Sprawozdanie 1

Dawid Grabek

Wstęp

Całość zadania ma być zrealizowana w przestrzeni nazw (namespace) o nazwie zad1. Na wstępie należy utworzyć manifest (plik yaml) deklarujący przestrzeń nazw zad1. Następnie uruchomić ten obiekt (tą przestrzeń nazw). Następnie należy utworzyć zestaw plików manifestów (plików yaml) opisujących obiekty środowiska Kubernetes zgodnie z poniższymi założeniami:

```
/d/Studia/Semestr_9/chmury
$ cat zad1.yaml
apiVersion: v1
kind: Namespace
metadata:
  name: zad1


/d/Studia/Semestr_9/chmury
$ kubectl apply -f zad1.yaml
namespace/zad1 created

/d/Studia/Semestr_9/chmury
$ kubectl get namespaces
NAME                   STATUS   AGE
default                Active   34d
kube-node-lease        Active   34d
kube-public            Active   34d
kube-system            Active   34d
kubernetes-dashboard   Active   34d
lab4                   Active   4d14h
myworld                Active   32d
production             Active   32d
restricted             Active   18d
zad1                   Active   12s
```

*Rysunek 1. Tworzenie przestrzeni nazw **zad1** do sprawozdania*

Zadanie 1

Utworzyć plik yaml tworzący dla przestrzeni nazw zad1 zestaw ograniczeń na zasoby (quota) o następujących parametrach:
- maksymalna liczba Pod-ów: 10
- dostępne zasoby CPU: 2 CPU (2000m)
- dostępna ilość pamięci RAM: 1,5Gi

```
/d/Studia/Semestr_9/chmury
$ cat resource-quota.yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: zad1-quota
  namespace: zad1
spec:
  hard:
    pods: "10"
    requests.cpu: "2000m"
    requests.memory: "1.5Gi"
    limits.cpu: "2000m"
    limits.memory: "1.5Gi"


/d/Studia/Semestr_9/chmury
$ kubectl apply -f resource-quota.yaml
resourcequota/zad1-quota created
```

Rysunek 2 Stworzenie ograniczeń zasobów

```
/d/Studia/Semestr_9/chmury
$ kubectl get resourcequota zad1-quota -n zad1
NAME         AGE   REQUEST                                                 LIMIT
zad1-quota   19s   pods: 0/10, requests.cpu: 0/2, requests.memory: 0/1536Mi   limits.cpu: 0/2, limits
.memory: 0/1536Mi
```

Rysunek 3 Weryfikacja zadania

Zadanie 2

Utworzyć plik yaml tworzący Pod-a w przestrzeni nazw zad1 o nazwie worker. Pod ma bazować na obrazie nginx i mieć następujące ograniczenia na wykorzystywane zasoby:
limits:
memory: 200Mi
cpu: 200m
requests:
memory: 100Mi
cpu: 100m

```
/d/Studia/Semestr_9/chmury
$ cat pod-worker.yaml
apiVersion: v1
kind: Pod
metadata:
  name: worker
  namespace: zad1
spec:
  containers:
  - name: nginx
    image: nginx
    resources:
      limits:
        memory: "200Mi"
        cpu: "200m"
      requests:
        memory: "100Mi"
        cpu: "100m"


/d/Studia/Semestr_9/chmury
$ kubectl apply -f pod-worker.yaml
pod/worker created
```

Rysunek 4 Stworzenie pod'a worker

```
/d/Studia/Semestr_9/chmury
$ kubectl get pods -n zad1
NAME     READY   STATUS    RESTARTS   AGE
worker   1/1     Running   0          67s
```

Rysunek 5 Weryfikacja zadania

Zadanie 3

Bazując na przykładzie application/php-apache.yaml i/lub z dokumentacji Kubernetes: https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscalewalkthrough/ należy zmodyfikować wskazany wyżej plik yaml tak by obiekty Deployment i Service utworzone zostały w przestrzeni nazw zad1. Jednocześnie obiekt Deployment ma mieć następujące ograniczenia na wykorzystywane zasoby:

limits:
  memory: 250Mi
  cpu: 250m
requests:
  memory: 150Mi
  cpu: 150m

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: php-apache
  namespace: zad1
spec:
  replicas: 1
  selector:
    matchLabels:
      app: php-apache
  template:
    metadata:
      labels:
        app: php-apache
    spec:
      containers:
      - name: php-apache
        image: registry.k8s.io/hpa-example
        ports:
        - containerPort: 80
        resources:
          limits:
            memory: "250Mi"
            cpu: "250m"
          requests:
            memory: "150Mi"
            cpu: "150m"
```

Rysunek 6 deployment.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: php-apache
  labels:
    run: php-apache
spec:
  ports:
  - port: 80
  selector:
    run: php-apache
```

Rysunek 7 service.yaml

Zadanie 4

Należy utworzyć plik yaml definiujący obiekt HorizontalPodAutoscaler, który pozwoli na utoskalowanie wdrożenia (Deployment) php-apache z zastosowaniem następujących
minReplicas: 1
maxReplicas: ???????
targetCPUUtilizationPercentage: 50
Wartość maxReplicas należy określić samodzielnie, tak by nie przekroczyć parametrów quoty dla przestrzeni nazw zad5. W sprawozdaniu należy UZASADNIĆ PRZYJĘTY DOBÓR WARTOŚCI.

```yaml
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: php-apache-hpa
  namespace: zad1
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: php-apache
  minReplicas: 1
  maxReplicas: 5
  targetCPUUtilizationPercentage: 50
```

Rysunek 8 hpa.yaml

```
/d/Studia/Semestr_9/chmury
$ kubectl apply -f hpa.yaml
horizontalpodautoscaler.autoscaling/php-apache-hpa created
```

Rysunek 9 Uruchomienie hpa.yaml

```
/d/Studia/Semestr_9/chmury
$ kubectl get hpa -n zad1
NAME             REFERENCE                TARGETS       MINPODS   MAXPODS   REPLICAS   AGE
php-apache-hpa   Deployment/php-apache   cpu: 0%/50%   1         5         1          35s
```

Rysunek 10 Weryfikacja zadania

Uzasadnienie:

W zadaniu 4 ustaliliłem wartość maxReplicas na 5. Uzasadnienie wynika z ograniczeń nałożonych przez Resource Quota w przestrzeni nazw zad1, które definiują

- Maksymalna liczba podów: **10**
- Dostępne zasoby CPU: **2000m (2 CPU)**
- Dostępna ilość pamięci RAM: **1.5Gi**

Obliczenia:

1. Każdy pod w Deployment ma przypisane:
    a. CPU
        i. limits.cpu = 250m
        ii. requests.cpu = 150m
    b. RAM:
        i. limits.memory = 250Mi
        ii. requests.memory = 150Mi.
2. Zasoby wymagane dla 1 poda:
    a. CPU = 250m
    b. RAM = 250Mi
3. Maksymalna liczba podów ograniczona zasobami:
    a. CPU: *2000m / 250m=8*
    b. RAM: *1536Mi / 250Mi=6.144*
4. Najmniejszy wynik (CPU vs RAM) decyduje: maksymalnie **6 podów**.
5. Biorąc pod uwagę, że w namespace może znajdować się również pod worker, który zajmuje:
    a. CPU: 200m
    b. RAM: 200Mi
6. Ograniczenia zasobów dla pozostałych podów:
    a. CPU: *2000m−200m=1800m*
    b. RAM: *1536Mi−200Mi=1336Mi*
7. Po podzieleniu zasobów na pod:
    a. CPU: *1800m / 250m=7.2*
    b. RAM: *1336Mi / 250Mi=5.344*

Finalna liczba replik: **5** (dla zachowania limitów RAM).

Zadanie 5

Należy utworzyć obiekty zdeklarowane w opracowanych plikach yaml. Następnie potwierdzić ich poprawne uruchomienie za pomocą samodzielnie dobranego polecenia (poleceń).

W poprzednich zadaniach utworzyłem już obiekty oraz wtedy sprawdziłem poprawność.

Zadanie 6

Ponownie, bazując na przykładach z instrukcji do lab5 i/lub linku podanego w punkcie 3, należy uruchomić aplikację generującą obciążenie dla aplikacji php-apache i tym samym inicjalizujące proces autoskalowania wdrożenia tej aplikacji. Za pomocą samodzielnie dobranych poleceń i wyniku ich działania proszę potwierdzić dobór parametrów z punktu 4.

```
grabekdawid@LAPTOP-AQ4LOQIO /mnt/d/Studia/Semestr_9/chmury
 % kubectl run -i --tty load-generator --rm --image=busybox:1.28 --restart=Never -- \
  /bin/sh -c "while sleep 0.01; do wget -q -O- http://php-apache-service.zad1.svc.cluster.local; done"
If you don't see a command prompt, try pressing enter.
OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!
```

Rysunek 11 Uruchomienie przeciążenia

```
grabekdawid@LAPTOP-AQ4LOQIO /mnt/c/WINDOWS/system32
 % kubectl get hpa -n zad1 --watch
NAME             REFERENCE                   TARGETS        MINPODS   MAXPODS   REPLICAS   AGE
php-apache-hpa   Deployment/php-apache   cpu: 0%/50%    1         5         1          40m
php-apache-hpa   Deployment/php-apache   cpu: 75%/50%   1         5         1          40m
php-apache-hpa   Deployment/php-apache   cpu: 75%/50%   1         5         2          40m
php-apache-hpa   Deployment/php-apache   cpu: 108%/50%  1         5         2          41m
php-apache-hpa   Deployment/php-apache   cpu: 91%/50%   1         5         2          42m
php-apache-hpa   Deployment/php-apache   cpu: 91%/50%   1         5         4          42m
php-apache-hpa   Deployment/php-apache   cpu: 57%/50%   1         5         4          43m
php-apache-hpa   Deployment/php-apache   cpu: 47%/50%   1         5         4          44m
php-apache-hpa   Deployment/php-apache   cpu: 48%/50%   1         5         4          45m
php-apache-hpa   Deployment/php-apache   cpu: 47%/50%   1         5         4          46m
php-apache-hpa   Deployment/php-apache   cpu: 48%/50%   1         5         4          49m
php-apache-hpa   Deployment/php-apache   cpu: 38%/50%   1         5         4          50m
```

Rysunek 12 Weryfikacja autoskalowania

Zgodnie z wynikami, HorizontalPodAutoscaler (HPA) działa poprawnie. Oto, co się dzieje:

Obciążenie CPU: Widać, że CPU w podach przekracza próg 50%, co skutkuje skalowaniem aplikacji.

Początkowo liczba replik to 1.

Gdy obciążenie CPU wzrastało do 75%, HPA zwiększyło liczbę replik do 2, a później do 4.

Skalowanie: HPA poprawnie reaguje na wzrost obciążenia CPU i skaluje liczbę podów, nie przekraczając ustawionego limitu maxReplicas (5).

Podsumowanie

- HPA działa poprawnie, a liczba replik dynamicznie rośnie w odpowiedzi na obciążenie CPU.
- Wszystko jest skonfigurowane zgodnie z oczekiwaniami, a mechanizm autoskalowania działa efektywnie.