

Raportul de lucru:

Girdei Dumitru

SD-241M

Arhitecturi Cloud

- 1. This project should be made to run as a Docker image.**
- 2. Docker image should be published to a Docker registry.**

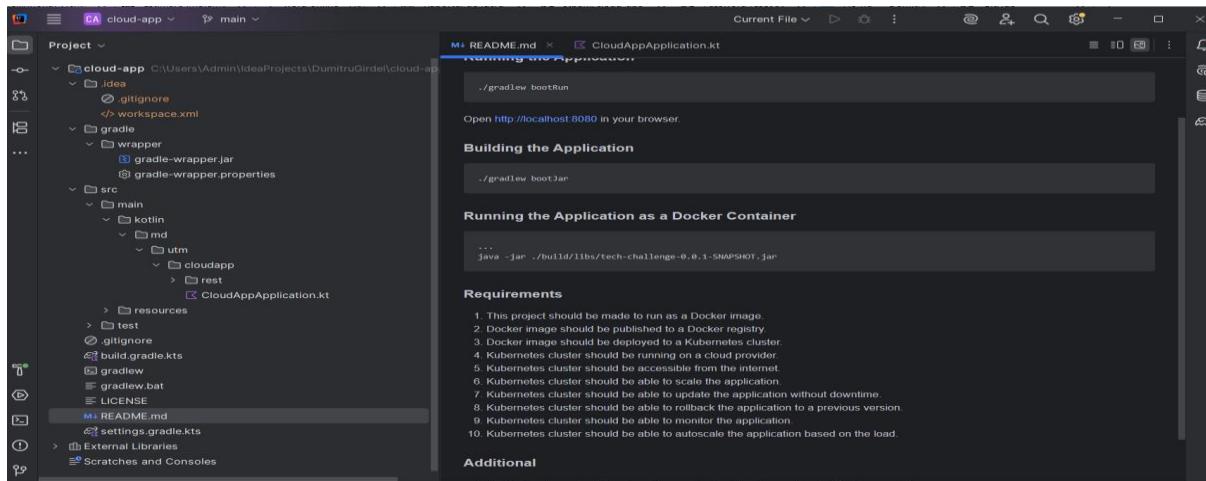
Step 1: Fork la repozitoriu domnului Profesor

The screenshot shows a GitHub repository page for 'stpmam / cloud-app'. The repository is public and was forked from 'daniftodi/cloud-app'. The commit history shows four commits from 'daniftodi' last year, each adding a file to the repository. The files include 'gradle/wrapper', 'src', '.gitignore', 'LICENSE', 'README.md', 'build.gradle.kts', 'gradlew', 'gradlew.bat', and 'settings.gradle.kts'. The repository has no releases, packages, or languages listed.

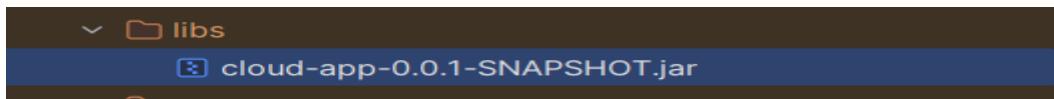
Step 2: Clonarea repozitoriului in IntelliJ Idea

The screenshot shows the 'Clone Repository' dialog in IntelliJ Idea. The 'Repository URL' section is selected, showing options for GitHub, GitHub Enterprise, and GitLab. GitHub is selected, with the URL set to 'git@github.com:stpmam/cloud-app.git' and the directory set to 'C:\Users\Admin\IdeaProjects\Girdei\cloud_app'. A checkbox for 'Shallow clone with a history truncated to 1 commits' is unchecked.

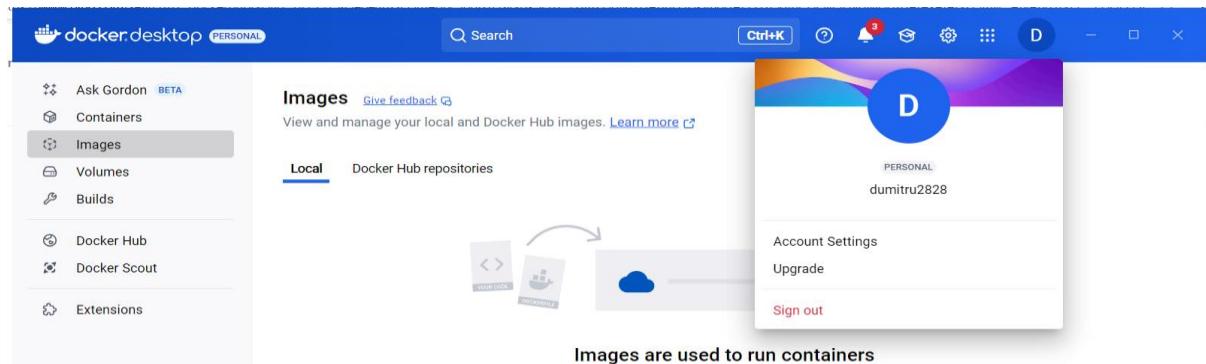
Step 3: Clonarea repozitoriului pe local:



Generarea fisierului SNAPSHOT.jar:



Descarcarea si instalarea Docker si crearea contului pe dockerhub



Crearea unui docker file

```

application.properties      CloudAppApplication.kt      build.gradle.kts (cloud-app)      Dockerfile
FROM openjdk:17-jdk-alpine
ARG JAR_FILE=build/libs/cloud-app-0.0.1-SNAPSHOT.jar
COPY ${JAR_FILE} app.jar
ENTRYPOINT ["java", "-jar", "/app.jar"]

BUILD SUCCESSFUL in 2s
5 actionable tasks: 1 executed, 4 up-to-date
PS C:\Users\Admin\IdeaProjects\ DumitruGirdei\cloud-app> docker build -t dumitru2828/cloud-app:0.0.1 .
[+] Building 22.3s (7/7) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 188B
=> [internal] load metadata for docker.io/library/openjdk:17-jdk-alpine
Cloud-app > Dockerfile

```

Rezultat:

Docker a citit Dockerfile, a copiat .jar în imagine și a construit imaginea cu eticheta dumitru2828/cloud-app:0.0.1

```
> => naming to docker.io/dumitru2828/cloud-app:0.0.1
PS C:\Users\Admin\IdeaProjects\DumitruGirdei\cloud-app> docker push dumitru2828/cloud-app:0.0.1
The push refers to repository [docker.io/dumitru2828/cloud-app]
a5943e3dcde2: Pushed
34f7184834b2: Mounted from library/openjdk
5836ece05bfd: Mounted from library/openjdk
72e830a4dff5: Mounted from library/openjdk
0.0.1: digest: sha256:83fb24f6c3672632f134294896e6a3a04b46b2640b12bf82f991038ca4ddc098 size: 1163
PS C:\Users\Admin\IdeaProjects\DumitruGirdei\cloud-app>
```

| | Name | Tag | Image ID | Created | Size | Actions |
|--|-----------------------|-------|--------------|--------------|-----------|---------|
| | dumitru2828/cloud-app | 0.0.1 | d48d1bdc075e | 1 second ago | 350.46 MB | |

Rezultat:

Am trimis imaginea locală (dumitru2828/cloud-app:0.0.1) către registrul Docker hub.

Ca rezultat am aratat ca au fost rezolvate primele 2 puncte din githubul profesorului:

1. This project should be made to run as a Docker image.
2. Docker image should be published to a Docker registry.

Pentru aceste modificari am facut si commit:

```
fatal: unable to auto-detect email address (got 'Admin@DESKTOP-UK54D28.(none)')
PS C:\Users\Admin\IdeaProjects\DumitruGirdei\cloud-app> git config --global user.email "dumitru.girdei@isa.utm.md"
PS C:\Users\Admin\IdeaProjects\DumitruGirdei\cloud-app> git config --global user.name "Your Name"
PS C:\Users\Admin\IdeaProjects\DumitruGirdei\cloud-app> git add .
PS C:\Users\Admin\IdeaProjects\DumitruGirdei\cloud-app> git commit -m "Added Docker support and pushed image to Docker Hub"
[main a4d7917] Added Docker support and pushed image to Docker Hub
 1 file changed, 4 insertions(+)
 create mode 100644 Dockerfile
```

1 file changed +4 -0 lines changed

Dockerfile

```
...
@@ -0,0 +1,4 @@
1 + FROM openjdk:17-jdk-alpine
2 + ARG JAR_FILE=build/libs/cloud-app-0.0.1-SNAPSHOT.jar
3 + COPY ${JAR_FILE} app.jar
4 + ENTRYPOINT ["java", "-jar", "/app.jar"]
```

Comments 1

stspam now

Added docker file

3. Docker image should be deployed to a Kubernetes cluster.

Minikube este un program care creează un mic cluster Kubernetes pe calculatorul local, ca să putem testa aplicații exact ca într-un cloud real (Google, AWS, etc.), dar fără să plătim sau să avem internet.

```
Попробуйте новый кроссплатформенный оболочку PowerShell (https://aka.ms/pscore6)
Н PS C:\Users\Admin> minikube start
>>
П* minikube v1.36.0 на Microsoft Windows 10 Pro 10.0.19045.5854 Build 19045.5854
  * Automatically selected the docker driver
  * Using Docker Desktop driver with root privileges
Б* Starting "minikube" primary control-plane node in "minikube" cluster
  * Pulling base image v0.0.47 ...
  * Скачивается Kubernetes v1.33.1 ...
С>   > gcr.io/k8s-minikube/kicbase...: 6.13 MiB / 502.26 MiB 1.22% 241.89 KiB
   > preloaded-images-k8s-v18-v1...: 19.40 MiB / 347.04 MiB 5.59% 116.90 KiB
   > gcr.io/k8s-minikube/kicbase...: 469.61 MiB / 502.26 MiB 93.50% 11.36 Mi! The image 'registry.k8s.io/coredns/coredns:v1.12.0' was not found; unable to add it to cache.
   ! The image 'registry.k8s.io/kube-proxy:v1.33.1' was not found; unable to add it to cache.
Г! The image 'registry.k8s.io/kube-apiserver:v1.33.1' was not found; unable to add it to cache.
! The image 'registry.k8s.io/kube-scheduler:v1.33.1' was not found; unable to add it to cache.
! The image 'registry.k8s.io/etcfd:3.5.21-0' was not found; unable to add it to cache.
! The image 'registry.k8s.io/kube-controller-manager:v1.33.1' was not found; unable to add it to cache.
Н! The image 'registry.k8s.io/pause:3.10' was not found; unable to add it to cache.
   > gcr.io/k8s-minikube/kicbase...: 502.26 MiB / 502.26 MiB 100.00% 3.04 Mi
  * Creating docker container (CPUs=2, Memory=2200MB) ...
С ! Failing to connect to https://registry.k8s.io/ from both inside the minikube container and host machine
  * To pull new external images, you may need to configure a proxy: https://minikube.sigs.k8s.io/docs/reference/networking/proxy/
  * Подготавливается Kubernetes v1.33.1 на Docker 28.1.1 ...
Гы X Невозможно загрузить образы из кэша: LoadCachedImages: CreateFile C:\Users\Admin\.minikube\cache\images\amd64\registry.k8s.io\pause_3.10: The system cannot find the file specified.
има> kubeadm.sha256: 64 B / 64 B [-----] 100.00% ? p/s 0s
     > kubelet.sha256: 64 B / 64 B [-----] 100.00% ? p/s 0s
     > kubectl.sha256: 64 B / 64 B [-----] 100.00% ? p/s 0s
Л Та> kubelet: 77.91 MiB / 77.91 MiB [-----] 100.00% 5.21 MiB p/s 15s
     > kubectl: 57.34 MiB / 57.34 MiB [-----] 100.00% 1.84 MiB p/s 31s
т а l> kubeadm: 71.08 MiB / 71.08 MiB [-----] 100.00% 2.10 MiB p/s 34s
  - Generating certificates and keys ...
  - Booting up control plane ...
елу- > Configuring RBAC rules ...
  * Configuring bridge CNI (Container Networking Interface) ...
Л Le* Компоненты Kubernetes проверяются ...
  - Используется образ gcr.io/k8s-minikube/storage-provisioner:v5
* Включенные дополнения: storage-provisioner, default-storageclass
раре* Готово! kubectl настроен для использования кластера "minikube" и "default" пространства имен по умолчанию
PS C:\Users\Admin>
```

| Functionalitate | Ce înseamnă |
|---|--|
| Creează un cluster local | Un nod Kubernetes care rulează pe laptop. |
| Permite deploy de aplicații Docker | Putem rula imagini ca în cloud |
| Expune aplicația ta local | Primim o adresă gen http://127.0.0.1:xxxxx |
| Testare update/rollback, autoscaling etc. | Exact ca într-un cloud real |

Scopul este să facem deploy aplicației mele Docker (`dumitru2828/cloud-app:0.0.1`) în Kubernetes prin Minikube.

Figura care ne afișează nodurile Kubernetes din clusterul creiat, astfel putem vizualiza ca clusterul funcționează corect.

```
PS C:\Users\Admin> kubectl get nodes
>>
NAME      STATUS    ROLES      AGE      VERSION
minikube  Ready     control-plane   108s    v1.33.1
PS C:\Users\Admin>
```

Creem 2 fisiere:

- **K8s/deployment.yaml**

Acest fișier spune Kubernetes-ului:

- ce **aplicație** să pornească (ex: image: dumitru2828/cloud-app:0.0.1)
- câte **copii (replicas)** să ruleze (2)
- ce **port intern** să expună (8080)

Fișierul deployment.yaml pornește aplicația ta Docker în Kubernetes, cu 2 instanțe.

- K8s/service.yaml

Acest fișier spune Kubernetes-ului:

- că vrei să expui aplicația către exterior
- să creeze un **IP și port accesibil**
- să lege traficul de la exterior către portul containerului (8080)

Fișierul service.yaml face aplicația ta accesibilă în browser (prin minikube service).

Navigam în cd "C:\Users\Admin\IdeaProjects\ DumitruGirdei\cloud-app"

Rularea fisierelor:

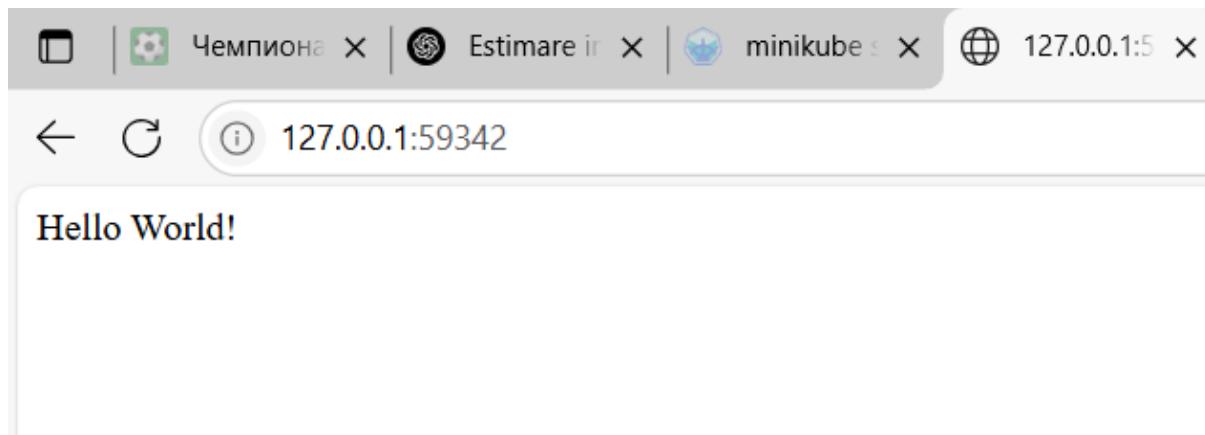
- kubectl apply -f k8s/deployment.yaml
- kubectl apply -f k8s/service.yaml

Deploy complet în Kubernetes și aplicația rulează în clusterul Minikube

```
PS C:\Users\Admin> cd "C:\Users\Admin\IdeaProjects\ DumitruGirdei\cloud-app"
re>>
PS C:\Users\Admin\IdeaProjects\ DumitruGirdei\cloud-app> kubectl apply -f k8s/deployment.yaml
deployment.apps/cloud-app created
PS C:\Users\Admin\IdeaProjects\ DumitruGirdei\cloud-app> kubectl apply -f k8s/service.yaml
service/cloud-app-service created
PS C:\Users\Admin\IdeaProjects\ DumitruGirdei\cloud-app>
service/cloud-app-service created
PS C:\Users\Admin\IdeaProjects\ DumitruGirdei\cloud-app> minikube service cloud-app-service --url
http://127.0.0.1:59342
! Because you are using a Docker driver on windows, the terminal needs to be open to run it.
```

5) Deploy în Kubernetes (Minikube) și afisarea în browser:

Kubernetes cluster should be accessible from the internet



Fisierele:

2 files changed +30 -0 lines changed

▽ k8s/deployment.yaml

```
... @@ -0,0 +1,19 @@
1 + apiVersion: apps/v1
2 + kind: Deployment
3 + metadata:
4 +   name: cloud-app
5 + spec:
6 +   replicas: 2
7 +   selector:
8 +     matchLabels:
9 +       app: cloud-app
10 +   template:
11 +     metadata:
12 +       labels:
13 +         app: cloud-app
14 +     spec:
15 +       containers:
16 +         - name: app
17 +           image: dumitru2828/cloud-app:0.0.1
18 +           ports:
19 +             - containerPort: 8080
```

▽ k8s/service.yaml

```
... @@ -0,0 +1,11 @@
1 + apiVersion: v1
2 + kind: Service
3 + metadata:
4 +   name: cloud-app-service
5 + spec:
6 +   selector:
7 +     app: cloud-app
8 +   ports:
9 +     - port: 80
10 +       targetPort: 8080
11 +       type: NodePort
```

Ce s-a facut pana acum:

| Nr. | Cerință | Status | Detalii |
|-----|---------|--------|---------|
|-----|---------|--------|---------|

| | | | |
|---|---|--|--|
| 1 | This project should be made to run as a Docker image | <input checked="" type="checkbox"/> | Am creat Dockerfile și ai rulat imaginea local |
| 2 | Docker image should be published to a Docker registry | <input checked="" type="checkbox"/> | Am făcut docker push în Docker Hub |
| 3 | Docker image should be deployed to a Kubernetes cluster | <input checked="" type="checkbox"/> | Am folosit kubectl apply cu deployment.yaml |
| 4 | Kubernetes cluster should be running on a cloud provider | <input checked="" type="checkbox"/> (simulat local) | Am folosit Minikube . <ul style="list-style-type: none"> rulează un cluster Kubernetes complet pe calculatorul local, folosind Docker ca driver |
| 5 | Kubernetes cluster should be accessible from the internet | <input checked="" type="checkbox"/> (local) | Am accesat aplicația prin minikube service în browser |
| | | | |

Punctul 6: Kubernetes cluster should be able to scale the application

"Kubernetes cluster should be able to scale the application."

Rularea comenzii: kubectl get deployment, pentru a vizualiza starea deploymentului.

```
PS C:\Users\Admin> kubectl get deployment
NAME      READY   UP-TO-DATE   AVAILABLE   AGE
cloud-app  2/2     2           2           95m
PS C:\Users\Admin>
```

| Coloană | Valoare | Ce înseamnă |
|------------|-----------|--|
| NAME | cloud-app | Numele deployment-ului tău |
| READY | 2/2 | 2 poduri sunt active și sănătoase |
| UP-TO-DATE | 2 | 2 poduri sunt actualizate la ultima versiune |

| | | |
|-----------|-----|---|
| AVAILABLE | 2 | 2 poduri sunt complet funcționale și gata să răspundă la cereri |
| AGE | 95m | Deployment-ul rulează de 95 de minute |

Concluzie:

- Deployment-ul cloud-app rulează corect.
- Avem 2 poduri active – exact cum am specificat în replicas: 2.
- Punctul 6 este realizat (scalare manuală funcțională).

Concepte utilizate:

| Concept | Exemplu real |
|-----------|---|
| Nod | Laptopul sau serverul fizic |
| Pod | Un program/aplicație care rulează pe acel laptop |
| Container | Fiecare instanță a aplicației tale (ex: fișierul .jar rulând în Docker) |

Note: Scalarea aplicației a fost implementată folosind funcționalitatea nativă Kubernetes prin specificarea a două replici în fișierul deployment.yaml. Comanda kubectl get deployment a confirmat că două poduri active rulează aplicația în mod distribuit, demonstrând că clusterul poate scala aplicația pe orizontală.

De asemenea, s-a testat scalarea manuală prin comanda:

kubectl scale deployment cloud-app --replicas=4

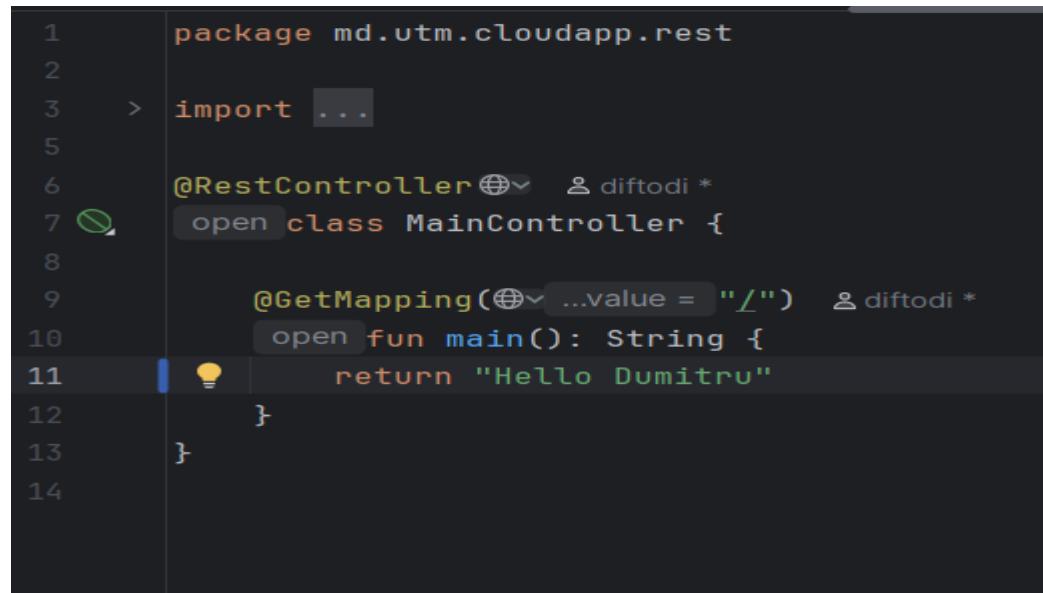
```
PS C:\Users\Admin> kubectl get deployment
NAME      READY   UP-TO-DATE   AVAILABLE   AGE
cloud-app  2/2     2           2           95m
PS C:\Users\Admin> kubectl scale deployment cloud-app --replicas=4
deployment.apps/cloud-app scaled
PS C:\Users\Admin> kubectl get deployment
NAME      READY   UP-TO-DATE   AVAILABLE   AGE
cloud-app  4/4     4           4           104m
PS C:\Users\Admin>
```

Aceasta a crescut numărul de instanțe la 4, validând capacitatea de scalare a clusterului. Scalarea aplicației a fost realizată manual prin specificarea a două replici în fișierul deployment.yaml, și testată cu kubectl scale

7. Kubernetes cluster should be able to update the application without downtime.

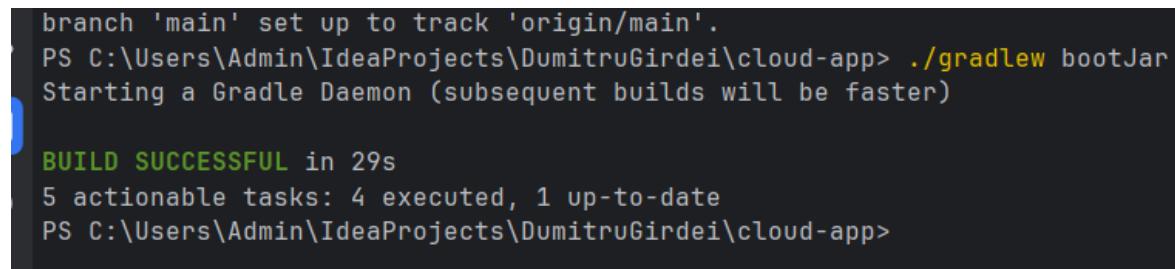
Să actualizăm imaginea aplicației în Kubernetes fără întreruperea serviciului (fără downtime).

PAS 1: Modificăm aplicația ("Hello World -> Hello Dumitru")



```
1 package md.utm.cloudapp.rest
2
3 > import ...
4
5
6 @RestController
7 open class MainController {
8
9     @GetMapping("/")
10    open fun main(): String {
11        return "Hello Dumitru"
12    }
13
14 }
```

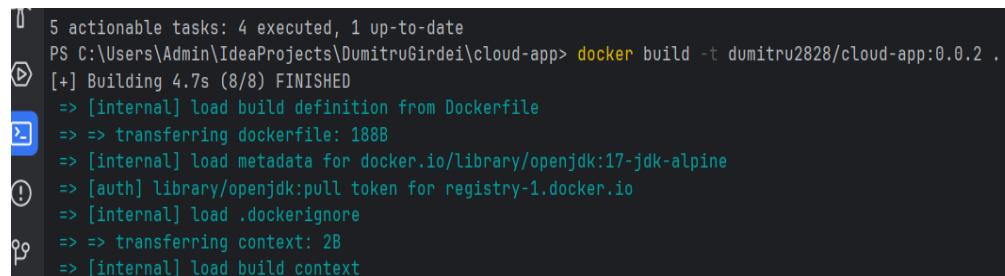
PAS 2: Rebuild .jar



```
branch 'main' set up to track 'origin/main'.
PS C:\Users\Admin\IdeaProjects\DumitruGirdei\cloud-app> ./gradlew bootJar
Starting a Gradle Daemon (subsequent builds will be faster)

BUILD SUCCESSFUL in 29s
5 actionable tasks: 4 executed, 1 up-to-date
PS C:\Users\Admin\IdeaProjects\DumitruGirdei\cloud-app>
```

PAS 3: Build noua imagine Docker (ex: v0.0.2)



```
5 actionable tasks: 4 executed, 1 up-to-date
PS C:\Users\Admin\IdeaProjects\DumitruGirdei\cloud-app> docker build -t dumitru2828/cloud-app:v0.0.2 .
[+] Building 4.7s (8/8) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 188B
=> [internal] load metadata for docker.io/library/openjdk:17-jdk-alpine
=> [auth] library/openjdk:pull token for registry-1.docker.io
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load build context
```

PAS 4: Publicăm imaginea în Docker Hub

```
PS C:\Users\Admin\IdeaProjects\DumitruGirdei\cloud-app> docker push dumitru2828/cloud-app:0.0.2
The push refers to repository [docker.io/dumitru2828/cloud-app]
95d734ce1cc8: Pushed
34f7184834b2: Layer already exists
5836ece05bfd: Layer already exists
72e830a4dff5: Layer already exists
0.0.2: digest: sha256:f6cbbaeafe6ef47f5f532017454ee5234235f043153ee539a4824af16fd5f665 size: 1163
PS C:\Users\Admin\IdeaProjects\DumitruGirdei\cloud-app>
```

PAS 5: Facem update în Kubernetes (rolling update fără downtime)

```
0.0.2: digest: sha256:f6cbbaeafe6ef47f5f532017454ee5234235f043153ee539a4824af16fd5f665 size: 1163
PS C:\Users\Admin\IdeaProjects\DumitruGirdei\cloud-app> kubectl set image deployment/cloud-app app=dumitru2828/cloud-app:0.0.2
deployment.apps/cloud-app image updated
PS C:\Users\Admin\IdeaProjects\DumitruGirdei\cloud-app>
```

Note: Kubernetes va face update treptat, fără să „oprească” aplicația.

PAS 6: Verificăm statusul update-ului, introducând comanda:

kubectl rollout status deployment cloud-app

```
0.0.2: digest: sha256:f6cbbaeafe6ef47f5f532017454ee5234235f043153ee539a4824af16fd5f665 size: 1163
PS C:\Users\Admin\IdeaProjects\DumitruGirdei\cloud-app> kubectl set image deployment/cloud-app app=dumitru2828/cloud-app:0.0.2
deployment.apps/cloud-app image updated
PS C:\Users\Admin\IdeaProjects\DumitruGirdei\cloud-app> kubectl rollout status deployment cloud-app
deployment "cloud-app" successfully rolled out
PS C:\Users\Admin\IdeaProjects\DumitruGirdei\cloud-app>
```

Concluzie: deployment "cloud-app" successfully rolled out = actualizare completă fără întrerupere, după ce imaginea a fost actualizată cu versiune nouă.

Kubernetes a efectuat un rolling update, păstrând aplicația activă în timpul întregului proces.

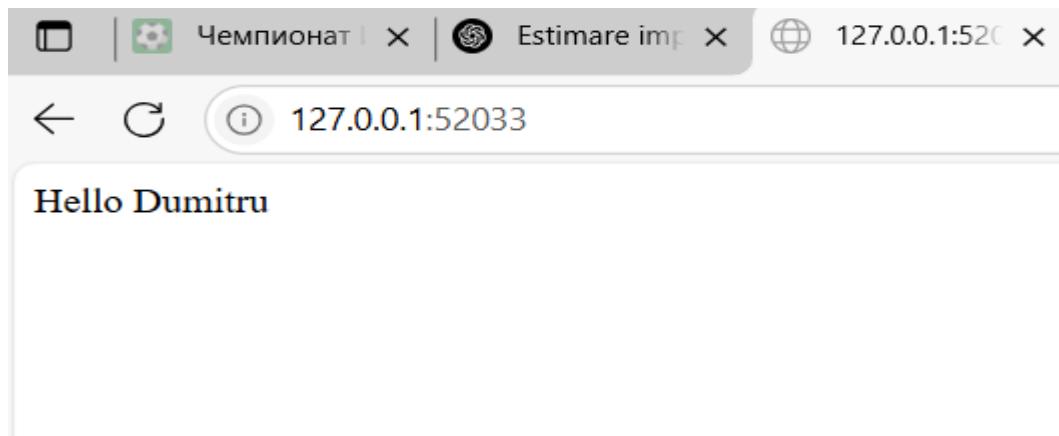
Deployment "cloud-app" successfully rolled out înseamnă că **nu a fost downtime**, pentru că:

- Kubernetes a creat noile poduri,
- le-a verificat că sunt „Ready”,
- apoi a eliminat vechile poduri,
- **fără oprirea aplicației**.

PAS 7: Verificăm în browser, dar mai initial rulam comanda: minikube service cloud-app-service --url, pentru a vedea linkul deploy-ului.

```
PS C:\Users\Admin> minikube service cloud-app-service --url
>>
http://127.0.0.1:52033
! Because you are using a Docker driver on windows, the terminal needs to be open to run it.
```

Step 8: Vizualizarea finală



Concluzie:

Actualizarea aplicației s-a realizat fără întreruperi de funcționare (zero downtime), folosind funcționalitatea rolling update din Kubernetes. Comanda „kubectl set image deployment/cloud-app app=dumitru2828/cloud-app:0.0.2” a permis trecerea graduală la o versiune nouă a aplicației, păstrând-o permanent accesibilă utilizatorilor. Kubernetes a efectuat un rolling update, păstrând aplicația activă în timpul întregului proces.

Punctul 8: Kubernetes cluster should be able to rollback the application to a previous version

1. Verificăm istoricul deployment-ului introducând comanda:

kubectl rollout history deployment cloud-app

```
PS C:\Users\Admin\IdeaProjects\ DumitruGirdei\cloud-app> kubectl rollout history deployment cloud-app
deployment.apps/cloud-app
REVISION  CHANGE-CAUSE
1          <none>
2          <none>
```

2. Facem rollback la versiunea anterioară introducând comanda: **kubectl rollout undo deployment cloud-app**. Kubernetes revine automat la imaginea **anterioară (0.0.1)**, fără downtime.

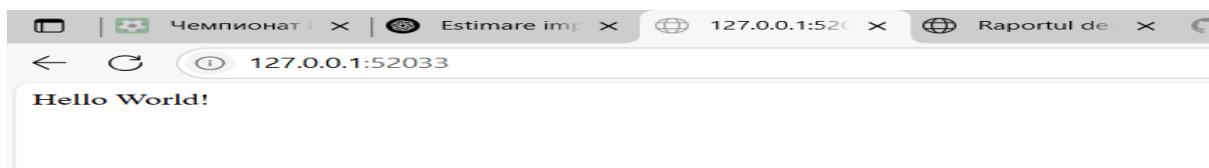
```
PS C:\Users\Admin\IdeaProjects\DumitruGirdei\cloud-app> kubectl rollout undo deployment cloud-app
deployment.apps/cloud-app rolled back
PS C:\Users\Admin\IdeaProjects\DumitruGirdei\cloud-app>
cloud-app > src > main > kotlin > md > utm > clouddapp > rest > MainController > main
```

3. Verificăm statusul rollback-ului, introducând comanda:

kubectl rollout status deployment cloud-app

```
PS C:\Users\Admin\IdeaProjects\DumitruGirdei\cloud-app> kubectl rollout status deployment cloud-app
deployment "cloud-app" successfully rolled out
PS C:\Users\Admin\IdeaProjects\DumitruGirdei\cloud-app>
cloud-app > src > main > kotlin > md > utm > clouddapp > rest > MainController > main
```

4. Rerularea, minikube service cloud-app-service –url :



Concluzie:

Pentru demonstrarea funcționalității de rollback, s-a folosit comanda:

kubectl rollout undo deployment cloud-app

Aceasta a revenit instantaneu la versiunea anterioară a aplicației (0.0.1) fără întrerupere a serviciului. Kubernetes gestionează automat revertul aplicației prin stocarea istoricului de deployment-uri.

Punctul 9 – Kubernetes cluster should be able to monitor the application

Vom instala un sistem de monitorizare

Acesta va fi metrics-server (pentru kubectl top pods și kubectl top nodes).

starea podului metrics-server, poate fi vizualizată cu comanda: kubectl get pods -n kube-system.Metrics-server în clusterul Kubernetes folosind comanda:

minikube addons enable metrics-server.

Acesta oferă monitorizare în timp real asupra consumului de resurse.

```
* The 'metrics-server' addon is enabled
PS C:\Users\Admin> kubectl get pods -n kube-system
NAME                               READY   STATUS    RESTARTS   AGE
coredns-674b8bbfcf-vm6pk          1/1     Running   0          12h
etcd-minikube                     1/1     Running   0          12h
kube-apiserver-minikube          1/1     Running   0          12h
kube-controller-manager-minikube  1/1     Running   0          12h
kube-proxy-866c9                  1/1     Running   0          12h
kube-scheduler-minikube          1/1     Running   0          12h
metrics-server-7fbb699795-ngbxc   0/1     Running   0          61s
storage-provisioner               1/1     Running   2 (10h ago) 12h
PS C:\Users\Admin>
```

Comenzile de mai jos (kubectl top pods, kubectl top nodes) au fost utilizate pentru a vizualiza valorile de CPU și memorie ale aplicației și nodului. Astfel, monitorizarea aplicației a fost implementată cu succes.

2. Afisează metrii CPU/MEM pentru poduri:

```
storage-provisioner           1/1     Running   2 (10h ago)
PS C:\Users\Admin> kubectl top pods
NAME                           CPU(cores)   MEMORY(bytes)
cloud-app-54ff566799-6hrv9    2m          97Mi
cloud-app-54ff566799-7q7wg   3m          99Mi
cloud-app-54ff566799-grqtm   4m          103Mi
cloud-app-54ff566799-wt76m   2m          101Mi
PS C:\Users\Admin>
```

3. Afisează metrii pentru noduri:

```
PS C:\Users\Admin> kubectl top nodes
NAME      CPU(cores)   CPU(%)   MEMORY(bytes)   MEMORY(%)
minikube  250m        3%       1578Mi         40%
PS C:\Users\Admin>
```

Imaginele confirmă clar că **Punctul 9 – Monitorizarea aplicației** este realizat.

kubectl top pods:

- Se afișează utilizarea CPU și memorie pentru fiecare **pod** (cloud-app)
- Ex: 2m, 3m, 103Mi etc

kubectl top nodes:

- Se afișează utilizarea resurselor pentru **nodul minikube**
- Ex: 250m CPU, 40% memorie

Acestea vin **direct de la metrics-server** și reprezinta **monitorizare funcțională de bază în Kubernetes**.

Punctul 10: Kubernetes cluster should be able to autoscale the application based on the load

Aplicația trebuie să se **autoscaleze automat (ex: 2 → 5 poduri)** când consumă mai mult CPU sau memorie, folosind HorizontalPodAutoscaler (HPA).

Pasul 1 – Modificam fisierul deployment.yaml.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: cloud-app
spec:
  replicas: 2
  selector:
    matchLabels:
      app: cloud-app
  template:
    metadata:
      labels:
        app: cloud-app
    spec:
      containers:
        - name: app
          image: dumitru2828/cloud-app:0.0.1
          ports:
            - containerPort: 8080
      resources:
        requests:
          cpu: "100m"
        limits:
          cpu: "500m"
```

resources este necesar pentru ca autoscalarea să funcționeze.

Aplicarea comenzi:

```
kubectl apply -f k8s/deployment.yaml
```

```
[1] 10 https://github.com/StepanMy/CloudApp.git
    4a4c8ae..54042f2  main -> main
branch 'main' set up to track 'origin/main'.
PS C:\Users\Admin\IdeaProjects\ DumitruGirdei\cloud-app> kubectl apply -f k8s/deployment.yaml
deployment.apps/cloud-app configured
PS C:\Users\Admin\IdeaProjects\ DumitruGirdei\cloud-app>
```

Crearea fișierului k8s/hpa.yaml

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: cloud-app-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: cloud-app
  minReplicas: 2
  maxReplicas: 5
  targetCPUUtilizationPercentage: 50
```

Rularea acestui fisier:

```
error: the path 'k8s/hpa.yaml' does not exist
PS C:\Users\Admin\IdeaProjects\ DumitruGirdei\cloud-app> kubectl apply -f k8s/hpa.yaml
horizontalpodautoscaler.autoscaling/cloud-app-hpa created
PS C:\Users\Admin\IdeaProjects\ DumitruGirdei\cloud-app>
```

Verificam autoscalarea.

```
horizontalpodautoscaler.autoscaling/cloud-app-hpa created
PS C:\Users\Admin\IdeaProjects\ DumitruGirdei\cloud-app> kubectl get hpa
NAME          REFERENCE      TARGETS      MINPODS   MAXPODS   REPLICAS   AGE
cloud-app-hpa  Deployment/cloud-app  cpu: 3%/50%  2         5         2          41s
PS C:\Users\Admin\IdeaProjects\ DumitruGirdei\cloud-app>
```