Implementacja i testy skalowalności systemu wideokonferencyjnego

2025-11-17

Jakub Jędrzejczyk **Sebastian Kwaśniak** **Anna Berkowska**

Wprowadzenie do Kubernetes

Czym jest Kubernetes?

- System orkiestracji kontenerów open-source.
- Umożliwia automatyczne wdrażanie, skalowanie i zarządzanie aplikacjami kontenerowymi.
- Początkowo opracowany przez Google, obecnie rozwijany przez Cloud Native Computing Foundation (CNCF).

Architektura Kubernetes

- **Master Node** – zarządza klastrem:
  - kube-apiserver
  - etcd
  - kube-scheduler
  - kube-controller-manager
- **Worker Nodes** – uruchamiają kontenery:
  - kubelet
  - kube-proxy
  - Container Runtime (np. containerd)

Podstawowe obiekty

1. **Pod** – najmniejsza jednostka w Kubernetes, może zawierać jeden lub więcej kontenerów.
2. **Service** – stały punkt dostępu do Podów; definiuje sposób komunikacji wewnątrz klastra.
3. **Deployment** – zarządza replikacją i aktualizacją Podów.
4. **ConfigMap** i **Secret** – przechowują konfiguracje i dane poufne.

Skalowanie i samonaprawianie

- Kubernetes automatycznie przywraca niedziałające Pody.
- HPA (Horizontal Pod Autoscaler) skalują liczbę replik na podstawie obciążenia CPU, pamięci lub niestandardowych metryk.
- Rolling updates – aktualizacje aplikacji bez przestojów.

Sieć w Kubernetes

- Każdy Pod ma własny adres IP w obrębie klastra.
- Komunikacja realizowana przez CNI (Container Network Interface).
- Przykładowe implementacje: Calico, Flannel, Cilium.

Sieć w Kubernetes i znaczenie pluginów sieciowych

- Kubernetes wymaga warstwy sieciowej, aby umożliwić komunikację między Podami, Service'ami i zewnętrznym światem.
- Plugin sieciowy (CNI) implementuje tę warstwę i odpowiada za:
  - Przydzielanie adresów IP Podom.
  - Routing ruchu sieciowego w klastrze.
  - Polityki bezpieczeństwa sieciowej (Network Policies).

Wymagania API Container Network Interface (CNI)

- CNI definiuje prosty i elastyczny zestaw API potrzebnych do zarządzania siecią kontenerów.
- Podstawowe API, które każdy CNI plugin musi implementować to:
  - **ADD**: Tworzy i konfiguruje interfejs sieciowy dla kontenera (Pod), przydziela adres IP i ustawia routing.
  - **DEL**: Usuwa interfejs sieciowy oraz zwalnia zasoby sieciowe po usunięciu kontenera.
  - **CHECK**: (opcjonalne) Sprawdza czy konfiguracja sieci dla kontenera jest prawidłowa i działa.

Popularne pluginy sieciowe

Plugin	Zalety	Zastosowania
Calico	Zaawansowane polityki sieci, wysoka skalowalność	Bezpieczeństwo, chmury hybrydowe
Flannel	Prosty setup, minimalna konfiguracja	Małe i średnie klastry
Cilium	Wykorzystuje eBPF, dobre do mikroserwisów	Zaawansowane polityki, observability
Weave Net	Łatwy w instalacji, sieć nakładkowa	Łatwy start, małe środowiska

Wpływ wyboru pluginu sieciowego na klaster

- Wydajność komunikacji między Podami zależy od implementacji pluginu.
- Ochrona i kontrola ruchu sieciowego realizowana przez polityki sieciowe.
- Kompatybilność z infrastrukturą chmurową i sprzętową.
- Możliwość debugowania i monitorowania ruchu sieciowego.

Container Runtime Interface (CRI) w Kubernetes

- CRI to standardowy interfejs API gRPC umożliwiający komunikację między kubeletem (agentem na węźle) a środowiskiem uruchomieniowym kontenerów.
- Pozwala Kubernetes na obsługę różnych środowisk uruchomieniowych kontenerów (np. containerd, CRI-O) bez potrzeby zmian w klastrze.
- Umożliwia elastyczność i unika uzależnienia od jednego dostawcy środowiska kontenerowego.

Jak działa CRI?

- Kubelet działa jako klient, wywołując metody CRI do zarządzania cyklem życia kontenerów na węźle.
- CRI dostarcza dwie podstawowe usługi:
  - **RuntimeService** – zarządza uruchamianiem, zatrzymywaniem i monitorowaniem kontenerów oraz sandboxów Podów.
  - **ImageService** – zarządza pobieraniem, kasowaniem i zarządzaniem obrazami kontenerów.

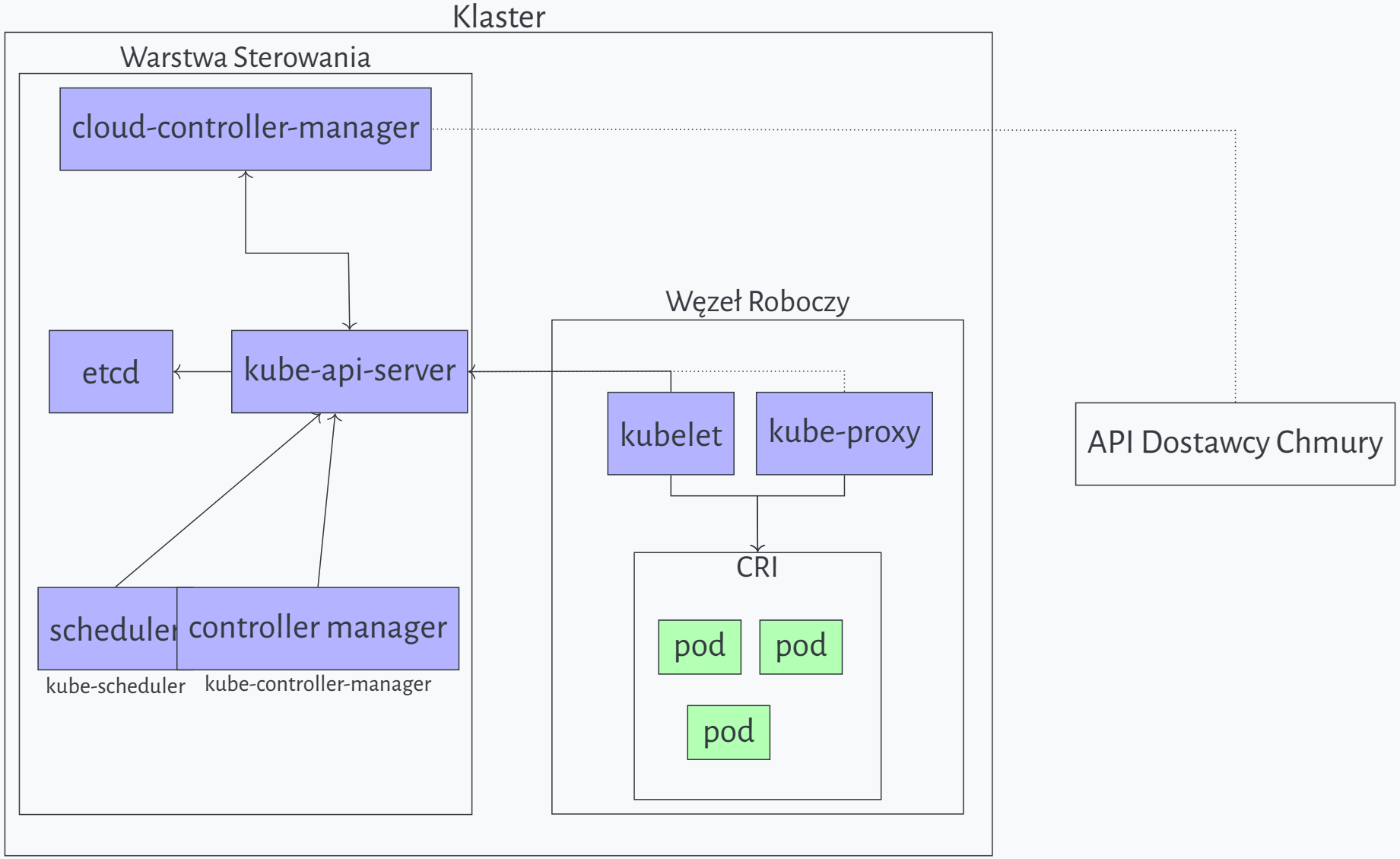
Znaczenie CRI dla Kubernetes

- Separacja odpowiedzialności: Kubernetes operuje na wysokim poziomie, a szczegóły dotyczące uruchamiania kontenerów deleguje do CRI.
- Ułatwia stosowanie różnych silników kontenerowych zgodnych z CRI.
- Zapewnia spójność działania klastra niezależnie od konkretnego środowiska uruchomieniowego.

Przykłady środowisk zgodnych z CRI

- **containerd** – lekki i popularny runtime stworzony pod kątem Kubernetes.
- **CRI-O** – specjalistyczne środowisko lightweight, zoptymalizowane pod Kubernetes.
- **Docker (przez shim containerd)** – Kubernetes korzysta z Docker pośrednio przez warstwę CRI.

Architektura Kubernetes



Rysunek 1: Architektura Kubernetes