Kubernetes Training

Lab Guide

Lab 1: Setting Up Kubernetes and explore kubectl

Training Goals Covered:

- Install Kubernetes tools and start a single-node cluster.
- Explore the *kubectl* command.

Steps:

🔥Helper: Basic controls | minikube

1. **Install Minikube & kubectl**

- Follow the official installation guide for Minikube and kubectl.
- The `kubectl` version should be within one minor version difference of the Kubernetes server version.

```
Shell
minikube version
minikube version: v1.37.0
```

2. **Start Minikube using Multi-Node Cluster (One Control-Plane, One Worker)**

```
Shell
minikube start --nodes=2
```

3. **Check Minikube status**

```
Shell
minikube status

minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured

minikube-m02
type: Worker
host: Running
```

```
kubelet: Running
```

4. Install metrics server and dashboard addons on minikube.

```Shell
minikube addons enable metrics-server
minikube addons enable dashboard
```

```Shell
minikube addons list | grep enabled
| dashboard                   | minikube | enabled ✅ | Kubernetes
| metrics-server              | minikube | enabled ✅ | Kubernetes
```

🔥 **Productivity hacks:**

Alias
Save time and effort by setting up an alias for `kubectl`
Instead of typing the full command every time, you can create a shortcut to speed up your workflow.

For the current session, just type this on your terminal:

```None
alias k='kubectl'
```

To set permanently, edit the .bashrc file on your home directory and add:

```None
alias k=kubectl
```

After that reload your shell to apply the changes:

```
None
source ~/.bashrc
```

Autocomplete
Enable Autocomplete for `kubectl` 🚀
Get command suggestions and autocomplete to boost your efficiency!

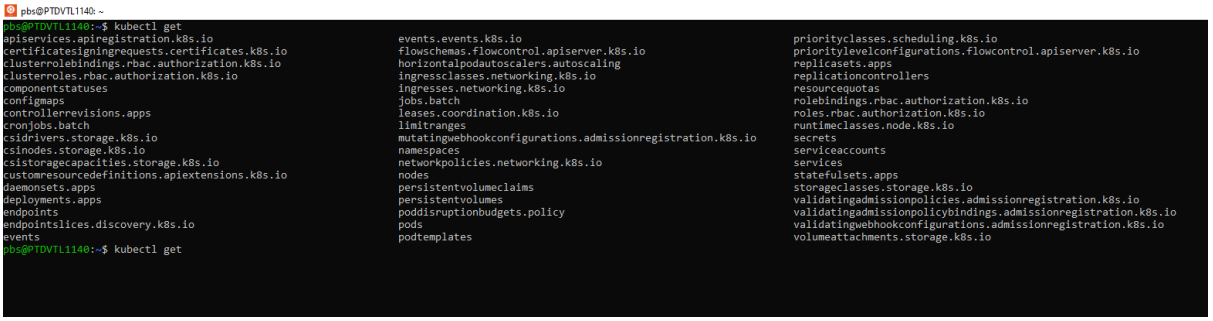
For the current session, just type this on your terminal:

```
None
source <(kubectl completion bash)
```

Now if you type:

```
None
kubectl get (press TAB KEY twice)
```

You will get something like:



Or type resource first letters + TAB KEY, like

```
None
kubectl get dep
# you will get
kubectl get deployments.apps
```

It is quite useful to remember the correct names and save some time.

Explain
Stuck on a Kubernetes concept? Need help setting up a volume in a pod? 🤔

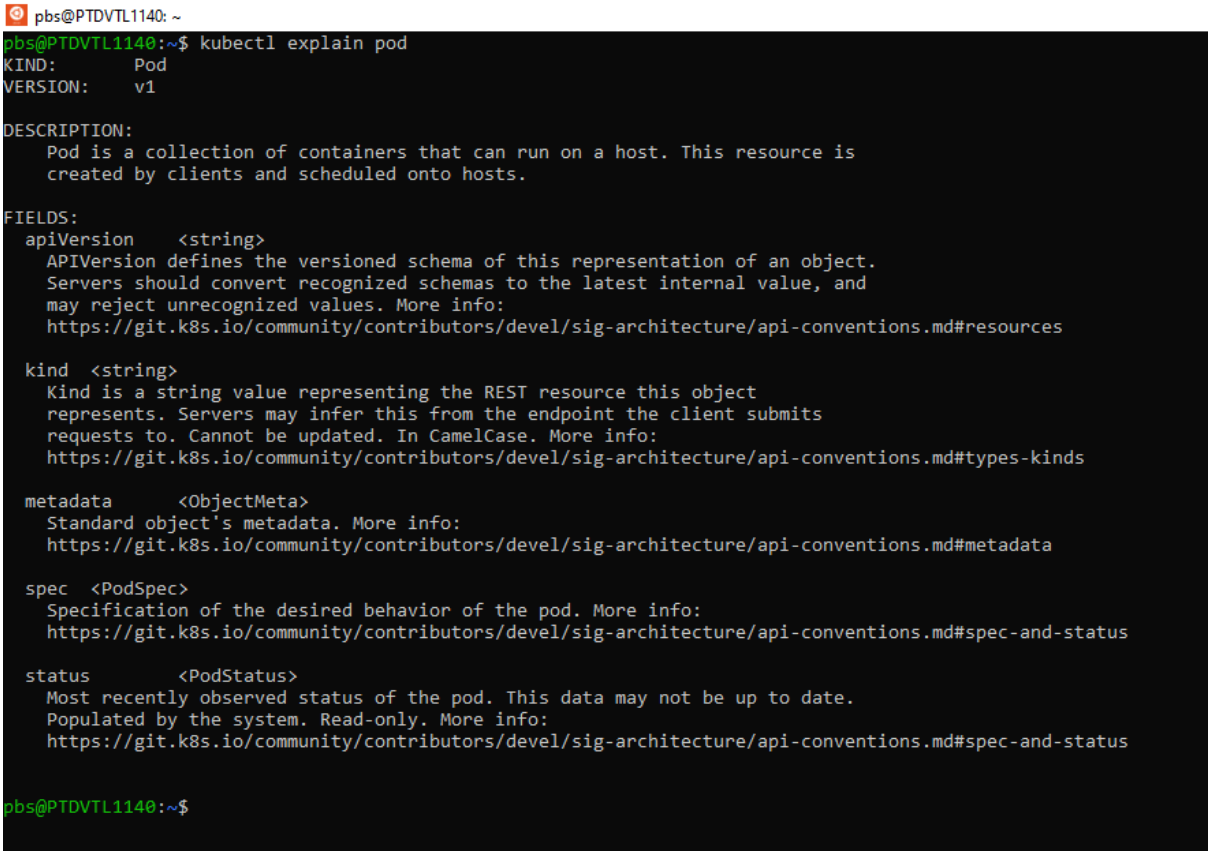
Not sure about the correct syntax?
Just use **Explain** and get clear, concise answers instantly

```
None
kubectl explain (OBJECT TO BE EXPLAINED)
```

Example, explain the main fields of a pod:

```
None
kubectl explain pod
```

The result is:



If you need more details, you can go deeper:

```
None
kubectl explain pod.{FIELD-NAME}
```

So to get the details of the pod spec

```
None
kubectl explain pod.spec
```

And deeper:

```
None
kubectl explain pod.spec.volumes.hostPath
```

Get the help you need without leaving your terminal, saving time and keeping your workflow uninterrupted.

Additional useful documentation can be found at the following URLs:

- Kubernetes Documentation
- kubectl Cheat Sheet | Kubernetes
- Kubernetes-Cheat-Sheet.pdf

Explore *kubectl* command by checking the following topics:

- Check **nodes** status.
- Check pods in **all namespaces**.
- Check detailed information about the worker node.
- Check metrics for the **worker** node.

Lab 2: Working with Pods and Namespaces

Training Goals Covered:

- Create a Namespace. Deploy a simple pod and explore its lifecycle.
- Use imperative commands and declarative configuration for it.

Steps:

- Via imperative commands:
  - Create a namespace called **demo**.
  - Deploy a simple pod as **my-app** using the nginx Docker image in the namespace demo. Include the label *app=my-app*.
  - Confirm that the pod is running.
  - Check pod details (for example, start date).
  - Check pod logs.
  - List pod in the namespace app **without** using the option “-n”.

    - Delete pod and the namespace.
- Via declarative configuration (YAML manifest)
    - Recreate the namespace.
    - Recreate pod. Include a label as **“name : my-app”** and container port as **8088**.
    - Delete pod and namespace.
- Extra
    - Create a pod yaml (NS default) from the imperative command with only essential specs (without the extra fields as UID, resource versions, status).

Lab 3 : Multi-Container Pods

Training Goals Covered:

- Explore multi-container pods.

Steps:

1. Create the multi-container as shown below (multi-container-demo.yaml)

```shell
apiVersion: v1
kind: Pod
metadata:
  name: multi-container-demo
spec:
  # Shared storage for both containers
  volumes:
  - name: log-storage
    emptyDir: {}

  containers:
  # Container 1: The "Producer" (Simulates an app writing logs)
  - name: app-container
    image: busybox
    command: ["sh", "-c", "while true; do date >> /var/log/app.log; sleep 5;
done"]
    volumeMounts:
    - name: log-storage
      mountPath: /var/log

  # Container 2: The "Sidecar" (Simulates a log processor)
  - name: sidecar-container
    image: busybox
    command: ["sh", "-c", "tail -f /var/log/app.log"]
    volumeMounts:
    - name: log-storage
```

```
      mountPath: /var/log
```

2. Check the number of containers ready.
3. Examine the logs for the `sidecar-container` to confirm they replicate the entries being written by the `app-container`, and use this to interact with the specific containers.
4. Change the multi-container-demo.yaml to force a ErrImagePull on the container sidecar-container. Check the number of containers ready.
5. Delete the resources.

Lab 4: Deployments

Training Goals Covered:

- Use Deployments to manage application scaling and updates.
- Use imperative commands.

Steps:

1. Create a deployment as **my-deploy** with **4** replicas in the default namespace using the image **nginx:1.28**.
2. Review the deployment location (nodes) of the pods.
3. Increase the number of replicas to **5**.
4. Monitor the rollout and check the revision content.
5. Describe the deployment and check the name of the container.
6. Pause the deployment rollout.
7. Change the image of the deployment to **nginx:1.29**.
8. Confirm that the pods of the deployment are running under the new version.
9. Rollback the deployment to the first revision.
10. Delete the resources.

Lab 5 : Services

Training Goals Covered:

- Expose applications within and outside the cluster.
- Use declarative configuration (YAML files).

Steps:

1. Create a Deployment with 2 replicas as **my-deploy** using the image **paulbouwer/hello-kubernetes:1.10** on port **8080**.
2. Create a ClusterIP service to internally expose your deployment on port **80**.
3. Deploy a troubleshooting pod using the image **nicolaka/netshoo**t to check the connectivity to the service.

🔥nicolaka/netshoot - Docker Image

```shell
kubectl run tmp-shell --rm -i --tty --image nicolaka/netshoot -- /bin/bash
```

4. Expose the Deployment externally using **NodePort** on port **30001**.
5. Access the NodePort service using the minikube command for it.

```shell
minikube service <service_name> --url
```

6. Delete the resources.

Lab 6 : Init Containers

Training Goals Covered:

- Understand the lifecycle of an Init Container.
- Use an Init Container to "block" an application from starting until a dependency is met.
- Observe the Pod status transitions.

Steps

1. Create an application (pod as init-demo-pod.yaml file) that requires a specific Service to exist before it starts.

```shell
apiVersion: v1
kind: Pod
metadata:
  name: init-demo-pod
  labels:
    app: init-demo
spec:
  containers:
  - name: main-app
```

```
    image: busybox
    command: ['sh', '-c', 'echo "The app is running!" && sleep 3600']

  # This runs BEFORE the main-app
  initContainers:
  - name: wait-for-service
    image: busybox
    command: ['sh', '-c', "until nslookup
myservice.default.svc.cluster.local; do echo waiting; sleep 2; done"]
```

2. Verify the pod's status to confirm it is awaiting the service to become available.

🔥Debug Init Containers | Kubernetes

3. Create the service **my-service**.
4. Verify again the pod's status.
5. Delete the resources.

Lab 7: ConfigMaps

Training Goals Covered:

- Explore ConfigMaps.

Steps:

1. Create a ConfigMap as **web-config** with the key values **MAX_CONNECTIONS=500** and **THEME=dark**.
2. Create a pod **env-demo-pod** and inject the configmap via Environment Variables. Check the ENV value.

🔥The baseline YAML is described below (not complete).

Shell

```
apiVersion: v1
kind: Pod
metadata:
  name: env-demo-pod
spec:
  containers:
  - name: busybox-container
    image: busybox
    command: ["sh", "-c", "env && sleep 3600"]
```

```
    env:
      - name: MAX_CONN
        valueFrom:
          configMapKeyRef:
            name:
            key:
```

3. Change the configmap by updating the MAX_CONNECTIONS from 500 to 100.
4. Check that the ENV value for MAX_CONNECTIONS is changed.
5. Create a pod as **volume-demo-pod** and mount ConfigMap as a volume into **/etc/config**.

🔥The baseline YAML is described below (not complete).

Shell

```
apiVersion: v1
kind: Pod
metadata:
  name: volume-demo-pod
spec:
  volumes:
    - name: config-vol
      configMap:
        name:
  containers:
  - name: busybox-container
    image: busybox
    command: ["sh", "-c", "ls /etc/config && sleep 3600"]
    volumeMounts:
      - name: config-vol
        mountPath:
        readOnly: true
```

6. Check the value of MAX_CONNECTIONS.
7. Change the configmap by updating the MAX_CONNECTIONS from 100 to 300.
8. Check the value of MAX_CONNECTIONS again.
9. Delete the resources.

Lab 8: Secrets

Training Goals Covered:

- Explore secrets.

Steps:

1. Create a secret as **app-db-secret** with the values **username=admin** and **password=P@ssw0rd123**.
2. Create a pod **secret-env-pod** and inject the secret via Environment Variables. Check the ENV value.

🔥The baseline YAML is described below (not complete).

```shell
apiVersion: v1
kind: Pod
metadata:
  name: secret-env-pod
spec:
  containers:
  - name: busybox
    image: busybox
    command: ["sh", "-c", "echo 'Password is set' && sleep 3600"]
    env:
      - name: DB_PASSWORD
        valueFrom:
          secretKeyRef:
            name:
            key:
```

3. Create a pod as **secret-vol-pod** and mount secret as a volume into **/etc/db-creds**. Check the password value inside the pod.

🔥The baseline YAML is described below (not complete).

```shell
apiVersion: v1
kind: Pod
metadata:
  name: secret-vol-pod
spec:
  volumes:
    - name: secret-vol
      secret:
```

```
      secretName:
  containers:
  - name: busybox
    image: busybox
    command: ["sh", "-c", "sleep 3600"]
    volumeMounts:
      - name: secret-vol
        mountPath:
        readOnly: true
```

4. Define a new, arbitrary value of your choice for the secret.
5. Check the value again inside the pod.
6. Delete the resources.

Lab 9: Persistent Storage

Training Goals Covered:

- Attach storage to a pod and explore data mapping.
- Use declarative configuration (YAML files).

Steps:

1. Create a Persistent Volume (2GB) mapped to the host path “/data”.

🔥Use storage class as *manual*.

2. Create a Persistent Volume Claim (1GB).
3. Deploy a Pod using the Persistent Storage claimed. Use nginx image, container port 80 and "/usr/share/nginx/html" as mount path.
4. Create a sample file with a random text in the mount path refereed.
5. Check the content of the file in the host path.

🔥minikube ssh

```
Shell
minikube ssh
```

6. Delete the pod created on step 3.
7. Recreate the pod. Check the file is accessible in the pod.

8. Delete the resources.

Lab 10 : Node Selector and Node Affinity

Training Goals Covered:

- Assign Labels to Minikube nodes.
- Use NodeSelector for simple, strict placement.
- Use Node Affinity for advanced, flexible placement rules.

Steps

1. Add a label **disktype=ssd** on node **minikube**.
2. Check the label on node minikube.

```
Shell
kubectl get nodes --show-labels
```

3. Develop a pod definition (`nodeselector-pod.yaml`) that utilizes a node selector to explicitly target the *minikube* node for placement. Verify that the pod is successfully scheduled onto the intended node.

🔥The YAML is described below (not complete).

```
Shell
apiVersion: v1
kind: Pod
metadata:
  name: ssd-pod
spec:
  containers:
  - name: busybox
    image: busybox
    command: ["sh", "-c", "echo 'I am on the SSD node'; sleep 3600"]
  nodeSelector:
    disktype:
```

4. Add a label **zone=prod** on node **minikube-m02**.
5. Check the label on node **minikube-m02**.
6. Use `prod-affinity-pod.yaml` to create a Node Affinity pod. Verify that the pod is scheduled on **minikube-m02**.

🔥The YAML is described below (not complete).

```
Shell
apiVersion: v1
kind: Pod
metadata:
  name: prod-affinity-pod
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key:
            operator: In
            values:
            -
  containers:
  - name: busybox
    image: busybox
    command: ["sh", "-c", "echo 'I am in the prod zone'; sleep 3600"]
```

7. Create a pod (impossible-pod.yaml) by including a node selector as color equal to blue.

🔥The YAML is described below (not complete).

```
Shell
apiVersion: v1
kind: Pod
metadata:
  name: impossible-pod
spec:
  containers:
  - name: busybox
    image: busybox
    command: ["sh", "-c", "echo 'Im blue ba da bee'; sleep 3600"]
  nodeSelector:
```

8. Check the pod's status to determine the reason it is in its current state.
9. Delete the resources and remove labels on nodes.

Lab 11: Resources and Limits

Training Goals Covered:

- Define requests and limits for a pod.
- Expose the pod using a ClusterIP service.
- Stress the pod and check the OOMKilled status.

Steps:

1. Create a pod (limits.yaml) with the requests (cpu: 200m, mem: 150Mi) and limits (cpu: 500m, mem: 250Mi) .

🔥The YAML is described below (not complete).

```shell
apiVersion: v1
kind: Pod
metadata:
  name: limits
  labels:
    name: limits
spec:
  containers:
  - name: app
    image: paulbouwer/hello-kubernetes:1.10
    ports:
      - containerPort: 8080
```

2. Expose the pod as service (limits-service) inside the cluster on port 80.
3. Deploy a pod with image nicolaka/netshoot as helper.

🔥[nicolaka/netshoot - Docker Image]

```shell
kubectl run tmp-shell --rm -i --tty --image nicolaka/netshoot -- /bin/bash
```

4. Stress the pod by sending continuous requests to the service using the command below. Check the status of the pod after some time.

```shell
kubectl run tmp-shell --rm -i --tty --image nicolaka/netshoot -- /bin/bash
tmp-shell:~# fortio load -qps 1000 -t 0
http://<service_ip_or_fqdn>:<service_port>
```

5. Delete the resources.

Lab 12 : Horizontal Pod Autoscaling

Training Goals Covered:

- Deploy a workload and configure Horizontal Pod Autoscaling (HPA).
- Apply stress to the deployed workload.
- Observe and analyze the resulting scaling behavior.

Steps:

1. Create a Deployment named **scaler-challenge**.
    a. Use the image as **registry.k8s.io/hpa-example** and port **80**.
    b. Define a CPU **request of 100m** and a **limit of 200m**.
    c. Expose it via a **Service** named **scaler-servic**e on port **80**.
2. Create the HPA.
    a. Metrics: Average CPU **Utilization** at **60%**.

🔥The YAML is described below (not complete).

```shell
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: scaler-challenge-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: scaler-challenge
  minReplicas: 2
  maxReplicas: 6
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type:
        averageUtilization:
```

3. Stress the workload.
    a. Open a terminal and watch the HPA.

```shell
kubectl get hpa scaler-challenge-hpa -w
```

b. Run the stress test.

```shell
kubectl run load-gen --image=busybox:1.28 --restart=Never -- /bin/sh -c
"while true; do wget -q -O- http://scaler-service; done"
```

4. Answer the questions below:
    a. Why did the Pod count immediately jump to 2 even before you started the load generator?
    b. What was the highest number of replicas reached during the stress test?
    c. After you delete the load-gen pod, how long does it take for the replicas to scale back down to 2?

5. Delete the resources.

Lab 13: Vertical Pod Autoscaling

Training Goals Covered:

- Deploy a workload which includes a stress script command.
- Configure Vertical Pod Autoscaling (VPA).
- Observe and analyze the behavior.

Pre-Requisites:

- Install and validate VPA on Minikube cluster.

```shell
git clone https://github.com/kubernetes/autoscaler.git
cd autoscaler/vertical-pod-autoscaler
```

```shell
./hack/vpa-up.sh
```

```
Shell
kubectl get pods -n kube-system | grep vpa

vpa-admission-controller-795598f856-b8qhl   1/1     Running   0
24m
vpa-recommender-5689665744-k5b6d            1/1     Running   0
25m
vpa-updater-6cf6bc7ff8-x9rld                1/1     Running   0
25m
```

Steps:

1. Configure the VPA (my-vpa) with the following:
    a. updateMode: "**Off**"
    b. Target: **deployment/hamster**

🔥The YAML is described below (not complete).

```
Shell
apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name:
spec:
  targetRef:
    apiVersion: "apps/v1"
    kind: Deployment
    name:
  updatePolicy:
    updateMode: ""
```

2. Watch the VPA.

```
Shell
kubectl get vpa my-vpa -w
```

The field **PROVIDED** appears as **False**. Why?

3. Create a deployment named **hamster** and deploy it.
    a. Image: registry.k8s.io/ubuntu-slim:0.14
    b. Resources
        i. Requests: 100m CPU and 50Mi Memory.

🔥The YAML is described below (not complete).

```shell
apiVersion: apps/v1
kind: Deployment
metadata:
  name:
spec:
  selector:
    matchLabels:
      app: hamster
  template:
    metadata:
      labels:
        app: hamster
    spec:
      containers:
        - name:
          image:
          resources:
            requests:
              cpu: ""
              memory: ""
          command: ["/bin/sh"]
          args: ["-c", "while true; do timeout 0.5s yes >/dev/null; sleep 0.5s; done"]
```

4. Check again the PROVIDED field in the VPA.

5. Understand the VPA recommendations.

ℹ️Recommendations

Property	**Meaning**	**Role in Scaling**
Target	The **optimal** resource amount the VPA thinks your pod needs.	This is the value that will actually be applied to your pod's **Requests** when it scales.
Lower Bound	The minimum "safe" amount of resources.	If your pod's *current* request falls **below** this, the VPA considers it

		under-provisioned and will trigger a scale-up.
Upper Bound	The maximum "sane" amount of resources.	If your pod's *current* request is **above** this, the VPA considers it over-provisioned (wasting resources) and will trigger a scale-down.
Uncapped Target	The "pure" recommendation without safety nets.	This is what the VPA *wants* to set if you didn't have minAllowed or maxAllowed constraints in your policy. It's for your information only.

ℹ️**When does a Pod actually restart?**
The VPA Updater (in Auto or Recreate mode) follows this logic to decide if your pod needs to be evicted:

- Trigger Rule: A pod is evicted only if its current Request is **less than the Lower Bound** OR **greater than the Upper Bound**.

6. Delete resources.

Lab 14: Vertical Pod Autoscaling

Training Goals Covered:

- Apply a Taint to a specific Minikube node.
- Observe a Pod being "repelled" by a Taint.
- Create a Pod with a Toleration to "bypass" the Taint.

Steps:

1. Add a taint to the node minikube-m02.

```shell
kubectl taint nodes minikube-m02 department=finops:NoSchedule
```

ℹ️NoSchedule → This means no new Pods will be placed here unless they have a matching toleration.

2. Deploy a pod using nodeSelector to define the node (minikube-m02) where the pod should be deployed.

```shell
apiVersion: v1
kind: Pod
metadata:
  name: pod
spec:
  containers:
  - name: busybox
    image: busybox
    command: ["sh", "-c", "sleep 3600"]
  nodeSelector:
    kubernetes.io/hostname: minikube-m02
```

3. Observe what happens to the pod.
4. Add a Toleration to the pod.

```shell
apiVersion: v1
kind: Pod
metadata:
  name: tolerated-pod
spec:
  containers:
  - name: busybox
    image: busybox
    command: ["sh", "-c", "sleep 3600"]
  tolerations:
  - key: "department"
    operator: "Equal"
    value: "finops"
    effect: "NoSchedule"
  nodeSelector:
    kubernetes.io/hostname: minikube-m02
```

5. Observe again what happened to the pod.
6. Delete the resources.

Lab 15: RBAC

Training Goals Covered:

- Define a Role and RoleBinding for pods view only.
- Explore the behaviour.

Steps:

1. Create a Service Account to represent a user. The Service Account should be defined as bob.

```
Shell
kubectl create serviceaccount bob
```

2. Create a Role defining the only actions to be performed on pods are get, list and watch.

🔥The YAML is described below (not complete).

```
Shell
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: default
  name: pod-viewer
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: []
```

3. Create a RoleBinding to bind the Service Account to the defined Role.

🔥The YAML is described below (not complete).

```
Shell
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: read-pods-bob
```

```
  namespace: default
subjects:
- kind: ServiceAccount
  name:
  namespace: default
roleRef:
  kind: Role
  name: pod-viewer
  apiGroup: rbac.authorization.k8s.io
```

4. Use the command **kubectl auth can-i** to explore the permissions defined.

```
Shell
kubectl auth can-i list pods --as=system:serviceaccount:default:bob
```

→ *Can Bob list pods?*

```
Shell
kubectl auth can-i delete pods --as=system:serviceaccount:default:bob
```

→ *Can Bob list pods?*

```
Shell
kubectl auth can-i get secrets --as=system:serviceaccount:default:bob
```

→ *Can Bob view Secrets?*

5. Delete resources.