Orkiestracja kontenerów przy użyciu Kubernetes

mgr inż. Jakub Woźniak

Zarządzanie Systemami Rozproszonymi
Instytut Informatyki
Politechnika Poznańska

Wprowadzenie do Kubernetes

- Kubernetes to platforma do zarządzania aplikacjami kontenerowymi w klastrze komputerów. Został zaprojektowany przez Google i oddany społeczności open-source w 2014 roku.
- Kubernetes zapewnia automatyzację, skalowanie i zarządzanie aplikacjami w kontenerach.
- Rozwiązuje problemy związane z zarządzaniem setkami lub tysiącami kontenerów na wielu serwerach.
- **Historia:**
  - Kubernetes wywodzi się z wewnętrznej platformy Google o nazwie Borg, która zarządzała kontenerami na potrzeby aplikacji internetowych Google.
  - Projekt został przekazany społeczności open-source pod nazwą Kubernetes, a jego celem była standaryzacja zarządzania kontenerami.

Podstawowe pojęcia w Kubernetes

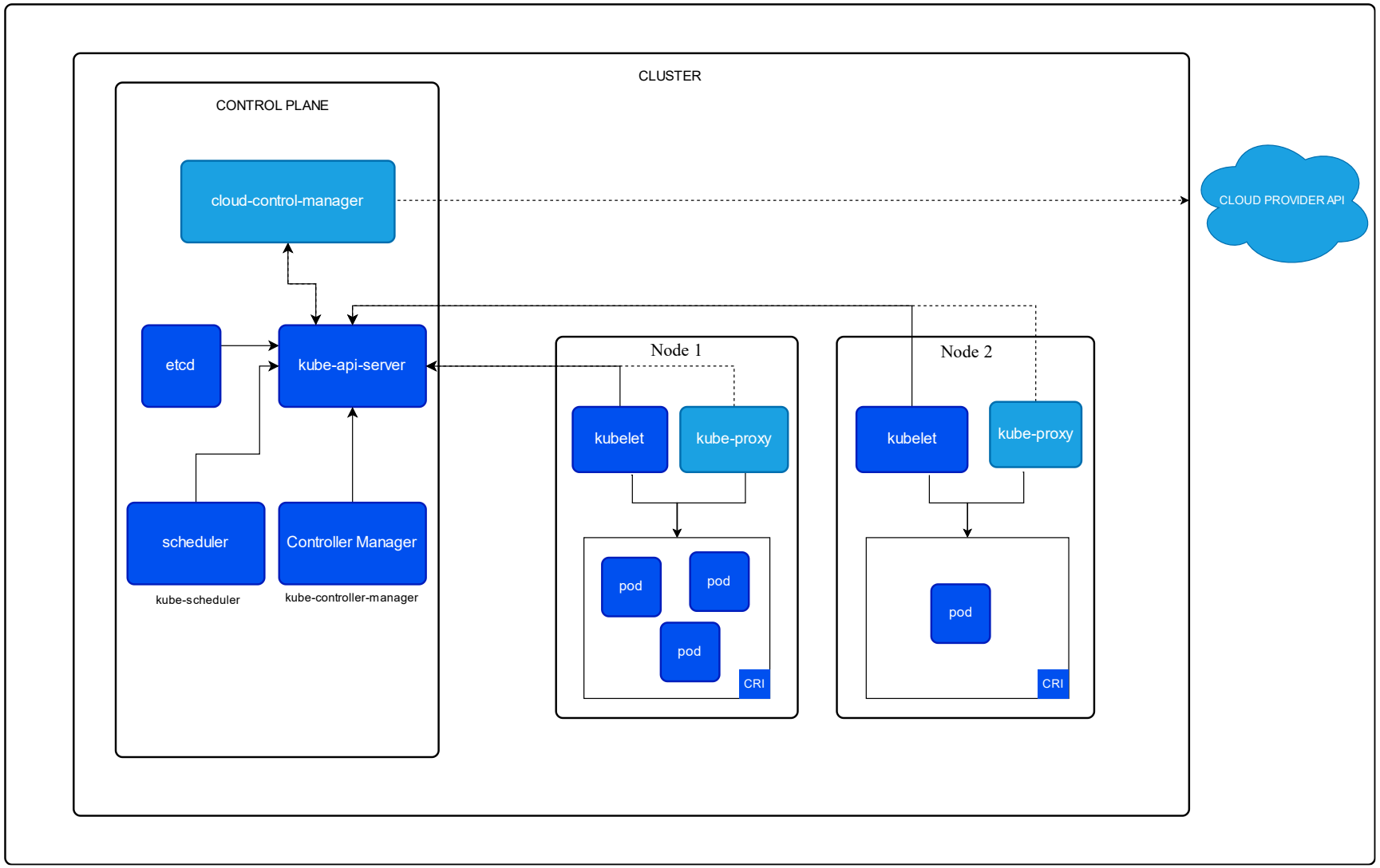
- **Klaster**: Zestaw maszyn (fizycznych lub wirtualnych), które działają razem jako jedna jednostka do uruchamiania aplikacji kontenerowych.
- **Węzeł**: Maszyna (fizyczna lub wirtualna) w klastrze Kubernetes. Każdy węzeł uruchamia przynajmniej jeden kontener.
- **Master Node**: Zarządza klastrem, kontroluje, które kontenery są uruchamiane na jakich węzłach.
- **Worker Node**: Węzeł wykonujący aplikacje. Otrzymuje polecenia od Master Node, aby uruchomić kontenery.
- **Pod**: Najmniejsza i najprostsza jednostka w Kubernetes. Zawiera jeden lub więcej kontenerów, które współdzielą zasoby, takie jak sieć i system plików.

Kubernetes vs. Docker Swarm

- **Kubernetes**:
  - Automatyzacja zarządzania kontenerami, replikacją, aktualizacjami, load balancingiem.
  - Posiada wsparcie dla zaawansowanych funkcji, takich jak autoskalowanie i rolling updates.
  - Popularność wynika z bogatego ekosystemu, wsparcia dużych firm (np. Google, Red Hat, AWS), i szerokiej adaptacji w świecie open-source.
- **Docker Swarm**:
  - Prostszy w konfiguracji, ale oferuje mniejszą elastyczność i funkcjonalność.
  - Działa dobrze dla mniejszych środowisk, ale ma problemy ze skalowaniem na większe produkcje.
- **Dlaczego Kubernetes?**
  - Lepsza obsługa dużych klastrów.
  - Rozbudowane możliwości automatyzacji i zarządzania.
  - Szerokie wsparcie od dostawców chmurowych i duża społeczność.

Architektura

- **Klaster Kubernetes** składa się z:
  - **Master Node**: Odpowiada za zarządzanie klastrem, planowanie zadań, monitorowanie stanu oraz podejmowanie decyzji dotyczących wdrażania aplikacji.
  - **Worker Nodes**: To maszyny (fizyczne lub wirtualne) uruchamiające aplikacje w kontenerach.
- **Komponenty Worker Node:**
  - **kubelet**: Agenta, który zarządza Podami uruchomionymi na Worker Node, komunikuje się z Master Node.
  - **kube-proxy**: Odpowiada za zarządzanie ruchem sieciowym do i z Podów w klastrze.
  - **Container Runtime**: Oprogramowanie, które obsługuje kontenery, np. Docker, containerd, czy CRI-O.
- **Komponenty Master Node:**
  - **kube-apiserver**: Centralny punkt komunikacji z klastrem. Odbiera żądania od użytkowników i innych komponentów Kubernetes, które korzystają z API.
  - **etcd**: Rozproszona baza danych, w której przechowywane są wszystkie informacje o stanie klastra.
  - **kube-scheduler**: Odpowiada za przydzielanie Podów do odpowiednich Worker Nodes na podstawie dostępnych zasobów.
  - **kube-controller-manager**: Uruchamia kontrolery odpowiedzialne za utrzymywanie stanu klastra (np. odtwarzanie Podów, skalowanie).
  - **cloud-controller-manager**: Integruje Kubernetes z platformą chmurową (w przypadku środowisk chmurowych).


CLUSTER
CONTROL PLANE
cloud-control-manager
CLOUD PROVIDER API
etcd
kube-api-server
Node 1
Node 2
kubelet
kube-proxy
kubelet
kube-proxy
scheduler
Controller Manager
kube-scheduler
kube-controller-manager
pod
pod
pod
pod
CRI
CRI

YAML – podstawa Infrastructure as Code

- W Kubernetes wszystkie zasoby są definiowane za pomocą plików YAML.
- YAML umożliwia zdefiniowanie infrastruktury jako kodu (Infrastructure as Code), co pozwala na wersjonowanie, kontrolę i automatyzację.
- Dlaczego YAML?
  - Umożliwia powtarzalność wdrożeń.
  - Łatwość edycji, wersjonowania i automatyzacji przy pomocy systemów CI/CD.

Pody – Najmniejsza jednostka pracy w Kubernetes

- **Pod** to najmniejsza jednostka Kubernetes, która uruchamia kontenery. Każdy Pod może zawierać jeden lub więcej kontenerów, które współdzielają sieć i woluminy.
- Pody są efemeryczne – mogą być usuwane i ponownie uruchamiane w zależności od stanu aplikacji i potrzeb klastra.
- Wykorzystanie Podów w praktyce – monitorowanie, usuwanie, restarty.

```
apiVersion: v1
kind: Pod
metadata:
  name: my-nginx-pod
  labels:
    app: nginx
spec:
  containers:
  - name: nginx-container
    image: nginx:latest
    ports:
    - containerPort: 80
  - name: sidecar-container
    image: busybox
    command: ['sh', '-c', 'while true; do echo hello; sleep 10;
done']
```

Deployment – Automatyzacja zarządzania Podami

- **Deployment** w Kubernetes automatyzuje proces wdrażania i zarządzania Podami.
- Główne funkcje Deploymentu:
  - Automatyczne odtwarzanie Podów w razie awarii.
  - Zapewnienie, że zawsze uruchomiona jest określona liczba Podów.
  - Umożliwienie bezprzerwowych aktualizacji (rolling updates).

```
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
```

Service – Stabilna komunikacja między Podami

- **Service** w Kubernetes umożliwia stabilną komunikację między Podami oraz z zewnętrznymi klientami.Typy Service:
  - **ClusterIP**: Umożliwia komunikację tylko wewnątrz klastra.
  - **NodePort**: Otwiera port na każdym węźle i udostępnia usługę poza klastrem.
  - **LoadBalancer**: Automatycznie tworzy Load Balancer, wystawiając aplikację na zewnątrz, często używany w chmurach.
- Service umożliwia dostęp do aplikacji nawet wtedy, gdy Pody są dynamicznie usuwane i tworzone.

PersistentVolume i PersistentVolumeClaim

- **PersistentVolume (PV)**: Fizyczny zasób do przechowywania danych w klastrze Kubernetes, np. przestrzeń dyskowa na serwerze.
- **PersistentVolumeClaim (PVC)**: Żądanie od aplikacji na określoną ilość przestrzeni dyskowej.
- Zastosowanie:
  - Przechowywanie trwałych danych, np. dla baz danych, upload plików, itp.

ConfigMap – Przechowywanie konfiguracji aplikacji

- **ConfigMap**: Umożliwia przechowywanie konfiguracji aplikacji w formacie klucz-wartość, np. ustawienia plików konfiguracyjnych.
- ConfigMap pozwala na dynamiczne dostosowywanie konfiguracji bez konieczności odbudowywania kontenerów.
- ConfigMap może być zamontowany jako wolumin lub jako zmienne środowiskowe.
- W przypadku przechowywania danych wrażliwych (klucze API, hasła) istnieje obiekt typu **Secret**, o podobnej strukturze, ale dedykowany przechowywaniu haseł i podobnych obiektów.

Horizontal Pod Autoscaler (HPA)

- **HPA**: Automatycznie dostosowuje liczbę replik Podów w zależności od obciążenia CPU.
- Działa w oparciu o metryki zużycia zasobów, np. CPU lub pamięci, i automatycznie skalibruje aplikację.
- HPA nie skaluje węzłów, jest to odpowiedzialność zazwyczaj dostawcy cloud i zewnętrznych programów.

Monitorowanie i obserwowalność

- Monitorowanie jest kluczowe w Kubernetes do śledzenia stanu aplikacji i zasobów.
- Narzędzia takie jak Kubernetes Dashboard lub Prometheus/Grafana mogą dostarczać informacje o stanie klastra.
- Przykład użycia kubectl do sprawdzenia stanu aplikacji:
  - kubectl logs – wyświetla logi z kontenera.
  - kubectl describe – szczegóły dotyczące zasobu.

Role Based Access Control

- **RBAC (Role-Based Access Control)**: Mechanizm zarządzania dostępem do zasobów Kubernetes, oparty na przypisaniu ról użytkownikom.
- **Role**: Definiuje uprawnienia dla zasobów w określonym zakresie (np. na poziomie namespace lub całego klastra).
- **RoleBinding**: Przypisuje rolę użytkownikowi lub grupie użytkowników.
- **ClusterRole**: Definiuje uprawnienia na poziomie całego klastra.
- **ClusterRoleBinding**: Przypisuje ClusterRole użytkownikowi lub grupie użytkowników.

Kubernetes produkcyjnie

- **Kubernetes w chmurze**:
  - Usługi Kubernetes dostępne w chmurze (GKE – Google Kubernetes Engine, EKS – Elastic Kubernetes Service, AKS – Azure Kubernetes Service).
  - Zaletą chmurowego Kubernetes jest automatyczne zarządzanie infrastrukturą przez dostawców chmurowych.
  - Łatwa skalowalność i integracja z innymi usługami chmurowymi.
- **Kubernetes on-premise**:
  - Minikube, K3s – Lekkie wersje Kubernetes dla małych środowisk lub testowania.
  - Trudniejsza konfiguracja i zarządzanie infrastrukturą (potrzebne są umiejętności administratorskie w k8s).
  - Zaleta: większa kontrola nad infrastrukturą i brak zależności od dostawców chmurowych.

Podsumowanie

- Kubernetes stał się standardem do zarządzania aplikacjami kontenerowymi w dużej skali.
- Dzięki automatyzacji, skalowalności i wsparciu dużych firm, Kubernetes jest wybierany przez większość organizacji rozwijających aplikacje w środowiskach rozproszonych.
- **Wyzwania w Kubernetes**:
  - Złożoność zarządzania: Kubernetes wymaga zaawansowanych umiejętności administracyjnych.
  - Koszty: Wdrożenie i utrzymanie Kubernetes na dużą skalę może być kosztowne.
- **Przyszłość Kubernetes**:
  - Lepsze wsparcie dla rozproszonych aplikacji i mikroserwisów.
  - Rozwój funkcji związanych z bezpieczeństwem (np. rozszerzone RBAC, lepsze wsparcie dla polityk bezpieczeństwa).
  - Dalszy rozwój narzędzi do monitorowania i automatyzacji (np. lepsza integracja z Prometheus, Grafana).