Kubernetes: Orkiestracja Kontenerów

Zarządzanie Systemami Rozproszonymi

mgr inż. Jakub Woźniak

Politechnika Poznańska
Wydział Informatyki i Telekomunikacji

Plan wykładu

Problem: Podejście imperatywne

Tradycyjny DevOps wymaga ręcznych poleceń na każdym hoście:

```
ssh user@host1 "docker run -d myapp:1.0"
ssh user@host2 "docker run -d myapp:1.0"
ssh user@host3 "docker run -d myapp:1.0"

# Jesli host1 padnie - reczny restart
ssh user@host1 "docker ps"
ssh user@host1 "docker start kontener"
```

Problemy:

- Operacje niepowtarzalne
- Nie skaluje się (100 hostów = 100 poleceń)
- Brak historii zmian
- Ręczny failover

Rozwiązanie: Kubernetes (deklaratywnie)

Opisujesz **pożądany stan** zamiast instrukcji krokowych:

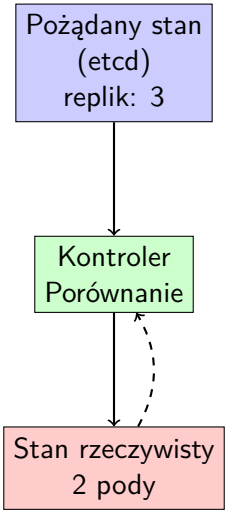
```
apiVersion: apps/v1
kind: Deployment
spec:
  replicas: 3
  template:
    spec:
      containers:
      - name: aplikacja
        image: myapp:1.0
```

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

Korzyści: Automatyczne skalowanie, naprawa awarii, aktualizacja i wycofanie

Pętla rekoncyliacji

Nieskończona pętla porównująca pożądany stan z rzeczywistym:



etcd - Baza danych klastra

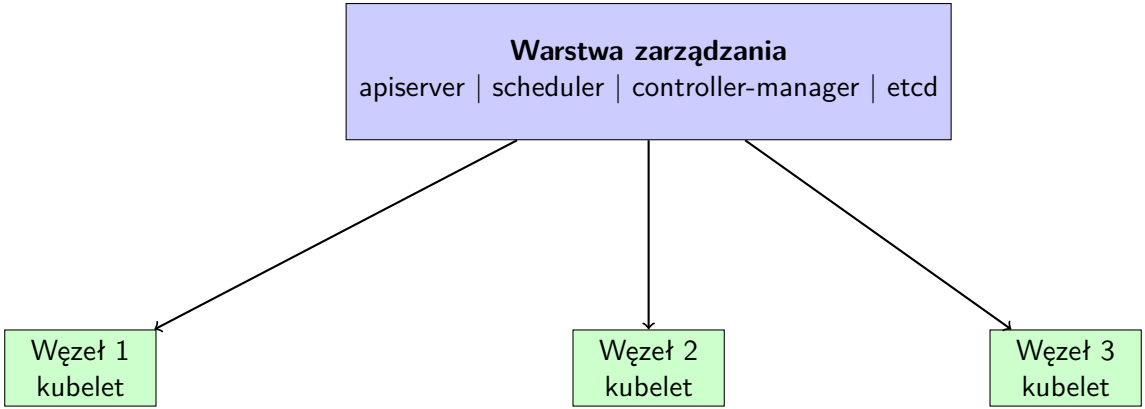
Rozproszone magazyny klucz-wartość przechowujące WSZYSTKIE obiekty

- Konsensus poprzez algorytm Raft
- 3 węzły: toleruje 1 awarię
- 5 węzłów: toleruje 2 awarie
- Leader replikuje zmiany do pomocników

Standardowy, ale nie wymagany

W przypadku Azure Kubernetes Service, etcd jest zastąpiony przez Azure CosmosDB.

Komponenty klastra



Warstwa zarządzania: Zarządza stanem, planuje rozmieszczenie
Węzły robocze: Uruchamiają pody

Główne komponenty

kube-apiserver

- Punkt wejściowy do klastra (REST API)
- Walidacja, autoryzacja (RBAC), zapis do etcd

kube-scheduler

- Przypisuje pody do węzłów na podstawie dostępnych zasobów

controller-manager

- Uruchamia kontrolery (Deployment, ReplicaSet, Service)
- Pętla rekonsyliacji dla każdego typu obiektu

kubelet

- Agent na węźle, uruchamia kontenery
- Obsługuje sondy (readiness/liveness)

Pod - Najmniejsza jednostka

Grupa kontenerów dzielących sieć i magazyn danych:

```
apiVersion: v1
kind: Pod
spec:
  containers:
  - name: aplikacja
    image: myapp:1.0
    resources:
      requests:
        memory: "256Mi"
        cpu: "100m"
      limits:
        memory: "512Mi"
        cpu: "500m"
```

requests = minimum zasobów dla harmonogramowania
limits = maksimum zasobów (jądro zatrzyma proces jeśli przekroczony)

Service - Stabilny punkt dostępu

Abstrakcja zapewniająca rozdzielanie obciążenia:

```
apiVersion: v1
kind: Service
spec:
  type: ClusterIP
  selector:
    app: siec
  ports:
  - port: 80
    targetPort: 8080
```

Typy:

- ClusterIP (dostęp wewnątrz klastra)
- NodePort (dostęp zewnętrzny przez port węzła)
- LoadBalancer (cloud LB)

Deployment - Zarządzanie replikami

```yaml
apiVersion: apps/v1
kind: Deployment
spec:
  replicas: 3
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
      maxSurge: 1
  template:
    spec:
      containers:
      - name: app
        image: myapp:1.5.0
```

RollingUpdate: Stopniowo zamienia stare pody nowymi
Historia: kubectl rollout history / undo

CNI - Container Network Interface

Wtyczka	Warstwa danych	Wydajność
Cilium	eBPF	95 Gbps
Calico	iptables	40 Gbps
Flannel	VXLAN	Niska

Test 2024: eBPF (Cilium) lepiej skaluje przy złożonych politykach sieciowych

Ingress - Routing warstwy HTTP

```
apiVersion: networking.k8s.io/v1
kind: Ingress
spec:
  rules:
  - host: example.com
    http:
      paths:
      - path: /api
        backend:
          service:
            name: api
            port:
              number: 8080
```

Kontrolery: NGINX, AWS ALB, Traefik, Istio Gateway

PersistentVolume i StorageClass

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: szybki-dysk
provisioner: ebs.csi.aws.com
parameters:
  type: io2
  iops: "1000"
---
apiVersion: v1
kind: PersistentVolumeClaim
spec:
  storageClassName: szybki-dysk
  resources:
    requests:
      storage: 50Gi
```

Dynamiczne tworzenie: Sterownik CSI automatycznie tworzy urządzenie

RollingUpdate

Stopniowo zamienia stare pody nowymi:

Czas	Stan
0	5x v1.0
1	4x v1.0, 1x v2.0
3	2x v1.0, 3x v2.0
5	0x v1.0, 5x v2.0

Zalety: Zero przerwania usług
Wady: Powolne przy dużych wdrożeniach

Blue-Green i Canary

Blue-Green:

- Dwa równoległe środowiska
- Przełączenie Service selector (natychmiastowe)
- Wada: podwójne zasoby

Canary:

- 5 procent ruchu do nowej wersji
- Monitorowanie wskaźników
- Stopniowo: 25 procent, 50 procent, 100 procent
- Wymaga Service Mesh (Istio/Linkerd)

Helm vs Kustomize

Aspekt	**Helm**	**Kustomize**
Zastosowanie	Pakiety	Środowiska
Szablonowanie	Szablony Go	Łączenie
Ekosystem	70k+ chartów	Mniejszy

Praktyka:

- Helm dla bibliotek (PostgreSQL, Redis)
- Kustomize dla aplikacji (test/przygotowanie/produkcja)

Główne pojęcia

1. Deklaracja zamiast instrukcji
2. Pętla rekonsyliacji (samonaplanianie)
3. etcd jako źródło prawdy
4. Architektura sterowana zdarzeniami
5. Zawsze używaj YAML (GitOps)

Dobre praktyki:

- Oddajmy zarządzanie komuś innemu ;)
- Limity zasobów i polityki sieciowe
- Kontenery bez uprawnień roota
- Monitorowanie warstwy zarządzania

Dziękuję za uwagę!

Pytania?

Jakub Woźniak
jakub.wozniak@cs.put.poznan.pl