

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»
ІНСТИТУТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ ТА
ЕЛЕКТРОННОЇ ІНЖЕНЕРІЇ

Кафедра
інформаційно
комунікаційних
технологій



КУРСОВИЙ ПРОЄКТ

З дисципліни

«Проектування та розгортання інформаційно-комунікаційних
мереж»

Виконав:
Дорожовець В. В.
гр. ІТПА-11

Прийняв:
доцент каф ІКТ
Красько О. В.

Львів 2025

ЗАВДАННЯ НА КУРСОВИЙ ПРОЄКТ

Побудова високо доступної системи з використанням контейнеризації та оркестрації:

- Використання Kubernetes для оркестрації контейнерів.
- Забезпечення високої доступності та масштабованості інформаційних систем.

ЗМІСТ

ВСТУП.....	5
РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ПОБУДОВИ ВИСОКО ДОСТУПНИХ СИСТЕМ З ВИКОРИСТАННЯМ КОНТЕЙНЕРИЗАЦІЇ ТА ОРКЕСТРАЦІЇ	6
1.1 Сутність високої доступності та масштабованості	6
1.2 Основи контейнеризації та її переваги	6
1.3 Архітектура та функціональні можливості систем оркестрації	7
1.4 Структура та взаємодія компонентів сучасного кластерного середовища	8
РОЗДІЛ 2. ПРАКТИЧНА РЕАЛІЗАЦІЯ ВИСОКО ДОСТУПНОЇ СИСТЕМИ НА БАЗІ КОНТЕЙНЕРНОЇ ОРКЕСТРАЦІЇ.....	9
2.1 Підготовка серверної інфраструктури та мережових налаштувань	9
2.2 Організація базової конфігурації системного середовища	9
2.3 Встановлення та налаштування платформ для керування контейнерами	11
2.4 Ініціалізація кластерного середовища	12
2.5 Побудова контейнеризованого веб-застосунку	15
2.6 Інтеграція застосунку в кластерну інфраструктуру	16
2.7 Забезпечення масштабованості та відмовостійкості сервісу	18
2.8 Засоби моніторингу та діагностики працездатності	20
2.9 Інструменти для автоматизації та підтримки кластерної роботи	22
ВИСНОВОК	23
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	24

ВСТУП

Сучасні інформаційні системи потребують не лише високої продуктивності, але й безперебійної роботи, особливо в умовах постійного зростання навантаження та вимог до доступності. У зв'язку з цим зростає інтерес до технологій, які дозволяють створювати масштабовані, гнучкі та надійні програмні рішення. Однією з таких технологій є контейнеризація, що у поєднанні з оркестрацією контейнерів забезпечує ефективне управління ресурсами та високий рівень доступності застосунків.

У даній курсовій роботі розглянуто підходи до побудови високо доступних систем з використанням Kubernetes - популярної системи оркестрації контейнерів з відкритим кодом. Kubernetes забезпечує автоматичне розгортання, масштабування та управління контейнеризованими застосунками, що робить його ідеальним інструментом для створення надійних інфраструктур.

Мета даної роботи — продемонструвати практичну реалізацію високо доступної інформаційної системи за допомогою Kubernetes, зосереджуючи увагу на основних компонентах, налаштуванні кластеру, розгортанні сервісу та забезпеченні його стійкості до збоїв. У практичній частині було розгорнуто кластер з одним головним (master) вузлом і двома робочими (node) вузлами, на якому було реалізовано масштабований веб-застосунок на Flask.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ПОБУДОВИ ВИСОКО ДОСТУПНИХ СИСТЕМ З ВИКОРИСТАННЯМ КОНТЕЙНЕРИЗАЦІЇ ТА ОРКЕСТРАЦІЇ

1.1 Сутність високої доступності та масштабованості

Висока доступність (High Availability, HA) - це властивість системи залишатися працездатною навіть у разі відмови окремих її компонентів. Основною метою високої доступності є мінімізація простоїв та забезпечення безперервного надання послуг кінцевим користувачам. Для досягнення цього зазвичай використовують резервування ресурсів, автоматичне виявлення збоїв і їх усунення, а також балансування навантаження.

Масштабованість (Scalability) - це здатність системи адаптуватися до змін обсягу навантаження шляхом збільшення або зменшення обчислювальних ресурсів. Масштабування може бути горизонтальним (додавання нових вузлів або екземплярів сервісу) або вертикальним (збільшення ресурсів одного вузла). Висока масштабованість є критично важливою для систем, які працюють з великими обсягами даних або обслуговують велику кількість користувачів.

Разом, ці характеристики формують фундамент для створення надійних та ефективних інформаційних систем, здатних працювати у динамічному середовищі з мінімальними ризиками простоїв.

1.2 Основи контейнеризації та її переваги

Контейнеризація - це технологія, яка дозволяє упаковувати програмне забезпечення разом з усіма його залежностями в ізольоване середовище, яке називається контейнером. На відміну від традиційної віртуалізації, контейнери використовують ядро операційної системи хост-машини, що забезпечує меншу накладну витрату на ресурси та швидший запуск.

Основними принципами контейнеризації є:

- ізоляція процесів та ресурсів;
- повторюваність середовища;
- портативність між різними середовищами (від розробки до продакшну).

Серед ключових переваг контейнеризації:

- Швидке розгортання та оновлення застосунків;
- Легка масштабованість;
- Ефективне використання ресурсів;
- Покращена безпека через ізоляцію;
- Зручність CI/CD-процесів (безперервної інтеграції та доставки).

Контейнеризація стала основою для сучасних підходів до розробки та експлуатації програмного забезпечення, зокрема в мікросервісній архітектурі.

1.3 Архітектура та функціональні можливості систем оркестрації

Оркестрація контейнерів - це процес автоматичного управління розгортанням, масштабуванням та моніторингом контейнеризованих застосунків. В умовах великих інфраструктур без належної оркестрації стає складно ефективно керувати сотнями або тисячами контейнерів.

Kubernetes (або K8s) - це система оркестрації контейнерів з відкритим кодом, розроблена компанією Google. Вона дозволяє автоматизувати розгортання, управління та масштабування контейнеризованих застосунків у розподіленому середовищі. Kubernetes надає потужні механізми для автоматичного балансування навантаження, відновлення після збоїв, оновлень без простоїв та багато інших функцій, що полегшують управління контейнерами на великій кількості хостів.

Основні компоненти Kubernetes включають:

- **Master:** управляє кластером, приймає рішення про розподіл навантаження та управління контейнерами.
- **Node:** робочі вузли, на яких безпосередньо запускаються контейнери.
- **Pod:** найменша одиниця, що містить один або кілька контейнерів, які мають спільне середовище виконання.

Kubernetes дозволяє створювати високо доступні та масштабовані системи, що є основою для сучасних хмарних інфраструктур.

1.4 Структура та взаємодія компонентів сучасного кластерного середовища

Kubernetes-кластер складається з кількох основних компонентів, які працюють разом для забезпечення ефективного управління контейнерами. Кластер зазвичай включає два типи компонентів: контрольну площину (Control Plane) та робочі вузли (Nodes).

1. **Контрольна площина (Control Plane)** - відповідальна за управління кластером та прийняття рішень про розподіл ресурсів і керування контейнерами. Основні компоненти контрольної площини:

- **API сервер (kube-apiserver):** основний інтерфейс для взаємодії з кластером, обробляє запити користувачів та клієнтів.
- **Менеджер контролерів (kube-controller-manager):** відповідає за управління різними контролерами, які забезпечують стабільність стану кластеру, наприклад, контролер реплік або контролер відновлення.
- **Менеджер планувальника (kube-scheduler):** розподіляє поди між доступними вузлами в кластері, визначаючи, на якому вузлі буде запущено новий контейнер.
- **etcd:** розподілена база даних, яка зберігає всю конфігурацію та стан кластеру. Вона є основним джерелом правди для Kubernetes.

2. **Робочі вузли (Nodes)** - це фізичні або віртуальні машини, на яких фактично запускаються контейнери. Кожен робочий вузол містить кілька компонентів:

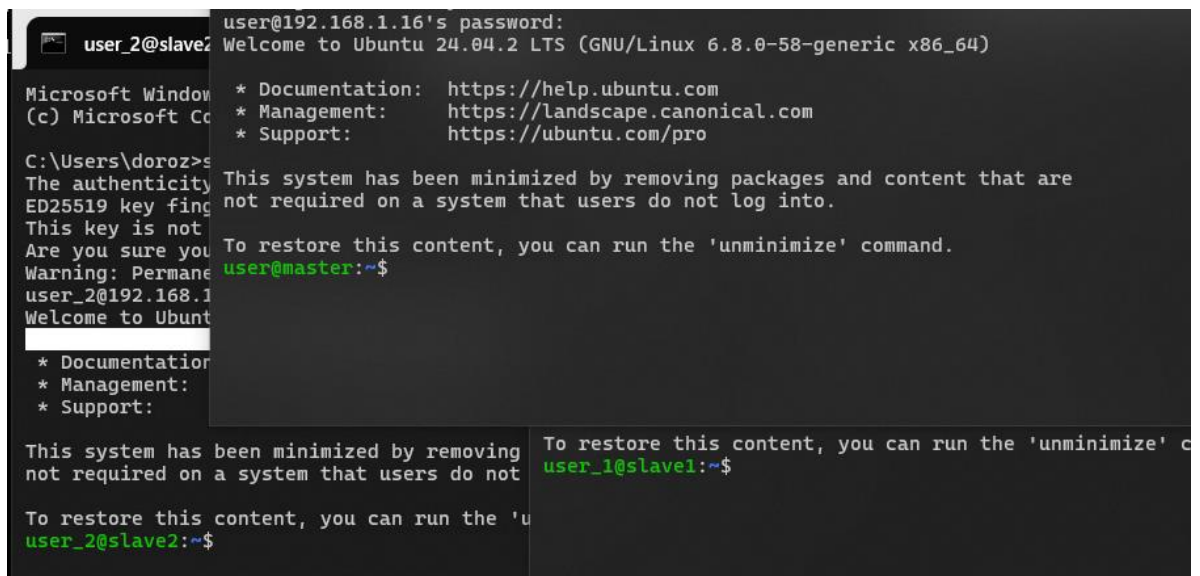
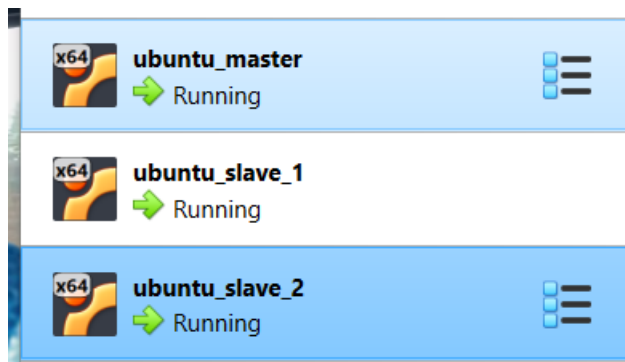
- **kubelet:** агент, що працює на кожному вузлі і забезпечує запуск контейнерів, згідно з наказами від API сервера.
- **kube-proxy:** відповідає за мережеве маршрутизування та балансування навантаження між контейнерами в межах вузла.
- **Docker/containerd:** платформи контейнеризації, які забезпечують запуск контейнерів на вузлі.

Ці компоненти працюють разом для забезпечення надійності, масштабованості та високої доступності Kubernetes-кластера.

РОЗДІЛ 2. ПРАКТИЧНА РЕАЛІЗАЦІЯ ВИСОКО ДОСТУПНОЇ СИСТЕМИ НА БАЗІ КОНТЕЙНЕРНОЇ ОРКЕСТРАЦІЇ

2.1 Підготовка серверної інфраструктури та мережевих налаштувань

Для початку я завантажив і інсталиював програму Virtual Box, а також завантажив ios образ “Ubuntu Server 24.04.2 LTS”. Далі я був розгорнув і налаштував три віртуальні машини під назвами: “slave_1”, “slave_2” і “master”. Кожна машина мала виділено 2Гб оперативної пам’яті, 2 ядра CPU, 25Gb дискового простору і включений Bridged Adapter для комунікації один з одним.



2.2 Організація базової конфігурації системного середовища

Перед початком роботи, я розширив дисковий простір на max 25Gb для кожного з хостів:

```
sudo vgs
sudo lvextend -l +100%FREE /dev/ubuntu-vg/ubuntu-lv
sudo resize2fs /dev/ubuntu-vg/ubuntu-lv
```

Далі я налаштував статичну ір адресу для кожного з хостів:


```
#master: 192.168.32.110/24, slave_1: 192.168.32.111/24, slave_2: 192.168.32.112/24
sudo nano /etc/netplan/01-netcfg.yaml
sudo chmod 600 /etc/netplan/01-netcfg.yaml
network:
  version: 2
  ethernets:
    enp0s3:
      dhcp4: no
      addresses:
        - 192.168.32.108/24
      routes:
        - to: default
          via: 192.168.32.1
      nameservers:
        addresses: [8.8.8.8, 8.8.4.4]

sudo netplan apply
```

Додав відповідність між IP-адресами та іменами хостів на кожному з хостів:

```
sudo nano /etc/hosts
192.168.32.110 master
192.168.32.111 slave1
192.168.32.112 slave2
```

Вимкнув swap на кожному із хостів, щоб розгорнута система працювала коректно:

```
sudo swapoff -a
sudo sed -i '/ swap / s/^(\.*\)$/#\1/g' /etc/fstab
```

Увімкнув модулі на кожному із хостів, щоб розгорнута система працювала коректно:

```
sudo nano /etc/modules-load.d/modules.conf
br_netfilter
overlay
```

Надав доступ певним портам на кожному із хостів, щоб система могла коректно комунікувати одна з одною:

```
sudo ufw enable
## master
## kubernetes
sudo ufw allow ssh
sudo ufw allow in 6443/tcp
sudo ufw allow in 443/tcp
sudo ufw allow in 2379:2380/tcp
sudo ufw allow in 10250/tcp
sudo ufw allow in 10259/tcp
sudo ufw allow in 10257/tcp

## flannel network
sudo ufw allow in 8285/udp
sudo ufw allow in 8472/udp

## slaves
## kubernetes
sudo ufw allow ssh
sudo ufw allow in 6443/tcp
sudo ufw allow in 443/tcp
sudo ufw allow in 10250/tcp
sudo ufw allow in 10256/tcp
sudo ufw allow in 30000:32767/tcp

## flannel network
sudo ufw allow in 8285/udp
sudo ufw allow in 8472/udp
```

2.3 Встановлення та налаштування платформ для керування контейнерами

Встановив Docker, containerd, kubelet, kubeadm і kubectl на кожному із хостів за допомогою документації:

```
sudo apt install docker-ce
sudo apt install kubelet kubeadm kubectl
sudo apt install containerd
```

```
user@master:~$ docker -v
Docker version 28.1.1, build 4eba377
```

```

user@master:~$ kubectl version
Client Version: v1.33.0
Kustomize Version: v5.6.0
The connection to the server localhost:8080 was refused - did you specify the right host or port?
user@master:~$ kubeadm version
kubeadm version: &version.Info{Major:"1", Minor:"33", EmulationMajor:"", EmulationMinor:"", MinCompatibilityMajor:"", MinCompatibilityMinor:"", GitVersion:"v1.33.0", GitCommit:"60a317eadfcb839692a68eab88b2096f4d708f4f", GitTreeState:"clean", BuildDate:"2025-04-23T13:05:48Z", GoVersion:"go1.24.2", Compiler:"gc", Platform:"linux/amd64"}
user@master:~$ kubelet --version
Kubernetes v1.33.0
user@master:~$

```

2.4 Ініціалізація кластерного середовища

Перед ініціалізацією системи, потрібно провести базове налаштування системи. Спочатку я згенерував новий конфіг файл для containerd, а потім змінив версію плагіна і увімкнув SystemdCgroup на кожному із хостів:

```

sudo mkdir /etc/containerd/
sudo sh -c "containerd config default > /etc/containerd/config.toml"
sudo nano /etc/containerd/config.toml
[plugins."io.containerd.grpc.v1.cri"] #додаву в /etc/containerd/config.toml
    sandbox_image = "registry.k8s.io/pause:3.10"

[plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc] #додаву в
/etc/containerd/config.toml

...

[plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc.options]
    SystemdCgroup = true

```

Налаштував параметри ядра для роботи Kubernetes-кластера на кожному із хостів:

```

cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
net.ipv4.ip_forward = 1
net.ipv4.conf.all.forwarding = 1
EOF
sudo sysctl --system

```

Завантажив та встановив CNI-плагіни для роботи мережі в Kubernetes на кожному із хостів:

```

sudo mkdir /opt/cni/bin

```

```
curl -L -o cni-plugins.tgz
https://github.com/containernetworking/plugins/releases/download/v1.7.1/cni-plugins-
linux-amd64-v1.7.1.tgz
sudo tar -C /opt/cni/bin -xzf cni-plugins.tgz
```

Налаштував утиліту crictl для взаємодії з containerd на кожному із хостів:

```
cat <<EOF | sudo tee /etc/crictl.yaml
runtime-endpoint: unix:///var/run/containerd/containerd.sock
image-endpoint: unix:///var/run/containerd/containerd.sock
timeout: 10
debug: false
EOF
```

Зафіксував версії основних компонентів Kubernetes і перезапустив необхідні сервіси на кожному із хостів:

```
sudo apt-mark hold kubelet kubeadm kubectl
sudo systemctl daemon-reload

sudo systemctl enable docker
sudo systemctl restart docker

sudo systemctl enable kubelet
sudo systemctl restart kubelet

sudo systemctl enable containerd
sudo systemctl restart containerd
```

Ініціалізував Kubernetes-кластер та встановив мережу flannel для коректної роботи pods на хості **master**:

```
sudo kubeadm config images pull

sudo kubeadm init --apiserver-advertise-address=192.168.32.110 --apiserver-cert-extra-
sans=192.168.32.110 --node-name master --pod-network-cidr=10.244.0.0/16

mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

```
kubectl apply -f https://github.com/flannel-io/flannel/releases/latest/download/kube-flannel.yml
```

```
sudo kubeadm token create --print-join-command
```

Потім підключив кожен вузол до розгорнутої системи, а саме: “slave1” і “slave2”:

```
sudo kubeadm join 192.168.32.110:6443 --token byu3jz.67ezr1kpvibyngi2 --discovery-token-ca-cert-hash sha256:3dbe8575a246d5f15e09dadd0d91ed197be1f56cb46c16be1ffb666d4d9c383b
```

```
[master] ~
[kubeconfig] Writing "kubelet.conf" kubeconfig file
[kubeconfig] Writing "controller-manager.conf" kubeconfig file
[kubeconfig] Writing "scheduler.conf" kubeconfig file
[etcd] Creating static Pod manifest for local etcd in "/etc/kubernetes/manifests"
[control-plane] Using manifest folder "/etc/kubernetes/manifests"
[control-plane] Creating static Pod manifest for "kube-apiserver"
[control-plane] Creating static Pod manifest for "kube-controller-manager"
[control-plane] Creating static Pod manifest for "kube-scheduler"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Starting the kubelet
[wait-control-plane] Waiting for the kubelet to boot up the control plane as static Pods from directory "/etc/kubernetes/manifests"
[kubelet-check] Waiting for a healthy kubelet at https://127.0.0.1:10248/healthz. This can take up to 4m0s
[kubelet-check] The kubelet is healthy after 1.002100727s
[control-plane-check] Waiting for healthy control plane components. This can take up to 4m0s
[control-plane-check] Checking kube-apiserver at https://192.168.32.110:6443/liveness
[control-plane-check] Checking kube-controller-manager at https://127.0.0.1:10257/healthz
[control-plane-check] Checking kube-scheduler at https://127.0.0.1:10259/liveness
[control-plane-check] kube-controller-manager is healthy after 2.768886963s
[control-plane-check] kube-scheduler is healthy after 4.368224818s
[control-plane-check] kube-apiserver is healthy after 6.003847745s
[upload-config] Storing the configuration used in ConfigMap "kubeadm-config" in the "kube-system" Namespace
[kubelet] Creating a ConfigMap "kubelet-config" in namespace kube-system with the configuration for the kubelets in the cluster
[upload-certs] Skipping phase. Please see --upload-certs
[mark-control-plane] Marking the node master as control-plane by adding the labels: [node-role.kubernetes.io/control-plane node.kubernetes.io/exclude-from-external-load-balancers]
[mark-control-plane] Marking the node master as control-plane by adding the taints: [node-role.kubernetes.io/control-plane:NoSchedule]
[bootstrap-token] Using token: l2x6mw.14eumm0ld49b3r1q
[bootstrap-token] Configuring bootstrap tokens, cluster-info ConfigMap, RBAC Roles
[bootstrap-token] Configured RBAC rules to allow Node Bootstrap tokens to get nodes
[bootstrap-token] Configured RBAC rules to allow Node Bootstrap tokens to post CSRs in order for nodes to get long term certificate credentials
[bootstrap-token] Configured RBAC rules to allow the csrapprover controller automatically approve CSRs from a Node Bootstrap Token
[bootstrap-token] Configured RBAC rules to allow certificate rotation for all node client certificates in the cluster
[bootstrap-token] Creating the "cluster-info" ConfigMap in the "kube-public" namespace
[kubelet-finalize] Updating "/etc/kubernetes/kubelet.conf" to point to a rotatable kubelet client certificate and key
[addons] Applied essential addon: CoreDNS
[addons] Applied essential addon: kube-proxy

Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config

Alternatively, if you are the root user, you can run:

export KUBECONFIG=/etc/kubernetes/admin.conf

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
https://kubernetes.io/docs/concepts/cluster-administration/addons/

Then you can join any number of worker nodes by running the following on each as root:

kubeadm join 192.168.32.110:6443 --token l2x6mw.14eumm0ld49b3r1q \
--discovery-token-ca-cert-hash sha256:80fad4a80bceef93e8391d2b7010a61b78cfaba343904c3b54a843f41e0db3c
```

```
user@master ~$ kubectl get pods -A -o wide
NAMESPACE NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES
default multitool 1/1 Running 0 2m30s 10.244.1.7 slave2 <none> <none>
kube-flannel kube-flannel-ds-c7qsh 1/1 Running 0 15m 192.168.32.112 slave2 <none> <none>
kube-flannel kube-flannel-ds-df9ww 1/1 Running 0 15m 192.168.32.110 master <none> <none>
kube-flannel kube-flannel-ds-lfgzd 1/1 Running 0 15m 192.168.32.111 slave1 <none> <none>
kube-system coredns-674b8bbfcf-ktggq 1/1 Running 0 4m52s 10.244.1.6 slave2 <none> <none>
kube-system coredns-674b8bbfcf-r8lwr 1/1 Running 0 4m52s 10.244.2.4 slave1 <none> <none>
kube-system etcd-master 1/1 Running 20 15m 192.168.32.110 master <none> <none>
kube-system kube-apiserver-master 1/1 Running 20 15m 192.168.32.110 master <none> <none>
kube-system kube-controller-manager-master 1/1 Running 0 15m 192.168.32.110 master <none> <none>
kube-system kube-proxy-7hggg 1/1 Running 0 15m 192.168.32.110 master <none> <none>
kube-system kube-proxy-q2qst 1/1 Running 0 15m 192.168.32.111 slave1 <none> <none>
kube-system kube-proxy-wfzwx 1/1 Running 0 15m 192.168.32.112 slave2 <none> <none>
kube-system kube-scheduler-master 1/1 Running 179 15m 192.168.32.110 master <none> <none>

user@master ~$ kubectl get services --all-namespaces
NAMESPACE NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
default kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 23m
kube-system kube-dns ClusterIP 10.96.0.10 <none> 53/UDP,53/TCP,9153/TCP 23m

user@master ~$ kubectl get componentstatuses
Warning: v1 ComponentStatus is deprecated in v1.19+
NAME STATUS MESSAGE ERROR
scheduler Healthy ok
controller-manager Healthy ok
etcd-0 Healthy ok

user@master ~$ kubectl get nodes -o wide
NAME STATUS ROLES AGE VERSION INTERNAL-IP EXTERNAL-IP OS-IMAGE KERNEL-VERSION CONTAINER-RUNTIME
master Ready control-plane 23m v1.33.0 192.168.32.110 <none> Ubuntu 24.04.2 LTS 6.8.0-59-generic containerd://1.7.27
slave1 Ready <none> 20m v1.33.0 192.168.32.111 <none> Ubuntu 24.04.2 LTS 6.8.0-59-generic containerd://1.7.27
slave2 Ready <none> 20m v1.33.0 192.168.32.112 <none> Ubuntu 24.04.2 LTS 6.8.0-59-generic containerd://1.7.27

user@master ~$ kubectl get svc kubernetes
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 23m

user@master ~$
```

2.5 Побудова контейнеризованого веб-застосунку

Підготував структуру проєкту для Flask-додатку на хості master:

```
mkdir my_app
cd my_app/
touch app.py
touch requirements.txt
touch Dockerfile
```

Створив простий Flask-додаток та Dockerfile для контейнеризації на хості master:

```
nano app.py
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello():
    return "Hello from container Kubernetes!"

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)

nano requirements.txt
flask

nano Dockerfile
FROM python:3.9-slim

WORKDIR /app

COPY requirements.txt .
RUN pip install -r requirements.txt

COPY app.py .

CMD ["python", "app.py"]
```


Створив Docker-образ Flask-додатку та завантажив його на Docker Hub:



```
docker build -t my-flask-app:latest .
docker images

docker login
sudo docker tag my-flask-app:latest volodymyrdorozhovets/coursework:latest
sudo docker push volodymyrdorozhovets/coursework:latest
```

volodymyrdorozhovets/coursework

Last pushed 2 minutes ago • Repository size: 49 MB

coursework 

Add a category  

General

Tags


Image Management **BETA**

Collaborators



Webhooks

Settings

Tags

 DOCKER SCOUT INACTIVE
[Activate](#)

This repository contains 0 tag(s).

Tag	OS	Type	Pulled	Pushed
 latest		Image	less than 1 day	2 minutes

[See all](#)

2.6 Інтеграція застосунку в кластерну інфраструктуру

Створив файли конфігурації для Kubernetes на хості master:

```
mkdir k8s
cd k8s/
touch deployment.yaml
touch service.yaml
```

Створив файли конфігурації для Kubernetes на хості master:

```
nano deployment.yaml
apiVersion: apps/v1
```

```
kind: Deployment
metadata:
  name: flask-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: flask-app
  template:
    metadata:
      labels:
        app: flask-app
    spec:
      containers:
        - name: flask-container
          image: volodymyrdorozhovets/coursework:latest
          ports:
            - containerPort: 5000
```

```
nano service.yaml
apiVersion: v1
kind: Service
metadata:
  name: flask-service
spec:
  selector:
    app: flask-app
  type: NodePort
  ports:
    - protocol: TCP
      port: 80
      targetPort: 5000
      nodePort: 30007
```

Зробив деплой на хості master:

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



```

user@master ~$ kubectl get pods -A -o wide

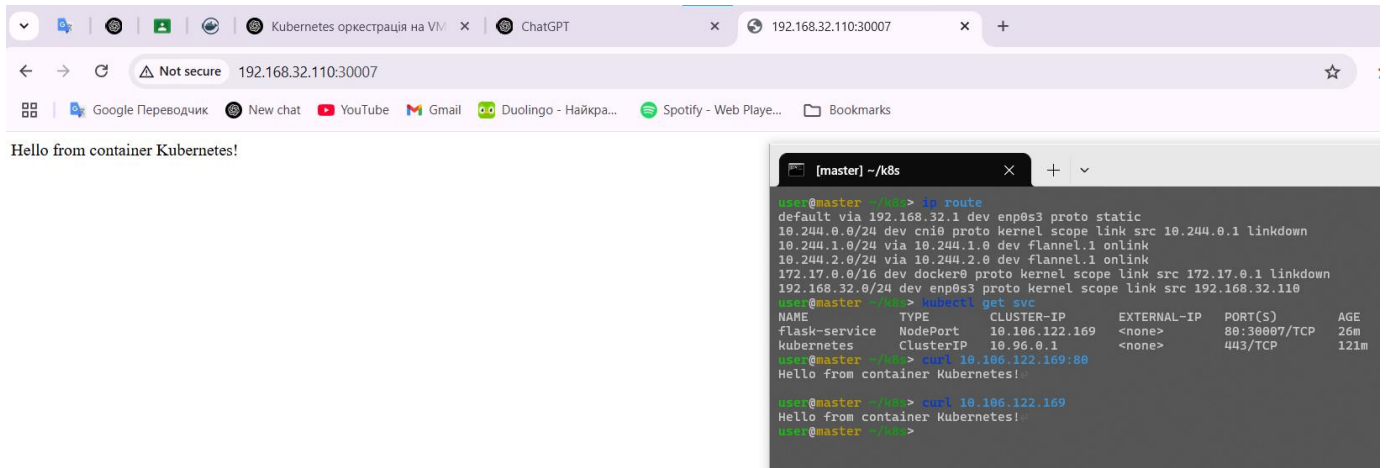
```

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES
default	flask-app-77bc849f95-g5mnq	1/1	Running	0	6m7s	10.244.2.9	slave1	<none>	<none>
default	flask-app-77bc849f95-ppsxh	1/1	Running	0	6m7s	10.244.1.11	slave2	<none>	<none>
default	flask-app-77bc849f95-xqh4c	1/1	Running	0	6m7s	10.244.2.8	slave1	<none>	<none>
kube-flannel	kube-flannel-ds-8z4rh	1/1	Running	0	47m	192.168.32.111	slave1	<none>	<none>
kube-flannel	kube-flannel-ds-bq5qw	1/1	Running	0	47m	192.168.32.112	slave2	<none>	<none>
kube-flannel	kube-flannel-ds-c5bpm	1/1	Running	0	47m	192.168.32.110	master	<none>	<none>
kube-system	coredns-674b8bbfcf-92clr	1/1	Running	0	48m	10.244.1.10	slave2	<none>	<none>
kube-system	coredns-674b8bbfcf-xg2r6	1/1	Running	0	48m	10.244.2.7	slave1	<none>	<none>
kube-system	etcd-master	1/1	Running	22 (50m ago)	48m	192.168.32.110	master	<none>	<none>
kube-system	kube-apiserver-master	1/1	Running	22 (50m ago)	48m	192.168.32.110	master	<none>	<none>
kube-system	kube-controller-manager-master	1/1	Running	2 (50m ago)	48m	192.168.32.110	master	<none>	<none>
kube-system	kube-proxy-jmvsd	1/1	Running	0	48m	192.168.32.111	slave1	<none>	<none>
kube-system	kube-proxy-s5nlt	1/1	Running	0	48m	192.168.32.110	master	<none>	<none>
kube-system	kube-proxy-wxj9w	1/1	Running	0	48m	192.168.32.112	slave2	<none>	<none>
kube-system	kube-scheduler-master	1/1	Running	181 (50m ago)	48m	192.168.32.110	master	<none>	<none>

```

user@master ~$ kubectl get pods -A -o wide

```



2.7 Забезпечення масштабованості та відмовостійкості сервісу

Налаштував автоматичне масштабування в Kubernetes. Спочатку я встановив metrics-server:

```

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

```

Налаштував metrics-server згідного вимог:

```

kubectl -n kube-system edit deployment metrics-server
- --kubelet-insecure-tls

```

Потім змінив код мого основного файлу deployment.yaml для коректного збору даних metrics-server:

```

nano deployment.yaml
resources:
  requests:
    cpu: "100m"
    memory: "128Mi"
  limits:
    cpu: "500m"
    memory: "256Mi"

```

Застосував зміни у файлі і налаштував autoscale. Тепер якщо навантаження CPU pods flask-app збільшиться до 50%, то система почне збільшувати репліки:

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

```
kubectl autoscale deployment flask-app --cpu-percent=50 --min=3 --max=10
```

```
kubectl get hpa
```

```
user@master ~/k8s> kubectl get pods -A -o wide
```

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES
default	flask-app-77bc849f95-g5mnq	1/1	Running	0	60m	10.244.2.9	slave1	<none>	<none>
default	flask-app-77bc849f95-ppsxb	1/1	Running	0	60m	10.244.1.11	slave2	<none>	<none>
default	flask-app-77bc849f95-xqh4c	1/1	Running	0	60m	10.244.2.8	slave1	<none>	<none>
default	multitool	1/1	Running	0	49m	10.244.1.12	slave2	<none>	<none>
kube-flannel	kube-flannel-ds-8z4rh	1/1	Running	0	102m	192.168.32.111	slave1	<none>	<none>
kube-flannel	kube-flannel-ds-bq5qw	1/1	Running	0	102m	192.168.32.112	slave2	<none>	<none>
kube-flannel	kube-flannel-ds-c5bpm	1/1	Running	0	102m	192.168.32.110	master	<none>	<none>
kube-system	coredns-674b8bbfcf-92clr	1/1	Running	0	102m	10.244.1.10	slave2	<none>	<none>
kube-system	coredns-674b8bbfcf-xg2r6	1/1	Running	0	102m	10.244.2.7	slave1	<none>	<none>
kube-system	etcd-master	1/1	Running	22 (105m ago)	102m	192.168.32.110	master	<none>	<none>
kube-system	kube-apiserver-master	1/1	Running	22 (105m ago)	102m	192.168.32.110	master	<none>	<none>
kube-system	kube-controller-manager-master	1/1	Running	2 (105m ago)	102m	192.168.32.110	master	<none>	<none>
kube-system	kube-proxy-jmvsd	1/1	Running	0	102m	192.168.32.111	slave1	<none>	<none>
kube-system	kube-proxy-s5nlt	1/1	Running	0	102m	192.168.32.110	master	<none>	<none>
kube-system	kube-proxy-wxj9w	1/1	Running	0	102m	192.168.32.112	slave2	<none>	<none>
kube-system	kube-scheduler-master	1/1	Running	181 (105m ago)	102m	192.168.32.110	master	<none>	<none>
kube-system	metrics-server-8467fcc7b7-zkj4p	1/1	Running	0	75s	10.244.2.11	slave1	<none>	<none>

```
[master] ~/k8s
```

```
user@master ~/k8s> kubectl top pods
```

NAME	CPU(cores)	MEMORY(bytes)
flask-app-58f7464b5f-7mp6f	1m	19Mi
flask-app-58f7464b5f-9ctjt	1m	19Mi
flask-app-58f7464b5f-r7v2b	1m	19Mi

```
user@master ~/k8s> kubectl top nodes
```

NAME	CPU(cores)	CPU(%)	MEMORY(bytes)	MEMORY(%)
master	119m	5%	1351Mi	72%
slave1	30m	1%	695Mi	37%
slave2	43m	2%	724Mi	38%

```
user@master ~/k8s> kubectl get hpa
```

```
user@master ~/k8s> kubectl autoscale deployment flask-app --cpu-percent=50 --min=3 --max=10
```

```
user@master ~/k8s> kubectl get hpa
```

```
Error from server (AlreadyExists): horizontalpodautoscalers.autoscaling "flask-app" already exists
```

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
flask-app	Deployment/flask-app	cpu: 1%/50%	3	10	3	22h

```
user@master ~/k8s>
```

Створив та застосував PodDisruptionBudget для забезпечення високої доступності. Тепер якщо буде відбуватися оновлення додатку, то 1 pod flask-app повинен працювати:

```
touch pdb.yaml
```

```
nano pdb.yaml
```

```
apiVersion: policy/v1
```

```
kind: PodDisruptionBudget
```

```
metadata:
```

```
  name: flask-app-pdb
```

```
spec:
```

```
  minAvailable: 1
```

```
  selector:
```

```
    matchLabels:
```

```
      app: flask-app
```

```
kubectl apply -f pdb.yaml
kubectl get pdb
```

```
[master] ~/k8s
```

```
user@master ~/k8s> kubectl apply -f pdb.yaml
poddisruptionbudget.policy/flask-app-pdb created
user@master ~/k8s> kubectl get pdb
```

NAME	MIN AVAILABLE	MAX UNAVAILABLE	ALLOWED DISRUPTIONS	AGE
flask-app-pdb	1	N/A	2	3s

```
user@master ~/k8s> |
```

2.8 Засоби моніторингу та діагностики працездатності

В якості тестування системи, я спочатку вирішив перевірити чи система відновлюється після видалення всіх pods:

```

user@master ~$ kubectl delete pod --all -A
pod "flask-app-65cf6c5db9-8d2tn" deleted
pod "flask-app-65cf6c5db9-jldzz" deleted
pod "flask-app-65cf6c5db9-psswq" deleted
pod "kube-flannel-ds-9s6bl" deleted
pod "kube-flannel-ds-d8pzq" deleted
pod "kube-flannel-ds-qdj4v" deleted
pod "coredns-674b8bbfcf-9m2qf" deleted
pod "coredns-674b8bbfcf-gtpd9" deleted
pod "etcd-master" deleted
pod "kube-apiserver-master" deleted
pod "kube-controller-manager-master" deleted
pod "kube-proxy-crbb1" deleted
pod "kube-proxy-frscp" deleted
pod "kube-proxy-x9hs4" deleted
pod "kube-scheduler-master" deleted
pod "metrics-server-8467fcc7b7-2rwv4" deleted
user@master ~$ kubectl get pods -A -o wide

```

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES
default	flask-app-65cf6c5db9-blxkk	1/1	Running	0	45s	10.244.1.39	slave2	<none>	<none>
default	flask-app-65cf6c5db9-kcjm	1/1	Running	0	45s	10.244.2.29	slave1	<none>	<none>
default	flask-app-65cf6c5db9-pq4fh	1/1	Running	0	45s	10.244.1.37	slave2	<none>	<none>
kube-flannel	kube-flannel-ds-4mpvw	1/1	Running	0	43s	192.168.32.111	slave1	<none>	<none>
kube-flannel	kube-flannel-ds-jngw4	1/1	Running	0	43s	192.168.32.112	slave2	<none>	<none>
kube-flannel	kube-flannel-ds-nmqnk	1/1	Running	0	43s	192.168.32.110	master	<none>	<none>
kube-system	coredns-674b8bbfcf-694hb	1/1	Running	0	45s	10.244.2.30	slave1	<none>	<none>
kube-system	coredns-674b8bbfcf-kvjqq	1/1	Running	0	45s	10.244.1.38	slave2	<none>	<none>
kube-system	etcd-master	1/1	Running	23 (3h3m ago)	45s	192.168.32.110	master	<none>	<none>
kube-system	kube-apiserver-master	1/1	Running	23 (3h3m ago)	45s	192.168.32.110	master	<none>	<none>
kube-system	kube-controller-manager-master	1/1	Running	3 (3h3m ago)	45s	192.168.32.110	master	<none>	<none>
kube-system	kube-proxy-r5x2x	1/1	Running	0	44s	192.168.32.112	slave2	<none>	<none>
kube-system	kube-proxy-wf4z8	1/1	Running	0	44s	192.168.32.110	master	<none>	<none>
kube-system	kube-proxy-wrt64	1/1	Running	0	44s	192.168.32.111	slave1	<none>	<none>
kube-system	kube-scheduler-master	1/1	Running	182 (3h3m ago)	44s	192.168.32.110	master	<none>	<none>
kube-system	metrics-server-8467fcc7b7-jbv9z	1/1	Running	0	44s	10.244.1.40	slave2	<none>	<none>

```

user@master ~$ kubectl get pods

```

Другий етап тестування є перевірка PodDisruptionBudget:

```
[master] kubectl drain slave2 -> + v
```

```
user@master ~$ kubectl drain slave1 --ignore-daemonsets --delete-emptydir-data
node/slave1 cordoned
Warning: Ignoring DaemonSet-managed Pods: kube-flannel/kube-flannel-ds-qmvpw, kube-system/kube-
evicting pod kube-system/coredns-67d8bbbfcd-mndss
evicted pod default/flask-app-65cfc5db9-qgqfh
pod/coredns-67d8bbbfcd-q9ubh evicted
pod/flask-app-65cfc5db9-kcjvf evicted
node/slave1 drained
[master] ~
```

```
[master] ~$ kubectl drain slave2 --ignore-daemonsets --delete-emptydir-data
node/slave2 cordoned
Warning: Ignoring DaemonSet-managed Pods: kube-flannel/kube-flannel-ds-jngwd, kube-system/kube-
evicting pod kube-system/metrics-server-8467fccc7b-jbvzr
evicted pod default/flask-app-65cfc5db9-nmpdp
evicted pod default/flask-app-65cfc5db9-blvkx
evicted pod default/flask-app-65cfc5db9-pqgfh
evicted pod kube-system/coredns-67d8bbbfcd-kvjvq
error when evicting pods/*flask-app-65cfc5db9-pqgfh* -n "default": (will retry after 5s): Cannot
pod/metrics-server-8467fccc7b-jbvzr evicted
evicted pod default/flask-app-65cfc5db9-pqgfh
error when evicting pods/*flask-app-65cfc5db9-pqgfh* -n "default": (will retry after 5S): Cannot evict pod as it would violate the pod's disruption budget.
evicted pod default/kube-proxy-wt8ts
evicted pod default/flask-app-65cfc5db9-pqgfh
error when evicting pods/*flask-app-65cfc5db9-pqgfh* -n "default": (will retry after 5S): Cannot evict pod as it would violate the pod's disruption budget.
evicted pod default/flask-app-65cfc5db9-pqgfh
error when evicting pods/*flask-app-65cfc5db9-pqgfh* -n "default": (will retry after 5S): Cannot evict pod as it would violate the pod's disruption budget.
evicted pod default/flask-app-65cfc5db9-pqgfh
error when evicting pods/*flask-app-65cfc5db9-pqgfh* -n "default": (will retry after 5S): Cannot evict pod as it would violate the pod's disruption budget.
evicted pod default/flask-app-65cfc5db9-pqgfh
error when evicting pods/*flask-app-65cfc5db9-pqgfh* -n "default": (will retry after 5S): Cannot evict pod as it would violate the pod's disruption budget.
evicted pod default/flask-app-65cfc5db9-pqgfh
error when evicting pods/*flask-app-65cfc5db9-pqgfh* -n "default": (will retry after 5S): Cannot evict pod as it would violate the pod's disruption budget.
evicted pod default/flask-app-65cfc5db9-pqgfh
error when evicting pods/*flask-app-65cfc5db9-pqgfh* -n "default": (will retry after 5S): Cannot evict pod as it would violate the pod's disruption budget.
evicted pod default/flask-app-65cfc5db9-pqgfh
error when evicting pods/*flask-app-65cfc5db9-pqgfh* -n "default": (will retry after 5S): Cannot evict pod as it would violate the pod's disruption budget.
evicted pod default/flask-app-65cfc5db9-pqgfh
error when evicting pods/*flask-app-65cfc5db9-pqgfh* -n "default": (will retry after 5S): Cannot evict pod as it would violate the pod's disruption budget.
```

```
user@master ~$ kubectl get pods -A -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
default	0/1	Pending	0	38s	<none>	<none>
Flask-app-65cfc5db9-8stxc	1/1	Running	0	24m	19.240.1.37	slave2
Flask-app-65cfc5db9-pqgfh	0/1	Pending	0	38s	<none>	<none>
kube-flannel	1/1	Running	0	22m	192.168.32.111	slave1
kube-flannel-ds-jmgvw	1/1	Running	0	22m	192.168.32.112	slave2
kube-flannel-ds-meqtk	1/1	Running	0	22m	192.168.32.110	master
coredns-67d8bbbfcd-xndjs	1/1	Running	0	196s	19.240.0.11	master
kube-system	1/1	Running	0	38s	19.240.0.12	master
etcd-master	1/1	Running	23 (3h6m ago)	24s	192.168.32.110	master
kube-controller-manager-master	1/1	Running	3 (3h6m ago)	24s	192.168.32.110	master
kube-proxy-r5k2x	1/1	Running	0	23s	192.168.32.112	slave2
kube-proxy-wt8ts	1/1	Running	0	23s	192.168.32.110	master
kube-proxy-rt64	1/1	Running	0	23s	192.168.32.111	slave1
kube-scheduler-master	1/1	Running	182 (3h6m ago)	23s	192.168.32.110	master
metrics-server-8467fccc7b-6fpns	0/1	Pending	0	38s	<none>	<none>

```
user@master ~$
```


І третій етап тестування це є навантаження CPU. Я був зайшов на один з род flask-app, встановив програму stress і почав навантажувати 1 ядро CPU з двох

МОЖЛИВИХ:

```
[master] kubectl exec -it pod - × + v
root@flask-app-65cf6c5db9-pq4fh:/app# stress --cpu 1 --timeout 60
stress: info: [138] dispatching hogs: 1 cpu, 0 io, 0 vm, 0 hdd
stress: info: [138] successful run completed in 60s
root@flask-app-65cf6c5db9-pq4fh:/app#
```

```
user@master ~/k8s> kubectl top pods
NAME CPU(cores) MEMORY(bytes)
flask-app-65cf6c5db9-85tcx 1m 19Mi
flask-app-65cf6c5db9-8h9vx 1m 19Mi
flask-app-65cf6c5db9-bglhb 1m 19Mi
flask-app-65cf6c5db9-pq4fh 501m 41Mi
flask-app-65cf6c5db9-px8lr 1m 19Mi
flask-app-65cf6c5db9-t2k45 28m 19Mi
flask-app-65cf6c5db9-xqhpj 1m 19Mi
flask-app-65cf6c5db9-zw7x5 1m 19Mi
user@master ~/k8s> kubectl get hpa
NAME REFERENCE TARGETS MINPODS MAXPODS REPLICAS AGE
flask-app Deployment/flask-app cpu: 84%/50% 3 10 8 151m
user@master ~/k8s> kubectl get pods -A -o wide
NAMESPACE NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES
default flask-app-65cf6c5db9-4xdng 1/1 Running 0 16s 10.244.2.35 slave1 <none> <none>
default flask-app-65cf6c5db9-85tcx 1/1 Running 0 6m54s 10.244.1.43 slave2 <none> <none>
default flask-app-65cf6c5db9-8h9vx 1/1 Running 0 106s 10.244.2.32 slave1 <none> <none>
default flask-app-65cf6c5db9-bglhb 1/1 Running 0 31s 10.244.1.45 slave2 <none> <none>
default flask-app-65cf6c5db9-pq4fh 1/1 Running 0 10m 10.244.1.37 slave1 <none> <none>
default flask-app-65cf6c5db9-px8lr 1/1 Running 0 106s 10.244.2.31 slave1 <none> <none>
default flask-app-65cf6c5db9-t2k45 1/1 Running 0 31s 10.244.2.34 slave1 <none> <none>
default flask-app-65cf6c5db9-wcjpj 1/1 Running 0 16s 10.244.1.46 slave2 <none> <none>
default flask-app-65cf6c5db9-xqhpj 1/1 Running 0 106s 10.244.2.33 slave1 <none> <none>
default flask-app-65cf6c5db9-zw7x5 1/1 Running 0 6m54s 10.244.1.44 slave2 <none> <none>
kube-flannel kube-flannel-ds-dmpvw 1/1 Running 0 10m 192.168.32.111 slave1 <none> <none>
kube-flannel kube-flannel-ds-jngw4 1/1 Running 0 10m 192.168.32.112 slave2 <none> <none>
kube-flannel kube-flannel-ds-nmqnk 1/1 Running 0 10m 192.168.32.110 master <none> <none>
kube-system coredns-674b8bbfcf-ndnws 1/1 Running 0 8m2s 10.244.0.11 master <none> <none>
kube-system coredns-674b8bbfcf-x7h8j 1/1 Running 0 6m54s 10.244.0.12 master <none> <none>
kube-system etcd-master 1/1 Running 23 (3h13m ago) 10m 192.168.32.110 master <none> <none>
kube-system kube-apiserver-master 1/1 Running 23 (3h13m ago) 10m 192.168.32.110 master <none> <none>
kube-system kube-controller-manager-master 1/1 Running 3 (3h13m ago) 10m 192.168.32.110 master <none> <none>
kube-system kube-proxy-r5x2x 1/1 Running 0 10m 192.168.32.112 slave2 <none> <none>
kube-system kube-proxy-wf4z8 1/1 Running 0 10m 192.168.32.110 master <none> <none>
kube-system kube-proxy-wrt64 1/1 Running 0 10m 192.168.32.111 slave1 <none> <none>
kube-system kube-scheduler-master 1/1 Running 182 (3h13m ago) 10m 192.168.32.110 master <none> <none>
kube-system metrics-server-8467fcc7b7-6fpn5 1/1 Running 0 6m54s 10.244.1.42 slave2 <none> <none>
user@master ~/k8s> |
```

```
user@master ~/k8s> kubectl describe hpa flask-app
Name: flask-app
Namespace: default
Labels: <none>
Annotations: <none>
CreationTimestamp: Fri, 09 May 2025 16:20:30 +0000
Reference: Deployment/flask-app
Metrics:
  resource cpu on pods (as a percentage of request): 1% (1m) / 50%
Min replicas: 3
Max replicas: 10
Deployment pods: 3 current / 3 desired
Conditions:
  Type Status Reason Message
  ----
  AbleToScale True ReadyForNewScale recommended size matches current size
  ScalingActive True ValidMetricFound the HPA was able to successfully calculate a replica count from cpu resource utilization (percentage of request)
  ScalingLimited True TooFewReplicas the desired replica count is less than the minimum replica count
Events:
  Type Reason Age From Message
  ----
  Warning FailedGetResourceMetric 12m (x21 over 95m) horizontal-pod-autoscaler failed to get cpu utilization: unable to get metrics for resource cpu: unable to fetch metrics from resource metrics API
  Warning FailedComputeMetricsReplicas 12m (x21 over 95m) horizontal-pod-autoscaler invalid metrics (1 invalid out of 1), first error is: failed to get cpu resource metric value: failed to get cpu utilization: unable to get metrics for resource cpu: unable to fetch metrics from resource metrics API; the server is currently unable to handle the request (get pods.metrics.k8s.io)
  Normal SuccessfulRescale 8m25s horizontal-pod-autoscaler New size: 6; reason: cpu resource utilization (percentage of request) above target
  Normal SuccessfulRescale 7m10s horizontal-pod-autoscaler New size: 8; reason: cpu resource utilization (percentage of request) above target
  Normal SuccessfulRescale 6m55s horizontal-pod-autoscaler New size: 10; reason: cpu resource utilization (percentage of request) above target
  Normal SuccessfulRescale 6m55s horizontal-pod-autoscaler New size: 6; reason: All metrics below target
  Normal SuccessfulRescale 6m55s horizontal-pod-autoscaler New size: 3; reason: All metrics below target
user@master ~/k8s> kubectl get pods -A -o wide
NAMESPACE NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES
default flask-app-65cf6c5db9-pq4fh 1/1 Running 0 17m 10.244.1.37 slave2 <none> <none>
default flask-app-65cf6c5db9-px8lr 1/1 Running 0 8m26s 10.244.2.31 slave1 <none> <none>
default flask-app-65cf6c5db9-xqhpj 1/1 Running 0 8m26s 10.244.2.33 slave1 <none> <none>
kube-flannel kube-flannel-ds-dmpvw 1/1 Running 0 17m 192.168.32.111 slave1 <none> <none>
kube-flannel kube-flannel-ds-jngw4 1/1 Running 0 17m 192.168.32.112 slave2 <none> <none>
kube-flannel kube-flannel-ds-nmqnk 1/1 Running 0 17m 192.168.32.110 master <none> <none>
kube-system coredns-674b8bbfcf-ndnws 1/1 Running 0 14m 10.244.0.11 master <none> <none>
kube-system coredns-674b8bbfcf-x7h8j 1/1 Running 0 13m 10.244.0.12 master <none> <none>
kube-system etcd-master 1/1 Running 23 (3h19m ago) 17m 192.168.32.110 master <none> <none>
kube-system kube-apiserver-master 1/1 Running 23 (3h19m ago) 17m 192.168.32.110 master <none> <none>
kube-system kube-controller-manager-master 1/1 Running 3 (3h19m ago) 17m 192.168.32.110 master <none> <none>
kube-system kube-proxy-r5x2x 1/1 Running 0 17m 192.168.32.112 slave2 <none> <none>
kube-system kube-proxy-wf4z8 1/1 Running 0 17m 192.168.32.110 master <none> <none>
kube-system kube-proxy-wrt64 1/1 Running 0 17m 192.168.32.111 slave1 <none> <none>
kube-system kube-scheduler-master 1/1 Running 182 (3h19m ago) 17m 192.168.32.110 master <none> <none>
kube-system metrics-server-8467fcc7b7-6fpn5 1/1 Running 0 13m 10.244.1.42 slave2 <none> <none>
user@master ~/k8s> |
```

2.9 Інструменти для автоматизації та підтримки кластерної роботи

Серед інструментів які я використовував під час налаштування системи є перший з них, це reset всієї системи:

```
sudo kubeadm reset -f # Скидає налаштування Kubernetes

sudo rm -rf ~/.kube # Видаляє конфігурації kubectl
sudo rm -rf /var/lib/kubelet # Видаляє дані та конфігурації kubelet
sudo rm -rf /var/lib/containerd # Видаляє дані контейнерного runtime (containerd)
sudo rm -rf /var/lib/dockerhim # Видаляє залишки dockershim (взаємодія між Docker і Kubernetes)
sudo rm -rf /var/lib/etcd # Видаляє дані etcd (зберігання конфігурацій Kubernetes)
sudo rm -rf /var/run/kubernetes # Видаляє тимчасові файли Kubernetes
sudo rm -rf /var/lib/cni # Видаляє дані CNI (Container Network Interface)
sudo rm -rf /etc/kubernetes # Видаляє конфігураційні файли Kubernetes
sudo rm -rf /etc/containerd # Видаляє конфігурації containerd
sudo rm -rf /etc/cni # Видаляє конфігурації CNI
sudo rm -rf /run/flannel # Видаляє тимчасові файли Flannel (мережевий плагін)
sudo rm -rf /opt/cni/bin # Видаляє виконувані файли CNI плагінів

sudo ip link delete cni0 # Видаляє мережевий інтерфейс CNI

# Очищає iptables правила для NAT, raw, mangle і загальних
sudo iptables -t nat -F && sudo iptables -t nat -X
sudo iptables -t raw -F && sudo iptables -t raw -X
sudo iptables -t mangle -F && sudo iptables -t mangle -X
sudo iptables -F && sudo iptables -X

sudo ip route flush table main # Очищає маршрути в основній таблиці маршрутизації
```

А другий це є дебаг системи:

```
journalctl -u kubelet -f # Виводить журнали сервісу kubelet в реальному часі

kubectl get pods -A -o wide # Отримує список всіх подів у всіх просторах імен з додатковою інформацією
kubectl get services --all-namespaces # Отримує список всіх сервісів у всіх просторах імен
kubectl get componentstatuses # Перевіряє статус компонентів Kubernetes
kubectl get nodes -o wide # Отримує інформацію про вузли з додатковою інформацією
kubectl get events -A --sort-by=.metadata.creationTimestamp # Отримує всі події, відсортовані за часом створення
kubectl get events -A --field-selector type=Warning # Отримує тільки попередження з подій
kubectl logs -n kube-system ... # Виводить журнали з конкретного пода в просторі імен kube-system
kubectl describe pod calico-node-6jhnf -n calico-system # Детальний опис конкретного пода в просторі імен calico-system

kubectl top pods # Отримує інформацію про використання ресурсів подами
kubectl top nodes # Отримує інформацію про використання ресурсів вузлами
kubectl get hpa # Отримує інформацію про Horizontal Pod Autoscalers

sudo crictl ps # Переглядає запущені контейнери за допомогою CRI (Container Runtime Interface)
sudo systemctl --system # Перезавантажує всі налаштування systemctl (керування параметрами ядра)
lsmod | grep br_netfilter # Перевіряє, чи завантажений модуль br_netfilter для обробки мережевих фільтрів
```

ВИСНОВОК

У результаті виконання даної курсової роботи була здійснена практична реалізація високо доступної системи з використанням Kubernetes для оркестрації контейнеризованих застосунків. У процесі роботи було налаштовано кластер з одним майстер-вузлом та двома робочими вузлами, що дозволило продемонструвати основні можливості Kubernetes для забезпечення високої доступності та масштабованості систем.

Основні результати:

1. Контейнеризація та оркестрація забезпечили високу ефективність використання ресурсів, можливість масштабування та простоту управління застосунками на різних етапах їхнього життєвого циклу.
2. Налаштування Kubernetes-кластера дозволило досягти високої доступності системи через використання механізмів автоматичного відновлення після збоїв та балансування навантаження.
3. Реалізація Flask-додатку в контейнерах та його масштабування в Kubernetes продемонстрували, як ефективно можна управляти контейнерами у великому кластері без простоїв.
4. Огляд компонентів Kubernetes та їхнього взаємозв'язку дозволив зрозуміти основні принципи роботи оркестрації та їхню роль у забезпеченні безперервної роботи застосунків.

Загалом, використання Kubernetes для оркестрації контейнерів значно спрощує управління складними інформаційними системами, забезпечує їх високу доступність та дозволяє ефективно масштабувати ресурси відповідно до вимог навантаження. Ці підходи є важливими для створення надійних та гнучких систем, що здатні адаптуватися до змінних умов та вимог.

Детальна реалізація проєкту, з вихідним кодом та конфігураційними файлами, доступна у відкритому репозиторії за посиланням:

<https://github.com/VolodymyrDorozhovets/coursework>.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

- [1] <https://medium.com/@rphilogene/what-is-kubernetes-what-you-need-to-know-as-a-developer-674af25e3947>
- [2] <https://middleware.io/blog/containerization/>
- [3] <https://docs.docker.com/engine/install/ubuntu/>
- [4] <https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/>
- [5] <https://medium.com/@mrdevsecops/set-up-a-kubernetes-cluster-with-kubeadm-508db74028ce>
- [6] <https://monowar-mukul.medium.com/kubernetes-create-a-new-token-and-join-command-to-rejoin-add-worker-node-74bbe8774808>
- [7] <https://stackoverflow.com/questions/60420546/how-do-i-fix-a-dial-tcp-10-96-0-1443-i-o-timeout-error-for-operator-pod-instal>
- [8] <https://kubernetes.io/docs/reference/networking/ports-and-protocols/>
- [9] <https://medium.com/@sainathmitalakar/kubernetes-addons-and-metrics-server-53abc91d76a1>
- [10] <https://medium.com/@yakuphanbilgic3/exploring-horizontal-pod-autoscaling-in-kubernetes-5e84d1b25202>