Advanced Kubernetes

Gaining Mastery over your Deployments

Session Labs: Revision 3.0 - 07/15/23

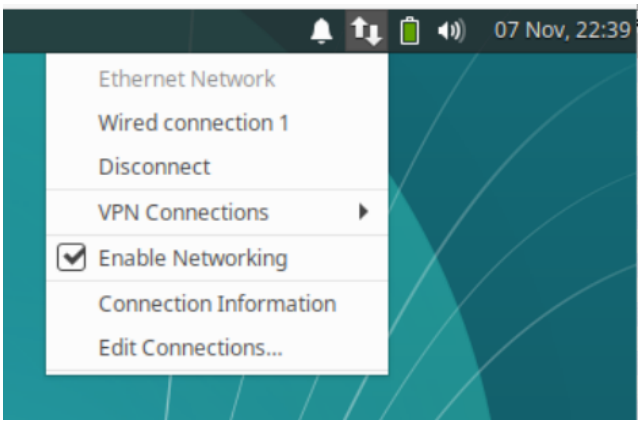
Brent Laster

Important Prereq: These labs assume you have already followed the instructions in the separate setup document http://github.com/skilldocs/adv-k8s and have either setup your own cluster and applications per those instructions or have VirtualBox up and running on your system and have downloaded the *adv-k8s-2.1.ova* file and loaded it into VirtualBox. If you have not done that, please refer to the setup document at https://github.com/skilldocs/adv-k8s/blob/main/adv-k8s-setup.pdf for the workshop and complete the steps in it before continuing!

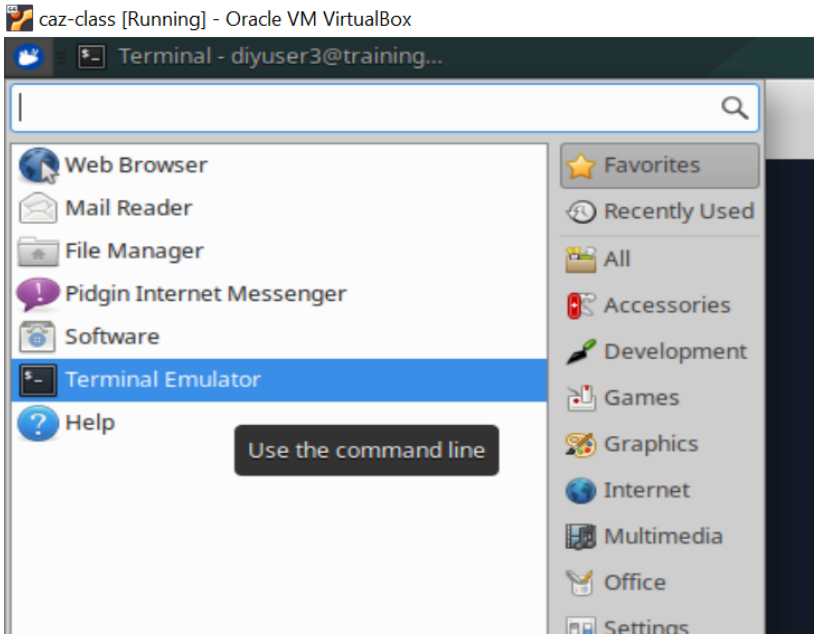
Throughout the labs, we will denote anything that is specific to the VirtualBox environment with a phrase like "**If running in the VM**".

Startup - to do before first lab

1. **If running in the VM,** make sure networking is enabled (you can ping google.com, etc.). If not, check to see if you need to enable networking by selecting the up/down arrow icon at top right and selecting the option to "Enable Networking". See screenshot below.



Open a terminal session by using the one on your desktop or clicking on the little mouse icon in the upper left corner and selecting **Terminal Emulator** from the drop-down menu.



2. Get the latest files for the class. For this course, we will be using a main directory *k8s-ps* with subdirectories under it for the various labs.

If running in the VM

In the terminal window, cd into the main directory and update the files.

```
$ cd adv-k8s
$ git stash
$ git pull
```

If NOT running in the VM (and not already done)

```
$ git clone https://github.com/skillrepos/adv-k8s
```

3. **If running in the VM**, start up the Kubernetes (minikube) instance on this system using a script in the *extras* subdirectory. This will take several minutes to run.

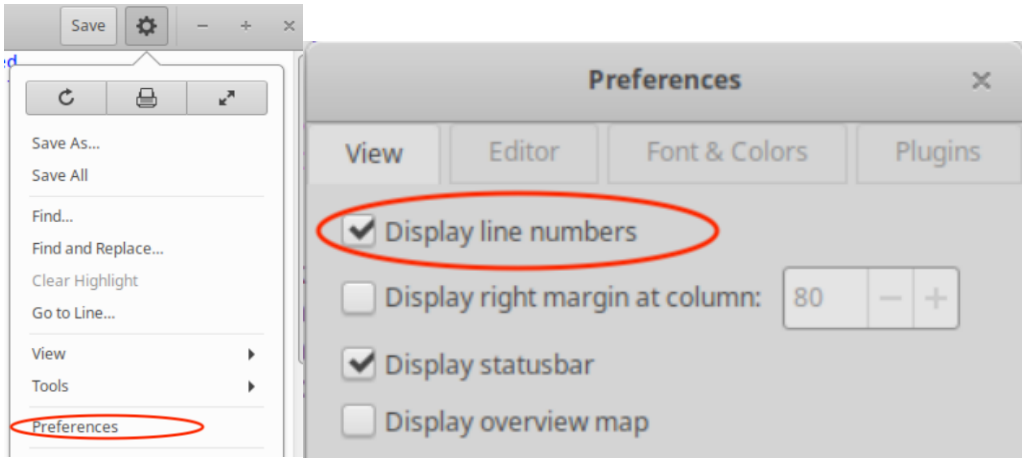
```
$ ./extra/start-kube.sh
```

4. Optional - setup alias. In these labs and on the VM, "k" is aliased to "kubectl". **If you are not running in the VM, you can usually do this via the following command if you want:**

```
$ alias k=kubectl
```

Notes about editors: If you are NOT running in the VM, substitute your editor for any occurrences of "gedit". If you are running in the VM, and need to turn on line numbers, see instructions below.

Click on the gear icon at the top right, selecting Preferences, then clicking the top box in the "View" tab to "Display line numbers". Then click the "x" in the upper right to close the dialog box.



Lab 1: Working with Kubernetes Probes

Purpose: In this lab, we'll learn how Kubernetes uses probes for determining the health of pods, how to set them up, and how to debug problems around them.

1. For this workshop, files that we need to use are contained in the directory **adv-k8s** in the home directory on the disk. Under that directory are subdirectories for each of the main topics that we will cover. For this section, we'll use files in the **roar-probes** directory. (roar is the name of the sample app we'll be working with.) Change into that directory. In the terminal emulator, enter

   ```
   $ cd ~/adv-k8s/roar-probes
   ```

2. In this directory, we have Helm charts to deploy the database and webapp parts of our application. You can use the "tree" command to see the overall structure if you are interested. Then create a namespace named "probes" to hold the deployment. Finally, deploy the app into the cluster with the "helm install" command. (In the last command, the first occurrence of "probes" indicates the namespace, the second is the name of the release, and the "." means using the files from this directory.)

   ```
   $ tree
   ```

   ```
   $ k create ns probes
   ```

   (The kubectl command is aliased to just "k" on this machine.)

```
$ helm install -n probes probes .
```

3. Now let's see how things are progressing. Take a look at the overall status of the pods, also showing the labels on the pods.

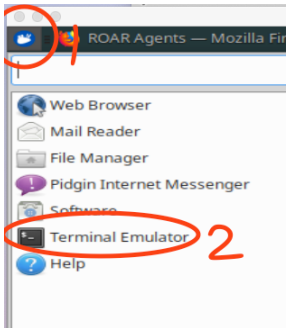
```
$ k get pods -n probes --show-labels
```

4. You should see that while the web pod is running, the database pod is not ready at all. (If the web pod isn't "Running" yet, it may still need time to startup. You can run the command again after a minute or so to give it time to finish starting up.). Now, let's do a "describe" operation on the pod itself. We'll specify the pod using one of its labels instead of having to use the pod name.

```
$ k describe -n probes pod -l app=mysql
```

5. Note the error message near the bottom of the output mentioning the readiness probe failed. The readiness probe in this case is just an exec of a command to invoke mysql. The error implies that the call to "mysql" failed. But note that it doesn't say it couldn't find it. Rather, it wasn't valid to call it that way since it tried to invoke it without a valid name and password to login.

6. You can see the YAML for this in the deployment template in the corresponding Helm chart. In a separate terminal window, take a look at that and find the section near the bottom with the **readinessProbe** spec. (The figure below shows how to open an additional terminal window.)




```
$ cat ~/adv-k8s/roar-probes/charts/roar-db/templates/deployment.yaml
```

7. We actually don't need to have a command login to verify readiness – we just need to know the mysql application responds. Let's fix this by simply calling the "version" command - which we should be able to do without a login.

8. You can choose to edit the deployment file with the "gedit" editor. Or you can use the "meld" tool to add the differences from a file that already has them. If using the meld tool, select the left arrow to add the changes from the second file into the deployment.yaml file. Then save the changes.

   Switch to ~/adv-k8s/roar-probes if not already there.

```
$ cd ~/adv-k8s/roar-probes
```

Either do: (reminder **if not running in the VM, substitute your editor for "gedit"**)

```
$ gedit charts/roar-db/templates/deployment.yaml
```

And change

```
readinessProbe:
  exec:
    command:
    - mysql
  failureThreshold: 3
  initialDelaySeconds: 5
```

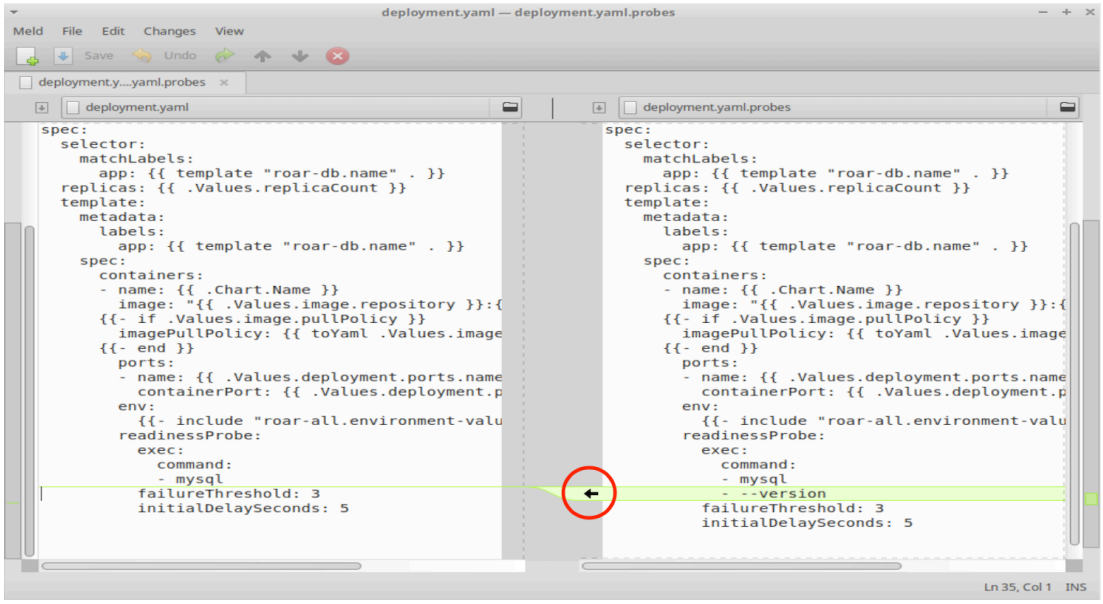
To add the line shown in bold

```
readinessProbe:
  exec:
    command:
    - mysql
    - --version
  failureThreshold: 3
  initialDelaySeconds:  5
```

Or:

```
$ meld charts/roar-db/templates/deployment.yaml ../extra/deployment.yaml.probes
```

Then click on the arrow circled in red in the figure. This will update the template file with the change. Then Save your changes and exit meld.



9. Upgrade the helm installation. After a minute or so, you can verify that you have a working mysql pod. (You may have to wait a moment and then check again.)

```
$ helm upgrade -n probes probes .
```

10. At this point, you can get the service's nodeport and then open up a browser on localhost to the URL below to see the application running. (You can use the browser shortcut on your desktop.)

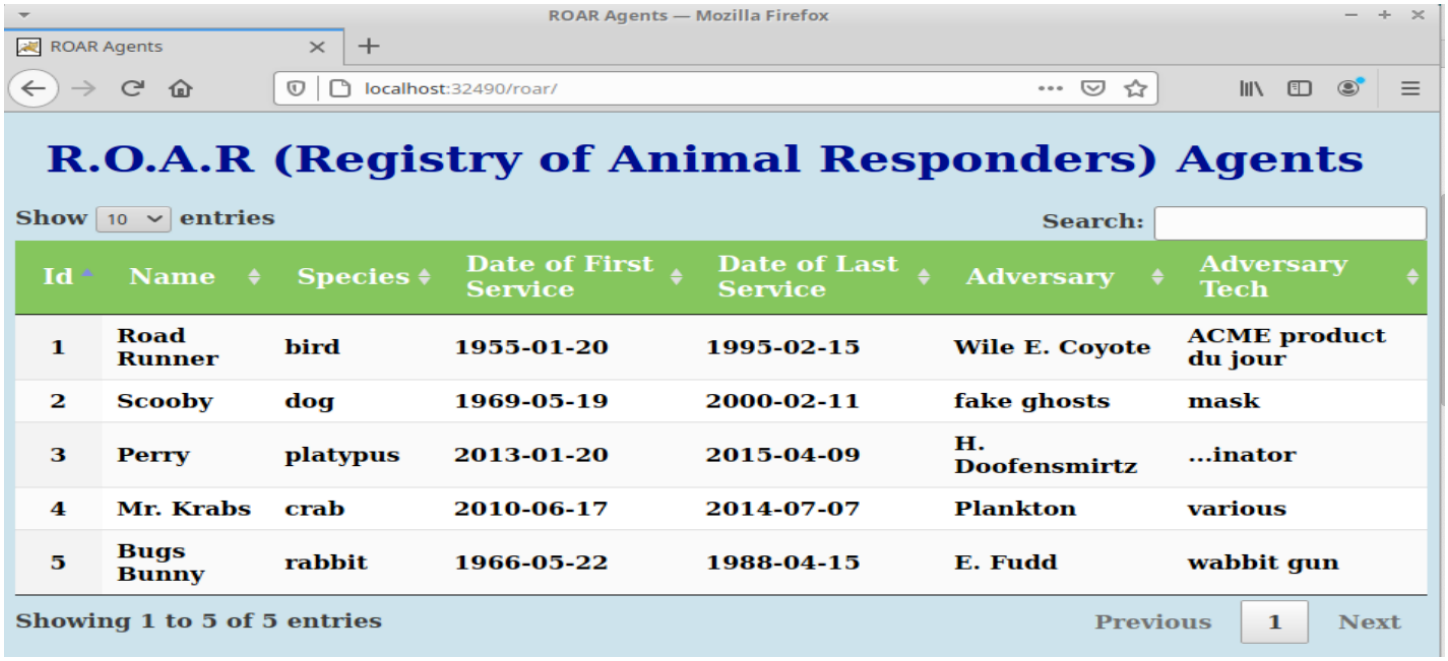
```
$ k get svc -n probes
```

Look for the port > 3000 after the 8089: in the roar-web line. Plug that value in for <port> below.

http://localhost:<port>/roar/

(Note: depending on your setup, you may need to do a port-forward command or similar first.)

```
$ k port-forward <roar-web pod name> <nodeport>:8080 http://localhost:<nodeport>/roar/
```



END OF LAB

Lab 2: Working with Quotas

Purpose: In this lab, we'll explore some of the simple ways we can account for resource usage with pods and nodes, and setup and use quotas.

1. Before we launch any more deployments, let's set up some specific policies and classes of pods that work with those policies. First, we'll setup some priority classes. Take a look at the definition of the pod priorities and then apply the definition to create them.

```
$ cd ~/adv-k8s/extra
$ cat pod-priorities.yaml
$ k apply -f ./pod-priorities.yaml
```

2. Now, that the priority classes have been created, we'll create some resource quotas built around them. Since quotas are namespace-specific, we'll go ahead and create a new namespace to work in. Take a look at the definition of the quotas and then apply the definition to create them.

```
$ k create ns quotas
$ cat pod-quotas.yaml
$ k apply -f ./pod-quotas.yaml -n quotas
```

3. After setting up the quotas, you can see how things are currently setup and allocated.

```
$ k get priorityClasses

$ k describe quota -n quotas
```

4. In the roar-quota directory we have a version of our charts with requests, limits and priority classes assigned. You can take a look at those by looking at the end of the deployment.yaml templates. After that, go ahead and install the release.

```
$ cd ~/adv-k8s/roar-quotas

$ cat charts/roar-db/templates/deployment.yaml

$ cat charts/roar-web/templates/deployment.yaml

$ helm install -n quotas quota .
```

5. After a few moments, take a look at the state of the pods in the deployment. Notice that while the web pod is running, the database one does not exist. Let's figure out why. Since there is no pod to do a describe on, we'll look for a replicaset.

```
$ k get pods -n quotas
$ k get rs -n quotas
```

6. Notice the mysql replicaset. It has DESIRED=1, but CURRENT=0. Let's do a describe on it to see if we can find the problem.

```
$ k describe -n quotas rs -l app=mysql
```

7. What does the error message say? The request for memory we asked for the pod exceeds the quota for the quota "pods-average". If you recall, the pods-average one has a memory limit of 5Gi. The pods-critical one has a higher memory limit of 10Gi. So let's change priority class for the mysql pod to be critical.

```
$ gedit charts/roar-db/templates/deployment.yaml
```

change the last line from

```
priorityClassName: average
```

to

```
priorityClassName: critical
```

being careful not to change the spaces at the start of the line.

Save your changes and exit the editor. (Ignore the Gtk-WARNING message)

8. Upgrade the Helm release to get your changes deployed and then look at the pods again.

```
$ helm upgrade -n quotas quota .
$ k get pods -n quotas
```

9. Notice that while the mysql pod shows up in the list, its status is "Pending". Let's figure out why that is by doing a describe on it.

```
$ k describe -n quotas pod -l app=mysql
```

10. The error message indicates that there are no nodes available with enough memory to schedule this pod. Note that this does not reference any quotas we've setup. Let's get the list of nodes (there's only 1 in the VM) and check how much memory is available on our node. Use the first command to get the name of the node and the second to check how much memory it has.

```
$ k get nodes
$ k describe node <node-name-goes-here> | grep memory
```

11. Our mysql pod is asking for an unrealistically large number (to provoke the error). Even if it were just the under the amount available on the node, other processes running on the node in other namespaces could be using several Gi.

12. Getting back to our needs let's drop the limit and request values down to 5 and 3 respectively and see if that fixes things.

```
$ gedit charts/roar-db/templates/deployment.yaml
```

change the two lines near the bottom from

```
          memory: "100Gi"
```

to

```
          memory: "5Gi"  (for limits)
```

and

```
          memory: "1Gi"  (for requests)
```

Save your changes and exit the editor.

13. Do a helm upgrade and add the "--recreate-pods" option to force the pods to be recreated. After a moment if you check, you should see the pods running now. Finally, you can check the quotas again to see what is being used.

```
$ helm upgrade -n quotas quota --recreate-pods .
(ignore the deprecated warning)
$ k get pods -n quotas
$ k describe quota -n quotas
```

END OF LAB

Lab 3: Selecting Nodes

Purpose: In this lab, we'll explore some of the ways we can tell Kubernetes which node to schedule pods on

1. The files for this lab are in the roar-affin subdirectory. Change to that, create a namespace, and do a Helm install of our release.

```
$ cd ~/adv-k8s/roar-affin
$ k create ns affin
$ helm install -n affin affin .
```

2. Take a look at the status of the pods in the namespace. You'll notice that they are not ready. Let's figure out why. Start with the mysql one and do a describe on it.

```
$ k get pods -n affin
$ k describe -n affin pod -l app=mysql
```

3. In the output of the describe command, in the Events section, you can see that it failed to be scheduled because there were "0/1 nodes are available: 1 node(s) didn't match node selector". And further up, you can see that it is looking for a Node-Selector of "type=mini".

4. This means the pod definition expected at least one node to have a label of "type=mini". Take a look at what labels are on our single node now.

```
$ k get nodes --show-labels
```

5. Since we don't have the desired label on the node, we'll add it and then verify it's there.

```
$ k label node <node-name-goes-here> type=mini
$ k get nodes --show-labels | grep type
```

6. At this point, if you look again at the pods in the namespace you should see that the mysql pod is now running. Also, if you do a describe on it, you'll see an entry in the Events: section where it was scheduled.

```
$ k get pods -n affin
$ k describe -n affin pod -l app=mysql
```

7. Now, let's look at the web pod. If you do a describe on it, you'll see similar messages about problems scheduling. But the node-selector entry will not list one. This is because we are using the node affinity functionality here. You can see the affinity definition by running the second command below.

```
$ k describe pod -n affin -l app=roar-web
$ k get -n affin pod -l app=roar-web -o yaml | grep affinity -A10
```

8. In the output from the grep, you can see that the nodeAffinity setting is "*requiredDuringSchedulingIgnoredDuringExecution*" and it would match up with a label of "*system=minikube*" or "*system=single*". But let's assume that we don't really need a node like that, it's only a preference. If that's the case we can change the pod spec to use "*preferredDuringSchedulingIgnoredDuringExecution*". To do that you can either edit the deployment.yaml file directly

```
$ gedit charts/roar-web/templates/deployment.yaml
```

and change

```
requiredDuringSchedulingIgnoredDuringExecution:
  nodeSelectorTerms:
  - matchExpressions:
```

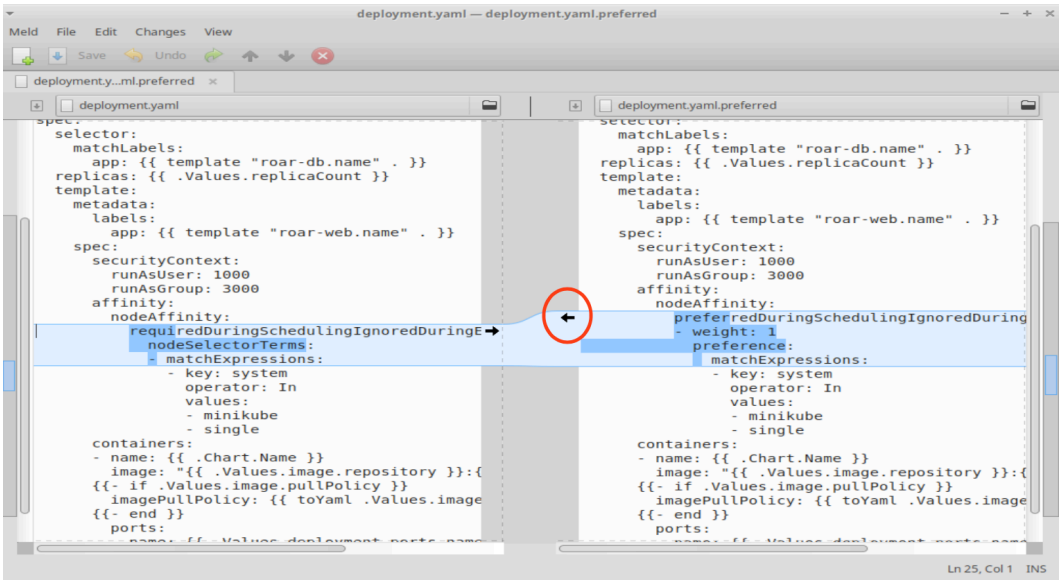
to

```
preferredDuringSchedulingIgnoredDuringExecution:
- weight: 1
  preference:
    matchExpressions:
```

Or, you can use the meld tool to merge in the changes from a file in the extra area. Just issue the command below, click on the right arrow (the one circled in the figure) and save the files and exit meld.

(This assumes you are in the adv-k8s/roar-affin directory.)

```
$ meld charts/roar-web/templates/deployment.yaml ../extra/deployment.yaml.preferred
```



9. Now, upgrade the deployment with the recreate-pods option to see the changes take effect.

```
$ helm upgrade -n affin affin --recreate-pods .
```

10. After a few moments, you should be able to get the list of pods and see that the web one is running now too. You can do a describe if you want and see that it has been assigned to the training1 node since it was no longer a requirement to match those labels.

```
$ k get pods -n affin
```

END OF LAB

Lab 4: Working with Taints and Tolerations

Purpose: In this lab, we'll explore some of the uses of taints and tolerations in Kubernetes

1. The files for this lab are in the roar-taint subdirectory. Change to that, create a namespace, and do a Helm install of our release.

```
$ cd ~/adv-k8s/roar-taint
$ k create ns taint
$ helm install -n taint taint .
```

2. At this point, all pods should be running because there are no taints on the node. (You can do a get on the pods to verify if you want.). Let's add a taint on the node that implies that pods must be part of the roar app to be scheduled.

```
$ k get pods -n taint
$ k taint nodes <node-name-goes-here> roar=app:NoSchedule
```

3. Now, let's delete the release and install again. Then take a look at the pods.

```
$ helm delete -n taint taint
$ helm install -n taint taint .
$ k get pods -n taint
```

4. The web pod has failed to be scheduled. Do a describe to see why.

```
$ k describe -n taint pod -l app=roar-web
```

5. Notice that it says "*1 node(s) had taints that the pod didn't tolerate.*" So our database pod must have had a toleration for it since it was running. Take a look at the two tolerations in the database pod (at the end of the deployment.yaml file).

```
$ cat charts/roar-db/templates/deployment.yaml
```

6. Notice the toleration for "*roar*" and "*Exists*". This says that the pod can run on the node even if the taint we created in step 2 above is there - regardless of the value. We need to add this to our web pod spec so it can run there as well.

   You can do this either by editing the file directly

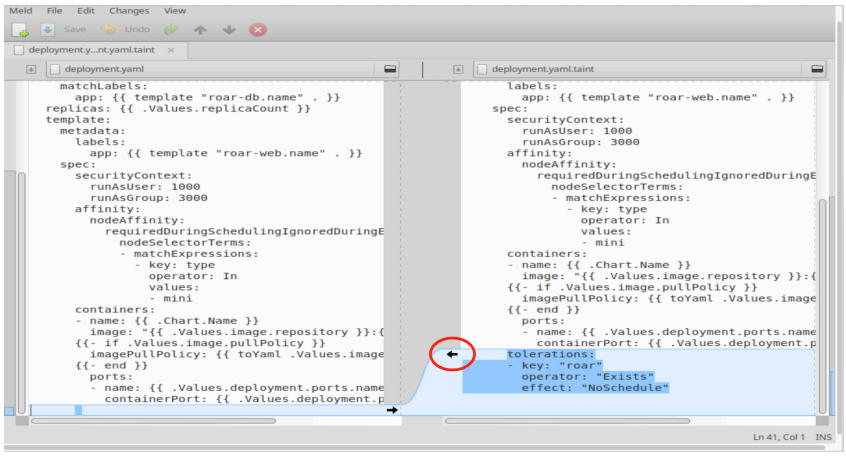
```
$ gedit charts/roar-web/templates/deployment.yaml
```

   and adding these lines (lining up with the same starting column as "containers:")

```
tolerations:
 - key: "roar"
   operator: "Exists"
   effect: "NoSchedule"
```

   Or you can use the meld tool by running the following command, clicking on the right arrow (the one circled) and then saving the file and exiting meld.

```
$ meld charts/roar-web/templates/deployment.yaml ../extra/deployment.yaml.taint
```



7. Now with the toleration added for the web pod, do an upgrade to see if we can get the web pod scheduled now.

```
$ helm upgrade -n taint taint  .
$ k get pods -n taint
```

8. Now let's add one more taint for the other toleration that the mysql pod had. Afterwards, take a look at the state of the pods.

```
$ k taint nodes <node-name-goes-here> use=database:NoExecute
$ k get pods -n taint
```

9. Why is the web pod not running? The database pod has a toleration for this taint. You can see that in the charts/roar-db/templates/deployment.yaml file near the bottom. You can also do a describe on the web pod again if you want to see that it didn't tolerate the new taint.

```
$ cat charts/roar-db/templates/deployment.yaml
$ k describe -n taint pod -l app=roar-web
```

10. But the web pod doesn't have this toleration, so because of the "No Execute" policy, it gets kicked out. We could add a toleration to the web pod spec for this, but for simplicity, let's just remove the taint to get things running again.

```
$ k taint nodes <node-name-goes-here> use:NoExecute-
$ k get pods -n taint
```

11. Go ahead and remove the other taint to prepare for future labs.

```
$ k taint nodes <node-name-goes-here> roar:NoSchedule-
```

END OF LAB

Lab 5 - Working with Pod Security Admission Controllers

Purpose: In this lab, we'll learn more about what a pod security admission controller is and why they are needed.

1. The files for this lab are in the roar-context subdirectory. Change to that, create a namespace, and do a Helm install of our release.

```
$ cd ../roar-context

$ k create ns context

$ helm install -n context context .
```

2. Our pods should be running now. But we want to make sure that our current workloads do not potentially violate the baseline policy. So we'll do a dry-run on the namespace to check.

```
$ k get pods -n context

$ k label --dry-run=server --overwrite ns context  pod-security.kubernetes.io/enforce=baseline
```

3. It looks like our workloads are good if we want to just enforce the baseline policy. Let's check though for the restricted policy.

```
$ k label --dry-run=server --overwrite ns context pod-security.kubernetes.io/enforce=restricted
```

4. Notice the warning messages from this run. Let's see what would happen if we were to actually enforce the restricted policy instead. In the directory **extra**, there is a file named psa-ns.yml that has a definition for a namespace with restricted policy enforced. Go ahead and look at that file and then apply it to create the namespace.

```
$ cat ../extra/psa-ns.yml

$ k apply -f ../extra/psa-ns.yml
```

5. Now, let's see what happens when we try to install the same helm chart in the new namespace.

```
$ helm install -n psa context .
```

6. Helm reports that it is deployed. But take a look at the pods in the namespace.

```
$ k get pods -n psa
```

7. There are no pods there because they were not permitted. Let's update the deployment manifest for the database charts to fix the issues. Either use the meld command below or edit the file **roar-context/charts/roar-db/templates/deployment.yaml** and add these lines (lining up with the same starting column as "ports:" and "env:")

   Meld option:

```
$ meld charts/roar-db/templates/deployment.yaml ..extra/deployment.yaml.psa-db
```

   Editor option:

```
        securityContext:
          allowPrivilegeEscalation: false
          capabilities:
            drop: ["ALL"]
      securityContext:
        runAsUser: 1000
        runAsGroup: 999
        runAsNonRoot: true
        seccompProfile:
          type: RuntimeDefault
```

See screenshot below for reference:

roar-context > charts > roar-db > templates > ! deployment.yaml

```
20        containers:
21        - name: {{ .Chart.Name }}
22          image: "{{ .Values.image.repository }}:{{ .Values.image.tag }}"
23        {{- if .Values.image.pullPolicy }}
24          imagePullPolicy: {{ toYaml .Values.image.pullPolicy }}
25        {{- end }}
26          ports:
27          - name: {{ .Values.deployment.ports.name }}
28            containerPort: {{ .Values.deployment.ports.containerPort }}
29          env:
30            {{- include "roar-all.environment-values" . | indent 10 }}
31          securityContext:
32            allowPrivilegeEscalation: false
33            capabilities:
34              drop: ["ALL"]
35        securityContext:
36          runAsUser: 1000
37          runAsGroup: 999
38          runAsNonRoot: true
39          seccompProfile:
40            type: RuntimeDefault
41
42
```

8. Now, upgrade the deployment to deploy the new manifest. And verify that the mysql pod has been admitted and has started up.

```
$ helm upgrade -n psa context .

$ k get pods -n psa
```

9. Repeat steps 7 and 8 for the web deployment manifest - the file **roar-context/charts/roar-web/templates/deployment.yaml**

Meld option:

```
$ meld charts/roar-web/templates/deployment.yaml ..extra/deployment.yaml.psa-web
```

END OF LAB