Helm and Kustomize

Orchestrating your Deployments to Kubernetes

Session Labs: Revision 1.1 - 11/03/20

Brent Laster

Important Note: Prior to starting with this document, you should have the ova file for the class (the virtual machine) already loaded into Virtual Box and have ensured that it is startable. See the setup doc helm-setup.pdf in http://github.com/brentlaster/safaridocs for instructions and other things to be aware of.

Lab 1: Repos and Charts

Purpose: In this lab, we'll start working with Helm pulling down a Helm chart, examining it, and then installing it.

1. First, let's look at what repositories we have available to our local helm instance. **Open a terminal emulator** and enter

```
$ helm repo list
```

2. We have a local repository available via the ChartMuseum application that is already setup and running on your VM. There is a chart in there for our sample application named "roar-helm". Let's see what we can find out about it.

```
$ helm search repo roar-helm

$ helm show chart local/roar-helm
```

3. Now let's pull a copy of the chart down to look at the file structure. Change into the directory for the workshop, pull a copy and look at the structure.

```
$ cd hk-ws

$ helm pull --untar local/roar-helm

$ cd roar-helm

$ tree
```

You can further look at individual files if you want.

4. Now let's install a release of roar-helm in our local Kubernetes. We'll need to first create a namespace and then install the release. We also need to give it a name when we install. (The Kubernetes command line application, kubectl, is aliased to “k” on this machine.)

```
$ k create ns roar

$ helm install -n roar roar .
```

5. Take a look at the release that has been installed in the cluster.

```
$ helm list -n roar
```

You should see output like the following:

```
NAME   NAMESPACE   REVISION   UPDATED                                       STATUS     CHART         APP VERSION
roar   roar        1          2020-06-03 22:23:59.206949569 -0400 EDT   deployed   roar-web-0.1.0
```

6. Now take a look at the resources that are installed in the cluster.

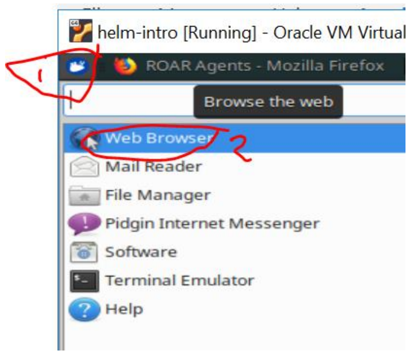
```
$ k get all -n roar
```

7. Find the nodeport where the app is running and open it up in a browser.

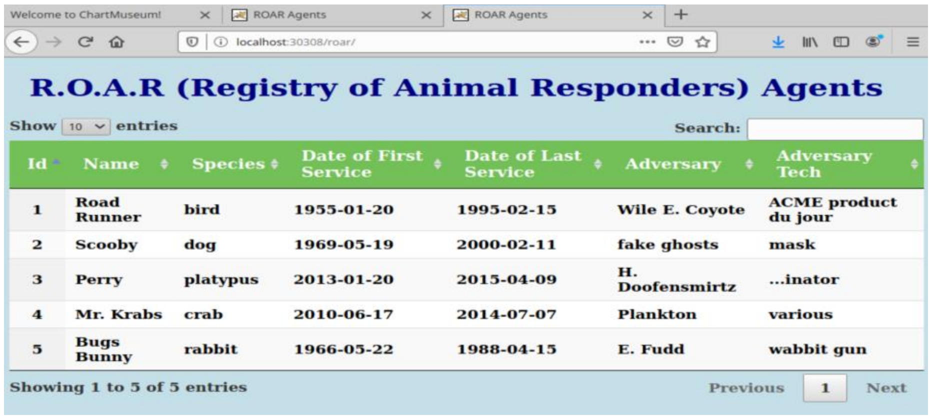
```
$ k get svc -n roar
```

Look for the NodePort setting in the service output (should be a number > 30000 after "8089:")

8. Open up a browser and go to http://localhost:<NodePort>/roar/


helm-intro [Running] - Oracle VM Virtual
ROAR Agents - Mozilla Firefox
Browse the web
1
Web Browser
2
Mail Reader
File Manager
Pidgin Internet Messenger
Software
Terminal Emulator
Help

You should see something like this:



R.O.A.R (Registry of Animal Responders) Agents

Show 10 entries Search:

Id	Name	Species	Date of First Service	Date of Last Service	Adversary	Adversary Tech
1	Road Runner	bird	1955-01-20	1995-02-15	Wile E. Coyote	ACME product du jour
2	Scooby	dog	1969-05-19	2000-02-11	fake ghosts	mask
3	Perry	platypus	2013-01-20	2015-04-09	H. Doofensmirtz	...inator
4	Mr. Krabs	crab	2010-06-17	2014-07-07	Plankton	various
5	Bugs Bunny	rabbit	1966-05-22	1988-04-15	E. Fudd	wabbit gun

Showing 1 to 5 of 5 entries Previous 1 Next

Lab 2: Templating with Helm

Purpose: In this lab, you'll get to see how we can change hard-coded values into templates, override values, and and upgrade releases through Helm.

1. Take a look at the deployment template in the roar-helm directory and notice what the "image" value is set to. In *hk-ws/roar-helm*:

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

   Notice that the value for image is hardcoded to "*bclaster/roar-db-image:v1*".

2. We are going to change this to use the Helm templating facility. This means we'll change this value in the deployment.yaml file to have "placeholders". And we will put the default values we want to have in the values.yaml file. 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.

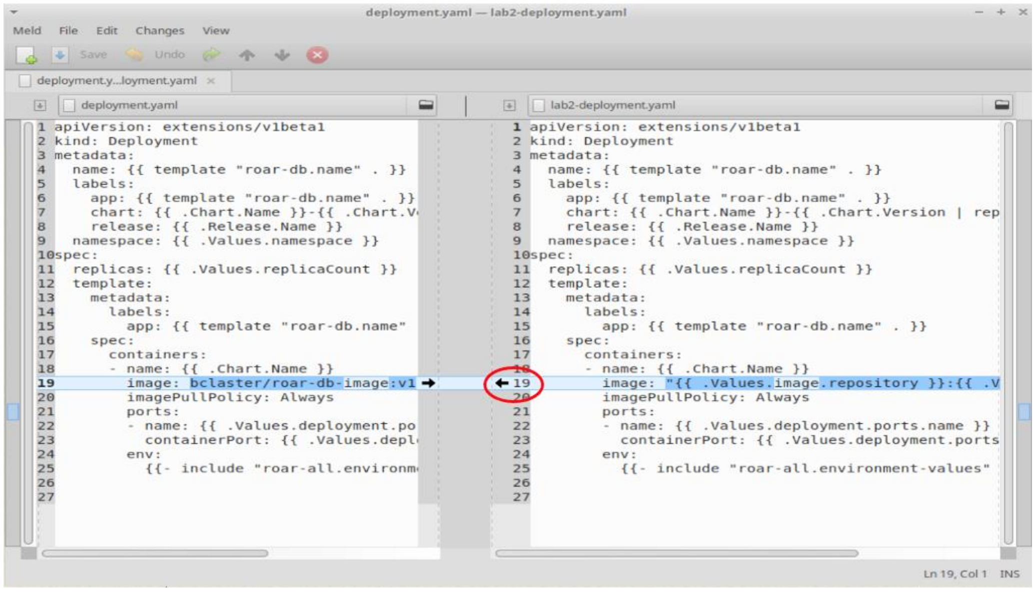
   Either do:

```
      $ gedit charts/roar-db/templates/deployment.yaml
And change
      image: bclaster/roar-db-image:v1
To
      image: "{{ .Values.image.repository }}:{{ .Values.image.tag }}"
```

   Or:

```
      $ meld charts/roar-db/templates/deployment.yaml ../extra/lab2-
deployment.yaml
```

   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.


deployment.yaml — lab2-deployment.yaml
Meld File Edit Changes View
Save Undo
deployment.y...loyment.yaml
deployment.yaml
lab2-deployment.yaml
Ln 19, Col 1 INS

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: {{ template "roar-db.name" . }}
  labels:
    app: {{ template "roar-db.name" . }}
    chart: {{ .Chart.Name }}-{{ .Chart.V
    release: {{ .Release.Name }}
  namespace: {{ .Values.namespace }}
spec:
  replicas: {{ .Values.replicaCount }}
  template:
    metadata:
      labels:
        app: {{ template "roar-db.name"
    spec:
      containers:
      - name: {{ .Chart.Name }}
        image: bclaster/roar-db-image:v1
        imagePullPolicy: Always
        ports:
        - name: {{ .Values.deployment.po
          containerPort: {{ .Values.depl
        env:
          {{- include "roar-all.environm
```

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: {{ template "roar-db.name" . }}
  labels:
    app: {{ template "roar-db.name" . }}
    chart: {{ .Chart.Name }}-{{ .Chart.Version | rep
    release: {{ .Release.Name }}
  namespace: {{ .Values.namespace }}
spec:
  replicas: {{ .Values.replicaCount }}
  template:
    metadata:
      labels:
        app: {{ template "roar-db.name" . }}
    spec:
      containers:
      - name: {{ .Chart.Name }}
        image: "{{ .Values.image.repository }}:{{ .V
        imagePullPolicy: Always
        ports:
        - name: {{ .Values.deployment.ports.name }}
          containerPort: {{ .Values.deployment.ports
        env:
          {{- include "roar-all.environment-values"
```

3. Now that we've updated the deployment template, we need to add default values. We'll use the same approach as in the previous step to add defaults for the *image.repository* and *image.tag* values in the chart's *values.yaml* file.

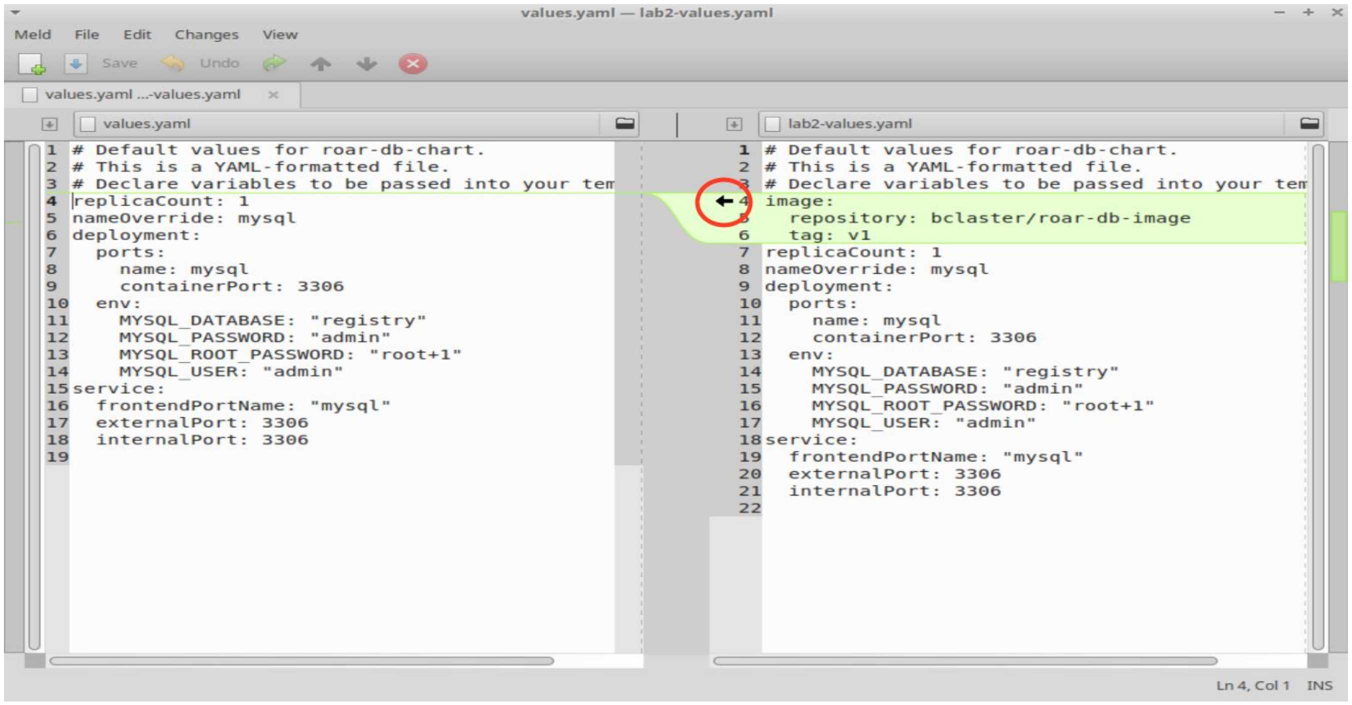
   Either do:

   ```
         $ gedit charts/roar-db/values.yaml
   And add to the top of the file:
         image:
           repository: bclaster/roar-db-image
           tag: v1
   ```

   Or:

   ```
         $ meld charts/roar-db/values.yaml ../extra/lab2-values.yaml
   ```

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



```
# Default values for roar-db-chart.
# This is a YAML-formatted file.
# Declare variables to be passed into your tem
replicaCount: 1
nameOverride: mysql
deployment:
  ports:
    name: mysql
    containerPort: 3306
  env:
    MYSQL_DATABASE: "registry"
    MYSQL_PASSWORD: "admin"
    MYSQL_ROOT_PASSWORD: "root+1"
    MYSQL_USER: "admin"
service:
  frontendPortName: "mysql"
  externalPort: 3306
  internalPort: 3306
```

```
# Default values for roar-db-chart.
# This is a YAML-formatted file.
# Declare variables to be passed into your tem
image:
  repository: bclaster/roar-db-image
  tag: v1
replicaCount: 1
nameOverride: mysql
deployment:
  ports:
    name: mysql
    containerPort: 3306
  env:
    MYSQL_DATABASE: "registry"
    MYSQL_PASSWORD: "admin"
    MYSQL_ROOT_PASSWORD: "root+1"
    MYSQL_USER: "admin"
service:
  frontendPortName: "mysql"
  externalPort: 3306
  internalPort: 3306
```

4. Run a lint check on your updated chart files.

   ```
         $ helm lint
   ```

5. Uninstall the existing release.

   ```
         $ helm uninstall -n roar roar
   ```

6. Now let's install the updated release.

```
$ helm install -n roar roar .
```

7. Find the nodeport where the app is running and open it up in a browser.

```
$ k get svc -n roar
```

Look for the NodePort setting in the service output (should be a number > 30000 after "8089:")

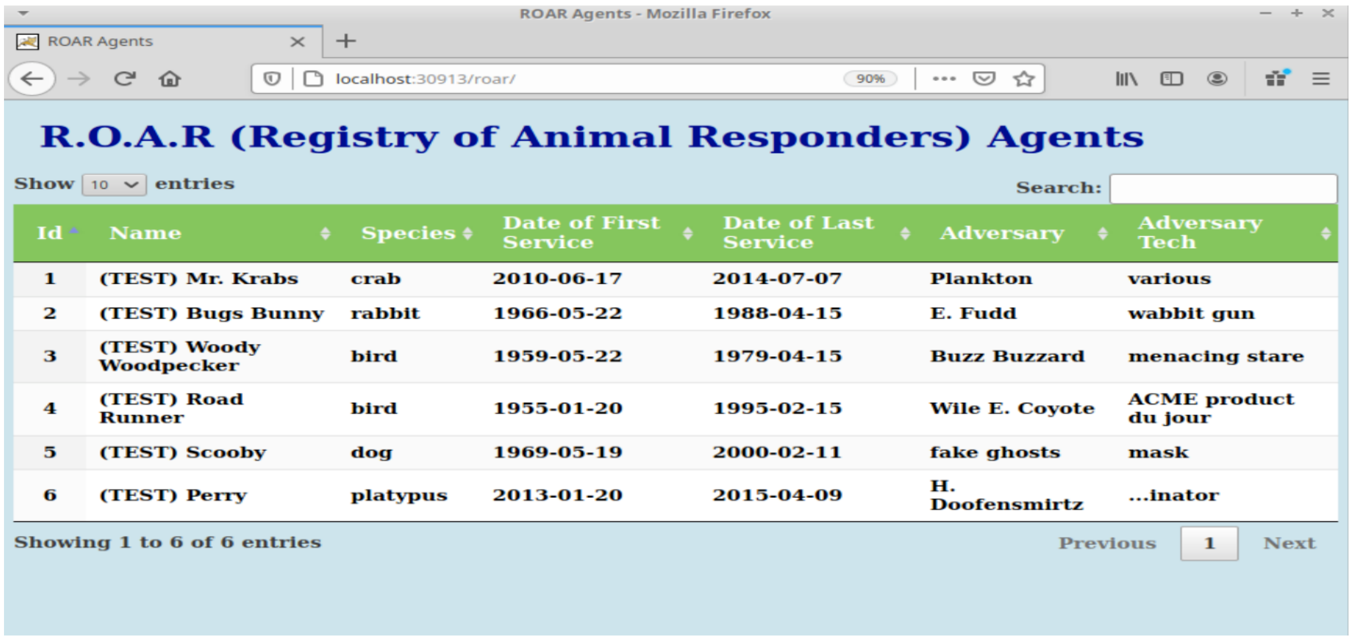
8. Open up a browser and go to http://localhost:<NodePort>/roar/

   You should see the same webapp and data as before.

9. Now let's see how to override values from the helm command line. We'll do an upgrade and pass in a new image value that is for a "test version" of the database.

```
$  helm upgrade -n roar roar . --recreate-pods
--set roar-db.image.repository=bclaster/roar-db-test,roar-db.image.tag=v4
```

10. Refresh the browser and you should see the test data showing up as in the picture below.



R.O.A.R (Registry of Animal Responders) Agents

Show 10 entries   Search:

Id	Name	Species	Date of First Service	Date of Last Service	Adversary	Adversary Tech
1	(TEST) Mr. Krabs	crab	2010-06-17	2014-07-07	Plankton	various
2	(TEST) Bugs Bunny	rabbit	1966-05-22	1988-04-15	E. Fudd	wabbit gun
3	(TEST) Woody Woodpecker	bird	1959-05-22	1979-04-15	Buzz Buzzard	menacing stare
4	(TEST) Road Runner	bird	1955-01-20	1995-02-15	Wile E. Coyote	ACME product du jour
5	(TEST) Scooby	dog	1969-05-19	2000-02-11	fake ghosts	mask
6	(TEST) Perry	platypus	2013-01-20	2015-04-09	H. Doofensmirtz	...inator

Showing 1 to 6 of 6 entries   Previous 1 Next

Lab 3: Using Functions and Pipelines

Purpose: In this lab, we'll see how to use functions and pipelines to expand what we can do in Helm charts.

1. In our last lab, we saw how to overwrite our production database with a test database via a command line option. But what if we want to build this option into the chart?

   For switching between the prod and test database, we'll add two different versions of the image to choose from - one for prod and one for test. We'll also add a condition to report a failure if we don't have a valid entry passed.

   Edit and modify the *charts/roar-db/values.yaml* file and update as indicated below or use the meld command for a simpler experience.

   Either do:

   ```
   $ gedit charts/roar-db/values.yaml
   ```

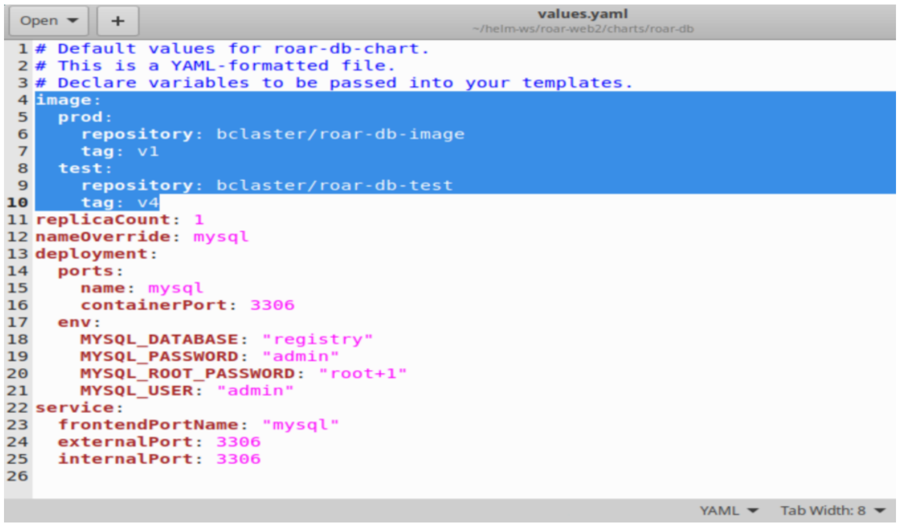
   And replace these lines

   ```
   image:
     repository: bclaster/roar-db-image
     tag: v1
   ```

   with the following lines

   ```
   image:
     prod:
       repository: bclaster/roar-db-image
       tag: v1
     test:
       repository: bclaster/roar-db-test
       tag: v4
   ```

   (See screenshot below and pay attention to the spaces and alignment.)

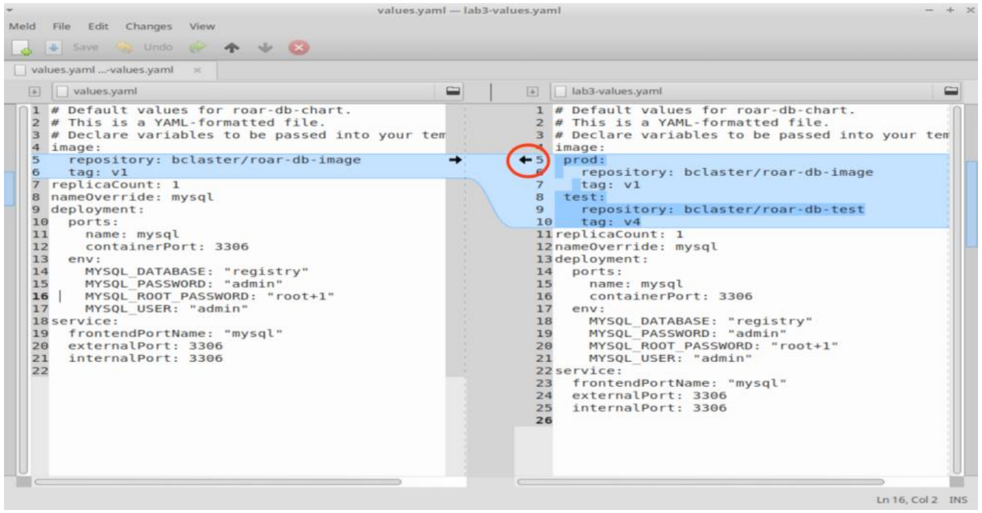


```
# Default values for roar-db-chart.
# This is a YAML-formatted file.
# Declare variables to be passed into your templates.
image:
  prod:
    repository: bclaster/roar-db-image
    tag: v1
  test:
    repository: bclaster/roar-db-test
    tag: v4
replicaCount: 1
nameOverride: mysql
deployment:
  ports:
    name: mysql
    containerPort: 3306
  env:
    MYSQL_DATABASE: "registry"
    MYSQL_PASSWORD: "admin"
    MYSQL_ROOT_PASSWORD: "root+1"
    MYSQL_USER: "admin"
service:
  frontendPortName: "mysql"
  externalPort: 3306
  internalPort: 3306
```

Or:

```
$ meld charts/roar-db/values.yaml ../extra/lab3-values.yaml
```

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


values.yaml — lab3-values.yaml
Meld File Edit Changes View
Save Undo
values.yaml ...-values.yaml
values.yaml
lab3-values.yaml
Ln 16, Col 2 INS

```
# Default values for roar-db-chart.
# This is a YAML-formatted file.
# Declare variables to be passed into your tem
image:
  repository: bclaster/roar-db-image
  tag: v1
replicaCount: 1
nameOverride: mysql
deployment:
  ports:
    name: mysql
    containerPort: 3306
  env:
    MYSQL_DATABASE: "registry"
    MYSQL_PASSWORD: "admin"
    MYSQL_ROOT_PASSWORD: "root+1"
    MYSQL_USER: "admin"
service:
  frontendPortName: "mysql"
  externalPort: 3306
  internalPort: 3306
```

```
# Default values for roar-db-chart.
# This is a YAML-formatted file.
# Declare variables to be passed into your tem
image:
 prod:
   repository: bclaster/roar-db-image
   tag: v1
 test:
   repository: bclaster/roar-db-test
   tag: v4
replicaCount: 1
nameOverride: mysql
deployment:
  ports:
    name: mysql
    containerPort: 3306
  env:
    MYSQL_DATABASE: "registry"
    MYSQL_PASSWORD: "admin"
    MYSQL_ROOT_PASSWORD: "root+1"
    MYSQL_USER: "admin"
service:
  frontendPortName: "mysql"
  externalPort: 3306
  internalPort: 3306
```

2. Now, we'll add some processing to enable selecting one of these images based on passing a setting for "stage" of either "PROD" or "TEST". The design logic (not what we actually add) is

```
if stage = PROD
      use image.prod.repository :  image.prod.tag
else if stage = TEST
      use image.test.repository : image.test.tag
else
      display error message that we don't have a valid stage setting
```

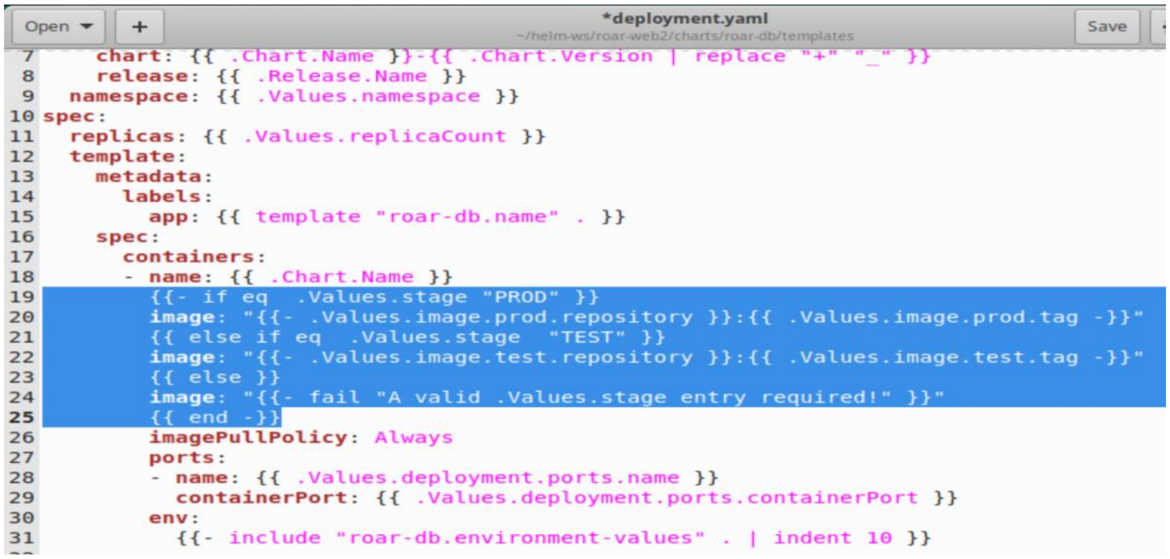
Translating this into templating syntax, we use an if/(else if)/else flow. The "eq" function is used for comparison, passing the two things to compare after it. Also, there is a "required" function that will handle the default error checking. Putting it all together, it could look like this:

```
{{- if eq .Values.stage  "PROD" }}
  image: "{{- .Values.image.prod.repository }}:{{ .Values.image.prod.tag -}}"
{{ else if eq .Values.stage  "TEST" }}
  image: "{{- .Values.image.test.repository }}:{{ .Values.image.test.tag -}}"
{{ else }}
  image: "{{- fail "A valid .Values.stage entry required!"  }}"
{{ end -}}
```

3. In the ~/hk-ws/roar-helm directory, edit the *charts/roar-db/templates/deployment.yaml* file and replace line 19 with the code from above in the containers section (see screenshot below). Pay

attention to indentation and use spaces not tabs. Or use the meld command further down for a simpler experience:

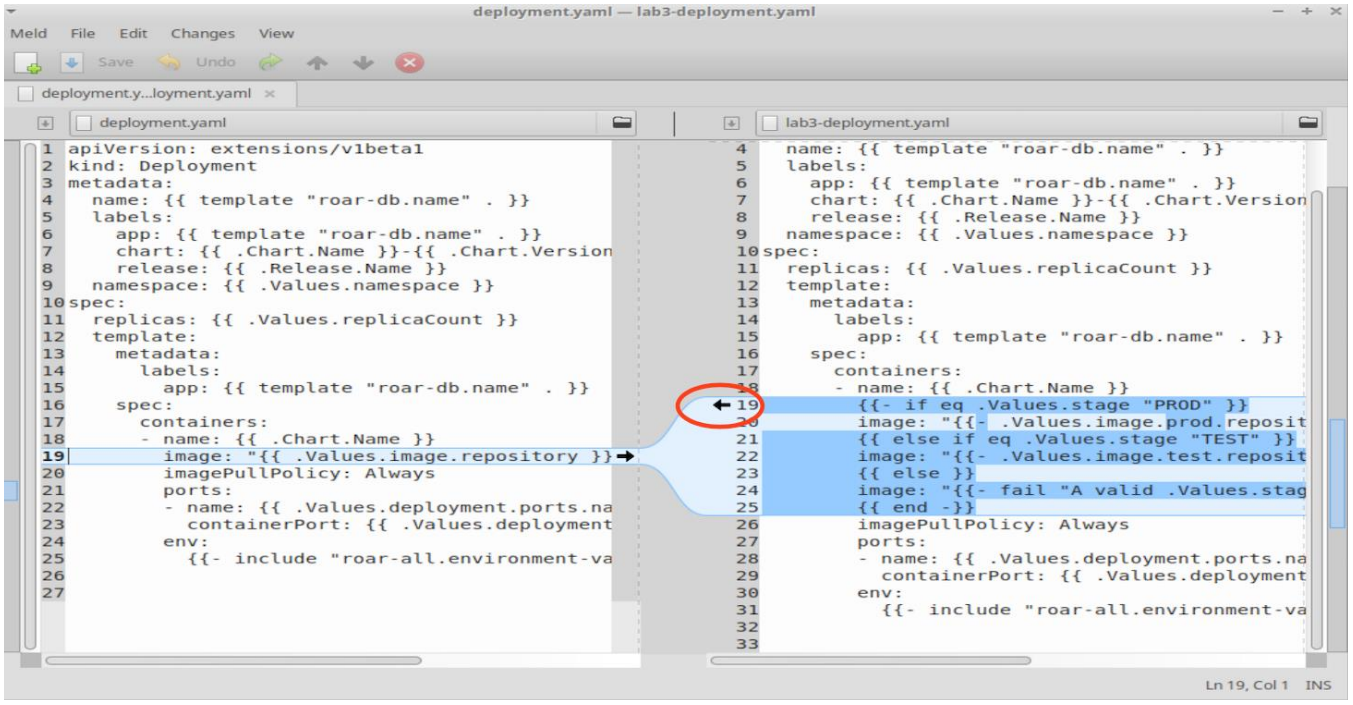
```
$ gedit charts/roar-db/templates/deployment.yaml
  <replace the existing image line with the lines from step 2>
```



```
    chart: {{ .Chart.Name }}-{{ .Chart.Version | replace "+" "_" }}
    release: {{ .Release.Name }}
  namespace: {{ .Values.namespace }}
spec:
  replicas: {{ .Values.replicaCount }}
  template:
    metadata:
      labels:
        app: {{ template "roar-db.name" . }}
    spec:
      containers:
      - name: {{ .Chart.Name }}
        {{- if eq  .Values.stage "PROD" }}
        image: "{{- .Values.image.prod.repository }}:{{ .Values.image.prod.tag -}}"
        {{ else if eq  .Values.stage  "TEST" }}
        image: "{{- .Values.image.test.repository }}:{{ .Values.image.test.tag -}}"
        {{ else }}
        image: "{{- fail "A valid .Values.stage entry required!" }}"
        {{ end -}}
        imagePullPolicy: Always
        ports:
        - name: {{ .Values.deployment.ports.name }}
          containerPort: {{ .Values.deployment.ports.containerPort }}
        env:
          {{- include "roar-db.environment-values" . | indent 10 }}
```

Or

```
$ meld charts/roar-db/templates/deployment.yaml ../extra/lab3-deployment.yaml
```



```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: {{ template "roar-db.name" . }}
  labels:
    app: {{ template "roar-db.name" . }}
    chart: {{ .Chart.Name }}-{{ .Chart.Version
    release: {{ .Release.Name }}
  namespace: {{ .Values.namespace }}
spec:
  replicas: {{ .Values.replicaCount }}
  template:
    metadata:
      labels:
        app: {{ template "roar-db.name" . }}
    spec:
      containers:
      - name: {{ .Chart.Name }}
        image: "{{ .Values.image.repository }}
        imagePullPolicy: Always
        ports:
        - name: {{ .Values.deployment.ports.na
          containerPort: {{ .Values.deployment
        env:
          {{- include "roar-all.environment-va
```

```
  name: {{ template "roar-db.name" . }}
  labels:
    app: {{ template "roar-db.name" . }}
    chart: {{ .Chart.Name }}-{{ .Chart.Version
    release: {{ .Release.Name }}
  namespace: {{ .Values.namespace }}
spec:
  replicas: {{ .Values.replicaCount }}
  template:
    metadata:
      labels:
        app: {{ template "roar-db.name" . }}
    spec:
      containers:
      - name: {{ .Chart.Name }}
        {{- if eq .Values.stage "PROD" }}
        image: "{{- .Values.image.prod.reposit
        {{ else if eq .Values.stage "TEST" }}
        image: "{{- .Values.image.test.reposit
        {{ else }}
        image: "{{- fail "A valid .Values.stag
        {{ end -}}
        imagePullPolicy: Always
        ports:
        - name: {{ .Values.deployment.ports.na
          containerPort: {{ .Values.deployment
        env:
          {{- include "roar-all.environment-va
```

4. Save your changes and do a dry-run to make sure the image comes out as expected. To simplify seeing the change, we'll grep for image with a few lines of context.

```
$ helm install --set roar-db.stage=TEST --dry-run FOO . | grep image -n3
```

You should see output like the following:

```
65-    spec:
66-      containers:
67-      - name: roar-db
68:        image: "bclaster/roar-db-test:v4"
69:        imagePullPolicy: Always
70-        ports:
71-        - name: mysql
72-          containerPort: 3306
--
```

5. While this appears to work, what happens if we pass a lower-case value?

```
$ helm install --set roar-db.stage=test   --dry-run FOO . | grep image -n3
```

Notice we get the error message saying we don't have a valid .Values.stage value (as we might expect).

6. This is not what we want. Let's make our arguments case-insensitive. To do this, we'll pipe the value we pass in through a pipeline and to another template function called "**upper**" to upper-case it first.

Edit the deployment file and make the changes shown below or use the meld option for a simpler experience.

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

Change line 19 from

```
{{- if eq .Values.stage  "PROD" }}
```

to

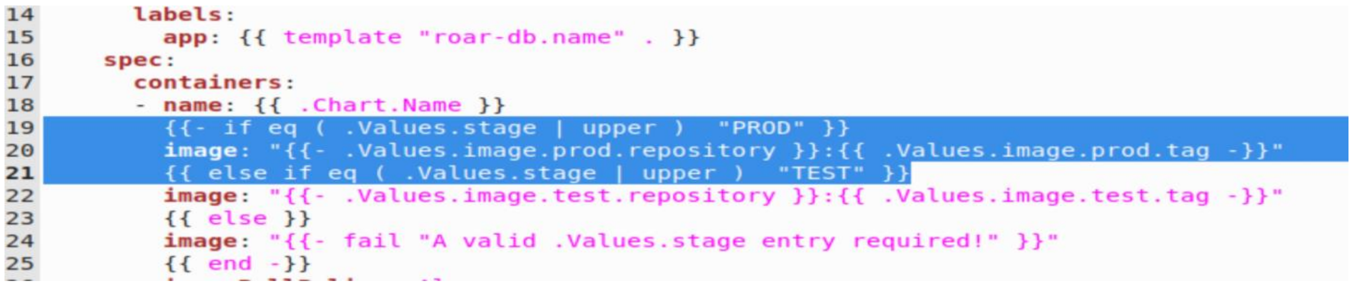
```
{{- if eq  ( .Values.stage | upper )  "PROD" }}
```

and change line 21 from

```
        {{ else if eq .Values.stage  "TEST" }}
```

to

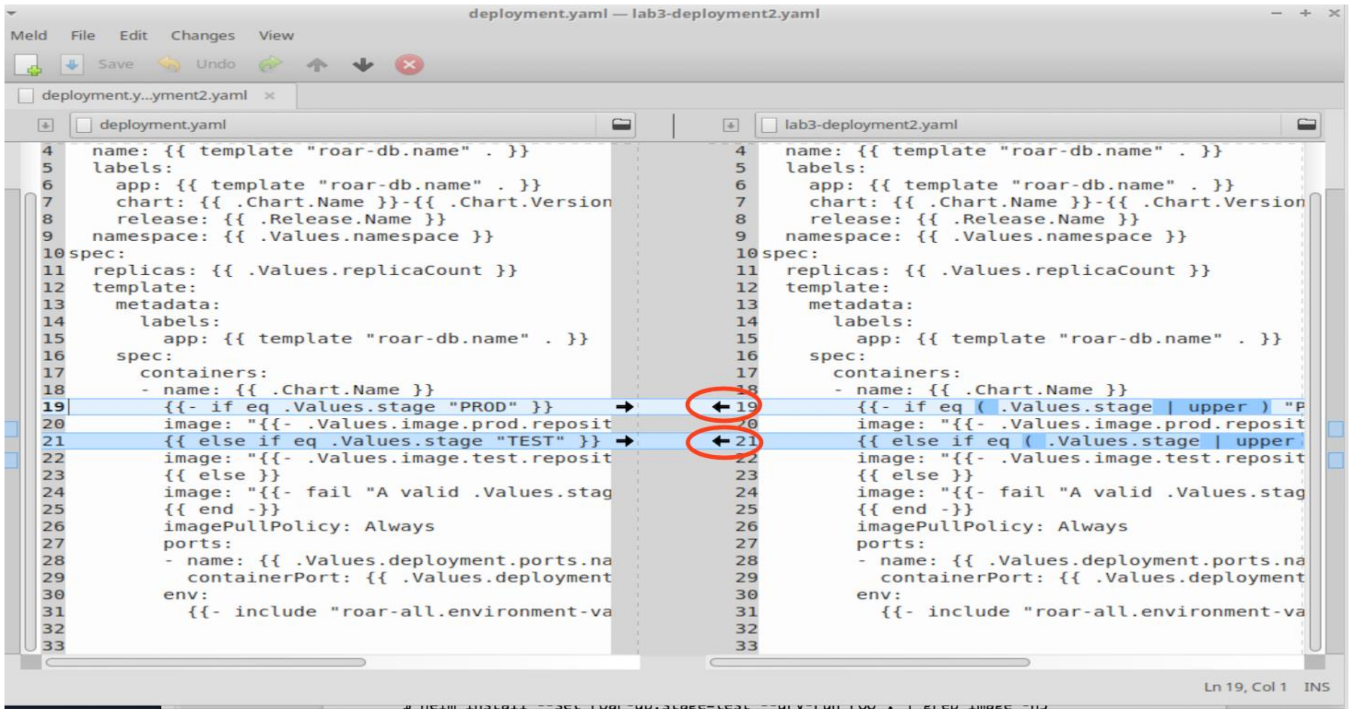
```
        {{ else if eq ( .Values.stage | upper )  "TEST" }}
```



```
      labels:
        app: {{ template "roar-db.name" . }}
    spec:
      containers:
        - name: {{ .Chart.Name }}
        {{- if eq ( .Values.stage | upper )  "PROD" }}
        image: "{{- .Values.image.prod.repository }}:{{ .Values.image.prod.tag -}}"
        {{ else if eq ( .Values.stage | upper )  "TEST" }}
        image: "{{- .Values.image.test.repository }}:{{ .Values.image.test.tag -}}"
        {{ else }}
        image: "{{- fail "A valid .Values.stage entry required!" }}"
        {{ end -}}
```

Or

```
$ meld charts/roar-db/templates/deployment.yaml ../extra/lab3-deployment2.yaml
```



deployment.yaml — lab3-deployment2.yaml
Meld File Edit Changes View
Save Undo
deployment.y...yment2.yaml
deployment.yaml
lab3-deployment2.yaml
Ln 19, Col 1 INS

deployment.yaml

```
  name: {{ template "roar-db.name" . }}
  labels:
    app: {{ template "roar-db.name" . }}
    chart: {{ .Chart.Name }}-{{ .Chart.Version
    release: {{ .Release.Name }}
  namespace: {{ .Values.namespace }}
spec:
  replicas: {{ .Values.replicaCount }}
  template:
    metadata:
      labels:
        app: {{ template "roar-db.name" . }}
    spec:
      containers:
      - name: {{ .Chart.Name }}
        {{- if eq .Values.stage "PROD" }}
        image: "{{- .Values.image.prod.reposit
        {{ else if eq .Values.stage "TEST" }}
        image: "{{- .Values.image.test.reposit
        {{ else }}
        image: "{{- fail "A valid .Values.stag
        {{ end -}}
        imagePullPolicy: Always
        ports:
        - name: {{ .Values.deployment.ports.na
          containerPort: {{ .Values.deployment
        env:
          {{- include "roar-all.environment-va
```

lab3-deployment2.yaml

```
  name: {{ template "roar-db.name" . }}
  labels:
    app: {{ template "roar-db.name" . }}
    chart: {{ .Chart.Name }}-{{ .Chart.Version
    release: {{ .Release.Name }}
  namespace: {{ .Values.namespace }}
spec:
  replicas: {{ .Values.replicaCount }}
  template:
    metadata:
      labels:
        app: {{ template "roar-db.name" . }}
    spec:
      containers:
      - name: {{ .Chart.Name }}
        {{- if eq ( .Values.stage | upper ) "P
        image: "{{- .Values.image.prod.reposit
        {{ else if eq ( .Values.stage | upper
        image: "{{- .Values.image.test.reposit
        {{ else }}
        image: "{{- fail "A valid .Values.stag
        {{ end -}}
        imagePullPolicy: Always
        ports:
        - name: {{ .Values.deployment.ports.na
          containerPort: {{ .Values.deployment
        env:
          {{- include "roar-all.environment-va
```

7. Save your changes and try running the command with the lower-case setting again. This time it should work.

```
$ helm install --set roar-db.stage=test --dry-run FOO . | grep image -n3
```

8. Finally let's try deploying a running test instance of our application and a running instance of the prod version of our application.

```
$ k create ns roar-prod
$ helm install -n roar-prod --set roar-db.stage=PROD roar-prod .
$ k create ns roar-test
$ helm install -n roar-test --set roar-db.stage=TEST roar-test .
```

9. Now, you can get the NodePort for both versions. and open them in a browser to see the results.

```
$ k get svc -n roar-prod | grep web
$ k get svc -n roar-test | grep web
```

10. As a reminder, the port numbers that are > 30000 are the ones you want. You can open each of them in a browser at http://localhost:<nodeport>/roar/ and view the different instances.

Lab 4 – Using Kustomize

Purpose: In this lab, we'll look at how to deploy into Kubernetes with Kustomize, an alternative approach to Helm.

1. Switch to the roar-kz subdirectory and use the tree command to look at the structure.

```
$ cd ~/hk-ws/roar-kz
$ tree
```

2. Notice how we have the base and overlay directories laid out and the occurrences of the *kustomization.yaml* files. Take a look at the kustomization.yaml files in the *base, overlays/stage,* and *overlays/prod* area to see what kind of changes these are layering on top of our specs. What is each one adding/modifying?

```
$ cat base/kustomization.yaml
$ cat overlays/stage/kustomization.yaml
$ cat overlays/prod/kustomization.yaml
```

3. Let's do a kustomize build of the base area and capture the yaml that is produced for Kubernetes. Then let's also do a kustomize build of the overlays/staging area and capture that output.

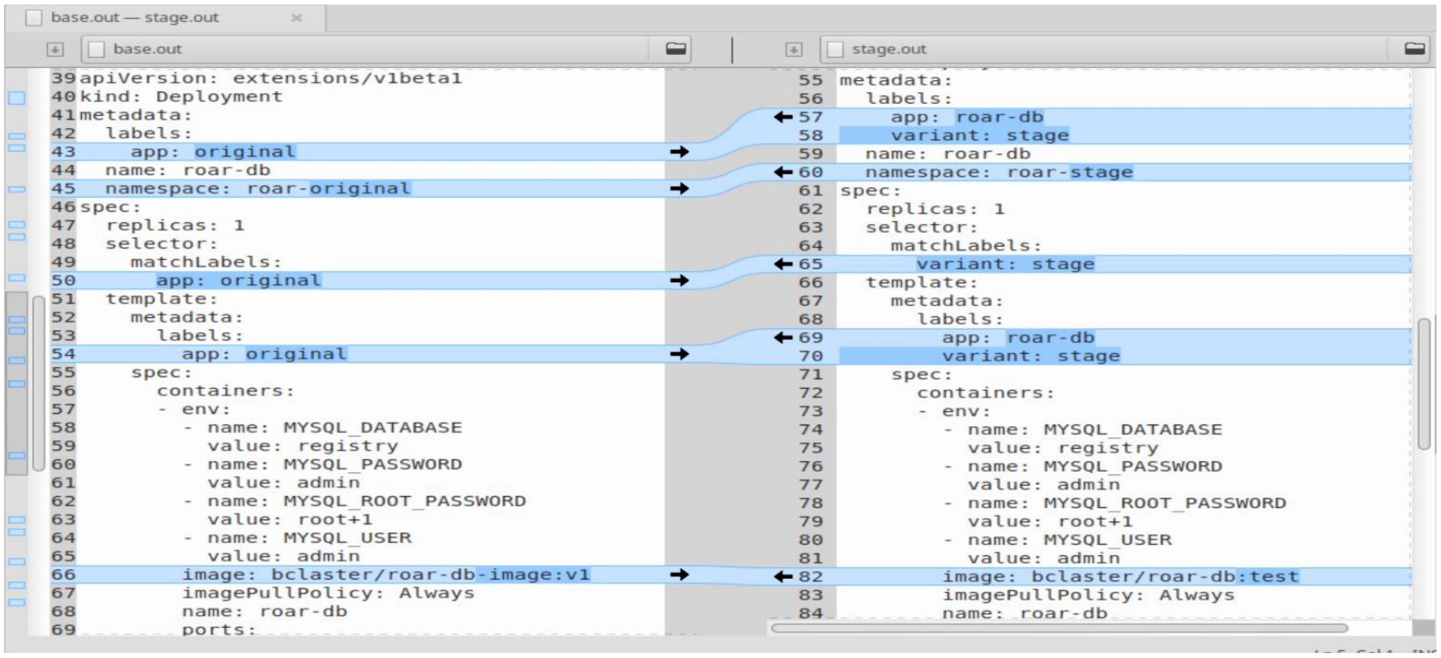
```
$ kz build base > base.out

$ kz build overlays/stage > stage.out
```

4. Let's look at the difference between the kustomize build of base and the kustomize build of stage with the meld tool. What do you see as different?

```
$ meld base.out stage.out
```

(NOTE: You do not need to click any arrows here or make any changes. This is just to look at differences.)

Notice some of the changes in the database deployment spec now as shown in the excerpt below:



```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  labels:
    app: original
  name: roar-db
  namespace: roar-original
spec:
  replicas: 1
  selector:
    matchLabels:
      app: original
  template:
    metadata:
      labels:
        app: original
    spec:
      containers:
      - env:
        - name: MYSQL_DATABASE
          value: registry
        - name: MYSQL_PASSWORD
          value: admin
        - name: MYSQL_ROOT_PASSWORD
          value: root+1
        - name: MYSQL_USER
          value: admin
        image: bclaster/roar-db-image:v1
        imagePullPolicy: Always
        name: roar-db
        ports:
```

```
metadata:
  labels:
    app: roar-db
    variant: stage
  name: roar-db
  namespace: roar-stage
spec:
  replicas: 1
  selector:
    matchLabels:
      variant: stage
  template:
    metadata:
      labels:
        app: roar-db
        variant: stage
    spec:
      containers:
      - env:
        - name: MYSQL_DATABASE
          value: registry
        - name: MYSQL_PASSWORD
          value: admin
        - name: MYSQL_ROOT_PASSWORD
          value: root+1
        - name: MYSQL_USER
          value: admin
        image: bclaster/roar-db:test
        imagePullPolicy: Always
        name: roar-db
```

5. Now, let's go ahead and apply the yaml produced from the kustomize build output to the cluster. We do this by simply piping it through Kubernetes apply.

```
$ kz build overlays/stage | k apply -f -
```

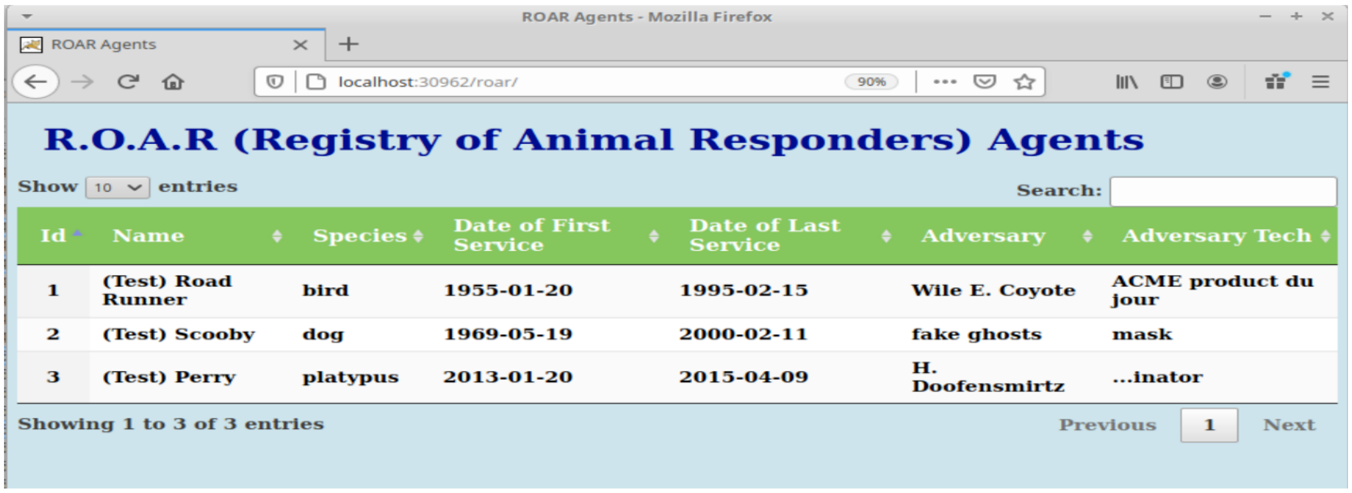
6. The preceding step should have created a new *roar-stage* namespace with a running instance of our app. Find the node for this as you have before and bring up the running instance.

```
$ kubectl -n roar-stage get svc
```

Note that the type of service for roar-web is “NodePort”. As before, this means we have a port open on the Kubernetes node that we can access the service through. Find the nodePort under the PORT(S) column and after the service port (8089) and before the “/TCP”. For example, if we have **8089:31789/TCP** in that column, then the actual nodePort we need is **31789**.

In the web browser, go to the url below, substituting in the nodePort from the step above for “<nodePort>”. You should see the running application with some test data.

http://localhost:<nodePort>/roar/



R.O.A.R (Registry of Animal Responders) Agents

Show 10 entries Search:

Id	Name	Species	Date of First Service	Date of Last Service	Adversary	Adversary Tech
1	(Test) Road Runner	bird	1955-01-20	1995-02-15	Wile E. Coyote	ACME product du jour
2	(Test) Scooby	dog	1969-05-19	2000-02-11	fake ghosts	mask
3	(Test) Perry	platypus	2013-01-20	2015-04-09	H. Doofensmirtz	...inator

Showing 1 to 3 of 3 entries Previous 1 Next

Lab 5 – Working with Kustomize Generators

Purpose: In this lab, we’ll look at how to work with a unique feature in Kustomize - generators - specifically - ConfigMapGenerators.

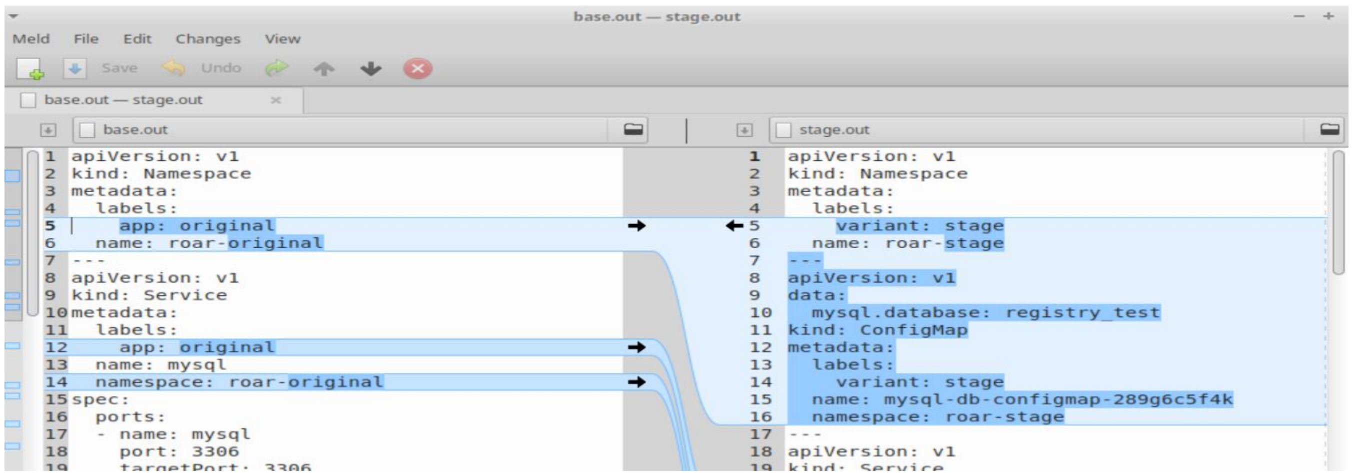
1. In the kustomization.yaml file we used for *overlays/stage*, we created a *ConfigMapGenerator* which made a config map with a hash. The hash acts like a key that the deployment knows about. Take a quick look at how the configmap generator is defined in our yaml file (at the bottom).

```
$ cat overlays/stage/kustomization.yaml
```

2. Then notice how it is rendered (in the staging area only) by looking at the top (right side) of the differences between the base and stage output.

```
$ meld base.out stage.out
```

(NOTE: You do not need to click any arrows here or make any changes. This is just to look at differences.)



```
apiVersion: v1
kind: Namespace
metadata:
  labels:
    app: original
  name: roar-original
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: original
  name: mysql
  namespace: roar-original
spec:
  ports:
  - name: mysql
    port: 3306
    targetPort: 3306
```

```
apiVersion: v1
kind: Namespace
metadata:
  labels:
    variant: stage
  name: roar-stage
---
apiVersion: v1
data:
  mysql.database: registry_test
kind: ConfigMap
metadata:
  labels:
    variant: stage
  name: mysql-db-configmap-289g6c5f4k
  namespace: roar-stage
---
apiVersion: v1
kind: Service
```

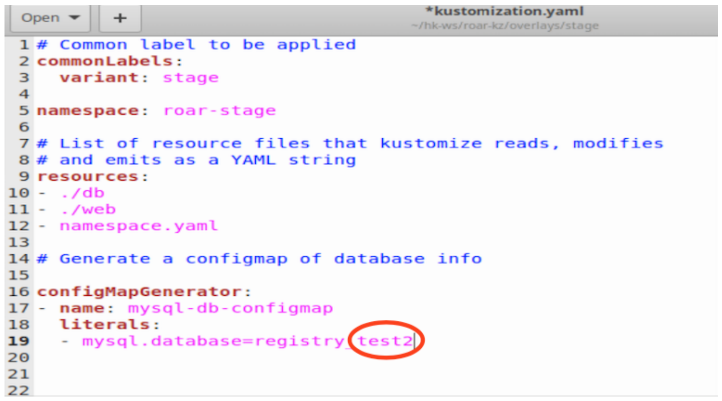
3. Notice the unique identifier value that's added on to it after "mysql-db-configmap". That's the generator part. Take a look at how this configmap is instantiated in the system.

```
$ kubectl describe configmap -n roar-stage
```

4. Now, let's change the configmap by pointing to a different database for testing.

```
$ gedit overlays/stage/kustomization.yaml
```

Then add a "2" onto the last line – change =registry_test to =registry_test2



```
# Common label to be applied
commonLabels:
  variant: stage

namespace: roar-stage

# List of resource files that kustomize reads, modifies
# and emits as a YAML string
resources:
- ./db
- ./web
- namespace.yaml

# Generate a configmap of database info

configMapGenerator:
- name: mysql-db-configmap
  literals:
  - mysql.database=registry test2
```

Save your changes and close the editor.

5. Now let's see what the new generated configmap difference that our change made compared to the previous one. (Note the space followed by a dash at the end of the line.)

```
$ kz build overlays/stage | diff -c3 stage.out -
```

Notice here that a new configmap was rendered AND the web deployment was updated as well.

6. Build and apply the updated configmap.

```
$ kz build overlays/stage | k apply -f -
```

Notice here that a new configmap was created AND the running web deployment was configured (updated) as well.

7. OPTIONAL: Build and apply the kustomization from the prod area which will add in persistent storage.

   Take a look at what's mounted right now (should be nothing).

```
$ ls /mnt
```

   Build and apply.

```
$ kz build overlays/prod | k apply -f -
```

Notice there's a "pv" (persistent volume) object created as well as a "pv-claim" (persistent volume claim) object created.

You'll be able to see the persistent storage at /mnt/data.

```
$ ls /mnt/data
```

END OF LAB