Опис завдання:

Написати маніфести для створення деплойменту пода, в якому:

1. працює контейнер з busybox
2. кожні 5 секунд він одночасно пише на STDOUT і у файл на файловій системі hostname контейнера й дату
3. файл повинен лежати у Persistent Volume і бути доступним після перезапуску пода
4. ім’я файлу передається у под за допомогою ConfigMap
5. у пода повинно бути 3 репліки, під час експериментів треба проскейлити под у більшу й меншу сторону та переконатися, шо всі поди пишуть у потрібний файл
6. налаштувати HPA для деплоймента
7. додати у кластер prometheus, fluentd, loki, grafana
8. зібрати логи та метрики до grafana

План

1. Створення контейнера з busybox
    1.1. Створюємо простір імен bb-namespace
    1.2. Створюємо Persistent Volume (PV)
    1.3. Створюємо Persistent Volume Claim (PVC)
    1.4. Створюємо ConfigMap
    1.5. Створюємо Deployment Pod з busybox
    1.6. Створюємо HPA для деплоймента
2. Додаємо у кластер prometheus, fluentd, loki, grafana
3. Збирання логів та метрик до grafana

Передумови

Попередньо маємо налаштовану VM з встановленим Docker, та прописані у файлі hosts ip адреси нашої VM та nginx.edu.local і grafana.edu.local на 127.0.0.1. Також для цього завдання створюємо окрему теку lesson11 і всі подальші дії відбуваються саме в цій теці. Для виконання ДЗ буде потрібно створити ще додаткові теки для PV, але це є частиною ДЗ і буде виконано далі послідовно

!!Для перевірки ДЗ бажано використовувати yaml файли які додаються додатково до цього ДЗ, бо через копіпаст форматування може бути трохі змінене і це може зашкодити його застосуванню

1. Створення контейнера з busybox

1.1. Створюємо простір імен bb-namespace

Команда:

```
sudo microk8s.kubectl create namespace bb-namespace
```

Відгук:
namespace/bb-namespace created

1.2. Створюємо Persistent Volume (PV)

Створюємо теку /mnt/data/busybox

Команда:
```
sudo mkdir /mnt/data/busybox
```
Відгук:
...

Даємо теці права

Команда:
```
sudo chmod -R 777 /mnt/data/busybox
```
Відгук:

Далі стисло – команда – відгук

Створюємо файл yaml для Persistent Volume

Команда:
```
nano bb-pv.yaml
```
Відгук:
...

Заповнюємо файл наступним вмістом:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: bb-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: microk8s-hostpath
  hostPath:
    path: "/mnt/data/busybox"
```

Зберігаємо. Застосовуємо новостворений yaml файл

Команда:
```
sudo microk8s.kubectl apply -f bb-pv.yaml
```
Відгук:
persistentvolume/bb-pv created

Перевіряємо

Команда:

```
sudo microk8s.kubectl get pv
```

Відгук:

```
NAME                                       CAPACITY   ACCESS MODES
RECLAIM POLICY   STATUS      CLAIM
STORAGECLASS        VOLUMEATTRIBUTESCLASS   REASON   AGE
bb-pv                                      1Gi        RWO
Retain           Available
microk8s-hostpath   <unset>                            108s
```

1.3. Створюємо Persistent Volume Claim (PVC)

Створюємо файл yaml для Persistent Volume Claim

Команда:

```
nano bb-pvc.yaml
```

Відгук:

```
...
```

Заповнюємо файл наступним вмістом:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: bb-pvc
  namespace: bb-namespace
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: microk8s-hostpath
```

Зберігаємо. Застосовуємо новостворений yaml файл

Команда:

```
sudo microk8s.kubectl apply -f bb-pvc.yaml
```

Відгук:

```
persistentvolumeclaim/bb-pvc created
```

Перевіряємо

Команда:

```
sudo microk8s.kubectl get pvc -n bb-namespace
```

Відгук:

```
NAME     STATUS    VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS
VOLUMEATTRIBUTESCLASS   AGE
bb-pvc   Pending                                      microk8s-
hostpath   <unset>                 5m12s
```

1.4. Створюємо ConfigMap

Команда:

```
nano bb-config.yaml
```

Відгук:

...

Заповнюємо файл наступним вмістом:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: bb-config
  namespace: bb-namespace
data:
  MSGFILE: "bb-msg.log"
```

Зберігаємо. Застосовуємо новостворений yaml файл

Команда:

```
sudo microk8s.kubectl apply -f bb-config.yaml
```

Відгук:

```
configmap/bb-config created
```

Перевіряємо

Команда:

```
sudo microk8s.kubectl get configmap -n bb-namespace
```

Відгук:

```
NAME               DATA   AGE
bb-config          1      59s
```

1.5. Створюємо Deployment Pod з busybox

Команда:

```
nano bb-deployment.yaml
```

Відгук:

...

Заповнюємо файл наступним вмістом:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: bb-deployment
  namespace: bb-namespace
spec:
  replicas: 3
  selector:
    matchLabels:
      app: bb-app
  template:
```

```
    metadata:
      labels:
        app: bb-app
    spec:
      volumes:
      - name: bb-pvc
        persistentVolumeClaim:
          claimName: bb-pvc
      containers:
      - name: busybox
        image: busybox:1.28
        ports:
        - containerPort: 80
        resources:
          requests:
            cpu: "250m"
            memory: "128Mi"
          limits:
            cpu: "500m"
            memory: "256Mi"
        envFrom:
          - configMapRef:
              name: bb-config
        command:
          - "/bin/sh"
          - "-c"
          - 'while true; do echo "$(hostname) : $(date)"; echo "$
(hostname) : $(date)" >> "/mnt/data/busybox/$(MSGFILE)"; sleep 5;
done'
        volumeMounts:
        - mountPath: /mnt/data/busybox
          name: bb-pvc
```

Зберігаємо. Застосовуємо новостворений yaml файл

Команда:

```
sudo microk8s.kubectl apply -f bb-deployment.yaml
```

Відгук:

```
deployment.apps/bb-deployment created
```

Перевіряємо

Команда:

```
sudo microk8s.kubectl get pods -n bb-namespace
```

Відгук:

```
NAME                             READY   STATUS    RESTARTS   AGE
bb-deployment-5748d4c779-5q8rh   1/1     Running   0          29s
bb-deployment-5748d4c779-bst6c   1/1     Running   0          29s
bb-deployment-5748d4c779-nsjgp   1/1     Running   0          29s
```

Зменшуємо кількість подів

Команда:

```
sudo microk8s.kubectl scale deployment bb-deployment --
replicas=1 -n bb-namespace
```

Відгук:

```
deployment.apps/bb-deployment scaled
```

Перевіряємо

Команда:

```
sudo microk8s.kubectl get pods -n bb-namespace
```

Відгук:

```
bb-deployment-5748d4c779-8kc9s   1/1     Running   0          21m
```

Збільшуємо кількість подів

Команда:

```
sudo microk8s.kubectl scale deployment bb-deployment --
replicas=10 -n bb-namespace
```

Відгук:

```
deployment.apps/bb-deployment scaled
```

Перевіряємо

Команда:

```
sudo microk8s.kubectl get pods -n bb-namespace
```

Відгук:

```
NAME                             READY   STATUS    RESTARTS   AGE
bb-deployment-5748d4c779-8kc9s   1/1     Running   0          26m
bb-deployment-5748d4c779-92j85   0/1     Pending   0          70s
bb-deployment-5748d4c779-d84lq   0/1     Pending   0          70s
bb-deployment-5748d4c779-dx6mq   1/1     Running   0          70s
bb-deployment-5748d4c779-gcwlc   0/1     Pending   0          70s
bb-deployment-5748d4c779-gn2xp   1/1     Running   0          70s
bb-deployment-5748d4c779-lzrsc   0/1     Pending   0          70s
bb-deployment-5748d4c779-pv6w6   1/1     Running   0          70s
bb-deployment-5748d4c779-vmbdl   1/1     Running   0          70s
bb-deployment-5748d4c779-ztbzs   1/1     Running   0          70s
```

Є думка, шо деякі поди не стартанули, того шо в кожному відпрацьовує команда, яка кожні п’ять секунд згідно ДЗ пише на диск у файл і в STDOUT hostname контейнера й дату. Також ми вказали у нашому деплойменті кількість ресурсів які виділяються для одного POD, і скоріше всього цих ресурсів для 10 POD не достатньо, того вони стоять в очікуванні. З 5 подами все ок, як ви побачите далі у прикладі з HPA. 6 подів працюють ок, перевірив тіку шо, тільки збільшую до 7, то 7 под має статус – Pending

Перевірка запису у STDOUT

Команда:

```
sudo microk8s.kubectl logs bb-deployment-5748d4c779-8kc9s -n
bb-namespace
```

Відгук:
```
bb-deployment-5748d4c779-8kc9s : Sun Mar  9 12:04:09 UTC 2025
bb-deployment-5748d4c779-8kc9s : Sun Mar  9 12:04:14 UTC 2025
bb-deployment-5748d4c779-8kc9s : Sun Mar  9 12:04:19 UTC 2025
bb-deployment-5748d4c779-8kc9s : Sun Mar  9 12:04:24 UTC 2025
bb-deployment-5748d4c779-8kc9s : Sun Mar  9 12:04:29 UTC 2025
bb-deployment-5748d4c779-8kc9s : Sun Mar  9 12:04:34 UTC 2025
bb-deployment-5748d4c779-8kc9s : Sun Mar  9 12:04:39 UTC 2025
bb-deployment-5748d4c779-8kc9s : Sun Mar  9 12:04:44 UTC 2025
bb-deployment-5748d4c779-8kc9s : Sun Mar  9 12:04:49 UTC 2025
bb-deployment-5748d4c779-8kc9s : Sun Mar  9 12:04:54 UTC 2025
bb-deployment-5748d4c779-8kc9s : Sun Mar  9 12:04:59 UTC 2025
```

Там ціле простирадло, того кинув лише частину лога, сподіваюся цього достатньо
В файл на диску всі працюючи поди також пишуть без зауважень
Шмат лога для підтвердження нижче:

```
bb-deployment-5748d4c779-8kc9s : Sun Mar  9 12:38:46 UTC 2025
bb-deployment-5748d4c779-gn2xp : Sun Mar  9 12:38:48 UTC 2025
bb-deployment-5748d4c779-vmbdl : Sun Mar  9 12:38:48 UTC 2025
bb-deployment-5748d4c779-pv6w6 : Sun Mar  9 12:38:48 UTC 2025
bb-deployment-5748d4c779-dx6mq : Sun Mar  9 12:38:48 UTC 2025
bb-deployment-5748d4c779-ztbzs : Sun Mar  9 12:38:48 UTC 2025
bb-deployment-5748d4c779-8kc9s : Sun Mar  9 12:38:51 UTC 2025
bb-deployment-5748d4c779-gn2xp : Sun Mar  9 12:38:53 UTC 2025
bb-deployment-5748d4c779-vmbdl : Sun Mar  9 12:38:53 UTC 2025
bb-deployment-5748d4c779-pv6w6 : Sun Mar  9 12:38:53 UTC 2025
bb-deployment-5748d4c779-ztbzs : Sun Mar  9 12:38:53 UTC 2025
bb-deployment-5748d4c779-dx6mq : Sun Mar  9 12:38:53 UTC 2025
bb-deployment-5748d4c779-8kc9s : Sun Mar  9 12:38:56 UTC 2025
bb-deployment-5748d4c779-8kc9s : Sun Mar  9 12:39:01 UTC 2025
bb-deployment-5748d4c779-8kc9s : Sun Mar  9 12:39:06 UTC 2025
```

1.6. Створюємо HPA для деплоймента

Створюємо сервіс для нашого деплоймента, не впевнений шо він загалом потрібен саме для цього контейнера з busybox, або для HPA. Але най буде, раптом шо, є сервіс і порти які можна використовувати

Команда:
```
     nano bb-svc.yaml
```
Відгук:
```
...
```

Заповнюємо файл наступним вмістом:

```
apiVersion: v1
kind: Service
metadata:
 name: bb-svc
```

```
 namespace: bb-namespace
 labels:
   run: bb-app
spec:
 ports:
 - protocol: TCP
   port: 80
   targetPort: 80
   nodePort: 30080
 type: NodePort
 selector:
   run: bb-app
```

Зберігаємо. Застосовуємо новостворений yaml файл

Команда:

```
sudo microk8s.kubectl apply -f bb-svc.yaml
```

Відгук:

```
service/bb-svc created
```

Перевіряємо

Команда:

```
sudo microk8s.kubectl get svc -n bb-namespace
```

Відгук:

```
NAME     TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)
AGE
bb-svc   NodePort   10.152.183.74   <none>        80:30080/TCP
84s
```

Створюємо yaml файл для HPA

Тут треба обмовитися, шо можна було поставити minReplicas в 1, але наш деплоймент зі старту має 3 репліки, і мені здається, шо знижувати власні спроможності зі старту HPA якось ризиковано. А також ми одразу можемо подивитися чи спрацював наш HPA збільшивши кількість подів. Хоча може я й помиляюся, бо там же чиста динаміка дій

Команда:

```
nano bb-hpa.yaml
```

Відгук:

```
...
```

Заповнюємо файл наступним вмістом:

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
 name: bb-hpa
 namespace: bb-namespace
spec:
```

```
 scaleTargetRef:
   apiVersion: apps/v1
   kind: Deployment
   name: bb-deployment
 minReplicas: 5
 maxReplicas: 10
 targetCPUUtilizationPercentage: 50
```

Зберігаємо. Застосовуємо новостворений yaml файл

Команда:

```
sudo microk8s.kubectl apply -f bb-hpa.yaml
```

Відгук:

```
horizontalpodautoscaler.autoscaling/bb-hpa created
```

Перевіряємо

Команда:

```
sudo microk8s.kubectl get hpa -n bb-namespace
```

Відгук:

```
NAME     REFERENCE                   TARGETS       MINPODS
MAXPODS   REPLICAS   AGE
bb-hpa   Deployment/bb-deployment   cpu: 0%/50%   5         10
5          82s
```

Команда:

```
sudo microk8s.kubectl get pods -n bb-namespace
```

Відгук:

```
NAME                             READY   STATUS    RESTARTS   AGE
bb-deployment-5748d4c779-5q8rh   1/1     Running   0          15m
bb-deployment-5748d4c779-8kc9s   1/1     Running   0
2m22s
bb-deployment-5748d4c779-bst6c   1/1     Running   0          15m
bb-deployment-5748d4c779-nsjgp   1/1     Running   0          15m
bb-deployment-5748d4c779-wlmbr   1/1     Running   0
2m22s
```

2. Додаємо у кластер prometheus, fluentd, loki, grafana

Для виконання другої частини ДЗ створимо все знову, шоб не плутатися у просторах імен та іншому. Для чистоти експерименту, бо не маючи майже ніякого досвіду, легко зробити шось не правильно. Для цього буде використана розшифровка уроку №12

Створимо простір імен для NGINX

Команда:

```
sudo microk8s.kubectl create namespace nginx
```

Відгук:

```
namespace/nginx created
```

Створимо Persistent Volume для постійного зберігання наших данних

Команда:

```
nano ng-pv.yaml
```

Відгук:

...

Заповнюємо файл наступним вмістом:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nginx-php-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: microk8s-hostpath
  hostPath:
    path: "/mnt/data/nginx-php"
```

Зберігаємо і застосовуємо цей файл, попередньо створивши відповідні теки у хостовий файловій системі

```
sudo mkdir -p /mnt/data/nginx-php
```

Також додамо в новостворену теку index файл для нашого NGINX

```
sudo tee /mnt/data/nginx-php/index.php <<EOF
<?php
echo "Hello from " . gethostname();
?>
EOF
```

Дамо рекурсивно права на файли і теки

```
sudo chmod -R 777 /mnt/data/nginx-php
```

Тепер застосуємо наш yaml файл для PersistentVolume

Команда:

```
sudo microk8s.kubectl apply -f ng-pv.yaml
```

Відгук:

```
persistentvolume/nginx-php-pv created
```

Тепер створимо Persistent Volume Claim для того шоб отримати доступ до створеного нами раніше Persistent Volume

Команда:

```
nano ng-pvc.yaml
```

Відгук:

```
...
```

Заповнюємо файл наступним вмістом

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nginx-php-pvc
  namespace: nginx
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: microk8s-hostpath
```

Зберігаємо і застосовуємо

Команда:

```
sudo microk8s.kubectl apply -f ng-pvc.yaml
```

Відгук:

```
persistentvolumeclaim/nginx-php-pvc created
```

Створимо **ConfigMap** для деплоймента NGINX, де вкажемо налаштування по замовченню

Команда:

```
nano nginx-configmap.yaml
```

Відгук:

```
...
```

Заповнюємо файл наступним вмістом

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx-config
  namespace: nginx
data:
  default.conf: |
    server {
        listen 80;
        server_name localhost;
        root /var/www/html;
        index index.php index.html index.htm;
        location / {
            try_files $uri $uri/ =404;
```

```
        }
        location ~ \.php$ {
            include fastcgi_params;
            fastcgi_pass 127.0.0.1:9000;
            fastcgi_index index.php;
            fastcgi_param SCRIPT_FILENAME
$document_root$fastcgi_script_name;
        }
    }
```
Зберігаємо і застосовуємо цей файл

Команда:
```
sudo microk8s.kubectl apply -f nginx-configmap.yaml
```
Відгук:
```
configmap/nginx-config created
```

Робимо Deployment Nginx + PHP

Команда:
```
nano ng-deployment.yaml
```
Відгук:
```
...
```

Заповнюємо файл наступним вмістом

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-php
  namespace: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx-php
  template:
    metadata:
      labels:
        app: nginx-php
    spec:
      containers:
        - name: nginx
          image: nginx:latest
          ports:
            - containerPort: 80
          volumeMounts:
            - name: nginx-php-storage
              mountPath: /var/www/html
            - name: nginx-config
              mountPath: /etc/nginx/conf.d/default.conf
              subPath: default.conf
        - name: php-fpm
```

```
          image: php:fpm
          volumeMounts:
            - name: nginx-php-storage
              mountPath: /var/www/html
      volumes:
        - name: nginx-php-storage
          persistentVolumeClaim:
            claimName: nginx-php-pvc
        - name: nginx-config
          configMap:
            name: nginx-config
```

Зберігаємо і застосовуємо цей файл

Команда:

```
sudo microk8s.kubectl apply -f ng-deployment.yaml
```

Відгук:

```
deployment.apps/nginx-php created
```

Створимо Service для Nginx де вкажемо протоколи й порти за якими він буде доступний

Команда:

```
nano ng-service.yaml
```

Відгук:

```
...
```

Заповнюємо файл наступним вмістом

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-php-service
  namespace: nginx
spec:
  selector:
    app: nginx-php
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
      nodePort: 30080
  type: NodePort
```

Зберігаємо і застосовуємо цей файл

Команда:

```
sudo microk8s.kubectl apply -f ng-service.yaml
```

Відгук:

```
service/nginx-php-service created
```

Тепер перевіримо чи працює те шо ми зробили

Команда:

```
curl http://localhost:30080
```

Відгук:

```
Hello from nginx-php-96697cbc-bkkhd
```

Працює!))

Переходимо до частини моніторингу
Створимо для цього окремий простір імен

Команда:

```
sudo microk8s.kubectl create namespace monitoring
```

Відгук:

```
namespace/monitoring created
```

Також створимо теку для loki на файловій хост-системі і дамо їй відповідні права

```
sudo mkdir /mnt/data/loki
sudo chmod -R 777 /mnt/data/loki
```

Тепер створимо Persistent Volume для loki

Команда:

```
nano loki-pv.yaml
```

Відгук:

```
...
```

Заповнюємо файл наступним вмістом

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: loki-pv
spec:
  capacity:
    storage: 2Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: microk8s-hostpath
  hostPath:
    path: "/mnt/data/loki"
```

Зберігаємо і застосовуємо файл

Команда:

```
sudo microk8s.kubectl apply -f loki-pv.yaml
```

Відгук:

```
persistentvolume/loki-pv created
```

Створюємо Persistent Volume Claim для loki

Команда:

```
nano loki-pvc.yaml
```

Відгук:

```
...
```

Заповнюємо файл наступним вмістом

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: loki-pvc
  namespace: monitoring
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi
  storageClassName: microk8s-hostpath
```

Зберігаємо і застосовуємо цей файл

Команда:

```
sudo microk8s.kubectl apply -f loki-pvc.yaml
```

Відгук:

```
persistentvolumeclaim/loki-pvc created
```

Створюємо ConfigMap для налаштувань Loki

Команда:

```
nano loki-config.yaml
```

Відгук:

```
...
```

Заповнюємо файл наступним вмістом

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: loki-config
  namespace: monitoring
data:
  loki-config.yaml: |
    auth_enabled: false
    server:
      http_listen_port: 3100
    ingester:
      wal:
        enabled: true
        dir: /var/loki/wal
```

```
      lifecycler:
        address: 127.0.0.1
        ring:
          kvstore:
            store: inmemory
          replication_factor: 1
      chunk_idle_period: 5m
      chunk_retain_period: 30s
    schema_config:
      configs:
        - from: 2020-10-24
          store: boltdb-shipper
          object_store: filesystem
          schema: v11
          index:
            prefix: index_
            period: 24h
    storage_config:
      boltdb_shipper:
        active_index_directory: /var/loki/index
        cache_location: /var/loki/index_cache
        shared_store: filesystem
      filesystem:
        directory: /var/loki/chunks
    compactor:
      working_directory: /var/loki/compactor
      shared_store: filesystem
```

Зберігаємо і застосовуємо цей файл

Команда:

```
sudo microk8s.kubectl apply -f loki-config.yaml
```

Відгук:

```
configmap/loki-config created
```

Робимо Deployment Loki

Команда:

```
nano loki-deployment.yaml
```

Відгук:

```
...
```

Заповнюємо файл наступним вмістом

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: loki
  namespace: monitoring
spec:
  replicas: 1
  selector:
```

```
    matchLabels:
      app: loki
  template:
    metadata:
      labels:
        app: loki
    spec:
      containers:
        - name: loki
          image: grafana/loki:2.8.2
          args:
            - -config.file=/etc/loki/loki-config.yaml
          ports:
            - containerPort: 3100
          volumeMounts:
            - name: config
              mountPath: /etc/loki
            - name: storage
              mountPath: /var/loki
      volumes:
        - name: config
          configMap:
            name: loki-config
        - name: storage
          persistentVolumeClaim:
            claimName: loki-pvc
```

Зберігаємо і застосовуємо цей файл

Команда:

```
sudo microk8s.kubectl apply -f loki-deployment.yaml
```

Відгук:

```
deployment.apps/loki created
```

Перевірмо чи працює loki

Команда:

```
sudo microk8s.kubectl logs -n monitoring -l app=loki
```

Відгук:

```
level=info ts=2025-03-10T12:52:35.840479785Z caller=loki.go:499
msg="Loki started"
level=info ts=2025-03-10T12:52:38.842361222Z caller=worker.go:209
msg="adding connection" addr=127.0.0.1:9095
level=info ts=2025-03-10T12:52:38.842437858Z
caller=scheduler.go:681 msg="this scheduler is in the
ReplicationSet, will now accept requests."
level=info ts=2025-03-10T12:52:40.841000808Z
caller=compactor.go:411 msg="this instance has been chosen to run
the compactor, starting compactor"
level=info ts=2025-03-10T12:52:40.841201947Z
caller=compactor.go:440 msg="waiting 10m0s for ring to stay stable
and previous compactions to finish before starting compactor"
```

```
level=info ts=2025-03-10T12:52:45.845461965Z
caller=frontend_scheduler_worker.go:107 msg="adding connection to
scheduler" addr=127.0.0.1:9095
level=info ts=2025-03-10T12:53:35.678397691Z
caller=table_manager.go:134 msg="uploading tables"
level=info ts=2025-03-10T12:53:35.678459951Z
caller=table_manager.go:166 msg="handing over indexes to shipper"
level=info ts=2025-03-10T12:54:35.674088633Z
caller=table_manager.go:134 msg="uploading tables"
level=info ts=2025-03-10T12:54:35.674132197Z
caller=table_manager.go:166 msg="handing over indexes to shipper"
```

Бачимо шо loki стартував і вже йдуть його логи

Тепер додамо Service шоб Loki став доступний на порту 3100

Команда:

```
nano loki-service.yaml
```

Відгук:

```
...
```

Заповнимо файл відповідним вмістом

```
apiVersion: v1
kind: Service
metadata:
  name: loki
  namespace: monitoring
spec:
  selector:
    app: loki
  ports:
    - protocol: TCP
      port: 3100
      targetPort: 3100
```

Зберігаємо і застосовуємо цей файл

Команда:

```
sudo microk8s.kubectl apply -f loki-service.yaml
```

Відгук:

```
service/loki created
```

Перевіряємо створений сервіс

Команда:

```
sudo microk8s.kubectl exec -it nginx-php-96697cbc-hf248 -n
nginx — sh
```

Відгук:

```
Defaulted container "nginx" out of: nginx, php-fpm
#
```

Команда:

```
     curl http://loki.monitoring.svc.cluster.local:3100/ready
```
Відгук:
```
ready
```

Сервіс для loki працює на призначеному порту і loki відгукається

Для збирання, обробки та пересилки наших логів та метрик налаштуємо Fluent Bit

Створюємо ConfigMap з налаштуваннями для Fluent Bit

Команда:
```
     nano fluentbit-config.yaml
```
Відгук:
```
...
```

Заповнимо файл відповідним вмістом

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: fluentbit-config
  namespace: monitoring
data:
  fluent-bit.conf: |
    [SERVICE]
        Flush        5
        Log_Level    info
    [INPUT]
        Name              tail
        Path              /var/log/containers/*.log
        Parser            docker
        Tag               kube.*
        Refresh_Interval  5
    [OUTPUT]
        Name        loki
        Match       *
        Host        loki
        Port        3100
        Labels      job=nginx-logs
```

Зберігаємо і застосовуємо файл fluentbit-config.yaml

Команда:
```
     sudo microk8s.kubectl apply -f fluentbit-config.yaml
```
Відгук:
```
configmap/fluentbit-config created
```

Тепер для запуску Fluent Bit створимо для нього Daemonset

Команда:

```
     nano fluentbit-deployment.yaml
```
Відгук:
```
...
```

Заповнюємо файл наступним вмістом

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluentbit
  namespace: monitoring
spec:
  selector:
    matchLabels:
      app: fluentbit
  template:
    metadata:
      labels:
        app: fluentbit
    spec:
      containers:
        - name: fluentbit
          image: fluent/fluent-bit:latest
          volumeMounts:
            - name: varlog
              mountPath: /var/log
            - name: config
              mountPath: /fluent-bit/etc/
      volumes:
        - name: varlog
          hostPath:
            path: /var/log
        - name: config
          configMap:
            name: fluentbit-config
```

Зберігаємо і застосовуємо цей файл

Команда:
```
     sudo microk8s.kubectl apply -f fluentbit-deployment.yaml
```
Відгук:
```
daemonset.apps/fluentbit created
```

Перевірмо наші новостворені створені поди які працюють у просторі імен monitoring

Команда:
```
     sudo microk8s.kubectl get pods -n monitoring
```
Відгук:
```
NAME                    READY   STATUS    RESTARTS   AGE
fluentbit-rpp4t         1/1     Running   0          2m10s
loki-78d7b7458c-d5gth   1/1     Running   0          22h
```

Бачимо шо демон fluentbit вдало стартував і працює, а loki загалом працює ще з учора)) Продовжуємо

Робимо деплоймент Grafana

Команда:

```
nano grafana-deployment.yaml
```

Відгук:

```
...
```

Заповнюємо файл наступним вмістом:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: grafana
  namespace: monitoring
spec:
  replicas: 1
  selector:
    matchLabels:
      app: grafana
  template:
    metadata:
      labels:
        app: grafana
    spec:
      containers:
        - name: grafana
          image: grafana/grafana
          ports:
            - containerPort: 3000
```

Зберігаємо і застосовуємо файл

Команда:

```
sudo microk8s.kubectl apply -f grafana-deployment.yaml
```

Відгук:

```
deployment.apps/grafana created
```

Перевіримо

Команда:

```
sudo microk8s.kubectl get pods -n monitoring
```

Відгук:

```
NAME                       READY   STATUS    RESTARTS   AGE
fluentbit-rpp4t            1/1     Running   0          19m
grafana-85b785d45d-5t4xl   1/1     Running   0          8m9s
loki-78d7b7458c-d5gth      1/1     Running   0          22h
```

Бачимо шо grafana стартувала і працює разом з іншими нашими подами

Додамо сервіс для grafana

Команда:
```
nano grafana-service.yaml
```
Відгук:
```
...
```

Заповнюємо файл наступним вмістом:

```
apiVersion: v1
kind: Service
metadata:
  name: grafana
  namespace: monitoring
spec:
  selector:
    app: grafana
  ports:
    - protocol: TCP
      port: 3000
```

Зберігаємо і застосовуємо файл

Команда:
```
sudo microk8s.kubectl apply -f grafana-service.yaml
```
Відгук:
```
service/grafana created
```

Давайте перевірмо і наші створені сервіси

Команда:
```
sudo microk8s.kubectl get svc -n monitoring
```
Відгук:
```
NAME      TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)
AGE
grafana   ClusterIP   10.152.183.233   <none>        3000/TCP
3m45s
loki      ClusterIP   10.152.183.62    <none>        3100/TCP
47h
```

Працюють рідненькі))

Бачимо шо сервіси для графани і локі мають власні айпі дреси і працюю на заданих для них портах

Тепер створимо Ingress для Grafana

Команда:
```
nano grafana-ingress.yaml
```

Відгук:
...

Заповнюємо файл відповідним вмістом:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: grafana-ingress
  namespace: monitoring
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: public
  rules:
  - host: grafana.edu.local
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: grafana
            port:
              number: 3000
```

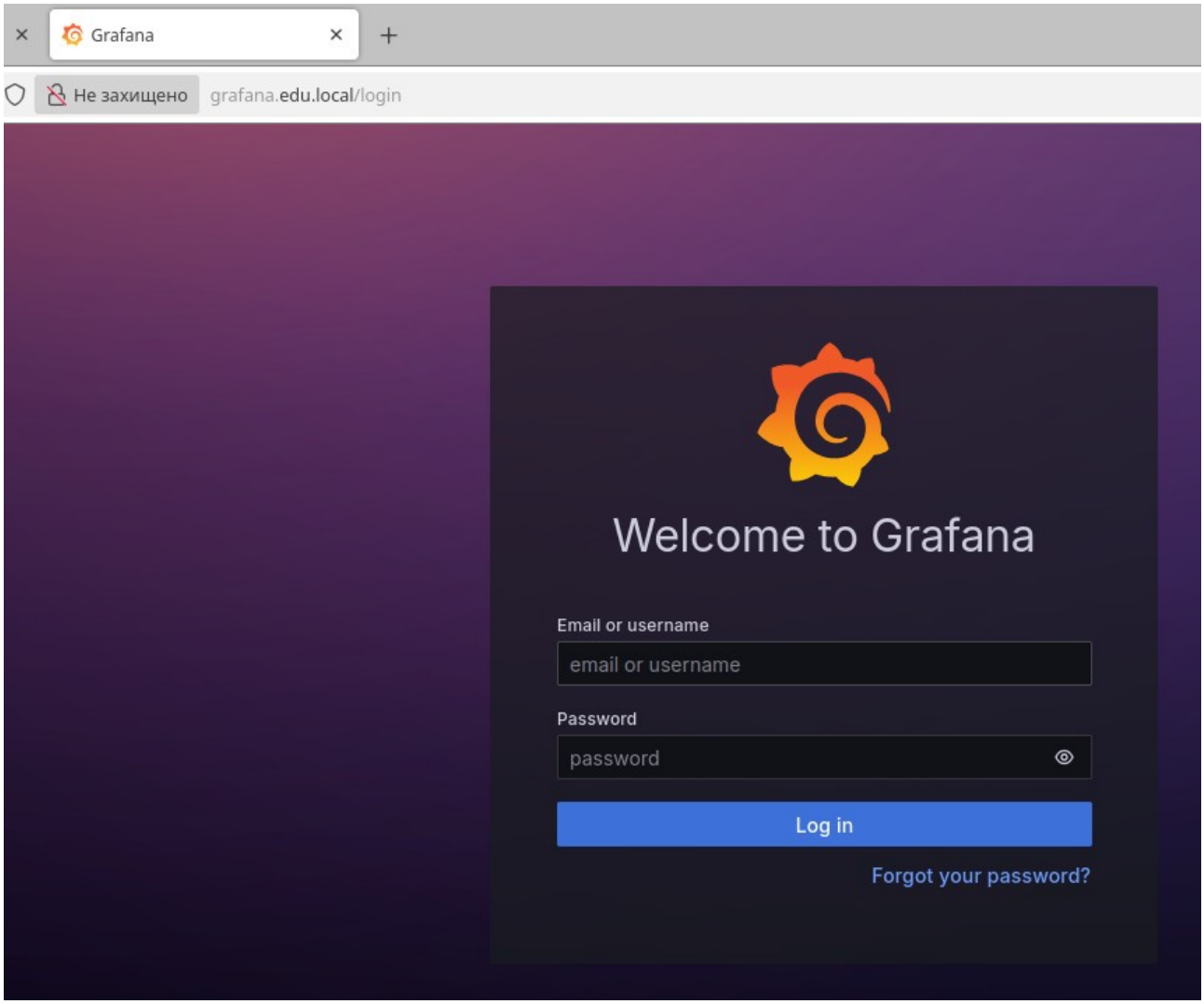
Зберігаємо і застосовуємо цей файл

Команда:
```
     sudo microk8s.kubectl apply -f grafana-ingress.yaml
```
Відгук:
```
ingress.networking.k8s.io/grafana-ingress created
```

Перевірмо чи з’явився він

Команда:
```
     sudo microk8s.kubectl get ingress –all-namespaces
```
Відгук:
```
NAMESPACE    NAME              CLASS    HOSTS
ADDRESS     PORTS   AGE
monitoring   grafana-ingress   public   grafana.edu.local
127.0.0.1   80      3m42s
```

Бачимо шо він є, працює у просторі імен monitoring, має хост ім’я grafana.edu.local та ip 127.0.0.1 і використовує 80 порт. Додамо запис у hosts файл нашої VM, тобто прив’яжемо grafana.edu.local до ip 127.0.0.1. І тепер можемо звернутися по цьому імені до grafana через наш браузер



Все ок, grafana доступна через браузер, здається наш ingress відпрацював

Перевірмо командою curl

Команда:

```
curl http://grafana.edu.local:80
```

Відгук:

```
<a href="/login">Found</a>.
```

І тут все ок, grafana відгукається

Тепер додамо Ingress для Nginx

Команда:

```
nano nginx-ingress.yaml
```

Відгук:

```
...
```

Заповнюємо файл наступним вмістом

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: nginx-ingress
  namespace: nginx
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: public
  rules:
  - host: nginx.edu.local
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: nginx-php-service
            port:
              number: 80
```

Зберігаємо і застосовуємо цей файл

Команда:

```
sudo microk8s.kubectl apply -f nginx-ingress.yaml
```

Відгук:

```
ingress.networking.k8s.io/nginx-ingress created
```

Перевіряємо

Команда:

```
sudo microk8s.kubectl get ingress --all-namespaces
```

Відгук:

```
NAMESPACE    NAME              CLASS    HOSTS
ADDRESS     PORTS   AGE
monitoring   grafana-ingress   public   grafana.edu.local
127.0.0.1   80      128m
nginx        nginx-ingress     public   nginx.edu.local
127.0.0.1   80      70s
```

Бачимо шо ingress для nginx створився, має власну зовнішню адресу, локальний ip VM та 80 порт. Додамо і для нього запис у hosts файл і перевірмо як це працює

Команда:

```
curl http://nginx.edu.local:80
```

Відгук:

```
Hello from nginx-php-96697cbc-hf248
```

Чудово працює

Подивимося на всі наші працюючи контейнери у просторі імен monitoring

Команда:

```
sudo microk8s.kubectl get pods -n monitoring
```

Відгук:

```
NAME                       READY   STATUS    RESTARTS   AGE
fluentbit-rpp4t            1/1     Running   0          169m
grafana-85b785d45d-5t4xl   1/1     Running   0          158m
loki-78d7b7458c-d5gth      1/1     Running   0          25h
```

Бачимо шо їх у нас три, всі мають по одному екземпляру і працюють добре

Тепер подивимося на наш nginx, він працює у власному просторі імен nginx

Команда:

```
sudo microk8s.kubectl get pods -n nginx
```

Відгук:

```
NAME                       READY   STATUS    RESTARTS   AGE
nginx-php-96697cbc-hf248   2/2     Running   0          25h
```

Бачимо шо POD працює і має два контейнера, це ті контейнери шо ми прописали у нього в deployment файлі – nginx та php-fpm. Працює вже більше доби, кількість реплік я навмисно опустив до 1, шоб заощадити ресурси. Подивимось на цей POD:

Команда:

```
sudo microk8s.kubectl describe pod nginx-php-96697cbc-hf248 -
n nginx
```

Відгук:

```
Name:             nginx-php-96697cbc-hf248
Namespace:        nginx
Priority:         0
Service Account:  default
Node:             virtualbird/10.0.2.15
Start Time:       Tue, 11 Mar 2025 14:14:58 +0000
Labels:           app=nginx-php
                  pod-template-hash=96697cbc
Annotations:      cni.projectcalico.org/containerID:
7a544c175691681a97ff602204b28d9e59b257249f43b6c44869f30f6aeaba0e
                  cni.projectcalico.org/podIP: 10.1.6.84/32
                  cni.projectcalico.org/podIPs: 10.1.6.84/32
Status:           Running
IP:               10.1.6.84
IPs:
  IP:           10.1.6.84
Controlled By:  ReplicaSet/nginx-php-96697cbc
Containers:
  nginx:
```

```
    Container ID:
containerd://42b627ee5a1c437b6118a05878fc0644f08d3131e2f7a88e8f2bc
1bcb7ee1dd9
    Image:          nginx:latest
    Image ID:
docker.io/library/nginx@sha256:9d6b58feebd2dbd3c56ab5853333d627cc6
e281011cfd6050fa4bcf2072c9496
    Port:           80/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Tue, 11 Mar 2025 14:15:00 +0000
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /etc/nginx/conf.d/default.conf from nginx-config
(rw,path="default.conf")
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-
access-wwws5 (ro)
      /var/www/html from nginx-php-storage (rw)
  php-fpm:
    Container ID:
containerd://4725815f1814780fa20fdd5657809f0bb16aceccf02ba6a98ab26
b512a3e6e3b
    Image:          php:fpm
    Image ID:
docker.io/library/php@sha256:775866885609759de2ea7ebf4deff118431b6
32fce2b638244c69fe477f3e715
    Port:           <none>
    Host Port:      <none>
    State:          Running
      Started:      Tue, 11 Mar 2025 14:15:00 +0000
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-
access-wwws5 (ro)
      /var/www/html from nginx-php-storage (rw)
Conditions:
  Type                        Status
  PodReadyToStartContainers   True
  Initialized                 True
  Ready                       True
  ContainersReady             True
  PodScheduled                True
Volumes:
  nginx-php-storage:
    Type:       PersistentVolumeClaim (a reference to a
PersistentVolumeClaim in the same namespace)
    ClaimName:  nginx-php-pvc
    ReadOnly:   false
```

```
  nginx-config:
    Type:      ConfigMap (a volume populated by a ConfigMap)
    Name:      nginx-config
    Optional:  false
  kube-api-access-wwws5:
    Type:                     Projected (a volume that contains
injected data from multiple sources)
    TokenExpirationSeconds:  3607
    ConfigMapName:           kube-root-ca.crt
    ConfigMapOptional:       <nil>
    DownwardAPI:             true
QoS Class:                   BestEffort
Node-Selectors:              <none>
Tolerations:                 node.kubernetes.io/not-
ready:NoExecute op=Exists for 300s

node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:                      <none>
```

Тепер почнемо ту частину ДЗ в якої додамо Prometeus, а також змінимо файл деплойменту loki додаванням до нього Liveness і Readiness Probes і додамо метрики

Додаємо Liveness і Readiness Probes до деплоймент файлу loki

Команда:

nano loki-deployment.yaml

Відгук:

…

І додаємо строки виділені червоним (у pdf версії) в те саме місце у деплойменті:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: loki
  namespace: monitoring
spec:
  replicas: 1
  selector:
    matchLabels:
      app: loki
  template:
    metadata:
      labels:
        app: loki
    spec:
      containers:
        - name: loki
          image: grafana/loki:2.8.2
          args:
```

```
            - -config.file=/etc/loki/loki-config.yaml
          ports:
            - containerPort: 3100
          livenessProbe:
            httpGet:
              path: /ready
              port: 3100
            initialDelaySeconds: 30
            periodSeconds: 10
            failureThreshold: 3
          readinessProbe:
            httpGet:
              path: /ready
              port: 3100
            initialDelaySeconds: 10
            periodSeconds: 5
            successThreshold: 1
          volumeMounts:
            - name: config
              mountPath: /etc/loki
            - name: storage
              mountPath: /var/loki
      volumes:
        - name: config
          configMap:
            name: loki-config
        - name: storage
          persistentVolumeClaim:
            claimName: loki-pvc
```

Зберігаємо і застосовуємо цей файл

Команда:

```
sudo microk8s.kubectl apply -f loki-deployment.yaml
```

Відгук:

```
deployment.apps/loki configured
```

Команда:

```
sudo microk8s.kubectl get pods -n monitoring
```

Відгук:

```
NAME                       READY   STATUS    RESTARTS   AGE
fluentbit-rpp4t            1/1     Running   0          4h22m
grafana-85b785d45d-5t4xl   1/1     Running   0          4h11m
loki-69b5f68c75-fpccg      1/1     Running   0          94s
```

Бачимо шо под з локі перестворився

Команда:

```
sudo microk8s.kubectl describe pod loki-69b5f68c75-fpccg -n
monitoring
```

Відгук:

```
Name:             loki-69b5f68c75-fpccg
```

```
Namespace:        monitoring
Priority:         0
Service Account:  default
Node:             virtualbird/10.0.2.15
Start Time:       Wed, 12 Mar 2025 17:09:16 +0000
Labels:           app=loki
                  pod-template-hash=69b5f68c75
Annotations:      cni.projectcalico.org/containerID:
97150bcdaeef8e1da60b69fa7f2b27b9fd9abf974368973bff31605b774293c8
                  cni.projectcalico.org/podIP: 10.1.6.126/32
                  cni.projectcalico.org/podIPs: 10.1.6.126/32
Status:           Running
IP:               10.1.6.126
IPs:
  IP:           10.1.6.126
Controlled By:  ReplicaSet/loki-69b5f68c75
Containers:
  loki:
    Container ID:
containerd://f659aef8eb65e5d411fc2f5ccfe1fb979b391888e322fbe3d6490
3481a3f044d
    Image:         grafana/loki:2.8.2
    Image ID:
docker.io/grafana/loki@sha256:b1da1d23037eb1b344cccfc5b587e30aed60
ab4cad33b42890ff850aa3c4755d
    Port:          3100/TCP
    Host Port:     0/TCP
    Args:
      -config.file=/etc/loki/loki-config.yaml
    State:          Running
      Started:      Wed, 12 Mar 2025 17:09:16 +0000
    Ready:          True
    Restart Count:  0
    Liveness:       http-get http://:3100/ready delay=30s
timeout=1s period=10s #success=1 #failure=3
    Readiness:      http-get http://:3100/ready delay=10s
timeout=1s period=5s #success=1 #failure=3
    Environment:    <none>
    Mounts:
      /etc/loki from config (rw)
      /var/loki from storage (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-
access-r7s4v (ro)
Conditions:
  Type                        Status
  PodReadyToStartContainers   True
  Initialized                 True
  Ready                       True
  ContainersReady             True
  PodScheduled                True
Volumes:
  config:
```

```
    Type:      ConfigMap (a volume populated by a ConfigMap)
    Name:      loki-config
    Optional:  false
  storage:
    Type:       PersistentVolumeClaim (a reference to a
PersistentVolumeClaim in the same namespace)
    ClaimName:  loki-pvc
    ReadOnly:   false
  kube-api-access-r7s4v:
    Type:                     Projected (a volume that contains
injected data from multiple sources)
    TokenExpirationSeconds:  3607
    ConfigMapName:           kube-root-ca.crt
    ConfigMapOptional:       <nil>
    DownwardAPI:             true
QoS Class:                   BestEffort
Node-Selectors:              <none>
Tolerations:                 node.kubernetes.io/not-
ready:NoExecute op=Exists for 300s

node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:
  Type     Reason     Age                  From
Message
  ----     ------     ----                 ----
-------
  Normal   Scheduled  2m14s                default-scheduler
Successfully assigned monitoring/loki-69b5f68c75-fpccg to
virtualbird
  Normal   Pulled     2m15s                kubelet
Container image "grafana/loki:2.8.2" already present on machine
  Normal   Created    2m15s                kubelet
Created container: loki
  Normal   Started    2m15s                kubelet
Started container loki
  Warning  Unhealthy  2m1s                 kubelet
Readiness probe failed: Get "http://10.1.6.126:3100/ready":
context deadline exceeded (Client.Timeout exceeded while awaiting
headers)
  Warning  Unhealthy  107s (x3 over 117s)  kubelet
Readiness probe failed: HTTP probe failed with statuscode: 503
  Warning  Unhealthy  105s                 kubelet
Liveness probe failed: HTTP probe failed with statuscode: 503
```

Пере-створився, але шось у нього не все добре
Перевірмо події

Команда:
```
sudo microk8s.kubectl events -n monitoring
```
Відгук:
```
LAST SEEN               TYPE      REASON              OBJECT
MESSAGE
```

```
5m15s                   Normal    Pulled              Pod/loki-
69b5f68c75-fpccg    Container image "grafana/loki:2.8.2" already
present on machine
5m15s                   Normal    Created             Pod/loki-
69b5f68c75-fpccg    Created container: loki
5m15s                   Normal    Started             Pod/loki-
69b5f68c75-fpccg    Started container loki
5m1s                    Warning   Unhealthy           Pod/loki-
69b5f68c75-fpccg    Readiness probe failed: Get
"http://10.1.6.126:3100/ready": context deadline exceeded
(Client.Timeout exceeded while awaiting headers)
4m47s (x3 over 4m57s)   Warning   Unhealthy           Pod/loki-
69b5f68c75-fpccg    Readiness probe failed: HTTP probe failed with
statuscode: 503
4m45s                   Warning   Unhealthy           Pod/loki-
69b5f68c75-fpccg    Liveness probe failed: HTTP probe failed with
statuscode: 503
4m42s                   Normal    Killing             Pod/loki-
78d7b7458c-d5gth    Stopping container loki
4m42s                   Normal    SuccessfulDelete
ReplicaSet/loki-78d7b7458c   Deleted pod: loki-78d7b7458c-d5gth
4m42s                   Normal    ScalingReplicaSet
Deployment/loki             Scaled down replica set loki-
78d7b7458c from 1 to 0
```

Шось там не добре, под створився, стартував, але перевірки не пройшов. Почекаємо трішки часу і зробимо власну перевірку

Команда:

```
curl http://10.1.6.126:3100/ready
```

Відгук:

```
ready
```

Бачимо шо все з локі добре, він відгукається і працює добре, бо я вже й через графана це перевірив)). Тож треба було деплойменту дати троху часу шоб піднятися

Тепер приступимо до деплойменту Prometheus

Створимо для Prometheus ConfigMap

Команда:

```
nano prom-config.yaml
```

Відгук:

...

Заповнюємо файл наступним вмістом:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-config
```

```
  namespace: monitoring
data:
  prometheus.yml: |
    global:
      scrape_interval: 15s
    scrape_configs:
      - job_name: 'kubernetes'
        static_configs:
          - targets: ['localhost:9100']
```

Зберігаємо і застосовуємо цей файл

Команда:

```
sudo microk8s.kubectl apply -f prom-config.yaml
```

Відгук:

```
configmap/prometheus-config created
```

Створюємо файл деплойменту

Команда:

```
nano prom-deployment.yaml
```

Відгук:

...

Заповнюємо файл цім вмістом:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: prometheus
  namespace: monitoring
spec:
  replicas: 1
  selector:
    matchLabels:
      app: prometheus
  template:
    metadata:
      labels:
        app: prometheus
    spec:
      containers:
        - name: prometheus
          image: prom/prometheus
          args:
            - "--config.file=/etc/prometheus/prometheus.yml"
          ports:
            - containerPort: 9090
          volumeMounts:
            - name: config-volume
              mountPath: /etc/prometheus/
      volumes:
```

```
        - name: config-volume
          configMap:
            name: prometheus-config
```

Зберігаємо і застосовуємо

Команда:

```
sudo microk8s.kubectl apply -f prom-deployment.yaml
```

Відгук:

```
deployment.apps/prometheus created
```

Перевіряймо

Команда:

```
sudo microk8s.kubectl get pods -n monitoring
```

Відгук:

```
NAME                          READY   STATUS    RESTARTS   AGE
fluentbit-rpp4t               1/1     Running   0          5h27m
grafana-85b785d45d-5t4xl      1/1     Running   0          5h16m
loki-69b5f68c75-fpccg         1/1     Running   0          66m
prometheus-56d98cf486-5tjc4   1/1     Running   0          65s
```

Все ок, под створено і він працює

Створюємо для нього сервіс

Команда:

```
nano prom-service.yaml
```

Відгук:

```
...
```

Заповнюємо файл наступним вмістом

```
apiVersion: v1
kind: Service
metadata:
  name: prometheus
  namespace: monitoring
spec:
  selector:
    app: prometheus
  ports:
    - protocol: TCP
      port: 9090
      targetPort: 9090
```

Зберігаємо і застосовуємо цей файл

Команда:

```
sudo microk8s.kubectl apply -f prom-service.yaml
```

Відгук:

```
service/prometheus created
```

Перевіряємо:

Команда:
sudo microk8s.kubectl get svc -n monitoring
Відгук:
```
NAME         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
grafana      ClusterIP   10.152.183.173   <none>        3000/TCP   3h51m
loki         ClusterIP   10.152.183.62    <none>        3100/TCP   2d4h
prometheus   ClusterIP   10.152.183.82    <none>        9090/TCP   42s
```

Бачимо шо сервіс створено і він працює

3. Збирання логів та метрик до grafana

Тепер налаштовуємо метрики
Створюємо для метрик сервіс-акаунт, роль для кластера, прив’язку ролей кластера, деплоймент та сервіс. І все це одним yaml файлом

Команда:
nano kube-state-metrics.yaml
Відгук:
...

Заповнюємо файл цім вмістом:

```
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: kube-state-metrics
  namespace: monitoring

---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: kube-state-metrics
rules:
  - apiGroups: [""]
    resources:
      - pods
      - nodes
      - namespaces
      - persistentvolumeclaims
      - persistentvolumes
      - services
    verbs: ["list", "watch"]
  - apiGroups: ["apps"]
    resources:
      - deployments
      - daemonsets
```

```
      - replicasets
      - statefulsets
    verbs: ["list", "watch"]
  - apiGroups: ["batch"]
    resources:
      - jobs
      - cronjobs
    verbs: ["list", "watch"]

---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: kube-state-metrics
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: kube-state-metrics
subjects:
  - kind: ServiceAccount
    name: kube-state-metrics
    namespace: monitoring

---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kube-state-metrics
  namespace: monitoring
spec:
  replicas: 1
  selector:
    matchLabels:
      app: kube-state-metrics
  template:
    metadata:
      labels:
        app: kube-state-metrics
    spec:
      serviceAccountName: kube-state-metrics
      containers:
        - name: kube-state-metrics
          image: registry.k8s.io/kube-state-metrics/kube-state-metrics:v2.10.1
          ports:
            - name: http
              containerPort: 8080

---
apiVersion: v1
kind: Service
metadata:
  name: kube-state-metrics
```

```
  namespace: monitoring
  labels:
    app: kube-state-metrics
spec:
  selector:
    app: kube-state-metrics
  ports:
    - name: http
      port: 8080
      targetPort: http
```

Зберігаємо і застосовуємо цей файл

Команда:

```
sudo microk8s.kubectl apply -f kube-state-metrics.yaml
```

Відгук:

```
serviceaccount/kube-state-metrics created
clusterrole.rbac.authorization.k8s.io/kube-state-metrics created
clusterrolebinding.rbac.authorization.k8s.io/kube-state-metrics created
deployment.apps/kube-state-metrics created
service/kube-state-metrics created
```

Перевіряймо

Команда:

```
sudo microk8s.kubectl get serviceaccount -n monitoring
```

Відгук:

```
NAME              SECRETS   AGE
default           0         2d6h
kube-state-metrics   0         10m
```

Команда:

```
sudo microk8s.kubectl get clusterrole
```

Відгук:

```
NAME                                                   CREATED AT
calico-cni-plugin                                      2025-02-02T15:58:03Z
calico-kube-controllers                                2025-02-02T15:58:03Z
calico-node                                            2025-02-02T15:58:03Z
cert-manager-cainjector                                2025-02-03T08:32:54Z
cert-manager-cluster-view                              2025-02-03T08:32:54Z
cert-manager-controller-approve:cert-manager-io        2025-02-03T08:32:54Z
cert-manager-controller-certificates                   2025-02-03T08:32:54Z
cert-manager-controller-certificatesigningrequests     2025-02-03T08:32:54Z
cert-manager-controller-challenges                     2025-02-03T08:32:54Z
cert-manager-controller-clusterissuers                 2025-02-03T08:32:54Z
cert-manager-controller-ingress-shim                   2025-02-03T08:32:54Z
cert-manager-controller-issuers                        2025-02-03T08:32:54Z
cert-manager-controller-orders                         2025-02-03T08:32:54Z
cert-manager-edit                                      2025-02-03T08:32:54Z
cert-manager-view                                      2025-02-03T08:32:54Z
cert-manager-webhook:subjectaccessreviews              2025-02-03T08:32:54Z
coredns                                                2025-02-02T15:58:05Z
```

kube-state-metrics 2025-03-12T18:40:49Z
kubernetes-dashboard 2025-02-03T06:34:26Z
microk8s-hostpath 2025-02-02T16:01:44Z
nginx-ingress-microk8s-clusterrole 2025-02-02T16:04:59Z
system:aggregated-metrics-reader 2025-02-03T06:34:25Z
system:metrics-server 2025-02-03T06:34:25Z

Команда:

sudo microk8s.kubectl get clusterrolebinding

Відгук:

NAME ROLE
AGE
calico-cni-plugin ClusterRole/calico-cni-plugin
38d
calico-kube-controllers ClusterRole/calico-kube-controllers
38d
calico-node ClusterRole/calico-node
38d
cert-manager-cainjector ClusterRole/cert-manager-cainjector
37d
cert-manager-controller-approve:cert-manager-io ClusterRole/cert-manager-
controller-approve:cert-manager-io 37d
cert-manager-controller-certificates ClusterRole/cert-manager-
controller-certificates 37d
cert-manager-controller-certificatesigningrequests ClusterRole/cert-manager-
controller-certificatesigningrequests 37d
cert-manager-controller-challenges ClusterRole/cert-manager-
controller-challenges 37d
cert-manager-controller-clusterissuers ClusterRole/cert-manager-
controller-clusterissuers 37d
cert-manager-controller-ingress-shim ClusterRole/cert-manager-
controller-ingress-shim 37d
cert-manager-controller-issuers ClusterRole/cert-manager-
controller-issuers 37d
cert-manager-controller-orders ClusterRole/cert-manager-
controller-orders 37d
cert-manager-webhook:subjectaccessreviews ClusterRole/cert-manager-
webhook:subjectaccessreviews 37d
coredns ClusterRole/coredns
38d
kube-state-metrics ClusterRole/kube-state-metrics
13m
kubernetes-dashboard ClusterRole/kubernetes-dashboard
37d
metrics-server:system:auth-delegator ClusterRole/system:auth-
delegator 37d
microk8s-admin ClusterRole/admin
37d
microk8s-hostpath ClusterRole/microk8s-hostpath
38d
nginx-ingress-microk8s ClusterRole/nginx-ingress-microk8s-
clusterrole 38d

system:metrics-server ClusterRole/system:metrics-server 37d

Команда:

sudo microk8s.kubectl get pods -n monitoring

Відгук:

```
NAME                                  READY   STATUS              RESTARTS   AGE
fluentbit-rpp4t                       1/1     Running             0          6h7m
grafana-85b785d45d-5t4xl              1/1     Running             0          5h56m
kube-state-metrics-669cccd79c-tjnsp   0/1     ContainerCreating   0          15m
loki-69b5f68c75-fpccg                 1/1     Running             0          106m
prometheus-56d98cf486-5tjc4           1/1     Running             0          40m
```

Команда:

sudo microk8s.kubectl get svc -n monitoring

Відгук:

```
NAME                 TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
grafana              ClusterIP   10.152.183.173   <none>        3000/TCP   4h26m
kube-state-metrics   ClusterIP   10.152.183.39    <none>        8080/TCP   16m
loki                 ClusterIP   10.152.183.62    <none>        3100/TCP   2d5h
prometheus           ClusterIP   10.152.183.82    <none>        9090/TCP   35m
```

Бачимо шо все створилося і працює

Тепер створимо DaemonSet та сервіс для нього

Команда:

nano node-exporter.yaml

Відгук:

...

Заповнюємо файл наступним вмістом:

```
---
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: node-exporter
  namespace: monitoring
  labels:
    app: node-exporter
spec:
  selector:
    matchLabels:
      app: node-exporter
  template:
    metadata:
      labels:
        app: node-exporter
    spec:
      hostPID: true
      hostNetwork: true
```

```
      containers:
        - name: node-exporter
          image: quay.io/prometheus/node-exporter:v1.7.0
          ports:
            - name: metrics
              containerPort: 9100
          volumeMounts:
            - name: proc
              mountPath: /host/proc
              readOnly: true
            - name: sys
              mountPath: /host/sys
              readOnly: true
            - name: root
              mountPath: /rootfs
              readOnly: true
          args:
            - '--path.procfs=/host/proc'
            - '--path.sysfs=/host/sys'
            - '--path.rootfs=/rootfs'
      volumes:
        - name: proc
          hostPath:
            path: /proc
        - name: sys
          hostPath:
            path: /sys
        - name: root
          hostPath:
            path: /

---
apiVersion: v1
kind: Service
metadata:
  name: node-exporter
  namespace: monitoring
  labels:
    app: node-exporter
spec:
  selector:
    app: node-exporter
  ports:
    - name: metrics
      port: 9100
      targetPort: 9100
```

Зберігаємо і застосовуємо

Команда:

```
sudo microk8s.kubectl apply -f node-exporter.yaml
```

Відгук:

daemonset.apps/node-exporter created
service/node-exporter created

Перевіймо

Команда:
sudo microk8s.kubectl get daemonset -n monitoring
Відгук:
```
NAME          DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE
fluentbit     1         1         1       1            1           <none>          6h19m
node-exporter 1         1         1       1            1           <none>          6m57s
```

Команда:
sudo microk8s.kubectl get svc -n monitoring
Відгук:
```
NAME               TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
grafana            ClusterIP   10.152.183.173   <none>        3000/TCP   4h37m
kube-state-metrics ClusterIP   10.152.183.39    <none>        8080/TCP   27m
loki               ClusterIP   10.152.183.62    <none>        3100/TCP   2d5h
node-exporter      ClusterIP   10.152.183.70    <none>        9100/TCP   7m36s
prometheus         ClusterIP   10.152.183.82    <none>        9090/TCP   46m
```

Все створилося і працює добре

Оновимо ConfigMap Prometheus

Команда:
nano prom-config.yaml
Відгук:
...

Заповнимо файл наступним вмістом:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-config
  namespace: monitoring
data:
  prometheus.yml: |
    global:
      scrape_interval: 15s
    scrape_configs:
      - job_name: 'node-exporter'
        static_configs:
          - targets: ['node-exporter.monitoring.svc.cluster.local:9100']

      - job_name: 'kube-state-metrics'
        static_configs:
          - targets: ['kube-state-metrics.monitoring.svc.cluster.local:8080']
```

```
    - job_name: 'kubernetes-nodes'
      kubernetes_sd_configs:
        - role: node
      relabel_configs:
        - action: labelmap
          regex: __meta_kubernetes_node_label_(.+)

    - job_name: 'kubelet'
      metrics_path: /metrics
      scheme: https
      kubernetes_sd_configs:
        - role: node
      tls_config:
        insecure_skip_verify: true
      bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
      relabel_configs:
        - action: labelmap
          regex: __meta_kubernetes_node_label_(.+)

    - job_name: 'kubernetes-pods'
      kubernetes_sd_configs:
        - role: pod
      relabel_configs:
        - source_labels:
[__meta_kubernetes_pod_annotation_prometheus_io_scrape]
          action: keep
          regex: true
        - source_labels:
[__meta_kubernetes_pod_annotation_prometheus_io_path]
          action: replace
          target_label: __metrics_path__
          regex: (.+)
        - source_labels: [__address__,
__meta_kubernetes_pod_annotation_prometheus_io_port]
          action: replace
          regex: (.+):(?:\d+);(\d+)
          replacement: $1:$2
          target_label: __address__
```

Зберігаємо і застосовуємо цей файл

Команда:

```
sudo microk8s.kubectl apply -f prom-config.yaml
```

Відгук:

```
configmap/prometheus-config configured
```

Після цього прийшлося ребутнути VM бо позависали поди - kube-state-metrics та prometheus

Після ребута

Команда:

sudo microk8s.kubectl port-forward svc/prometheus -n monitoring 9090:9090

Відгук:

Forwarding from 127.0.0.1:9090 -> 9090
Forwarding from [::1]:9090 -> 9090
Перевірмо node-exporter

Команда:

curl http://10.0.2.15:9100/metrics

Відгук:

node_network_iface_id{device="calif96f54ece13"} 18
node_network_iface_id{device="califd7994c7e68"} 7
node_network_iface_id{device="docker0"} 3
node_network_iface_id{device="enp0s3"} 2
node_network_iface_id{device="lo"} 1
node_network_iface_id{device="vxlan.calico"} 8
HELP node_network_iface_link Network device property: iface_link
TYPE node_network_iface_link gauge
node_network_iface_link{device="cali06417df87b0"} 2
node_network_iface_link{device="cali0761c7440db"} 2
node_network_iface_link{device="cali181349b82a9"} 2
node_network_iface_link{device="cali1f427c7a51e"} 2

Дуже велике простирадло метрик, все працює добре

Тепер перевірмо kube-state-metrics

Команда:

curl http://10.1.6.117:8080/metrics

Відгук:

s-56d98cf486",owner_is_controller="true"} 1
kube_pod_owner{namespace="nginx",pod="nginx-php-96697cbc-hf248",uid="dd2d2f04-e718-4aac-a25f-b6f1d8faa46f",owner_kind="ReplicaSet",owner_name="nginx-php-96697cbc",owner_is_controller="true"} 1
kube_pod_owner{namespace="cert-manager",pod="cert-manager-webhook-7749797f6-qskj9",uid="30b5b7d4-3b3c-4e8b-987f-b2546cbc98a2",owner_kind="ReplicaSet",owner_name="cert-manager-webhook-7749797f6",owner_is_controller="true"} 1
kube_pod_owner{namespace="kube-system",pod="dashboard-metrics-scraper-5bd45c9dd6-9495r",uid="bcb9a77b-c727-46a0-9a45-3e654603a313",owner_kind="ReplicaSet",owner_name="dashboard-metrics-scraper-5bd45c9dd6",owner_is_controller="true"} 1
kube_pod_owner{namespace="kube-system",pod="hostpath-provisioner-c778b7559-khqrk",uid="8e525dc8-355b-4437-8209-2063c9b97cda",owner_kind="ReplicaSet",owner_name="hostpath-provisioner-c778b7559",owner_is_controller="true"} 1
kube_pod_owner{namespace="kube-system",pod="metrics-server-7dbd8b5cc9-mt9xq",uid="499217d3-feb4-4b05-a925-27cb3162d9d6",owner_kind="ReplicaSet",owner_name="metrics-server-7dbd8b5cc9",owner_is_controller="true"} 1

kube_pod_owner{namespace="cert-manager",pod="cert-manager-cainjector-fd9bf654b-7cncr",uid="4a4db035-5b57-4e70-8ecc-d5002f5a522b",owner_kind="ReplicaSet",owner_name="cert-manager-cainjector-fd9bf654b",owner_is_controller="true"} 1
kube_pod_owner{namespace="monitoring",pod="node-exporter-jhf9m",uid="93c0b36f-c3ae-4dc6-82f5-f4e776eb713b",owner_kind="DaemonSet",owner_name="node-exporter",owner_is_controller="true"} 1
kube_pod_owner{namespace="monitoring",pod="loki-69b5f68c75-fpccg",uid="d3ae003b-f4aa-4c9f-99dc-e16e565ebe7c",owner_kind="ReplicaSet",owner_name="loki-69b5f68c75",owner_is_controller="true"} 1
HELP kube_pod_restart_policy [STABLE] Describes the restart policy in use by this pod.
TYPE kube_pod_restart_policy gauge
kube_pod_restart_policy{namespace="monitoring",pod="loki-69b5f68c75-fpccg",uid="d3ae003b-f4aa-4c9f-99dc-e16e565ebe7c",type="Always"} 1
kube_pod_restart_policy{namespace="kube-system",pod="coredns-656767cfc8-xxkkg",uid="78ace4ed-dfec-494b-abc9-142ba6fb7117",type="Always"} 1
kube_pod_restart_policy{namespace="kube-system",pod="kubernetes-dashboard-977cd9cbc-wc4fl",uid="c57b1f8f-8448-478a-a894-643f2e465b78",type="Always"} 1
kube_pod_restart_policy{namespace="monitoring",pod="fluentbit-rpp4t",uid="4b429ea9-edf2-4080-b146-44b573ebbfa4",type="Always"} 1
kube_pod_restart_policy{namespace="monitoring",pod="grafana-85b785d45d-5t4xl",uid="06f22f57-1ef0-462f-8c7e-cc25692ccd4e",type="Always"} 1
kube_pod_restart_policy{namespace="cert-manager",pod="cert-manager-ff4b94468-8cfxx",uid="d8fcd5c4-4d7e-441c-a02c-958857e6b335",type="Always"} 1
kube_pod_restart_policy{namespace="ingress",pod="nginx-ingress-microk8s-controller-kdwbk",uid="3323b0ff-dab7-4caa-ae1b-22902f065803",type="Always"} 1
kube_pod_restart_policy{namespace="kube-system",pod="calico-kube-controllers-5947598c79-qhtq9",uid="cc2f789b-83f1-429f-8f7d-2d84d35da0e6",type="Always"} 1
kube_pod_restart_policy{namespace="kube-system",pod="calico-node-gf44k",uid="42331272-08c2-4aa0-8913-b498c32ebcf7",type="Always"} 1
kube_pod_restart_policy{namespace="monitoring",pod="kube-state-metrics-669cccd79c-tjnsp",uid="f35498e6-091d-400e-b0d8-21ddf2e22420",type="Always"} 1
kube_pod_restart_policy{namespace="monitoring",pod="prometheus-56d98cf486-7wcph",uid="1c8bfedf-6d3c-4492-9ebd-b9572681d1f2",type="Always"} 1

Теж чи мале простирадло метрик, тож і тут все працює

Команда:

curl http://localhost:9100/metrics

Відгук:

node_network_iface_id{device="calif96f54ece13"} 18

node_network_iface_id{device="califd7994c7e68"} 7
node_network_iface_id{device="docker0"} 3
node_network_iface_id{device="enp0s3"} 2
node_network_iface_id{device="lo"} 1
node_network_iface_id{device="vxlan.calico"} 8
HELP node_network_iface_link Network device property: iface_link
TYPE node_network_iface_link gauge
node_network_iface_link{device="cali06417df87b0"} 2
node_network_iface_link{device="cali0761c7440db"} 2
node_network_iface_link{device="cali181349b82a9"} 2
node_network_iface_link{device="cali1f427c7a51e"} 2
node_network_iface_link{device="cali207e534a67a"} 2
node_network_iface_link{device="cali34279ac7e42"} 2
node_network_iface_link{device="cali3442986506b"} 2
node_network_iface_link{device="cali5a5071bf8a7"} 2
node_network_iface_link{device="cali5dccc245ac0"} 2
node_network_iface_link{device="cali7d16972341b"} 2
node_network_iface_link{device="cali84142cffaea"} 2
node_network_iface_link{device="cali8515ac45b50"} 2
node_network_iface_link{device="calid781513baca"} 2
node_network_iface_link{device="calif78bdac7a2c"} 2
node_network_iface_link{device="calif96f54ece13"} 2
node_network_iface_link{device="califd7994c7e68"} 2
node_network_iface_link{device="docker0"} 3
node_network_iface_link{device="enp0s3"} 2
node_network_iface_link{device="lo"} 1

І тут все працює

Також Grafana і Prometheus та Prometheus Node Exporter
доступні через браузер по таких ip адресах:
http://grafana.edu.local/
http://10.1.6.89:9090/targets
http://localhost:9100/
Вони працездатні і з ними можна працювати

Домашнє завдання до уроку 11:
Написати маніфести для створення деплойменту пода, в якому:

1. працює контейнер з busybox
2. кожні 5 секунд він одночасно пише на STDOUT і у файл на файловій системі hostname контейнера й дату
3. файл повинен лежати у Persistent Volume і бути доступним після перезапуску пода
4. ім’я файлу передається у под за допомогою ConfigMap
5. у пода повинно бути 3 репліки, під час експериментів треба проскейлити под у більшу й меншу сторону та переконатися, шо всі поди пишуть у потрібний файл
6. налаштувати HPA для деплоймента
7. додати у кластер prometheus, fluentd, loki, grafana
8. зібрати логи та метрики до grafana

Студент: Олександр Болотов

Дата виконання завдання: 13.03.2025