Chapter 6

Creating Applications from OpenShift Templates

Goal	Describe the elements of a template and create a multicontainer application template.
Objectives	• Describe the elements of an OpenShift template. • Build a multicontainer application from a custom template.
Sections	• Describing the Elements of an OpenShift Template (and Quiz) • Creating a Multicontainer Template (and Guided Exercise)
Lab	Creating Applications from OpenShift Templates

Describing the Elements of an OpenShift Template

Objectives

After completing this section, you should be able to describe the elements of an OpenShift template.

Describing a Template

An OpenShift *template* is a YAML or JSON file consisting of a set of OpenShift resources. Templates define *parameters* that are used to customize the resource configuration. OpenShift processes templates by replacing parameter references with values and creating a customized set of resources.

OpenShift templates offer similar functionality to Kubernetes Helm charts. You can use both Helm charts or OpenShift templates to manage your multi-container applications similarly. However, Helm charts are not in the scope of this course.

A template is useful when you want to deploy a set of resources as a single unit, rather than deploying them individually. Example use cases for when to use a template include:

- An independent software vendor (ISV) provides a template to deploy their product on OpenShift. The template contains configuration details of the containers that make up the product, along with a deployment configuration such as the number of replicas, services and routes, persistent storage configuration, health checks, and resource limits for compute resources such as CPU, memory, and I/O.
- Your multitier application consists of a number of separate components, such as a web server, an application server, and a database, and you would like to deploy these components together as a single unit on OpenShift to simplify the process of deploying the application in a staging environment. This will allow your QA and acceptance testing teams to rapidly provision applications for testing.

Template Syntax

The syntax of an OpenShift template follows the general syntax of an OpenShift resource, but with the **`objects`** attribute in place of the **`spec`** attribute. A template usually includes **`parameters`** and **`labels`** attributes, as well as annotations in its metadata.

The following listing shows a sample template definition in YAML format and illustrates the main syntax elements. As with any OpenShift resource, you can also define templates in JSON syntax:

```
apiVersion: template.openshift.io/v1
kind: Template ❶
metadata:
  name: mytemplate
  annotations:
    description: "Description" ❷
objects: ❸
- apiVersion: v1
  kind: Pod
```

```
    metadata:
      name: myapp
    spec:
      containers:
      - env:
        - name: MYAPP_CONFIGURATION
          value: ${MYPARAMETER} ❹
        image: myorganization/myapplication
        name: myapp
        ports:
        - containerPort: 80
          protocol: TCP
  parameters: ❺
  - description: Myapp configuration data
    name: MYPARAMETER
    required: true
  labels: ❻
    mylabel: myapp
```

❶ Template Resource type
❷ Optional annotations for use by OpenShift tools
❸ Resource list
❹ Reference to a template parameter
❺ Parameter list
❻ Label list

The resource list of a template usually includes other resources like build configurations, deployment configurations, persistent volume claims (PVC), services, and routes.

Any attribute in the resource list can reference the value of any parameter.

You can define custom labels for a template. OpenShift adds these labels to all resources created by the template.

When a template defines multiple resources, it is essential to consider the order in which these resources are defined to accommodate dependencies between resources. OpenShift does not report an error if a resource references a dependent resource that does not exist.

A process that is triggered by a resource might fail if it starts before a dependent resource. One example is a build configuration that references an image stream as the output image, and your template defines that image stream after the build configuration. In this scenario, it is possible that the build configuration starts a build before the image stream exists.

Defining Template Parameters

The previous example template included the definition of a required parameter. Both optional and required parameters can provide default values. For example:

```
parameters:
- description: Myapp configuration data
  name: MYPARAMETER
  value: /etc/myapp/config.ini
```

OpenShift can generate random default values for parameters. This is useful for secrets and passwords:

```
parameters:
- description: ACME cloud provider API key
  name: APIKEY
  generate: expression
  from:"[a-zA-Z0-9]{12}"
```

The syntax for generated values is a subset of the Perl regular expression syntax. See the references at the end of this section for the full template parameter expression syntax.

Adding a Template to OpenShift

To add a template to OpenShift, use either the **`oc create`** command or the web console (using the import YAML page) to create the template resource from the template definition file, the same way you would for any other kind of resource.

It is a common practice to create projects that hold only templates because OpenShift allows users to easily share templates between multiple users and projects. These projects usually include other potentially shared resources, such as image streams.

A default installation of Red Hat OpenShift Container Platform provides several templates in the **`openshift`** project. All OpenShift cluster users have read access to the **`openshift`** project, but only cluster administrators have permission to create or delete templates in this project.

Browsing Templates with the OpenShift CLI

OpenShift provides a number of templates in the **`openshift`** namespace by default. You can create applications from these templates, as well as customize the provided templates to suit the needs of your application.

To view the list of provided templates, use the **`oc get templates`** command:

```
[user@host ~]$ oc get templates -n openshift
```

To view the list of parameters provided by the template as well as a brief description, use the **`oc process`** command. For example, to view the parameters for a Node.js and MongoDB template provided by OpenShift:

```
[user@host ~]$ oc process --parameters -n openshift nodejs-mongodb-example
```

To export the template as a YAML file to use as a base for your own custom template, use the **`oc get`** command. For example, to export the **`nodejs-mongodb-example`** template provided by OpenShift in YAML format:

```
[user@host ~]$ oc get template nodejs-mongodb-example -o yaml -n openshift
```

You can also use the **`oc describe template`** command to get a more detailed description of a template. For example, to describe the **`nodejs-mongodb-example`** template provided by OpenShift:

```
[user@host ~]$ oc describe template nodejs-mongodb-example -n openshift
  Name:  nodejs-mongodb-example  1
  Namespace: openshift
  Created: 6 days ago
  Labels:  samples.operator.openshift.io/managed=true
...output omitted...
  Annotations: iconClass=icon-nodejs
      openshift.io/display-name=Node.js + MongoDB (Ephemeral)
      openshift.io/documentation-url=https://github.com/sclorg/nodejs-ex
...output omitted...
  Parameters:    2
      Name:  NAME
      Display Name: Name
      Description: The name assigned to all of the frontend objects defined in
 this template.
      Required:  true
      Value:  nodejs-mongodb-example

      Name:  NAMESPACE
      Display Name: Namespace
      Description: The OpenShift Namespace where the ImageStream resides.
      Required:  true
      Value:  openshift

      Name:  NODEJS_VERSION
      Display Name: Version of NodeJS Image
      Description: Version of NodeJS image to be used (6, 8, or latest).
      Required:  true
      Value:  8
...output omitted...
  Object Labels: app=nodejs-mongodb-example,template=nodejs-mongodb-example 3

  Message: The following service(s) have been created in your project: ${NAME},
 ${DATABASE_SERVICE_NAME}.

      For more information about using this template, including OpenShift
 considerations, see https://github.com/sclorg/nodejs-ex/blob/master/README.md.

  Objects:    4
      Secret  ${NAME}
      Service  ${NAME}
      Route  ${NAME}
      ImageStream  ${NAME}
      BuildConfig  ${NAME}
      DeploymentConfig ${NAME}
      Service  ${DATABASE_SERVICE_NAME}
      DeploymentConfig ${DATABASE_SERVICE_NAME}
```

1 The name of the template.

2 A list of parameters the template offers, including whether they are required and any default values.

3 A list of labels that OpenShift applies to all resources it creates from the template.

4 A summary of all the resources that OpenShift creates from the template.

References

Further information is available in the *Using Templates* chapter in the documentation for Red Hat OpenShift Container Platform 4.5; at
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html/images/using-templates

Quiz

Describing the Elements of an OpenShift Template

Choose the correct answers to the following questions:

1. What is the purpose of an OpenShift template?

a. To describe the resource configuration for a single container.

b. To encapsulate a set of OpenShift resources for reuse.

c. To provide custom configurations for an OpenShift cluster.

d. To customize the appearance of the web console.

2. Which two of the following approaches define a template parameter as optional? (Choose two.)

a. Set the **`required`** attribute to **`false`**.

b. Set the **`required`** attribute to **`not`**.

c. Omit the **`required`** attribute.

d. Set the **`required`** attribute to **`expression`**.

e. Set the **`from`** attribute to a regular expression.

3. Which of the following commands returns the number of parameters and objects provided by a template?

a. **`oc export`**.

b. **`oc describe`**.

c. **`oc create`**.

d. **`oc get`**.

e. None of the above.

4. Which three statements describe valid locations to insert template parameter references? (Choose three.)

a. As the name of an environment variable inside a pod resource.

b. As the key for a label in all resources created by the template.

c. As the value for the **`host`** attribute in a route resource.

d. As the value for an annotation in the template.

e. As the value of an environment variable inside a pod resource.

Solution

Describing the Elements of an OpenShift Template

Choose the correct answers to the following questions:

1. What is the purpose of an OpenShift template?

a. To describe the resource configuration for a single container.
b. To encapsulate a set of OpenShift resources for reuse.
c. To provide custom configurations for an OpenShift cluster.
d. To customize the appearance of the web console.

2. Which two of the following approaches define a template parameter as optional? (Choose two.)

a. Set the `required` attribute to `false`.
b. Set the `required` attribute to `not`.
c. Omit the `required` attribute.
d. Set the `required` attribute to `expression`.
e. Set the `from` attribute to a regular expression.

3. Which of the following commands returns the number of parameters and objects provided by a template?

a. `oc export`.
b. `oc describe`.
c. `oc create`.
d. `oc get`.
e. None of the above.

4. Which three statements describe valid locations to insert template parameter references? (Choose three.)

a. As the name of an environment variable inside a pod resource.
b. As the key for a label in all resources created by the template.
c. As the value for the `host` attribute in a route resource.
d. As the value for an annotation in the template.
e. As the value of an environment variable inside a pod resource.

Creating a Multicontainer Template

Objectives

After completing this section, you should be able to build a multicontainer application from a custom template.

Creating a Template from Application Resources

A typical scenario for using OpenShift templates is to deploy an application that requires multiple sets of pods, each one with a deployment configuration, services, and other auxiliary resources such as secrets and persistent volume claims.

Consider an application that connects to a database to store and manage data. A template for this application would include:

- A deployment configuration and a service for the application, as well as another deployment configuration and service for the database.
- Two image streams to point to the container images: one for the application, and another for the database.
- A secret for database access credentials.
- A persistent volume claim for storing the database data.
- A route for external access to the application.

You can create the required template in several different ways. The most common approaches are discussed in the next section.

Manually Creating Resource Files

If you are comfortable with the syntax of YAML or JSON files, and you are familiar with the attributes for each of the OpenShift resource types, you can create templates by hand.

The general process for this approach is described below.

1. Create a basic template skeleton with basic attributes such as a name, tags, and other metadata information and declare it as an OpenShift **`Template`** resource.
2. Look at existing templates that come installed with OpenShift by default, and customize them to your needs by cutting and pasting relevant resources for your application.
3. To further customize the template, use the **`oc explain`** and **`oc api-resources`** commands to view the attributes for each resource type that you want to customize and then add it to your template.

Concatenating Resource Files

The **`oc new-app`** command can create a resource definition file, instead of creating resources in the current project, by using the **`-o`** option. You can concatenate multiple resource definition files inside the resources list of a skeleton template definition file.

The general process for this approach is described below.

1. Create a basic template skeleton with basic attributes such as a name, tags, and other metadata information and declare it as an OpenShift **`Template`** resource.
2. Use the **`oc new-app`** command with the **`-o`** option to export the resource configuration to a file.
3. Clean the runtime attributes from the generated resource file.
4. Cut and paste the resource list into a skeleton template file.

Exporting Existing Resources

The **`oc get`** command can create a resource definition file by using the **`-o`** and **`--export`** options together. The **`-o`** option takes either **`yaml`** or **`json`** as a parameter, and exports the resource definition in YAML or JSON format respectively. After you export a set of resources to a template file, you can add annotations and parameters as desired.

The general process for this approach is described below.

1. Create a basic template skeleton with basic attributes such as a name, tags, and other metadata information and declare it as an OpenShift **`Template`** resource.
2. Use the **`oc get`** command with the **`-o --export`** options to export the resource configuration to a file.
3. Clean the runtime attributes from the generated resource file.
4. Cut and paste the resource list into a skeleton template file.

When you use the **`oc get`** command to export resources, select only the necessary resource types. Include only resources that you would need to deploy an application. Do not include derived resources such as builds, deployments, replication controllers, or pods.

The order in which you list the resources in the **`oc get`** command is important. You need to export any dependent resources first, and then the resources that depend on them. For example, you need to export image streams before the build configurations and deployment configurations that reference those image streams.

The following example creates a YAML file containing different types of resources in the current project:

```
[user@host ~]$ oc get -o yaml --export is,bc,dc,svc,route > mytemplate.yaml
```

Depending on your needs, add more resource types to the previous command. For example, add **`secret`** before **`bc`** and **`dc`**. It is safe to add **`pvc`** to the end of the list of resource types because a deployment waits for a persistent volume claim to bind.

Cleaning and Editing Templates

The **`oc get`** and **`oc new-app`** commands do not generate resource definitions that are ready for use in a template. These resource definitions contain runtime information that a template does not need, and some of it could prevent the template from working at all. Examples of runtime information are attributes such as **`status`**, **`creationTimeStamp`**, **`image`**, and **`uid`**, as well as most annotations that start with the **`openshift.io/generated-by`** prefix.

Some resource types, such as secrets, require special handling. It is not possible to initialize key values inside the **`data`** attribute using template parameters. The **`data`** attribute from a secret resource needs to be replaced by the **`stringData`** attribute, and all key values need to be unencoded.

Another example is image stream resources. OpenShift caches container images from external registries using the internal registry, and the exported image stream points to the internal registry. If you use this image stream resource in a template, it fails because the container image does not exist in the current project. You need to change the image stream to point to the external registry.

Describing OpenShift Resource Types

To create a template from scratch, or to clean a resource file generated by the **`oc get`** command, you need to know which attributes contain runtime information. This is explained in the Red Hat OpenShift Container Platform product documentation. See the references at the end of this section. Another way is to use the **`oc explain`** command.

With a resource type as an argument, the **`oc explain`** command lists the top-level attributes for that resource type:

```
[user@host ~]$ oc explain routes
DESCRIPTION:
A route allows developers to expose services through an HTTP(S) aware load
 balancing and proxy layer via a public DNS entry.
...output omitted...
FIELDS:
   apiVersion <string>
     ...output omitted...

   kind <string>
     ...output omitted...

   metadata <Object>
     Standard object metadata.

   spec <Object> -required-
     spec is the desired state of the route
...output omitted...
```

Most resource types have many levels of attributes. Use dot notation to request attributes from lower levels:

```
[user@host ~]$ oc explain routes.spec
RESOURCE: spec <Object>

DESCRIPTION:
     spec is the desired state of the route
     ...output omitted...
FIELDS:
   to <Object> -required-
     to is an object the route should use as the primary backend.
     ...output omitted...

   wildcardPolicy <string>
```

```
       Wildcard policy if any for the route. Currently only 'Subdomain' or 'None'
       is allowed.
...output omitted...
```

Use the **`oc api-resources`** command for a complete list of supported resources in the OpenShift instance.

Creating an Application from a Template

You can deploy an application directly from a template resource definition file. The **`oc new-app`** command and the **`oc process`** command use the template file as input and process it to apply parameters and create resources.

You can also pass a template file to the **`oc create`** command to create a template resource in OpenShift. If the template is meant to be reused by multiple developers, it is better to create the template resource in a shared project. If the template is meant to be used by a single deployment, it is better to keep it in a file.

Whereas the **`oc new-app`** command creates resources from the template, the **`oc process`** command creates a resource list from the template. You need to either save the resource list generated by the **`oc process`** command to a file or pass it as input to the **`oc create`** command to create the resources from the list.

For both the **`oc new-app`** command and the **`oc process`** command, each parameter value has to be provided using a different **`-p`** option. Another option that both commands accept is the **`-l`** option, which adds a label to all resources created from the template.

The following is an example **`oc new-app`** command that deploys an application from a template file:

```
[user@host ~]$ oc new-app --file mytemplate.yaml -p PARAM1=value1 \
> -p PARAM2=value2
```

The following is an example **`oc process`** command that deploys an application from a template file:

```
[user@host ~]$ oc process -f mytemplate.yaml -p PARAM1=value1 \
> -p PARAM2=value2 > myresourcelist.yaml
```

The file generated by the previous example would then be given to the **`oc create`** command:

```
[user@host ~]$ oc create -f myresourcelist.yaml
```

You can combine the previous two examples using a pipe:

```
[user@host ~]$ oc process -f mytemplate.yaml -p PARAM1=value1 \
> -p PARAM2=value2 | oc create -f -
```

Red Hat recommends using the **`oc new-app`** command rather than the **`oc process`** command. You can use the **`oc process`** command with the **`-f`** option to list only those parameters defined by the specified template:

```
[user@host ~]$ oc process -f mytemplate.yaml --parameters
```

Note

There is no section in the OpenShift 4.5 web console to list or view templates. Nevertheless, you can list available templates in **Administrator** → **Home** → **Search** and **Developer** → **Search** pages.

You cannot create applications from custom templates in the OpenShift 4.5 web console by default. Cluster administrators can install additional templates into the Developer Catalog. Templates in the Developer Catalog can be used to create new applications.

References

Further information about creating and using templates is available in the *Using Templates* chapter of the Red Hat OpenShift Container Platform documentation at
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.5/html/images/using-templates

Guided Exercise

Creating a Multicontainer Template

In this exercise, you will create an OpenShift template that deploys a multicontainer application. The template includes an HTTP API, a database, and persistent storage.

Outcomes

You should be able to:

- Create a template from a running application using the **`oc get`** command.
- Clean the template to remove runtime information.
- Add parameters to the template.
- Create a new application from the template using the OpenShift command-line tool (**`oc`**).

Before You Begin

To perform this exercise, ensure you have access to:

- A running OpenShift cluster.
- The PHP 7.2 S2I builder image, and the MySQL 5.7 database image required by the template.
- The sample application (quotes) in the Git repository.

Run the following command on the **`workstation`** VM to validate the prerequisites, provision persistent storage, deploy the multicontainer application to the ***`youruser`*****`-quotes-dev`** project, and download the required files to complete this exercise:

```
[student@workstation ~]$ lab create-template start
```

To review how the application is deployed, refer to the **`new-app-db.sh`**, **`add-vol.sh`**, **`new-app-php.sh`**, and **`add-route.sh`** scripts in the **`~/DO288/labs/create-template`** folder.

1. Inspect the quotes application deployed to the ***`youruser`*****`-quotes-dev`** project.

   1.1. Load your classroom environment configuration.

   Run the following command to load the environment variables created in the first guided exercise:

```
[student@workstation ~]$ source /usr/local/etc/ocp4.config
```

   1.2. Log in to OpenShift using your developer user account:

```
[student@workstation ~]$ oc login -u ${RHT_OCP4_DEV_USER} \
> -p ${RHT_OCP4_DEV_PASSWORD} ${RHT_OCP4_MASTER_API}
Login successful.
...output omitted...
```

1.3. Set ***youruser*-quotes-dev** as your current active project:

```
[student@workstation ~]$ oc project ${RHT_OCP4_DEV_USER}-quotes-dev
```

1.4. The database and application are already deployed. Wait for the build to complete and for both the database and application pods to be ready and running:

```
[student@workstation ~]$ oc status
In project youruser-quotes-dev on server
https://api.cluster.domain.example.com:6443

http://quotesapi-youruser-quotes-dev.apps.cluster.domain.example.com to pod port
 8080-tcp (svc/quotesapi)
dc/quotesapi deploys istag/quotesapi:latest <-
  bc/quotesapi source builds https://github.com/youruser/DO288-apps on openshift/
php:7.2
  deployment #1 deployed 28 seconds ago - 1 pod

svc/quotesdb - 172.30.239.22:3306
  dc/quotesdb deploys openshift/mysql:5.7
    deployment #2 deployed about a minute ago - 1 pod
    deployment #1 failed about a minute ago: newer deployment was found running
...output omitted...
```

The previous output shows that the database is deployed from a container image, and the application is deployed from source code.

1.5. Verify that the project includes a persistent volume claim:

```
[student@workstation ~]$ oc get pvc
NAME              STATUS   VOLUME       CAPACITY   ACCESS MODES   ...
quotesdb-claim    Bound    pvc-df1c...  1Gi        RWO            ...
```

1.6. Verify that the project includes a route:

```
[student@workstation ~]$ oc get route/quotesapi -o jsonpath='{.spec.host}{"\n"}'
quotesapi-youruser-quotes-dev.apps.cluster.domain.example.com
```

Do not try to test the application. It will fail because the database is not initialized. The only use of the ***youruser*-quotes-dev** project is to export its resources to a template.

2. Create the template definition file. Start with a basic template definition file and then export the required resources one by one, clean out the runtime information, and then copy the cleaned resource configuration to the template.

Create a new file called **`quotes-template-clean.yaml`** in the **`/home/student`** directory. Add the following YAML snippet to declare this resource definition file as an OpenShift template:

```
apiVersion: template.openshift.io/v1
kind: Template
metadata:
  name: quotes
  annotations:
    openshift.io/display-name: Quotes Application
    description: The Quotes application provides an HTTP API that returns a
 random, funny quote.
    iconClass: icon-php
    tags: php,mysql
objects:
```

You can also copy the YAML snippet from the **`~/DO288/solutions/create-template/new-template.yaml`** file.

Warning
Ensure that the **`description`** attribute value is in a single continuous line with no line breaks. You will get errors for improperly formatted YAML files when the template is parsed later in the exercise.

3. Export the resources in the project one by one, starting with the image stream resources.

3.1. Export the image stream:

```
[student@workstation ~]$ oc get -o yaml --export is > /tmp/is.yaml
```

3.2. Make a copy of the exported file and clean the runtime attributes from it:

```
[student@workstation ~]$ cp /tmp/is.yaml /tmp/is-clean.yaml
```

3.3. Clean the runtime attributes from the **`/tmp/is-clean.yaml`** file.

A copy of the cleaned file is available at **`~/DO288/labs/create-template/is-clean.yaml`**. Compare your cleaned file against this version and make the necessary edits to make them the same.

Remove the first two lines from the file:

```
apiVersion: v1
items:
```

3.4. There are two image stream resources in the exported file. The first is for the quotesapi application, and the second for the quotesdb database. Remove the entire image stream resource for the quotesdb database. The deployment configuration for the database will create the image stream automatically.

Starting with the second occurrence of the image stream resource, that is:

```
- apiVersion: image.openshift.io/v1
kind: ImageStream
metadata:
  annotations:
    openshift.io/image.dockerRepositoryCheck: "2019-06-21T02:54:22Z"
  creationTimestamp: "2019-06-21T02:54:22Z"
  generation: 2
  name: quotesdb

    ...output omitted...
```

Delete all lines beginning from the above listed snippet to the end of the file.

3.5. The following steps involve cleaning the image stream resource for the quotesapi application only.

Remove the **openshift.io/generated-by**, **creationTimestamp**, **generation**, **namespace**, **resourceVersion**, **selfLink**, and **uid** attributes.

Remove the **managedFields**, and **status** attributes including their child attributes.

4. Export and clean the build configuration.

4.1. Export the build configuration:

```
[student@workstation ~]$ oc get -o yaml --export bc > /tmp/bc.yaml
```

4.2. Make a copy of the exported file and clean the runtime attributes from it:

```
[student@workstation ~]$ cp /tmp/bc.yaml /tmp/bc-clean.yaml
```

Clean the runtime attributes from the **/tmp/bc-clean.yaml** file.

A copy of the cleaned file is available at **~/DO288/labs/create-template/bc-clean.yaml**. Compare your cleaned file against this version and make the necessary edits to make them the same.

Ignore differences in the **secret** and **uri** attributes. These values will be different for you.

4.3. Remove the first two lines of the file:

```
apiVersion: v1
items:
```

4.4. Remove the **openshift.io/generated-by**, **creationTimestamp**, **generation**, **namespace**, **resourceVersion**, **selfLink**, and **uid** attributes.

4.5. Remove the **managedFields** attribute and all its child attributes.

4.6. Remove the **namespace** attribute that refers to the **youruser-quotes-dev** project. Do not remove the **namespace: openshift** reference under the **sourceStrategy** attribute lower down in the file.

4.7. Remove the **lastTriggeredImageID** attribute under **imageChange**.

4.8. Remove the **status** attribute and all its child attributes at the bottom of the file.

4.9. Remove the **kind: List** attribute at the bottom of the file, and all other attributes under it.

5. Export and clean the deployment configuration for the quotesapi application and the quotesdb database.

5.1. Export the deployment configuration:

```
[student@workstation ~]$ oc get -o yaml --export dc > /tmp/dc.yaml
```

5.2. Make a copy of the exported file and clean the runtime attributes from it:

```
[student@workstation ~]$ cp /tmp/dc.yaml /tmp/dc-clean.yaml
```

5.3. Clean the runtime attributes from the **/tmp/dc-clean.yaml** file.

A copy of the cleaned file is available at **~/DO288/labs/create-template/dc-clean.yaml**. Compare your cleaned file against this version and make the necessary edits to make them the same.

5.4. Remove the first two lines of the file:

```
apiVersion: v1
items:
```

5.5. Remove all references to **openshift.io/generated-by**, **creationTimestamp**, **generation**, **resourceVersion**, **selfLink**, and **uid** attributes in the file.

5.6. Remove the **namespace** attribute that refers to the **youruser-quotes-dev** project. Do not remove the **namespace: openshift** reference under the **imageChangeParams** attribute lower down in the file.

5.7. Remove the **managedFields** attribute and all its child attributes.

5.8. Remove all references to **image** and **lastTriggeredImage** attributes in the file.

5.9. Remove the **status** attribute and all its child attributes for both deployment configuration resources.

5.10. Remove the **kind: List** attribute at the bottom of the file, and all other attributes under it.

6. Export and clean the service configuration for the quotesapi application and the quotesdb database.

6.1. Export the service configuration:

```
[student@workstation ~]$ oc get -o yaml --export svc > /tmp/svc.yaml
```

6.2. Make a copy of the exported file and clean the runtime attributes from it:

```
[student@workstation ~]$ cp /tmp/svc.yaml /tmp/svc-clean.yaml
```

6.3. Clean the runtime attributes from the **/tmp/svc-clean.yaml** file.

A copy of the cleaned file is available at **~/DO288/labs/create-template/svc-clean.yaml**. Compare your cleaned file against this version and make the necessary edits to make them the same.

6.4. Remove the first two lines of the file:

```
apiVersion: v1
items:
```

6.5. Remove all references to **openshift.io/generated-by**, **creationTimestamp**, **namespace**, **resourceVersion**, **selfLink**, and **uid** attributes in the file.

6.6. Remove the **managedFields** attribute and all its child attributes.

6.7. Remove all references to the **clusterIP** attribute under the **spec** attribute in the file.

6.8. Remove the **status** attribute and all its child attributes for both service resources.

6.9. Remove the **kind: List** attribute at the bottom of the file, and all other attributes under it.

7. Export and clean the route configuration for the quotesapi application.

7.1. Export the route configuration:

```
[student@workstation ~]$ oc get -o yaml --export route > /tmp/route.yaml
```

7.2. Make a copy of the exported file and clean the runtime attributes from it:

```
[student@workstation ~]$ cp /tmp/route.yaml /tmp/route-clean.yaml
```

7.3. Clean the runtime attributes from the **/tmp/route-clean.yaml** file.

A copy of the cleaned file is available at **~/DO288/labs/create-template/route-clean.yaml**. Compare your cleaned file against this version and make the necessary edits to make them the same.

7.4. Remove the first two lines of the file:

```
apiVersion: v1
items:
```

7.5. Remove all references to **openshift.io/generated-by**, **creationTimestamp**, **namespace**, **resourceVersion**, **selfLink**, and **uid** attributes in the file.

7.6. Remove the **managedFields** attribute and all its child attributes.

7.7. Remove the **host** and **subdomain** attributes under the **spec** attribute.

7.8. Remove the **status** attribute and all its child attributes.

7.9. Remove the **kind: List** attribute at the bottom of the file, and all other attributes under it.

8. Export and clean the persistent volume configuration (PVC) for the quotesdb database.

8.1. Export the persistent volume configuration:

```
[student@workstation ~]$ oc get -o yaml --export pvc > /tmp/pvc.yaml
```

8.2. Make a copy of the exported file and clean the runtime attributes from it:

```
[student@workstation ~]$ cp /tmp/pvc.yaml /tmp/pvc-clean.yaml
```

8.3. Clean the runtime attributes from the **/tmp/pvc-clean.yaml** file.

A copy of the cleaned file is available at **~/DO288/labs/create-template/pvc-clean.yaml**. Compare your cleaned file against this version and make the necessary edits to make them the same.

8.4. Remove the first two lines of the file:

```
apiVersion: v1
items:
```

8.5. Remove all attributes under the **metadata.annotations** attribute.

8.6. Remove all references to **creationTimestamp**, the **finalizers** attribute and its children, as well as the **namespace**, **resourceVersion**, **selfLink**, and **uid** attributes in the file.

8.7. Remove the **managedFields** attribute and all its child attributes.

8.8. Remove the **dataSource** attribute under the **spec** attribute.

8.9. Remove the **storageClassName**, **volumeMode**, and **volumeName** attributes under the **spec** attribute.

8.10. Remove the **status** attribute and all its child attributes.

8.11. Remove the **kind: List** attribute at the bottom of the file, and all other attributes under it.

9. Copy the individual resource configuration YAML snippets from the cleaned files into the **/home/student/quotes-template-clean.yaml** file under the **objects** attribute in the following order:

```
[student@workstation ~]$ cat /tmp/is-clean.yaml >> ~/quotes-template-clean.yaml
[student@workstation ~]$ cat /tmp/bc-clean.yaml >> ~/quotes-template-clean.yaml
[student@workstation ~]$ cat /tmp/dc-clean.yaml >> ~/quotes-template-clean.yaml
[student@workstation ~]$ cat /tmp/svc-clean.yaml >> ~/quotes-template-clean.yaml
[student@workstation ~]$ cat /tmp/route-clean.yaml >> ~/quotes-template-clean.yaml
[student@workstation ~]$ cat /tmp/pvc-clean.yaml >> ~/quotes-template-clean.yaml
```

A copy of the final cleaned template is available at **~/DO288/solutions/create-template/quotes-template-clean.yaml**. Compare your final cleaned template against this file.

Ignore differences in the **secret** and **uri** attributes. These values will be different for you. You will parameterize these values in the next step.

10. Add parameters to make the template reusable.

To keep this guided exercise short, you will add only three parameters: one for the Git URL where the application source code is stored, and two for passwords and secrets.

10.1. Inspect the file that contains the parameter definition. Notice that only the **`APP_GIT_URL`** parameter is required. The **`PASSWORD`** and **`SECRET`** parameters have random default values:

```
[student@workstation ~]$ cat ~/DO288/labs/create-template/parameters.yaml
parameters:
- name: APP_GIT_URL
  displayName: Application Source Git URL
  description: The Git URL of the application source code
  required: true
- name: PASSWORD
  displayName: Database Password
  description: Password to access the database
  generate: expression
  from: '[a-zA-Z0-9]{16}'
- name: SECRET
  displayName: Webhook Secret
  description: Secret for webhooks
  generate: expression
  from: '[a-zA-Z0-9]{40}'
```

10.2. Make a copy of the final cleaned template YAML file before adding the parameters:

```
[student@workstation ~]$ cp ~/quotes-template-clean.yaml ~/quotes-template.yaml
```

10.3. Open the **`quotes-template.yaml`** file in a text editor. Copy the contents of the **`parameters.yaml`** file to the end of the **`quotes-template.yaml`** file.

10.4. Change the **`secret`**, **`password`**, and **`uri`** attributes to reference template parameters.

Use the following listing as a guide to make the changes:

```
...output omitted...
      kind: BuildConfig
      ...output omitted...
      name: quotesapi
      ...output omitted...
      source:
      contextDir: quotes
      git:
        uri: ${APP_GIT_URL}
      type: Git
      ...output omitted...
      triggers:
      - github:
        secret: ${SECRET}
      type: GitHub
      - generic:
        secret: ${SECRET}
      ...output omitted...
```

```
    kind: DeploymentConfig
    ...output omitted...
    name: quotesapi
    ...output omitted...
        - name: DATABASE_PASSWORD
          value: ${PASSWORD}
    ...output omitted...
    kind: DeploymentConfig
    ...output omitted...
    name: quotesdb
    ...output omitted...
        - name: MYSQL_PASSWORD
          value: ${PASSWORD}
    ...output omitted...
```

10.5. To make the MySQL database container persistent, you need to add a **volumeMounts** attribute to the database deployment configuration, and reference the persistent volume claim resource declared in the template.

Add the following lines to the quotesdb deployment configuration as follows:

```
...output omitted...
      terminationMessagePath: /dev/termination-log
      terminationMessagePolicy: File
      volumeMounts:
      - mountPath: /var/lib/mysql/data
        name: quotesdb-volume-1
  dnsPolicy: ClusterFirst
  restartPolicy: Always
...output omitted...
```

10.6. The template file is now complete. Save your edits.

To verify the changes you made during this step, review the **quotes-template.yaml** file in the **~/DO288/solutions/create-template** folder. If you are uncertain about your edits, you can copy the solution file and continue to the next step.

11. Create a new application from the template.

11.1. Create the ***youruser*-myquotes** project:

```
[student@workstation ~]$ oc new-project ${RHT_OCP4_DEV_USER}-myquotes
Now using project "youruser-myquotes" on server
"https://api.cluster.domain.example.com:6443"
...output omitted...
```

12. Create a new instance of the quotes application from the template.

12.1. Use the **oc new-app** command to create a new instance of the quotes application from the template:

```
[student@workstation ~]$ oc new-app --file=quotes-template.yaml \
> -p APP_GIT_URL=https://github.com/${RHT_OCP4_GITHUB_USER}/DO288-apps \
> -p PASSWORD=mypass
```

```
--> Deploying template "youruser-myquotes/quotes" to project youruser-myquotes

Quotes Application
------------------
The Quotes application provides an HTTP API that returns a random, funny quote.

* With parameters:
 * Application Source Git URL=https://github.com/youruser/DO288-apps
 * Database Password=mypass
 * Webhook Secret=D6xlih2F3KIyYwsIXs3nLysGHJbi6JLhtaul6I10 # generated

--> Creating resources ...
imagestream.image.openshift.io "quotesapi" created
buildconfig.build.openshift.io "quotesapi" created
deploymentconfig.apps.openshift.io "quotesapi" created
deploymentconfig.apps.openshift.io "quotesdb" created
service "quotesapi" created
service "quotesdb" created
route.route.openshift.io "quotesapi" created
persistentvolumeclaim "quotesdb-claim" created
--> Success
...output omitted...
```

Note

If you see an error at this step, ensure that there are no extra line breaks, and that your template YAML file is properly indented. OpenShift displays the line number in the template where processing failed. Use this information to identify and fix the issues.

12.2. Wait for the build to complete and for the quotesdb and quotesapi applications to have one pod each ready and running.

```
[student@workstation ~]$ oc get pods
NAME                 READY   STATUS     RESTARTS   AGE
...output omitted...
quotesapi-1-tphzw    1/1     Running    0          3m12s
quotesdb-1-rk8mw     1/1     Running    0          4m28s
```

13. Populate the database and test the application.

13.1. Inspect the script that populates the database.

The **populate-db.sh** script in the **~/DO288/labs/create-template** directory uses a port-forwarding tunnel to connect to the database pod and a local MySQL client to run an SQL script:

```
[student@workstation ~]$ cat ~/DO288/labs/create-template/populate-db.sh
...output omitted...
echo 'Creating tunnel...'
pod=$(oc get pod -l deploymentconfig=quotesdb -o name)
oc port-forward $(basename ${pod}) 30306:3306 &
tunnel=$!
sleep 3
```

```
echo 'Initializing the database...'
mysql -h127.0.0.1 -P30306 -uquoteapp -pmypass quotesdb \
< ~/DO288/labs/create-template/quote.sql
sleep 3
echo 'Terminating tunnel...'
kill ${tunnel}
```

13.2. Run the **populate-db.sh** script:

```
[student@workstation ~]$ ~/DO288/labs/create-template/populate-db.sh
Creating tunnel...
Forwarding from 127.0.0.1:30306 -> 3306
Forwarding from [::1]:30306 -> 3306
Initializing the database...
Handling connection for 30306
Terminating tunnel...
```

13.3. Get the route host name for the application:

```
[student@workstation ~]$ oc get route/quotesapi -o jsonpath='{.spec.host}{"\n"}'
quotesapi-youruser-quotes-dev.apps.cluster.domain.example.com
```

13.4. Use the **curl** command and the host name from the previous step to access the application. Your output might be different from this example:

```
[student@workstation ~]$ curl \
> quotesapi-${RHT_OCP4_DEV_USER}-myquotes.${RHT_OCP4_WILDCARD_DOMAIN}/get.php
Veni, vidi, vici...
```

14. Clean up. Delete the projects from OpenShift and release the persistent storage allocated for this exercise.

Open a terminal window on the **workstation** VM and run the following command to perform the cleanup tasks:

```
[student@workstation ~]$ lab create-template finish
```

This concludes the guided exercise.

▶ Lab

Creating Applications from OpenShift Templates

Performance Checklist

In this lab, you will deploy the To Do List application from a template that you will complete for reuse.

Outcomes

You should be able to complete a template that deploys a database pod and a web application pod, use the template to deploy the application, and verify that the application works.

Before You Begin

To perform this exercise, ensure you have access to:

- A running OpenShift cluster.
- The sample application (todo-single) in the Git repository.
- The npm dependencies required by the application (**`restify`**, **`sequelize`**, and **`mysql`**).
- The Node.js 10 S2I builder image, and the MySQL 5.7 database image required by the template.

Run the following command on the **`workstation`** VM to validate the prerequisites and to download the starter project files and solution files:

```
[student@workstation ~]$ lab review-template start
```

Requirements

The application is a To Do List application written in JavaScript. It consists of a web front end, based on the AngularJS web framework, and an HTTP API back end, based on Node.js. The back end uses the Restify and Sequelize frameworks.

Both the application front end and back end run on the same container. The application is deployed from source code stored in a Git repository. A MySQL database is used as the data store. The application initializes the database on startup.

The template takes the following parameters:

- **`APP_GIT_URL`**: The Git URL where the source code for the To Do List application is stored.
- **`HOSTNAME`**: The host name used to access the To Do List application.
- **`NPM_PROXY`**: The URL of the NPM repository server.
- **`SECRET`**: The secret for OpenShift webhooks.

- **PASSWORD**: The database connection password. The user name is fixed in the template.
- **CLEAN_DATABASE**: A flag that indicates whether the application initializes the database on startup.

The template parameters have the following restrictions:

- The **APP_GIT_URL**, **HOSTNAME**, **NPM_PROXY**, and **CLEAN_DATABASE** parameters are required.
- The **SECRET** and **PASSWORD** parameters have random default values.
- The **CLEAN_DATABASE** parameter has the default value of **true**.

A starter template file called **todo-template.yaml** is provided in the **~/DO288/labs/review-template** folder. It contains the resources required to deploy the application. The resources are in the correct order, and already cleaned up. You are required to add the missing parameters and add references to the parameters in the resource list.

Deploy the application according to the following requirements:

- The application project is called **youruser-review-template**.
- Use the **~/DO288/labs/review-template/oc-new-app.sh** script to deploy the application. Edit the script as needed to pass any required parameters.
- The application initializes the database on startup.
- The password to access the database is **mypass**.
- Npm modules required to build the application are available from:

  ```
  http://nexus-common.apps.cluster.domain.example.com/repository/nodejs
  ```

- If you prefer to test the application with a web browser, use the web front end to add some entries to the To Do List application interactively. The application user interface is accessible from the following URL:

  ```
  http://youruser-todo.apps.cluster.domain.example.com/todo/index.html
  ```

- If you prefer to test the application using the command line, use the following URL to submit an HTTP GET request to the application back end:

  ```
  http://youruser-todo.apps.cluster.domain.example.com/todo/api/items-
  count
  ```

  The reply includes a **count** attribute, even if there are no entries in the application. An error reply means the database was not initialized correctly.

Steps

1. Review the **todo-template.yaml** starter template file in the **~/DO288/labs/review-template** folder. Make a copy of this file in the **/home/student/** directory, and use this copy to make your changes throughout the lab. Identify and add any missing parameters at the end of the file. Identify any parameter references missing from the template resource list and add them as needed. Use the existing parameter definitions and parameter references as a guide to add the missing ones.

You are not required to, but you can inspect the **`todo-template-clean.yaml`** file in the same folder. This file contains the application resources exported by the **`oc get`** command, with the runtime attributes already removed. Do not make any changes to this file.

You are not required to, but you can also inspect the **`new-app-db.sh`**, **`new-app-node.sh`**, and **`add-route.sh`** files in the same folder that contain the commands used to generate the resources that were exported to the **`todo-template-clean.yaml`** file. Do not make any changes to these files and do not run any of them.

2. Create a new project and deploy the To Do List application using the template definition file you completed during the previous step.

Review the **`oc-new-app.sh`** starter script file in the **`~/DO288/labs/review-template`** folder. Make a copy of this file in the **`/home/student/`** directory and use this copy to make your changes throughout the lab. Identify and add any missing parameters to the **`oc new-app`** command line. Run the finalized **`oc-new-app.sh`** script to deploy the application.

Do not perform any manual steps to initialize the database. The To Do List application creates the required database tables if you pass the correct set of parameters to the template. The database tables do not need an initial data set.

3. Test the To Do List application using either a web browser or the command line. Do not forget to source the variables from the **`/usr/local/etc/ocp4.config`** before logging in to OpenShift.

4. Grade your work.

Run the following command on the **`workstation`** VM to verify that all tasks were accomplished:

```
[student@workstation ~]$ lab review-template grade
```

5. Clean up. Delete the project from OpenShift.

Open a terminal window on the **`workstation`** VM and run the following command to perform the cleanup tasks:

```
[student@workstation ~]$ lab review-template finish
```

This concludes the lab.

Solution

Creating Applications from OpenShift Templates

Performance Checklist

In this lab, you will deploy the To Do List application from a template that you will complete for reuse.

Outcomes

You should be able to complete a template that deploys a database pod and a web application pod, use the template to deploy the application, and verify that the application works.

Before You Begin

To perform this exercise, ensure you have access to:

- A running OpenShift cluster.
- The sample application (todo-single) in the Git repository.
- The npm dependencies required by the application (**`restify`**, **`sequelize`**, and **`mysql`**).
- The Node.js 10 S2I builder image, and the MySQL 5.7 database image required by the template.

Run the following command on the **`workstation`** VM to validate the prerequisites and to download the starter project files and solution files:

```
[student@workstation ~]$ lab review-template start
```

Requirements

The application is a To Do List application written in JavaScript. It consists of a web front end, based on the AngularJS web framework, and an HTTP API back end, based on Node.js. The back end uses the Restify and Sequelize frameworks.

Both the application front end and back end run on the same container. The application is deployed from source code stored in a Git repository. A MySQL database is used as the data store. The application initializes the database on startup.

The template takes the following parameters:

- **`APP_GIT_URL`**: The Git URL where the source code for the To Do List application is stored.
- **`HOSTNAME`**: The host name used to access the To Do List application.
- **`NPM_PROXY`**: The URL of the NPM repository server.
- **`SECRET`**: The secret for OpenShift webhooks.

- **PASSWORD**: The database connection password. The user name is fixed in the template.
- **CLEAN_DATABASE**: A flag that indicates whether the application initializes the database on startup.

The template parameters have the following restrictions:

- The **APP_GIT_URL**, **HOSTNAME**, **NPM_PROXY**, and **CLEAN_DATABASE** parameters are required.
- The **SECRET** and **PASSWORD** parameters have random default values.
- The **CLEAN_DATABASE** parameter has the default value of **true**.

A starter template file called **todo-template.yaml** is provided in the **~/DO288/labs/review-template** folder. It contains the resources required to deploy the application. The resources are in the correct order, and already cleaned up. You are required to add the missing parameters and add references to the parameters in the resource list.

Deploy the application according to the following requirements:

- The application project is called **youruser-review-template**.
- Use the **~/DO288/labs/review-template/oc-new-app.sh** script to deploy the application. Edit the script as needed to pass any required parameters.
- The application initializes the database on startup.
- The password to access the database is **mypass**.
- Npm modules required to build the application are available from:

  ```
  http://nexus-common.apps.cluster.domain.example.com/repository/nodejs
  ```

- If you prefer to test the application with a web browser, use the web front end to add some entries to the To Do List application interactively. The application user interface is accessible from the following URL:

  ```
  http://youruser-todo.apps.cluster.domain.example.com/todo/index.html
  ```

- If you prefer to test the application using the command line, use the following URL to submit an HTTP GET request to the application back end:

  ```
  http://youruser-todo.apps.cluster.domain.example.com/todo/api/items-
  count
  ```

  The reply includes a **count** attribute, even if there are no entries in the application. An error reply means the database was not initialized correctly.

Steps

1. Review the **todo-template.yaml** starter template file in the **~/DO288/labs/review-template** folder. Make a copy of this file in the **/home/student/** directory, and use this copy to make your changes throughout the lab. Identify and add any missing parameters at the end of the file. Identify any parameter references missing from the template resource list and add them as needed. Use the existing parameter definitions and parameter references as a guide to add the missing ones.

   You are not required to, but you can inspect the **todo-template-clean.yaml** file in the same folder. This file contains the application resources exported by the **oc get** command, with the runtime attributes already removed. Do not make any changes to this file.

You are not required to, but you can also inspect the **`new-app-db.sh`**, **`new-app-node.sh`**, and **`add-route.sh`** files in the same folder that contain the commands used to generate the resources that were exported to the **`todo-template-clean.yaml`** file. Do not make any changes to these files and do not run any of them.

1.1. Create a copy of the starter template to make edits:

```
[student@workstation ~]$ cp ~/DO288/labs/review-template/todo-template.yaml \
> ~/todo-template.yaml
```

1.2. Identify any missing parameters in the template file.

Open the **`~/todo-template.yaml`** file with a text editor.

Scroll down to the end of the file. The file contains four parameters called **`APP_GIT_URL`**, **`HOSTNAME`**, **`NPM_PROXY`**, and **`SECRET`**. These parameter definitions are complete and do not need any changes.

The **`PASSWORD`** and **`CLEAN_DATABASE`** parameters are missing and need to be added.

1.3. Add the missing parameters to the **`~/todo-template.yaml`** template file.

Append the following lines to the file. You can copy these lines from the **`add-parameters.yaml`** file in the **`~/DO288/solutions/review-template`** folder:

```
- name: PASSWORD
  displayName: Database Password
  description: Password to access the database
  generate: expression
  from: '[a-zA-Z0-9]{16}'
- name: CLEAN_DATABASE
  displayName: Initialize the database
  description: If 'true', the database is cleaned when the application starts.
  required: true
  value: "true"
```

1.4. Identify the missing references to parameter values in the template resource list.

Some of the resources already reference the parameters correctly. The following listing highlights the parameter references that are already in the template file. You do not need to change any of them:

```
...output omitted...
  kind: BuildConfig
  ...output omitted...
    name: todoapp
    ...output omitted...
    triggers:
    - github:
        secret: ${SECRET}
      type: GitHub
    - generic:
        secret: ${SECRET}
  ...output omitted...
  kind: DeploymentConfig
    ...output omitted...
    name: todoapp
    ...output omitted...
        - env:
          - name: DATABASE_NAME
            value: tododb
          - name: DATABASE_PASSWORD
            value: ${PASSWORD}
          - name: DATABASE_SVC
            value: tododb
          - name: DATABASE_USER
            value: todoapp
          - name: DATABASE_INIT
            value: ${CLEAN_DATABASE}
...output omitted...
```

A few resources are missing references to parameters. The following listing highlights the attributes that are set to hard-coded values and need to be replaced by parameter references:

```
...output omitted...
  kind: BuildConfig
  ...output omitted...
    name: todoapp
  ...output omitted...
    strategy:
      sourceStrategy:
        env:
        - name: npm_config_registry
          value: http://INVALIDHOST.NODOMAIN.NUL
  ...output omitted...
  kind: DeploymentConfig
  ...output omitted...
    name: tododb
    ...output omitted...
        - env:
          - name: MYSQL_DATABASE
            value: tododb
          - name: MYSQL_PASSWORD
            value: FIXEDPASSWD
...output omitted...
  kind: Route
  ...output omitted...
    name: todoapp
   ...output omitted...
  spec:
    host: http://INVALIDHOST.NODOMAIN.NUL
...output omitted...
```

1.5. Add the missing references to parameters to the **`~/todo-template.yaml`** file. The following listing shows the changes you need to make:

```
...output omitted...
  kind: BuildConfig
  ...output omitted...
    name: todoapp
  ...output omitted...
    strategy:
      sourceStrategy:
        env:
        - name: npm_config_registry
          value: ${NPM_PROXY}
  ...output omitted...
  kind: DeploymentConfig
  ...output omitted...
    name: tododb
    ...output omitted...
        - env:
          - name: MYSQL_DATABASE
            value: tododb
          - name: MYSQL_PASSWORD
            value: ${PASSWORD}
...output omitted...
  kind: Route
  ...output omitted...
    name: todoapp
   ...output omitted...
  spec:
    host: ${HOSTNAME}
...output omitted...
```

1.6. Review your edits and save the changes to the **~/todo-template.yaml** template file. You can compare your edits with the solution file in the **~/DO288/solutions/review-template** folder.

2. Create a new project and deploy the To Do List application using the template definition file you completed during the previous step.

Review the **oc-new-app.sh** starter script file in the **~/DO288/labs/review-template** folder. Make a copy of this file in the **/home/student/** directory and use this copy to make your changes throughout the lab. Identify and add any missing parameters to the **oc new-app** command line. Run the finalized **oc-new-app.sh** script to deploy the application.

Do not perform any manual steps to initialize the database. The To Do List application creates the required database tables if you pass the correct set of parameters to the template. The database tables do not need an initial data set.

2.1. Create a copy of the starter script to make edits:

```
[student@workstation ~]$ cp ~/DO288/labs/review-template/oc-new-app.sh \
> ~/oc-new-app.sh
```

2.2. Identify any missing parameters in the **~/oc-new-app.sh** script file.

Open the **~/oc-new-app.sh** file with a text editor.

The script passes values for the **APP_GIT_URL**, **NPM_PROXY**, **PASSWORD**, and **CLEAN_DATABASE** parameters.

The values passed for **APP_GIT_URL**, **NPM_PROXY**, and **PASSWORD** are correct; you do not need to change them. You need to change the value for the **CLEAN_DATABASE** parameter to **true**.

You also need to add a command-line option to pass a value for the **HOSTNAME** parameter.

2.3. Make changes to the **~/oc-new-app.sh** script.

Use the following listing as a guide to make the changes:

```
oc new-app  --as-deployment-config --name todo --file ~/todo-template.yaml \
> -p APP_GIT_URL=https://github.com/${RHT_OCP4_GITHUB_USER}/DO288-apps \
> -p NPM_PROXY=http://${RHT_OCP4_NEXUS_SERVER}/repository/nodejs \
> -p PASSWORD=mypass \
> -p CLEAN_DATABASE=true \
> -p HOSTNAME=${RHT_OCP4_DEV_USER}-todo.${RHT_OCP4_WILDCARD_DOMAIN}
```

Do not add line breaks to any parameter value. The value for **NPM_PROXY** needs to be on a single line. Recall that you do not need to change it from the starter file.

2.4. Review your edits and save the changes to the **~/oc-new-app.sh** file. You can compare your edits with the solution file in the **~/DO288/solutions/review-template** folder.

3. Test the To Do List application using either a web browser or the command line. Do not forget to source the variables from the **/usr/local/etc/ocp4.config** before logging in to OpenShift.

3.1. Load your classroom environment configuration.

Run the following command to load the environment variables created in the first guided exercise:

```
[student@workstation ~]$ source /usr/local/etc/ocp4.config
```

3.2. Log in to OpenShift using your developer user account:

```
[student@workstation ~]$ oc login -u ${RHT_OCP4_DEV_USER} \
> -p ${RHT_OCP4_DEV_PASSWORD} ${RHT_OCP4_MASTER_API}
Login successful.
...output omitted...
```

3.3. Create the ***youruser*-review-template** project:

```
[student@workstation ~]$ oc new-project ${RHT_OCP4_DEV_USER}-review-template
Now using project "youruser-review-template" on server
"https://api.cluster.domain.example.com:6443"
...output omitted...
```

3.4. Run the **oc-new-app.sh** script to create all of the resources from the **todo-template.yaml** template file:

```
[student@workstation ~]$ ~/oc-new-app.sh
...output omitted...
--> Creating resources...
    imagestream "todoapp" created
    buildconfig "todoapp" created
    deploymentconfig "todoapp" created
    deploymentconfig "tododb" created
    service "todoapp" created
    service "tododb" created
    route "todoapp" created
--> Success
...output omitted...
```

3.5. Wait for the To Do List application build to finish:

```
[student@workstation ~]$ oc logs bc/todoapp -f
...output omitted...
Push successful
```

3.6. Wait for the application and the database pod to be ready and running:

```
[student@workstation ~]$ oc get pods
NAME              READY     STATUS      RESTARTS   AGE
...output omitted...
todoapp-1-nch6c   1/1       Running     0          1m
tododb-1-lpk63    1/1       Running     0          2m
```

3.7. Get the route host name for the application:

```
[student@workstation ~]$ oc get route/todoapp -o jsonpath='{.spec.host}{"\n"}'
youruser-todo.apps.cluster.domain.example.com
```

3.8. Test the application using the route host name obtained from the previous step.

If you prefer to test the To Do List application using a web browser, access the following URL and add some entries to the list:

`http://youruser-todo.apps.cluster.domain.example.com/todo/index.html`

3.9. If you prefer to test the To Do List application using the command line, open a terminal window and run the following command:

```
[student@workstation ~]$ curl -siw "\n" \
> http://${RHT_OCP4_DEV_USER}-todo.${RHT_OCP4_WILDCARD_DOMAIN}\
> /todo/api/items-count
HTTP/1.1 200 OK
...output omitted...
{"count":0}
```

4. Grade your work.

Run the following command on the **`workstation`** VM to verify that all tasks were accomplished:

```
[student@workstation ~]$ lab review-template grade
```

5. Clean up. Delete the project from OpenShift.

Open a terminal window on the **`workstation`** VM and run the following command to perform the cleanup tasks:

```
[student@workstation ~]$ lab review-template finish
```

This concludes the lab.

Summary

In this chapter, you learned:

- A template is a parameterized list of OpenShift resources. A typical use case for templates is deploying applications composed of multiple pods.
- Template parameters can be required or optional, can define default values, and can define randomly generated default values that are useful for secrets and passwords.
- Typical methods of creating a template are concatenating the output of multiple **`oc new-app -o`** commands and exporting existing resources using the **`oc get -o yaml --export`** command.
- The **`oc describe`** and **`oc explain`** commands help developers to find information about all valid attributes for a given kind of resource.