1. Úvod

V tomto článku do větších podrobností rozebereme tři rozdílné způsoby, jakými běžně provádíme deploy aplikací v K8s – tedy **Deployment, StatefulSet** a **DaemonSet**. Vše si budeme demonstrovat na jednoduchém image busybox.

2. Deployment

Nejjednodušším a také nejpoužívanějším způsobem, jak provést deploy vaší aplikace, je rozhodně prostý Deployment. Jedná se Kubernetes kontrolér, který porovnává aktuální stav clusteru ke stavu, který je popsán ve vašem Deployment manifestu – tzv. stav žádaný.

Tedy pokud vytvoříte Deployment aplikace s jednou replikou, pak tento kontrolér zjistí, že požadovaný stav je **ReplicaSet 1**, přičemž současný stav je **ReplicaSet 0**, tedy vytvoří ReplicaSet, který zase následně vytvoří potřebný pod. Na příkladu tedy, Deployment **Busytest** vytvoří ReplicaSet **Busytest-<replica-set-id>**, který následně vytvoří pod **Busytest-<replica-set>-<pod-id>**

Deployment se nejčastěji používá pro stateless aplikace. Pokud chcete, je možné ukládat stav Deploymentu přiřazením **PersistentVolume**, čímž se z něj stává stateful aplikace. Je si však třeba uvědomit, že všechny pody budou sdílet stejný volume, a tím pádem i všechna data na něm, a je potřeba na toto myslet při návrhu aplikace a jejího zacházení s daty.

Abychom provedli deploy naší aplikace **Busytest,** použijeme následující manifest:

```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: busytest
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: busytest
    spec:
      containers:
```

```
      - name: busytest
        image: "k8s-registry/busybox:latest"
        volumeMounts:
        - name: busytest
          mountPath: /app/
      volumes:
      - name: busytest
        persistentVolumeClaim:
          claimName: busytest
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: busytest
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 50Mi
  storageClassName: default
```

V logu aplikace pak uvidíme, že logy začínají řádkem 1. Pokud ale provedeme naškálování Deploymentu na 2 repliky pomocí

```
kubectl scale deployment busytest --replicas=2
```

a podíváme se do logů nově přibyvšího podu, zjistíme, že logy začínají minimálně řádkem 2 a ne všechna čísla budou nutně zastoupena. Jedná se o krásnou demonstraci sdíleného PV, kde kubernetes logs čte sice jen řádky daného podu, ale soubor, ze kterého čte, je sdílen mezi více pody.

Deploymenty tedy tvoří ReplicaSety, které tvoří pody, a kdykoliv dojde k úpravě Deploymentu pomocí **RollingUpdate** (což je defaultní chování), je vytvořen nový ReplicaSet a dojde k přesunu podů ze starého ReplicaSetu do nového ReplicaSetu. To všechno v takzvané **Controlled Rate**, tedy tak, aby vždy běžel alespoň 1 pod a bylo nedostupných, co nejméně podů je možné (toto se dá omezovat přes **.spec.strategy.rollingUpdate.maxUnavailable**).

Tedy v případě škálování výše se stane následující:

1. Vytvořen nový ReplicaSet s hodnotou 2.
2. Vytvořen
    a. Pod 1 v novém ReplicaSetu
    b. Pod 2 v novém ReplicaSetu
3. Nový ReplicaSet je ve stavu Ready
4. Starý ReplicaSet škálován na hodnotu 0
5. Pod 1 ve starém ReplicaSetu je smazán
6. Starý ReplicaSet je smazán.

Pokud by však v bodě 3. nebyl nový ReplicaSet ve stavu Ready – tedy pokud by došlo k chybě při Deploymentu – nepostoupil by Deployment ke smazání starého ReplicaSetu.

Deployment zároveň umožňuje manuální návrat (rollback) k předchozí verzi ReplicaSetu. Přes

```
kubectl rollout history deployment.v1beta1.apps/busytest
```

zjistíme historii daného Deploymentu, najdeme bod, kde se stav pokazil, a pak jej přes

```
kubectl rollout undo deployment.v1beta1.apps/busytest --to-revision=
```

vrátíme zpět do bodu před chybou.

3. StatefulSet

StatefulSet je Kubernetes resource používaný na udržování stateful aplikací. Stará se o deploy a škálování skupiny podů a zajišťuje pořadí a unikátnost těchto podů.

Stejně jako u Deployment se jedná o kontrolér, ale na rozdíl od něj nevytváří ReplicaSet, ale místo toho pody vytváří rovnou sám a pojmenovává si je. Kupříkladu u StatefulSetu s názvem **Busytest** vznikne nejdříve pod **busytest-0** poté **busytest-1** atd.

Každá replika StatefulSetu má svůj vlastní stav a každý pod si tak vytvoří svůj vlastní **Persistent Volume Claim**. Tedy StatefulSet s 3 replikami vytvoří 3 pody, každý s jeho vlastním volume, tedy dohromady 3 PVC.

Abychom provedli deploy naší aplikace **Busytest**, použijeme následující manifest:

```
piVersion: apps/v1
kind: StatefulSet
metadata:
  name: busytest
spec:
  serviceName: "busytest-app"
  selector:
    matchLabels:
      app: busytest
  replicas: 1
  template:
    metadata:
      labels:
        app: busytest
    spec:
      containers:
      - name: busytest
        image: "k8s-registry/busybox:latest"
        volumeMounts:
        - name: busytest
          mountPath: /app/
  volumeClaimTemplates:
  - metadata:
      name: busytest
    spec:
      accessModes: [ "ReadWriteMany" ]
      storageClassName: efs
      resources:
        requests:
          storage: 50Mi
```

Na rozdíl od Deploymentu po naškálování na 3 repliky pomocí

```
kubectl scale statefulsets busytest --replicas=3
```

a naběhnutí podů a nahlédnutí do jejich logů uvidíte, že každý log začíná od řádku jedna.

Vzhledem k tomu, že StatefulSety netvoří ReplicaSety ani nic podobného, není možné rollbackovat StatefulSet do předchozí verze. Můžete pouze smazat nebo škálovat množství podů. Pokud updatujete StatefulSet, pak provedením RollingUpdate stejně jako je tomu u deploymentů. V případě selhání však nejde provést rollback, pokud se při deploymentu něco pokazí, ale deploy se zastaví hned u prvního pokaženého podu – tedy by stále nemělo dojít k výpadku celé aplikace, pouze ke snížení jejího počtu podů, dokud problém manuálně neopravíte.

4. DaemonSet

DaemonSet je kontrolér, který se stará, aby na každém nodu z clusteru běžel právě jeden pod. Tedy pokud přidáme či odstraníme node, pak DaemonSet přidá či odstraní pod. Nejběžnějšími příklady užití DaemonSetu jsou monitorovací exportéry (kupř. NodeExporter) a daemony sbírající logy (kupř. Fluentd).

Nicméně DaemonSety nepouštějí pody na nodech, které jsou označeny pomocí

```
kubectl taint nodes <node> key=<key>
```

jako nody, kde nemá dojít ke spuštění. Kupříkladu master nody. To vše samozřejmě pouze pokud pod nemá nadefinováno, aby daný taint ignoroval. Pro ukázku následuje **tained master node**:

```
taints:
  - effect: NoSchedule
    key: node-role.kubernetes.io/master
```

Pokud chceme, aby náš pod neignoroval tento node, pak je potřeba v DaemonSet manifestu přidat následující toleration, kdy budete ignorovat všechny tainty **NoSchedule**:

```
    spec:
      tolerations:
      - effect: NoSchedule
        operator: Exists
```

Abychom provedli deploy naší aplikace **Busytest,** použijeme následující manifest:

```
apiVersion: apps/v1
kind: DaemonSet
```

```
metadata:
  name: busytest-app
spec:
  selector:
    matchLabels:
      app: busytest
  template:
    metadata:
      name: busytest-app
      labels:
        app: busytest
    spec:
      tolerations:
      - effect: NoSchedule
        operator: Exists
      containers:
      - name: busytest
        image: "k8s-registry/busybox:latest"
        volumeMounts:
        - name: busytest
          mountPath: /app/
      volumes:
      - name: busytest
        persistentVolumeClaim:
          claimName: busytest
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: busytest
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 50Mi
  storageClassName: efs
```

V momentě, kdy provedete deploy tohoto DaemonSetu, vytvoří se počet podů odpovídající počtu nodů v clusteru. Vytvořený PVC se bude chovat stejně jako u Deploymentu, tedy všechny pody budou sdílet jeden PVC, a jejich logy tak budou zapisovány do jednoho stejného souboru. Pokud provedete update DaemonSetu stejně jako u předchozích typů, provede RollingUpdate, ale zároveň stejně jako u StatefulSetu nemá ReplicaSet tedy nejde provádět rollback.