Kubernetes

Ondřej Smola

29.04.2022 | CTC

Osnova

- Nasazení aplikací
  - Pod
  - ReplicaSet
  - Deployment
  - StatefulSet
  - DaemonSet
  - Job and Cronjob
- Síťový model
- Úložiště

Objekt

- Perzistetní entita popisující
  - Kontejnerizovanou aplikaci uvnitř klusteru s umístěním na stroji
  - Dostupné prostředky (cpu, mem, disk, network)
  - Pravidla (restart, health-check, upgrade, fault-tolerance)
- Popisuje požadovaný stav, k8s provádí akce, aby bylo dosaženo cílového stavu

- Práce pomocí
  - kubectl (https://kubernetes.io/docs/reference/kubectl)
  - API (https://kubernetes.io/docs/concepts/overview/kubernetes-api/)
  - V obou případech komunikace s API serverem

Objekt

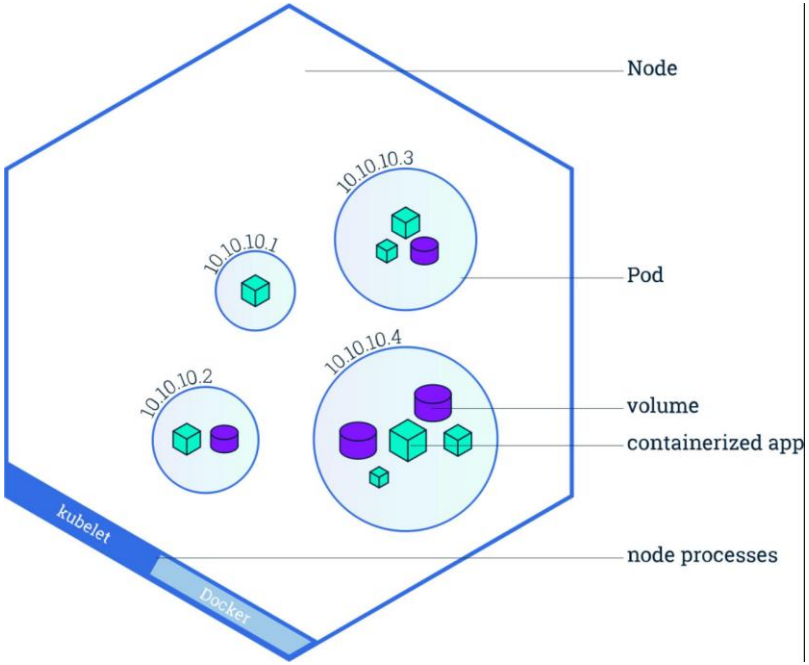
- spec, požadovaný stav
- status, aktuální stav
- apiVersion: verze K8S
- kind: druh objektu

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2 # tells deployment to run 2 pods matching the template
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

- metadata
  - name, unikátní jméno v rámci jmenného prostoru
  - UID, pro identifikací znovuvytvoření se stejným jménem
  - namespace, jmenný prostor

Pod

- Základní stavební jednotka pro plánování
- Jedna instance aplikace naplánovaná na jeden stroj
- Sdílí síťové a diskové prostředky




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

Pod

- Kontejner per pod
  - V rámci podu běží jediný kontejner
  - Abstrakce před správou kontejneru
- Více kontejnerů per pod
  - Abstrakce aplikační jednotky
  - 2 mikroslužby a cache
  - Sdílí prostředky, jsou spravovány společně

- Není možné spravovat jednotlivé kontejnery, pouze celý pod
- Práce přímo s pody se používá velmi zřídka, místo toho se pracuje s **kontroléry objektů** (deployment, statefulset …)

Kontroléry objektů

- Spravují a zapouzdřují práci s pody
  - Deployment
  - StatefulSet
  - DaemonSet
- Úkoly
  - Dynamické škálování podů
  - Nahrazení podu při aplikační chybě nebo pádu stroje
  - Aktualizaci podů při změně konfigurace
- Pody vytváří na základě šablony (template)
- Změna v konfiguraci

Šablona

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2 # tells deployment to run 2 pods matching the template
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

Šťítky

```
"metadata": {
  "labels": {
    "key1" : "value1",
    "key2" : "value2"
  }
}
```

- klíč=hodnota (hodnotou může být prázdný řetězec)
- Slouží k organizaci a identifikaci objektů (ne unikátní)
- Pojmenování
  - prefix/klíč
    - myapp.com/environment=production
    - kubernetes.io/arch=amd64 (rezervované)
  - klíč
    - environment=production
- Využívány kontroléry pro identifikaci vlastních objektů
- Výběr pomocí selektorů

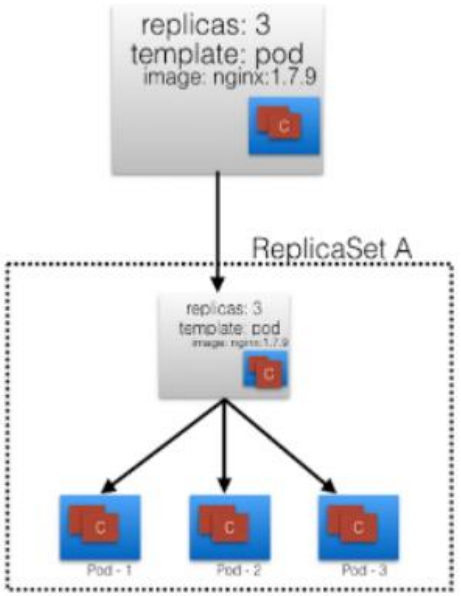
Selektory šťítků

- Definice pro výběr objektů
- Pomocí rovnosti
  - “environment=production,tier!=frontend”
  - oddělení čárkou znamená logické AND
- Pomocí množin
  - environment in (production, qa)
  - tier notin (frontend, backend)
  - partition
  - !partition

- kubectl get pods -l “environment=production,tier=frontend”

Deployment

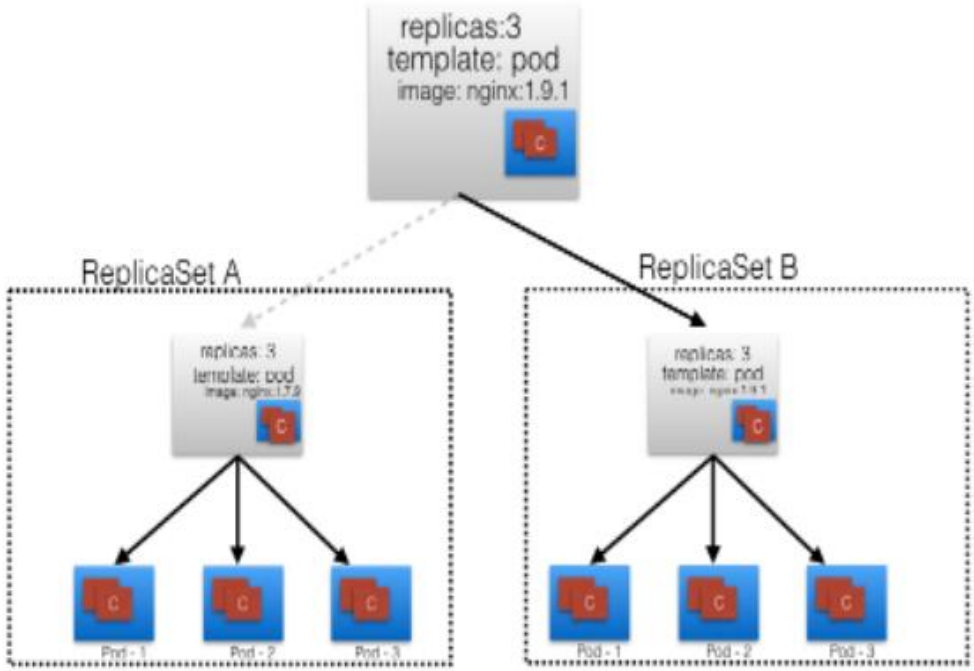
- Poskytuje deklarativní aktualizaci podů v rámci replika setu.
- Řízeno pomocí DeploymentController, který běží na masteru
- Zajišťuje, aby aktuální stav podů odpovídal požadovanému stavu
- Aktualizace
  - Rollout
  - Rollback
- Značka pod-template-hash



Deployment (ReplicaSet A Created)

Deployment aktualizace

- Rolling update
- Revize



Deployment (ReplicaSet B Created)

ReplicaSet

- Zajišťuje požadovaný počet podů
- Pody identifikuje pomocí selektorů značek
- Nové pody vytváří pomocí šablony
  - metadata.ownerReferences

- Základní atributy
  - selector
  - template
  - spec.replicas

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

StatefulSet

- ReplicaSet pro aplikace se stavem (např. databáze)
- Vhodné pokud je požadován
  - Stabilní, unikátní síťový identifikátor
  - Stabilní perzistentní úložiště
  - Systematické a kontrolované nasazení a škálování
  - Kontrolované automatické aktualizace

- Pro síťovou identitu lze použít headless service (viz dale)
- Pro ukládání dat používají Persistentní úložiště

DaemonSet

- Pody běží na všech nebo vybraných nodech
- Typické úlohy
  - Monitoring
  - Kolekce logů
  - Pody zajišťující
    - připojování úložiště (CSI)
    - síťovou konektivitu (CNI)
- Automaticky přidává a odebírá pody při přidání a odebrání nodu

Job

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: pi
spec:
  template:
    spec:
      containers:
      - name: pi
        image: perl
        command: ["perl",  "-Mbignum=bpi", "-wle", "print bpi(2000)"]
      restartPolicy: Never
  backoffLimit: 4
```

- Úloha
  - Dokončení když pod je úspěšně ukončen
  - Počet dokončení `.spec.completions`
  - Pracovní fronta – N podů `.spec.parallelism`
- Mód dokončení
  - Bez indexu
  - Indexovaný
- TTL pro odstranění dokončených podů
- CronJob – spouštění Job podle plánu (cron format)

Konfigurační mapy

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: game-demo
data:
  # property-like keys; each key maps to a simple value
  player_initial_lives: "3"
  ui_properties_file_name: "user-interface.properties"

  # file-like keys
  game.properties: |
    enemy.types=aliens,monsters
    player.maximum-lives=5
  user-interface.properties: |
    color.good=purple
    color.bad=yellow
    allow.textmode=true
```

- Využití
  - Argumenty podu
  - Proměnné prostředí podu
  - Read-only souboru uvnitř podu
  - Možnost načtení pomocí API
- Neměly by obsahovat hesla a tajné informace
  - Použití secret

- V případě mapování pomocí souboru jsou automaticky aktualizovány
- Možnost nastavit jako neměnné (immutabilní)

```yaml
spec:
  containers:
    - name: demo
      image: alpine
      command: ["sleep", "3600"]
      env:
        # Define the environment variable
        - name: PLAYER_INITIAL_LIVES # Notice that the case is different here
                                     # from the key name in the ConfigMap.
          valueFrom:
            configMapKeyRef:
              name: game-demo           # The ConfigMap this value comes from.
              key: player_initial_lives # The key to fetch.
        - name: UI_PROPERTIES_FILE_NAME
          valueFrom:
            configMapKeyRef:
              name: game-demo
              key: ui_properties_file_name
      volumeMounts:
      - name: config
        mountPath: "/config"
        readOnly: true
  volumes:
    # You set volumes at the Pod level, then mount them into containers inside that Pod
    - name: config
      configMap:
        # Provide the name of the ConfigMap you want to mount.
        name: game-demo
        # An array of keys from the ConfigMap to create as files
        items:
        - key: "game.properties"
          path: "game.properties"
        - key: "user-interface.properties"
          path: "user-interface.properties"
```

Secret

- Vychází z konfiguračních map
- Typy
  - opaque
  - ssh-auth
  - tls
  - dockercfg
- Data
  - ukládána jako Base64 enkódovaná
  - nejsou vypisována na konzoli
  - nejsou ale šifrována

```
kubectl create secret generic test-db-secret --from-literal=user=testuser --from-literal=password=iluvtests
```

Jmenné prostory

- Isolace objektů do oddělených prostorů
- Pravidla pak specifikována per prostor

- Základní jmenné prostory
  - default
  - kube-system
  - kube-node-lease

- Některé objekty nemají jmenný prostor
  - Node
  - Persistentní disky

- kubectl -n my-namespace get pods

Síťový model

Síťový model

- Každý pod má svoji IP adresu
  - Není třeba řešit mapování portů
- Pravidla
  - Každý pod může komunikovat s jiným podem bez použití NAT
  - Agenti na stroji mohou komunikovat s pody na stroji
  - Pody v jmenném prostoru hosta mohou komunikovat s ostatními pody
- Kompatibilní s síťováním v rámci VM
- Implementace závisí na poskytovateli
  - Container network interface

Komunikace

- Kontejnery v rámci podu komunikují pomocí síťě
- Komunikace mezi pody probíhá pomocí poskytovatele síťování uvnitř k8s – CNI
- Externí (i interní) přístup a směrování mezi skupinami podů probíhá pomocí služeb (Service)

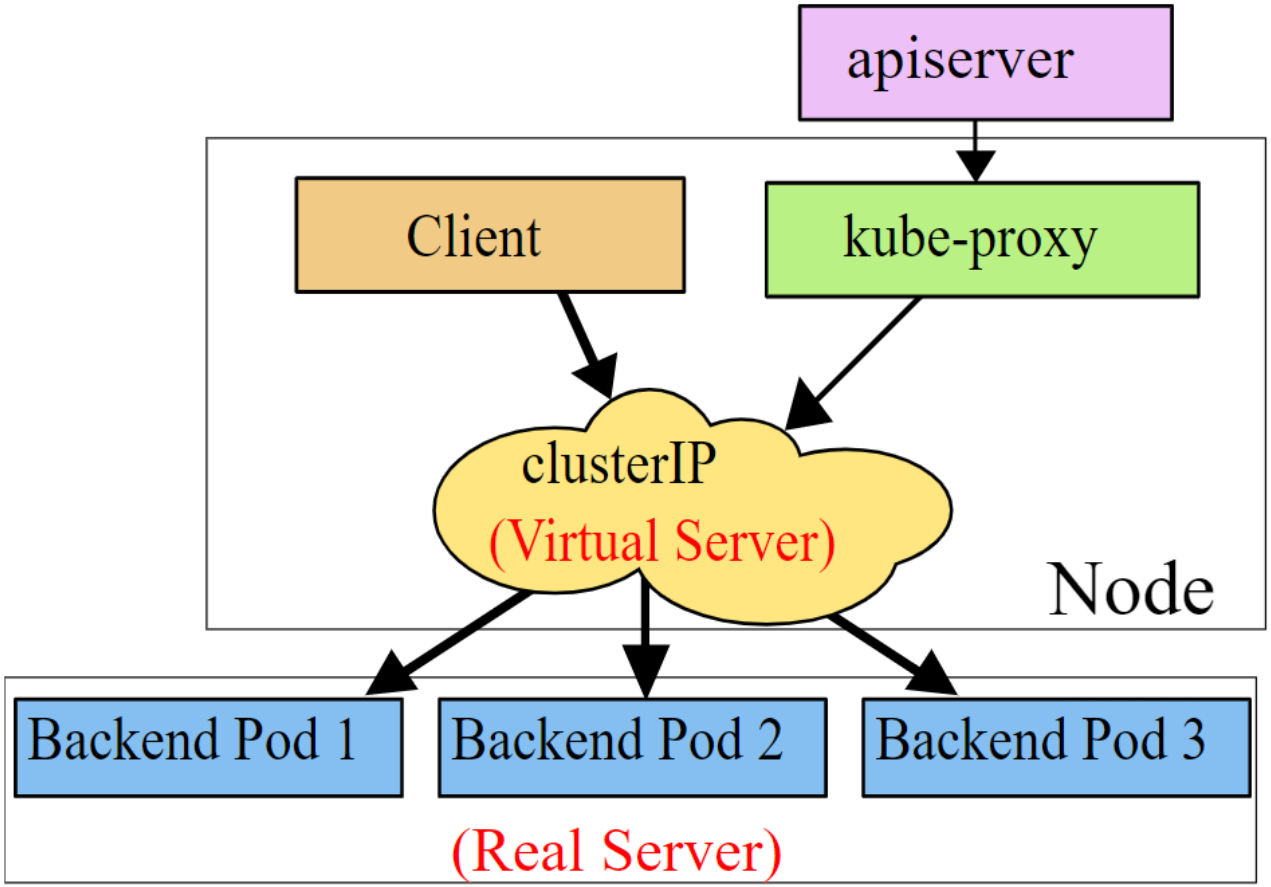
- Příklady poskytovatelů – CNI
  - Cillium (https://cilium.io/)
  - Calico (https://www.tigera.io/project-calico/)
  - Flannel (https://github.com/flannel-io/flannel)
  - Antrea (https://antrea.io/)

Služba

- Síťová abstrakce nad skupinou podů
- Definuje logický výběr podů pomocí selektoru a způsob přístupu k nim

- Cílem je umožnit propojení nezávislých skupin podů
- Typy
  - ClusterIP
  - LoadBalancer
  - NodePort
  - ExternalName

Implementace pomocí kube-proxy/iptables



Detekce služeb uvnitř podu

- Proměnné prostředí
  - {SVCNAME}_SERVICE_HOST a {SVCNAME}_SERVICE_PORT

```
REDIS_MASTER_SERVICE_HOST=10.0.0.11
REDIS_MASTER_SERVICE_PORT=6379
REDIS_MASTER_PORT=tcp://10.0.0.11:6379
REDIS_MASTER_PORT_6379_TCP=tcp://10.0.0.11:6379
REDIS_MASTER_PORT_6379_TCP_PROTO=tcp
REDIS_MASTER_PORT_6379_TCP_PORT=6379
REDIS_MASTER_PORT_6379_TCP_ADDR=10.0.0.11
```

- DNS
  - Řešení pomocí add-on CoreDNS (https://coredns.io/)
  - my-service.my-ns
- Headless service
  - Název služby vrací seznam DNS záznamů s IP podů

Typ ClusterIP

- Selector
- TargetPort
  - jméno
  - číslo

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    app.kubernetes.io/name: proxy
spec:
  containers:
  - name: nginx
    image: nginx:11.14.2
    ports:
      - containerPort: 80
        name: http-web-svc
```

```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  selector:
    app.kubernetes.io/name: proxy
  ports:
  - name: name-of-service-port
    protocol: TCP
    port: 80
    targetPort: http-web-svc
```

Typ NodePort

- Otevře port na každém stroji v clusteru, který bude směrován na vybrané pody

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  type: NodePort
  selector:
    app: MyApp
  ports:
      # By default and for convenience, the `targetPort` is set to the same value as the `port` field.
    - port: 80
      targetPort: 80
      # Optional field
      # By default and for convenience, the Kubernetes control plane will allocate a port from a range (default: 30000-32767)
      nodePort: 30007
```

Typ LoadBalancer

- Především pro použití v cloudu
- Služba dostane veřejnou IP adresu
- Zpracován externě cloudovým správcem
- Výsledná adresa a postup je uložen v poli status

```yaml
apiVersion: v1
kind: Service
metadata:
  name: redis-service
spec:
  type: LoadBalancer
  selector:
    app: redis
  ports:
    - protocol: TCP
      port: 6379
status:
  loadBalancer:
    ingress:
    - ip: 192.0.2.127
```

Síťová pravidla

- Pravidla pro kontrolu síťových operací podů
  - Ingress – příchozí spojení
  - Egress – odchozí spojení

- Příklad – zakáž příchozí komunikace v jmenném prostoru

```yaml
---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny-ingress
spec:
  podSelector: {}
  policyTypes:
  - Ingress
```

Úložiště

Úložiště (Volume)

- Abstrakce adresáře s daty, které přežijí restart podu
- Možné sdílet mezi pody na různých nodech

- Implementace, způsob ukládání a obsah jsou definován typem

- Trvalé
- Dočasné
- Projekce

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: test-ebs
spec:
  containers:
  - image: k8s.gcr.io/test-webserver
    name: test-container
    volumeMounts:
    - mountPath: /test-ebs
      name: test-volume
  volumes:
  - name: test-volume
    # This AWS EBS volume must already exist.
    awsElasticBlockStore:
      volumeID: "<volume id>"
      fsType: ext4
```

Trvalé úložiště

- Azure, AWS, Ceph, nfs, local
- Módy přístupu
  - ReadWriteOnce
  - ReadOnlyMany
  - ReadWriteMany
  - ReadWriteOncePod
- Kapacita (8Gi)
  - Možnost změny kapacity
- Mód
  - Filesystem
  - Block

Trvalé úložiště

- Odstranění
  - Retain (zachovat data)
  - Recycle (rm –rf *)
  - Delete (odstranění)
- Přiřazeno pomocí PersistentVolumeClaim (PVC)

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: myclaim
spec:
  accessModes:
    - ReadWriteOnce
  volumeMode: Filesystem
  resources:
    requests:
      storage: 8Gi
  storageClassName: slow
  selector:
    matchLabels:
      release: "stable"
    matchExpressions:
      - {key: environment, operator: In, values: [dev]}
```

Dočasné úložiště

- Svázáno s životním cyklem podu, nemusí být trvalé

- emptyDir
  - dočasný adresář
  - svázán s životním cyklem podu
  - podpora tmpfs
- configMap, secret:
  - obsahem souboru jsou hodnoty z configMap/secret
- downwardAPI
  - poskytuje informace o podu uvnitř kontejneru

Projekce

- secret
- configMap
- downwardAPI

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: volume-test
spec:
  containers:
  - name: container-test
    image: busybox:1.28
    volumeMounts:
    - name: all-in-one
      mountPath: "/projected-volume"
      readOnly: true
  volumes:
  - name: all-in-one
    projected:
      sources:
      - secret:
          name: mysecret
          items:
            - key: username
              path: my-group/my-username
      - downwardAPI:
          items:
            - path: "labels"
              fieldRef:
                fieldPath: metadata.labels
            - path: "cpu_limit"
              resourceFieldRef:
                containerName: container-test
                resource: limits.cpu
      - configMap:
          name: myconfigmap
          items:
            - key: config
              path: my-group/my-config
```