Wrocław University of Science and Technology

Przetwarzanie danych masowych

Wykład 10 – Platformy zarządzania zasobami obliczeniowymi – Kubernetes

dr inż. Tomasz Kajdanowicz, Roman Bartusiak, Piotr Bielak, Krzysztof Rajda

15 grudnia 2021 r.

HR EXCELLENCE IN RESEARCH

Wrocław University of Science and Technology

Kubernetes

- platforma do zarządzania zasobami obliczeniowymi
- utworzona przez Google
- owoc pracy i ogromnego doświadczenia z rozproszonymi środowiskami
- open source
- duża społeczność
- Kubernetes = K8s

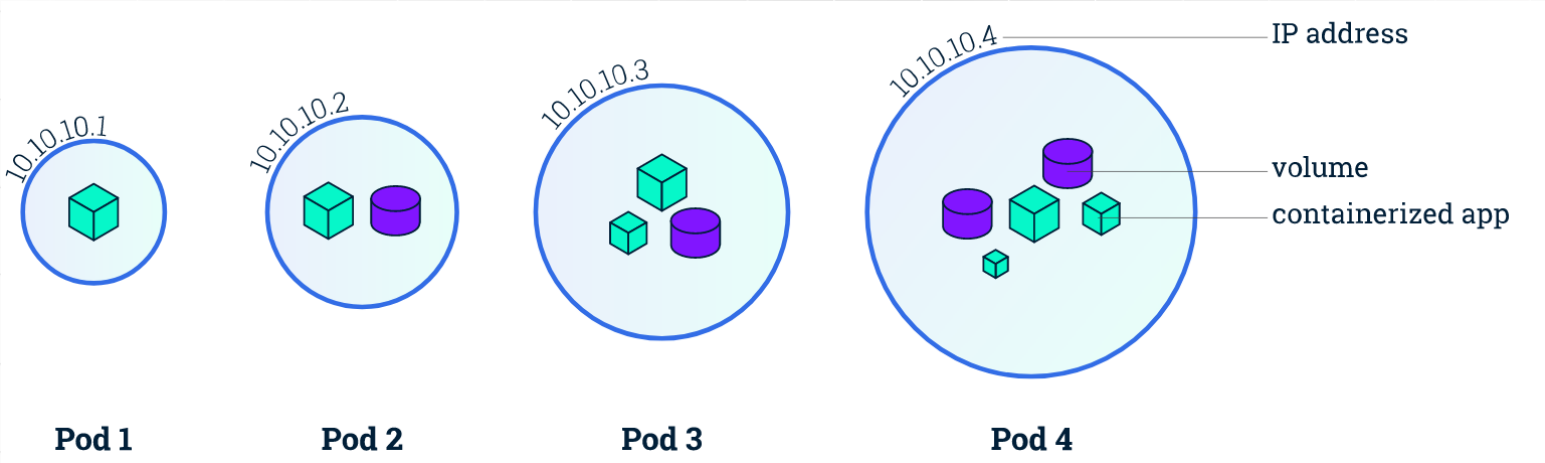
Wrocław University of Science and Technology

Kubernetes

- Nie jest prosty w instalacji / wdrożeniu
- Wiele zależnych serwisów
- Zarządzanie kontenerami
- K3S
  - uproszczenie K8s
  - łatwiejszy w instalacji i zarządzaniu
  - używa SQLite zamiast ETCD (może być wąskim gardłem w przypadku dużego ruchu)
- Lokalne uruchamanie
  - Minikube
  - microk8s
  - K3d

Kontener a pod

- najmniejszą jednostką w K8s jest **pod**
- koncepcyjnie podobny do kontenera
- jeden pod może zawierać wiele kontenerów
- "jeden pod = jedna aplikacja"
- każdy pod ma przypisany adres IP (brak routingu)



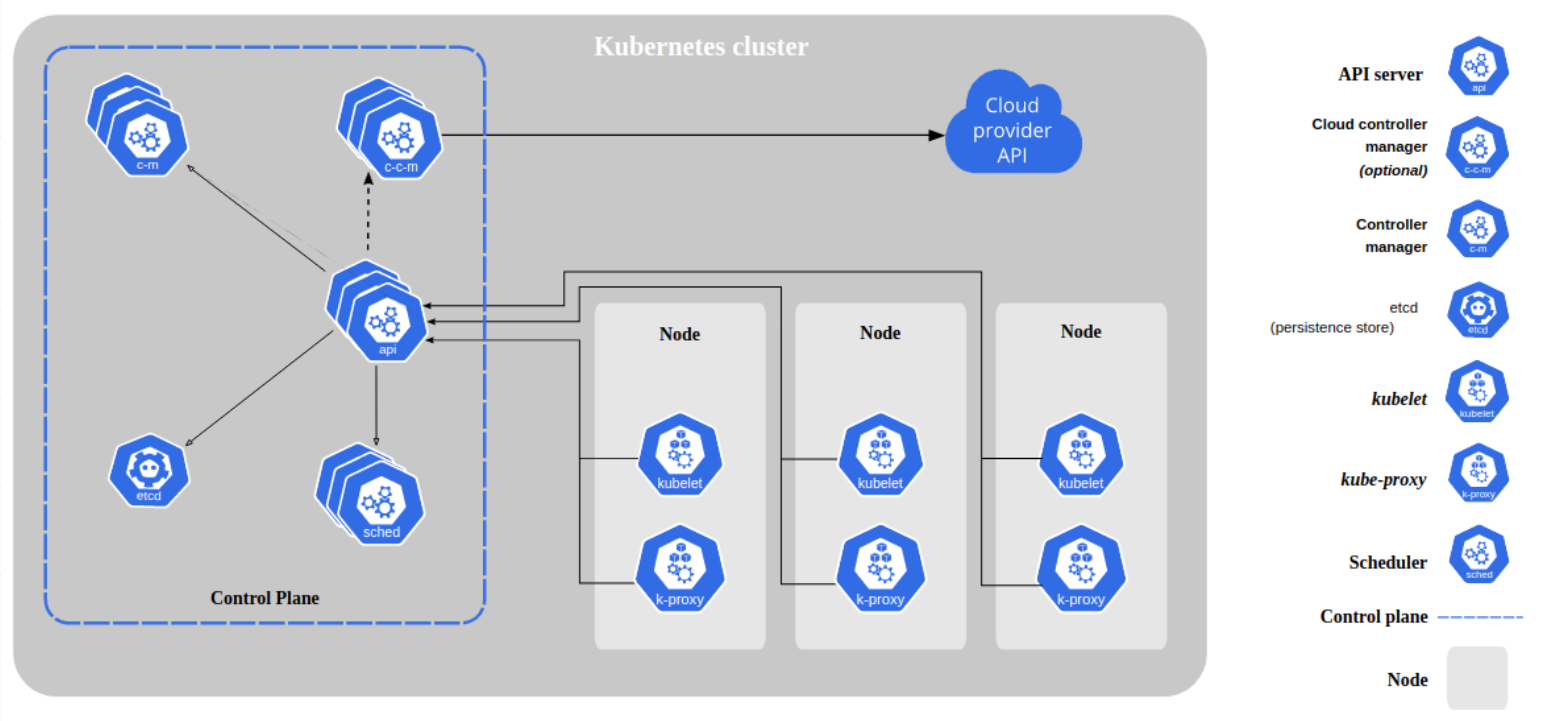
Źródło:
https://kubernetes.io/pl/docs/tutorials/kubernetes-basics/explore/explore-intro/

Architektura

Wyróżniamy dwa typy węzłów (w każdym działa wiele mikroserwisów):

- master (control plane):
  - API server,
  - Scheduler,
  - Controller Manager,
  - ETCD,
- worker:
  - Kubelet,
  - Kube-proxy,
  - *Środowisko uruchomieniowe kontenerów**

Architektura – diagram



Źródło: https://kubernetes.io/docs/concepts/overview/components/

Master – komponenty (1)

API server (*kube-apiserver)*:

- punkt dostępu do API Kubernetesa,
- front end dla control plane'a,
- zaprojektowany dla łatwego skalowania horyzontalnego (poprzez load balancery),

Master – komponenty (2)

Scheduler (*kube-scheduler)*:

- oczekuje na nowe **pody**, które jeszcze nie mają przypisanego węzła,
- wybiera węzeł, na którym zostaną uruchomione,
- uwzględnia:
  - indywidualne i grupowe wymagania zasobów (*resource requirements*),
  - ograniczenia sprzętowe i aplikacyjne (*hardware/software/policy constraints*),
  - specyfikacje nt. preferencji lokalizacji podów (*affinity and anti-affinity specifications*),
  - lokalność danych,
  - zaburzenia między-zadaniowe,
  - ograniczenia czasowe (*deadlines*),

Master – komponenty (3)

Controller Manager (*kube-controller-manager*) – uruchamia kontrolery:

- Node Controller – odpowiedzialny za obserwowanie węzłów i raportowanie gdy węzeł zostanie utracony (wyłączy się, wystąpi błąd, awaria sieci itp.)
- Replication Controller – odpowiedzialny za utrzymywanie właściwej liczby podów dla każdego obiektu *replication controller* w systemie
- Endpoints Controller – propaguje endpointy (tzn. łączy *Service* z podami)
- Service Account & Token Controllers – tworzy domyślne konta i klucze dostępu do API dla nowych przestrzeni nazw (*namespace*),

Master – komponenty (4)

ETCD:

- baza danych
- spójność i wysoka dostępność
- baza danych typu klucz-wartość (ang. *key-value store*)
- przechowuje wszystkie dane na temat klastra Kubernetesowego
- może być uruchomiona w trybie *standalone* albo w trybie *cluster* (wiele instancji),

Worker – komponenty (1)

kubelet:

- proces uruchomiony w tle $\rightarrow$ *daemon*
- uruchomiony na każdym węźle typu *worker*,
- zapewnia, że kontenery w podzie są uruchomione,
- zarządza tylko kontenerami utworzonymi przez Kubernetesa,

Worker – komponenty (2)

kube-proxy:

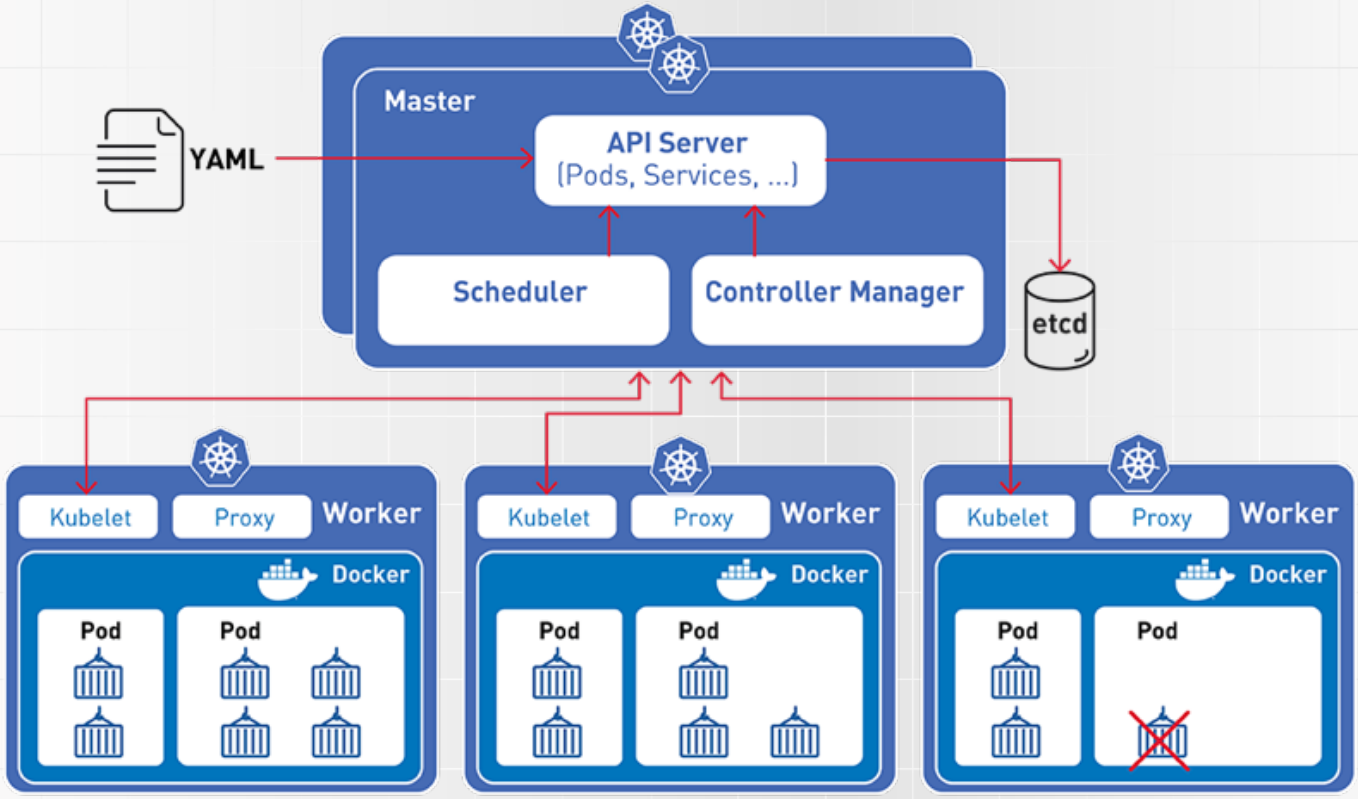
- proxy sieciowe,
- uruchomione na każdym węźle typu *worker*,
- zarządza regułami sieciowymi (ang. *network rules*) na węźle
- ruch wchodzący (ingress),
- ruch wychodzący (egress),
- wykorzystuje filtrowanie pakietów na poziomie systemu operacyjnego (jeśli dostępne),

Worker nodes components

Środowisko uruchomieniowe kontenerów:

- właściwe narzędzie, które uruchamia kontenery, zarządza obrazami itd.
- Docker, containerd,
- cri-o, rktlet,
- dowolna implementacja *Kubernetes CRI (Container Runtime Interface)*

Architektura – podsumowanie



Źródło: https://kubernetes.io

Obiekty w Kubernetesie (1)

- każdy obiekt jest opisywany za pomocą manifestu
- plik w formacie YAML
- specyfikujemy:
  - typ obiektu: `kind`
  - metadane i selektory: `metadata`, `selector`
  - właściwą zawartość obiektu / konfigurację: `spec`
- obiekty są tworzone za pomocą komendy:
  `kubectl apply -f <path/to/file.yaml>`
- alternatywnie można użyć narzędzia Helm (następny wykład!)

Obiekty w Kubernetesie (2)

Najważniejsze typy obiektów w K8s:

- Deployment
- Service
- DaemonSet
- StatefulSet
- ConfigMap
- Secret
- PersistentVolume(Claim)

Deployment

- podstawowy typ obiektu K8s
- w deklaratywny sposób opisuje pody
- może być również używany do zarządzania ReplicaSetami

```yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 1
  template:
    metadata:
      labels:
        app: nginx
    spec:
      volumes:
        - name: nginx-sample-page
          configMap:
            name: nginx-sample-page
      containers:
      - name: nginx
        image: nginx:latest
        ports:
          - containerPort: 80
        volumeMounts:
          - name: nginx-sample-page
            mountPath: /usr/share/nginx/html/index.html
            subPath: index.html
```

Service

- zapewnia routing sieciowy do i między podami
- typy:
  - ClusterIP – widoczność tylko wewnątrz klastra,
  - NodePort – aplikacja dostępna na wybranym porcie na każdym węźle,
  - LoadBalancer – wykorzystanie zewnętrznego load balancera,
  - ExternalName – dostęp do usługi poprzez nazwę np. `foo.bar.example.com`,

```yaml
---
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
  labels:
    app: nginx
spec:
  type: NodePort
  ports:
  - port: 80
    nodePort: 30570
    protocol: TCP
    name: http
  selector:
    app: nginx
```

Configmap

- sposób przekazywania plików konfiguracyjnych do aplikacji,
- dwa sposoby użycia:
  - przez plik
  - przez zmienne środowiskowe

```yaml
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx-sample-page
data:
  index.html: |
    <html>
      <head>
        <title>Example page</title>
      </head>
      <body>
        <h1>This page is served via Nginx!</h1>
      </body>
    </html>
```

Pozostałe typy obiektów (1)

- Daemonset:
  - podobny do Deploymentu
  - zapewnia że podane pody będą uruchomiona **na każdym** węźle
- Statefulset:
  - zaprojektowany dla aplikacji stanowych
  - podobne do Deploymentu
  - zapewnia uruchamianie konkretnych (replik) podów na tych samych węzłach
  - dostęp do wolumenów
  - stała adresacja sieciowa
- Secret:
  - sposób przekazywania haseł i innych poufnych danych konfiguracyjnych do aplikacji
  - podobne do Configmapy

Pozostałe typy obiektów (2)

- PersistentVolume(Claim):
  - sposób dostępu do wolumenów (nieulotnej przestrzeni dyskowej)
  - PersistentVolume (PV):
    - istniejący wolumen dyskowy
    - utworzony przez administratora
    - utworzony przez StorageClass
    - nie zostanie usunięty przy usuwaniu poda
  - PersistenVolumeClaim (PVC):
    - żądanie uzyskania wolumenu
    - podobne do poda
    - pod zużywa zasoby obliczeniowe, PVC – zasoby dyskowe (PV)
  - wybrane obsługiwane typy PV:
    - AWS Elastic Block Store (EBS)
    - Azure Disk & Azure File
    - CephFS volume
    - GCE Persistent Disk
    - hostPath - HostPath volume (tylko dla single node; nie działa dla multi-node)
    - Network File System (NFS) storage

Przykład PV i PVC (1)

PersistenVolume za pomocą `hostPath`:

```yaml
---
apiVersion: v1
kind: PersistentVolume
metadata:
  name: task-pv-volume
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/mnt/data"
```

Przykład PV i PVC (2)

PersistenVolumeClaim:

```yaml
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: task-pv-claim
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 3Gi
```

Przykład PV i PVC (3)

Wykorzystanie wolumenu w podzie:

```yaml
---
apiVersion: v1
kind: Pod
metadata:
  name: task-pv-pod
spec:
  volumes:
    - name: task-pv-storage
      persistentVolumeClaim:
        claimName: task-pv-claim
  containers:
    - name: task-pv-container
      image: nginx
      ports:
        - containerPort: 80
          name: "http-server"
      volumeMounts:
        - mountPath: "/usr/share/nginx/html"
          name: task-pv-storage
```

Wrocław University of Science and Technology

Przetwarzanie danych masowych

Wykład 10 – Platformy zarządzania zasobami obliczeniowymi – Kubernetes

dr inż. Tomasz Kajdanowicz, Roman Bartusiak, Piotr Bielak, Krzysztof Rajda

15 grudnia 2021 r.