

Criação do namespace do projeto:

```
kubectl create ns k8sjob
```

Criação do Manifesto de Deploy:

```
kubectl create deployment my-app --image nginx:1.27 --namespace k8sjob  
--dry-run=client -o yaml > deployment.yaml
```

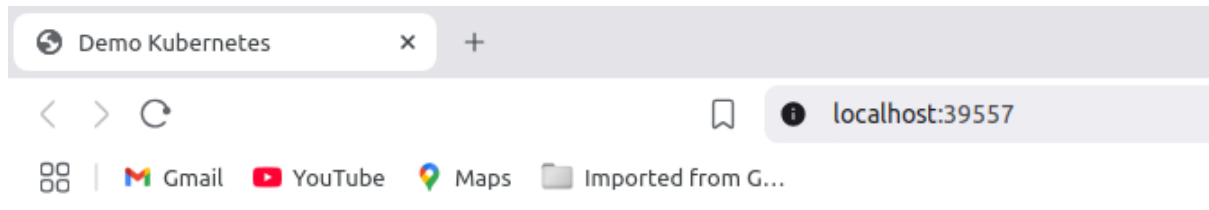
Adicionando limits e requests de cpu e memória no deployment:

```
resources:  
  limits:  
    memory: 300Mi  
  requests:  
    cpu: 10m  
    memory: 100Mi
```

Criando configmap com a página estática e mapeando como volume no deployment:

```
kubectl -n k8sjob create configmap nginx-html --from-file=index.html
```

```
labels:  
  app: my-app  
spec:  
  containers:  
  - image: nginx:1.27  
    name: nginx  
    ports:  
    - containerPort: 80  
    volumeMounts:  
    - name: html-volume  
      mountPath: /usr/share/nginx/html/index.html  
      subPath: index.html  
  resources:  
    limits:  
      memory: 300Mi  
    requests:  
      cpu: 10m  
      memory: 100Mi  
  volumes:  
  - name: html-volume  
    configMap:  
      name: nginx-html  
status: {}
```



Aplicação Rodando no Kubernetes!

Esta página foi servida pelo NGINX com conteúdo vindo de um ConfigMap.

Criando volume do tipo hostPath e mapeando o volume para armazenar os logs do nginx:

```
ports:
  - containerPort: 80
volumeMounts:
  - name: html-volume
    mountPath: /usr/share/nginx/html/index.html
    subPath: index.html
  - name: log-volume
    mountPath: /var/log/nginx
resources:
  limits:
    memory: 300Mi
  requests:
    cpu: 10m
    memory: 100Mi
volumes:
  - name: html-volume
    configMap:
      name: nginx-html
  - name: log-volume
    hostPath:
      path: /mnt/data/nginx-logs
      type: DirectoryOrCreate
status: {}
```

Aplicando o deployment:

```
kubectl apply -f deployment.yaml
```

Criando o service:

```
kubectl -n k8sjob expose deployment my-app
```

Aplicando o metrics server no cluster para disponibilizar as métricas de cpu e memória para o Horizontal Pod Auto Scaler:

```
kubectl apply -f  
https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml
```

Ajuste do metrics server para funcionar no cluster do tipo kind:

```
kubectl -n kube-system patch deployment metrics-server \\\n  --type=json \\\n  -p='[{"op": "add", "path": "/spec/template/spec/containers/0/args/-", "value": \\\n  "--kubelet-insecure-tls"}]'\nkubectl -n kube-system patch deployment metrics-server \\\n  --type=json \\\n  -p='[{"op": "add", "path": "/spec/template/spec/containers/0/args/-", "value": \\\n  "--kubelet-preferred-address-types=InternalIP"}]'
```

Aplicando o Horizontal Pod Auto Scaler:

```
apiVersion: autoscaling/v2\nkind: HorizontalPodAutoscaler\nmetadata:\n  name: my-app-hpa\n  namespace: k8sjob\nspec:\n  scaleTargetRef:\n    apiVersion: apps/v1\n    kind: Deployment\n    name: my-app\n  minReplicas: 1\n  maxReplicas: 3\n  metrics:\n    - type: Resource\n      resource:\n        name: cpu\n        target:\n          type: Utilization\n          averageUtilization: 50
```

kubectl apply -f hpa.yaml

Criação de um pod para testar o auto scaling da aplicação:

```
kubectl run -i --tty load-generator --image=busybox --restart=Never -- sh
```

Rodando Script de carga para aumentar o consumo de cpu e ativar o auto scaling:

```
while true; do wget -q -O- http://my-app.k8sjob.svc.cluster.local:8080; done
```

Evidência do auto scaling funcionando:

```
kubectl -n k8sjob get hpa
```

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
my-app-hpa	Deployment/my-app	cpu: 230%/50%	1	3	3	16m

