPort: 8080
      nodeSelector:
        "labo/todoservice": "true"
```

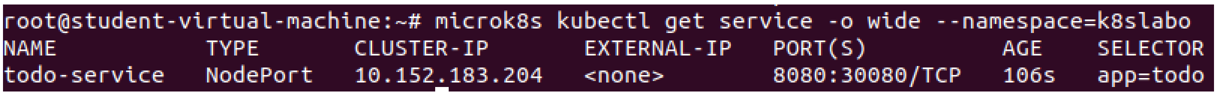


```
apiVersion: v1
kind: Service
metadata:
  name: todo-service
  namespace: k8slabo
spec:
  type: NodePort
  selector:
    app: todo
  ports:
      # By default and for convenie
field.
    - port: 8080
      targetPort: 8080
      # Optional field
      # By default and for convenie
 range (default: 30000-32767)
      nodePort: 30080
```

We will now verify that our Deployment and NodePort have been deployed in the correct namespace using the following commands:



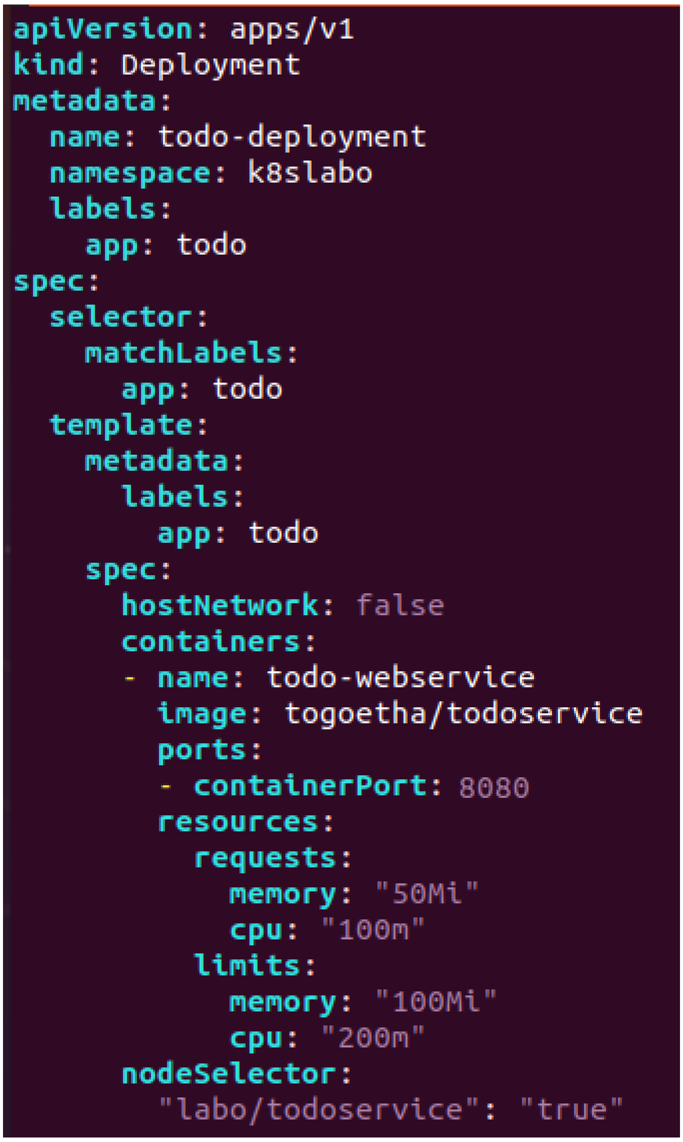
```
root@student-virtual-machine:~# microk8s kubectl get deployment -o wide
No resources found in default namespace.
root@student-virtual-machine:~# microk8s kubectl get deployment -o wide --namespace=k8slabo
NAME              READY   UP-TO-DATE   AVAILABLE   AGE   CONTAINERS         IMAGES
  SELECTOR
todo-deployment   1/1     1            1           81s   todo-webservice    togoetha/todoservice
  app=todo
```



```
root@student-virtual-machine:~# microk8s kubectl get service -o wide --namespace=k8slabo
NAME           TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE    SELECTOR
todo-service   NodePort   10.152.183.204  <none>        8080:30080/TCP   106s   app=todo
```

6. Resource restrictions

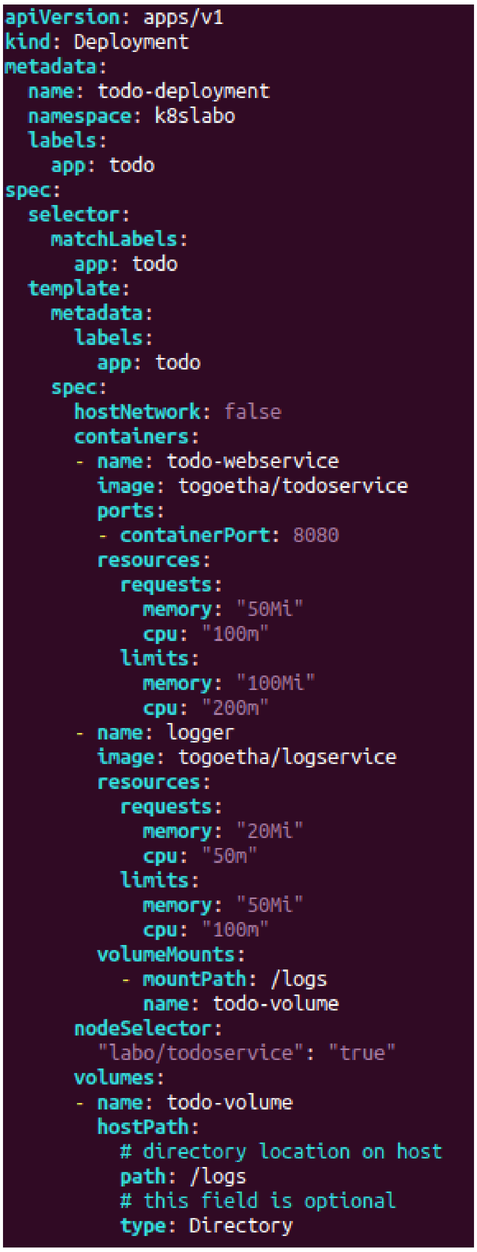
We will now add resource restrictions to our Deployment. The resource request should be 50Mi of memory and 100m (0.1) CPU, and the maximum use (limits) should be 100Mi and 200m CPU.



```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: todo-deployment
  namespace: k8slabo
  labels:
    app: todo
spec:
  selector:
    matchLabels:
      app: todo
  template:
    metadata:
      labels:
        app: todo
    spec:
      hostNetwork: false
      containers:
      - name: todo-webservice
        image: togoetha/todoservice
        ports:
        - containerPort: 8080
        resources:
          requests:
            memory: "50Mi"
            cpu: "100m"
          limits:
            memory: "100Mi"
            cpu: "200m"
      nodeSelector:
        "labo/todoservice": "true"
```

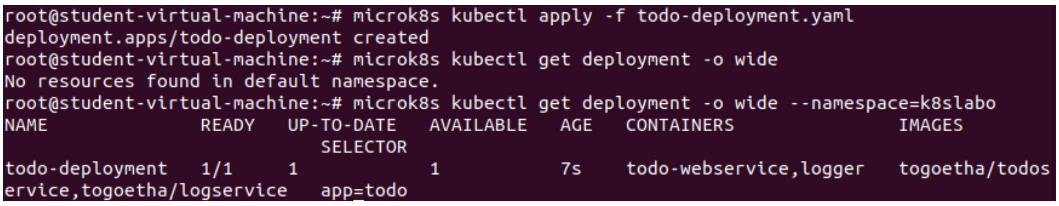
7. Adding a logger container

We now add a second container to the pod, serving as a logger. We also now want to access the file to which the logger writes its output from outside the container. We will need to mount a host directory to the /logs directory inside the container by creating a Volume of type hostPath, which points to a directory on the Kubernetes host (our VM), and then creating a volumeMount which binds it to a path in the container. Our updated Deployment YAML is as follows:



```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: todo-deployment
  namespace: k8slabo
  labels:
    app: todo
spec:
  selector:
    matchLabels:
      app: todo
  template:
    metadata:
      labels:
        app: todo
    spec:
      hostNetwork: false
      containers:
      - name: todo-webservice
        image: togoetha/todoservice
        ports:
        - containerPort: 8080
        resources:
          requests:
            memory: "50Mi"
            cpu: "100m"
          limits:
            memory: "100Mi"
            cpu: "200m"
      - name: logger
        image: togoetha/logservice
        resources:
          requests:
            memory: "20Mi"
            cpu: "50m"
          limits:
            memory: "50Mi"
            cpu: "100m"
        volumeMounts:
          - mountPath: /logs
            name: todo-volume
      nodeSelector:
        "labo/todoservice": "true"
      volumes:
      - name: todo-volume
        hostPath:
          # directory location on host
          path: /logs
          # this field is optional
          type: Directory
```

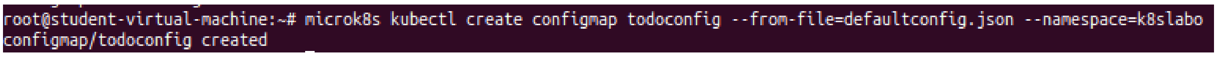
Verifying that everything was deployed correctly:



```
root@student-virtual-machine:~# microk8s kubectl apply -f todo-deployment.yaml
deployment.apps/todo-deployment created
root@student-virtual-machine:~# microk8s kubectl get deployment -o wide
No resources found in default namespace.
root@student-virtual-machine:~# microk8s kubectl get deployment -o wide --namespace=k8slabo
NAME              READY   UP-TO-DATE   AVAILABLE   AGE   CONTAINERS               IMAGES
   SELECTOR
todo-deployment   1/1     1            1           7s    todo-webservice,logger   togoetha/todos
ervice,togoetha/logservice   app=todo
```

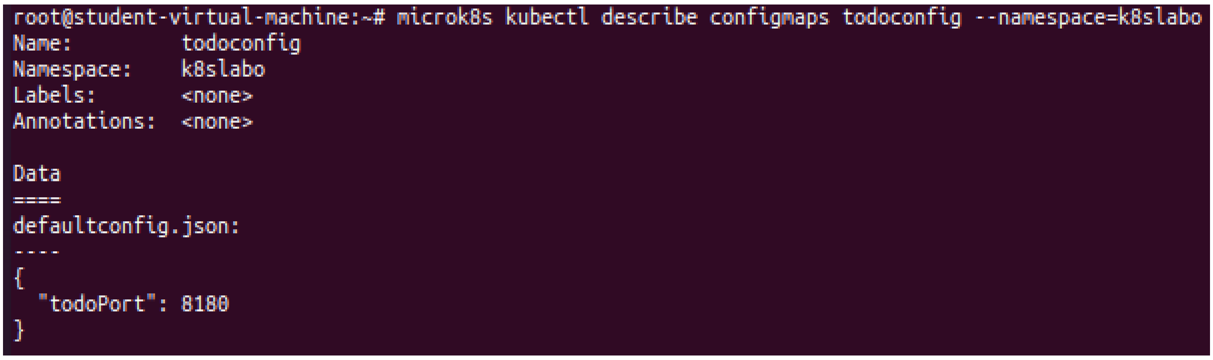
8. Configuring services via ConfigMap

We first create the ConfigMap, which will assign the todo service to port 8180 rather than 8080, and then import this file as the ConfigMap “todoconfig” in Kubernetes:



```
root@student-virtual-machine:~# microk8s kubectl create configmap todoconfig --from-file=defaultconfig.json --namespace=k8slabo
configmap/todoconfig created
```

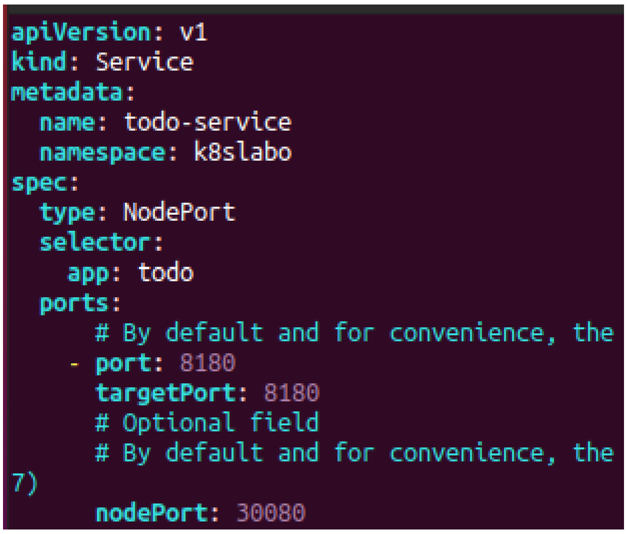
We can verify that the ConfigMap was created using following command:



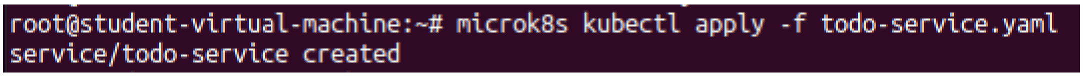
```
root@student-virtual-machine:~# microk8s kubectl describe configmaps todoconfig --namespace=k8slabo
Name:         todoconfig
Namespace:    k8slabo
Labels:       <none>
Annotations:  <none>

Data
====
defaultconfig.json:
----
{
  "todoPort": 8180
}
```

We must change our NodePort service to use the right port. The updated YAML is as follows:

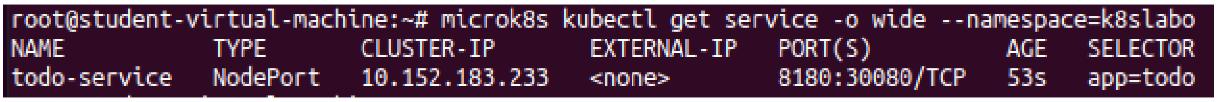


```
apiVersion: v1
kind: Service
metadata:
  name: todo-service
  namespace: k8slabo
spec:
  type: NodePort
  selector:
    app: todo
  ports:
      # By default and for convenience, the
    - port: 8180
      targetPort: 8180
      # Optional field
      # By default and for convenience, the
7)
      nodePort: 30080
```



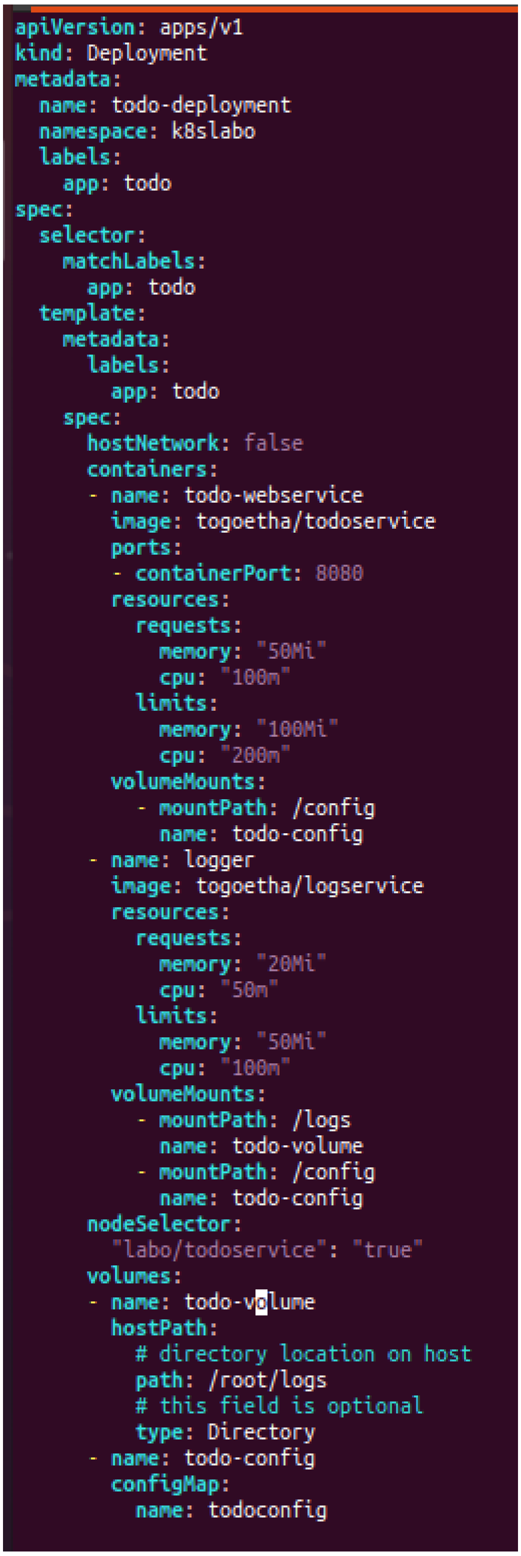
```
root@student-virtual-machine:~# microk8s kubectl apply -f todo-service.yaml
service/todo-service created
```

We verify that the service is indeed running on port 8180:



```
root@student-virtual-machine:~# microk8s kubectl get service -o wide --namespace=k8slabo
NAME           TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE   SELECTOR
todo-service   NodePort   10.152.183.233  <none>        8180:30080/TCP   53s   app=todo
```

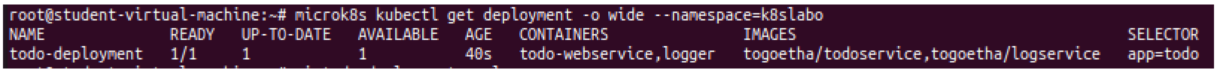
We will now add a ConfigMap volume to our Deployment and add a volumeMount to /config in both containers to use it. We must ensure that the configMap volume’s name is the same as the ConfigMap we created using **microk8s kubectl create configmap todoconfig --….**



```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: todo-deployment
  namespace: k8slabo
  labels:
    app: todo
spec:
  selector:
    matchLabels:
      app: todo
  template:
    metadata:
      labels:
        app: todo
    spec:
      hostNetwork: false
      containers:
      - name: todo-webservice
        image: togoetha/todoservice
        ports:
        - containerPort: 8080
        resources:
          requests:
            memory: "50Mi"
            cpu: "100m"
          limits:
            memory: "100Mi"
            cpu: "200m"
        volumeMounts:
          - mountPath: /config
            name: todo-config
      - name: logger
        image: togoetha/logservice
        resources:
          requests:
            memory: "20Mi"
            cpu: "50m"
          limits:
            memory: "50Mi"
            cpu: "100m"
        volumeMounts:
          - mountPath: /logs
            name: todo-volume
          - mountPath: /config
            name: todo-config
      nodeSelector:
        "labo/todoservice": "true"
      volumes:
      - name: todo-volume
        hostPath:
          # directory location on host
          path: /root/logs
          # this field is optional
          type: Directory
      - name: todo-config
        configMap:
          name: todoconfig
```

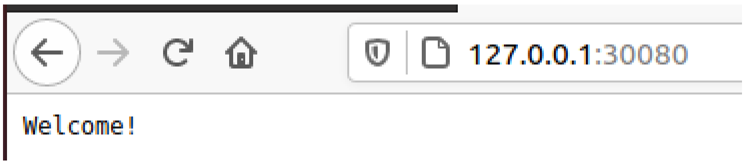
Note the volumes.hostPath.path’s value should be ‘/root/logs’ and not ‘/logs’, because we have to work with absolute paths.

We can verify that our Deployment has started:



```
root@student-virtual-machine:~# microk8s kubectl get deployment -o wide --namespace=k8slabo
NAME              READY   UP-TO-DATE   AVAILABLE   AGE   CONTAINERS               IMAGES                                      SELECTOR
todo-deployment   1/1     1            1           40s   todo-webservice,logger   togoetha/todoservice,togoetha/logservice    app=todo
```

We can reach it via our NodePort:



127.0.0.1:30080

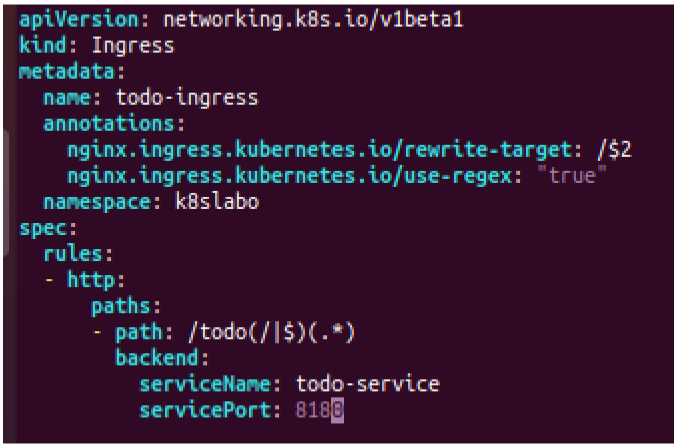
```
Welcome!
```

9. Configuring an ingress

We will now use an ingress with nginx, making our service available on http://localhost/todo. Firstly, we ensure that the ingress plugin is running using the commands:

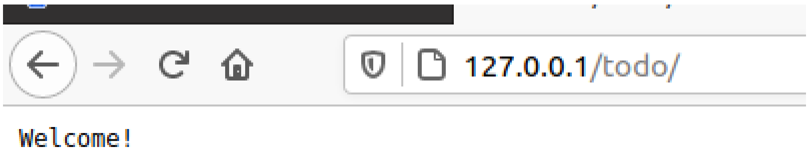
- **microk8s enable ingress**
- **microk8s kubectl create deployment nginx --image=nginx**

We will now create an Ingress “todo-ingress” for the service “todo-service” and map it to the path /todo.



```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: todo-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /$2
    nginx.ingress.kubernetes.io/use-regex: "true"
  namespace: k8slabo
spec:
  rules:
  - http:
      paths:
      - path: /todo(/|$)(.*)
        backend:
          serviceName: todo-service
          servicePort: 8180
```

We can verify that our ingress works correctly by navigating to 127.0.0.1/todo. It shows us our expected ‘Welcome!’ page.

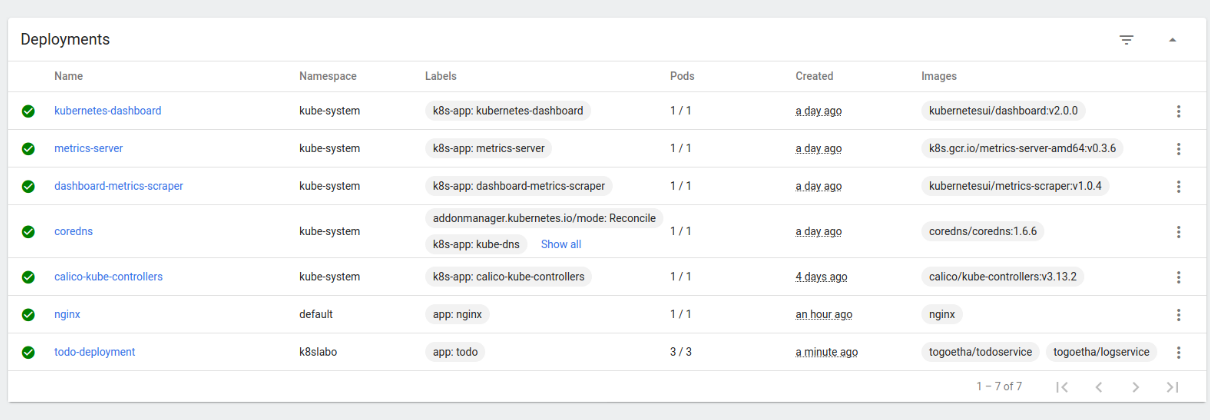


127.0.0.1/todo/

```
Welcome!
```

10. Metrics and scaling

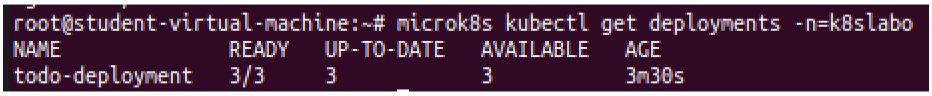
We now wish to add scalability to our Deployment. One way of doing this is using the replicas field in our Deployment YAML. Indeed, after redeploying our Deployment, we can see that there are now 3/3 pods instead of 1/1.



Deployments

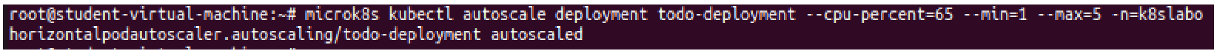
	Name	Namespace	Labels	Pods	Created	Images
✓	kubernetes-dashboard	kube-system	k8s-app: kubernetes-dashboard	1 / 1	a day ago	kubernetesui/dashboard:v2.0.0
✓	metrics-server	kube-system	k8s-app: metrics-server	1 / 1	a day ago	k8s.gcr.io/metrics-server-amd64:v0.3.6
✓	dashboard-metrics-scraper	kube-system	k8s-app: dashboard-metrics-scraper	1 / 1	a day ago	kubernetesui/metrics-scraper:v1.0.4
✓	coredns	kube-system	addonmanager.kubernetes.io/mode: Reconcile k8s-app: kube-dns Show all	1 / 1	a day ago	coredns/coredns:1.6.6
✓	calico-kube-controllers	kube-system	k8s-app: calico-kube-controllers	1 / 1	4 days ago	calico/kube-controllers:v3.13.2
✓	nginx	default	app: nginx	1 / 1	an hour ago	nginx
✓	todo-deployment	k8slabo	app: todo	3 / 3	a minute ago	togoetha/todoservice togoetha/logservice

We can also view the number of replicas using the following command:



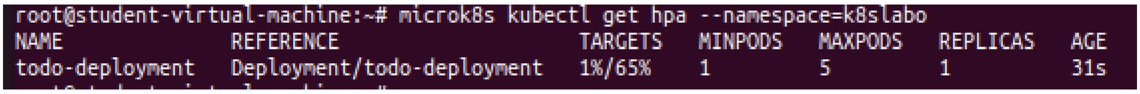
```
root@student-virtual-machine:~# microk8s kubectl get deployments -n=k8slabo
NAME              READY   UP-TO-DATE   AVAILABLE   AGE
todo-deployment   3/3     3            3           3m30s
```

However, this sort of static scaling could be overwhelmed by sudden spikes or waste resources by keeping instances up when they are not needed. We will now create a basic autoscaler which creates a minimum of 1 and a maximum of 5 instances, creating a new instance when the deployments go over 65% CPU use. We can do so by running the following command:



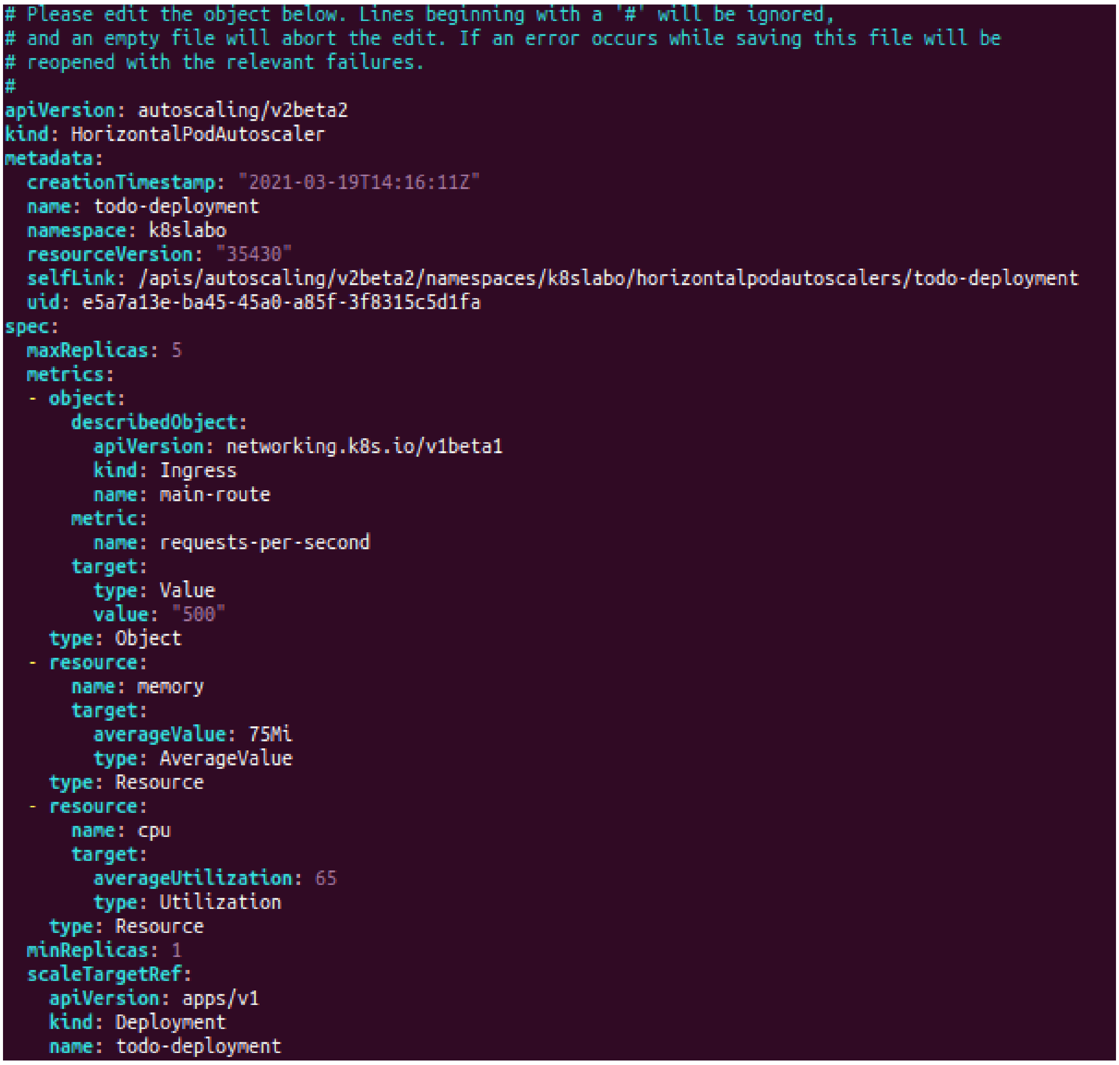
```
root@student-virtual-machine:~# microk8s kubectl autoscale deployment todo-deployment --cpu-percent=65 --min=1 --max=5 -n=k8slabo
horizontalpodautoscaler.autoscaling/todo-deployment autoscaled
```

We can confirm that the autoscaler has successfully deployed using the following command:

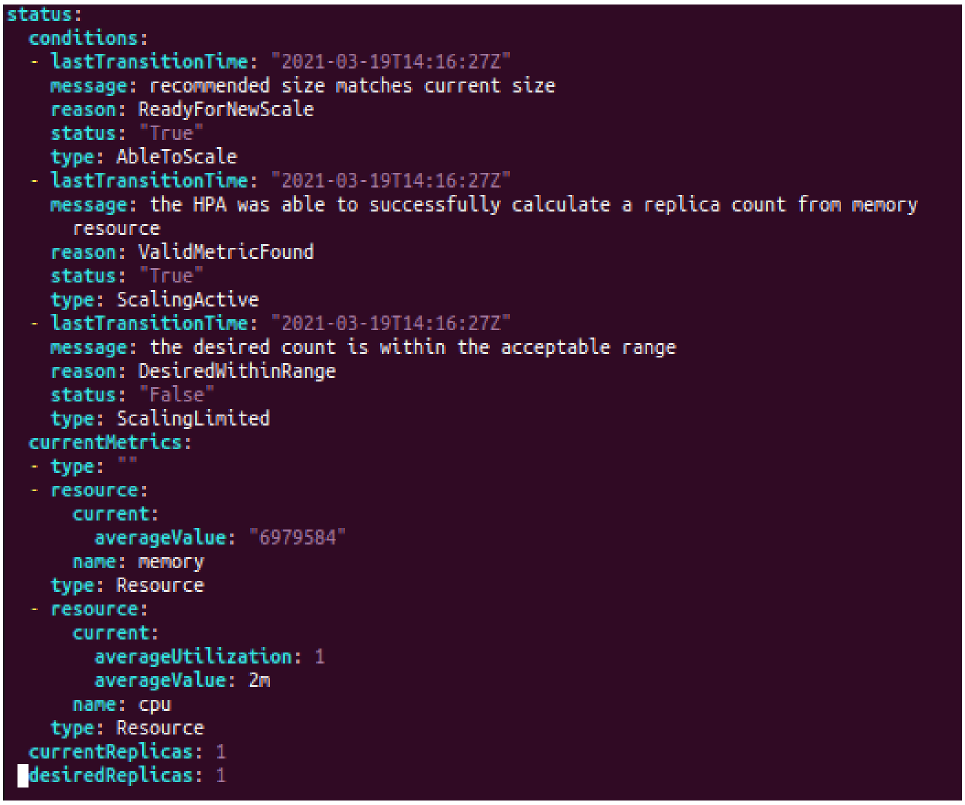


```
root@student-virtual-machine:~# microk8s kubectl get hpa --namespace=k8slabo
NAME              REFERENCE                    TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
todo-deployment   Deployment/todo-deployment   1%/65%    1         5         1          31s
```

We will now expand our autoscaler by adding a few more parameters. We want to extend the autoscaler so it scales up if a pod starts getting more than 500 requests per second, or if more than 75Mi of memory is used. Indeed, as can be seen in the autoscaler’s YAML on the next page, a requests per second metric and a memory metric were added alongside the cpu metric.



```
# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this file will be
# reopened with the relevant failures.
#
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  creationTimestamp: "2021-03-19T14:16:11Z"
  name: todo-deployment
  namespace: k8slabo
  resourceVersion: "35430"
  selfLink: /apis/autoscaling/v2beta2/namespaces/k8slabo/horizontalpodautoscalers/todo-deployment
  uid: e5a7a13e-ba45-45a0-a85f-3f8315c5d1fa
spec:
  maxReplicas: 5
  metrics:
  - object:
      describedObject:
        apiVersion: networking.k8s.io/v1beta1
        kind: Ingress
        name: main-route
      metric:
        name: requests-per-second
      target:
        type: Value
        value: "500"
    type: Object
  - resource:
      name: memory
      target:
        averageValue: 75Mi
        type: AverageValue
    type: Resource
  - resource:
      name: cpu
      target:
        averageUtilization: 65
        type: Utilization
    type: Resource
  minReplicas: 1
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: todo-deployment
```



```
status:
  conditions:
  - lastTransitionTime: "2021-03-19T14:16:27Z"
    message: recommended size matches current size
    reason: ReadyForNewScale
    status: "True"
    type: AbleToScale
  - lastTransitionTime: "2021-03-19T14:16:27Z"
    message: the HPA was able to successfully calculate a replica count from memory
      resource
    reason: ValidMetricFound
    status: "True"
    type: ScalingActive
  - lastTransitionTime: "2021-03-19T14:16:27Z"
    message: the desired count is within the acceptable range
    reason: DesiredWithinRange
    status: "False"
    type: ScalingLimited
  currentMetrics:
  - type: ""
  - resource:
      current:
        averageValue: "6979584"
      name: memory
    type: Resource
  - resource:
      current:
        averageUtilization: 1
        averageValue: 2m
      name: cpu
    type: Resource
  currentReplicas: 1
  desiredReplicas: 1
```