infoShare ACADEMY

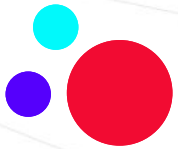
KUBERNETES – projekt

infoShare Academy

Piotr Ignatowski

infoShareAcademy.com

01. Małe podsumowanie

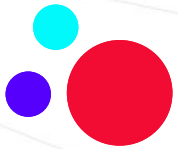


POD - grupa kontenerów na tej samej maszynie

Współdzieląca przede wszystkim sieciową przestrzeń nazw (widzą się po localhoście).

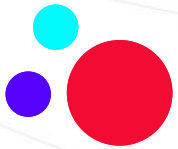
```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.14.2
    ports:
    - containerPort: 80
```

infoShare ACADEMY



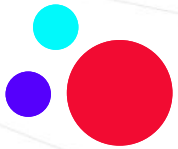
ReplicaSet - zarządza liczebnością podów stawianych ze wspólnego szablonu

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
  template:
    metadata:
      labels:
        tier: frontend
    spec:
      containers:
      - name: php-redis
        image: gcr.io/google_samples/gb-frontend:v3
```



Deployment - zarządza ReplicaSetami

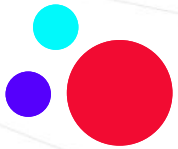
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```



StatefulSet - rozróżnialne instancje

https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/ - daje możliwość utworzenia replik, które zawsze wstają z tą samą nazwą, każda instancja może mieć swój własny Persistent Volume.

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
 name: web
spec:
 selector:
  matchLabels:
   app: nginx # has to match .spec.template.metadata.labels
 serviceName: "nginx"
 replicas: 3
 template:
  metadata:
   labels:
    app: nginx # has to match .spec.selector.matchLab
(...)
```

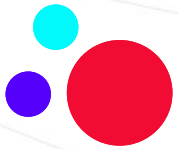


Sieć - ClusterIP service

Tworzy wpis w DNS w formacie <service_name>.<namespace>.<svc>.<cluster_domain>

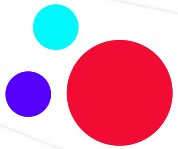
```
apiVersion: v1
kind: Service
metadata:
 name: my-service
spec:
 selector:
  app: MyApp
 ports:
  - protocol: TCP
    port: 80 # inbound port serwisu (w tym przypadku my-service:80)
    targetPort: 9376 # port w podach tworzących upstream
```

Taki serwis otrzyma IP sieci wewnątrzklastrowej. Stąd nazwa ClusterIP.



Sieć - service NodePort

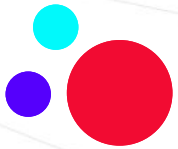
```
apiVersion: v1
kind: Service
metadata:
 name: my-service
spec:
 selector:
  app: MyApp
 ports:
  - protocol: TCP
    port: 80 # inbound port serwisu (w tym przypadku my-service:80)
    targetPort: 9376 # port w podach tworzących upstream
    nodePort: 30008 # port dostępny z “zewnątrz” na każdej maszynie mającej uruchomione kube-proxy
```



Sieć - service LoadBalancer

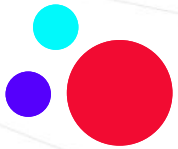
```
apiVersion: v1
kind: Service
metadata:
 name: my-service
spec:
 selector:
  app: MyApp
 ports:
  - protocol: TCP
    port: 80 # inbound port serwisu (w tym przypadku my-service:80)
    targetPort: 9376 # port w podach tworzących upstream
 type: LoadBalancer
```

Alokuje adres IP z sieci zewnętrznej - wymaga kontrolera obsługującego żądania o LoadBalancer



Sieć - headless service

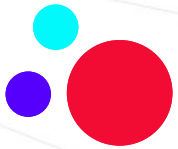
```
apiVersion: v1
kind: Service
metadata:
  name: headless-svc
spec:
  clusterIP: None
  selector:
    app: web
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
```



Volumes - emptyDir

Katalog współdzielony pomiędzy kontenerami w podzie.

```
apiVersion: v1
kind: Pod
metadata:
 name: test-pd
spec:
 containers:
 - image: k8s.gcr.io/test-webserver
   name: test-container
   volumeMounts:
   - mountPath: /cache
     name: cache-volume
 volumes:
 - name: cache-volume
   emptyDir: {}
```



Persystencja - Volumes/ConfigMap

```
apiVersion: v1
kind: Pod
metadata:
 name: configmap-pod
spec:
 containers:
  - name: test
    image: busybox:1.28
    volumeMounts:
     - name: config-vol
       mountPath: /etc/config
 volumes:
  - name: config-vol
    configMap:
     name: log-config
     items:
      - key: log_level
        path: log_level
```

Persystencja - Volumes/PersistentVolumes

PersistentVolumeClaim - prosi kontroler o przydzielnenie PV z określonej StorageClass.

W deploymencie każdy pod montuje to samo PV.

W StatefulSet każdy pod ma swoje PV.

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: foo-pvc

namespace: foo

spec:

storageClassName: "" *# Empty string must be explicitly set otherwise default StorageClass will be set*

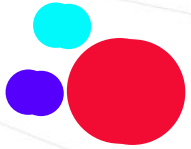
volumeName: foo-pv

Konfiguracja proxy - Ingress

Wymaga uruchomienia ingress controllera. Każdy controller ma przypisaną ingressClass.

```yaml
apiVersion: networking.k8s.io/v1
kind: IngressClass
metadata:
  ingressclass.kubernetes.io/is-default-class: "true"
  name: ngixn
spec:
  controller: k8s.io/ingress-nginx
```

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: simple-fanout-example
spec:
  ingressClassName: nginx # lub puste, jeśli nginx jest domyślną klasą ingresa
  rules:
  - host: foo.bar.com
    http:
      paths:
      - path: /foo
        pathType: Prefix
        backend:
          service:
            name: service1
            port:
              number: 4200
      - path: /bar
        pathType: Prefix
        backend:
          service:
            name: service2
            port:
              number: 8080
```



NetworkPolicy – firewall L3/L4

Bez względu na to jakiego pluginy sieciowego użyjemy (Calico, Cilium, Weave, CNI-Genie…), zawsze możemy skorzystać z API NetworkPolicy dającego nam separację sieciową naszych aplikacji.

Opisujemy reguły przychodzące (ingress) i wychodzące (egress).

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      role: db
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          project: myproject
    - podSelector:
        matchLabels:
          role: frontend
    ports:
    - protocol: TCP
      port: 6379
  egress:
  - to:
    - ipBlock:
        cidr: 10.0.0.0/24
    ports:
    - protocol: TCP
      port: 5978
```

02. Z czym się mierzymy?

Nadal wykorzystywać będziemy Vote App!

Przypominam link do projektu: https://github.com/dockersamples/example-voting-app/tree/master/k8s-specifications

Cele dzisiejszego projektu:

0) dzielimy aplikację na dwa namespace'y: datastores i apps

1) PostgreSQL jako StatefulSet

2) PersistentVolume dla bazy PostrgreSQL

3) POSTGRES_USER montowany jako env z configmapy (nawet, jeżeli pozostanie to wartość domyślna)

4) POSTGRES_PASSWORD montowane jako env z sekretu (jw)

5) PV dla Redisa

6) NetworkPolicy do komunikacji z Redisem i PostgreSQL

7) utworzenie certyfikatów dla dwóch użytkowników, developer i operator

8) utworzenie ról pozwalających developerowi na podgląd wszystkich obiektów, a operatorowi na wszystkie akcje (admin)