Guided Exercise: Deploy Applications from an Image and from a Template

Deploy a database from a container image and from a template by using the OpenShift command-line interface and compare the resources and attributes that each method generates.

Outcomes

- Deploy a database from a container image.
- Deploy a database from a template.

As the `student` user on the `workstation` machine, use the `lab` command to prepare your system for this exercise. This command ensures that the cluster is accessible and that all resources are available for this exercise.

```
[student@workstation ~]$ lab start deploy-newapp
```

Instructions

1. As the `developer` user, create a project and verify that it is not empty after creation.

   Log in to the OpenShift cluster as the `developer` user with `developer` as the password.

   ```
   [student@workstation ~]$ oc login -u developer -p developer \
     https://api.ocp4.example.com:6443
   Login successful
   ...output omitted...
   ```

   Create a project named `deploy-newapp`.

   ```
   [student@workstation ~]$ oc new-project deploy-newapp
   Now using project "deploy-newapp" on server "https://api.ocp4.example.com:6443".
   ...output omitted...
   ```

   The new project is automatically selected.

   Verify that no pod resources exist in the `deploy-newapp` project.

   ```
   [student@workstation ~]$ oc get pods
   No resources found in deploy-newapp namespace.
   ```

   Verify that the new project contains other types of resources, such as service accounts and secrets.

   ```
   [student@workstation ~]$ oc get serviceaccounts,secrets
   NAME                       SECRETS   AGE
   serviceaccount/builder     1         20s
   serviceaccount/default     1         20s
   serviceaccount/deployer    1         20s

   NAME                               TYPE                       DATA   AGE
   secret/builder-dockercfg-sczxg     kubernetes.io/dockercfg    1      15s
   secret/default-dockercfg-gsnqj     kubernetes.io/dockercfg    1      15s
   secret/deployer-dockercfg-6f8nm    kubernetes.io/dockercfg    1      15s
   ```

2. Create two PostgreSQL instances by using the `oc new-app` command with different options.

   View the `mysql-persistent` template definition to inspect the resources that it creates. Specify the project that houses the template by using the `-n openshift` option.

   ```
   [student@workstation ~]$ oc describe template mysql-persistent -n openshift
   Name:           mysql-persistent
   Namespace:      openshift
   ...output omitted...
   Objects:
       Secret                                  ${DATABASE_SERVICE_NAME}
       Service                                 ${DATABASE_SERVICE_NAME}
       PersistentVolumeClaim                   ${DATABASE_SERVICE_NAME}
       Deployment.apps                         ${DATABASE_SERVICE_NAME}
   ```

   The `objects` attribute specifies several resource definitions that are applied on using the template. These resources include one of each of the following types: secret, service (`svc`), persistent volume claim (`pvc`), and deployment.

Create an instance by using the `mysql-persistent` template. Specify the application name to `mysql`, and attach a custom `team=red` label to the created resources.

```
[student@workstation ~]$ oc new-app --name mysql \
  --template mysql-persistent \
  -l team=red \
  -p MYSQL_USER=developer \
  -p MYSQL_PASSWORD=developer
...output omitted...
--> Creating resources with label team=red ...
    secret "mysql" created
    service "mysql" created
    persistentvolumeclaim "mysql" created
    deployment.apps "mysql" created
--> Success
...output omitted...
```

The template creates resources of the types from the preceding step.

Run the `watch` command with the `oc get pods` command, and wait until the pod is running. The name of the pod is different in your cluster.

```
[student@workstation ~]$ watch oc get pods
NAME                   READY   STATUS      RESTARTS  AGE
mysql-8d7d996f-qn798   1/1     Running     0         60s
```

Press **Ctrl**+**C** to end the `watch` command after the pod displays `Running` in the `STATUS` column.

Create an instance by using a container image. Specify a `name` option and attach a custom `team=blue` label to the created resources.

```
[student@workstation ~]$ oc new-app --name db-image -l team=blue \
  --image registry.ocp4.example.com:8443/rhel9/mysql-80:1 \
  -e MYSQL_USER=developer \
  -e MYSQL_PASSWORD=developer \
  -e MYSQL_ROOT_PASSWORD=redhat
...output omitted...
--> Creating resources with label team=blue ...
    imagestream.image.openshift.io "db-image" created
    deployment.apps "db-image" created
    service "db-image" created
--> Success
...output omitted...
```

The command creates predefined resources that are needed to deploy an image. These resource types are image stream (`is`), deployment, and service (`svc`). Image streams and services are discussed in more detail elsewhere in the course.

Run the `watch` command with the `oc get pods` command, and wait until the pods are running. The command lists all pods and the value of the `team` label for each pod. The names of the pods are different in your cluster.

```
[student@workstation ~]$ watch oc get pods -L team
NAME                        READY   STATUS      RESTARTS   AGE    TEAM
db-image-8d4b97594-6jb85    1/1     Running     0          55s    blue
mysql-8d7d996f-qn798        1/1     Running     0          1m30s
```

Press **Ctrl**+**C** to end the `watch` command after the pods display `Running` in the `STATUS` column.

Without a `readinessProbe`, the `db-image` pod shows as ready before the MySQL service is ready for requests. Readiness probes are discussed elsewhere in the course.

Notice that only the `db-image` pod has a label that contains the word `team`. Pods that the `mysql-persistent` template creates do not have the `team=red` label, because the template does not define this label in its pod specification template.

3. Compare the resources that each image and template method creates.

   View the template-based pod and observe that it contains a readiness probe.

```
[student@workstation ~]$ oc get pods -l deployment=mysql \
  -o jsonpath='{.items[0].spec.containers[0].readinessProbe}' | jq
{
  "exec": {
    "command": [
      "/bin/sh",
      "-i",
      "-c",
      "MYSQL_PWD=\"$MYSQL_PASSWORD\" mysqladmin -u $MYSQL_USER ping"
    ]
  },
  "failureThreshold": 3,
  "initialDelaySeconds": 5,
  "periodSeconds": 10,
  "successThreshold": 1,
  "timeoutSeconds": 1
}
```

> **NOTE**
>
> The results of the preceding `oc` command are passed to the `jq` command, which formats the JSON output.

Observe that the image-based pod does not contain a readiness probe.

```
[student@workstation ~]$ oc get pods -l deployment=db-image \
  -o jsonpath='{.items[0].spec.containers[0].readinessProbe}' | jq
```

Observe that the template-based pod has a memory resource limit, which restricts allocated memory to the resulting pods. Resource limits are discussed in more detail elsewhere in the course.

```
[student@workstation ~]$ oc get pods -l deployment=mysql \
  -o jsonpath='{.items[0].spec.containers[0].resources.limits}' | jq
{
  "memory": "512Mi"
}
```

Observe that the image-based pod has no resource limits.

```
[student@workstation ~]$ oc get pods -l deployment=db-image \
  -o jsonpath='{.items[0].spec.containers[0].resources}' | jq
{}
```

Retrieve secrets in the project. Notice that the template produced a secret, whereas the pod that was created with only an image did not produce a secret.

```
[student@workstation ~]$ oc get secrets
NAME                       TYPE     DATA   AGE
...output omitted...
mysql                      Opaque   4      3m
```

4. Explore filtering resources via labels.

   Observe that when a label is not supplied, all services are shown.

```
[student@workstation ~]$ oc get services
NAME       TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
db-image   ClusterIP   172.30.38.113   <none>        3306/TCP   1m30s
mysql      ClusterIP   172.30.95.52    <none>        3306/TCP   2m30s
```

   Observe that when a label is supplied, only the services with the label are shown.

```
[student@workstation ~]$ oc get services -l team=red
NAME    TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)    AGE
mysql   ClusterIP   172.30.95.52   <none>        3306/TCP   2m43s
```

   Observe that not all resources include a label, such as the pods that are created with the template.

```
[student@workstation ~]$ oc get pods -l team=red
No resources found in deploy-newapp namespace.
```

5. Use labels to delete only the resources that are associated with the template-based deployment.

   Delete only the resources that use the `team=red` label by using the `oc delete all -l` command.

```
[student@workstation ~]$ oc delete all -l team=red
service "mysql" deleted
deployment.apps "mysql" deleted
...output omitted...
```

   Because the `oc delete all -l` command does not delete the secret and PVC resources, delete those resources manually by running the following command:

```
[studen@workstation ~]$ oc delete secret,pvc -l team=red
secret "mysql" deleted
persistentvolumeclaim "mysql" deleted
```

   Observe that the resources that the template created are deleted.

```
[student@workstation ~]$ oc get secret,svc,pvc,deployment -l team=red
...output omitted...
No resources found in deploy-newapp namespace.
```

   Observe that the image-based resources remain unchanged.

```
[student@workstation ~]$ oc get is,deployment,svc
NAME                                     IMAGE REPOSITORY              ...
imagestream.image.openshift.io/db-image   image-registry.openshift... ...
NAME                        READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/db-image    1/1     1            1           46m

NAME               TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)    AGE
service/db-image   ClusterIP   172.30.71.0   <none>        3306/TCP   46m
```

Finish

On the `workstation` machine, use the `lab` command to complete this exercise. This step is important to ensure that resources from previous exercises do not impact upcoming exercises.

```
[student@workstation ~]$ lab finish deploy-newapp
```