# Solution M4: Storage and Persistence

For this challenge we will assume that we are working as the **root** user on the control plane node of a three node **Kubernetes** cluster. The pod network plugin is of no importance, so it can be any of the available. We will execute all tasks in order but keep in mind that #2 expects that all artefacts from the #1 have been deleted

# Task 1

**Challenge:**

Configuration maps and secrets

* 1. Create a **ConfigMap** resource **hwcm** that:
     1. has two key-value pairs (**k8sver** and **k8sos**) initialized as literals that hold your **Kubernetes version** and the name of the **OS** where **Kubernetes** is running
     2. has two more key-value pairs (**main.conf** and **port.conf**) initialized from files. The first one (**main.conf**) should contain:

**# main.conf**

**name=homework**

**path=/tmp**

**certs=/secret**

And the second one (**port.conf**):

**8080**

* 1. Create a **Secret** resource **hwsec** that:
     1. Has two data entries – **main.key** and **main.crt** created from files
     2. The content for the above two generate by using the **openssl** utility. For example:

**openssl genrsa -out main.key 4096**

**openssl req -new -x509 -key main.key -out main.crt -days 365 -subj /CN=www.hw.lab**

* 1. Mount the above resources to a pod created from the **shekeriev/k8s-environ** image (used during the practice) by
     1. **k8sver** and **k8sos** should be mounted as environment variables with prefix **HW\_**
     2. **main.conf** should be mounted as a volume to the **/config** folder inside the container
     3. **port.conf** should be mounted as an environment variable **HW\_PORT**
     4. **main.key** and **main.crt** should be mounted as a volume to the **/secret** folder inside the container

**Solution:**

Before we create the configuration map, we must prepare the two files – **main.conf** and **port.conf**

Once, we have them, we can create the map with

**kubectl create configmap hwcm --from-literal=k8sver='1.21.6' --from-literal=k8sos='Debian 10' --from-file=main.conf=main.conf --from-file=port.conf=port.conf**

*Please note that yours may be different due to the version of Kubernetes or the OS you are using*

Again, we must prepare the key and the certificate first. Luckily, we are provided with the commands to do it

Once, we have them, we should create the secret with

**kubectl create secret generic hwsec --from-file=main.key=main.key --from-file=main.crt=main.crt**

To use the as requested, we can prepare a manifest (**pod-svc.yaml**) with the following content

apiVersion: v1

kind: Pod

metadata:

  name: pod-hw

  labels:

    app: environ

spec:

  containers:

  - image: shekeriev/k8s-environ

    name: cont-hw

    env:

    - name: HW\_K8SVER

      valueFrom:

        configMapKeyRef:

          name: hwcm

          key: k8sver

    - name: HW\_K8SOS

      valueFrom:

        configMapKeyRef:

          name: hwcm

          key: k8sos

    - name: HW\_PORT

      valueFrom:

        configMapKeyRef:

          name: hwcm

          key: port.conf

    volumeMounts:

    - name: config

      mountPath: /config

    - name: secret

      mountPath: /secret

  volumes:

  - name: config

    configMap:

      name: hwcm

      items:

      - key: main.conf

        path: main.conf

  - name: secret

    secret:

      secretName: hwsec

---

apiVersion: v1

kind: Service

metadata:

  name: svc-hw

spec:

  type: NodePort

  ports:

  - port: 80

    nodePort: 30001

    protocol: TCP

  selector:

    app: environ

Deploy it with

**kubectl apply -f pod-svc.yaml**

And open a browser tab and check the result

You must see the contents of the configuration map and the secret shown amongst the other information

We are done here 😊

# Task 2

**Challenge:**

1. Create and run a set of manifest files to spin the following application:

Chart, waterfall chart

Description automatically generated

Please note that:

* Service FE should be of type **NodePort**
* **Pod FE** should use **shekeriev/k8s-facts-fe** image and should be initialized with two environment variables – **FACTS\_SERVER** equal to the **name** of the **Service BE** and **FACTS\_PORT** equal to the **port** of **Service BE**
* **Pod FE** listens on port **5000/tcp**
* **Service BE** should be of type **ClusterIP** *(please note, that this is not the headless service but the “public” one)*
* **Pod BE** should use **shekeriev/k8s-facts** image and expects a volume to be mounted at **/data** folder
* **Pod BE** listens on port **5000/tcp**
* For the **PVs** and **PVCs** use **NFS** and storage capacity of **2Gi**
* Both the **Deployment** and the **StatefulSet** should spin three replicas

**Solution:**

First, we must take care of the storage part

We assume that there is a **NFS** server available and reachable by the name **nfs-server** from all the nodes

We also assume that there are three exported and writable folders - **/data/nfs/k8spv{a,b,c}**

Then, we should prepare three manifests – one for every persistent volume

The first (**pvssa.yaml**) may have the following content

apiVersion: v1

kind: PersistentVolume

metadata:

  name: pvssa

  labels:

    purpose: hw4

spec:

  capacity:

    storage: 2Gi

  volumeMode: Filesystem

  accessModes:

    - ReadWriteOnce

  persistentVolumeReclaimPolicy: Recycle

  mountOptions:

    - nfsvers=4.1

  nfs:

    path: /data/nfs/k8spva

    server: nfs-server

The other two (**pvssa.yaml** and **pvssa.yaml**) are similar but the name and the path are different

Then, we deploy them with

**kubectl apply -f pvssa.yaml**

**kubectl apply -f pvssb.yaml**

**kubectl apply -f pvssc.yaml**

Next, we must prepare the stateful set manifest (**ss.yaml**) file with the following content

apiVersion: apps/v1

kind: StatefulSet

metadata:

  name: facts

spec:

  selector:

    matchLabels:

      app: facts

  serviceName: facts

  replicas: 3

  # POD template

  template:

    metadata:

      labels:

        app: facts

    spec:

      terminationGracePeriodSeconds: 10

      containers:

      - name: main

        image: shekeriev/k8s-facts

        ports:

        - name: app

          containerPort: 5000

        volumeMounts:

        - name: facts-data

          mountPath: /data

  # VolumeClaim template

  volumeClaimTemplates:

  - metadata:

      name: facts-data

    spec:

      accessModes: [ "ReadWriteOnce" ]

      resources:

        requests:

          storage: 2Gi

And then deploy it with

**kubectl apply -f ss.yaml**

Then, we can set the headless service. Prepare a manifest file (**svcss.yaml**) with the following content

apiVersion: v1

kind: Service

metadata:

  name: facts

spec:

  selector:

    app: facts

  clusterIP: None

  ports:

  - port: 5000

    protocol: TCP

And deploy it with

**kubectl apply -f svcss.yaml**

Next step is to create the service, that will be used by the front-end part of the application

Create a service manifest file (**svcsscip.yaml**) with the following content

apiVersion: v1

kind: Service

metadata:

  name: factscip

spec:

  selector:

    app: facts

  type: ClusterIP

  ports:

  - port: 5000

    protocol: TCP

Deploy it with

**kubectl apply -f svcsscip.yaml**

Only two components remain – the deployment and the front-end service

For the deployment, we can create a manifest file (**fe-deployment.yaml**) with the following content

apiVersion: apps/v1

kind: Deployment

metadata:

  name: consumer-deploy

spec:

  replicas: 3

  selector:

    matchLabels:

      app: fun-facts

      role: consumer

  minReadySeconds: 15

  strategy:

    type: RollingUpdate

    rollingUpdate:

      maxUnavailable: 1

      maxSurge: 1

  template:

    metadata:

      labels:

        app: fun-facts

        role: consumer

    spec:

      containers:

      - name: cons-container

        image: shekeriev/k8s-facts-fe

        ports:

        - containerPort: 5000

        env:

        - name: FACTS\_SERVER

          value: "factscip"

        - name: FACTS\_PORT

          value: "5000"

And deploy it with

**kubectl apply -f fe-deployment.yaml**

Last, but not least, we must create a service manifest file (**fe-service.yaml**) with the following content

apiVersion: v1

kind: Service

metadata:

  name: consumer

  labels:

    app: fun-facts

    role: consumer

spec:

  type: NodePort

  ports:

  - port: 5000

    nodePort: 30001

    protocol: TCP

  selector:

    app: fun-facts

    role: consumer

Deploy this one as usual with

**kubectl apply -f fe-service.yaml**

Check the myriad of object we crated so far with

**kubectl get pod,statefulset,pv,pvc,svc,deployment,rs**

All should be up and running

Now, open a browser tab and navigate to **http://<cluster-ip>:30001**

Our application should be working as expected

Congrats! 😊