Chapter 4. Deploy Managed and Networked Applications on Kubernetes

Abstract

Goal	Deploy applications and expose them to network access from inside and outside a Kubernetes cluster.
Sections	• Deploy Applications from an Image and from a Template (and Guided Exercise) • Manage Long-lived and Short-lived Applications by Using the Kubernetes Workload API (and Guided Exercise) • Kubernetes Pod and Service Networks (and Guided Exercise) • Scale and Expose Applications to External Access (and Guided Exercise)
Lab	• Deploy Managed and Networked Applications on Kubernetes

Deploy Applications from an Image and from a Template

Objectives

- Identify the main resources and settings that Kubernetes uses to manage long-lived applications and demonstrate how OpenShift simplifies common application deployment workflows.

Deploying Applications

Microservices and DevOps are growing trends in enterprise software. Containers and Kubernetes gained popularity alongside those trends, and became categories of their own. Container-based infrastructures support most types of traditional and modern applications.

The *application* term can refer to a software system or to a service within it. Given this ambiguity, it is clearer to refer directly to resources, services, and other components.

Resources and Resource Definitions

Kubernetes manages applications, or services, as a loose collection of resources. Resources are configuration pieces for components in the cluster. When you create a resource, the corresponding component is not created immediately. Instead, the cluster is responsible for eventually creating the component.

A *resource type* represents a specific component type, such as a pod. Kubernetes ships with many default resource types, some of which overlap in function. Red Hat OpenShift Container Platform (RHOCP) includes the default Kubernetes resource types, and provides other resource types of its own. To add resource types, you can create or import CRDs.

Managing Resources

You can add, view, and edit resources in various formats, including YAML and JSON format. Traditionally, YAML is the most common format.

You can delete resources in batch by using label selectors or by deleting the entire project or namespace. For example, the following command deletes only deployments with the `app=my-app` label:

```
[user@host ~]$ oc delete deployment -l app=my-app
deployment.apps "my-app" deleted
```

Similar to creation, deleting a resource is not immediate, but is instead a request for eventual deletion.

> **NOTE**
>
> Commands that are executed without specifying a namespace are executed in the user's current namespace.

Common Resource Types and Their Uses

The following resources are standard OpenShift resources:

Templates

Similar to projects, templates are an RHOCP addition to Kubernetes. A template is a YAML manifest that contains parameterized definitions of one or more resources. RHOCP provides predefined templates in the `openshift` namespace.

You can process a template into a list of resources by using the `oc process` command, which replaces values and generates resource definitions. The resulting resource definitions create or update resources in the cluster by supplying them to the `oc apply` command.

For example, the following command processes the `mysql-template.yaml` template file and generates four resource definitions:

```
[user@host ~]$ oc process -f mysql-template.yaml -o yaml
apiVersion: v1
items:
- apiVersion: v1
  kind: Secret
  ...output omitted...
- apiVersion: v1
  kind: Service
  ...output omitted...
- apiVersion: v1
  kind: PersistentVolumeClaim
  ...output omitted...
- apiVersion: apps/v1
  kind: Deployment
  ...output omitted...
```

The `--parameters` option instead displays the parameters of a template. For example, the following command lists the parameters of the `mysql-template.yaml` file.

```
[user@host ~]$ oc process -f mysql-template.yaml --parameters
NAME                    DESCRIPTION ...output omitted...
...output omitted...
MYSQL_USER              Username for MySQL user   ...output omitted...
MYSQL_PASSWORD          Password for the MySQL connection    ...output omitted...
MYSQL_ROOT_PASSWORD     Password for the MySQL root user.    ...output omitted...
MYSQL_DATABASE          Name of the MySQL database accessed. ...output omitted...
VOLUME_CAPACITY         Volume space available for data,     ...output omitted...
```

You can use templates with the `new-app` command from RHOCP. In the following example, the `new-app` command uses the `mysql-persistent` template to create a MySQL application and its supporting resources:

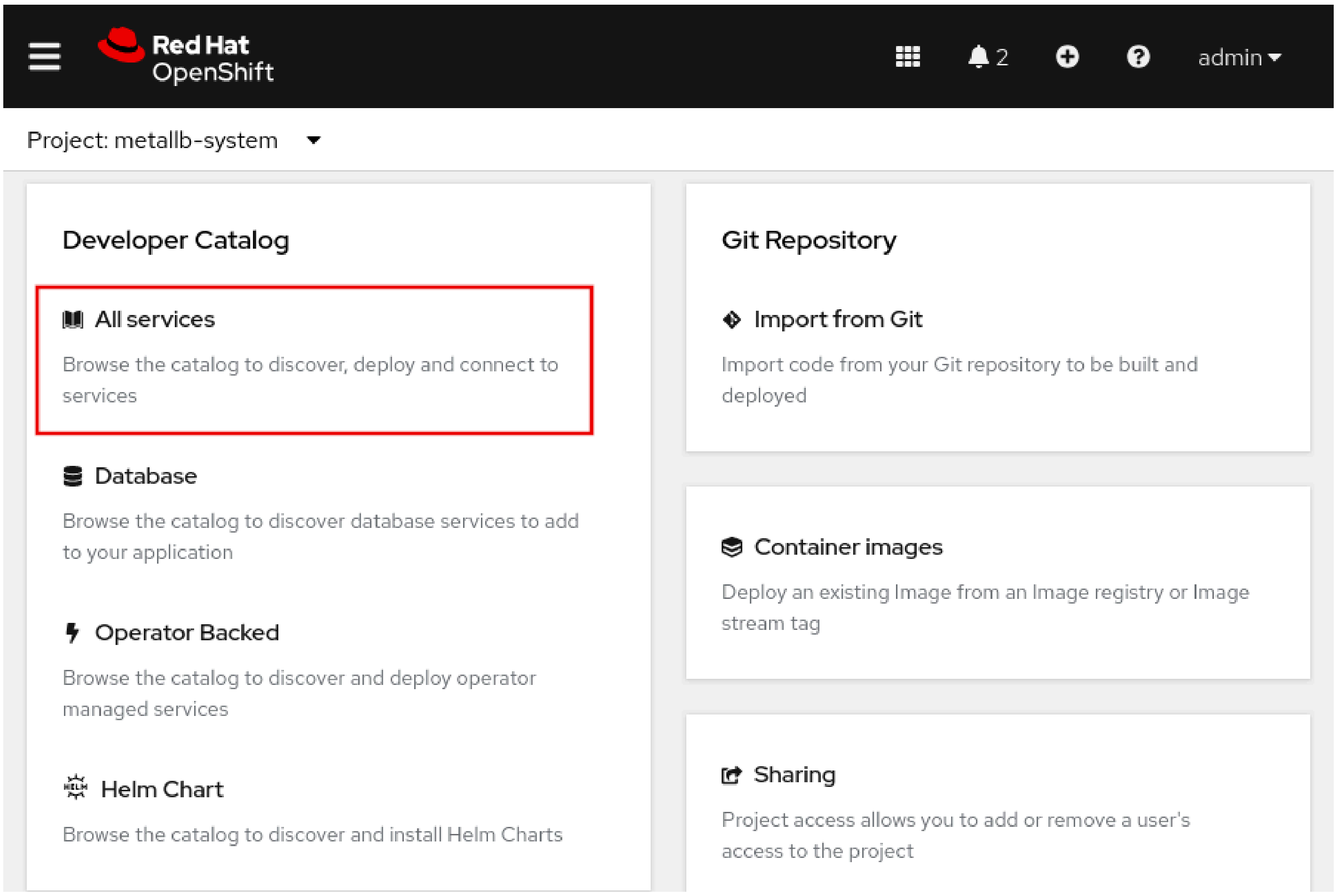
```
[user@host ~]$ oc new-app --template mysql-persistent
--> Deploying template "db-app/mysql-persistent" to project db-app
...output omitted...
     The following service(s) have been created in your project: mysql.

            Username: userQSL
            Password: pyf0yElPvFWYQQou
       Database Name: sampledb
      Connection URL: mysql://mysql:3306/
...output omitted...
     * With parameters:
        * Memory Limit=512Mi
        * Namespace=openshift
        * Database Service Name=mysql
        * MySQL Connection Username=userQSL # generated
        * MySQL Connection Password=pyf0yElPvFWYQQou # generated
        * MySQL root user Password=HHbdurqWO5gAog2m # generated
        * MySQL Database Name=sampledb
        * Volume Capacity=1Gi
        * Version of MySQL Image=8.0-el8

--> Creating resources ... (1)
    secret "mysql" created
    service "mysql" created
    persistentvolumeclaim "mysql" created
    deployment.apps "mysql" created
--> Success
...output omitted...
```

1. Several resources are created to meet the requirements of the deployment, including a secret, a service, and a persistent volume claim.

NOTE

You can specify environment variables to be configured in creating your application.

You can also use templates, helm charts, builder images, or devfiles to create applications from the web console by using the **Developer Catalog** page. From the **Developer** perspective, go to the **+Add** menu and click **All Services** in the **Developer Catalog** section to open the catalog.



Red Hat OpenShift

2 admin

Project: metallb-system

Developer Catalog

All services
Browse the catalog to discover, deploy and connect to services

Database
Browse the catalog to discover database services to add to your application

Operator Backed
Browse the catalog to discover and deploy operator managed services

Helm Chart
Browse the catalog to discover and install Helm Charts

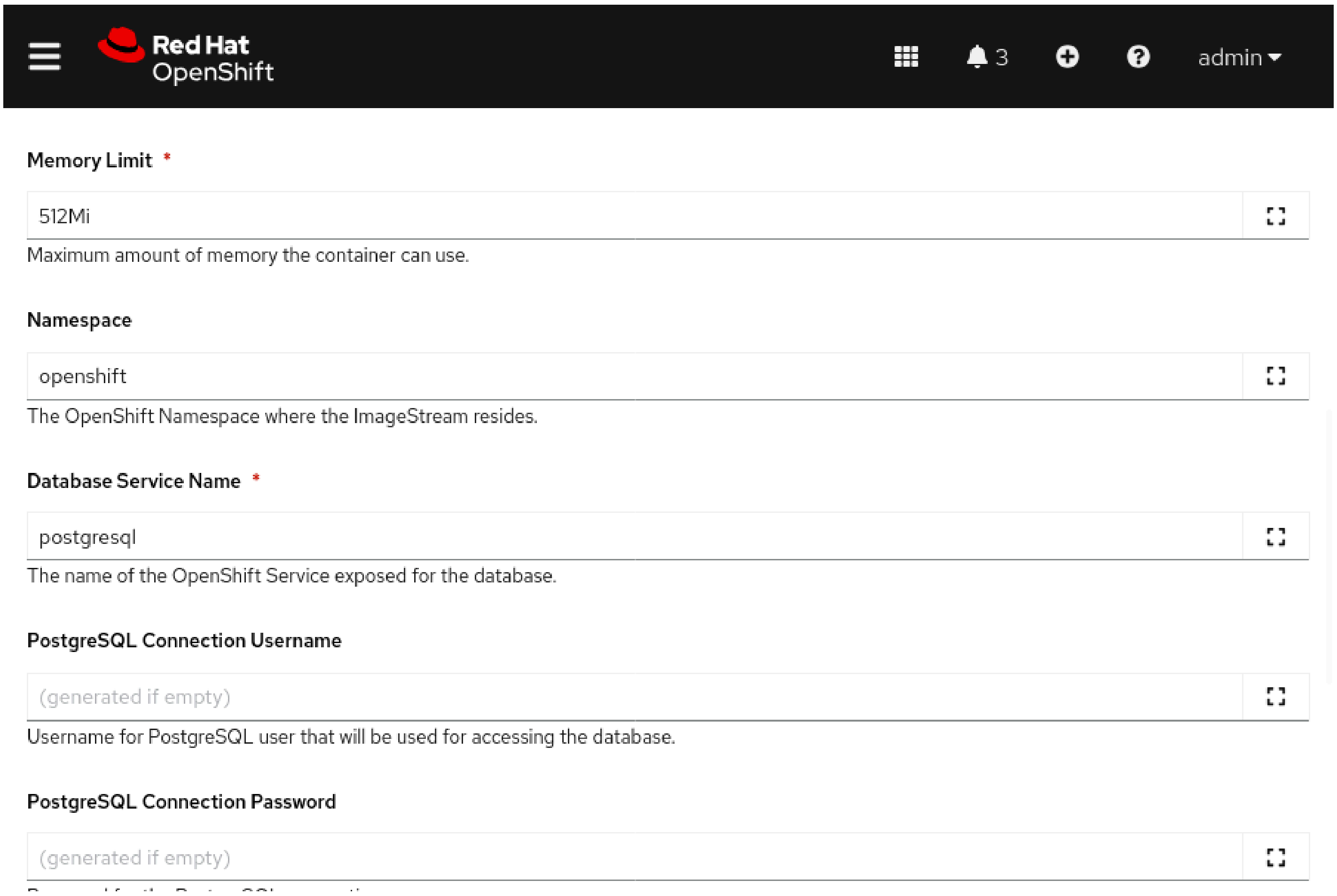
Git Repository

Import from Git
Import code from your Git repository to be built and deployed

Container images
Deploy an existing Image from an Image registry or Image stream tag

Sharing
Project access allows you to add or remove a user's access to the project

Then, enter the application name in the filter box to search for an application's template. You can instantiate a template and change the default values, such as the Git repository, the memory limits, and the application version.



Red Hat OpenShift

3 admin

Memory Limit *
512Mi
Maximum amount of memory the container can use.

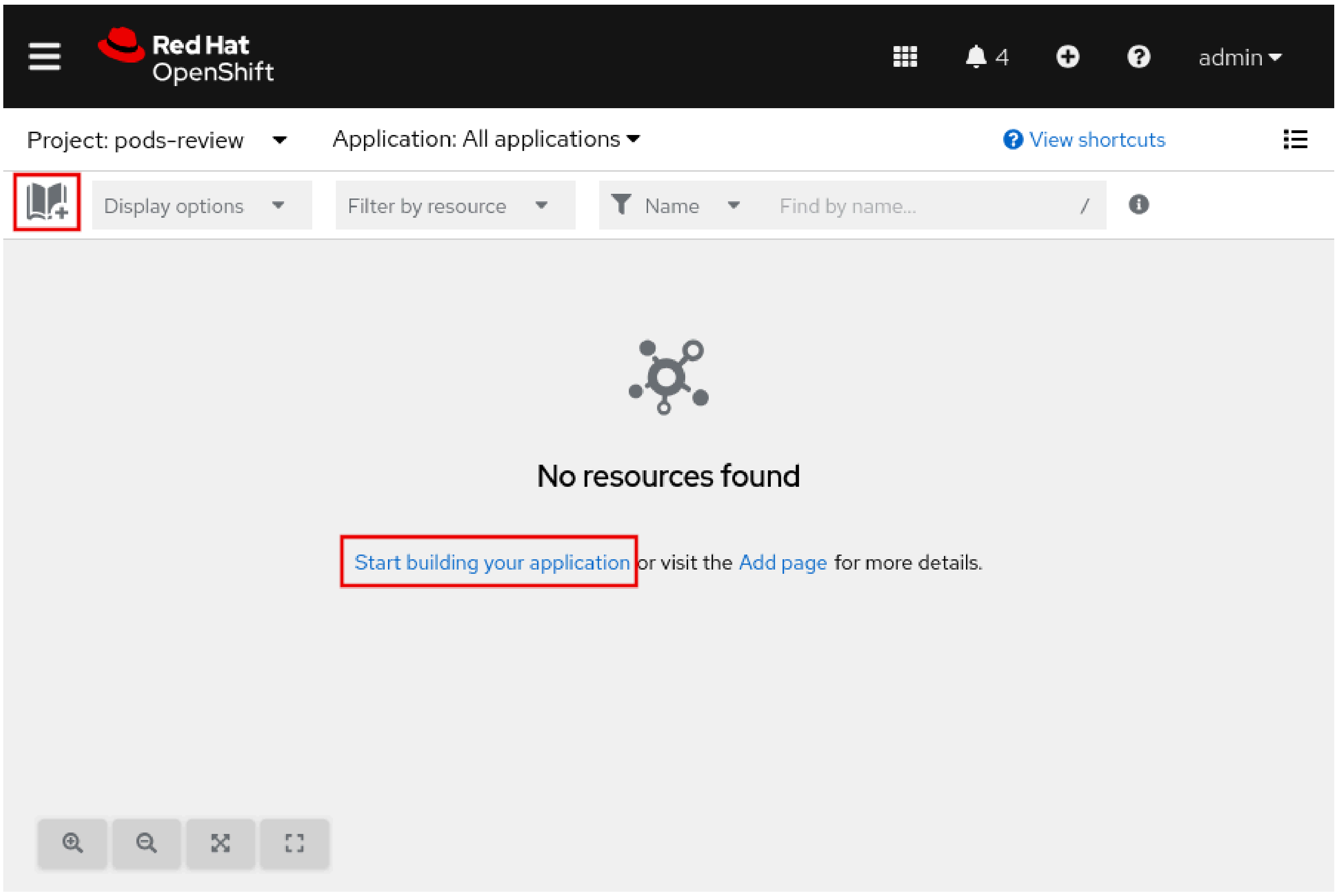
Namespace
openshift
The OpenShift Namespace where the ImageStream resides.

Database Service Name *
postgresql
The name of the OpenShift Service exposed for the database.

PostgreSQL Connection Username
(generated if empty)
Username for PostgreSQL user that will be used for accessing the database.

PostgreSQL Connection Password
(generated if empty)

You can also create an application from the **Topology** menu, and by clicking **Start building application** or clicking the book icon.



Pod

A pod is the smallest compute unit that you can define, deploy, and manage in RHOCP. A pod runs one or more containers that represent a single application. Containers in the pod share resources, such as networking and storage.

The following example shows the definition of a pod:

```
apiVersion: v1
kind: Pod 1
metadata: 2
  annotations: {}
  labels:
    deployment: docker-registry-1
  name: registry
  namespace: pod-registries
spec: 3
  containers:
  - env:
    - name: OPENSHIFT_CA_DATA
      value:
    image: openshift/origin-docker-registry:v0.6.2
    imagePullPolicy: IfNotPresent
    name: registry
    ports:
    - containerPort: 5000
      protocol: TCP
    resources: {}
    securityContext: {}
    volumeMounts:
    - mountPath: /registry
      name: registry-storage
  dnsPolicy: ClusterFirst
  imagePullSecrets:
  - name: default-dockercfg-at06w
  restartPolicy: Always
  serviceAccount: default
  volumes:
  - emptyDir: {}
    name: registry-storage
status: 4
  conditions: {}
```

1 The resource kind is set to `Pod`.

2 Information that describes your application, such as the name, project, attached labels, and annotations.

3 Section where the application requirements are specified, such as the container name, the container image, environment variables, volume mounts, network configuration, and volumes.

4 Indicates the last condition of the pod, such as the last probe time, the last transition time, the status setting as `true` or `false`, and more.

Deployments

A deployment describes the intended state of a component of the application as a pod template. A deployment manages one or more replica sets.

The following example shows the definition of a deployment:

```
apiVersion: apps/v1
kind: Deployment 1
metadata:
  name: hello-openshift 2
spec:
  replicas: 1 3
  selector:
    matchLabels:
      app: hello-openshift
  template: 4
    metadata:
      labels:
        app: hello-openshift
    spec:
      containers:
      - name: hello-openshift 5
        image: openshift/hello-openshift:latest 6
        ports: 7
        - containerPort: 80
```

1 The resource kind is set to `Deployment`.

2 Name of the deployment resource.

3. Number of running instances.
4. Section to define the metadata, labels, and the container information of the deployment resource.
5. Name of the container.
6. Resource of the image for creating the deployment resource.
7. Port configuration, such as the port number, the name of the port, and the protocol.

Projects

RHOCP adds projects to enhance the function of Kubernetes namespaces. A project is a Kubernetes namespace with additional annotations, and is the primary method for managing access to resources for regular users. Projects can be created from templates and must use RBAC for organization and permission management. Administrators must grant cluster users access to a project. If a cluster user is allowed to create projects, then the user automatically has access to their created projects.

Projects provide logical and organizational isolation to separate your application component resources. Resources in one project can access resources in other projects, but not by default.

The following example shows the definition of a project:

```
apiVersion: project.openshift.io/v1
kind: Project 1
metadata:
  name: test 2
spec:
  finalizers: 3
  - kubernetes
```

1. The resource kind is set to `Project`.
2. Name of the project.
3. A finalizer is a special metadata key that tells Kubernetes to wait until a specific condition is met before it fully deletes a resource.

Services

You can configure internal pod-to-pod network communication in RHOCP by using a service. Applications send requests to the service name and port. RHOCP provides a virtual network, which reroutes such requests to the pods that the service targets by using labels.

The following example shows the definition of a service:

```
apiVersion: v1
kind: Service 1
metadata:
  name: docker-registry 2
  namespace: test 3
spec:
  selector:
    app: MyApp 4
  ports:
  - protocol: TCP 5
    port: 80 6
    targetPort: 9376 7
```

1. The resource kind is set to `Service`.
2. Name of the service.
3. Project name where the service resource exists.
4. The label selector identifies all pods with the attached `app=MyApp` label and adds the pods to the service endpoints.
5. Internet protocol is set to `TCP`.
6. Port that the service listens on.
7. Port on the backing pods, which the service forwards connections to.

Persistent Volume Claims

RHOCP uses the Kubernetes persistent volume (PV) framework to enable cluster administrators to provision persistent storage for a cluster. Developers can use persistent volume claims (PVCs) to request PV resources without having specific knowledge of the underlying storage infrastructure. After a PV is bound to a PVC, that PV cannot then be bound to additional PVCs. Because PVCs are namespaced objects and PVs are globally scoped objects, this binding effectively scopes a bound PV to a single namespace until the binding PVC is deleted.

The following example shows the definition of a persistent volume claim:

```
apiVersion: v1
kind: PersistentVolumeClaim ❶
metadata:
  name: mysql-pvc ❷
spec:
  accessModes:
    - ReadWriteOnce ❸
  resources:
    requests:
      storage: 1Gi ❹
  storageClassName: nfs-storage ❺
status: {}
```

❶ The resource kind is set to `PersistentVolumeClaim`.

❷ Name of the service.

❸ The access mode, to define the read/write and mount permissions.

❹ The storage of the PVC.

❺ Name of the StorageClass that the claim requires.

Secrets

Secrets provide a mechanism to hold sensitive information, such as passwords, private source repository credentials, sensitive configuration files, and credentials to an external resource, such as an SSH key or an OAuth token. You can mount secrets into containers by using a volume plug-in. Kubernetes can also use secrets to perform actions, such as on pods, declaring environment variables. Secrets can store any type of data. Kubernetes and OpenShift support different types of secrets, such as service account tokens, SSH keys, and TLS certificates.

The following example shows the definition of a secret:

```
apiVersion: v1
kind: Secret ❶
metadata:
  name: example-secret ❷
  namespace: my-app ❸
type: Opaque ❹
data: ❺
  username: bXl1c2VyCg==
  password: bXlQQDU1Cg==
stringData: ❻
  hostname: myapp.mydomain.com
  secret.properties: |
    property1=valueA
    property2=valueB
```

❶The resource kind is set to `Secret`.

❷Name of the service.

❸Project name where the service resource exists.

❹Specifies the type of secret.

❺Specifies the encoded string and data.

❻Specifies the decoded string and data.

Managing Resources from the Command Line

Many Kubernetes and RHOCP commands can create and modify cluster resources. Some commands are part of core Kubernetes, whereas others are exclusive additions to RHOCP.

The resource management commands generally fall into one of two categories:

a. An *imperative* command instructs what the cluster does.

b. A *declarative* command defines the state that the cluster attempts to match.

Imperative Resource Management

The `create` command is an imperative way to create resources, and is included in both of the `oc` and `kubectl` commands. For example, the following command creates a deployment named `my-app` that creates pods that are based on the specified image.

```
[user@host ~]$ oc create deployment my-app --image example.com/my-image:dev
deployment.apps/my-app created
```

Use the `set` command to define attributes on a resource, such as environment variables. For example, the following command adds the `TEAM=red` environment variable to the preceding deployment.

```
[user@host ~]$ oc set env deployment/my-app TEAM=red
deployment.apps/my-app updated
```

Another imperative approach to creating a resource is the `run` command. In the following example, the `run` command creates the `example-pod` pod.

```
[user@host ~]$ oc run example-pod \ 1
 --image=registry.access.redhat.com/ubi8/httpd-24 \ 2
 --env GREETING='Hello from the awesome container' \ 3
 --port 8080 4
pod/example-pod created
```

1 The pod `.metadata.name` definition

2 The image for the single container in this pod

3 The environment variable for the single container in this pod

4 The port metadata definition

The imperative commands are a faster way of creating pods, because such commands do not require a pod object definition. Versioning and incremental changes require declarative manifests.

Generally, developers test a deployment by using imperative commands, and then use the imperative commands to generate the pod object definition. Use the `--dry-run=client` option to avoid creating the object in RHOCP. Additionally, use the `-o yaml` or `-o json` option to configure the definition format.

The following command is an example of generating the YAML definition for the `example-pod` pod:

```
[user@host ~]$ oc run example-pod \
 --image=registry.access.redhat.com/ubi8/httpd-24 \
 --env GREETING='Hello from the awesome container' \
 --port 8080 \
 --dry-run=client -o yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: example-pod
  name: example-pod
spec:
  containers:
...output omitted...
```

Managing resources in this way is imperative, because you are instructing the cluster what to do rather than declaring the intended outcomes.

Declarative Resource Management

Using the `create` command does not guarantee that you are creating resources imperatively. For example, providing a manifest declaratively creates the resources that are defined in the YAML file, such as in the following command.

```
[user@host ~]$ oc create -f my-app-deployment.yaml
deployment.apps/my-app created
```

RHOCP adds the `new-app` command, which provides another declarative way to create resources. This command uses heuristics to automatically determine which types of resources to create based on the specified parameters. For example, the following command deploys the `my-app` application by creating several resources, including a deployment resource, from a YAML manifest file:

```
[user@host ~]$ oc new-app --file=./example/my-app.yaml
...output omitted...
--> Creating resources ...
    imagestream.image.openshift.io "my-app" created
    deployment.apps "my-app" created
    services "my-app" created
...output omitted...
```

In both of the preceding `create` and `new-app` examples, you are declaring the intended state of the resources, and so they are declarative.

You can also use the `new-app` command with templates for resource management. A template describes the intended state of resources that must be created for an application to run, such as deployments and services. Supplying a template to the `new-app` command is a form of declarative resource management.

The `new-app` command also includes options, such as the `--param` option, that customize an application deployment declaratively. For example, when the `new-app` is used with a template, you can include the `--param` option to override a parameter value in the template.

```
[user@host ~]$ oc new-app --template mysql-persistent \
 --param MYSQL_USER=operator --param MYSQL_PASSWORD=myP@55 \
 --param MYSQL_DATABASE=mydata \
 --param DATABASE_SERVICE_NAME=db
--> Deploying template "db-app/mysql-persistent" to project db-app
...output omitted...
     The following service(s) have been created in your project: db.

            Username: operator
            Password: myP@55
       Database Name: mydata
      Connection URL: mysql://db:3306/
...output omitted...
     * With parameters:
        * Memory Limit=512Mi
        * Namespace=openshift
        * Database Service Name=db
        * MySQL Connection Username=operator
        * MySQL Connection Password=myP@55
        * MySQL root user Password=tlH8BThuVgnIrCon # generated
        * MySQL Database Name=mydata
        * Volume Capacity=1Gi
        * Version of MySQL Image=8.0-el8

--> Creating resources ...
    secret "db" created
    service "db" created
    persistentvolumeclaim "db" created
--> Success
...output omitted...
```

Similar to the `create` command, you can use the `new-app` command imperatively. When using a container image with the `new-app` command, you are instructing the cluster what to do, rather than declaring the intended outcomes. For example, the following command deploys the `example.com/my-app:dev` image by creating several resources, including a deployment resource:

```
[user@host ~]$ oc new-app --image example.com/my-app:dev
...output omitted...
--> Creating resources ...
    imagestream.image.openshift.io "my-app" created
    deployment.apps "my-app" created
...output omitted...
```

You can also supply a Git repository to the `new-app` command. The following command creates an application named `httpd24` by using a Git repository:

```
[user@host ~]$ oc new-app https://github.com/apache/httpd.git#2.4.56
...output omitted...
--> Creating resources ...
    imagestream.image.openshift.io "httpd24" created
    deployment.apps "httpd24" created
...output omitted...
```

Retrieving Resource Information

You can supply the `all` argument to commands, to specify a list of common resource types. However, the `all` option does not include every resource type. Instead, `all` is a shorthand form for a predefined subset of types. When you use this command argument, ensure that `all` includes any intended types to address.

You can view detailed information about a resource, such as the defined parameters, by using the `describe` command. For example, RHOCP provides templates in the `openshift` project to use with the `oc new-app` command. The following example command displays detailed information about the `mysql-ephemeral` template:

```
[user@host ~]$ oc describe template mysql-ephemeral -n openshift
Name:           mysql-ephemeral
Namespace:      openshift
...output omitted...
Parameters:
    Name:               MEMORY_LIMIT
    Display Name:       Memory Limit
    Description:        Maximum amount of memory the container can use.
    Required:           true
    Value:              512Mi

    Name:               NAMESPACE
    Display Name:       Namespace
    Description:        The OpenShift Namespace where the ImageStream resides.
    Required:           false
    Value:              openshift
...output omitted...
Objects:
    Secret                                                      ${DATABASE_SERVICE_NAME}
    Service                                                     ${DATABASE_SERVICE_NAME}
```

The `oc describe` command cannot generate structured output, such as the YAML or JSON formats. A structured format is required for the `oc describe` command to parse or filter the output with tools such as JSONPath or Go templates." Instead, use the `oc get` command to generate and to parse the structured output of a resource.

REFERENCES

OpenShift Container Platform Documentation - Understanding Deployments

OpenShift Container Platform Documentation - Working with Projects

OpenShift Container Platform Documentation - Creating Applications Using the CLI

OpenShift Container Platform Documentation - Using Pods