Kubernetes Workshop part 2

Wojciech Barczynski (wbarczynski.pro@gmail.com)

Contents

1 Prerequiments **2**

1.1 How to install 2

1.2 Verify the setup 2

2 Statefulsets **3**

3 Kubernetes Persistent Volumes **5**

4 Daemonset **7**

5 Exploring Namespace kube-system **7**

1 Prerequiments

You need to feel good with Command Line Interface. You should understand what Docker is.

- Workstation with Linux or OSX recommended.
- Software
  - k3s
  - Kubernetes CLI
  - Docker
  - minikube
- Tools
  - jq (stedolan.github.io/jq/)
- Good to have
  - hub.docker.com account or alternative docker repository

1.1 How to install

- K3S - github.com/k3s-io/k3s
- Kubernes CLI - kubernetes.io/docs/tasks/tools/
- minikube - minikube.sigs.k8s.io/docs/start/

1.2 Verify the setup

```
$ k3d cluster create --port "8080:8080@loadbalancer" \
                     --port "8000:80@loadbalancer" \
                     'k8s-w10i-workshop'
```

```
$ kubectl config use-context k3d-k8s-w10i-workshop

$ kubectl cluster-info

Kubernetes control plane is running at https://0.0.0.0:60602
CoreDNS is running at https://...
Metrics-server is running at https://...
```

2 Statefulsets

What if we want to have a database on Kubernetes? Maybe we would like to have deterministic names. Statefulsets comes to rescue:

1. Simple DB:

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: intro-db
  labels:
    app_deploy: intro-db
spec:
  replicas: 1
  selector:
    matchLabels:
      app: intro-db
  serviceName: "intro-db"
  template:
    metadata:
      labels:
        app: intro-db
    spec:
      containers:
      - name: db
        image: wojciech11/api-status:1.0.0
        env:
          - name: DB_NAME
            value: user
        ports:
        - containerPort: 80
```

Note down what happens after:

```
$ kubectl scale --replicas=2 statefulset  intro-db
```

2. What is a statefulset without a Persistent Volume (PV). Let's delete the previous statefulset and get a new one;

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: intro-db
  labels:
    app_deploy: intro-db
spec:
  replicas: 1
  selector:
    matchLabels:
      app: intro-db
  serviceName: "intro-db"
  template:
    metadata:
      labels:
        app: intro-db
    spec:
      containers:
      - name: db
        image: wojciech11/api-status:1.0.0
        env:
          - name: DB_NAME
            value: user
        ports:
        - containerPort: 80
        volumeMounts:
        - mountPath: /data
          name: intro-db-vol
      restartPolicy: Always
  volumeClaimTemplates:
  - metadata:
      name: intro-db-vol
    spec:
      accessModes:
        - ReadWriteOnce
      resources:
        requests:
          storage: 8Gi
```

Scale it up and check in particular *PV* and *PVC*.

```
$ kubectl get pv
$ kubectl get pvc
```

3 Kubernetes Persistent Volumes

A persistence storage that survives your pod being deleted.

1. Storage class

```
$ kubectl get storageclasses

NAME                 PROVISIONER               AGE
local-path (default)   rancher.io/local-path   3d
```

```
kubectl describe storageclasses local-path

NAME                 PROVISIONER               AGE
local-path (default)   rancher.io/local-path   3d
```

2. Persistence claim and Persistence volume

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: app-intro-vol
spec:
  accessModes:
  - ReadWriteOnce
  capacity:
    storage: 2Gi
  hostPath:
    path: /data/pv0001/
```

```yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: app-intro-pvc
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: ""
  volumeName: app-intro-vol
  resources:
    requests:
      storage: 1Gi
```

3. Let's use it:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: intro-app-pv-deploy
  labels:
    app_deploy: intro-app-pv
spec:
  replicas: 1
  selector:
    matchLabels:
      app: intro-app-pv
  template:
    metadata:
      labels:
        app: intro-app-pv
    spec:
      containers:
      - name: app
        image: wojciech11/api-status:1.0.0
        env:
          - name: DB_NAME
          value: user
        ports:
        - containerPort: 80
        volumeMounts:
        - mountPath: /data
          name: app-data
      volumes:
      - name: app-data
        persistentVolumeClaim:
          claimName: app-intro-pvc
```

4. Find where the mount point is on the host and create there file. Notice: *minikube ssh*

5. Find the file on the pod with mounted volume.

4 Daemonset

Why are good use cases for Daemonset?

- ingress controller
- fluentbit

5 Exploring Namespace kube-system

Let's look around what we have here. To be able to see how the Kubernetes looks like, we need to use minikube.

1. Start your minikube:

```
$ minikube start
$ minikube status

$ kubectl config use-context minikube
$ kubectl get nodes
```

2. Get the list of pods in namespace kube-sytem:

```
$ kubectl get po -n=kube-system
```

Use `kubectl describe po <pod-name> --namespace=kube-system` to find what the version is of:

- kube-proxy: . . .
- apiserver: . . .
- coredns: . . .

3. Get the list of services:

```
$ kubectl get svc --namespace=kube-system
```

Use `kubernetes describe svc <svc-name> --namespace=kube-system` to find the endpoints for:

- kube-dns: . . .

4. Logs:

```
$ kubectl logs coredns-c4c -n=kube-system
$ kubectl logs coredns-c4c -n=kube-system -f
$ kubectl logs coredns-c4c -n=kube-system --tail=10
```

Please display logs of:
`kube-apiserver`, `kube-proxy`, `kube-scheduler`, and `etcd-minikube`.
Later, we will also cover events:
`kubectl get events -n=kube-system`.

5. Kubernetes Dashboard:

```
# on normal deployment:
# $ kubernetes proxy
$ minikube dashboard
```

6. Basic metrics:

```
$ minikube addons enable metrics-server

# wait 5 seconds
$ kubectl top nodes
$ kubectl top pods -n kube-system
```

7.