Assigning a Kubernetes Pod to a Specific Node

You are working for BeeBox, a company that provides regular shipments of bees to customers. The company has a few pods running in their Kubernetes cluster that depend on special services that exist outside the cluster. These services are highly sensitive, and the security team has asked that they be exposed only to certain network segments.

Unfortunately, only the k8s-worker2 node exists in the network segment shared by these services. This means only pods on the k8s-worker2 node will be able to access these sensitive external services, and pods on the k8s-worker1 or k8s-control nodes cannot access them.

Your task is to reconfigure the auth-gateway pod and the auth-data deployment's replica pods so they will always run on the k8s-worker2 node.

Our first task is to get that auth-gateway pod configured so that it will only run on Worker2 node. We are going to use a nodeSelector on the pod, which is going to filter the nodes based on node labels.

1. Add a label to our k8s-worker2 node

kubectl label nodes k8s-worker2 external-auth-services=true

Figure 1-1

```
cloud_user@k8s-control:~$ kubectl get nodes
NAME          STATUS   ROLES           AGE   VERSION
k8s-control   Ready    control-plane   31m   v1.24.0
k8s-worker1   Ready    <none>          30m   v1.24.0
k8s-worker2   Ready    <none>          30m   v1.24.0
cloud_user@k8s-control:~$ kubectl label nodes k8s-worker2 external-auth-services=true
node/k8s-worker2 labeled
cloud_user@k8s-control:~$
```

Above you can see we added the kubectl label nodes command. And because worker2 has access to those external services, we need to create a label here called external-auth-services=true

Now that we added that label, we need to take a look at the BeeBox pod and showing all of the relevant objects for this lab are in the beebox-auth namespace

2. Look at the beebox pod/namespace

kubectl get pods -n beebox-auth -o wide

Figure 1-2

```
cloud_user@k8s-control:~$ kubectl  get pods -n beebox-auth -o wide
NAME                          READY   STATUS    RESTARTS   AGE   IP               NODE
auth-data-65b88b9d94-fd6bb    1/1     Running   0          41m   192.168.194.70   k8s-worker1
auth-data-65b88b9d94-mrkgp    1/1     Running   0          41m   192.168.194.68   k8s-worker1
auth-data-65b88b9d94-rz7k8    1/1     Running   0          41m   192.168.194.66   k8s-worker1
auth-gateway                  1/1     Running   0          41m   192.168.194.71   k8s-worker1
```

As you can see above the auth-gateway is running, but we want it to always run on worker2 not worker 1.

So in this lab we would have to go in the .yml file and edit the descriptor for that auth-gateway pod

3. Add a nodeSelector to the pod template in the deployment spec

Figure 1-3

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: auth-gateway
  namespace: beebox-auth
spec:
  nodeSelector:
    external-auth-services: "true"
  containers:
  - name: nginx
    image: nginx:1.19.1
    ports:
    - containerPort: 80
~
```

So we added the nodeSelector under spec and also it is important to put “true” in double quotes so that it is not interpreted as a Boolean.

Once you input those 2 save and exit

Normally after doing things in yml I remember doing the **kubectl create -f** command. But we need to actually delete the pod

4. Delete the auth-gateway pod

kubectl delete pod auth-gateway -n beebox-auth

5. Create/Recreate the pod

kubectl create -f auth-gateway.yml

So now we did the changes for that pod so go back to the command with “-o wide” (Step 2)

Figure 1-4

```
NAME                          READY   STATUS              RESTARTS   AGE   IP               NODE
auth-data-65b88b9d94-fd6bb    1/1     Running             0          52m   192.168.194.70   k8s-worker1
auth-data-65b88b9d94-mrkgp    1/1     Running             0          52m   192.168.194.68   k8s-worker1
auth-data-65b88b9d94-rz7k8    1/1     Running             0          52m   192.168.194.66   k8s-worker1
auth-gateway                  0/1     ContainerCreating   0          15s   <none>           k8s-worker2
```

So now you can see the auth-gateway is pointing to the worker2 node. And will be guaranteed to run on that node.

Since that is completed our next step is to do the same exact thing to our deployment descriptor.

Figure 1-5

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: auth-data
  namespace: beebox-auth
spec:
  replicas: 3
  selector:
    matchLabels:
      app: auth-data
  template:
    metadata:
      labels:
        app: auth-data
    spec:
      nodeSelector:
        external-auth-services: "true"
      containers:
      - name: nginx
        image: nginx:1.19.1
        ports:
        - containerPort: 80
```

As you can see above I made sure not to put it in the actually pod spec down below rather than the top portion. Hence the x and the arrow in the right place.

6. Since this is a deployment, we do not need to delete and recreate. We apply it

kubectl apply -f auth-data.yml

After applying this, that will go ahead and update our deployment as well as rolling out new pods to replace all the replica pods with the new configuration.

Figure 1-6

```
cloud_user@k8s-control:~$ kubectl  get pods -n beebox-auth -o wide
NAME                          READY   STATUS    RESTARTS   AGE   IP              NODE
auth-data-5b476c4fc6-c7gkb    1/1     Running   0          91s   192.168.126.4   k8s-worker2
auth-data-5b476c4fc6-ttdnl    1/1     Running   0          94s   192.168.126.2   k8s-worker2
auth-data-5b476c4fc6-wl7wt    1/1     Running   0          92s   192.168.126.3   k8s-worker2
auth-gateway                  1/1     Running   0          17m   192.168.126.1   k8s-worker2
```

As you see above, you can see now that all the pods, both auth-gateway and our deployment replicas are all running on k8s-worker 2 as desired.

So we have completed this task by reconfiguring our pods as well as our deployment to only run pods on that second worker node.