# Solution M1: Core Concepts

# Task 1

**Challenge:**

Execute a series of commands to

1. Create a namespace named **homework**
2. Create a **homework-1** pod in that namespace that uses this (**shekeriev/k8s-oracle**) image and set two labels on it (all at once) - **app=hw** and **tier=gold**
3. Remove the **tier=gold** label with a separate command
4. Create a **homework-2** pod in that namespace that uses this (**shekeriev/k8s-oracle**) image
5. Add a label to the second pod **app=hw** with a separate command
6. Set annotation on both pods with the following content **purpose=homework**
7. Create a **homerwork-svc** service to expose both pods on port **32000** of every node in the cluster

**Solution:**

We assume that all we are working on a machine with **minikube** installed and running

Open a terminal session and make sure that your cluster is ready and reachable

1. To create the required namespace execute

**kubectl create namespace homework**

1. Then create the first pod with

**kubectl run homework-1 --image=shekeriev/k8s-oracle --labels=app=hw,tier=gold --namespace homework**

1. We can remove the extra label with

**kubectl label pods homework-1 tier- --namespace homework**

1. Create the second pod

**kubectl run homework-2 --image=shekeriev/k8s-oracle --namespace homework**

1. We can add label at any time with

**kubectl label pods homework-2 app=hw --namespace homework**

1. Annotations are handled with a separate command. This one, we can execute it in two steps

**kubectl annotate pods homework-1 purpose=homework --namespace homework**

**kubectl annotate pods homework-2 purpose=homework --namespace homework**

Or in just one (by using selectors)

**kubectl annotate pods purpose=homework --namespace homework --selector=app=hw**

1. Creating a service can be achieved both via the **expose** and the **create** commands. However, they have different set of parameters. For this task, we must use the **create** command

**kubectl create service nodeport homework-svc --node-port=32000 --tcp=5000:5000 --namespace homework**

If we try now to access the application, we will notice that it won’t show. Should we want to know why, we can **describe** the service and notice that no pods are being associated (the **endpoints** list is empty). We can handle this by changing the automatically created selector (**app=homework-svc**) of the service with something more suitable (**app=hw**) that will select the two pods

**kubectl set selector service homework-svc app=hw --namespace homework**

Now, we can ask with

**minikube service list**

And using the correct URL open a browser tab and navigate to it. Our application should be there

Once done, we can easily clean up by deleting the namespace and all nested resources with

**kubectl delete namespace homework**

# Task 2

**Challenge:**

Create manifests for every object (the namespace, the two pods, and the service) from **task 1** and **apply** them one by one

**Solution:**

We assume that all objects created earlier are removed

We must create the following four files (you can check the files in **task2** folder). There are many ways to accomplish this. These include at least the following - writing them from scratch, **describe** the resources from **task 1** with output in **YAML** or use a combination of **run** and **create** commands in **dry-run** mode with output in **YAML**

In any case, the files may look like the ones shown bellow

One for the namespace (**hw-ns.yaml**) with the following minimal content

apiVersion: v1

kind: Namespace

metadata:

  name: homework

A set of two files for the pods. The first (**hw-pod-1.yaml**) with the following content

apiVersion: v1

kind: Pod

metadata:

  annotations:

    purpose: homework

  labels:

    app: hw

  name: homework-1

  namespace: homework

spec:

  containers:

  - image: shekeriev/k8s-oracle

    name: homework-1

And the second (**hw-pod-2.yaml**) with the following content

apiVersion: v1

kind: Pod

metadata:

  annotations:

    purpose: homework

  labels:

    app: hw

  name: homework-2

  namespace: homework

spec:

  containers:

  - image: shekeriev/k8s-oracle

    name: homework-2

Finally, the file (**hw-svc.yaml**) for the service

apiVersion: v1

kind: Service

metadata:

  labels:

    app: hw

  name: homework-svc

  namespace: homework

spec:

  ports:

  - name: 5000-5000

    nodePort: 32000

    port: 5000

    protocol: TCP

    targetPort: 5000

  selector:

    app: hw

  type: NodePort

Now, that we have them, we can (only if the resources from **task 1** are **deleted**) create the set again

**kubectl apply -f hw-ns.yaml**

**kubectl apply -f hw-pod-1.yaml**

**kubectl apply -f hw-pod-2.yaml**

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

Now, we can ask with

**minikube service list**

And using the correct URL open a browser tab and navigate to it. Our application should be there

Once done, we can easily clean up by deleting the namespace and all nested resources with

**kubectl delete namespace homework**

# Task 3

**Challenge:**

Is there a way to submit those manifests at once? Find and demonstrate **two ways** of doing it

**Solution:**

We assume that all objects created earlier are removed

First option is to recursively apply all the manifests

If we have them in folder **task2** we can do it by executing the following command (under **Windows**)

**kubectl apply -R -f .\task2\**

Or the following under **UNIX**-like OSes

**kubectl apply -R -f task2/**

Alternatively, we can create one big manifest file that contains all four files (check the **homework.yaml** file in the **task3** folder) and use it as we did so far

# Task 4

**Challenge:**

Try to translate the attached **docker-compose.yml** file to a set of **Kubernetes** objects and the corresponding manifest(s)

**Solution:**

It would be good to have access to a **Docker** instance to test the file and see how both containers work. If not, we can refer to the container’s descriptions in **Docker Hub**

Additionally, we may refer to the producer-consumer set of manifests used during the practice

For the **speaker** part we must create two manifests. One named **speaker-pod.yaml** with the following content

apiVersion: v1

kind: Pod

metadata:

  labels:

    app: hw

    role: speaker

  name: speaker

spec:

  containers:

  - image: shekeriev/k8s-speaker

    name: speaker

And one for the service (**speaker-svc.yaml**) with the following content

apiVersion: v1

kind: Service

metadata:

  labels:

    app: hw

  name: speaker

spec:

  ports:

  - name: speaker-port

    port: 5000

    protocol: TCP

    targetPort: 5000

  selector:

    app: hw

    role: speaker

For the **listener** part, again one for the pod (**listener-pod.yaml**)

apiVersion: v1

kind: Pod

metadata:

  labels:

    app: hw

    role: listener

  name: listener

spec:

  containers:

  - image: shekeriev/k8s-listener

    name: listener

And one for the service (**listener-svc.yaml**)

apiVersion: v1

kind: Service

metadata:

  labels:

    app: hw

  name: listener

spec:

  ports:

  - name: listener-port

    nodePort: 32000

    port: 5000

    protocol: TCP

    targetPort: 5000

  selector:

    app: hw

    role: listener

  type: NodePort

You can find the above files in the **task4** folder

Of course, we can create set of two files (one for every part) or just one big manifest file

# Task 5

**Challenge:**

Give the **KIND** tool a try and spin up a **Kubernetes** cluster in a **Docker** instance. Once done, start the pod (task 1.b) and the service (task 1.g) either imperatively or with manifests (for example, the ones from task 2)

**Solution:**

We assume that we have access to a **Docker** instance

First, we must consult the official documentation of **KIND** here: <https://kind.sigs.k8s.io/docs/user/quick-start/>

Then, based on the documentation, and our needs – to expose a service on particular node port (**32000**), we must prepare a configuration file (**kind-cluster.yaml**) with the following content

kind: Cluster

apiVersion: kind.x-k8s.io/v1alpha4

nodes:

- role: control-plane

  extraPortMappings:

  - containerPort: 32000

    hostPort: 8080

Of course, we must be sure that port **8080** is available. If not, we must change the above configuration. This will be the place, where we will expect to see the service. We need this double redirection because our cluster is running in **Docker**

Next, we can create the cluster with

**kind create cluster --name hw --config kind-cluster.yaml**

Then, using the manifests from **task 2**, we can create the required objects

**kubectl apply -f task2/hw-ns.yaml**

**kubectl apply -f task2/hw-pod-1.yaml**

**kubectl apply -f task2/hw-svc.yaml**

Then, we must be able to access the application with

**curl** [**http://localhost:8080**](http://localhost:8080)

The above should be executed on the **Docker** host

Once done, we can remove the cluster together with all objects with

**kind delete cluster --name hw**

You can find the cluster configuration file in the **task5** folder